UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Informatica e dell'Automazione

TESI DI LAUREA

Progettazione e implementazione di un sistema di e-learning per il supporto all'insegnamento della programmazione

Design and implementation of an e-learning system to support the teaching of programming

Relatore

Prof. Domenico Ursino

Candidato

Elia Pacioni

Correlatore

Prof. Francisco Fernández de Vega

ANNO ACCADEMICO 2020-2021

Sommario

Negli ultimi anni i sistemi di e-learning hanno assunto un'importanza sempre maggiore perché modificando la fruizione dei contenuti e le modalità di interazione tra studente e professore rendono la didattica flessibile. In questa tesi vengono presentate la progettazione e l'implementazione di un sistema di e-learning per il supporto all'insegnamento della programmazione che va incontro alle esigenze di studenti e professori. In particolare, viene innanzitutto descritta una prima fase di specifica ed analisi dei requisiti, seguita dalla progettazione della base di dati e della componente applicativa. Successivamente, viene implementato il sistema e viene effettuato il deploy per renderlo disponibile agli utenti finali. Infine, viene presentato il confronto con i sistemi correlati e vengono delineati gli obiettivi futuri del progetto.

Keyword: E-learning; Insegnamento della Programmazione; Ingegneria del Software; MariaDB; PHP; Laravel.

Indice

Introduzione **1**

1 Specifica e analisi dei requisiti **3**
1.1 Descrizione del progetto . . . 3
1.2 Glossario . . . 4
1.3 Requisiti funzionali . . . 4
1.4 Requisiti non funzionali . . . 6
1.5 Diagramma dei casi d'uso . . . 7
1.6 Diagramma delle attività . . . 7

2 Progettazione della base di dati sottostante **11**
2.1 Progettazione concettuale . . . 11
2.1.1 Identificazione entità fondamentali . . . 11
2.1.2 Schema scheletro . . . 12
2.1.3 Sviluppo componenti scheletro . . . 13
2.1.4 Schema E-R finale . . . 16
2.1.5 Analisi di qualità dello schema . . . 18
2.2 Progettazione Logica . . . 18
2.2.1 Tavola dei volumi e delle operazioni . . . 18
2.2.2 Ristrutturazione dello schema concettuale . . . 21
2.2.3 Schema E-R ristrutturato . . . 21
2.2.4 Normalizzazione . . . 24
2.2.5 Traduzione verso il modello relazionale . . . 24

3 Progettazione della componente applicativa **26**
3.1 Realizzazione Mockup . . . 26
3.2 Architettura e pattern . . . 29
3.3 Scelta dei linguaggi di programmazione . . . 34
3.4 Tecnologie utilizzate . . . 35

4 Implementazione del sistema **36**
4.1 Struttura dell'applicazione . . . 36
4.2 Creazione del model e uso di Eloquent ORM . . . 38
4.3 Organizzazione dei controller . . . 41
4.4 Gestione delle policy e validazione dei dati . . . 43

4.5 Gestione delle viste . . . 43
4.6 Gestione del multilingue . . . 45
4.7 Esecuzione del codice . . . 45
4.8 Monitoraggio . . . 51

5 Manuale del sistema **53**
5.1 Manuale dello studente . . . 53
5.1.1 Registrazione . . . 53
5.1.2 Area riservata . . . 55
5.1.3 Dashboard . . . 55
5.1.4 Gestione del profilo . . . 55
5.1.5 LiveCoding . . . 55
5.1.6 Gestione dei Progetti . . . 57
5.1.7 Gruppi . . . 58
5.1.8 ToDo List . . . 58
5.1.9 Feedback . . . 60
5.2 Manuale del professore . . . 61
5.2.1 Gestione degli utenti . . . 61
5.2.2 Gestione corsi . . . 62
5.2.3 Gestione gruppi . . . 64
5.2.4 Monitoraggio . . . 65

6 Analisi dei sistemi correlati **66**
6.1 Panoramica sui sistemi correlati . . . 66
6.1.1 Replit . . . 66
6.1.2 Codeanywhere . . . 67
6.2 Confronto con i sistemi correlati . . . 67

7 Conclusioni **69**

Bibliografia **70**

Ringraziamenti **71**

Elenco delle figure

1.1 Diagramma del caso d'uso "LiveCoding" . . . 7
1.2 Diagramma del caso d'uso "Esecuzione codice progetto" . . . 8
1.3 Diagramma del caso d'uso "Supervisione studente" . . . 8
1.4 Diagramma del caso d'uso "Immatricolazione studente" . . . 8
1.5 Diagramma delle attività dalla registrazione all'esecuzione del codice dello studente . . . 10

2.1 Entità fondamentali . . . 12
2.2 Schema scheletro . . . 12
2.3 Relazione user - project . . . 13
2.4 Relazione user - subject . . . 14
2.5 Relazione subject - project . . . 14
2.6 Relazione user - subject - project . . . 15
2.7 Relazione project - script . . . 15
2.8 Relazioni user - all . . . 16
2.9 Relazione historical . . . 17
2.10 Schema E/R . . . 17
2.11 Eliminazione delle gerarchie . . . 22
2.12 Eliminazione degli attributi multivalore . . . 22
2.13 Schema E/R ristrutturato . . . 23

3.1 Mockup delle sezioni . . . 27
3.2 Mockup della dashboard dello studente . . . 28
3.3 Mockup della dashboard dell'amministratore . . . 28
3.4 Mockup della dashboard dell'amministratore con menu ridotto . . . 29
3.5 Mockup della pagina di livecoding . . . 30
3.6 Mockup della pagina di livecoding mobile . . . 30
3.7 Mockup della pagina relativa al profilo utente . . . 31
3.8 Mockup della pagina relativa ai dettagli e alla supervisione del progetto . . . 31
3.9 Mockup della pagina relativa alla sessione di coding . . . 32
3.10 Mockup dell'organizzazione tab nella pagina sessione di coding . . . 32
3.11 Mockup della pagina relativa alla sessione di coding nella versione mobile (prima parte) . . . 33
3.12 Mockup della pagina relativa alla sessione di coding nella versione mobile (seconda parte) . . . 33

3.13 Modello MVC 35

4.1 Flusso di lavoro di una richiesta *HTTP* 37
4.2 Migration della tabella *Company* 39
4.3 Modello *User* 40
4.4 Definizione di un *seeder* 41
4.5 Definizione di una *factory* 42
4.6 Controller per la gestione dei progetti 44
4.7 Registrazione delle Policy 45
4.8 Policy per la gestione di un corso 46
4.9 Struttura delle cartelle contenti le viste 47
4.10 Vista del profilo dell'utente 47
4.11 Middleware per la gestione del multilingue 48
4.12 Controller per la gestione del multilingue 48
4.13 Server WebSocket 49
4.14 Controller WebSocket (prima parte) 50
4.15 Controller WebSocket (seconda parte) 51
4.16 Creazione del canale per la comunicazione WebSocket 52

5.1 Pulsante di registrazione 53
5.2 Form di registrazione 54
5.3 E-mail per la conferma della registrazione 54
5.4 Menù left sidebar 55
5.5 Dashboard dello studente 56
5.6 Pagina del profilo dell'utente 56
5.7 Modifica del profilo dell'utente 56
5.8 Pagina della funzionalità di *LiveCoding* 57
5.9 Form per la creazione di un progetto 57
5.10 Pagina per la visualizzazione dei progetti 58
5.11 Creazione di uno script 59
5.12 Sessione di coding 59
5.13 Lista dei gruppi dello studente 59
5.14 Pagina di dettaglio del gruppo 60
5.15 Pagina *ToDoList* 60
5.16 Invio di un feedback 61
5.17 Voce di menù per la gestione degli utenti 61
5.18 Attivazione dell'account dell'utente 61
5.19 Riepilogo dei corsi seguiti dagli studenti 62
5.20 Scelta dei corsi per lo studente 62
5.21 Creazione di un nuovo corso 63
5.22 Visualizzazione dei corsi di un ente 63
5.23 Visualizzazione della scheda dettagli di un corso 63
5.24 Visualizzazione dei corsi di un professore 64
5.25 Creazione di un gruppo 64
5.26 Visualizzazione dei gruppi 65
5.27 Scheda dei dettagli di un gruppo 65
5.28 Pagina della funzionalità di monitoraggio 65

6.1 Programmazione con *Replit* 67
6.2 Programmazione con *Codeanywhere* 68

Elenco delle tabelle

1.1 Glossario dei termini usati nell'applicazione 5

2.1 Tavola dei volumi 19
2.2 Tavola delle operazioni 20
2.3 Schema relazionale 25

6.1 Confronto con i sistemi correlati 68

Introduzione

L'e-learning è una modalità di formazione a distanza che permette di collegare i contenuti formativi con l'utente finale. Un sistema di e-learning è, generalmente, un'applicazione web, fruibile da qualsiasi dispositivo, che permette agli studenti di seguire le lezioni a distanza, in modalità sincrona o asincrona. Talvolta, questi sistemi vengono usati come supporto alla didattica in presenza, e non in sostituzione, di essa.

I sistemi di e-learning negli ultimi dieci anni sono stati protagonisti di numerosi dibattiti in ambito scolastico e accademico. Dal 2020 il loro utilizzo è aumentato esponenzialmente ed ha indotto a significativi cambiamenti sia nello svolgimento delle lezioni sia nelle relazioni con gli studenti. La crescita nell'utilizzo ha evidenziato, come principali vantaggi, la flessibilità nella fruizione dei contenuti, l'interattività fra professore e studenti nell'utilizzo e la condivisione dei documenti. La possibilità di seguire le lezioni in differenti modalità consente agli studenti una fruizione dei contenuti flessibile, potenzialmente più favorevole all'apprendimento, perché tiene conto di esigenze personali. Il professore può proporre esercizi mirati da far svolgere a singoli studenti ed, avendo la possibilità di monitorarne i risultati, può guidare questi ultimi in percorsi personalizzati in base al livello. La possibilità di condividere documenti in tempo reale permette di tenere traccia delle modifiche e di collaborare simultaneamente alla scrittura.

Accanto agli aspetti positivi, sono emersi, chiaramente, anche alcuni svantaggi considerevoli. Nelle lezioni e-learning è molto più difficile coinvolgere gli studenti; il rischio è trovarsi di fronte ad un "muro", con conseguente diminuzione della concentrazione. Ancora, a causa dell'utilizzo delle webcam, alcuni studenti potrebbero sentirsi sovraesposti, e questo potrebbe portare all'aumento del fenomeno del cyberbullismo. Infine, va considerato il *digital divide*, ovvero il divario esistente tra chi ha accesso effettivo alle tecnologie e chi ne è escluso, in modo parziale o totale; pur avendo accesso alle tecnologie possono presentarsi problemi tecnici dovuti alle connessioni o alle prestazioni dei dispositivi utilizzati.

Sicuramente i sistemi di e-learning avranno sempre più importanza in futuro. L'applicazione Educode, presentata nella tesi, è stata costruita per rispondere alle esigenze degli studenti e dei professori perché implementa la suddivisione in gruppi e l'interattività nell'utilizzo. Il sistema presentato può essere usato come supporto alla didattica in presenza o come software per tenere lezioni a distanza; la scelta è a discrezione dell'insegnante.

La presente tesi è composta da sette capitoli strutturati come di seguito specificato:

- Nel Capitolo 1 vengono definiti i requisiti, ponendo le basi per la progettazione dell'applicazione.

- Nel Capitolo 2 viene progettata la base di dati sottostante all'applicazione; la progettazione è articolata in due fasi ovvero progettazione concettuale e progettazione logica.
- Nel Capitolo 3 viene progettata la componente applicativa, definendo le tecnologie da utilizzare, l'architettura dell'applicazione e la grafica.
- Nel Capitolo 4 viene implementato il sistema; in particolare, viene descritta l'implementazione delle componenti principali del sistema e delle funzionalità specifiche.
- Nel Capitolo 5 vengono presentati i manuali del sistema; il primo manuale è per lo studente mentre il secondo per il professore.
- Nel Capitolo 6 vengono analizzati i sistemi correlati e vengono confrontati con EduCode per avere una panoramica del mercato e trarre spunti per gli sviluppi futuri.
- Nel Capitolo 7 vengono tratte le conclusioni in merito al lavoro svolto e vengono fissati gli obiettivi futuri del progetto.

CAPITOLO 1

Specifica e analisi dei requisiti

In questo capitolo viene introdotta l'idea progettuale e si definiscono i requisiti funzionali e non funzionali. L'analisi dei requisiti è una delle fasi fondamentali della progettazione poiché su di essa si costruisce l'intero progetto.

1.1 Descrizione del progetto

L'applicazione nasce in ambito universitario, come supporto ai corsi, per facilitare agli studenti l'apprendimento della programmazione e ai docenti la supervisione e l'assistenza. Ogni docente gestisce i propri corsi, organizzando gli studenti in gruppi ed assegnando ad ogni gruppo uno o più progetti. Il modo di apprendere e lavorare è di tipo collaborativo: sul codice da sviluppare possono intervenire nello stesso tempo più soggetti, funzionalità realizzabile grazie al salvataggio totale del codice in cloud. Esistono, ovviamente, altre piattaforme che permettono di programmare online; la particolarità di EduCode risiede nella presenza della funzione di monitoraggio che il professore può esercitare sui progetti dei propri studenti. EduCode permette di configurare in modo agevole l'ambiente di sviluppo idoneo al linguaggio di programmazione scelto. Vista la varietà di dispositivi ad oggi utilizzabili, questo aspetto è importante per contrastare il digital divide: grazie all'utilizzo di EduCode ed all'esecuzione del codice in cloud, anche i dispositivi meno recenti riescono a supportare lo sviluppo degli algoritmi più "pesanti" con le stesse prestazioni dei dispositivi attuali.

EduCode segue l'approccio allo sviluppo cloud nativo, ovvero utilizza il cloud computing per rendere l'applicazione scalabile, dinamica in cloud pubblici, privati e ibridi. Le tecnologie usate sono websocket, container, microservizi e infrastrutture distribuite. Le modalità di erogazione del software ne consentono la definizione di SaaS[1].

In base al tipo di account, l'utente può avere accesso a funzionalità aggiuntive per migliorare la propria esperienza nel sito. La possibilità di eseguire codice e programmare direttamente online è garantita dall'account base. Inoltre, sono disponibili convenzioni per enti ed aziende che permettono agli utenti iscritti di effettuare un upgrade dell'account.

[1]Software as a Service: modello di cloud computing; l'accesso al software avviene tramite Internet ed il modello di business è "pay per use".

1.2 Glossario

È opportuno definire in via preliminare un glossario che associ ad ogni termine significativo la definizione ed i possibili sinonimi. (Tabella 1.1).

1.3 Requisiti funzionali

I requisiti funzionali fissano il comportamento del sistema nelle specifiche condizioni e/o in corrispondenza di input particolari; in altre parole descrivono le caratteristiche e le funzioni che il sistema deve fornire. Nel caso di EduCode corrispondono a stabilire le modalità di accesso, le tipologie di utenti, l'organizzazione e le modalità di esecuzione del codice sorgente, le relazioni tra professori e corsi.

L'accesso a EduCode avviene attraverso il sito web che contiene la presentazione dell'applicazione e permette il login e/o la registrazione degli utenti. I requisiti funzionali per la gestione dell'account utente sono:

- effettuare la registrazione attraverso la compilazione del form con i dati richiesti: nome, e-mail, università o ente di appartenenza, password;
- confermare la registrazione dell'account tramite il link ricevuto per e-mail;
- effettuare il login e il logout;
- visualizzare e/o modificare le informazioni del profilo;
- eliminare l'account, nel rispetto dei requisiti del GDPR[2].

Gli utenti vengono gestiti con gerarchia piramidale che ne fissa le funzionalità in base al ruolo. I ruoli previsti vanno dal guest, che può visitare solo la parte pubblica del sito, all'admin, che controlla il sistema nella sua totalità. I ruoli possibili sono: admin, professore coordinatore, professore, studente, basic e guest. Funzioni esclusive dell'admin sono l'eliminazione e/o la sospensione di un account utente nonché l'inserimento di nuove università ed enti. Il ruolo predefinito al momento della registrazione è basic; solo l'amministratore può effettuare la modifica del ruolo, tranne nel caso del passaggio da basic a student, che può essere effettuato anche da un professore coordinatore.

I requisiti funzionali per la programmazione fissano due differenti modalità di esecuzione e organizzazione del codice sorgente: live coding e progetto. La funzionalità live coding permette di scrivere ed eseguire sul server di EduCode un codice, nel linguaggio di programmazione scelto senza tenerne traccia nel cloud. Viene utilizzata per test rapidi e prototipazione. L'altra funzionalità di coding è la gestione del progetto, ovvero un insieme di script che utilizzano lo stesso linguaggio. Al momento della creazione del progetto occorre scegliere il linguaggio di programmazione, il nome del progetto e la sua tipologia (personale o di gruppo). Il progetto viene memorizzato nel cloud e può essere eseguito e consultato da qualsiasi dispositivo. La relazione tra progetti e corsi non è obbligatoria perché un progetto può far capo ad un corso specifico o non essere legato ad alcun corso.

Di seguito vengono definiti i requisiti funzionali relativi al rapporto tra professori, corsi e gruppi. La creazione del corso spetta ad una figura specifica; ovvero il professore coordinatore che, al momento della creazione, ne stabilisce il nome, la finalità e indica un professore quale responsabile del corso. Il professore coordinatore ha i compiti di attivare gli account degli studenti e di immatricolarli. L'attivazione è preliminare all'immatricolazione ai corsi. Spetta

[2]General Data Protection Regulation: regolamento (UE) n. 2016/679; è un regolamento dell'Unione Europea in materia di trattamento dei dati personali e di e privacy protezione dei dati.

Termine	Descrizione	Sinonimi
User	Attore del sistema, indicato in genere con il suo ruolo	Utilizzatore, utente, persona
Account	Profilo dell'utente iscritto	Profilo, Conto
Role	Ruolo dell'utente nell'applicazione	Ruolo, carica, incarico
Live Coding	Pagina di esecuzione del codice senza creazione di un progetto	Codifica in tempo reale, esecuzione volatile
Project	Insieme di script riguardanti un singolo argomento o compito	Progetto
Script	File contenente codice sorgente	Programma, file eseguibile
Language	Linguaggio di programmazione afferente ad un progetto o utilizzato nel live coding	Linguaggio
Editor	Sezione dedicata alla scrittura del codice sorgente	
Companies	Ente o società accreditati all'utilizzo della piattaforma	Azienda, università, scuola, ente
Subject	Corso universitario, scolastico, di formazione	Corso, materia
Group	Entità che gestisce un progetto	Gruppo, team
User Activation	Attivazione account utente	Attivazione, abilitazione
Matriculation	Immatricolazione di uno studente ad un corso	Immatricolazione, iscrizione
Suspend User	Sospensione account utente	Sospensione, blocco, ban
Personal Project	Progetto utente personale, non collegato ad un corso e non condiviso con altri utenti	Progetto personale, progetto individuale
Group Project	Progetto utente di una materia o in team	Progetto di gruppo, progetto della materia
Terminal	Terminale per visualizzare i risultati dell'esecuzione	Terminale, linea di comando, finestra d'esecuzione
Monitoring	Sistema che permette al professore di monitorare in tempo reale il progetto di un utente	Monitoraggio
Reverse Monitoring	Sistema che permette agli utenti di vedere in tempo reale il progetto di un professore	Monitoraggio inverso, lezione in tempo reale

Tabella 1.1: Glossario dei termini usati nell'applicazione

al professore incaricato del corso creare i gruppi a cui assegnare i progetti; un gruppo può essere costituito anche da un singolo studente. Il professore svolge la funzione di supervisione ed assistenza agli studenti tramite il sistema di monitoraggio che permette la visione e la modifica in tempo reale dei progetti degli studenti del suo corso. È, anche, definita la funzione di monitoraggio inverso che consente al professore di condividere alcuni suoi progetti con gli studenti.

Inoltre, sono previsti requisiti funzionali traversali; questi sono:

- To Do List, per avere a disposizione note rapide per lo sviluppo;
- F.A.Q. sulla piattaforma, per la risoluzione dei problemi;
- Feedback, per segnalare bug o proporre suggerimenti.

Le funzionalità sopra definite vengono assegnate ai ruoli della gerarchia piramidale come di seguito specificato:

- *Guest*: visualizzazione del sito web pubblico.
- *Basic*: gestione dell'account, del live coding, della to do list, delle F.A.Q. e dell'invio del feedback.
- *Studente*: gestione dei progetti, gestione dei gruppi e del monitoraggio inverso.
- *Professore*: visualizzazione delle materie assegnate dall'istituto, monitoraggio dei progetti degli studenti, creazione e partecipazione delle videochiamate.
- *Professore coordinatore*: gestione dei corsi, gestione degli studenti (attivazione e immatricolazione).
- *Admin*: gestione degli utenti, gestione dell'università e degli enti, visualizzazione dei feedback.

1.4 Requisiti non funzionali

I requisiti non funzionali descrivono il sistema dal punto di vista qualitativo, focalizzando l'attenzione sull'esperienza utente, sull'usabilità del sistema e suelle sue prestazioni. Una corretta definizione dei requisiti non funzionali rende l'applicazione facile da utilizzare e studiata per gli utenti finali. Occorre, naturalmente, garantire il rispetto della privacy in conformità al GDPR.

L'ISO 9241-210 definisce l'esperienza d'uso come "le percezioni e le reazioni di un utente che derivano dall'uso o dall'aspettativa d'uso di un prodotto, sistema o servizio". La norma ISO 9241-11:1998, poi aggiornata dalla ISO 9241-210:2010, definisce l'usabilità come: "Il grado in cui un prodotto può essere usato da particolari utenti per raggiungere certi obiettivi con efficacia, efficienza, soddisfazione in uno specifico contesto d'uso".

Nel caso di EduCode l'esperienza utente e l'usabilità del sistema si traducono in:

- realizzazione di un sito web responsive, per garantire la fruibilità dei contenuti qualunque sia il dispositivo utilizzato per l'accesso;
- possibilità di personalizzare le dimensioni del terminale e dell'editor per facilitare la lettura e la scrittura;
- adattabilità della sintassi dell'editor al linguaggio di programmazione scelto;

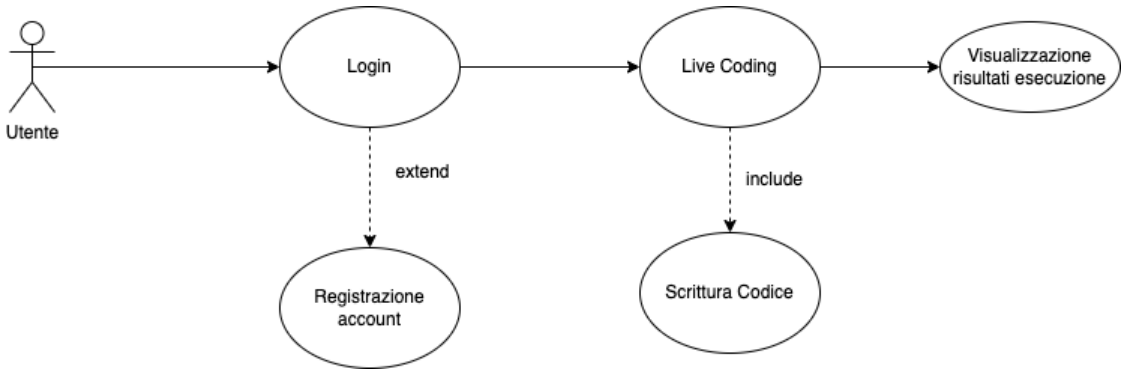


Figura 1.1: Diagramma del caso d'uso "LiveCoding"

- presenza di una dashboard utente riassuntiva delle funzioni fondamentali;
- supporto multilingue per l'internazionalizzazione dell'applicazione.

Le prestazioni che la piattaforma deve garantire comportano:

- evitare, quando possibile, di ricaricare la pagina;
- caricamento della pagina web in meno di tre secondi;
- supporto di almeno 50 utenti simultanei; tale valore è stato definito per la fase di testing, in base al numero di utenti che hanno accesso alla versione beta.

1.5 Diagramma dei casi d'uso

Un caso d'uso è la specifica di una sequenza di azioni che il sistema può eseguire interagendo con attori esterni, incluse eventuali sequenze alternative e sequenze d'errore.
I casi d'uso proposti illustrano le operazioni più rappresentative del sistema; ovvero:

- Avvio di una sessione di live coding, per tutte le azioni compiute dall'utente dal login alla visualizzazione del risultato (Figura 1.1).
- Sessione di lavoro con progetto; illustra il flusso delle operazioni dal login all'avvio di una sessione di lavoro, può comprendere la creazione del progetto e la scrittura del codice da eseguire (Figura 1.2).
- Sistema di monitoraggio per la supervisione dello studente da parte del professore. La funzione di monitoraggio è tra le attività centrali di EduCode. (Figura 1.3).
- Processo di immatricolazione dello studente; è un caso d'uso esclusivo del professore coordinatore (Figura 1.4).

1.6 Diagramma delle attività

Il diagramma della attività serve a descrivere la logica dell'applicazione. È simile ad un diagramma di flusso, ma supporta il parallelismo delle operazioni.

Poiché la logica delle operazioni all'interno dell'applicazione è comune ai diversi casi, di seguito viene presentato, a titolo esemplificativo, solo il diagramma delle attività per la

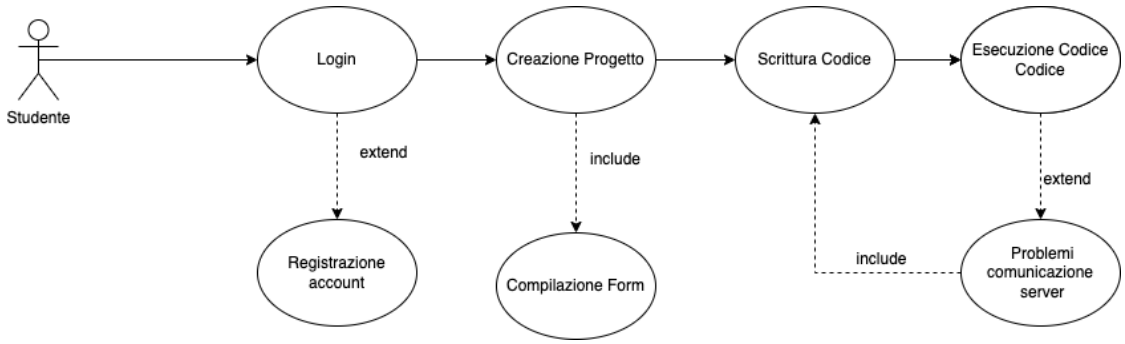


Figura 1.2: Diagramma del caso d'uso "Esecuzione codice progetto"

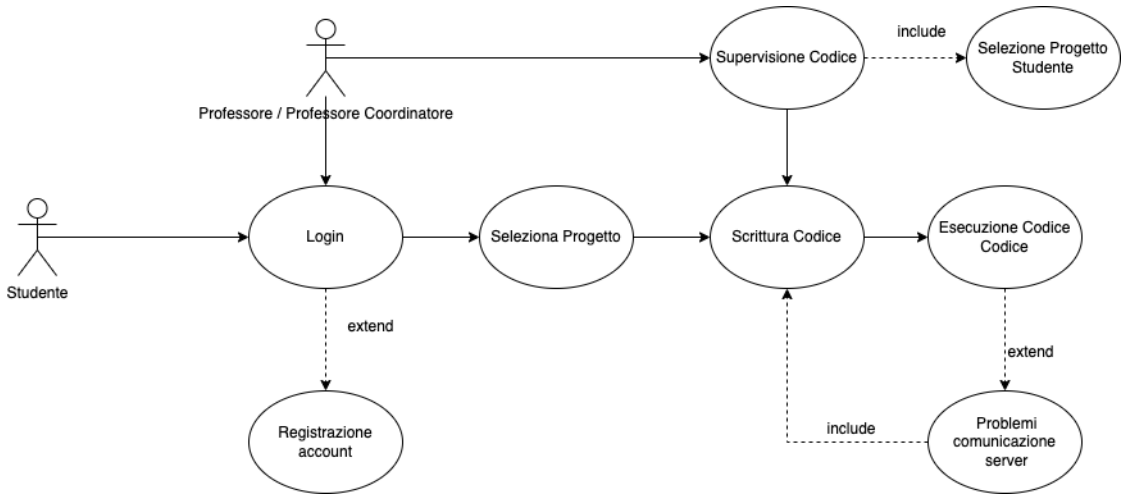


Figura 1.3: Diagramma del caso d'uso "Supervisione studente"

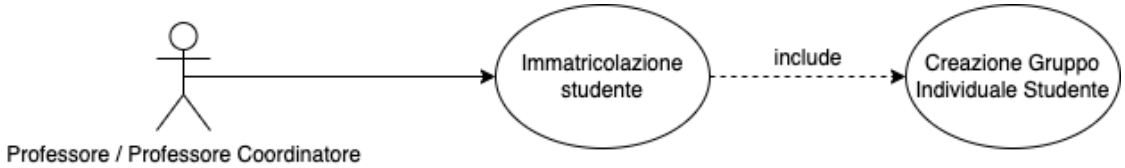


Figura 1.4: Diagramma del caso d'uso "Immatricolazione studente"

sessione di lavoro di un progetto, essa illustra le operazioni dalla registrazione dell'utente all'esecuzione del codice del progetto. Alla sessione di lavoro partecipano tre attori: studente, professore coordinatore e professore. Nel diagramma delle attività ogni attore è contenuto nel proprio blocco. Poiché le azioni dello studente costituiscono il cuore dell'attività, vengono messe nel blocco centrale; in parallelo, nei blocchi laterali, si svolgono le attività del professore coordinatore, nella fase iniziale, e del professore, nella fase finale.

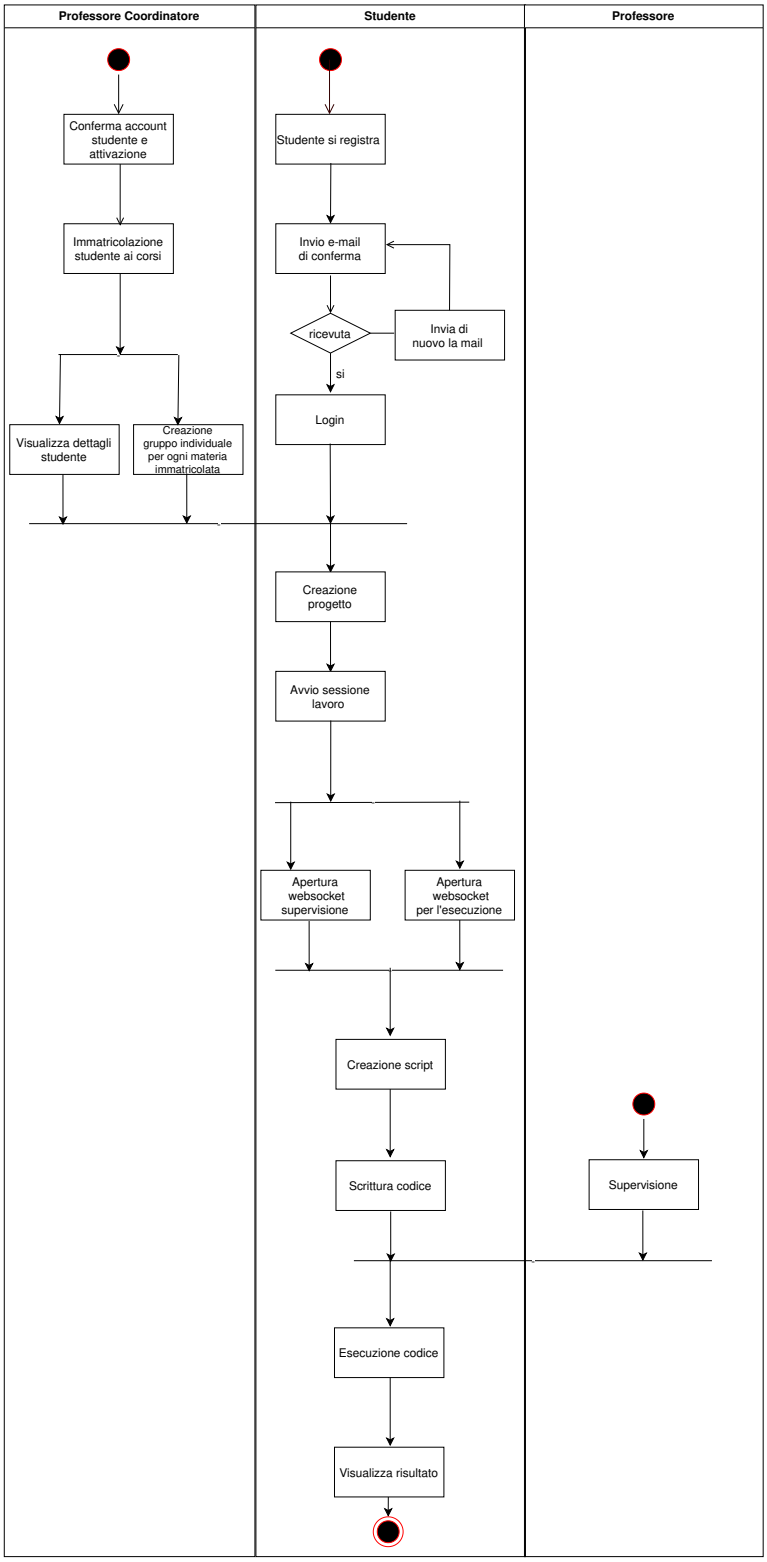


Figura 1.5: Diagramma delle attività dalla registrazione all'esecuzione del codice dello studente

CAPITOLO 2

Progettazione della base di dati sottostante

In questo capitolo viene illustrata la progettazione della base di dati. Esso rappresenta le fondamenta dell'applicazione da realizzare perché la sua corretta progettazione consente all'applicazione di essere scalabile nel tempo e ne supporta le successive modifiche.

2.1 Progettazione concettuale

L'analisi dei requisiti e delle specifiche permette di avere un quadro più ampio del progetto, dei problemi principali da affrontare e delle principali caratteristiche da schematizzare. Nella fase di progettazione concettuale non esiste una rappresentazione univoca delle specifiche, ma ogni informazione può essere rappresentata in modi differenti. Nel caso trattato sono state seguite le "regole concettuali" del modello E-R:

- Se un concetto ha proprietà rilevanti e/o descrive classi di oggetti autonome, è necessario rappresentarlo con un'entità.
- Se un concetto ha una struttura semplice e non possiede proprietà rilevanti associate, è opportuno rappresentarlo con un attributo di un altro concetto a cui si riferisce.
- Se vengono individuate due (o più) entità e nei requisiti compare un concetto che le associa, questo concetto può essere rappresentato da una relazione.
- Se uno o più concetti risultano essere casi particolari di un altro, è opportuno rappresentarli facendo uso di una generalizzazione.

La progettazione segue un approccio misto, ibrido tra le strategie top-down e bottom-up, che, sfruttando i benefici di entrambe le metodologie, garantisce una maggiore flessibilità e si adatta a progetti in continua evoluzione come EduCode. La strategia mista consiste nel dividere i requisiti in componenti separate e, allo stesso tempo, definire uno schema scheletro contenente i temi principali. Lo schema serve per avere una visione complessiva del progetto e aiuta nell'integrazione delle componenti sviluppate singolarmente.

2.1.1 Identificazione entità fondamentali

L'individuazione delle entità fondamentali fa parte del metodo "bottom-up"; consiste nell'individuare i concetti principali senza i quali lo schema non potrebbe essere rappresentato.

Nel caso di EduCode le entità fondamentali sono tre (Figura 2.1):

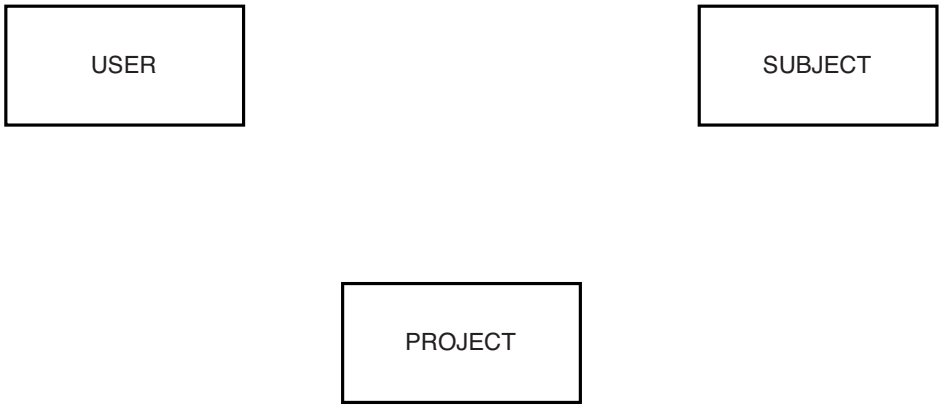


Figura 2.1: Entità fondamentali

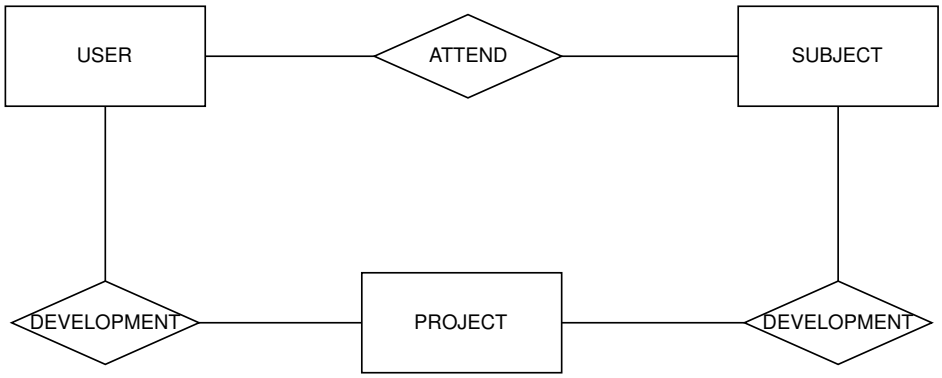


Figura 2.2: Schema scheletro

- *User*: indica l'entità che contiene tutte le tipologie di utente che possono accedere all'applicazione e fruire delle sue funzionalità.
- *Subject*: indica un corso universitario disponibile nella piattaforma.
- *Project*: indica l'identificatore univoco per gli script.

2.1.2 Schema scheletro

Lo schema scheletro riassume le entità fondamentali e le relazioni che le collegano. Questo concetto fa parte della strategia "top-down" e dà un'idea astratta di quello che sarà lo schema finale della base di dati. A partire da questo schema vengono eseguiti dei raffinamenti che porteranno via via ad una definizione concreta della base di dati; i raffinamenti vengono sviluppati tramite la decomposizione tipica del metodo "bottom-up" ed integrati successivamente nello schema scheletro.

La relazione tra le entità *user* e *project* descrive la partecipazione di uno o più utenti ad uno o più progetti; la relazione tra *user* e *subject* esprime la possibilità degli studenti di seguire i corsi proposti. Infine, la relazione tra *subject* e *project* illustra l'esistenza di progetti legati ad una materia. Lo schema scheletro mostrato nella Figura 2.2, potrebbe, a prima vista, presentare una circolarità; un esame più attento rivela che i percorsi *user - subject - project* e *user - project* sono alternativi.

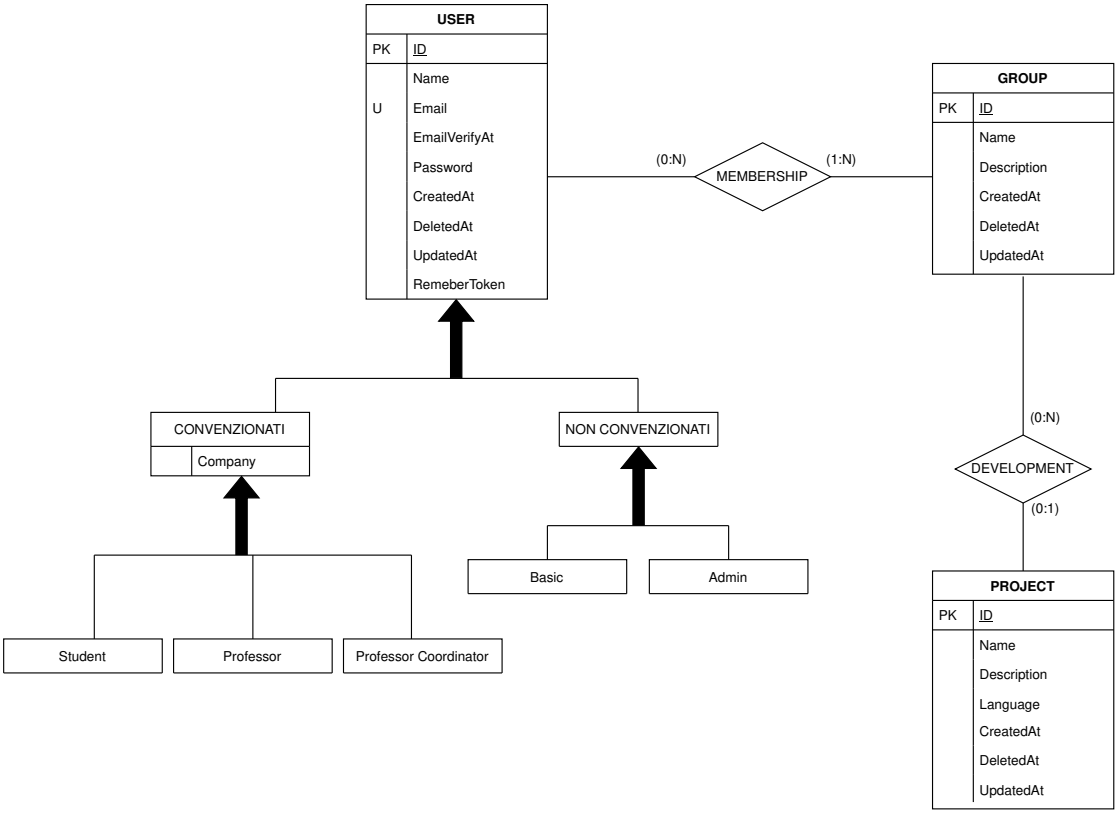


Figura 2.3: Relazione user - project

2.1.3 Sviluppo componenti scheletro

Per sviluppare le componenti dello schema scheletro inizialmente viene applicato il metodo "top-down" che consente di scindere le macro-relazioni e macro entità in concetti specifici. Per fondere e collegare opportunamente relazioni ed entità ottenute, si segue successivamente la strategia "bottom-up" .

La prima relazione presa in esame è: *user - project*. EduCode richiede che un utente possa gestire i propri progetti personali anche in gruppo; è necessario, quindi, rettificare la relazione tra *user* e *project* inserendo l'entità group. Un gruppo è composto da uno o più utenti e l'utente può far parte di più gruppi. Un gruppo può sviluppare più progetti ed un progetto sarà necessariamente di un gruppo (Figura 2.3).

Gli utenti in EduCode possono essere di diverso tipo: guest, basic, student, professor, professor coordinator, admin. Per rappresentare le diverse tipologie di utenti è utile una generalizzazione.

La relazione *user - subject* (Figura 2.4) descrive la partecipazione degli utenti ai corsi e la loro gestione da parte del professore incaricato. La partecipazione dello studente ai corsi è facoltativa; uno studente può partecipare a più corsi ed ogni corso può essere seguito da più studenti. Durante il corso gli studenti possono sviluppare dei progetti richiesti dal professore. Per integrare questa funzionalità è necessario rettificare la relazione tra *subject e project* inserendo l'entità *group* (Figura 2.5). Quindi, l'entità group è comune alla relazione *user - project* e *subject - project* (Figura 2.6).

Un progetto è un'identificatore univoco per raggruppare gli script; è necessario quindi introdurre la relazione *project - script*. Un progetto è composto da più script, tutti gli script di un progetto utilizzano lo stesso linguaggio di programmazione. Per ogni script verrà mantenuto lo storico delle modifiche, per questo viene introdotta l'entità *version*.

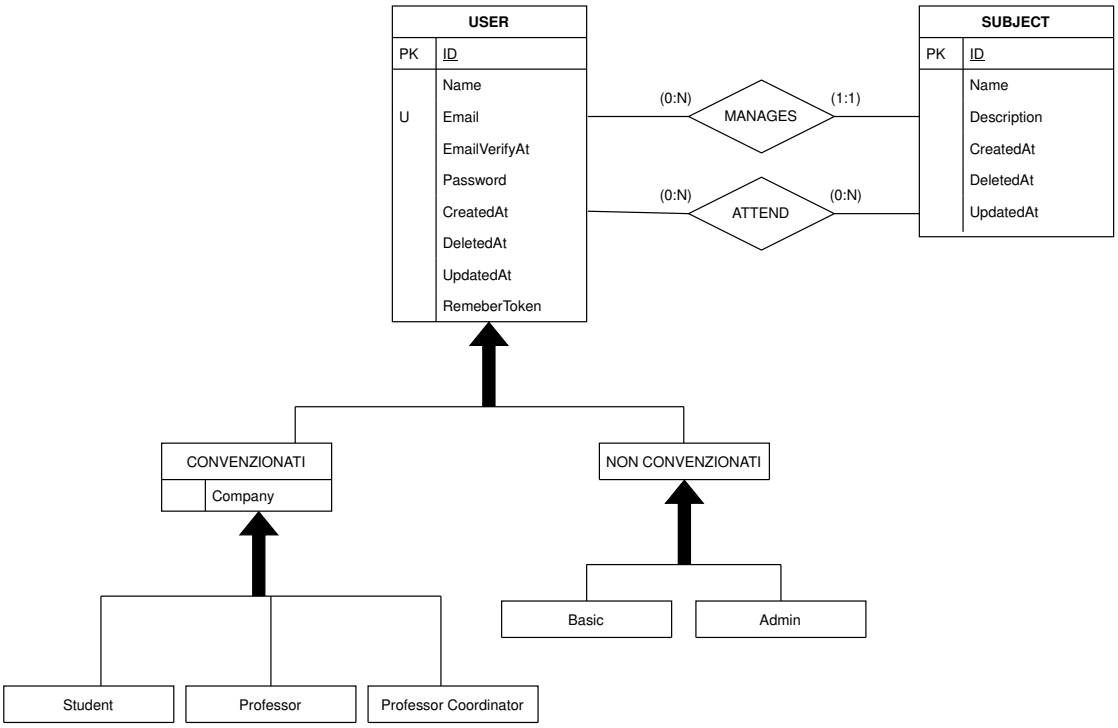


Figura 2.4: Relazione user - subject

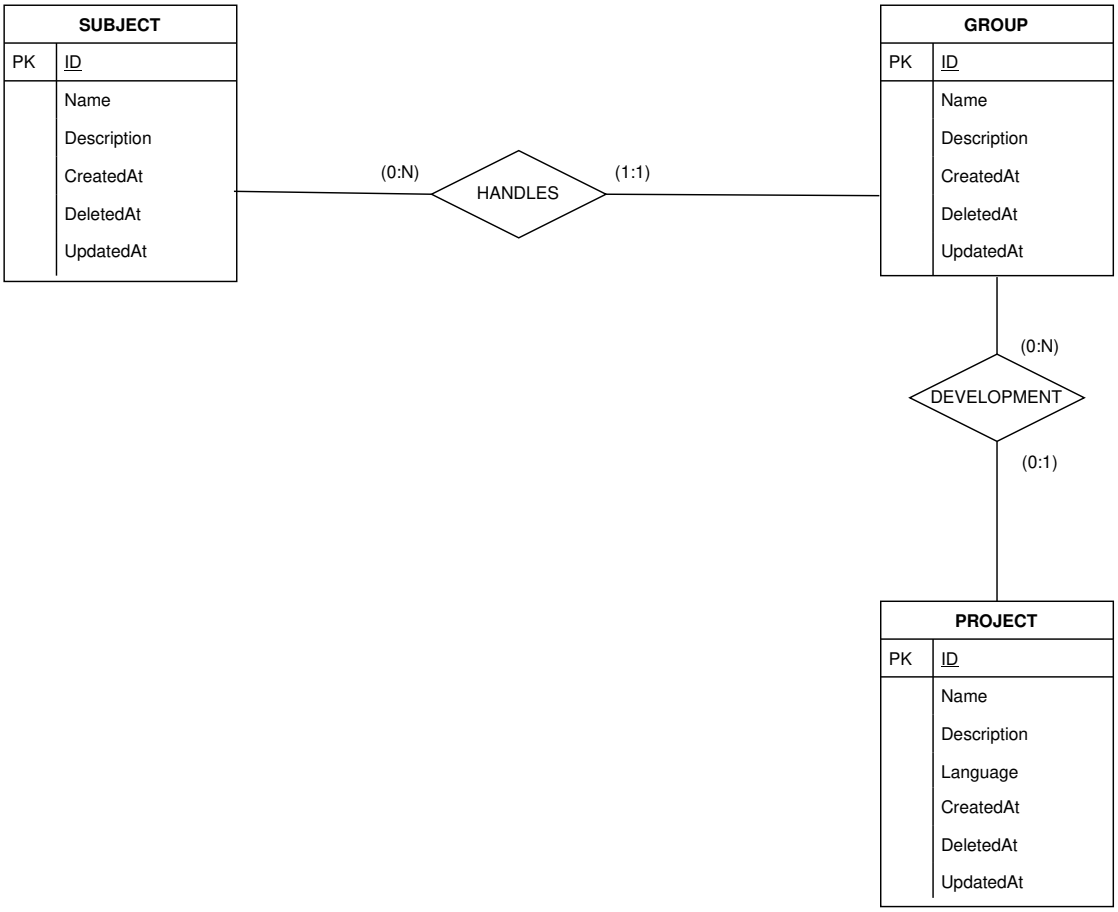


Figura 2.5: Relazione subject - project

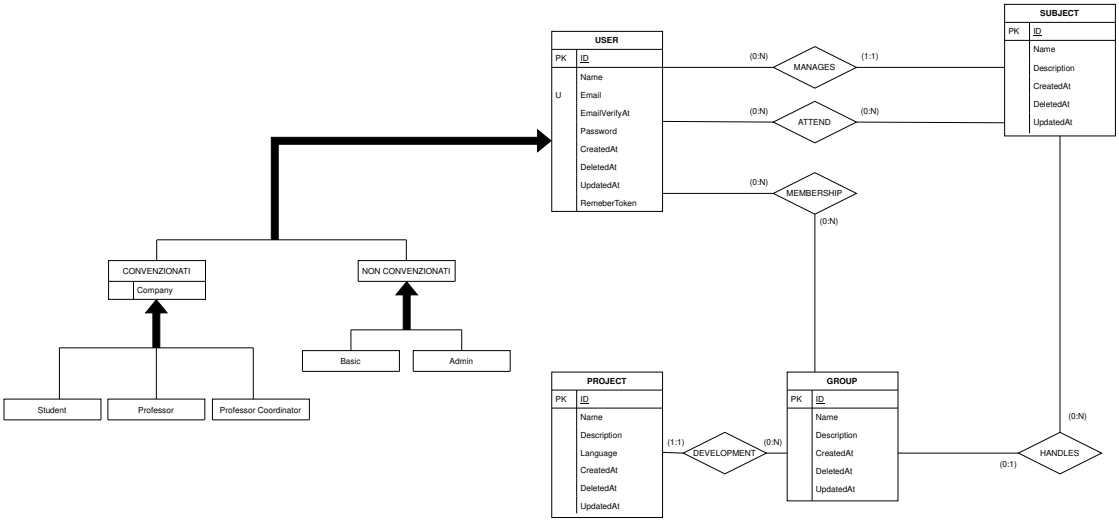


Figura 2.6: Relazione user - subject - project

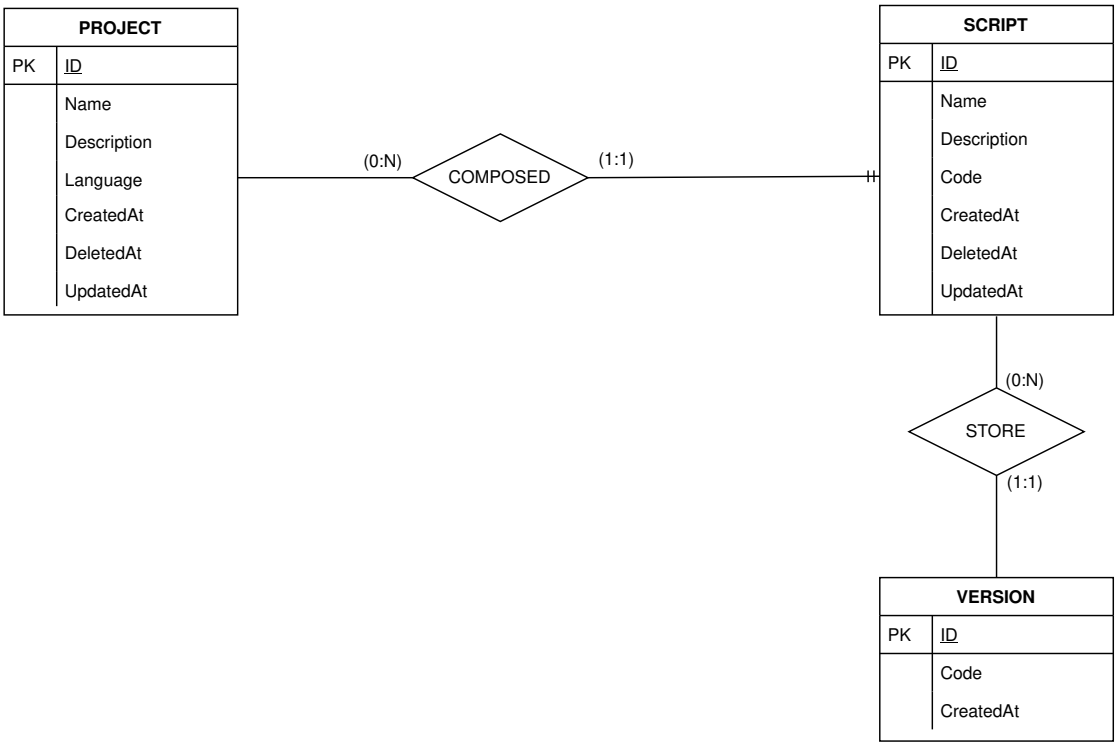


Figura 2.7: Relazione project - script

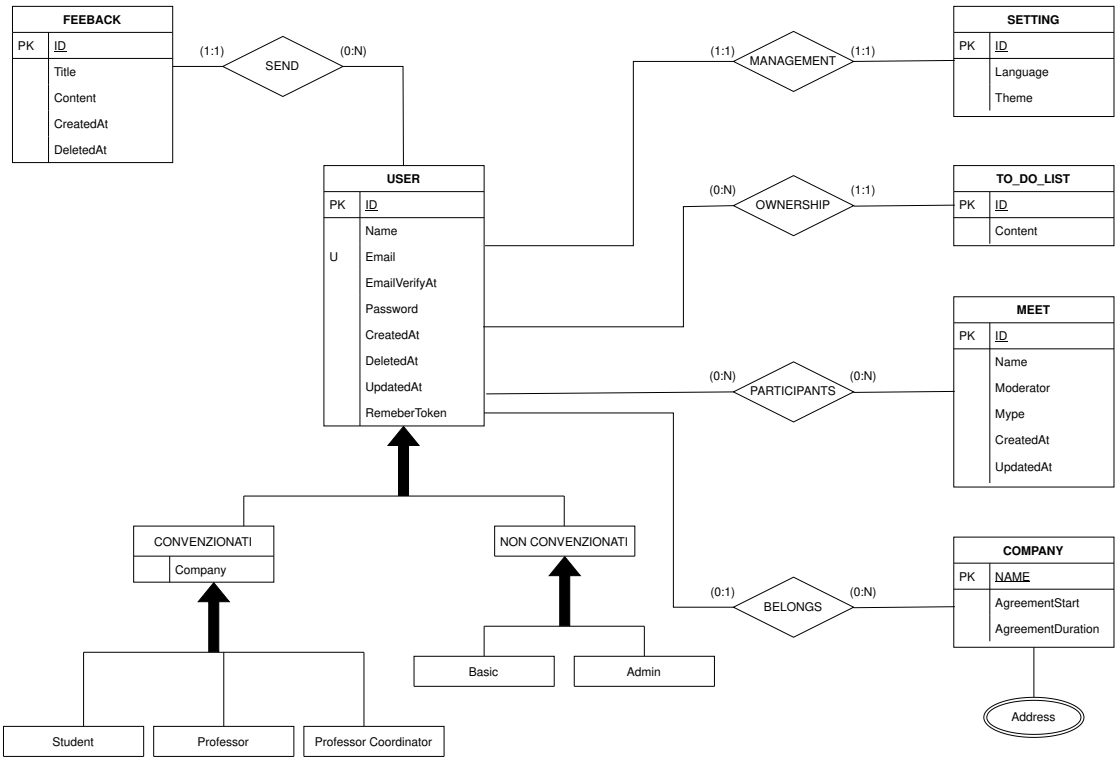


Figura 2.8: Relazioni user - all

Le relazioni sopra descritte servono a creare il cuore del progetto e rendere disponibili le funzionalità principali. Tuttavia, per rendere l'applicazione completa sono utili altre entità che ruotano tutte attorno alla figura dell'utente. Prima di introdurle è necessario descrivere meglio l'entità *user*.

L'entità *user* ha anche le seguenti relazioni (Figura 2.8):

- *User - company*: descrive l'appartenenza di un utente ad un ente.
- *User - setting*: raccoglie tutte le impostazioni per la personalizzazione dell'applicazione web.
- *User - todo*: descrive la gestione delle note da parte dell'utente.
- *User - feedback*: descrive la gestione dei feedback creati dall'utente.
- *User - meet*: descrive la creazione delle videochiamate da parte di un professore e la possibilità di partecipazione di un utente alle videochiamate.

Infine, ci sono le entità per la memorizzazione, con anonimato, di uno storico dei progetti e degli script degli utenti (Figura 2.9). Ogni utente che si elimina dalla piattaforma può scegliere di donare i propri progetti e codici a EduCode, consentendo, così, di analizzarli e di effettuare su di essi attività di machine learning.

2.1.4 Schema E-R finale

Lo schema E-R finale è composto dalle integrazioni dei sotto-schemi in uno schema generale, facendo riferimento allo schema scheletro (Figura 2.10).

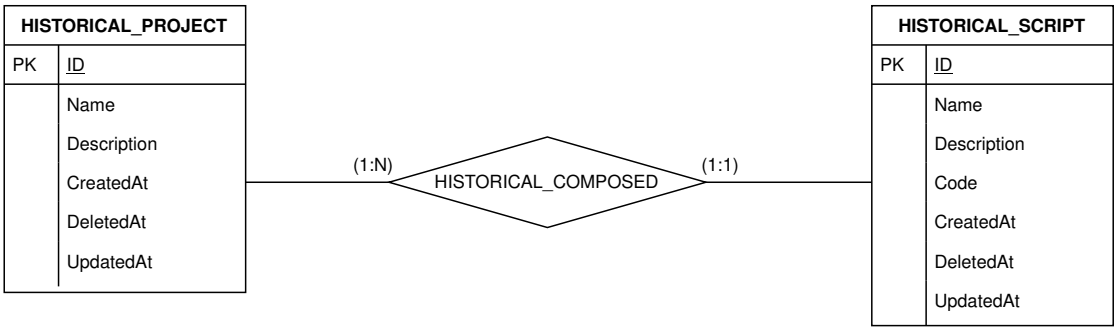


Figura 2.9: Relazione historical

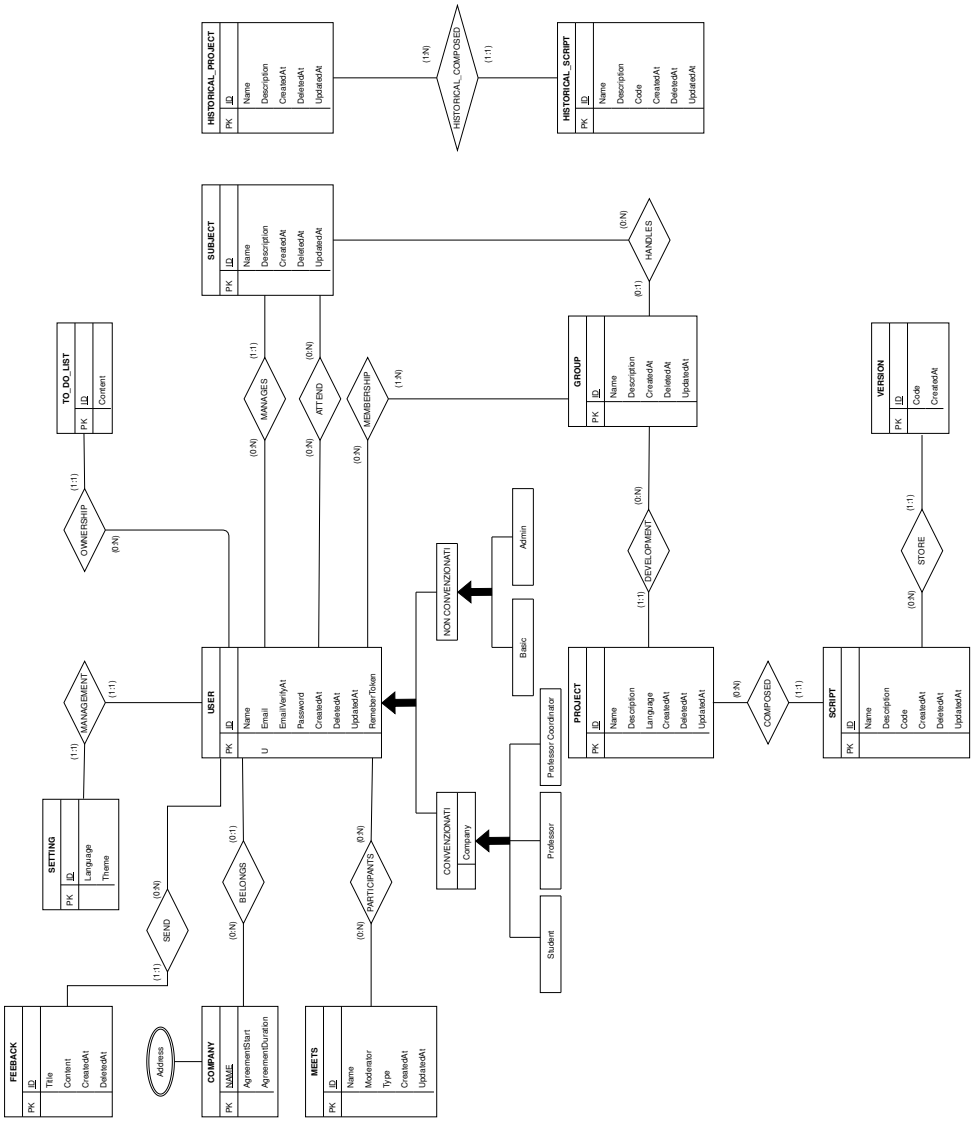


Figura 2.10: Schema E/R

2.1.5 Analisi di qualità dello schema

Uno schema concettuale, per essere di buona qualità, deve garantire alcune proprietà generali:

- *Correttezza*: uno schema concettuale è corretto quando utilizza in modo appropriato i costrutti del modello concettuale di riferimento.
- *Completezza*: uno schema concettuale è completo quando rappresenta in modo completo i dati trattati e quando è possibile eseguire tutte le operazioni previste. Per la verifica della completezza si può effettuare un test di navigazione dello schema durante le operazioni da svolgere.
- *Leggibilità*: uno schema concettuale è leggibile quando rappresenta le specifiche in modo naturale e comprensibile; lo schema deve essere autoesplicativo. Nella leggibilità rientrano anche i parametri estetici dello schema.
- *Minimalità*: uno schema concettuale è minimale quando non presenta ridondanze e cicli; tutte le specifiche sui dati vengono rappresentate una sola volta nello schema. Non sempre le ridondanze o i cicli rappresentano scelte errate, purché siano documentati e dettati da precise scelte progettuali. Il test per la minimalità viene effettuato per ispezione, verificando se ci sono ridondanze o cicli che possono essere eliminati senza alterare la completezza dello schema.

Lo schema di EduCode è corretto perché rappresenta in modo logico la realtà d'interesse; completo perché analizzando i requisiti iniziali copre tutti gli aspetti trattati; leggibile perché nonostante la quantità di dati da rappresentare, sono state evitate intersezioni e organizzate le tabelle in modo ordinato e chiaro. Lo schema però non può essere definito minimale per la presenza del ciclo; la scelta consapevole della circolarità è stata effettuata per garantire rapidità nella consultazione di dati utilizzati costantemente in molte operazioni. Il ciclo non limita in alcun modo la scalabilità dello schema e non ne compromette l'espansione futura.

2.2 Progettazione Logica

La progettazione logica consiste nel riorganizzare lo schema concettuale e nell'implementare la traduzione nel modello logico. La fase di ristrutturazione dello schema E/R è indipendente dal modello logico scelto, mentre la traduzione è strettamente collegata al modello scelto e include ottimizzazioni basate sulle caratteristiche del modello logico. Per EduCode è stato scelto il modello relazionale.

2.2.1 Tavola dei volumi e delle operazioni

Per ottimizzare alcuni indici di prestazione del database è necessario studiare, oltre allo schema, il volume dei dati e le caratteristiche delle operazioni. Queste informazioni consentono di effettuare uno studio di massima sul costo di un'operazione e sull'occupazione di memoria; questi rappresentano parametri fondamentali per ottimizzare il sistema.

Il volume dei dati viene schematizzato nella tavola dei volumi, dove vengono riportati i concetti dello schema ed il volume previsto a regime (Tabella 2.1).

Di seguito vengono riportate, nella tavola delle operazioni (Tabella 2.2), le operazioni più comuni, caratterizzate dalla frequenza prevista e dal di operazione. Nei casi analizzati le operazioni sono tutte di tipo interattivo.

I dati delle tabelle vengono utilizzati per ristrutturare lo schema concettuale, come illustrato nella prossima sottosezione.

Concetto	Tipo	Volume
user	E	500
subject	E	40
group	E	120
project	E	1500
script	E	5000
version	E	20000
to_do_list	E	1500
setting	E	500
feedback	E	2000
company	E	10
meets	E	20
historical_project	E	30
historical_script	E	30
manages	R	40
attend	R	320
membership	R	370
handles	R	120
development	R	1500
composed	R	5000
store	R	20000
ownership	R	1500
management	R	500
send	R	2000
belongs	R	400
participant	R	125

Tabella 2.1: Tavola dei volumi

Operazione	**Frequenza**
Inserimento nuovo utente	1 al giorno
Modifica profilo utente	2 a settimana
Eliminazione utente	1 al mese
Visualizzazione profilo utente	10 a settimana
Visualizzazione di tutti gli utenti	2 al giorno
Visualizzazione di tutti gli utenti di un istituto	1 al giorno
Visualizzazione di tutti gli utenti di una materia	3 al giorno
Inserimento nuova materia	40 l'anno
Modifica materia	10 l'anno
Eliminazione materia	3 l'anno
Visualizzazione materia	4 a settimana
Creazione nuovo gruppo	3 a settimana
Modifica gruppo	2 al mese
Eliminazione gruppo	3 l'anno
Visualizzazione gruppo	10 al giorno
Visualizzazione gruppi di una materia	5 al giorno
Creazione nuovo progetto	5 al giorno
Modifica progetto	3 a settimana
Visualizzazione progetto	20 al giorno
Visualizzazione progetti di un utente	10 al giorno
Visualizzazione progetti di un gruppo	10 al giorno
Eliminazione progetto	2 al mese
Creazione script	3 a settimana
Modifica script	1000 al giorno
Eliminazione script	2 a settimana
Visualizzazione script	20 al giorno
Creazione nuova versione	1000 al giorno
Eliminazione versione	1000 a settimana
Creazione nota	3 al giorno
Modifica nota	2 al giorno
Eliminazione nota	2 al giorno
Creazione feedback	1 al giorno
Modifica impostazioni	2 al mese
Creazione conferenza	10 al mese
Modifica conferenza	3 al mese
Eliminazione conferenza	10 al mese
Creazione storico progetto	4 al mese
Creazione storico script	16 al mese

Tabella 2.2: Tavola delle operazioni

2.2.2 Ristrutturazione dello schema concettuale

La ristrutturazione dello schema E/R, articolata in fasi successive, nel caso di EduCode richiede le seguenti attività:

- *Analisi delle ridondanze*: una ridondanza corrisponde alla presenza di un dato che può essere derivato; essa può presentare vantaggi e svantaggi. Il vantaggio consiste in una riduzione degli accessi, mentre gli svantaggi riguardano l'utilizzo della memoria. Per stabilire la necessità o meno di mantere una ridondanza va valutato il costo dell'operazione e l'occupazione di memoria che coinvolgono il dato interessato nel caso con e senza ridondanza.
- *Eliminazione delle gerarchie*: poiché le generalizzazioni non possono essere rappresentate dai DBMS tradizionali, è necessario risolverle. A tal fine sono previste tre opzioni:
  - *Accorpamento delle entità figlie nel genitore*: le entità figlie vengono eliminate e le loro proprietà vengono aggiunte all'entità genitore, insieme ad un ulteriore attributo per indicare il tipo di occorrenza trattata.
  - *Accorpamento del genitore nelle figlie*: l'entità genitore viene eliminata ed i suoi attributi vengono aggiunti alle figlie. Queste ultime ereditano anche le relazioni del genitore.
  - *Sostituzione della generalizzazione con associazioni*: vengono legate le entità figlie all'entità genitore tramite associazioni uno a uno.
- *Eliminazione degli attributi multivalore*: rappresenta un particolare tipo di partizionamento, utilizzato per eliminare gli attributi multivalore.

Analisi delle ridondanze

Lo schema E/R analizzato non presenta ridondanze. Dall'esame delle tavole dei volumi e delle operazioni non risulta necessario introdurre alcuna ridondanza.

Eliminazione gerarchie

Lo schema E/R prodotto contiene al proprio interno una gerarchia che, nel modello logico, deve essere opportunamente rimossa. Essa riguarda l'entità *user*; si è scelto di accorpare le entità figlie nel genitore; all'entità *user* vengono aggiunti il campo *company*, caratteristico degli utenti convenzionati, ed il campo *role*, per distinguere la tipologia di utente (Figura 2.11).

Eliminazione attributi multivalore

Lo schema concettuale presenta un solo attributo multivalore per l'indirizzo dell'ente, composto da: *indirizzo*, *numero civico*, *cap*, *città e provincia*; vista la semplicità del campo si è scelto di raggruppare i valori in un unico campo *address* presente nell'entità company (Figura 2.12).

2.2.3 Schema E-R ristrutturato

Terminata la ristrutturazione dello schema si ottiene lo schema E/R ristrutturato (Figura 2.13). Quest'ultimo verrà usato nel processo di traduzione verso il modello relazionale.

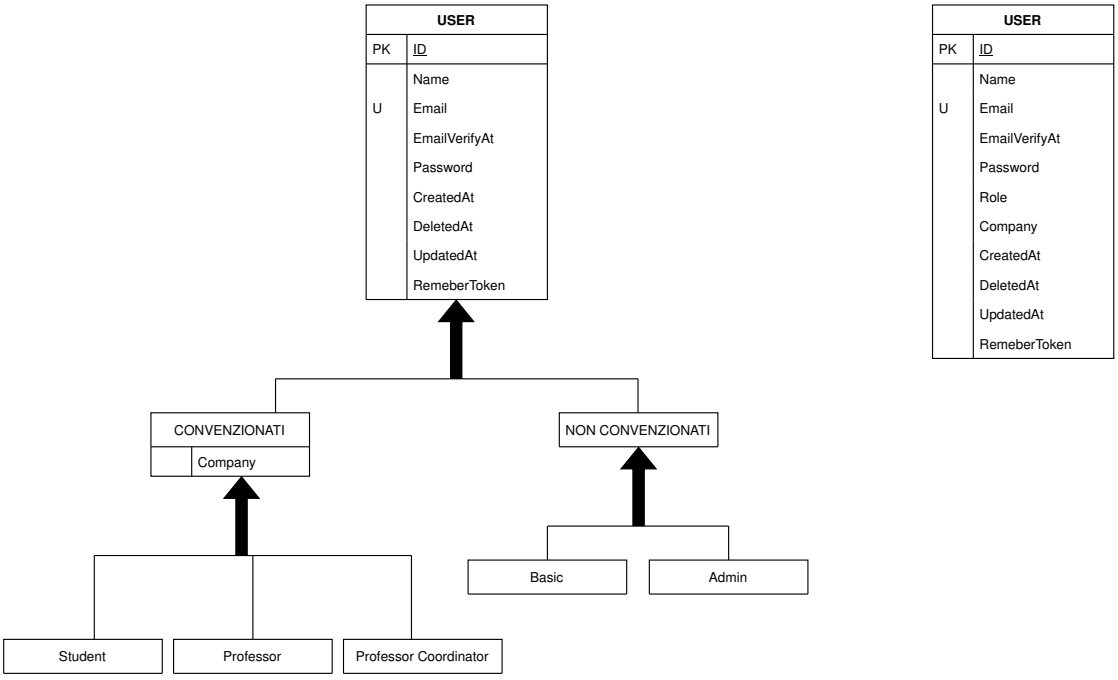


Figura 2.11: Eliminazione delle gerarchie

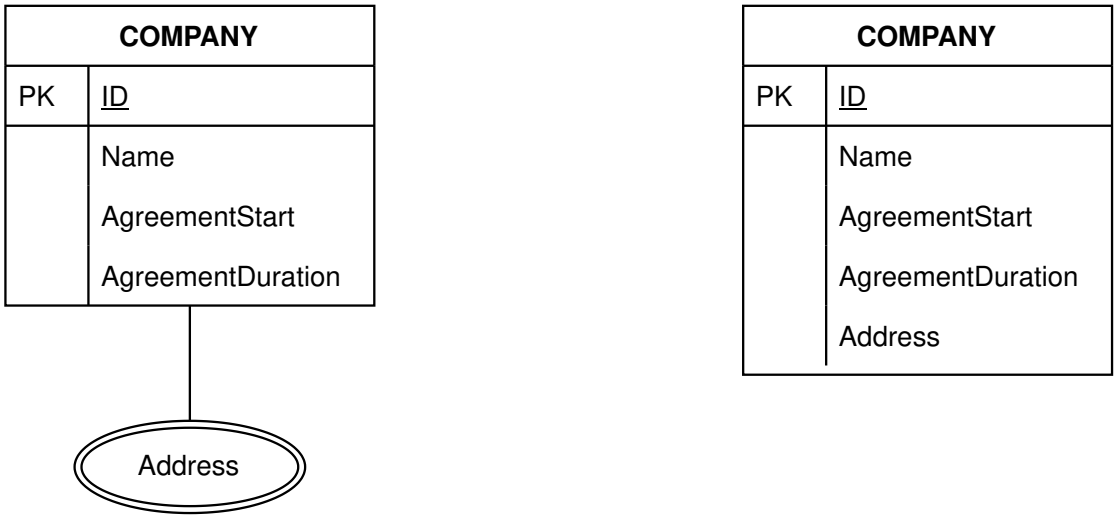


Figura 2.12: Eliminazione degli attributi multivalore

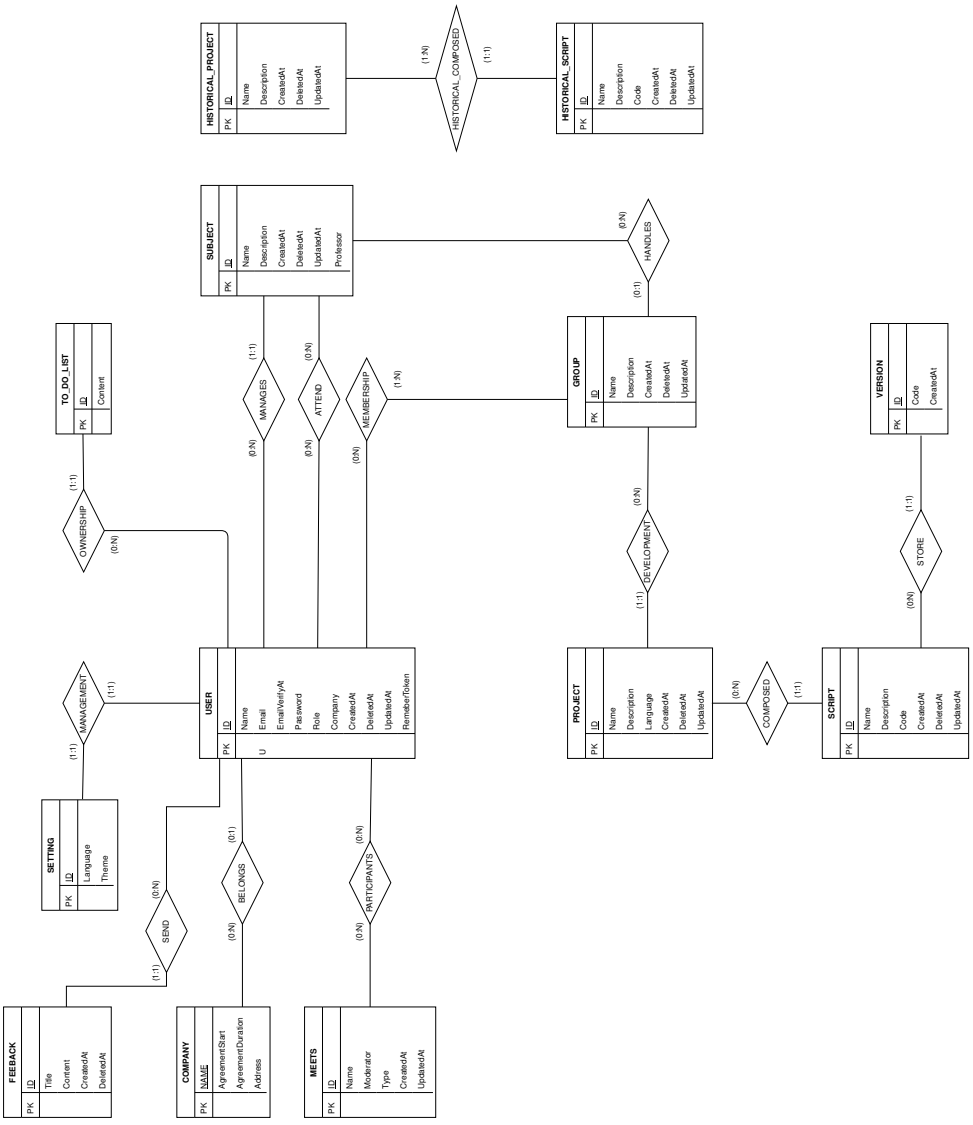


Figura 2.13: Schema E/R ristrutturato

2.2.4 Normalizzazione

La normalizzazione è un procedimento mirato all'eliminazione della ridondanza informativa e del rischio di incoerenza dello schema E-R. Tale processo si fonda sul criterio della dipendenza funzionale: se una relazione presenta più concetti tra loro indipendenti, la si decompone in relazioni più piccole, una per ogni concetto. Esistono vari livelli di normalizzazione (forme normali) che certificano la qualità dello schema. Ogni forma normale è più elevata della precedente, in quanto incorpora le caratteristiche della forma inferiore e applica ulteriori regole per la realizzazione di un progetto più efficiente. Lo schema di EduCode presenta associazioni binarie già nella forma normale di Boyce- Codd, che presuppone la rispondenza delle relazioni alla 1FN, alla 2FN e alla 3FN.

2.2.5 Traduzione verso il modello relazionale

La traduzione verso il modello logico scelto rappresenta la seconda fase della progettazione logica. Consiste nel costruire, a partire dallo schema E/R ristrutturato, lo schema logico equivalente. Per EduCode il modello di riferimento è il modello relazionale. Il processo di traduzione consiste nell'analizzare le entità e le associazioni e tradurle nei costrutti corrispondendi del modello relazionale. Il risultato della traduzione è riportato nella Tabella 2.3.

Entità - relazione	**Traduzione**	Vincolo
User	user(id, name, email, email_verify_at, password, role, company, created_at, deleted_at, updated_at, remember_token)	company->company.id
Subject	subject(id, name, description, created_at, deleted_at, updated_at, professor)	professor->user.id
Attend	attend(user_id, subject_id)	user_id->user.id, subject_id->subject.id
Group	group(id, name, description, subject_id, created_at, deleted_at, updated_at, individual_lock)	subject_id->subject.id
Membership	membership(user_id, group_id)	user_id->user.id, group_id->group.id
Project	project(id, name, description, group_id, created_at, deleted_at, updated_at, lanaguage)	group_id->group.id
Script	script(id, name, description, code, project_id, created_at, deleted_at, updated_at)	project_id->project.id
Version	version(id, script_id, code created_at, updated_at)	script_id->script.id
Company	company(name, agreement_start, agreement_duration, address, created_at, deleted_at, updated_at)	-
Setting	setting(id, language, theme, user_id)	user_id->user.id
ToDoList	to_do_list(id, content, user_id, created_at, deleted_at, updated_at)	user_id->user.id
Feedback	feedback(id, title, content, user_id, created_at, deleted_at, updated_at)	user_id->user.id
Meet	meet(id, name, moderator, type)	
Participants	participants(id, user_id, meet_id)	user_id->user.id, meet_id->meet.id
Historical Project	historical_project(id, name, description, created_at, deleted_at, updated_at, lanaguage)	-
Historical Script	historical_script(id, name, description, code, project_id, created_at, deleted_at, updated_at)	project_id->project.id

Tabella 2.3: Schema relazionale

CAPITOLO 3

Progettazione della componente applicativa

In questo capitolo vengono illustrate le fasi della progettazione della componente applicativa, dalla user interface alla scelta delle tecnologie e dei linguaggi da utilizzare.

3.1 Realizzazione Mockup

Il termine "mockup", letteralmente "modello", assume significati diversi a seconda del contesto di utilizzo. Nello sviluppo di un'applicazione i mockup permettono di mostrare, al committente o all'utente finale, una bozza dell'applicazione prima che venga sviluppata e servono di riferimento dal team di sviluppo frontend per programmarne l'interfaccia grafica. I mockup hanno una fedeltà variabile rispetto al prodotto finale.

I mockup realizzati per EduCode servono per fissarne i contenuti e definirne l'interfaccia grafica. Il layout dei contenuti determina la loro posizione all'interno dell'applicazione, stabilisce il contenuto di ogni pagina e, in conseguenza, il numero di passi necessari all'utente per eseguire una funzione. L'individuazione della palette dei colori e degli elementi tipografici - tipi di font e opportune dimensioni - da utilizzare favorisce l'ambientamento dell'utente e contribuisce a rendere l'applicazione realmente accessibile.

Tutti i mockup di EduCode hanno la stessa struttura di base (scheletro) che fissa la divisione di ciascuna pagina in tre sezioni (Figura 3.1):

- *Head menu*: contiene le informazioni dell'utente; è adatto per contenere una barra di ricerca e una sezione di notifiche.
- *Left sidebar*: contiene tutto ciò che riguarda la navigazione tra le pagine; all'occorrenza può essere nascosto per lasciare maggiore spazio ai contenuti.
- *Content*: è l'unica parte che varia al cambiamento della pagina; rappresenta il cuore dell'applicazione poiché al suo interno vengono organizzati i contenuti di ogni sezione.

In tutte le pagine i colori rispettano una logica comune. I pulsanti possono essere verdi, gialli, rossi e blu: Il verde serve per confermare un salvataggio o l'invio dei dati, il giallo viene usato per le modifiche, il rosso per l'eliminazione ed il blu negli altri casi. Anche gli altri contenuti hanno la loro palette di colori; ad esempio gli account utente, se sono attivi, vengono indicati in verde; per i sospesi viene visualizzata in rosso la data di sospensione. Come sfondo, per garantire visibilità ai contenuti, si usa il bianco, mentre i dettagli più piccoli

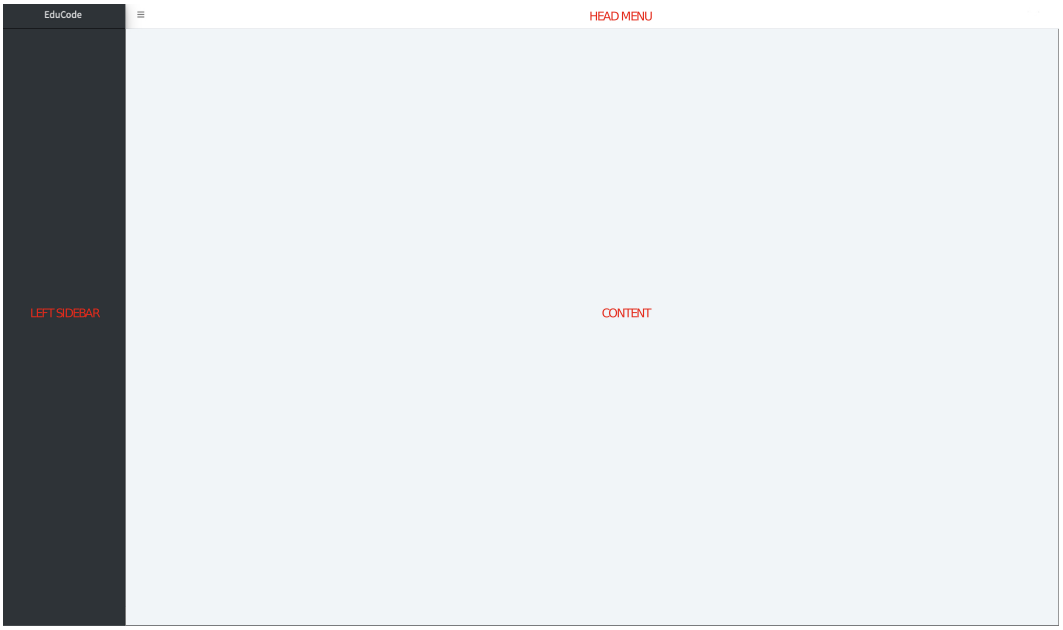


Figura 3.1: Mockup delle sezioni

vengono rappresentati in nero. Su sfondo bianco, le scritte sono nere o grigie, mentre su sfondo scuro sono sempre grigie. Nei pulsanti a sfondo pieno le scritte interne sono sempre bianche mentre nei pulsanti a sfondo bianco le scritte interne sono dello stesso colore del pulsante ed, al passaggio del mouse, i caratteri diventano bianchi e lo sfondo prende il colore del pulsante.

A partire dallo scheletro vengono, poi, definiti i mockup di pagina. Una volta effettuato il login, l'utente visualizza la dashboard e i contenuti della pagina differenziati in base al suo ruolo. Più specificatamente:

- *L'utente guest* visualizza solo le note ed il form per l'invio dei feeback.
- *Lo studente ed il professore* visualizzano la stessa dashboard del *guest* con l'aggiunta dei progetti (Figura 3.2).
- *L'amministratore del sistema* visualizza nella dashboard lo stato degli utenti iscritti, i feedback inviati, i propri progetti e le note (Figura 3.3).

Per favorire l'esperienza utente è possibile ridurre lo spazio dedicato al menù laterale, così da lasciare più spazio per i contenuti principali (Figura 3.4).

Le principali funzioni dell'applicazione sono illustrate da mockup specifici, di seguito riportati:

- *Livecoding* (Figura 3.5): visualizza, nell'area dei contenuti, due blocchi. Il primo, contiene l'editor per scrivere il codice, i comandi rapidi per muoversi tra le modifiche del codice sorgente ed il pulsante per mandare in esecuzione il codice. Il secondo contiene il terminale per visualizzare i risultati. Nell'area del terminale sono presenti il pulsante per cancellare il contenuto del terminale ed una *select* per scegliere il compilatore da utilizzare. Per entrambi i blocchi sono possibili la riduzione ad icona e la disposizione a schermo intero. Nella versione mobile (Figura 3.6) la struttura del sito web si adatta alle dimensioni del display, i contenuti vengono disposti in colonna per garantire la giusta leggibilità, ed il menu viene ridotto a icona.

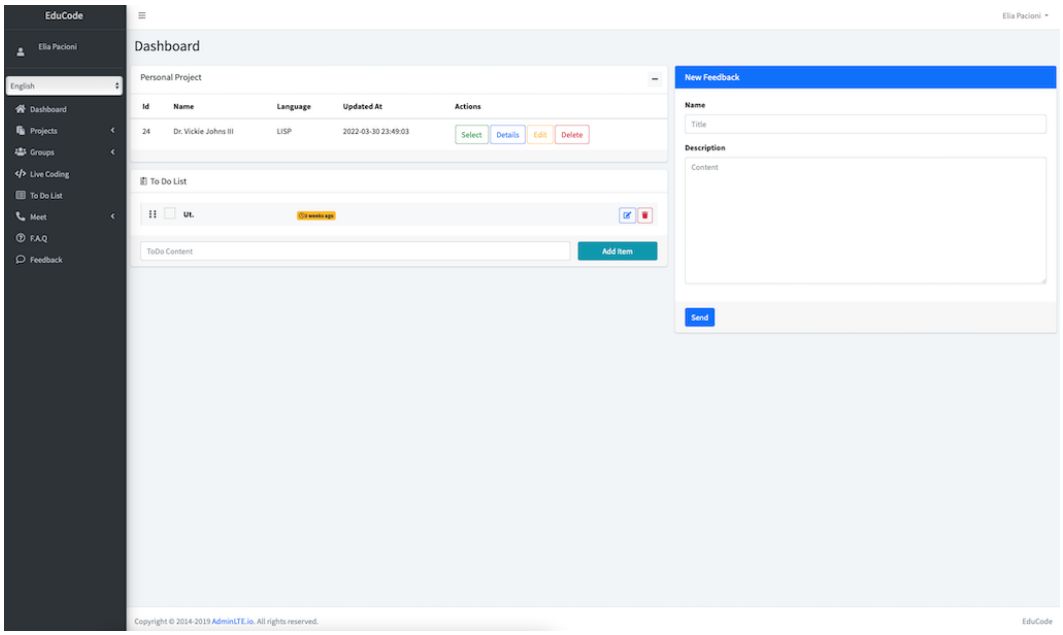


Figura 3.2: Mockup della dashboard dello studente

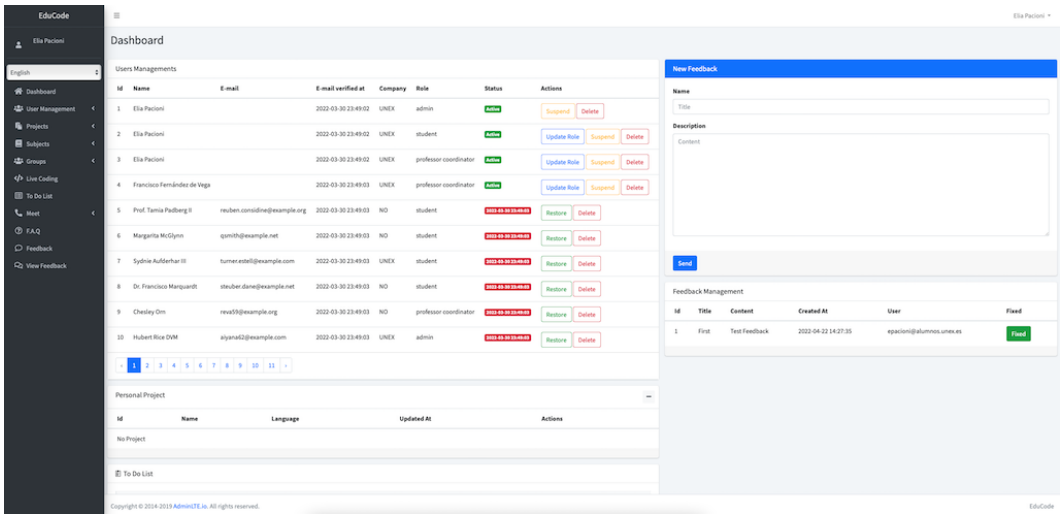


Figura 3.3: Mockup della dashboard dell'amministratore

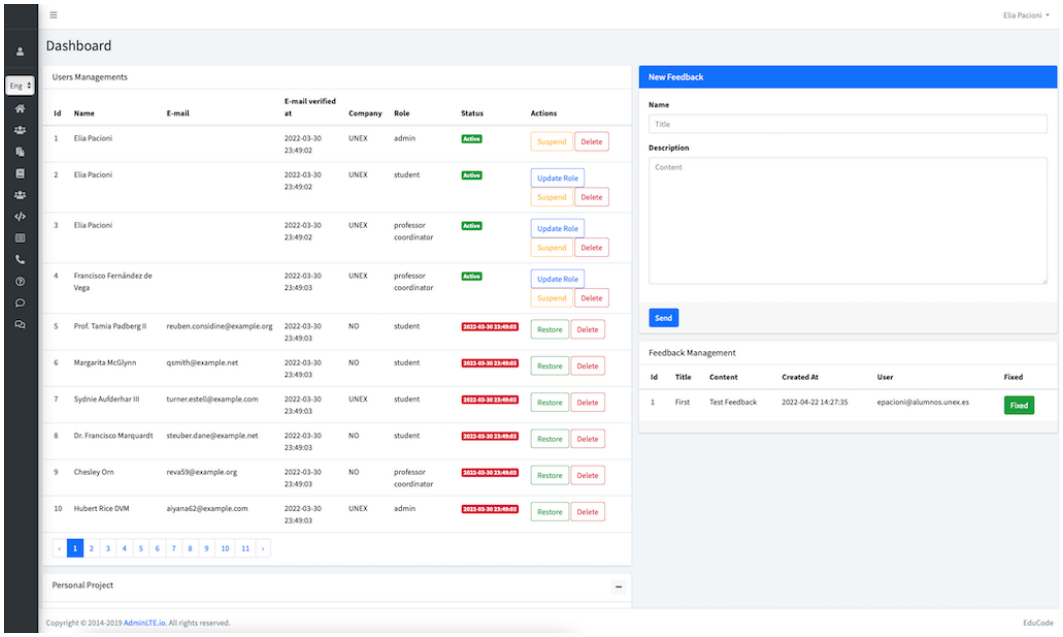


Figura 3.4: Mockup della dashboard dell'amministratore con menu ridotto

- *Profilo utente* (Figura 3.7): visualizza le informazioni del profilo e offre la possibilità di modificare ed eliminare i dati.
- *Dettagli progetto* (Figura 3.8): visualizza le informazioni del progetto, nell'area in alto sono presenti dei box che indicano il numero di sviluppatori del progetto, il numero di script ed il linguaggio con cui viene sviluppato il progetto. Nella parte bassa vengono visualizzati gli script; per ogni script vengono visualizzate le versioni, che possono essere ripristinate o eliminate. Nell'area a destra viene visualizzata la descrizione del progetto con l'id, la data ed il codice dello script selezionato. La sezione del codice, attraverso la funzione di monitoraggio, mostra in tempo reale le modifiche effettuate dagli altri sviluppatori sullo script selezionato. Attraverso questa pagina è possibile eliminare il progetto ed entrare nella *sessione di coding*.
- *Sessione di coding* (Figura 3.9): permette di programmare ed eseguire il codice del progetto selezionato. Nella parte alta è presente l'elenco degli script; una volta scelto lo script su cui lavorare è possibile ridurre la tab ad icona così da avere più spazio per il codice. Sotto gli script viene visualizzato l'editor che è lo stesso della sezione *livecoding*. Nella parte destra è presente il blocco del terminale e l'elenco delle note. In questa pagina, inoltre, è possibile organizzare la tab come si preferisce, semplicemente trascinando la tab scelta nel punto in cui si vuole ancorare (Figura 3.10). Come per il livecoding, anche in questa pagina, per la versione mobile (Figure 3.11 e 3.12) i contenuti vengono disposti in colonna.

3.2 Architettura e pattern

L'architettura software di un sistema è l'insieme delle strutture che comprendono gli elementi software, le relazioni tra di essi e le loro proprietà.

La costruzione di un'architettura efficace che permetta una rapida consegna del prodotto, affrontando al contempo gli obiettivi a lungo termine, può rivelarsi una sfida complessa;

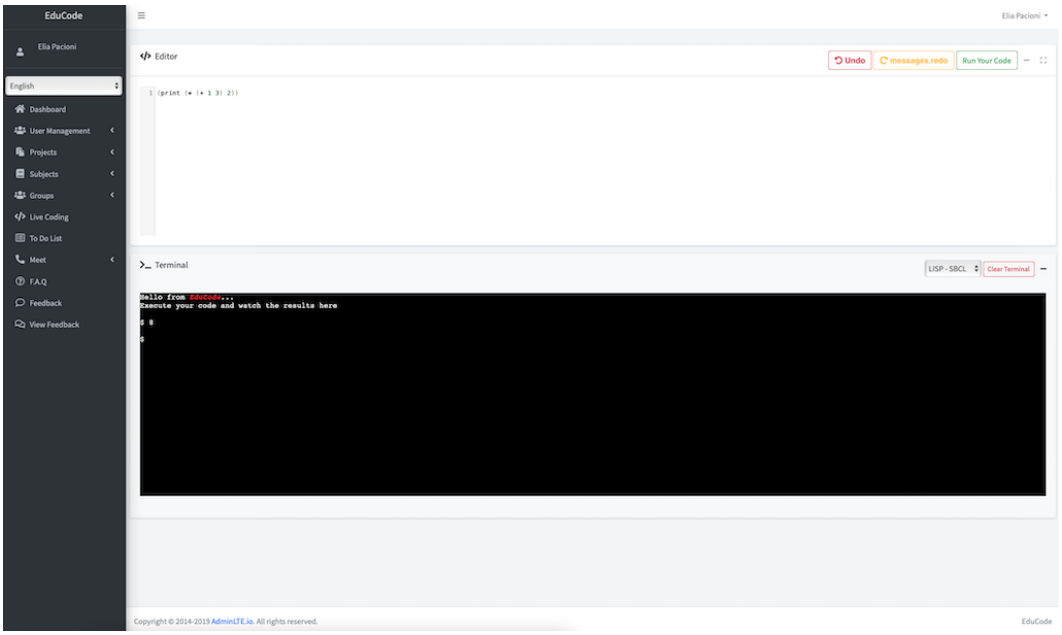


Figura 3.5: Mockup della pagina di livecoding

Figura 3.6: Mockup della pagina di livecoding mobile

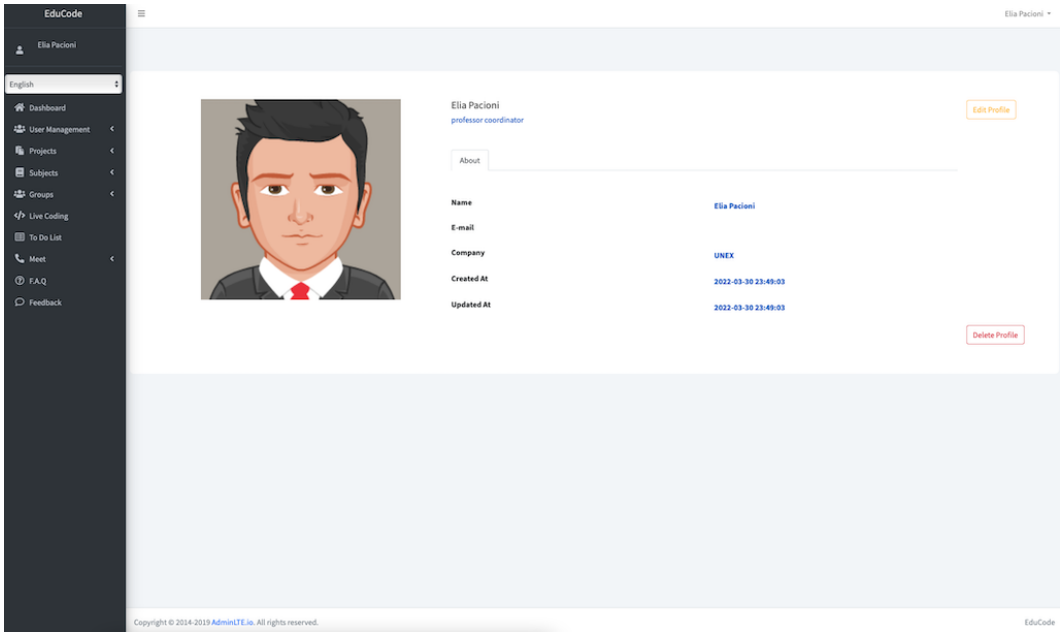


Figura 3.7: Mockup della pagina relativa al profilo utente

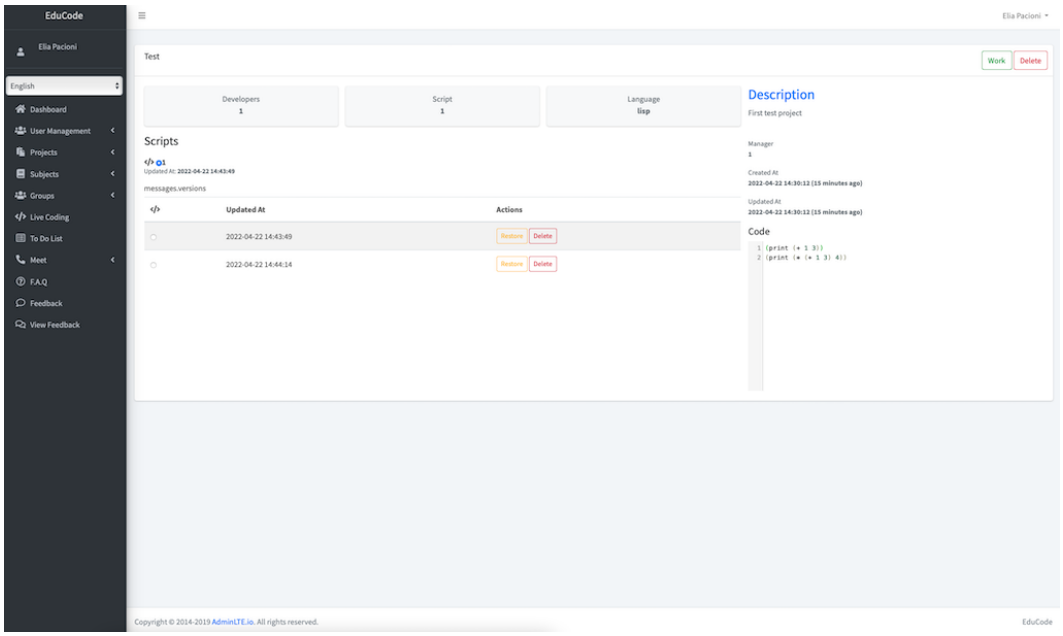


Figura 3.8: Mockup della pagina relativa ai dettagli e alla supervisione del progetto

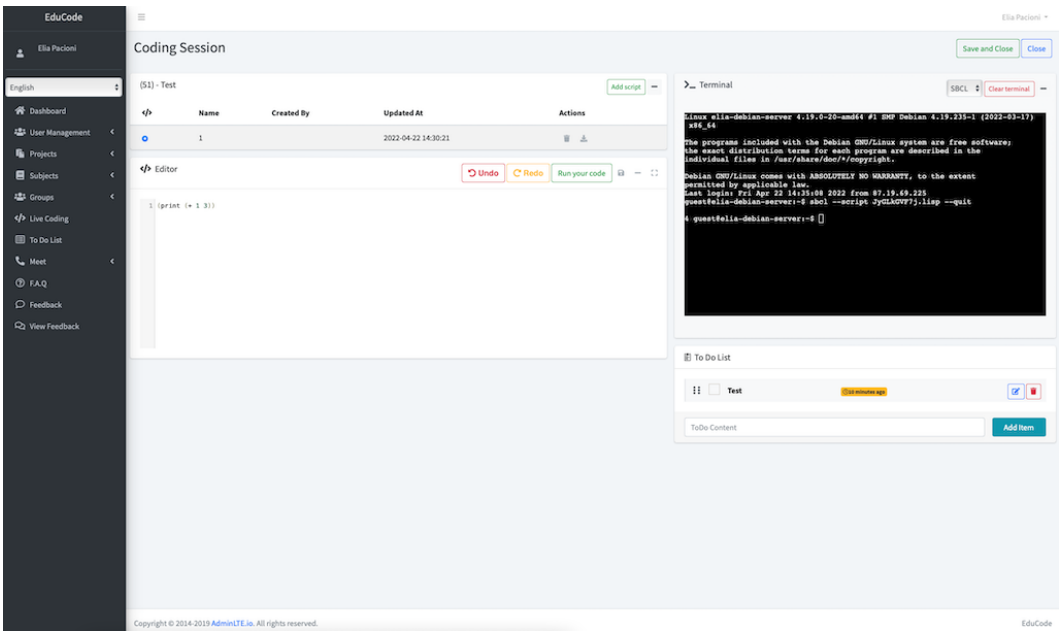


Figura 3.9: Mockup della pagina relativa alla sessione di coding

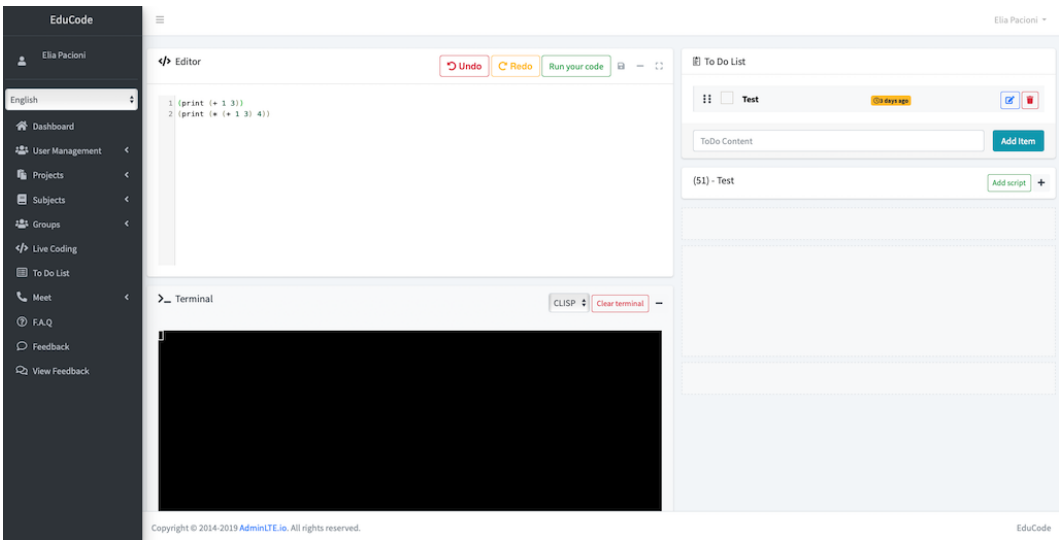


Figura 3.10: Mockup dell'organizzazione tab nella pagina sessione di coding

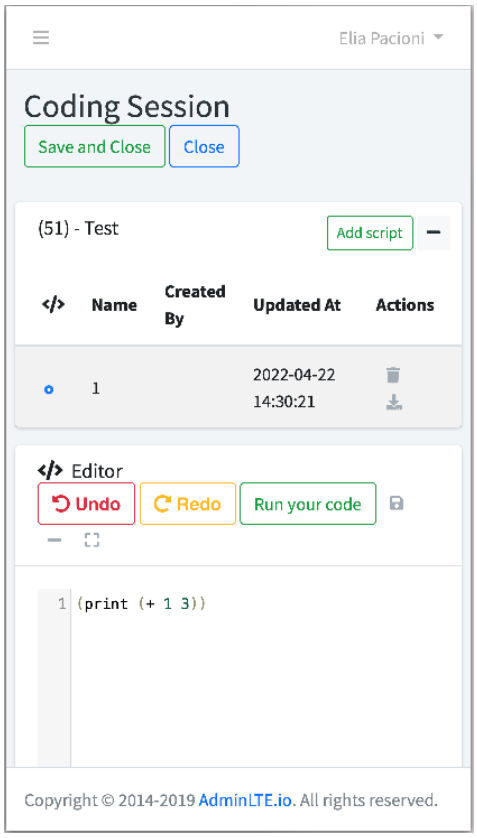


Figura 3.11: Mockup della pagina relativa alla sessione di coding nella versione mobile (prima parte)

Figura 3.12: Mockup della pagina relativa alla sessione di coding nella versione mobile (seconda parte)

non riuscire a identificare, dare priorità e gestire i trade-off tra le qualità architettoniche significative spesso porta a ritardi nel progetto, o peggio, a costose rielaborazioni.

L'assunto della *Legge di Conway (1968) - "Qualsiasi organizzazione che progetta un sistema (definito in senso lato) produrrà un progetto la cui struttura è una copia della struttura di comunicazione dell'organizzazione"* - mette in risalto la dipendenza dell'architettura software del progetto dall'organizzazione del team che deve svilupparlo.

Secondo la cosiddetta *Legge di Conway "inversa"*, la struttura dei team di un'organizzazione è determinata dall'architettura dei sistemi che produce.

Nel disegnare l'architettura software alla base di EduCode si è tenuto conto sia della *Legge di Conway* sia della sua inversa, arrivando, così, ad un'architettura ibrida: monolitica, per l'applicazione principale; a microservizi, per l'esecuzione del codice e la supervisione in tempo reale. L'architettura monolitica, metaforicamente ispirata dalla figura del monolite, è caratterizzata dal forte accoppiamento dei componenti che garantisce autonomia all'applicazione. L'architettura monolitica è stata scelta per l'applicazione principale per i vantaggi che presenta: massimizzazione delle performance, sicurezza dell'applicazione, efficacia nel debug. Infatti, con la comunicazione tra i componenti ridotta al minimo, si ottiene un'applicazione più rapida, soprattutto in fase di avvio; con un'applicazione incentrata in un unico blocco si agevola il controllo delle autorizzazioni ed il backup dei dati; infine, poiché tutto il sistema è sotto il suo controllo, lo sviluppatore può effettuare un debug efficace.

A tali vantaggi si contrappongono gli svantaggi rappresentati dalle difficoltà degli aggiornamenti e dalla ridotta scalabilità orizzontale; per aggiornare l'applicazione è necessario mandare offline l'intero sistema, aggiornare tutto il monolite ed, al termine, tornare online; in alcuni contesti questo timeout è inaccettabile.

Per erogare le funzionalità di esecuzione del codice e di supervisione in tempo reale, che sono le più significative e, computazionalmente parlando, le più dispendiose di EduCode, è stata implementata un'architettura a microservizi che offre un'ottima scalabilità orizzontale. Tale scelta permette, quindi, all'applicazione di scalare orizzontalmente e garantisce, anche, potenza di calcolo al crescere degli utenti ed all'aumentare della complessità computazionale dei codici eseguiti.

L'applicazione monolitica fa uso del *pattern architetturale MVC* (Figura 3.13) che consiste nel separare la logica di presentazione dei dati dalla logica di business. Il pattern si basa sui tre componenti:

- *Model*: il modello rappresenta la parte dell'applicazione che si occupa di manipolare i dati.
- *View*: la vista rappresenta l'interfaccia grafica dell'applicazione.
- *Controller*: racchiude la logica applicativa del software, la gestione degli eventi e mette in comunicazione la *view* e il model.

Il pattern MVC è molto diffuso nella programmazione ad oggetti perché, anche se aumenta i tempi necessari allo sviluppo del software, ne semplifica la manutenzione e l'aggiornamento.

3.3 Scelta dei linguaggi di programmazione

Quando si inizia un nuovo progetto la scelta del linguaggio di programmazione si rivela sempre complessa perché non ci sono linguaggi migliori o peggiori: semplicemente alcuni linguaggi si adattano meglio di altri ad un determinato tipo di progetto. Nella scelta del linguaggio intervengono molteplici parametri, ovvero l'architettura del progetto, le prestazioni

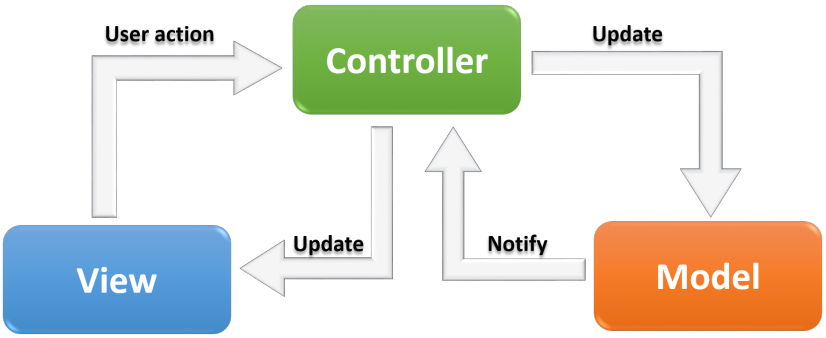


Figura 3.13: Modello MVC

da raggiungere, le conoscenze del team di sviluppo ed il tempo a disposizione per completare un progetto.

Analizzando le caratteristiche elencate per lo sviluppo di EduCode il linguaggio scelto è il *PHP*, acronimo ricorsivo di *Hypertext Preprocessor*; questo è un linguaggio di scripting interpretato, creato per la programmazione di pagine web dinamiche; è distribuito con la licenza *PHP License*, approvata come *Open Source* dalla *Open Source Initiative*[1]. La scelta di *PHP* è dovuta principalmente alla rapidità dello sviluppo; la comunità *PHP* ha sviluppato e rilasciato, in modo *open source*, molti framework e pacchetti per aiutare lo sviluppo anche dei progetti più piccoli e con meno risorse. Per lo sviluppo frontend vengono utilizzati il linguaggio di programmazione *Javascript* ed i linguaggi di markup *HTML* e *CSS*.

3.4 Tecnologie utilizzate

L'applicazione monolitica alla base di EduCode è sviluppata con il *framework open source Laravel*[2], basato sul pattern MVC, in accordo con le scelte progettuali effettuate.

L'utilizzo di un framework fornisce una struttura e un punto di partenza per creare l'applicazione, permettendo al programmatore di concentrarsi sulla creazione della logica di business senza preoccuparsi della struttura sottostante. Il frontend dell'applicazione è sviluppato a partire dal template web *AdminLte*, che fa uso delle librerie *Bootstrap* e *jQuery*. *Bootstrap* è una libreria open source progettata per supportare lo sviluppo rapido di interfacce grafiche responsive; è composta da due parti: la prima contiene i componenti *CSS*, la seconda raccoglie *plugin javascript* per animare i componenti grafici e renderli più accattivanti. *jQuery* è una libreria javascript leggera e molto veloce; viene usata per manipolare il *DOM*[3] nella pagina *HTML*, semplifica la gestione degli eventi e permette di animare le pagine *HTML*. In combinazione con jQuery viene utilizzata la tecnologia *AJAX*, acronimo che sta per Asynchronous JavaScript and XML, che aiuta a caricare dati dal server senza l'aggiornamento della pagina del browser. *JQuery* fornisce un ricco set di metodi *AJAX* per supportare lo sviluppo di applicazioni web e, di conseguenza, migliorare l'esperienza utente. Infine, per la fase di test e successivo deploy, EduCode usa un *VPS*[4] con sistema operativo *Debian 10* e server web *NGINX*.

[1] Organizzazione promotrice del software open source, membro dell'istituto europeo per le norme di telecomunicazione.

[2] Framework PHP creato da Taylor Otwell nel 2011.

[3] Letterlamente *Document Object Model*, è un modello ad oggetti del documento. È lo standard ufficiale del *W3C* per la rappresentazione dei documenti.

[4] Un VPS, o server privato virtuale, è una forma di cloud hosting multi-tenant in cui le risorse server virtualizzate sono rese disponibili a un utente finale su Internet tramite un cloud o un fornitore di hosting.

CAPITOLO 4

Implementazione del sistema

In questo capitolo viene illustrato il processo di codifica attraverso l'analisi della struttura dell'applicazione finale e delle molteplici componenti.

4.1 Struttura dell'applicazione

EduCode utilizza la struttura fornita dal framework *Laravel* per le applicazioni. Per comprendere completamente la struttura di un'applicazione *Laravel*, e di conseguenza quella di EduCode, è necessario esaminare il flusso di lavoro che sviluppa una richiesta *HTTP*. Come illustrato in Figura 4.1, l'utente genera una richiesta *HTTP* che viene ricevuta dal server web ed inoltrata all'applicazione *Laravel*. Il punto d'ingresso per tutte le richieste è il file *index.php* situato nella cartella *public*; esso carica la definizione dell'autoloader da Composer e recupera un'istanza dell'applicazione creata dal file *bootstrap/app.php*. Successivamente la richiesta viene inviata al *kernel HTTP*, situato in *app/Http/Kernel.php*, che si occupa di definire:

- i *bootstrapper* necessari a configurare le variabili d'ambiente, il gestore delle eccezioni ed i *service provider*;
- i *middleware HTTP* che devono essere oltrepassati da tutte le richieste *HTTP*.

L'esecuzione dei *service provider* è tra le fasi più importanti dell'avvio dell'applicazione perché responsabile dell'avvio dei componenti del framework, tra cui il database, le code, la validazione ed il router. I *service provider* vengono letti dal file di configurazione */config/app.php*, registrati e avviati dal metodo *boot* .

Una volta effettuate queste operazioni, la richiesta viene passata al *router* che la gestisce tramite un metodo o proprio o del *controller*. La risposta generata torna indietro attraverso il *middleware* della rotta. Una volta superato il *middleware*, la risposta passa al metodo *handle* del *kernel HTTP* che la restituisce al file *index.php*. Infine, quest'ultimo esegue il metodo *send* e, tramite il server web, invia la risposta all'utente.

Sebbene sia consentita senza restrizioni la modifica della struttura applicativa, la soluzione predefinita si adatta perfettamente alle esigenze di EduCode. La *root* dell'applicazione ha la seguente struttura:

- *App*: è la cartella principale dell'applicazione; la maggior parte dei file creati vanno posizionati al suo interno. A sua volta essa è strutturata in sottocartelle (analizzate

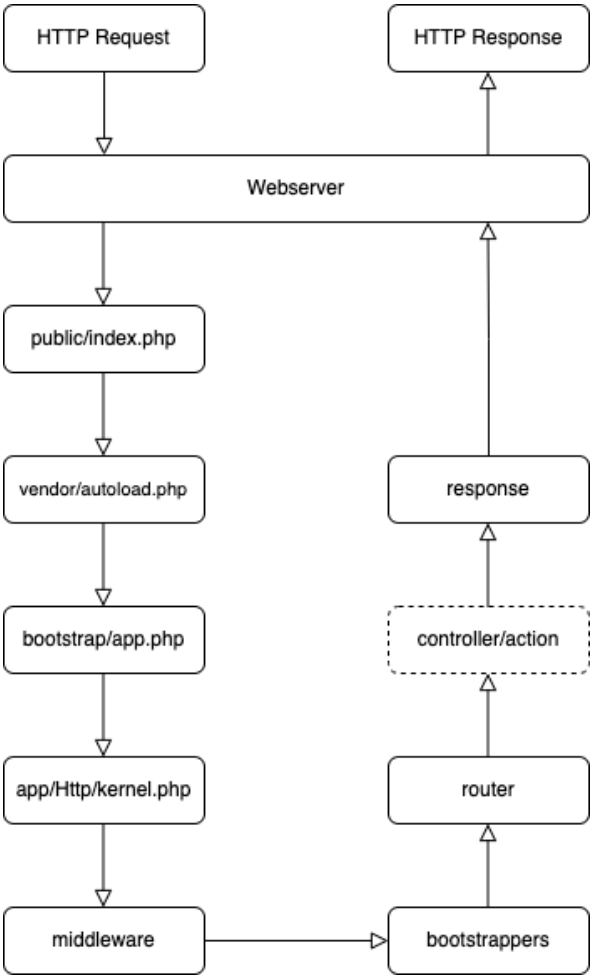


Figura 4.1: Flusso di lavoro di una richiesta *HTTP*

in dettaglio nelle sezioni successive): *console*, *events*, *exceptions*, *http*, *models*, *policies*, *provider*.

- *Bootstrap*: contiene il file *app.php* che avvia il framework e la cartella per la gestione della cache.
- *Config*: è la cartella contenente i file di configurazione dell'applicazione, fondamentale in applicazioni con un alto grado di personalizzazione.
- *Database*: è la cartella contenente i file riguardanti la creazione del database e l'inserimento dei dati di test.
- *Node_modules*: è la cartella contenente le dipendenze *NPM*[1], non sempre è presente nelle applicazioni *Laravel*.
- *Public*: è la cartella contenente il file *index.php* che rappresenta il punto d'ingresso dell'applicazione ospitando anche le immagini e le risorse pubbliche dell'applicazione.
- *Resources*: è la cartella contenente i file riguardanti il frontend.
- *Routes*: è la cartella contenente le definizioni delle rotte dell'applicazione. Laravel fornisce la seguente organizzazione predefinita delle rotte:

[1]NPM è un gestore di dipendenze, inizialmente per node.js e, successivamente, per javascript.

  - *Api*: il file *api.php* contiene le rotte che il *RouteServiceProvider* mette nel *gruppo middleware api*. Esse vengono utilizzate per esporre delle API RESTful.
  - *Channels*: il file *channels.php* viene usato per la gestione dei canali di trasmissione degli eventi che, nel caso di EduCode, sono i *websocket*.
  - *Console*: nel file *console.php* si possono definire i comandi della console. Questo file non definisce rotte HTTP, bensì punti di ingresso (rotte) per la console dell'applicazione.
  - *Web*: il file *web.php* contiene le rotte che il RouteServiceProvider inserisce nel *gruppo middleware web*, quest'ultimo fornisce lo stato della sessione, la protezione *CSRF*[2] e la crittografia dei cookie.

- *Storage*: è la cartella contenente i log dell'applicazione, i file salvati dall'utente ed i file in cache o in sessione.
- *Tests*: è la cartella contenente i testi automatici.
- *Vendor*: è la cartella contenente le dipendenze *PHP* gestite da *Composer*[3].
- *.env*: file contenente le configurazioni dell'ambiente di sviluppo; all'interno di questo file sono presenti i parametri di connessione al database, le credenziali per la comunicazioni websocket, le impostazioni del server e-mail e tutti i servizi aggiuntivi che vengono configurati nell'applicazione.
- *Package.json*: contiene l'elenco delle dipendenze del gestore NPM.
- *Composer.json*: contiene l'elenco delle dipendenze del gestore Composer.

4.2 Creazione del model e uso di Eloquent ORM

Laravel consente di interfacciarsi con molti *DBMS*; le stringhe di connessione per i DBMS disponibili sono presenti nel file *config/database.php*. EduCode utilizza *MariaDB* come *DBMS* ed usa il file *.env* per definire le variabili d'ambiente associate ai parametri di connessione letti dal file di configurazione. Tali parametri sono di seguito dettagliati:

- *DB_CONNECTION*: specifica il *DBMS* utilizzato.
- *DB_HOST*: indica l'indirizzo di MariaDB.
- *DB_PORT*: indica la porta su cui è in ascolto MariaDB.
- *DB_DATABASE*: indica il nome del database su cui operare.
- *DB_USERNAME*: indica il nome utente per accedere al database.
- *DB_PASSWORD*: indica la password per accedere al database.

Terminata la fase di configurazione è possibile connettersi ed iniziare a lavorare con il database attraverso gli strumenti offerti da *Laravel*. Per la definizione dello schema logico della base di dati, *Laravel* mette a disposizione le *migrations*, uno strumento simile al controllo di versione, che agisce sulle tabelle, sulle colonne e sulle relazioni. Una *migration* è una classe che contiene i metodi specifici *up* e *down*: il metodo *up* da usare per aggiungere tabelle, colonne o

```php
class CreateCompaniesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create( table: 'companies', function (Blueprint $table) {
            $table->string( column: 'name');
            $table->primary( columns: 'name');
            $table->timestamps();
            $table->softDeletes();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists( table: 'companies');
    }
}
```

Figura 4.2: Migration della tabella *Company*

indici al database; il metodo *down*, inverso del precedente, è da usare per l'eliminazione. In Figura 4.2 viene riportato il codice della *migration* per la creazione della tabella *Company*.

Per interagire con il database sono disponibili diversi strumenti, differenti per complessità e funzionalità; la soluzione adottata da EduCode è *Eloquent ORM*[4]. Questo *ORM* permette l'astrazione del codice dal DBMS attraverso la mappatura diretta di tutte le entità presenti nel database con oggetti, detti modello; ciascuna entità corrisponde ad un oggetto e ne mantiene le relative relazioni (uno a uno, uno a molti, molti a molti). *Eloquent ORM* fornisce, anche, protezione contro *SQL injection*[5]. Per illustrarne il funzionamento, a titolo esemplificativo, viene analizzato il modello *User* (Figura 4.3)

La classe *User* estende la classe *Authenticatable* allo scopo di rendere l'utente autenticabile attraverso i meccanismi definiti da *Laravel*. Essa implementa l'interfaccia *MustVerifyEmail* per obbligare l'utente a verificare la propria e-mail prima di poter usare l'applicazione. Tra le proprietà troviamo *$fillable*, *$hidden* e *$casts*. *$fillable* permette di assegnare i valori delle proprietà in massa; essa è utile quando si recuperano i dati da un form. *$hidden* nasconde le proprietà elencate dagli array associativi usati per gestire i modelli. *$casts* esegue il cast delle proprietà contenute nell'array al valore corrispondente. All'interno della classe vengono poi definite le relazioni che l'entità *User* ha con le altre entità. La definizione di tali relazioni permette di recuperare i dati correlati attraverso l'invocazione dei metodi definiti, che mascherano al programmatore le query da eseguire. Le principali ragioni sono le seguenti:

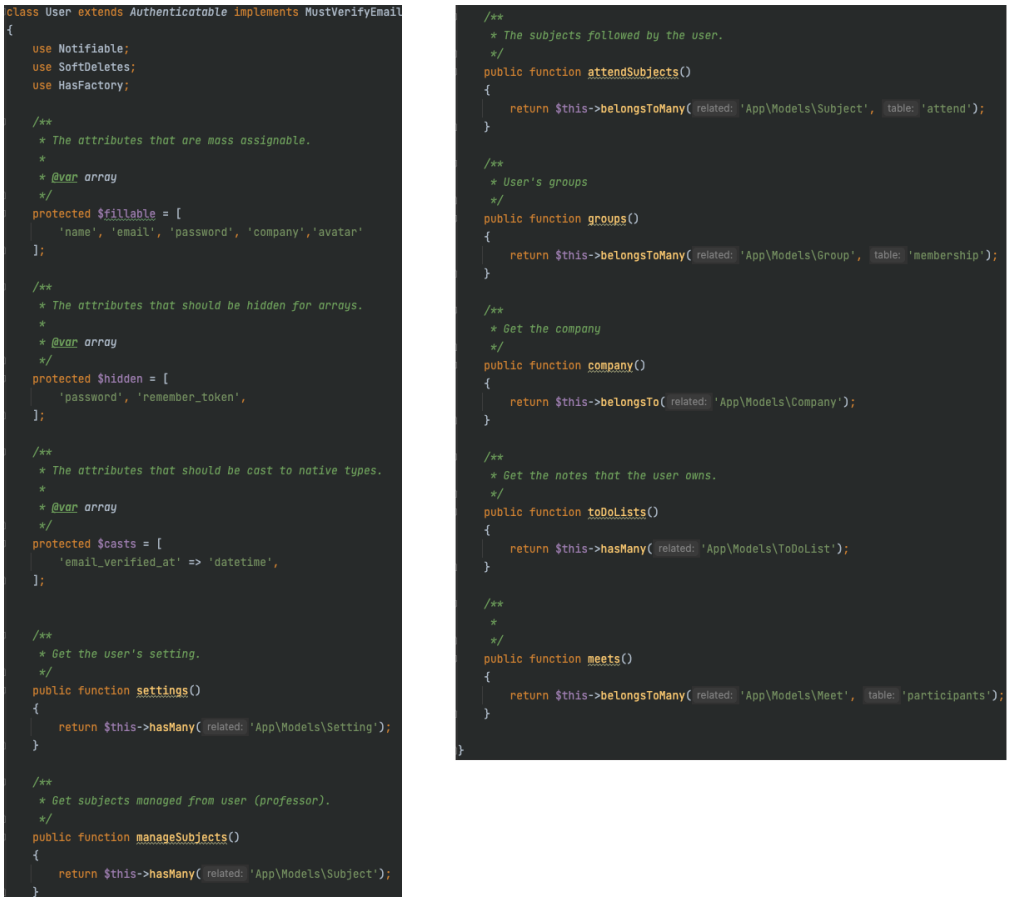
- *Settings*: mette in relazione la tabella *users* con la tabella *settings*; il metodo *$user->settings()* recupera tutte le impostazioni definite dall'utente.

[2]Cross-site request forgery è una vulnerabilità della sicurezza web che permette ad un attaccante di indurre gli utenti ad eseguire azioni che non intendono eseguire.

[3]Composer è un gestore di dipendenze PHP.

[4]ORM sta per Object-Relational Mapping; essa è una tecnica di programmazione che astrae il codice dal database.

[5]*SQL Injection* è una tecnica di iniezione di codice utilizzata per modificare o recuperare dati dai database SQL. Inserendo istruzioni SQL specializzate, un attaccante è in grado di eseguire comandi che permettono il recupero di dati dal database, la distruzione di dati sensibili, o altri comportamenti manipolativi.



```php
class User extends Authenticatable implements MustVerifyEmail
{
    use Notifiable;
    use SoftDeletes;
    use HasFactory;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name', 'email', 'password', 'company','avatar'
    ];

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password', 'remember_token',
    ];

    /**
     * The attributes that should be cast to native types.
     *
     * @var array
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];


    /**
     * Get the user's setting.
     */
    public function settings()
    {
        return $this->hasMany( related: 'App\Models\Setting');
    }

    /**
     * Get subjects managed from user (professor).
     */
    public function manageSubjects()
    {
        return $this->hasMany( related: 'App\Models\Subject');
    }

    /**
     * The subjects followed by the user.
     */
    public function attendSubjects()
    {
        return $this->belongsToMany( related: 'App\Models\Subject',  table: 'attend');
    }

    /**
     * User's groups
     */
    public function groups()
    {
        return $this->belongsToMany( related: 'App\Models\Group',  table: 'membership');
    }

    /**
     * Get the company
     */
    public function company()
    {
        return $this->belongsTo( related: 'App\Models\Company');
    }

    /**
     * Get the notes that the user owns.
     */
    public function toDoLists()
    {
        return $this->hasMany( related: 'App\Models\ToDoList');
    }

    /**
     *
     */
    public function meets()
    {
        return $this->belongsToMany( related: 'App\Models\Meet',  table: 'participants');
    }
}
```

Figura 4.3: Modello *User*

- *ManageSubjects*: mette in relazione la tabella *users* con la tabella *subjects*; il metodo *$user->manageSubjects()* recupera tutti i corsi di cui l'utente è responsabile.
- *AttendSubjects*: mette in relazione la tabella *users* con la tabella *subjects*; il metodo *$user->attendSubject()* recupera tutti i corsi seguiti dall'utente.
- *Groups*: mette in relazione la tabella *users* con la tabella *groups*; il metodo *$user->groups()* recupera tutti i gruppi di cui l'utente fa parte.
- *Company*: mette in relazione la tabella *users* con la tabella *companies*; il metodo *$user->company()* recupera la company alla quale appartiene l'utente.
- *ToDoLists*: mette in relazione la tabella *users* con la tabella *toDoLists*; il metodo *$user->toDoLists()* recupera tutte le note dell'utente.
- *Meets*: mette in relazione la tabella *users* con la tabella *meets*; il metodo *$user->meets()* recupera tutte le videochiamate a cui l'utente può partecipare.

In EduCode i modelli, che rappresentano le entità, vengono salvati in *app/models/*.

Una volta creati lo schema del database ed i modelli, è utile popolare il database con i dati di prova; allo scopo *Laravel* mette a disposizione i *seeder* e le *factory*, localizzati nelle rispettive cartelle *database/seeder* e *database/factory*. Per default viene definita una classe *DatabaseSeeder*. Da questa classe, invocando il metodo *call*, vengono richiamate, nell'ordine stabilito, le altre

```php
class ToDoListSeeder extends Seeder
{
    /**
     * Run the database seeders.
     *
     * @return void
     */
    public function run()
    {
        TodoList::factory( ...parameters: 100)->create()->each(function ($todo) {
            if(rand(1,100) % 2 == 0 )
                $todo->deleted_at = now();
            $todo->save();
        });
    }
}
```

Figura 4.4: Definizione di un *seeder*

classi *seeder*. Una classe *seeder* contiene solo il metodo *run*, che permette di eseguire i comandi per l'inserimento dei dati. Scrivere manualmente le query per l'inserimento di tutti i dati richiede molto tempo; i *model factory* permettono di ovviare a questo problema. Le classi factory forniscono, infatti, il metodo *definition* che restituisce un array associativo che ha come chiave il nome dell'attributo ed ha associato il valore corrispondente. Attraverso la proprietà *faker*, le *factory* hanno accesso alla libreria *Faker PHP*, e, quindi, si possono generare vari tipi di dati casuali. Di seguito vengono presentati i codici sorgente delle classi *seeder* e *factory* per la creazione delle note (Figure 4.4 e 4.5).

Il codice della classe *factory* serve per creare delle note casuali; all'interno del metodo *definition* viene usata la libreria *faker*, per generare una stringa compresa tra 5 e 30 caratteri per il contenuto della nota; viene, inoltre, selezionato randomicamente l'id dell'utente a cui appartiene la nota. L'utente scelto ha il ruolo diverso da *basic* perché gli utenti basic non possono salvare note. Una volta definita la *factory* può essere usato il metodo statico *factory* per creare un'istanza di *factory* per il modello di riferimento. All'interno del metodo *run* della classe *ToDoListSeeder* viene richiamato il metodo *factory* della *ToDoListFactory*, specificando di creare 100 elementi. Gli elementi creati vengono iterati con un ciclo *foreach* e randomicamente vengono selezionate delle note da impostare come eseguite.

4.3 Organizzazione dei controller

I *controller* racchiudono la logica applicativa di EduCode, la gestione degli eventi e mettono in comunicazione la *view* e il model. I *controller* gestiscono le operazioni *CRUD*[6] e le funzionalità avanzate riguardanti le risorse coinvolte. Sono memorizzati nella cartella *app/Http/Controllers*, organizzati per funzionalità: *FeedbackController*, *GroupController*, *HomeController*, *LanguageController*, *LiveCodingController*, *MeetController*, *ProjectController*, *ScriptController*, *SubjectController*, *TodoController*, *UserController*, *VersionController*.

I metodi dei *controller* vengono invocati dalle rotte associate. Una rotta viene definita dalla chiamata *HTTP (GET, POST, PUT, PATCH, DELETE)*, dall'*uri*, dal nome del metodo da eseguire e dal nome scelto per la rotta. A titolo esemplificativo viene riportata la definizione della

[6] Acronimo di *Create, Read, Update e Delete*: rappresentano le funzioni di base da eseguire su una risorsa.

```php
class ToDoListFactory extends Factory{
    /**
     * The name of the factory's corresponding model.
     *
     * @var string
     */
    protected $model = ToDoList::class;

    /**
     * Define the model's default state.
     *
     * @return array
     */
    public function definition()
    {
        return [
            'content' => $this->faker->text(rand(5,30)),
            'user_id' => User::where('role','<>','basic')->inRandomOrder()->first()->id
        ];
    }
}
```

Figura 4.5: Definizione di una *factory*

rotta per la funzionalità di *livecoding*: *Route::get('/livecode', 'LiveCodingController@liveCode'->name('livecode');*

Le rotte possono essere organizzate in gruppi per condividere gli attributi; in EduCode vengono raggruppate per favorire l'applicazione dei middleware che offrono un meccanismo di ispezione per le richieste *HTTP*.

Accanto al *middleware* personalizzato *multilingue*, esaminato nel paragrafo successivo, Educode utilizza i *middleware* predefiniti di *Laravel*; in particolare, per l'autenticazione il *middleware* verifica l'utente e permette alla richiesta di proseguire, se l'utente è autenticato, o reindirizza alla schermata di login, negli altri casi.

Le rotte definite sono raccolte nel file *routes/web.php*; i *middleware* vengono localizzati nella cartella *app/Http/Middleware*.

A titolo esemplificativo della struttura di un controller, la Figura 4.6, presenta il *controller* per la gestione dei progetti. Il metodo *costruttore* invoca il *middleware* per la verifica dell'account; se l'utente ha verificato l'e-mail può proseguire nelle sue operazioni, altrimenti viene bloccato. I metodi specifici del *controller* sono:

- *index*: mostra l'elenco dei progetti dell'utente;
- *create*: mostra il form per la creazione del progetto;
- *store*: crea il progetto;
- *show*: mostra la pagina dettagli del progetto;
- *edit*: mostra la pagina di modifica dei dati del progetto;
- *update*: salva le modifiche effettuate nel progetto;
- *destroy*: elimina definitivamente il progetto per l'utente e lo salva nello storico di EduCode;

- *work*: visualizza la pagina per la *sessione di coding*.

4.4 Gestione delle policy e validazione dei dati

Oltre a fornire servizi di autenticazione integrati, *Laravel* fornisce anche due metodi per gestire le autorizzazioni delle risorse: *gates* e *policy*. I *gates* forniscono un approccio basato sulle *closure*[7] mentre le *policies* sono classi che organizzano la logica di autorizzazione intorno a un particolare modello o risorsa; l'uso delle due tecniche non è esclusivo. EduCode, per le sue caratteristiche progettuali, fa un largo uso di *policies*.

Una volta creata la *policy*, è necessario registrarla in *app/Providers/AuthServiceProvider.php* per indicare a *Laravel* quale *policy* utilizzare per autorizzare le azioni di un determinato modello (Figura 4.7).

Dopo la registrazione, vengono definiti i metodi specifici della *policy*. A titolo esemplificativo viene analizzata la *policy* per la gestione di un corso (Figura 4.8).

I metodi specifici della *policy* per la gestione del corso sono: *viewAny, all, view, create, edit, update, forceDelete*.

Il metodo *viewAny* è legato all'azione di visualizzazione delle materie gestite da un professore. Esso riceve come parametro l'oggetto dell'utente che effettua la richiesta e ne verifica il ruolo, restituendo un valore booleano; tale valore è pari a *true* per la richiesta soddisfatta, il che avviene se l'utente (*professor* o un *professor coordinator*) è abilitato alla gestione dei propri corsi; esso a pari a *false* in caso contrario. Il valore di ritorno del metodo viene esaminato e gestito dal *controller*.

Il metodo *all* determina chi può visualizzare tutte le materie di EduCode; questa operazione deve essere permessa solo all'amministratore del sistema.

Il metodo *companySubjects* verifica che il ruolo dell'utente sia *professor coordinator* e, in caso affermativo, permette di visualizzare tutti i corsi di una *company*.

Il metodo *view* riceve due parametri: l'utente che fa la richiesta e la materia interessata dalla richiesta. Il compito del metodo è verificare se l'utente ha diritto a visualizzare i dettagli della materia; la risposta affermativa si produce se l'utente, con account diverso da *basic*, e il corso appartengono alla stessa company, oppure se l'utente è l'amministratore del sistema.

La creazione, l'aggiornamento e l'eliminazione di un corso sono operazioni esclusive del *professor coordinator*, se appartiene alla stessa company del corso, e dell'*amministratore* del sistema.

Una volta definita, la policy viene utilizzata nel *controller* con il metodo *authorize*; quest'ultimo accetta il nome dell'azione da autorizzare e, se necessaria, un'istanza del relativo modello o il nome della classe. Se l'azione non è autorizzata, il metodo *authorize* lancia un'eccezione *Illuminate\Auth\Access\AuthorizationException* che il gestore delle eccezioni di Laravel converte automaticamente in una risposta *HTTP* con un codice di stato *403*. Le policies possono essere usate anche nelle viste, come mostrato nella sottosezione dedicata alla gestione delle viste.

Le *policies* di EduCode vengono salvate nella cartella *app/Policies*.

4.5 Gestione delle viste

Le viste contengono la logica di presentazione dell'applicazione. Il *template engine Blade*, incluso in *Laravel*, permette di semplificare la rappresentazione dei dati, iniettati dal controller alla vista, e non penalizza il codice *PHP*. Tutti i template *Blade* sono, infatti, compilati in

[7] Funzioni anonime (chiamate anche lambda) restituiscono oggetti della classe Closure.

```php
class ProjectController extends Controller
{
    public function __construct()
    {
        $this->middleware( middleware: 'verified');
    }

    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        $this->authorize( ability: 'viewAny', arguments: Project::class );
        $projects = Auth::user()->projects()->paginate(10);

        // TODO temporary, to change
        if(Auth::user()->role == 'student') {
            $groups = Auth::user()->groups;
            $groups_projects = array();
            foreach($groups as $group)
                foreach($group->projects as $project)
                    array_push( &array: $groups_projects, $project);
        } else $groups_projects = null;

        return view( view: 'projects.index',compact( var_names: 'projects', var_names: 'groups_projects'));
    }


    /**
     * Show the form for creating a new resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function create()
    {
        $this->authorize( ability: 'create', arguments: Project::class);

        $groups = Auth::user()->groups()->pluck('name', 'id');
        $groups['null'] = 'Personal projects (No subject or group)';
        return view( view: 'projects.create', compact( var_names: 'groups'));
    }
    /**
     * Store a newly created resource in storage.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function store(Request $request)
    {
        $this->authorize( ability: 'create', arguments: Project::class);

        $this->validate($request, [
            'name' => 'required',
            ]);

        $input = $request->all();
        $input['manager'] = Auth::user()->id;

        if(!isset($input['group_id']) || $input['group_id'] == 'null')
            $input['group_id'] = null;

        $project = Project::create(
            [
                'name' => $input['name'],
                'description' => $input['description'],
                'manager' => Auth::user()->id,
                'language' => $input['language'],
                'group_id' => $input['group_id']

            ]
        );

        if($project->save()) {
            $message = 'Project created successfully';
            $type = 'success';
            if($project->group_id == null)
                $project->users()->attach(Auth::user()->id);
        } else {
            $message = 'Error in project creation';
            $type = 'error';
        }

        return redirect()->route( route: 'projects.index')
            ->with($type, $message);
    }
```

```php
    /**
     * Display the specified resource.
     *
     * @param  Project $project
     * @return \Illuminate\Http\Response
     */
    public function show(Project $project)
    {
        if(Auth::user()->can( abilities: 'view', $project)){
            return view( view: 'projects.show',compact( var_names: 'project'));
        } else
            return redirect()->route( route: 'home')
                ->with('error', 'You do not have permission to perform this operation');

    }

    /**
     * Show the form for editing the specified resource.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function edit(Project $project)
    {
        $this->authorize( ability: 'update', $project);
        return view( view: 'projects.create',compact( var_names: 'project'));
    }
     /**
      * Update the specified resource in storage.
      *
      * @param  \Illuminate\Http\Request  $request
      * @param  int  $id
      * @return \Illuminate\Http\Response
      */
     public function update(Request $request, Project $project)
     {
         $this->authorize( ability: 'update', $project);

         $this->validate($request, [
                 'name' => 'required',
             ]
         );

         $input = $request->all();
         $res = $project->update($input);

         if($res){
             $message = 'Project updated';
             $type = 'success';
         } else {
             $message = 'Error in project updating';
             $type = 'error';
         }

         return redirect()->route( route: 'projects.index')
         ->with($type, $message);
     }
    /**
     * Remove the specified resource from storage.
     *
     * @param  int  $id
     * @return \Illuminate\Http\Response
     */
    public function destroy(Project $project)
    {
        $this->authorize( ability: 'forceDelete', $project);
        $project->users()->detach();
        $scripts = Script::where('project_id', $project->id)->get();
        foreach($scripts as $script){
            $script->versions()->forceDelete();
        }
        Script::where('project_id', $project->id)->delete();
        $res = $project->forceDelete();
        if($res){
            $message = 'Project deleted';
            $type = 'success';
        } else {
            $message = 'Error in project deleting';
            $type = 'error';
        }

        return redirect()->route( route: 'projects.index')
            ->with($type, $message);
    }

    /**
     * @param Project $project
     */
    public function work(Project $project){
        $this->authorize( ability: 'work', $project);
        $todolists = Auth::user()->toDoLists()->withTrashed()->paginate(10);
        return view( view: 'projects.workNew', compact( var_names: 'project', var_names: 'todolists'));
    }
}
```

Figura 4.6: Controller per la gestione dei progetti

```php
/**
 * The policy mappings for the application.
 *
 * @var array
 */
protected $policies = [
    Feedback::class => FeedbackPolicy::class,
    Project::class => ProjectPolicy::class,
    User::class => UserPolicy::class,
    ToDoList::class => ToDoListPolicy::class,
    Subject::class => SubjectPolicy::class,
    Meet::class => MeetPolicy::class
];
```

Figura 4.7: Registrazione delle Policy

PHP e memorizzati nella cache finché non vengono modificati, per cui *Blade* non aggiunge *overhead* all'applicazione. I dati possono essere passati alla vista usando il secondo argomento dell'*helper view*: *view('subject.index', compact('subjects'))*. Le viste possono essere restituite dalle *rotte* o dai *controller*.

EduCode, per le sue caratteristiche progettuali, utilizza lo stesso layout di base per tutte le pagine. Con *Blade*, infatti, è possibile definire un template ed ereditarlo nelle viste specifiche tramite la direttiva *@extends*.

Infine, nelle viste è possibile utilizzare la visualizzazione condizionale, sfruttando le *policies*, con la direttiva: *@can('viewAny', \App\Models\User::class)*. I file dei template di *Blade* usano l'estensione *.blade.php*. Le viste, organizzate per funzionalità, sono memorizzate nella cartella *resources/views* in maniera gerarchica (Figura 4.9).

A titolo esemplificativo viene presentata la vista per la visualizzazione del profilo dell'utente (Figura 4.10).

4.6 Gestione del multilingue

Le lingue attualmente supportate da EduCode sono inglese, italiano e spagnolo. La gestione di più lingue è una funzionalità rilevante per la diffusione e l'internazionalizzazione di un'applicazione. Il multilingue in EduCode viene gestito dal *controller LanguageController.php* e dal *middleware LanguageManager.php* (Figure 4.11 e 4.12).

Nella fase di avvio dell'applicazione, il *middleware LanguageManager* esamina le impostazioni dell'utente verificando se una lingua è specificata come predefinita; nel caso di assenza, recupera la lingua impostata dal browser e la imposta come predefinita per l'applicazione.

Il *controller LanguageController* ha il metodo specifico per gestire il cambio della lingua. Al momento della richiesta, viene chiamato il metodo *langChange* che imposta come lingua predefinita quella scelta dall'utente e reindirizza alla pagina che ha inviato la richiesta.

4.7 Esecuzione del codice

La funzionalità di esecuzione del codice rappresenta il cuore di EduCode e la sua attuazione richiede l'interazione di diverse tecnologie. Ad esempio, nella pagina che gestisce la *sessione di coding* interagiscono editor di testo e terminale, implementati, appunto, con differenti tecnologie. L'editor è sviluppato a partire da *Code Mirror* e configurato per supportare

```php
class SubjectPolicy
{
    use HandlesAuthorization;

    /**
     * Determine whether the user can view models.
     *
     * @param  \App\Models\User  $user
     * @return mixed
     */
    public function viewAny(User $user)
    {
        return  $user->role === 'professor coordinator' || $user->role === 'professor';
    }

    /**
     * Determine whether the user can view any models.
     *
     * @param  \App\Models\User  $user
     * @return mixed
     */
    public function all(User $user)
    {
        return $user->role === 'admin';
    }

    /**
     * Determine whether the user can view models.
     *
     * @param  \App\Models\User  $user
     * @return mixed
     */
    public function companySubjects(User $user)
    {
        return  $user->role === 'professor coordinator';
    }

    /**
     * Determine whether the user can view the model.
     *
     * @param  \App\Models\User  $user
     * @param  \App\Models\Subject  $subject
     * @return mixed
     */
    public function view(User $user, Subject $subject)
    {
        return ($user->role != 'basic' && $user->company == $subject->user->company) || $user->role=='admin';
    }
    /**
     * Determine whether the user can create models.
     *
     * @param  \App\Models\User  $user
     * @return mixed
     */
    public function create(User $user)
    {
        return  $user->role === 'professor coordinator';
    }

    /**
     * Determine whether the user can update the model.
     *
     * @param  \App\Models\User  $user
     * @param  \App\Models\Subject  $subject
     * @return mixed
     */
    public function update(User $user, Subject $subject)
    {
        return $user->id == $subject->user_id || ($user->role == "professor coordinator"
                && $subject->user->company == $user->company) || $user->role=='admin';

    }

    /**
     * Determine whether the user can permanently delete the model.
     *
     * @param  \App\Models\User  $user
     * @param  \App\Models\Subject  $subject
     * @return mixed
     */
    public function forceDelete(User $user, Subject $subject)
    {
        return  $user->id == $subject->user_id || ($user->role == "professor coordinator"
                && $subject->user->company == $user->company) || $user->role == 'admin';
    }
}
```

Figura 4.8: Policy per la gestione di un corso

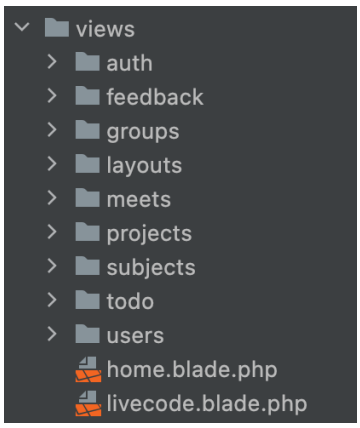
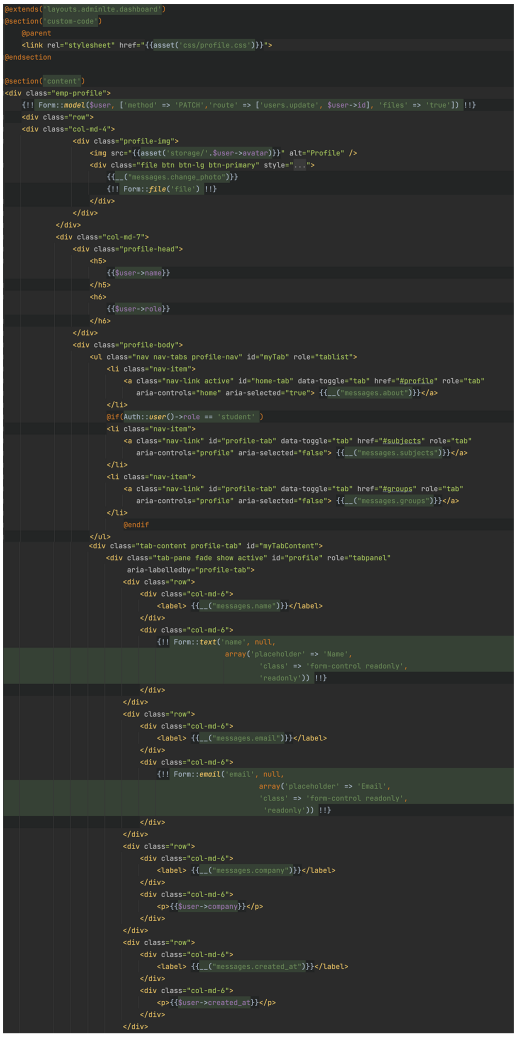


Figura 4.9: Struttura delle cartelle contenti le viste



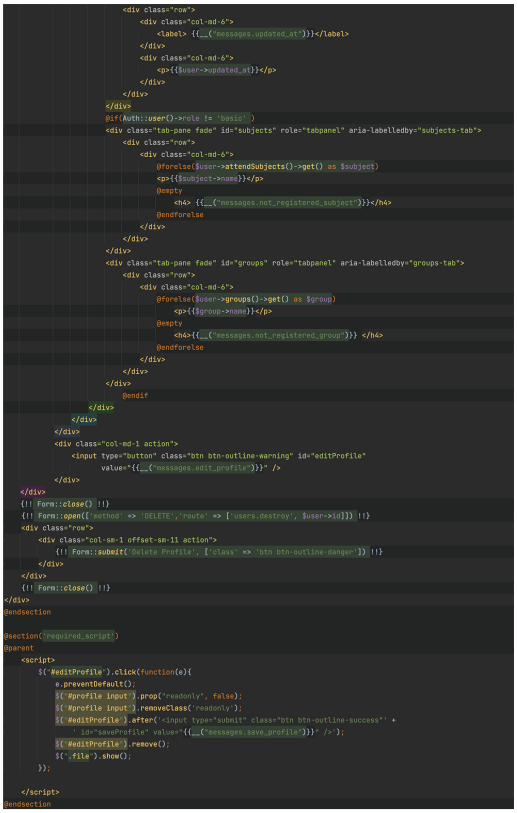


Figura 4.10: Vista del profilo dell'utente

```php
class LanguageManager
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle(Request $request, Closure $next)
    {
        if (session()->has('locale')) {
            App::setLocale(session()->get('locale'));
        }
        return $next($request);
    }
}
```

Figura 4.11: Middleware per la gestione del multilingue

```php
class LanguageController extends Controller
{
    /**
     * Change language used
     *
     * @return string
     */
    public function langChange(Request $request)
    {
        App::setLocale($request->lang);
        session()->put('locale',$request->lang);
        return redirect()->back();
    }
}
```

Figura 4.12: Controller per la gestione del multilingue

```php
use Ratchet\Server\IoServer;
use Ratchet\Http\HttpServer;
use Ratchet\WebSocket\WsServer;
use WebSocketsSsh\WebSocketsController;

require __DIR__ . '/vendor/autoload.php';

require('config.php');

$server = IoServer::factory(
            new HttpServer(
                new WsServer(
                    new WebSocketsController($configs)
                )
            ),
            8090
        );

$server->run();
```

Figura 4.13: Server WebSocket

la sintassi dei linguaggi implementati in EduCode. Nell'area dell'editor sono presenti i pulsanti per le funzioni *undo* e *redo*, per il salvataggio del codice e per la sua esecuzione. Inoltre, è possibile ridurre l'editor a icona o visualizzarlo su schermo intero, per rendere più facile la scrittura del codice sorgente. Il terminale è, invece, sviluppato a partire da xTerm e personalizzato per supportare la modalità responsive, tramite l'addon *xterm-addon-fit* e i *websocket*, per la gestione dell'input/output su di esso.

Al momento dell'avvio della *sessione di coding*, il terminale prova a stabilire la connessione *websocket*, necessaria per l'esecuzione degli script. All'apertura della connessione *websocket* si stabilisce la connessione *ssh* con il server di esecuzione, si verificano i parametri di esecuzione e si assegna un identificatore al cliente. Da questo momento, il terminale mostra lo stesso contenuto del terminale del server di esecuzione. In caso di problemi nello stabilire la connessione viene sollevata un'eccezione e viene riportato l'errore nel frontend. I parametri per l'apertura della connessione sono memorizzati nelle variabili d'ambiente e vengono salvati nel file *.env*. A connessione stabilita, EduCode può scambiare messaggi con il server di esecuzione. Ad ogni esecuzione viene inviato un messaggio; al momento della sua ricezione viene estratto dai dati il linguaggio in uso e viene generato un nome casuale. Lo script "nominato" viene mandato in esecuzione ed il suo risultato, inviato tramite *WebSocket*, e viene mostrato in tempo reale nel terminale dell'applicazione frontend. Al termine della sessione di lavoro le connessioni *ssh* e *WebSocket* vengono rilasciate e l'identificatore del cliente viene eliminato.

Il codice alla base della funzionalità descritta è composto da due parti: il *WebSocketServer* ed il *WebSocketController* (Figure 4.13, 4.14 e 4.15). Il file *WebSocketServer.php*, tramite la libreria *Ratchet*[8], esegue un server *WebSocket* sulla porta definita ed istanzia il *controller* che deve gestire i messaggi. Il *WebSocketController.php* definisce i metodi per la gestione della connessione e dei messaggi.

[8] *Ratchet* è una libreria PHP che fornisce agli sviluppatori gli strumenti per creare applicazioni bidirezionali in tempo reale tra client e server su WebSockets.

```php
<?php
namespace WebSocketsSsh;
use Ratchet\MessageComponentInterface;
use Ratchet\ConnectionInterface;


class WebSocketsController implements MessageComponentInterface {

    protected $clients;
    protected $connections;
    protected $shell;
    protected $configs;

    /**
     * Initialises the data required to manage the websockets connection.
     */
    public function __construct(array $configs) {
        $this->clients = new \SplObjectStorage;
        $this->connections = [];
        $this->shell = [];
        $this->configs = $configs;
    }

    /**
     * When a new connection is opened it will be passed to this method.
     * An ssh connection is created and a buffer is opened with the ssh shell.
     *
     * @param  ConnectionInterface $conn The socket/connection that just connected to your application
     * @throws \Exception
     */
    public function onOpen(ConnectionInterface $conn) {
        $this->clients->attach($conn);
        $this->connections[$conn->resourceId] = ssh2_connect($this->configs['SSH_IP'], $this->configs['SSH_PORT']);

        if(!$this->connections[$conn->resourceId])
            $conn->send("Connection error");

        if(ssh2_auth_password($this->connections[$conn->resourceId], $this->configs['SSH_USER'], $this->configs['SSH_PASSWORD'])){
            $this->shell[$conn->resourceId]=ssh2_shell($this->connections[$conn->resourceId], 'xterm');
            sleep(1);
            while($line = fgets($this->shell[$conn->resourceId])) {
                $conn->send(mb_convert_encoding($line, "UTF-8"));
            }
        } else {
            $conn->send("Authentication error");
            $conn->close();
            $this->connections[$conn->resourceId] = null;
            $this->clients->detach($conn);
        }
    }

    /**
     * Triggered when a client sends data through the socket.
     * It receives the command from the client and forwards it via the buffer to the ssh server.
     * The response from the ssh server is then sent to the client.
     * @param  \Ratchet\ConnectionInterface $conn The socket/connection that sent the message to your application
     * @param  string $msg The message received
     * @throws \Exception
     */
    public function onMessage(ConnectionInterface $from, $msg) {
        $programFileName = $this->getFileName(10);
        $data = json_decode($msg,true);

        switch ($data['interpreter']){
            case 'CLISP':
                $programFileName.=".lisp";
                fwrite($this->shell[$from->resourceId], "echo '$data[code]' > $programFileName".PHP_EOL);
                usleep(800000);
                while($line = fgets($this->shell[$from->resourceId]));
                fwrite($this->shell[$from->resourceId], "clisp $programFileName".PHP_EOL);
                sleep(1);
                while($line = fgets($this->shell[$from->resourceId]))
                    $from->send(mb_convert_encoding($line, "UTF-8"));
                break;
            case 'SBCL':
                $programFileName.=".lisp";
                fwrite($this->shell[$from->resourceId], "echo '$data[code]' > $programFileName".PHP_EOL);
                usleep(800000);
                while($line = fgets($this->shell[$from->resourceId]));
                fwrite($this->shell[$from->resourceId], "sbcl --script $programFileName --quit".PHP_EOL);
                sleep(1);
                while($line = fgets($this->shell[$from->resourceId]))
                    $from->send(mb_convert_encoding($line, "UTF-8"));
                break;
            case 'CLOJURE':
                $programFileName.=".clj";
                fwrite($this->shell[$from->resourceId], "echo '$data[code]' > $programFileName".PHP_EOL);
                usleep(800000);
                while($line = fgets($this->shell[$from->resourceId]));
                fwrite($this->shell[$from->resourceId], "clojure $programFileName".PHP_EOL);
                sleep(1);
                while($line = fgets($this->shell[$from->resourceId]))
                    $from->send(mb_convert_encoding($line, "UTF-8"));
                break;
            case 'ASSEMBLER':
                $programName = $programFileName;
                $programFileName.=".asm";
                fwrite($this->shell[$from->resourceId], "echo '$data[code]' > $programFileName".PHP_EOL);
                usleep(800000);
                while($line = fgets($this->shell[$from->resourceId]));
                fwrite($this->shell[$from->resourceId], "nasm -f elf $programFileName && ld -m elf_i386 -s -o $programName $programName.o && ./$programName  ".PHP_EOL);
                sleep(1);
                while($line = fgets($this->shell[$from->resourceId]))
                    $from->send(mb_convert_encoding($line, "UTF-8"));
                break;
            case 'SHELL':
                fwrite($this->shell[$from->resourceId], $data['code'].PHP_EOL);
                sleep(1);
                $i = 0;
                while($line = fgets($this->shell[$from->resourceId]))
                    if($i == 0) $i++;
                    else
                    $from->send(mb_convert_encoding($line, "UTF-8"));
                break;
            case 'NASM':
                $programName = $programFileName;
                $programFileName.=".asm";
                fwrite($this->shell[$from->resourceId], "echo '$data[code]' > $programFileName".PHP_EOL);
```

Figura 4.14: Controller WebSocket (prima parte)

```
                usleep(800000);
                while($line = fgets($this->shell[$from->resourceId]));
                fwrite($this->shell[$from->resourceId], "nasm -f elf $programFileName".PHP_EOL);
                sleep(1);
                while($line = fgets($this->shell[$from->resourceId]))
                    $from->send(mb_convert_encoding($line, "UTF-8"));
                break;
            case 'LINK':
                $programName = $programFileName;
                $programFileName.=".asm";
                fwrite($this->shell[$from->resourceId], "echo '$data[code]' > $programFileName".PHP_EOL);
                usleep(800000);
                while($line = fgets($this->shell[$from->resourceId]));
                fwrite($this->shell[$from->resourceId], "nasm -f elf $programFileName && ld -m elf_i386 -s -o $programName $programName.o".PHP_EOL);
                sleep(1);
                while($line = fgets($this->shell[$from->resourceId]))
                    $from->send(mb_convert_encoding($line, "UTF-8"));
                break;
            default:
                $from->send("COMMAND NOT FOUND");
                break;

        }
    }

    /**
     * This is called before or after a socket is closed (depends on how it's closed).
     * Deletes all references to the buffer and the ssh connection, then closes the websockets connection
     * @param  ConnectionInterface $conn The socket/connection that is closing/closed
     * @throws \Exception
     */
    public function onClose(ConnectionInterface $conn) {
        $this->clients->detach($conn);
        $this->connections[$conn->resourceId] = null;
        $this->shell[$conn->resourceId] = null;
        $conn->close();
    }

    /**
     * If there is an error or exception in the application flow, this method is called.
     * Report the error and call the method for closing the connection
     * @param  ConnectionInterface $conn
     * @param  \Exception $e
     * @throws \Exception
     */
    public function onError(ConnectionInterface $conn, \Exception $e) {
        $conn->send("Connection error");
        $this->onClose($conn);
    }

    /**
     * Generate a random file name
     * @param int $n
     * @return String
     */
    private function  getFileName(int $n) : String {
        $characters = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
        $randomString = '';

        for ($i = 0; $i < $n; $i++) {
            $index = rand(0, strlen($characters) - 1);
            $randomString .= $characters[$index];
        }
        return $randomString;
    }
}
```

Figura 4.15: Controller WebSocket (seconda parte)

4.8 Monitoraggio

La funzione di *monitoraggio*, che è strettamente connessa all'esecuzione del codice, rende possibile supervisionare in tempo reale lo sviluppo del progetto di uno studente. Per realizzarla viene utilizzato il servizio *Pusher*, che facilita lo sviluppo di funzionalità *real time* per applicazioni web e mobile perché definisce un livello tra il client ed il server e si occupa di mantenere le connessioni persistenti tramite WebSocket. La scelta di utilizzare un servizio esterno per la funzione di monitoraggio risponde alla necessità di decentralizzare il carico computazionale, alla possibilità di scalare rapidamente in modo orizzontale, ed offre una maggiore tolleranza agli errori poiché l'applicazione continua a funzionare senza problemi nel caso di interruzioni del servizio di monitoraggio.

Il funzionamento della funzione di supervisione viene definito nella vista della *sessione di coding*, mentre la creazione dei canali e l'inoltro dei messaggi viene definito, usando le *PUSHER API*, come evento, in *app/Events/CodeMonitoring.php* (Figura 4.16). Per il lato client, ad ogni carattere scritto nell'editor, viene inviato un messaggio tramite *WebSocket*. Per il lato server, una volta creato il canale e stabilite le regole di gestione, l'inoltro dei messaggi è gestito da *Pusher*.

```php
class CodeMonitoring implements ShouldBroadcastNow
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    protected $script;
    public $code;


    /**
     * Create a new event instance.
     *
     * @return void
     */
    public function __construct($script_id, $code)
    {
        $this->script = $script_id;
        $this->code = $code;
    }

    /**
     * Get the channels the event should broadcast on.
     *
     * @return \Illuminate\Broadcasting\Channel|array
     */
    public function broadcastOn()
    {
        return new PrivateChannel( name: 'coding.'.$this->script);
    }

    public function broadcastWith()
    {
        // This must always be an array. Since it will be parsed with json_encode()
        return [
            'code' => $this->code,
        ];
    }
}
```

Figura 4.16: Creazione del canale per la comunicazione WebSocket

CAPITOLO 5

Manuale del sistema

In questo capitolo vengono presentati i manuali per gli utenti finali, ovvero studenti e professori. Il manuale dello studente illustra le funzionalità a disposizione di quest'ultimo nella sua area riservata. Poiché il professor ed il professor coordinator possono accedere a tutte le funzionalità degli studenti, il manuale del professore ingloba quello dello studente e presenta le funzioni esclusive di quest'ultimo ruolo.

5.1 Manuale dello studente

Tutte le funzionalità a disposizione dello studente sono presenti nella sua area riservata; l'unica operazione da svolgere, all'esterno dell'area riservata, è la registrazione dell'account, che avviene nella pagina pubblica del sito web.

5.1.1 Registrazione

Per effettuare la registrazione su EduCode è necessario collegarsi al corrispettivo sito web dell'applicazione e cliccare sulla voce *Registrazione* (Figura 5.1).

L'utente viene reindirizzato alla pagina contenente il form di registrazione e deve compilare tutti i campi obbligatori (Figura 5.2). Particolare importanza riveste la scelta dell'ente di appartenenza dell'utente; da essa dipendono, infatti, le funzionalità non incluse nell'account base. Inviato il form, l'utente riceve, sull'indirizzo e-mail indicato in fase di registrazione, il link per l'attivazione dell'account (Figura 5.3).

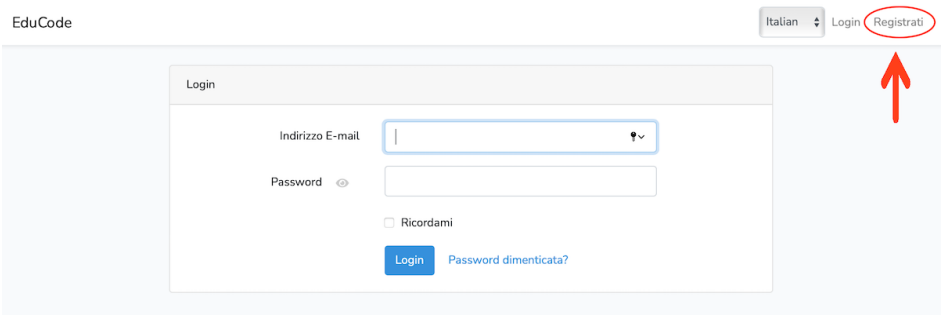


Figura 5.1: Pulsante di registrazione

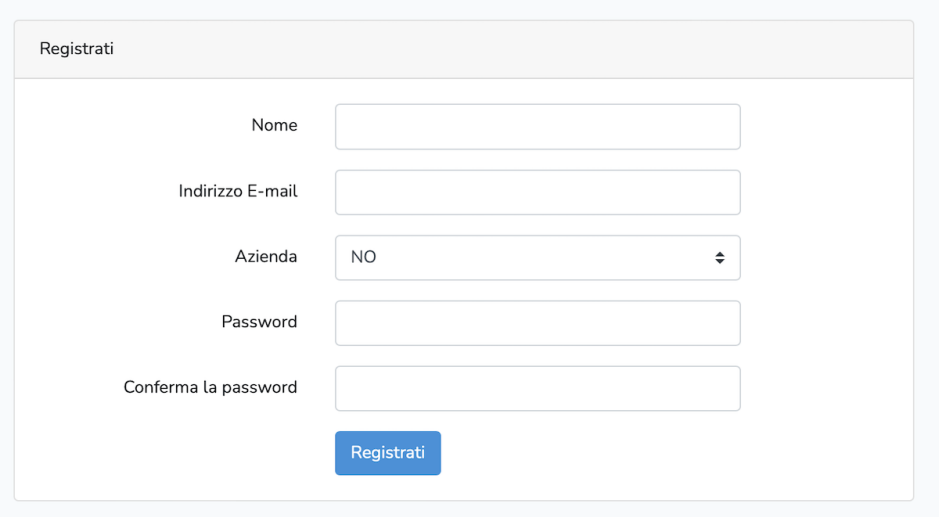


Figura 5.2: Form di registrazione

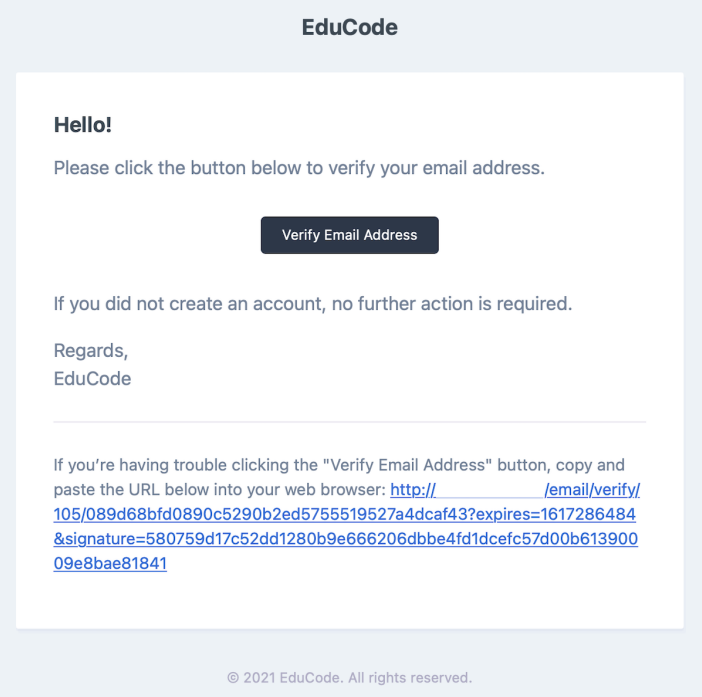


Figura 5.3: E-mail per la conferma della registrazione

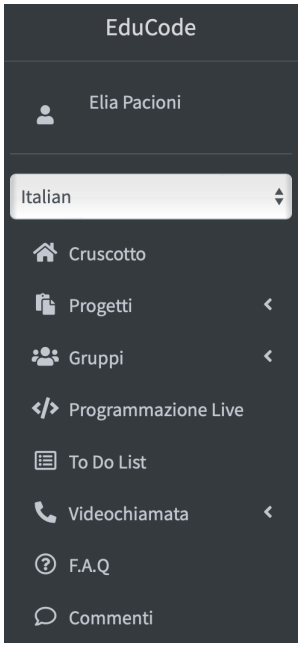


Figura 5.4: Menù left sidebar

Al momento della registrazione il ruolo dell'account è *basic*. Occorre l'abilitazione di un *professor coordinator* perché venga riconosciuto lo status di studente, con l'accesso a tutte le funzionalità collegate.

5.1.2 Area riservata

All'interno dell'area riservata, le funzionalità sono distribuite su due menù, ovvero il *top menù* posizionato in alto, e la *left sidebar* presente nella parte sinistra della pagina. Il *top menù* contiene la voce di menù per effettuare il logout. La *left sidebar* (Figura 5.4) contiene il menù principale con i collegamenti alle funzionalità. Lo studente può accedere alle pagine selezionando la voce di menù a cui è interessato.

5.1.3 Dashboard

La *dashboard* è la prima pagina visualizzata dopo il login. Essa riepiloga le informazioni delle sezioni principali, mostrando un prospetto completo della situazione dell'utente (Figura 5.5).

5.1.4 Gestione del profilo

Cliccando sul nome utente nella *left sidebar*, viene visualizzata la pagina per la gestione del profilo che mostra le informazioni dell'utente. Cliccando sul pulsante *aggiorna profilo* (Figura 5.6) è possibile modificare e-mail, nome e immagine del profilo. Per salvare le informazioni modificate è necessario fare click sul pulsante *salva profilo* (Figura 5.7). Sempre dalla stessa pagina, cliccando sulle apposite tab, lo studente visualizza i corsi seguiti e i gruppi di cui fa parte.

5.1.5 LiveCoding

La funzionalità di *LiveCoding* permette di eseguire il codice sorgente senza lasciare traccia sui server di EduCode. Nella pagina *LiveCoding* (Figura 5.8) viene visualizzato l'editor di testo,

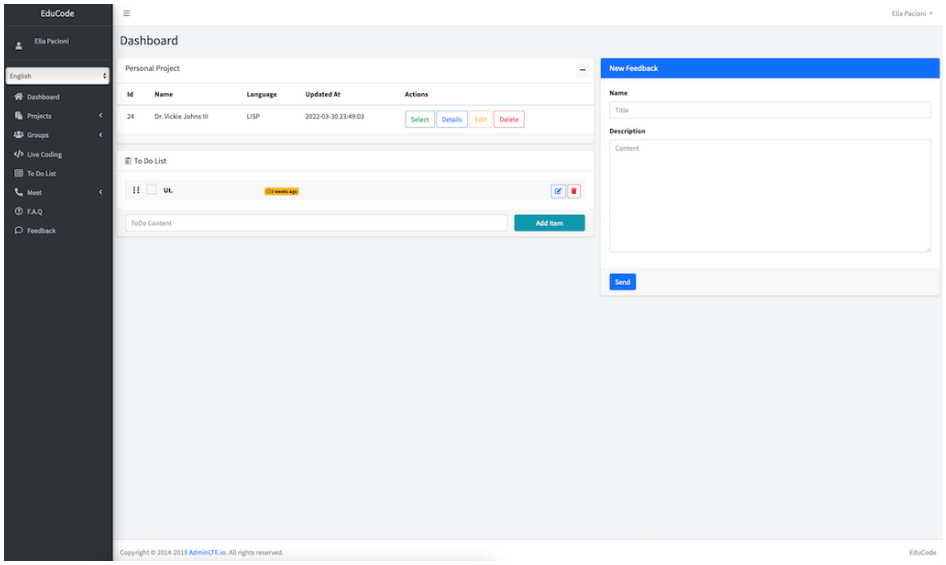


Figura 5.5: Dashboard dello studente

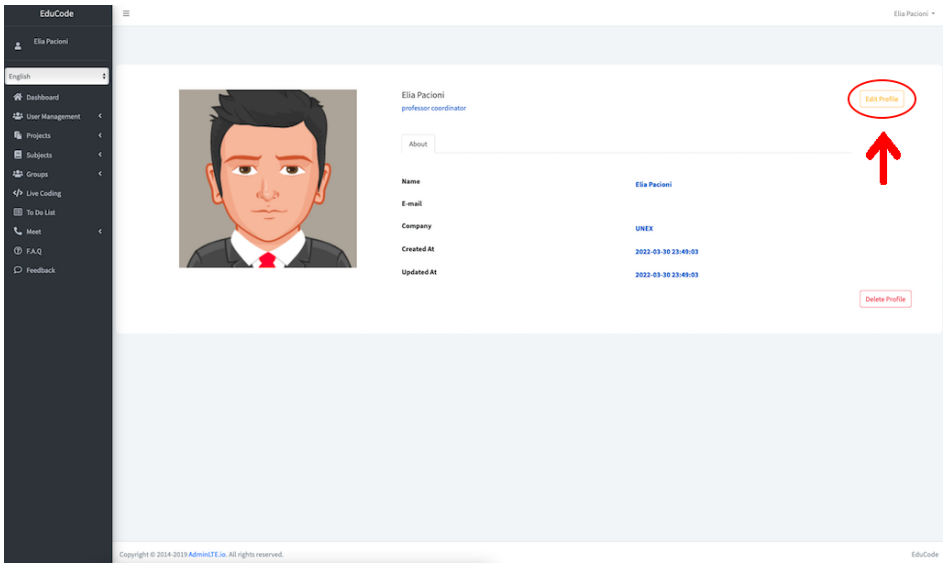


Figura 5.6: Pagina del profilo dell'utente

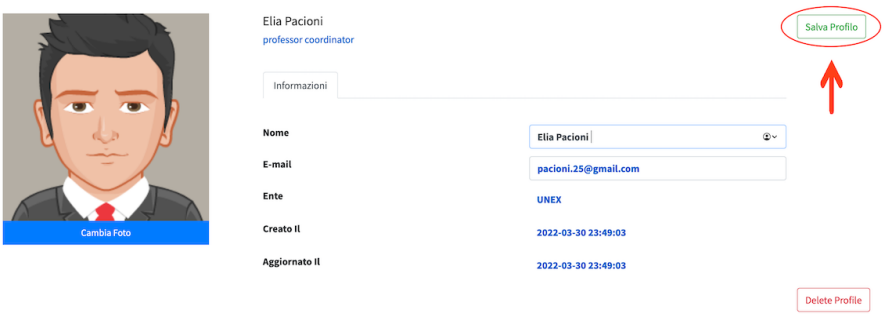


Figura 5.7: Modifica del profilo dell'utente

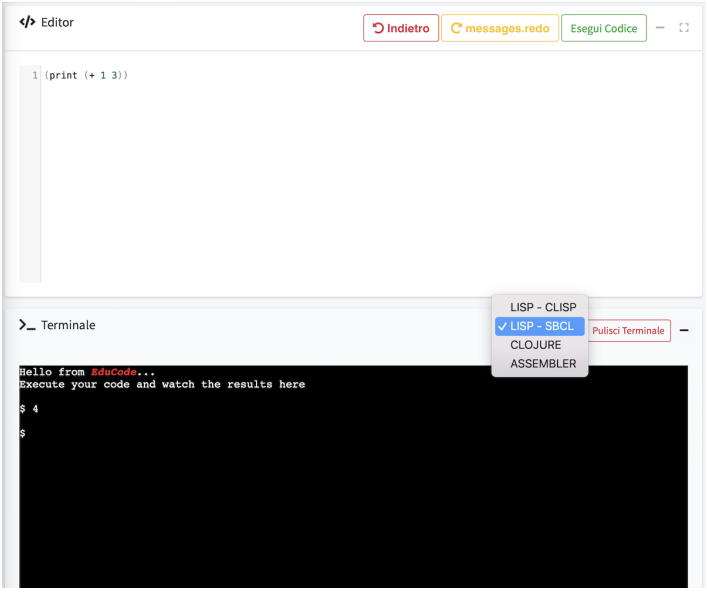


Figura 5.8: Pagina della funzionalità di *LiveCoding*

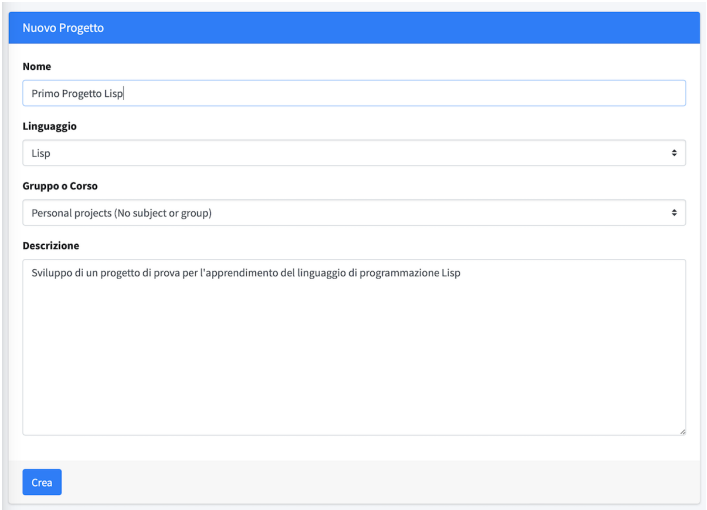


Figura 5.9: Form per la creazione di un progetto

per scrivere il proprio codice, la *listBox*, per scegliere il linguaggio di programmazione, il pulsante per l'esecuzione dello script ed il terminale per visualizzare il risultato dell'esecuzione. La funzionalità *LiveCoding* non abilita l'input nel terminale.

5.1.6 Gestione dei Progetti

Nuovo Progetto

La pagina *Nuovo progetto* consente di creare un nuovo progetto ed iniziare subito a programmare; si visualizza il form da compilare e vanno riempiti i seguenti campi (Figura 5.9):

- *Nome*: caratterizzato da una qualsiasi stringa alfanumerica.
- *Linguaggio di programmazione*: a scelta tra quelli proposti.

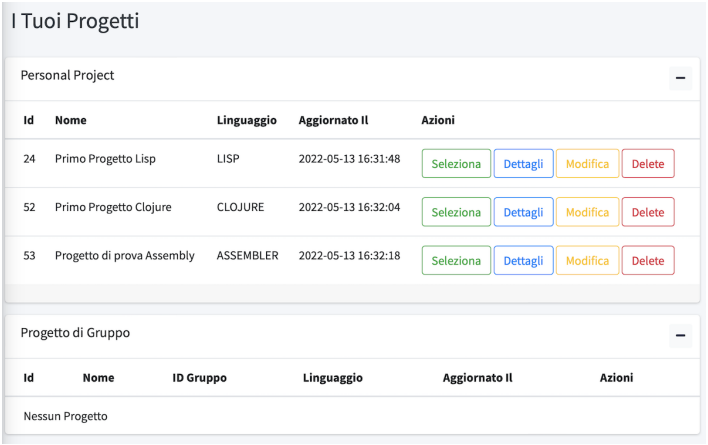


Figura 5.10: Pagina per la visualizzazione dei progetti

- *Gruppo o Corso*: a scelta tra le opzioni della *listbox*; questo campo serve a definire la tipologia di progetto.
- *Descrizione*: stringa alfanumerica che descrive il contenuto del progetto.

Con l'invio del form il progetto viene creato e l'utente viene indirizzato alla pagina di visualizzazione dei progetti.

Visualizza progetti

Questa è una pagina riassuntiva dei progetti a cui partecipa l'utente (Figura 5.10). Dalla tabella dei progetti è possibile eseguire tutte le operazioni collegate ad un progetto: visualizzazione dettagli, inizio *sessione di coding*, modifica ed eliminazione del progetto.

Sessione di programmazione

La sessione di programmazione consente di lavorare con un progetto e di eseguire il codice che si sta sviluppando. È possibile avviare una sessione dalla pagina *visualizza progetti* cliccando sull'apposito pulsante del progetto scelto. La prima volta che si lavora con un progetto, per abilitare tutte le funzionalità di esecuzione, è necessario creare almeno uno script con l'apposito pulsante *aggiungi script* (Figura 5.11); dopo di ciò si può iniziare a programmare. In base a linguaggio di programmazione scelto per il progetto vengono visualizzati pulsanti specifici; nel caso dell'*Assembler* vengono aggiunti i pulsanti *NASM* per assemblare, e *LINK* per "linkare". Il pulsante *Esegui codice* è presente per tutti i linguaggi di programmazione. I risultati dell'esecuzione e le eventuali richieste di input vengono visualizzate nel terminale (Figura 5.12).

5.1.7 Gruppi

La pagina di visualizzazione dei gruppi riassume tutti i gruppi di cui l'utente fa parte. Per ogni gruppo è possibile accedere alla pagina relativa ai dettagli per visualizzare le informazioni ed i progetti correlati (Figure 5.13 e 5.14).

5.1.8 ToDo List

La funzionalità *ToDo List* permette di memorizzare delle note, modificarle, segnarle come svolte ed eliminarle. Tutte le funzioni sono disponibili in un'unica pagina. Per la creazione di

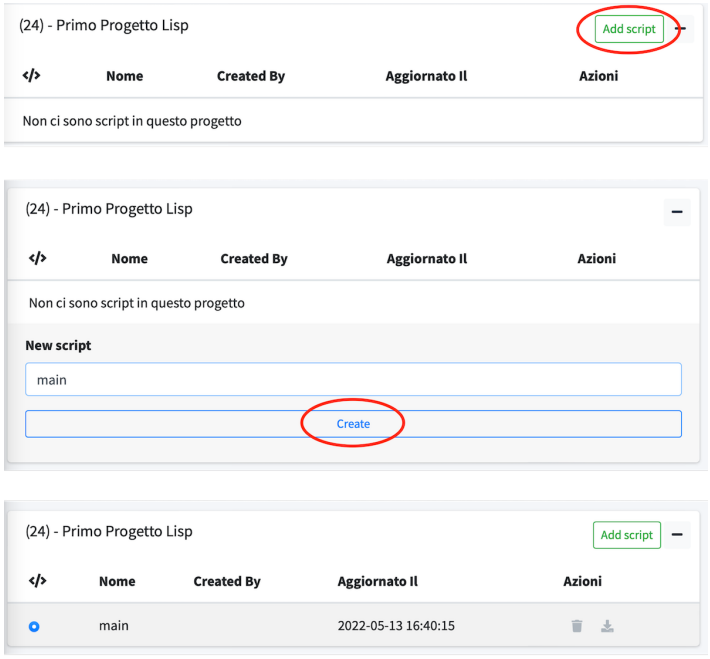


Figura 5.11: Creazione di uno script

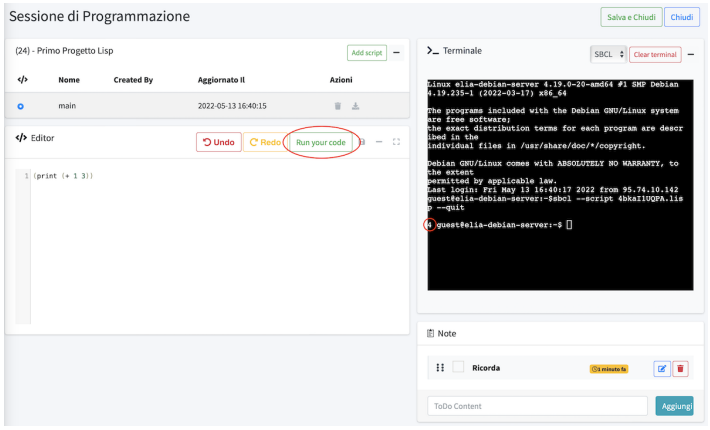


Figura 5.12: Sessione di coding

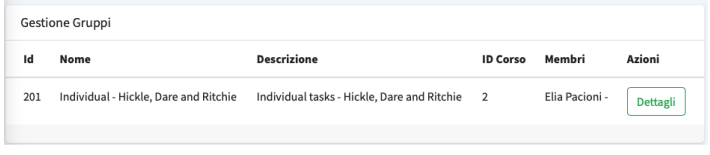


Figura 5.13: Lista dei gruppi dello studente

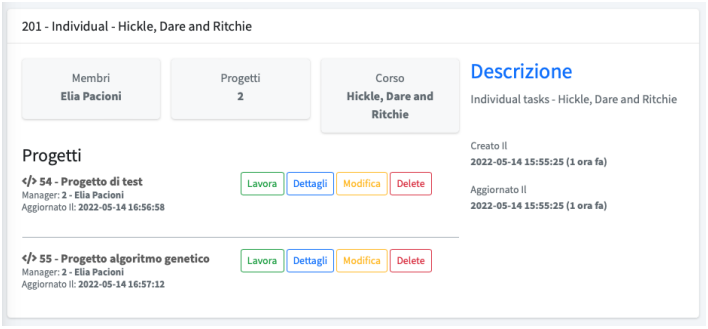


Figura 5.14: Pagina di dettaglio del gruppo

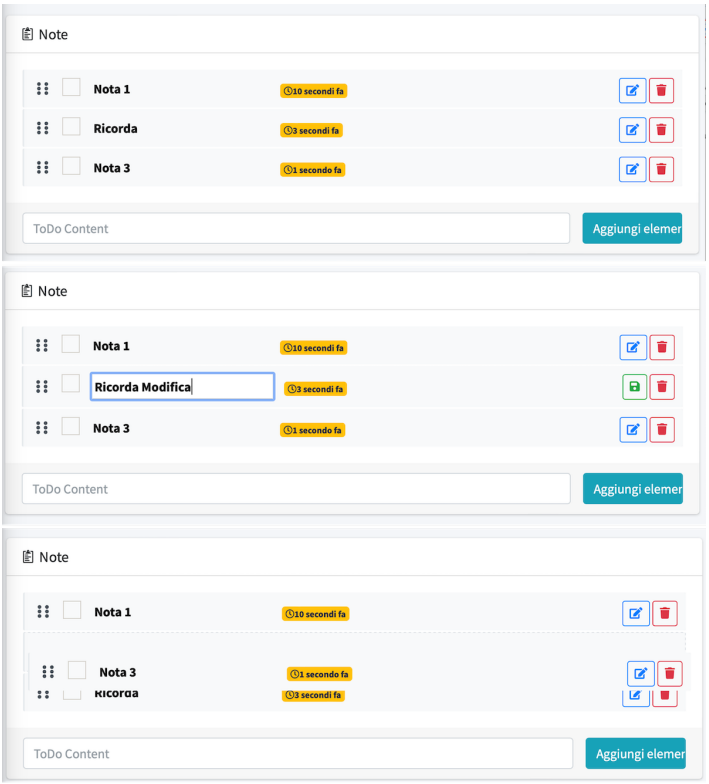


Figura 5.15: Pagina *ToDoList*

una nuova nota è presente un form in basso nella tab. Per la modifica e l'eliminazione sono presenti i tasti a destra, mentre, per contrassegnarla come svolta va utilizzata la checkbox a sinistra. Il *Drag and Drop* permette l'ordinamento a piacere delle note (Figura 5.15).

5.1.9 Feedback

La funzionalità *feedback* è di supporto allo sviluppo del progetto. Attraverso il form è possibile inviare, direttamente all'amministrazione, la segnalazione di un bug o la richiesta di una nuova funzionalità. Il form per la creazione di un feedback richiede di compilare i campi relativi al titolo e alla descrizione (Figura 5.16).

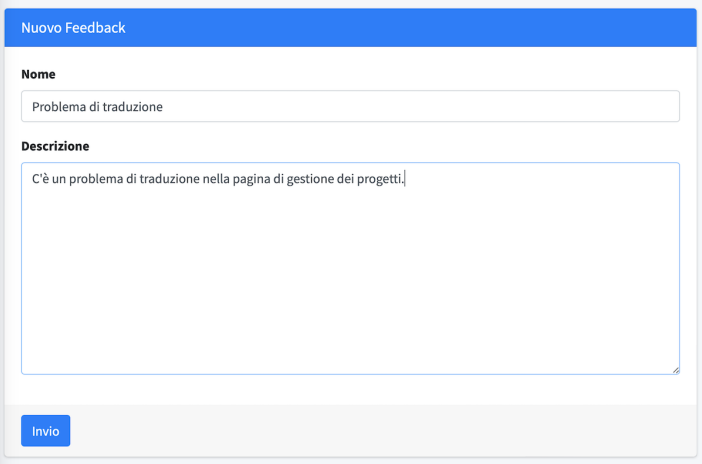


Figura 5.16: Invio di un feedback

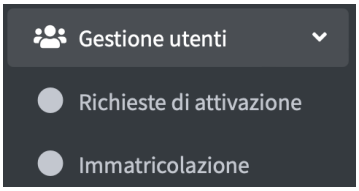


Figura 5.17: Voce di menù per la gestione degli utenti

5.2 Manuale del professore

5.2.1 Gestione degli utenti

La gestione degli utenti è una funzionalità esclusiva del *professor coordinator* (Figura 5.17).

Attivazione dell'account

Nella pagina *Attivazione account* vengono visualizzati tutti gli utenti appartenenti alla stessa *company* del *professor coordinator*. L'attivazione è necessaria per confermare lo status di studente all'utente che fino a quel momento ha un account *basic*. Per attivare l'account studente è necessario cliccare sul pulsante *attivazione* relativo al nominativo scelto (Figura 5.18).

Immatricolazione

La funzionalità d'immatricolazione permette d'iscrivere lo studente ai corsi. La pagina per l'immatricolazione mostra la lista degli studenti della *company*. Cliccando sul pulsante *immatricolazione* (Figura 5.19) viene visualizzato il form per selezionare i corsi seguiti dallo studente (Figura 5.20): a sinistra sono presenti tutti i corsi disponibili, a destra quelli seguiti dallo studente. Per aggiungere un corso è necessario selezionarlo dalla lista di sinistra e

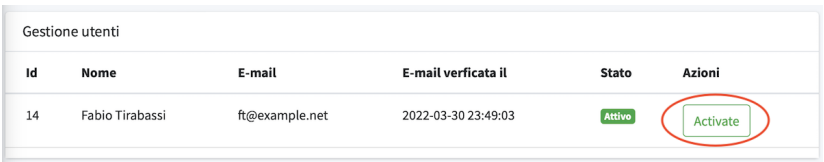


Figura 5.18: Attivazione dell'account dell'utente

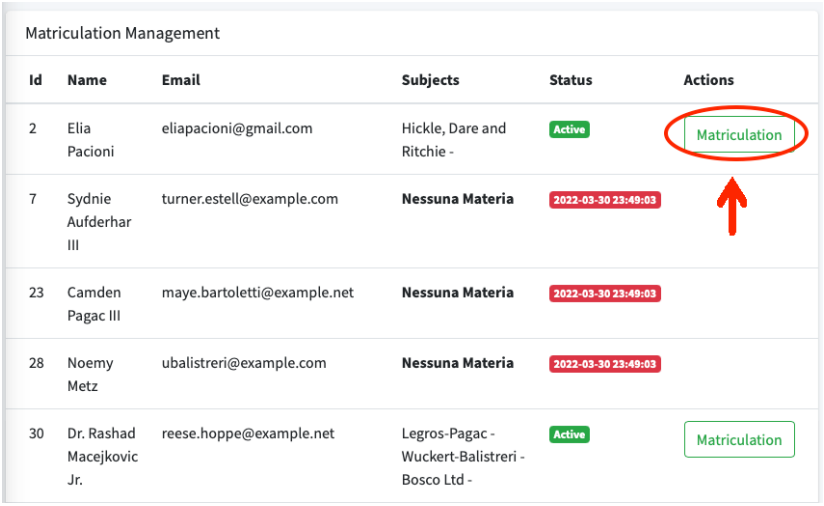


Figura 5.19: Riepilogo dei corsi seguiti dagli studenti

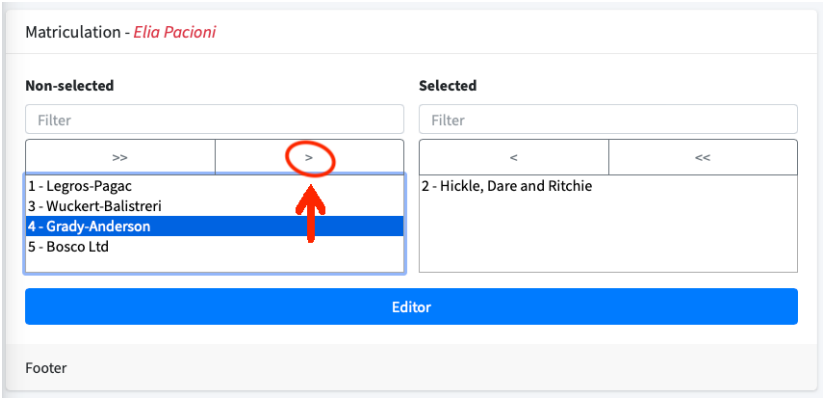


Figura 5.20: Scelta dei corsi per lo studente

premere la freccia per lo spostamento a destra; per eliminarlo si usa il procedimento inverso. Terminata l'immatricolazione, l'utente viene reindirizzato alla pagina precedente.

5.2.2 Gestione corsi

Nuovo corso

La creazione di un nuovo corso è una funzionalità esclusiva del *professor coordinator*. La pagina di creazione del corso (Figura 5.21) mostra il form contenente i campi nome, professore e descrizione. I campi nome e descrizione accettano delle stringhe alfanumeriche mentre il campo professore assegna il corso al professore selezionato nell'elenco. Il campo del form mostra i professori della stessa *company* del *professor coordinator*.

Visualizza corsi ente

La funzionalità di visualizzazione dei corsi di un ente è esclusiva del *professor coordinator*; essa permette di visualizzare i dettagli del corso (Figura 5.22) nonché di modificarlo con un form uguale a quello per la creazione del nuovo corso; è anche possibile eliminare il corso (Figura 5.23).

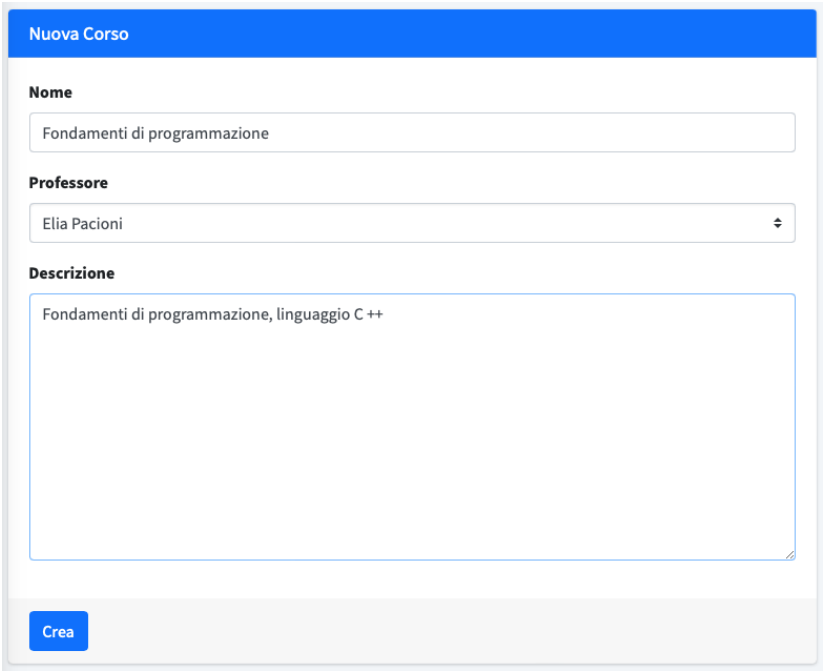


Figura 5.21: Creazione di un nuovo corso

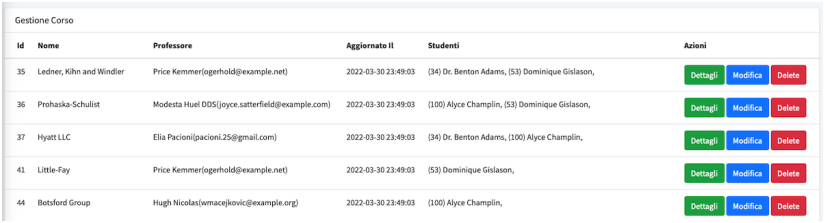


Figura 5.22: Visualizzazione dei corsi di un ente

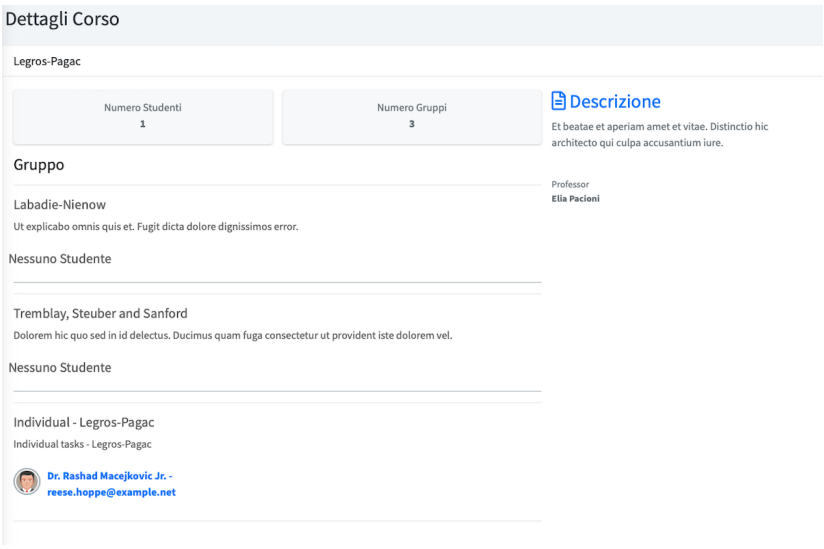


Figura 5.23: Visualizzazione della scheda dettagli di un corso

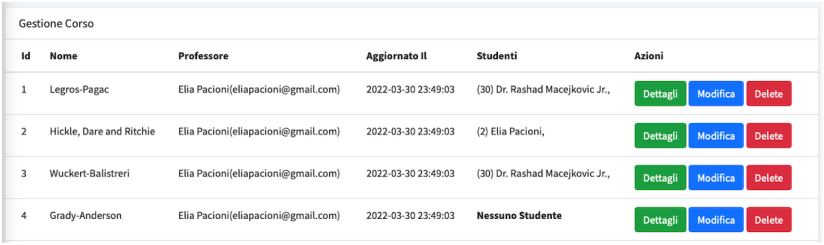


Figura 5.24: Visualizzazione dei corsi di un professore

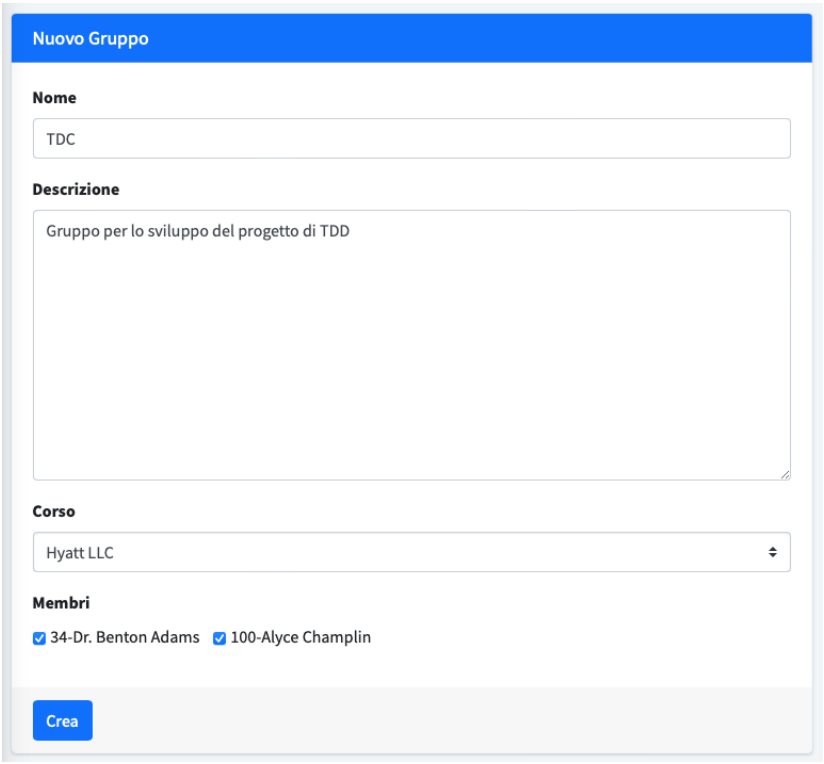


Figura 5.25: Creazione di un gruppo

Visualizza corsi

La pagina *visualizza corsi* mostra tutti i corsi gestiti dal professore (Figura 5.24); al suo interno è possibile visualizzare i dettagli del corso, modificare tali dettagli ed, infine, eliminare il corso.

5.2.3 Gestione gruppi

Nuovo Gruppo

La creazione di un gruppo viene effettuata compilando il relativo form (Figura 5.25). I dati richiesti sono il nome del gruppo, la descrizione, la materia di riferimento e gli studenti che ne fanno parte. Gli studenti appartenenti al gruppo possono essere modificati successivamente.

Dopo la creazione, gli studenti del gruppo possono creare progetti di gruppo, mentre il professore che supporta gli studenti nello sviluppo li può monitorare.

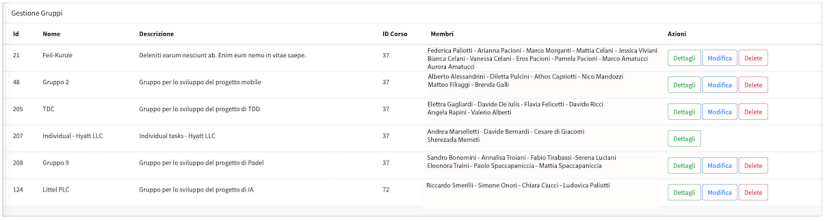


Figura 5.26: Visualizzazione dei gruppi

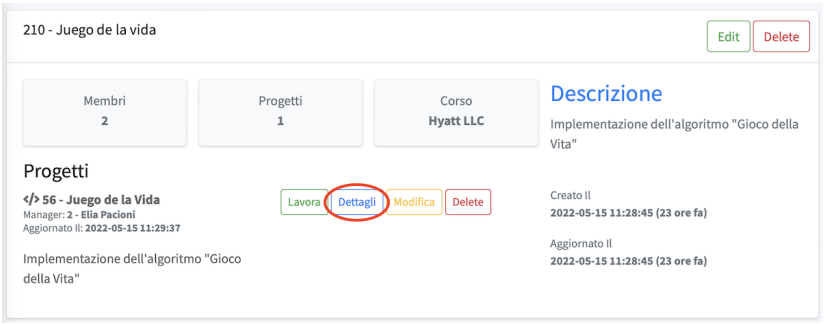


Figura 5.27: Scheda dei dettagli di un gruppo

Visualizza gruppi

Il professore visualizza tutti i gruppi delle materie che gestisce (Figura 5.26). Dalla pagina di visualizzazione egli può accedere alla pagina dei dettagli del gruppo; egli, inoltre, può modificare i componenti del gruppo oppure eliminarlo.

5.2.4 Monitoraggio

La funzionalità di monitoraggio consente al professore di supervisionare in tempo reale l'operato dei suoi studenti. È possibile utilizzare questa funzionalità solo sui progetti collegati alla materia impartita dal professore. Per utilizzare questa funzionalità è necessario accedere, dalla *left sidebar*, alla pagina *visualizza gruppi* e selezionare il gruppo interessato. Dalla pagina dei dettagli del gruppo viene visualizzato l'elenco dei progetti (Figura 5.27); una volta individuato il progetto da monitorare, cliccando sul pulsante *dettagli*, si accede alla pagina dei dettagli e, tramite essa, viene supervisionato il progetto. Il codice viene mostrato, una volta che è stato aggiornato in tempo reale, nella sezione *Code* a destra (Figura 5.28).

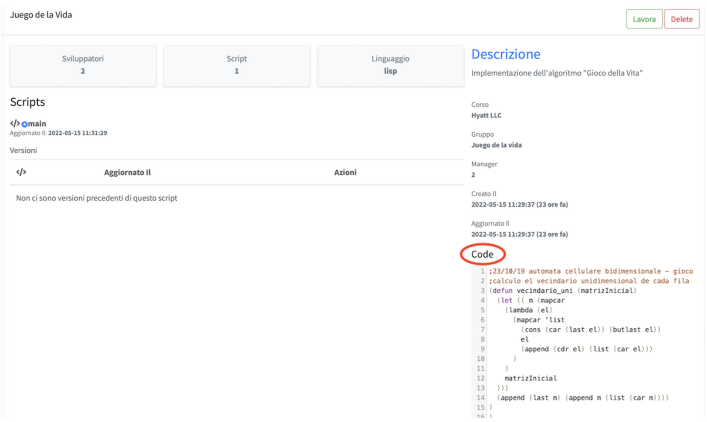


Figura 5.28: Pagina della funzionalità di monitoraggio

CAPITOLO 6

Analisi dei sistemi correlati

In questo capitolo vengono presentati alcuni sistemi correlati e vengono confrontati con EduCode.

6.1 Panoramica sui sistemi correlati

Lo sviluppo di un'applicazione e, più in generale, di un prodotto presuppone uno studio di mercato per capire la fattibilità del progetto, per definirne i punti di forza e individuarne le debolezze. Il primo passo per stabilire gli obiettivi di mercato è lo studio dei sistemi correlati.

Nel caso di EduCode, questi ultimi sono molteplici. Di seguito vengono analizzate due applicazioni tra quelle più conosciute e utilizzate: *Replit* e *Codeanywhere*.

6.1.1 Replit

Il nome *Replit*, precedentemente *Repl.it*, deriva dall'acronimo *REPL* che sta per *"Read-Evaluate-Print Loop"*. *Replit* è sviluppato dall'omonima start-up con sede a San Francisco, fondata nel 2016, che ha raccolto investimenti per 80 milioni di dollari ed ha un team composto da oltre trenta persone.

Replit viene definito come un'*IDE*[1] collaborativo, con il supporto alla multiprogrammazione in tempo reale e l'integrazione del codice con *GitHub*.

L'ambiente di sviluppo è basato sull'editor *Monaco*, sviluppato da Microsoft come base di *Visual Studio Code*. L'interfaccia di sviluppo è user friendly e ricca di funzioni. Dal 2022 *Replit* sta testando in beta l'utilizzo dell'editor *CodeMirror*, lo stesso utilizzato da EduCode fin dalla progettazione.

Per accedere alle funzionalità di sviluppo è necessario creare un account; la registrazione si può fare anche tramite social. Il software prevede l'attivazione di piani diversi che differiscono per costi e funzionalità dedicate. Con il piano base si possono creare infiniti progetti pubblici, con un utilizzo massimo di memoria di 500 MB. Acquistando il piano *Hacker* si possono aumentare le performance della macchina virtuale e sbloccare la possibilità di avere progetti privati. Il piano per studenti permette di provare gratuitamente le stesse funzionalità, aumentando di 4 volte le prestazioni della macchina virtuale base.

Per iniziare a programmare è necessario creare un progetto e scegliere il linguaggio di programmazione, il nome e la visibilità. Una volta creato il progetto, si avvia la macchina virtuale nella quale viene eseguito il programma sviluppato dall'utente (Figura 6.1).

[1] Ambiente di sviluppo integrato, è un software che supporta i programmatori nelle fasi di codifica e debug.

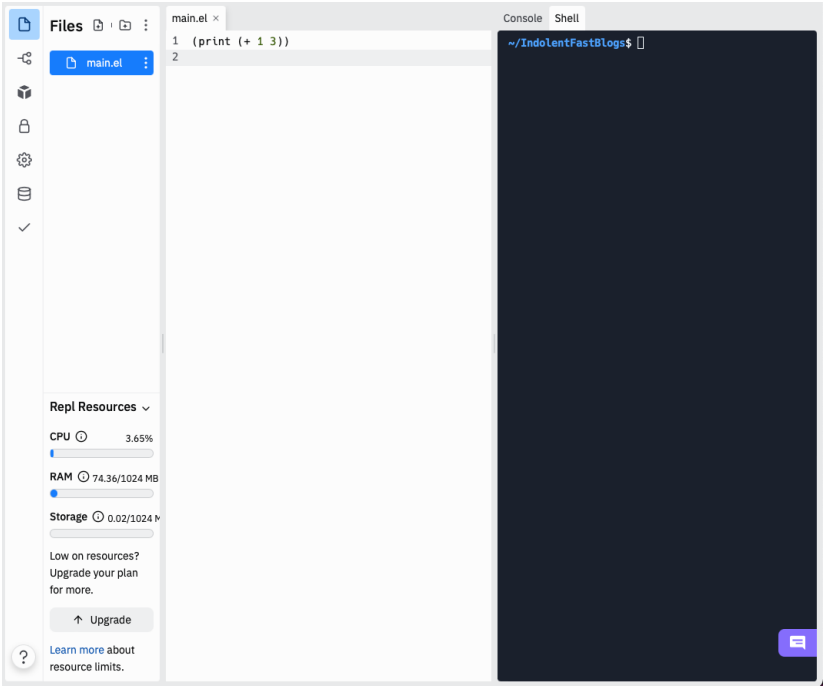


Figura 6.1: Programmazione con *Replit*

6.1.2 Codeanywhere

Codeanywhere è un IDE cloud multipiattaforma che possiede tutte le caratteristiche di un IDE desktop; esso supporta più di 120 linguaggi di programmazione, implementa la funzionalità di autocompletamento del codice e di formattazione per migliorare la leggibilità.

Si integra con GitHub e Bitbucket, per la condivisione dei file sorgenti, e supporta l'esecuzione dei progetti su server proprietari o dell'azienda.

Codeanywhere rappresenta una soluzione molto performante e mette a disposizione un ambiente di sviluppo cloud completo; esso supporta lo sviluppo collaborativo ed integra le chat in tempo reale (Figura 6.2.

Il pacchetto base è gratuito ma presenta molte limitazioni; per poter sfruttare la potenza dell'IDE è necessario acquistare i pacchetti a pagamento perché *Codeanywhere* ha come riferimento il mercato aziendale.

6.2 Confronto con i sistemi correlati

Il *core business* di *Replit* e *Codeanywhere* è orientato ai servizi per le aziende con l'offerta di un IDE cloud completo e performante. EduCode è stato progettato e sviluppato pensando alle necessità dell'ambiente universitario e degli studenti, ed ha come obiettivo principale l'apprendimento della programmazione. Pur rivolgendosi a mercati differenti è utile fare un confronto tra le applicazioni citate. Nella Tabella 6.1 sono messe confronto in forma riassuntiva le caratteristiche principali.

Le soluzioni presentate consentono, come EduCode, l'esecuzione di codice su un server remoto, che sia esso personale o dell'applicazione stessa. Dall'analisi effettuata viene evidenziato come EduCode, seppur con risorse limitate, implementi funzionalità all'avanguardia, precedendo talvolta applicazioni più mature e con team di sviluppo di grandi dimensioni.

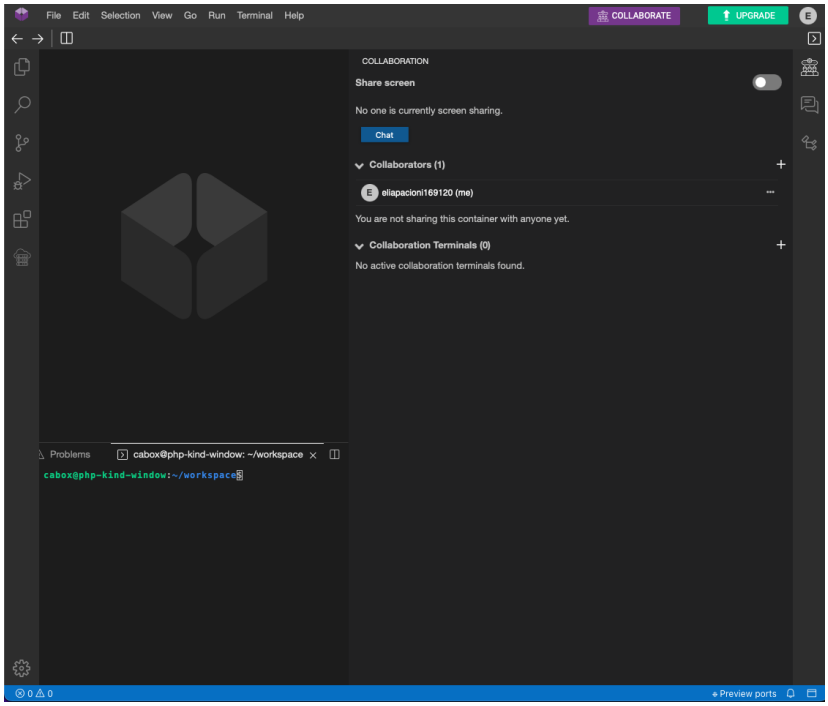


Figura 6.2: Programmazione con *Codeanywhere*

Funzionalità	EduCode	Replit	Codeanywhere
Esecuzione del codice remota	SI	SI	SI
Supporto multlinguaggio	SI	SI	SI
Editor	CodeMirror	Monaco	Monaco
Livecoding	SI	NO	NO
Supervisione docente	SI	SI	NO
Multilingua	SI	NO	NO
Integrazione GitHub	NO	SI	SI
Login tramite social	NO	SI	SI
Sistema di e-learning	SI	NO	NO
ToDo List	SI	NO	NO
Responsive	SI	SI	SI
Esecuzione su server personale	NO	NO	SI

Tabella 6.1: Confronto con i sistemi correlati

CAPITOLO 7

Conclusioni

In questa tesi abbiamo illustrato la progettazione e lo sviluppo di un sistema di e-learning, denominato EduCode, per il supporto all'insegnamento della programmazione. La progettazione dell'applicazione si è articolata nelle tre fasi: specifica e analisi dei requisiti, progettazione della base di dati, progettazione della componente applicativa. Successivamente, è stato implementato il sistema ed è stato reso disponibile online per consentire agli utenti di provarlo e di fornire dei feedback per migliorarne l'utilizzo.

Allo stato attuale, l'applicazione è un sistema di e-learning funzionante e funzionale; è possibile, quindi, affermare che il lavoro svolto costituisca un importante punto di partenza per lo sviluppo futuro e la crescita di EduCode.

Grazie al carattere internazionale ed al supporto di diversi professori, EduCode conta, ad oggi, circa 100 utenti. Attraverso i feedback degli utenti vengono monitorate costantemente le richieste e le problematiche presenti nell'applicazione. In particolare, i feedback vengono analizzati, classificati ed introdotti nella roadmap di sviluppo.

Dal 25 maggio al 22 giugno 2022, EduCode viene usato dalla *Fundación Universidad Sociedad de la Universidad de Extremadura*[1] come piattaforma di supporto al corso di *Clojure*.

Come obiettivi futuri, il più immediato per EduCode è far crescere la piattaforma in ambito universitario, in Italia ed all'estero, continuandone lo sviluppo per essere al passo con i competitor. Il successivo, possibile con l'aumentare dei dati raccolti, è introdurre all'interno dell'applicazione tecniche di Intelligenza Artificiale per aiutare gli utenti nel coding e per proporre versioni dei loro codici sorgente modificati con il supporto dell'Intelligenza Artificiale.

[1] Sito web: `http://www.uexfundacion.es`

Bibliografia

CABIBBO, L. (2021), *Architettura del Software - Strutture e Qualità*, EdizioniEfesto.

CROCKFORD, D. (2009), *Javascript, le tecniche per scrivere il codice migliore*, O'Reilly, Yahoo!Press, Tecniche nuove.

EARLE CASTLEDINE, C. S. (2013), *jQuery, guida completa*, Apogeo.

FOWLER, M. (2015), *UML DISTILLED, guida rapida al linguaggio di modellazione standard*, Pearson.

PAOLO ATZENI, S. C. (2014), *Basi di dati*, McGraw-Hill.

Websites consulted

- Debian – `https://www.debian.org/doc/`
- Nginx –`https://nginx.org/en/docs/`
- Digital Ocean, nginx configuration – `https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-debian-10`
- Laravel, documentazione – `https://laravel.com/docs/`
- AdminLte Teamplte – `https://adminlte.io/themes/AdminLTE/documentation/`
- Bootstrap – `https://getbootstrap.com/docs/`
- jQuery – `https://api.jquery.com`
- Pusher – `https://pusher.com/docs/`
- Laravel WebSocket – `https://beyondco.de/docs/laravel-websockets/`
- xTerm – `https://xtermjs.org/docs/`
- CodeMirror Editor – `https://codemirror.net`
- Ratchet WebSocket – `http://socketo.me`

Ringraziamenti

Questa tesi rappresenta la conclusione di un percorso molto importante; al suo interno sono racchiuse gioie, dolori, ansie, paure, sogni e progetti futuri. Devo ringraziare davvero tante persone, ognuno a modo suo è stato significativo e mi ha aiutato a raggiungere questo obiettivo!

Ringrazio i miei genitori che hanno gioito con me ad ogni esame superato e hanno cercato di confortarmi ogni volta che qualcosa è andato storto, senza mai farmi pesare i sacrifici fatti. Spero che questo traguardo possa in parte ripagali e renderli orgogliosi.

Ringrazio Arianna e Marco per essere stati presenti in tutto il mio percorso, per avermi ospitato a casa loro quando avevo bisogno di un posto tranquillo dove studiare e per le telefonate anche solo per farmi compagnia e tirarmi su di morale.

Ringrazio il mio relatore, Prof. Domenico Ursino, per la professionalità e l'infinita disponibilità. Anche nei momenti più difficili mi ha aiutato ed ha compreso le mie paure. Avere un Professore come lei è stata una fortuna.

Ringrazio il mio correlatore, Prof. Francisco Fernández De Vega, che fin dai primi mesi dell'Erasmus mi ha proposto di collaborare ai suoi progetti, supportandomi nello sviluppo e spronandomi a fare sempre meglio.

Ringrazio Rossella, mentore e guida fin dalla prima lezione di informatica. I suoi insegnamenti sono stati preziosi e mi hanno permesso di affrontare gli esami di informatica senza paura. È anche grazie a lei se ho resistito fino al raggiungimento di questo traguardo. Non ha mai smesso di credere in me.

Ringrazio i miei parenti, sempre presenti e pronti a tendermi una mano; le cene della domenica sera sono state il motivo principale per cui volevo tornare a casa.

Ringrazio tutti i miei amici, da quelli che conosco da anni a quelli che ho incontrato durante il percorso; ognuno a modo suo ha contributo a rendere unica e speciale questa esperienza. Grazie di cuore.

Ringrazio Alberto per le merendine tirate per svegliarmi ogni mattina, è merito suo se ho seguito le lezioni! I due anni passati da coinquilini sono tra i ricordi più belli del mio

percorso universitario.

Ringrazio Fabio, amico e compagno di studi. L'esame di elettronica preparato alternando studio e partite a biliardino è tra i periodi più divertenti e produttivi. Sempre pronto a supportarmi, consigliarmi e spronarmi a dare il massimo.

Ringrazio i miei compagni di studio per aver condiviso con me progetti, esami e giornate in aula studio, dove si finiva sempre per scherzare anche quando avevamo paura di non superare l'esame successivo. Con voi non ci si annoia mai.

Ringrazio Fede che mi ha sopportato, supportato ed incoraggiato, senza mai lasciarmi da solo. Grazie per aver compreso la scelta di partire per l'Erasmus anche se significava stare distanti per molto tempo, non lo scorderò mai. Grazie per aver affrontato la paura dell'aereo pur di passare insieme qualche giorno in più e visitare i posti in cui vivevo. Grazie per i momenti condivisi, lo studio, i viaggi o semplicemente una cena, ogni istante è stato fondamentale in questo percorso.

Infine, voglio ringraziare me stesso. Potrebbe sembrare narcisista, ma credo davvero che sia necessario dirmi grazie per essere andato avanti ad ogni costo. Se ho raggiunto questo traguardo è anche grazie alla mia forza di volontà ed a tutti voi che mi avete supportato.