



**UNIVERSITÀ
DEGLI STUDI
DI BERGAMO**

Corso di Laurea Magistrale in Ingegneria Informatica

Progetto per il corso di Progettazione, Algoritmi e
Computabilità

StudyMate

Docente:

Chiar.ma Prof. Patrizia Scandurra

Componenti del gruppo:

Cassina Andrea

Morè Gabriele

Olivieri Paolo

Anno Accademico 2022-2023

Immagini

1	Diagramma degli use case	4
2	Topologia del sistema	11
3	Pattern Model-View-Controller	13
1.1	Diagramma dei componenti 1	16
1.2	Diagramma dei componenti 2	16
1.3	Diagramma delle classi (interfacce)	17
1.4	Diagramma delle classi (tipi di dati)	18
1.5	Diagramma di deployment	19
1.6	Diagramma dei package	20
2.1	Diagramma entità relazioni ASP.NET	24
2.2	Diagramma entità relazioni	25
2.3	Diagramma delle classi (tipi di dati)	26
2.4	Diagramma dei componenti	27
2.5	Diagramma dei package	28
2.6	Metriche di qualità del codice	28
3.1	Diagramma entità relazioni	32
3.2	Diagramma delle classi (interfacce)	33
3.3	Diagramma dei package	34
3.4	Test API Get Courses	35
3.5	Test API Get Lessons	35
3.6	GET Courses	36
3.7	Risposta GET Courses	36
3.8	Verifica correttezza risposta GET Courses	36

3.9	GET Courses errata	36
3.10	Verifica correttezza risposta GET Courses con chiamata errata	36
3.11	GET Lessons	37
3.12	Risposta GET Lessons	37
3.13	Verifica correttezza risposta GET Lessons	37
3.14	GET Lessons errata	37
3.15	Verifica correttezza risposta GET Lessons con chiamata errata	38
3.16	Metriche di qualità del codice	38
4.1	Diagramma entità relazioni	40
4.2	Diagramma delle classi (tipi di dati)	41
4.3	Diagramma delle classi (interfacce)	42
4.4	Diagramma dei package	43
4.5	Metriche di qualità del codice	51
5.1	Schermata Login	52
5.2	Schermata Register	53
5.3	Dropdown scelta corso	53
5.4	Schermata Home	54
5.5	Schermata Calendar	55
5.6	Schermata Personal	56

Indice

0	Iterazione 0	2
0.1	Contesto e obiettivi	2
0.2	Analisi dei requisiti	3
0.2.1	Sistema e casi d'uso	3
0.2.2	Requisiti dell'algoritmo	10
0.2.3	Requisiti non funzionali	10
0.2.4	Architettura del sistema	11
0.2.5	Design patterns	11
0.2.6	Toolchain	13
1	Iterazione 1	15
1.1	Descrizione	15
1.2	Diagramma dei componenti	15
1.2.1	Raggruppamento use cases	15
1.3	Diagramma delle classi (interfacce)	17
1.4	Diagramma delle classi (tipi di dati)	18
1.5	Diagramma di deployment	18
1.6	Diagramma dei package	19
2	Iterazione 2	21
2.1	Descrizione	21
2.2	Casi d'uso	21
2.2.1	Diagramma entità relazioni Identity (ER)	23
2.3	Diagramma entità relazioni (ER)	25
2.4	Diagramma delle classi	25

2.5	Diagramma delle interfacce	26
2.6	Diagramma dei componenti	26
2.7	Diagramma dei package	27
2.8	Analisi statica del codice	28
3	Iterazione 3	29
3.1	Descrizione	29
3.2	Casi d'uso	29
3.3	Diagramma entità relazioni (ER)	32
3.4	Diagramma delle classi (interfacce)	33
3.5	Diagramma dei package	33
3.6	API	34
3.6.1	Recupero dei corsi - GET Courses	35
3.6.2	Recupero lezioni - GET Lessons	37
3.7	Analisi statica del codice	38
4	Iterazione 4	39
4.1	Descrizione	39
4.2	Casi d'uso	39
4.2.1	UC7 - Generazione del calendario	39
4.3	Diagramma entità relazioni (ER)	40
4.4	Diagramma delle classi	41
4.5	Diagramma delle interfacce	41
4.6	Diagramma dei package	43
4.7	Algoritmo di generazione del calendario	44
4.7.1	Funzione Greedy	44
4.7.2	Funzione SearchCandidates	45
4.7.3	Funzione SearchFreeHours	46
4.7.4	Funzione Ottimo	47
4.7.5	Funzione Ammissibile	48
4.8	Analisi della complessità	49
4.8.1	Complessità della funzione <i>SearchCandidates</i>	49
4.8.2	Complessità della funzione <i>SearchFreeHours</i>	50

4.8.3	Complessità della funzione <i>Ottimo</i>	50
4.8.4	Complessità della funzione <i>Ammissibile</i>	50
4.8.5	Complessità della funzione <i>Greedy</i>	50
4.9	Analisi statica del codice	51
5	Manuale utente	52
6	Conclusioni	57

Iterazione 0

0.1 Contesto e obiettivi

Il progetto che presentiamo in questa documentazione è una web-app pensata per aiutare gli studenti universitari a organizzare al meglio la propria giornata di studio. L'app offre diversi piani di studio che possono essere personalizzati in base ai corsi che l'utente vuole seguire e agli impegni che ha in agenda, come il lavoro, gli allenamenti e così via ai quali può essere attribuita una delle seguenti priorità:

- massima(1) (esempio: lavoro)
- media(2) (esempio: evento di routine)
- bassa(3) (esempio: lezione)

Grazie alla semplicità di utilizzo e all'interfaccia intuitiva, lo studente può facilmente gestire il suo piano di studio e aggiungere o rimuovere corsi e impegni in modo rapido e intuitivo. Un'altra importante funzionalità dell'app è quella di permettere allo studente di inserire un numero di ore di pausa giornaliera a sua scelta, considerando anche i suoi orari di sonno, i quali verranno impostati con una priorità massima. In questo modo, lo studente può pianificare le sue pause cosicché da avere il giusto equilibrio tra studio e riposo, garantendo una maggiore produttività e concentrazione durante le ore di studio. Lo scopo ultimo dell'app è quello di inserire gli slot di studio individuale nel calendario dello studente in modo automatico, tenendo conto del numero di crediti formativi di ciascun esame e utilizzando un algoritmo per calcolare il totale di ore di studio necessarie. In questo modo, lo studente può avere una panoramica chiara della sua giornata di studio e dei progressi fatti verso il raggiungimento degli obiettivi di apprendimento.

0.2 Analisi dei requisiti

All'interno del sistema è stato individuato un tipo di attore:

- lo studente per la gestione del suo calendario: si occuperà dell'inserimento, ed eventualmente dell'eliminazione, degli eventi a cui vuole partecipare e delle relative priorità, dell'inserimento degli orari di pausa e di sonno e la visualizzazione dell'intero calendario

0.2.1 Sistema e casi d'uso

L'utente deve poter effettuare le seguenti operazioni:

- UC1 - Effettuare la login per visualizzare il calendario
- UC2 - Effettuare la registrazione per l'iscrizione al sistema
- UC3 - Effettuare la logout per disconnettersi dal sistema
- UC4 - Gestire l'account
 - UC4.1 - Modifica delle credenziali
 - UC4.2 - Modifica degli orari di routine
- UC5 - Gestione impegni personali
 - UC5.1 - Inserimento impegni personali
 - UC5.2 - Modifica impegni personali
 - UC5.3 - Eliminazione impegni personali
- UC6 - Gestione dei corsi
 - UC6.1 - Inserimento dei corsi
 - UC6.2 - Modifica dei corsi
 - UC6.3 - Eliminazione dei corsi
- UC7 - Generazione del calendario

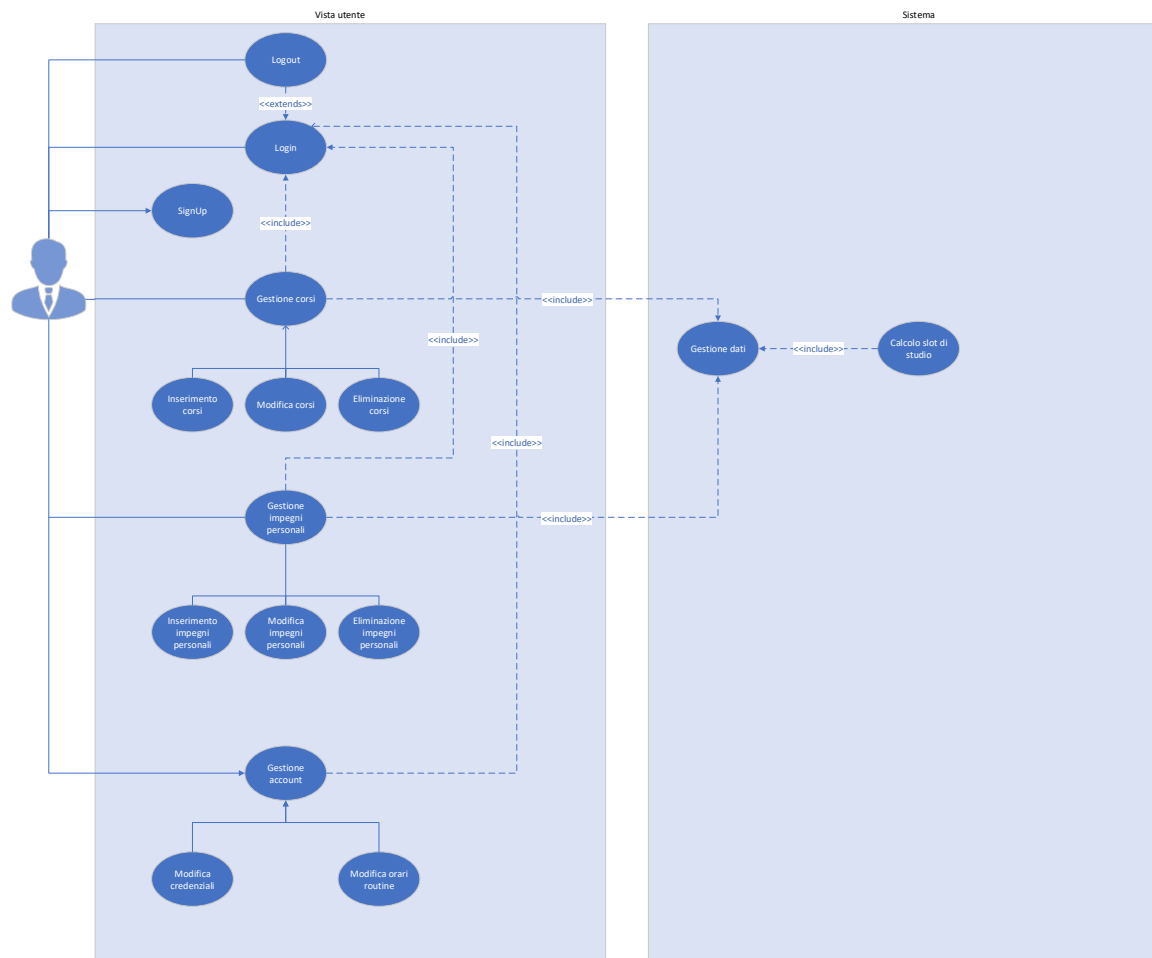


Figura 1: Diagramma degli use case

Di seguito si riporta il dettaglio di ciascun caso d'uso.

UC1	Login
<i>Descrizione:</i>	Autenticazione di un utente al sistema
<i>Attori:</i>	Utente
<i>Precondizioni:</i>	L'utente deve essere in possesso delle credenziali
<i>Flusso eventi:</i>	
	1. L'utente inserisce le proprie credenziali
	2. Il sistema controlla i dati inseriti e autentica l'utente
<i>Postcondizioni:</i>	Reindirizzamento alla pagina del calendario
<i>Eccezioni:</i>	L'utente inserisce credenziali errate

UC2	Sign-up
<i>Descrizione:</i>	Creazione del proprio account
<i>Attori:</i>	Utente
<i>Precondizioni:</i>	-
<i>Flusso eventi:</i>	<ol style="list-style-type: none"> 1. L'utente inserisce il proprio nome 2. L'utente inserisce il proprio cognome 3. L'utente inserisce il proprio username 4. L'utente inserisce la propria mail 5. L'utente inserisce la password
<i>Postcondizioni:</i>	Conferma della registrazione tramite mail
<i>Eccezioni:</i>	<ol style="list-style-type: none"> 1. Username già in uso 2. Mail non valida 3. Password non valida
UC3	Logout
<i>Descrizione:</i>	Disconnessione dal proprio account
<i>Attori:</i>	Utente
<i>Precondizioni:</i>	L'utente è già autenticato
<i>Flusso eventi:</i>	<ol style="list-style-type: none"> 1. L'utente richiede la disconnessione
<i>Postcondizioni:</i>	Reindirizzamento alla pagina di accesso
<i>Eccezioni:</i>	
UC4.1	Modifica delle credenziali

Descrizione: Modifica di username e password

Attori: Utente

Precondizioni: L'utente è autenticato

Flusso eventi:

1. L'utente modifica l'username

2. L'utente modifica la password

Postcondizioni: Reindirizzamento alla pagina di gestione account

Eccezioni:

1. Username già in uso

2. Password non valida

UC4.2	Modifica degli orari di routine
--------------	----------------------------------------

Descrizione: Modifica degli orari di routine personali

Attori: Utente

Precondizioni: L'utente è autenticato

Flusso eventi:

1. L'utente imposta gli orari di routine

Postcondizioni: Reindirizzamento alla pagina di gestione account

Eccezioni: -

UC5.1	Inserimento impegni personali
--------------	--------------------------------------

Descrizione: Inserimento degli impegni personali dell'utente

Attori: Utente

Precondizioni: L'utente è autenticato

Flusso eventi:

1. L'utente inserisce la descrizione dell'evento

2. L'utente inserisce l'orario dell'evento

3. L'utente inserisce la cadenza dell'evento

4. L'utente inserisce la priorità dell'evento

Postcondizioni: Reindirizzamento al calendario

Eccezioni: Sovrapposizione di un evento a priorità minore su un evento a priorità più alta

UC5.2 Modifica impegni personali

Descrizione: Modifica degli impegni personali dell'utente

Attori: Utente

Precondizioni:

1. L'utente è autenticato

2. L'utente ha già inserito degli eventi

Flusso eventi:

1. L'utente modifica la descrizione dell'evento

2. L'utente modifica l'orario dell'evento

3. L'utente modifica la cadenza dell'evento

4. L'utente modifica la priorità dell'evento

Postcondizioni: Reindirizzamento al calendario

Eccezioni: Sovrapposizione di un evento a priorità minore su un evento a priorità più alta

UC5.3 Eliminazione impegni personali

Descrizione: Eliminazione degli impegni personali dell'utente

Attori: Utente

Precondizioni:

1. L'utente è autenticato

2. L'utente ha già inserito degli eventi

Flusso eventi:

1. L'utente cancella l'evento

Postcondizioni: Reindirizzamento al calendario

Eccezioni: -

UC6.1	Scelta del corso di laurea
-------	----------------------------

Descrizione: Inserimento del corso di laurea a cui l'utente è iscritto

Attori: Utente

Precondizioni: L'utente è autenticato

Flusso eventi:

1. L'utente sceglie il corso di laurea

2. L'utente sceglie i corsi a scelta

Postcondizioni: Reindirizzamento al calendario

Eccezioni: -

UC6.2	Modifica dei corsi a scelta
-------	-----------------------------

Descrizione: Modifica dei corsi a scelta a cui l'utente si è precedentemente iscritto

Attori: Utente

Precondizioni:

1. L'utente è autenticato

2. L'utente ha precedentemente selezionato i corsi a scelta

Flusso eventi:

1. L'utente modifica il corso precedentemente inserito

Postcondizioni: Reindirizzamento al calendario

Eccezioni: -

UC6.3	Eliminazione dei corsi a scelta
--------------	----------------------------------------

Descrizione: Eliminazione dei corsi a scelta a cui l'utente si è precedentemente iscritto

Attori: Utente

Precondizioni:

1. L'utente è autenticato
2. L'utente ha precedentemente selezionato i corsi a scelta

Flusso eventi:

1. L'utente elimina il corso precedentemente inserito

Postcondizioni: Reindirizzamento al calendario

Eccezioni: -

UC7	Generazione del calendario
------------	-----------------------------------

Descrizione: Visualizzazione del calendario aggiornato

Attori: Utente

Precondizioni:

1. L'utente è autenticato

Flusso eventi:

1. L'utente accede al suo account
2. Viene caricata la pagina contenente il calendario

Postcondizioni: Reindirizzamento al calendario

Eccezioni: -

0.2.2 Requisiti dell'algoritmo

In fase di inizializzazione dell'app, l'utente, inserendo il corso di studi e gli impegni personali con cadenza regolare, ottiene dall'algoritmo il programma settimanale comprendente lezioni, ore di studio ed impegni personali.

In input, quindi, l'algoritmo riceve i seguenti dati:

- il piano di studio scelto dall'utente con gli orari delle lezioni
- gli impegni personali e di routine dell'utente
- le ore di pausa o libere dell'utente (orari di sonno, pranzo e cena, ecc.)
- le impostazioni scelte dall'utente (ad esempio le priorità)

In output restituisce il calendario settimanale con gli orari ottimizzati per lo studio, il tempo libero, le lezioni e così via. Una volta che l'utente inserisce un nuovo evento, l'algoritmo verifica che non vi siano sovrapposizioni. Nell'eventualità di una coincidenza, verrà selezionato l'impegno con priorità più alta e, nel caso in cui vi fossero più impegni con stessa priorità, l'utente dovrà selezionarne uno solo.

0.2.3 Requisiti non funzionali

In caso di variazioni di un evento pianificato dall'utente in una determinata giornata, il sistema dovrà ricalcolare il programma sulla base della modifica.

Per quanto riguarda la portabilità, la web-app deve essere accessibile da qualsiasi dispositivo, come computer, tablet o smartphone, in modo tale che gli studenti possano utilizzarla in movimento o da qualsiasi luogo.

Dal punto di vista della sicurezza, l'app deve garantire la protezione dei dati personali degli studenti e dei loro impegni, garantendo la privacy e la sicurezza dei dati, in modo che lo studente possa visualizzare esclusivamente il suo calendario.

Al fine di garantire la qualità di manutenibilità la progettazione e realizzazione di un sistema avverrà per componenti in modo da favorire eventuali modifiche future.

Infine, in quanto ad integrazione, l'app deve essere in grado di integrarsi con altre app e servizi, come ad esempio calendari o app di gestione del tempo, in modo da offrire una soluzione completa e facilmente integrabile per la pianificazione della settimana di studio.

0.2.4 Architettura del sistema

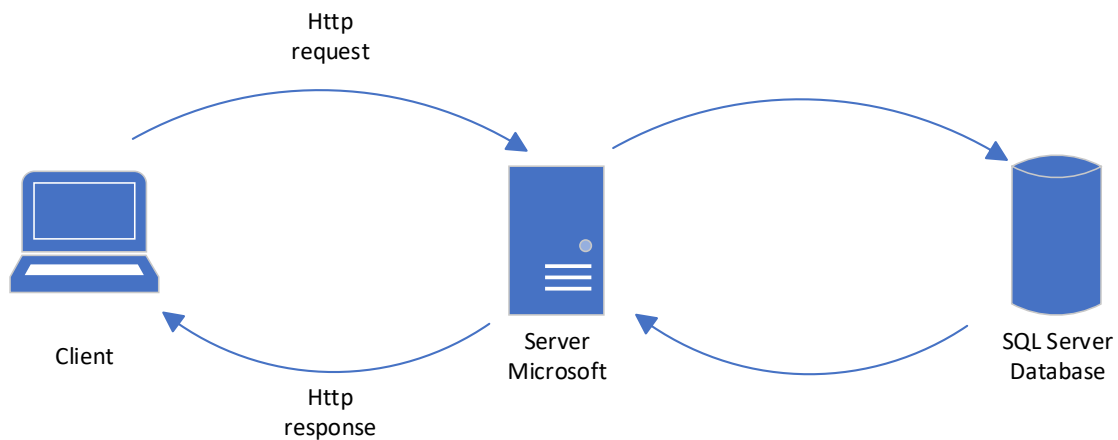


Figura 2: Topologia del sistema

Il sistema sarà costituito da 2 componenti principali:

1. un'applicazione web che espone tramite api REST le risorse necessarie, memorizzate in un database sulla stessa macchina
2. una single page application che fungerà da interfaccia tra l'utente e la web-app nell'applicazione web

0.2.5 Design patterns

Il sistema si baserà su una architettura client-server. Il progetto delle web api è strutturato secondo il pattern Model-View-Controller. In questo pattern, il codice viene suddiviso in tre componenti principali: il modello, la vista e il controller.

- Il modello rappresenta i dati e le regole di business dell'applicazione,
- la vista è l'interfaccia utente con cui l'utente interagisce
- il controller gestisce le interazioni tra modello e vista.

Per l'accesso ai dati si utilizza pattern Repository.

In particolare, verrà implementato con Entity Framework che è un framework di mapping object-relational (ORM) che fornisce un livello di astrazione tra il database relazionale e il codice delle applicazioni. Utilizza il concetto di "entità" per rappresentare i dati presenti nel database sotto forma di oggetti di classe. Gli sviluppatori possono

quindi utilizzare queste classi per interagire con il database senza dover scrivere codice SQL. A livello di runtime, Entity Framework utilizza ADO.NET per connettersi al database e eseguire query e operazioni di aggiornamento. ADO.NET fornisce una serie di componenti e API per la connessione ai database, l'esecuzione di query e il recupero dei risultati.

Nel progetto Client Web l'interazione tra interfaccia e dati avviene mediante pattern Model View Controller, con repository per l'accesso ai dati mediante chiamate HTTPS. Utilizzare il pattern MVC offre molti vantaggi, tra cui una maggiore separazione delle responsabilità, una maggiore facilità di manutenzione e una maggiore flessibilità nello sviluppo. Inoltre, esistono molti framework MVC disponibili che rendono più semplice l'utilizzo di questo pattern nello sviluppo di applicazioni web.

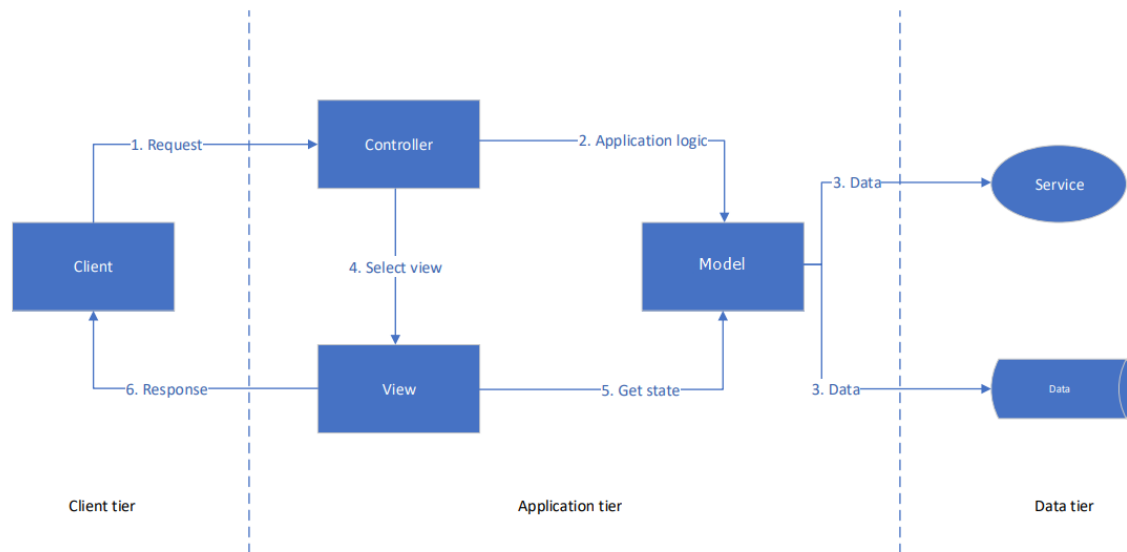


Figura 3: Pattern Model-View-Controller

0.2.6 Toolchain

Strumenti utilizzati per la realizzazione del progetto:

- Modellazione diagrammi: Microsoft Visio
- Implementazione software back-end:
 - Tecnologia: ASP.NET Core 6, linguaggio C#
 - Framework e librerie: Entity Framework (ORM), Swagger
 - IDE: Visual Studio 2022
 - DBMS: SQL Server
 - Gestione DB: SQL Server Management Studio (SSMS), Azure
- Implementazione client web:
 - Tecnologia: Linguaggio C#
 - Framework e librerie: RazorPages
 - IDE: Visual Studio 2022
- Analisi del software:
 - Analisi statica: Visual Studio code analysis
- Testing

- Postman
- Documentazione, versioning:
 - Versioning: Git con GitHub
 - Documentazione: Latex con Overleaf

Iterazione 1

1.1 Descrizione

In questa iterazione viene rappresentato il progetto nel suo complesso, mentre la progettazione dettagliata dei singoli componenti viene posticipata alle iterazioni successive.

1.2 Diagramma dei componenti

1.2.1 Raggruppamento use cases

Al fine di comprendere i componenti che formeranno il sistema, si sono organizzati gli use case in gruppi di affinità in base alle loro funzionalità:

- Authentication (UC1, UC2, UC3)
- User management (UC4.1, UC4.2)
- Task management (UC5.1, UC5.2, UC5.3, UC6.1, UC6.2, UC6.3)
- Task Scheduler (UC7)

L'elaborazione del diagramma è stata condotta in modo graduale, partendo dalla rappresentazione del sistema come un componente unico e di grandi dimensioni che esibisce alcune interfacce, per poi procedere alla suddivisione delle interfacce stesse in componenti più piccoli, al fine di perfezionare la struttura finale del sistema.

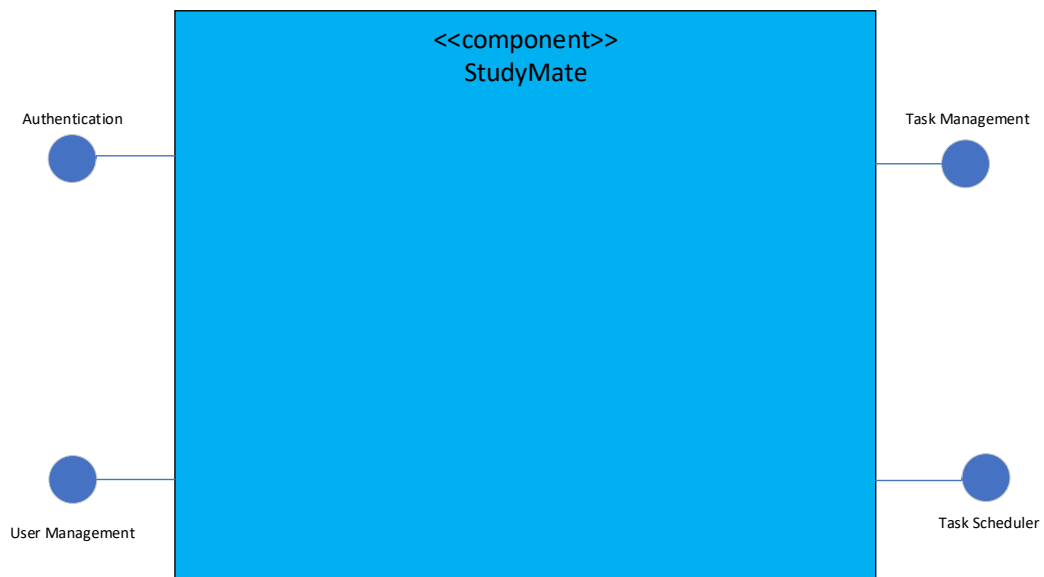


Figura 1.1: Diagramma dei componenti 1

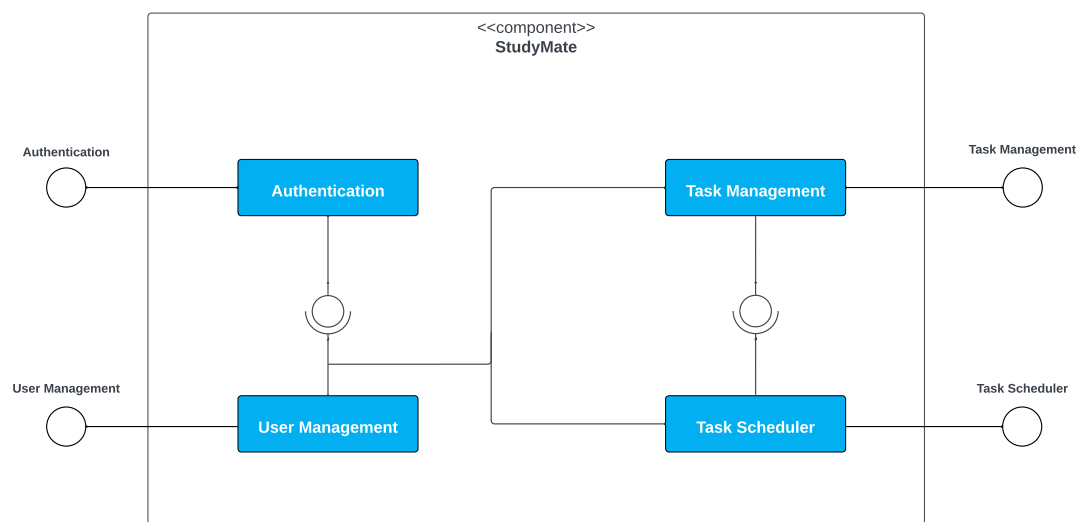


Figura 1.2: Diagramma dei componenti 2

1.3 Diagramma delle classi (interfacce)

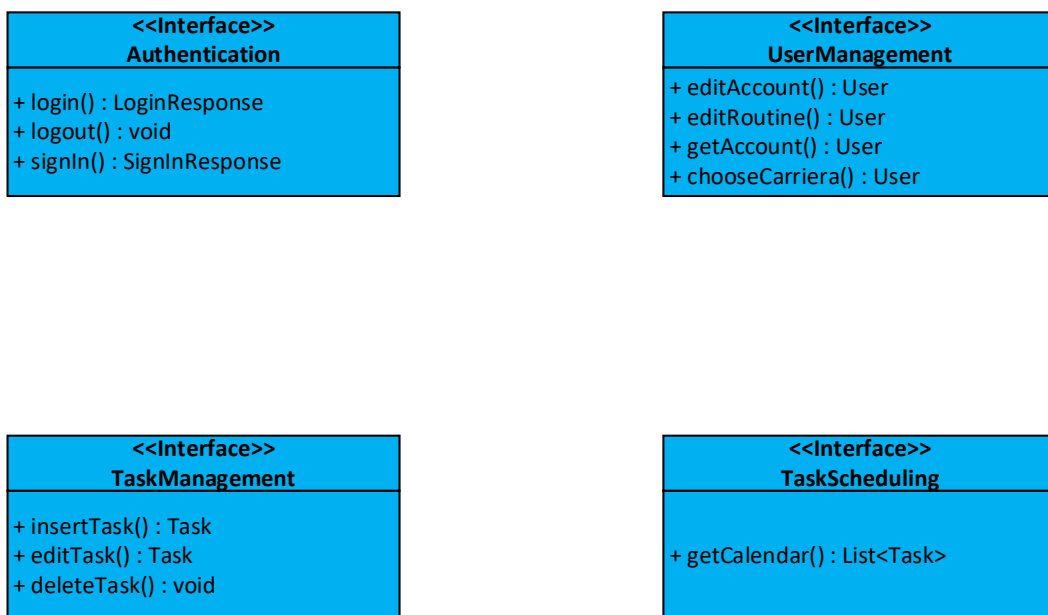


Figura 1.3: Diagramma delle classi (interfacce)

1.4 Diagramma delle classi (tipi di dati)

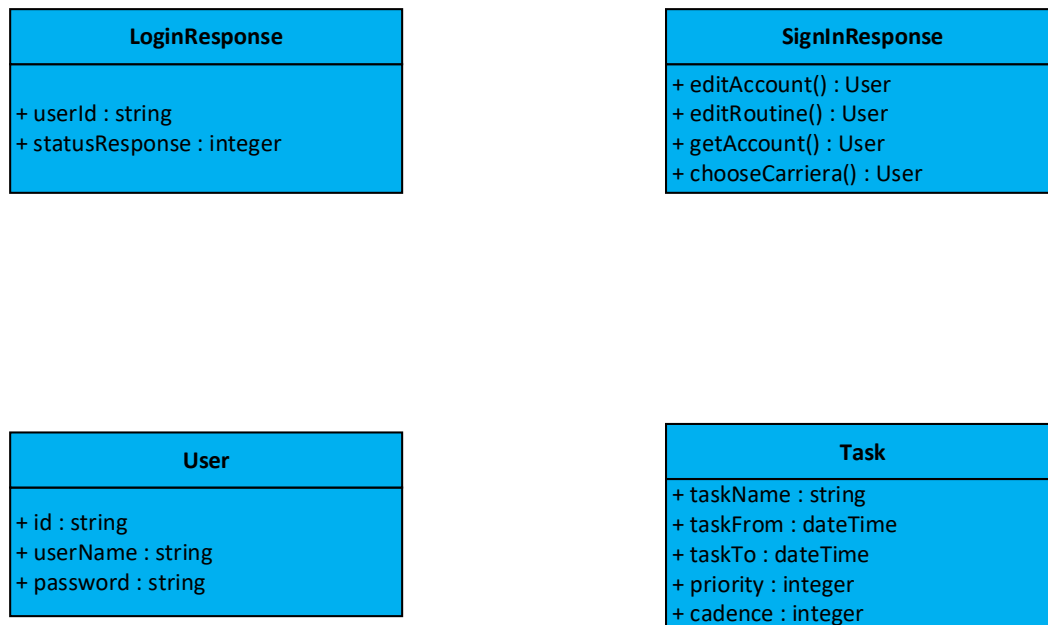


Figura 1.4: Diagramma delle classi (tipi di dati)

1.5 Diagramma di deployment

Compaiono 3 nodi:

- il server che esegue l'applicazione e contiene il database
- il server API che espone le carriere selezionabili dall'utente
- il dispositivo dell'utente per accedere al client web.

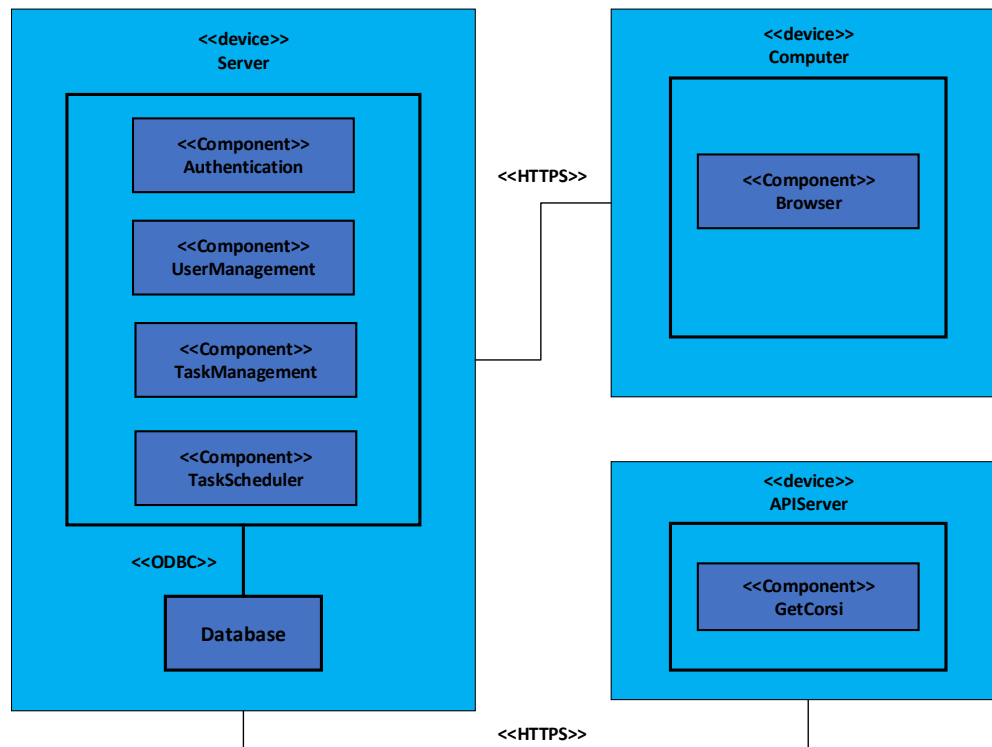


Figura 1.5: Diagramma di deployment

1.6 Diagramma dei package

La suddivisione in package si rifà al modello Model-View-Controller.

I package principali sono:

- StudyMate.Model: contiene i modelli che rappresentano i dati e la logica di business. Le funzioni di accesso al database e le funzioni per l'accesso alle API
- StudyMate.View: contiene le funzioni per la presentazione dei risultati. Generazione del codice HTML
StudyMate.View.Home
- StudyMate.Controller: contiene tutte le logiche, in particolare l'algoritmo Greedy per la generazione del calendario.

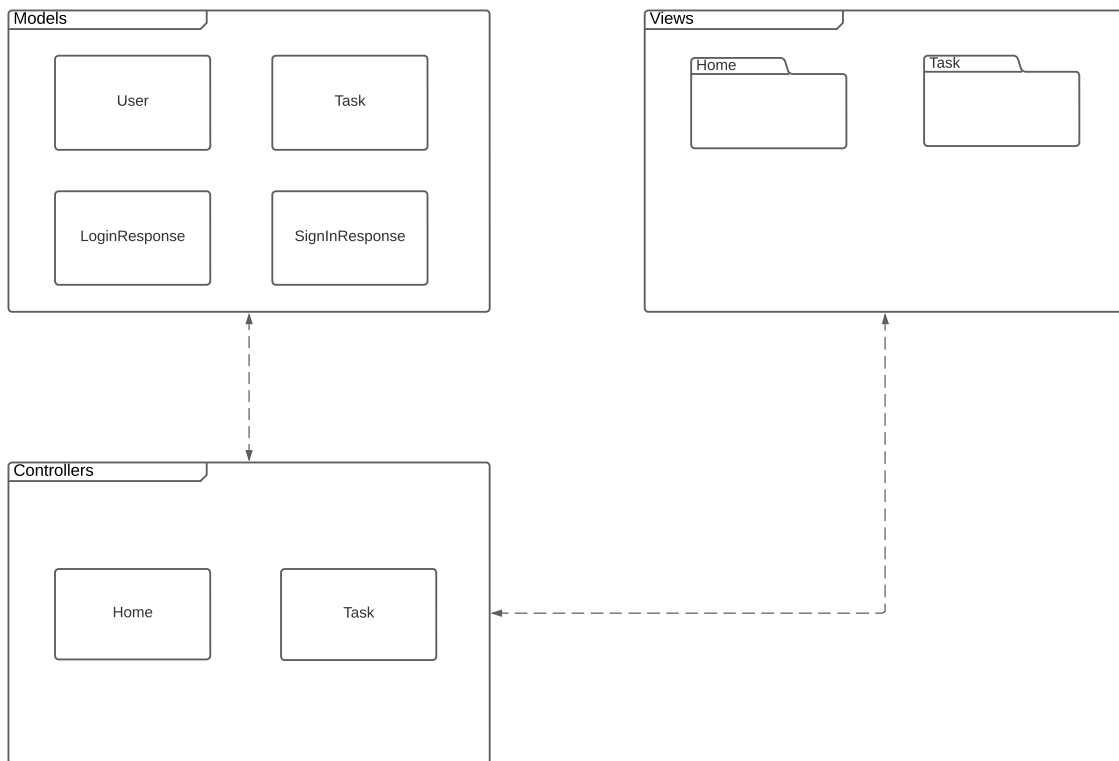


Figura 1.6: Diagramma dei package

Iterazione 2

2.1 Descrizione

Durante questa iterazione, sono stati progettati e sviluppati i casi d'uso per l'autenticazione (UC1, UC2, UC3), i quali forniscono le funzionalità di base necessarie per il resto del progetto. Al completamento di questa fase, gli utenti saranno in grado di autenticarsi nel sistema e accedere solo alle aree per cui sono autorizzati (per eseguire le funzionalità che verranno sviluppate nelle iterazioni successive).

2.2 Casi d'uso

Viene utilizzato "Identity", un framework di autenticazione e autorizzazione che fa parte di ASP.NET Core. Questo framework consente di gestire l'identità degli utenti all'interno di un'applicazione web, compreso l'accesso, la registrazione, il ripristino della password e la gestione dei ruoli.

ASP.NET Core Identity fornisce un set di funzionalità integrato che facilita la gestione dell'autenticazione e dell'autorizzazione all'interno di un'applicazione web (tra cui i provider di autenticazione come Facebook, Google, Twitter e Microsoft).

Oltretutto, è altamente personalizzabile, consentendo di configurare e adattare l'esperienza utente per soddisfare le esigenze specifiche dell'applicazione.

UC1	Login
<i>Descrizione:</i>	Autenticazione di un utente al sistema
<i>Attori:</i>	Utente

Precondizioni: L'utente deve essere in possesso delle credenziali

Flusso eventi:

1. L'utente inserisce le proprie credenziali
2. Il sistema controlla i dati inseriti e autentica

l'utente

Postcondizioni: Reindirizzamento alla pagina del calendario

Eccezioni: L'utente inserisce credenziali errate

UC2

Sign-up

Descrizione: Creazione del proprio account

Attori: Utente

Precondizioni: -

Flusso eventi:

1. L'utente inserisce il proprio nome
2. L'utente inserisce il proprio cognome
3. L'utente inserisce il proprio username
4. L'utente inserisce la propria mail
5. L'utente inserisce la password

Postcondizioni: Conferma della registrazione tramite mail

Eccezioni:

1. Username già in uso
 2. Mail non valida
 3. Password non valida
-

UC3	Logout
<i>Descrizione:</i>	Disconnessione dal proprio account
<i>Attori:</i>	Utente
<i>Precondizioni:</i>	L'utente è già autenticato
<i>Flusso eventi:</i>	
	1. L'utente richiede la disconnessione
<i>Postcondizioni:</i>	Reindirizzamento alla pagina di accesso
<i>Eccezioni:</i>	-

2.2.1 Diagramma entità relazioni Identity (ER)

Utilizzando il framework Identity di ASP.NET Core, è possibile creare facilmente un diagramma ER (Entity-Relationship) personalizzato per l'applicazione web.

Il framework Identity fornisce un modello di dati predefinito che rappresenta l'identità degli utenti, i ruoli e le autorizzazioni. Tuttavia, questo modello può essere personalizzato per soddisfare le esigenze specifiche dell'applicazione, come detto in precedenza. Utilizzando gli strumenti di scaffolding di ASP.NET Core, è possibile generare automaticamente codice per le entità del modello e la logica del controller corrispondente.

In questo modo, è possibile creare rapidamente un diagramma ER personalizzato per l'applicazione web, che riflette la struttura del database e la relazione tra le diverse entità.

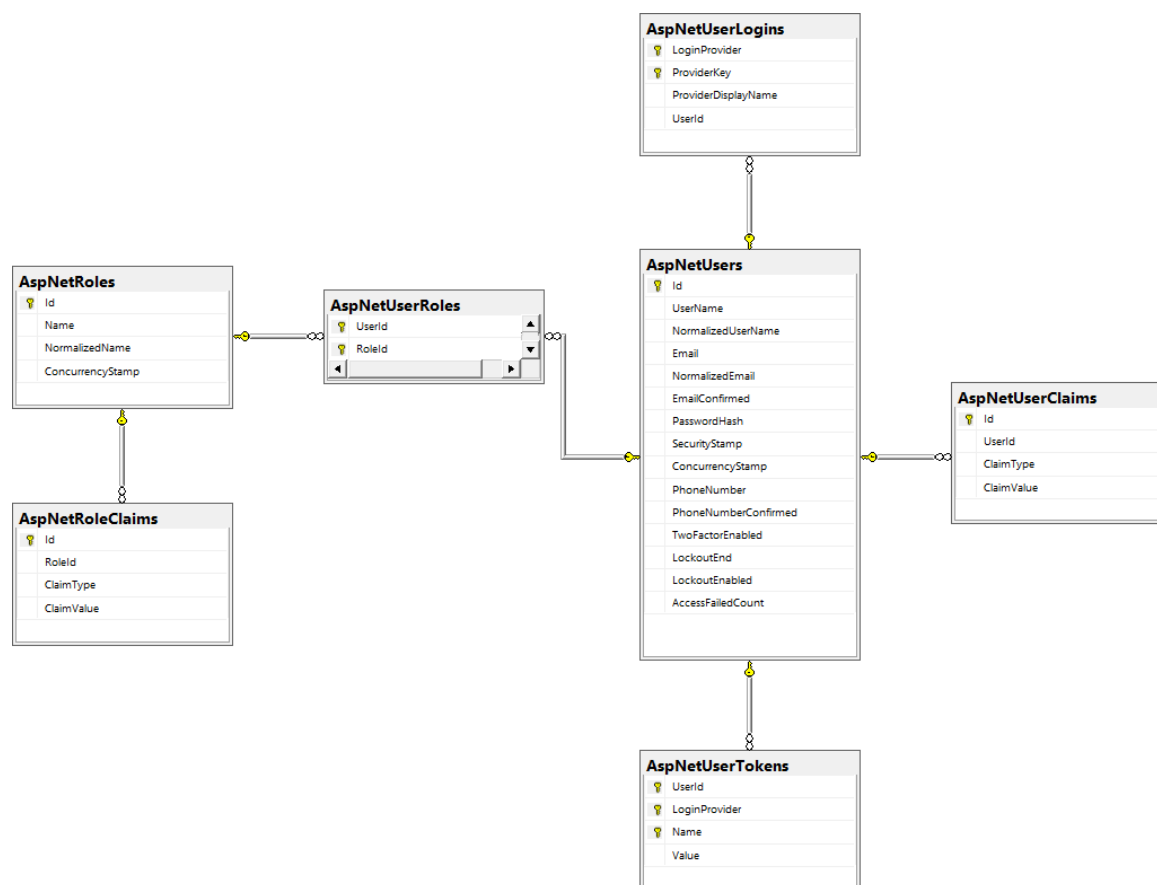


Figura 2.1: Diagramma entità relazioni AspNet

2.3 Diagramma entità relazioni (ER)

È stato necessario implementare la fase di login in questa iterazione, la quale richiede la creazione di un database. Il database è stato rappresentato nella figura seguente attraverso un diagramma ER.

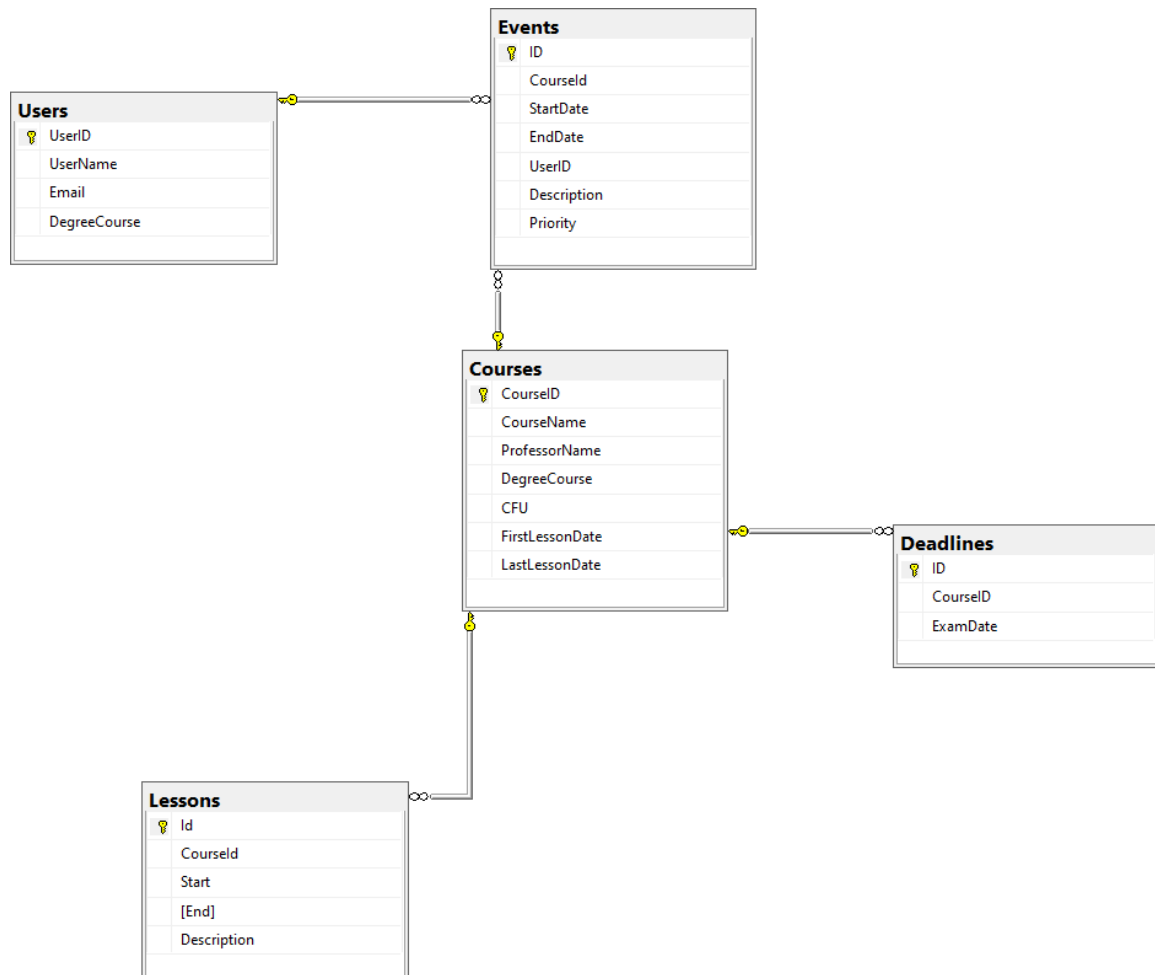


Figura 2.2: Diagramma entità relazioni

2.4 Diagramma delle classi

Nella fase corrente, alcune classi ipotizzate durante l'iterazione 1 sono state oggetto di modifiche.

In particolare, le classi *LoginResponse* e *SignInResponse* sono state eliminate.

La classe *User* è stata rinominata *ApplicationUser*, mentre la precedente classe *Task* è stata suddivisa in due: *Events* e *Course*, che rappresentano rispettivamente gli eventi personali aggiunti dallo studente e i corsi universitari che segue.

Aggiunte inoltre le nuove classi *Lesson*, indicante l'orario delle lezioni di un corso, e *Deadline*, corrispondente alla data dell'esame di un determinato corso.

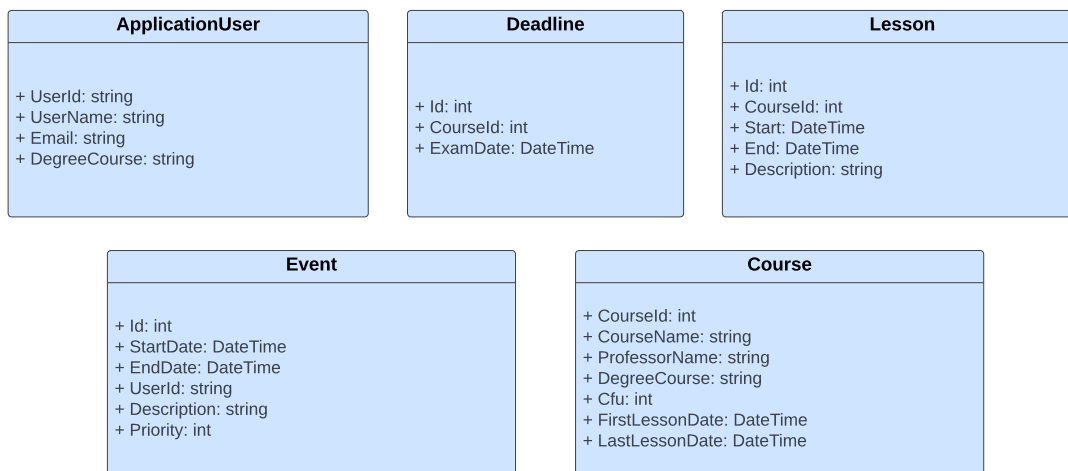


Figura 2.3: Diagramma delle classi (tipi di dati)

2.5 Diagramma delle interfacce

Il framework Identity di ASP.NET Core consente di gestire automaticamente le seguenti interfacce precedentemente implementate:

- **Authentication**: l'interfaccia di autenticazione gestisce l'accesso degli utenti all'applicazione web, fornendo funzionalità come la registrazione, l'accesso, la disconnessione e il ripristino della password.
- **UserManagement**: l'interfaccia di gestione utenti consente di creare, modificare e eliminare gli account degli utenti, oltre a fornire funzionalità di gestione dei ruoli e delle autorizzazioni.

Tutte queste interfacce sono gestite automaticamente dal framework Identity di ASP.NET Core per cui non vengono ulteriormente dettagliate nel seguito di questa documentazione. Rimangono inalterate le altre interfacce presenti nell'iterazione precedente.

2.6 Diagramma dei componenti

Nel seguito si mostra la gestione dei componenti Authentication e UserManagement gestiti dal componente esterno Identity Framework.

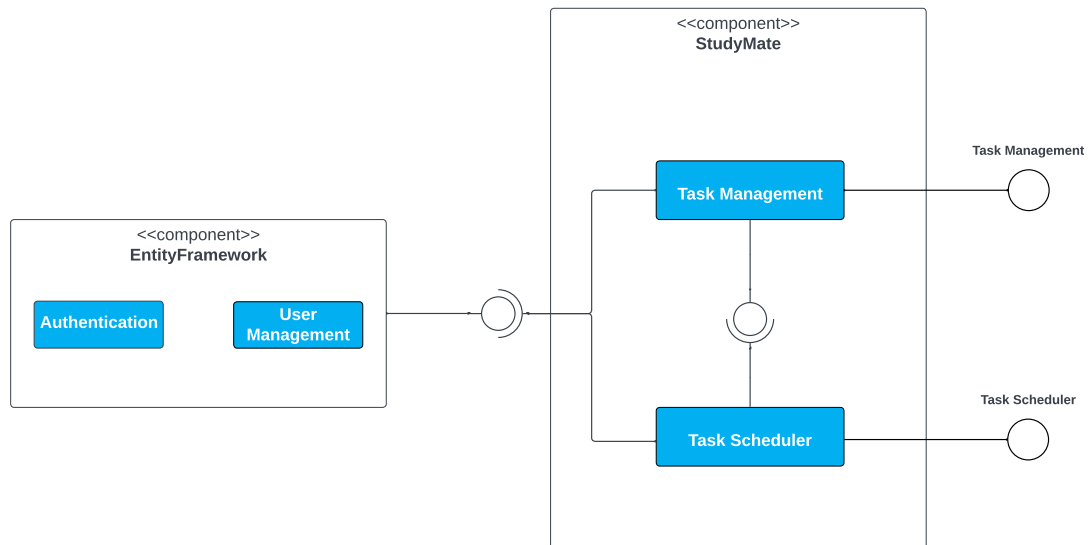


Figura 2.4: Diagramma dei componenti

2.7 Diagramma dei package

In questa iterazione sono stati sviluppati i package *Models* e *View*. In particolare abbiamo aggiunto all'interno del package *Models* in seguenti moduli:

- ApplicationUser
- PACContext
- Events
- ErrorViewModel
- Deadline
- Course
- EventViewModel
- Lesson

Invece, nel package *View* sono stati aggiunti i seguenti pacchetti:

- Events
- Shared

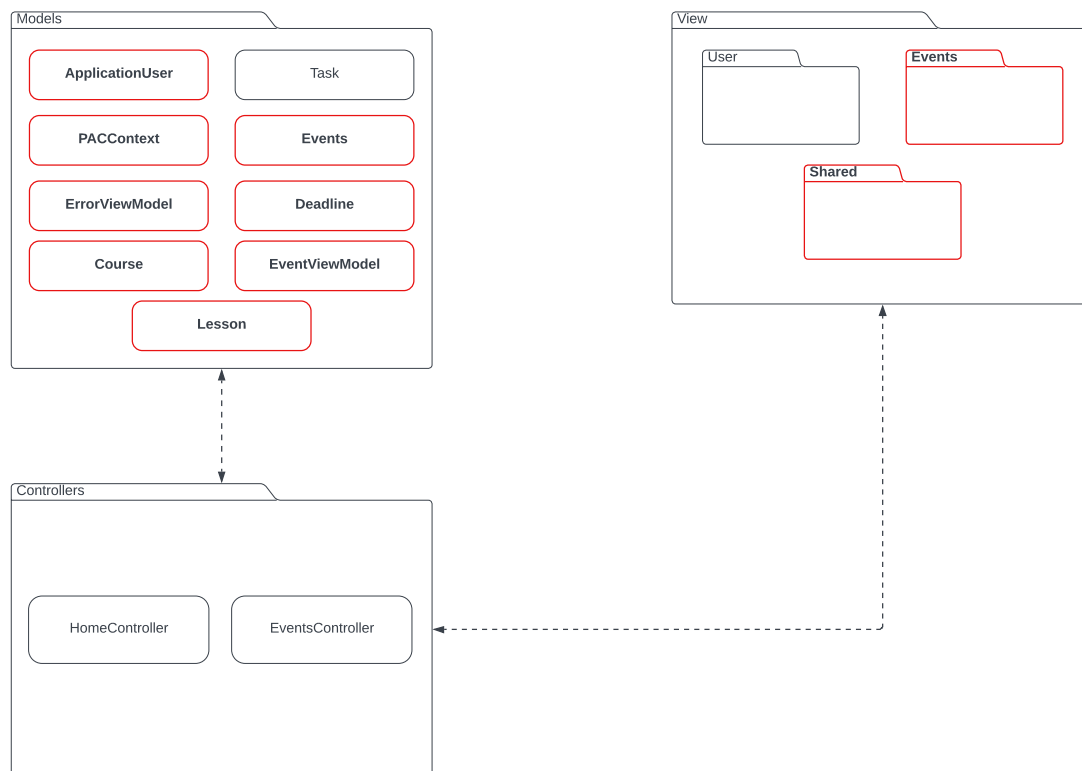


Figura 2.5: Diagramma dei package

2.8 Analisi statica del codice

Di seguito si riportano le metriche di qualità del codice fornite dallo strumento integrato di Visual Studio.

Gerarchia	Indice di manutenibilità	Complessità cicломatica	Profondità dell'ereditarietà	Accoppiamenti di classi	Righe di codice sorgente	Righe di codice eseguibile
StudyMate (Debug)	79	316	6	154	3.505	842
StudyMate.Data	100	1	6	3	10	0
StudyMate	50	3	1	40	57	24
StudyMate.Data.Migrations	37	4	2	32	751	208
StudyMate.Migrations	43	4	2	32	482	129
StudyMate.Services	84	11	1	16	88	30
StudyMate.Controllers	85	17	3	26	154	32
StudyMate.Models	95	110	2	27	240	70
AspNetCore	81	166	4	48	1.723	349

Figura 2.6: Metriche di qualità del codice

Iterazione 3

3.1 Descrizione

In questa iterazione si sono progettati e sviluppati i casi d'uso relativi alla gestione degli impegni personali e dei corsi di laurea a cui l'utente è iscritto (UC5.1, UC5.2, UC5.3, UC6.1).

3.2 Casi d'uso

Durante questa specifica fase del progetto, si sono dedicati gli sforzi per progettare e sviluppare i casi d'uso pertinenti alla gestione degli impegni personali (UC5.1, UC5.2, UC5.3) e alla selezione tramite API del profilo di laurea dell'utente (UC6.1). Alla fine di questa fase, gli utenti potranno gestire con facilità i propri impegni personali una volta effettuato l'accesso al sistema mediante l'autenticazione, che darà loro l'autorizzazione per accedere alle aree di gestione appositamente predisposte.

UC5.1	Inserimento impegni personali
<i>Descrizione:</i>	Inserimento degli impegni personali dell'utente
<i>Attori:</i>	Utente
<i>Precondizioni:</i>	L'utente è autenticato
<i>Flusso eventi:</i>	

1. L'utente inserisce la descrizione dell'evento
 2. L'utente inserisce l'orario dell'evento
 3. L'utente inserisce la cadenza dell'evento
 4. L'utente inserisce la priorità dell'evento
- Postcondizioni:* Reindirizzamento al calendario
- Eccezioni:* Sovrapposizione di un evento a priorità minore su un evento a priorità più alta
-

UC5.2	Modifica impegni personali
--------------	-----------------------------------

Descrizione: Modifica degli impegni personali dell'utente

Attori: Utente

Precondizioni:

1. L'utente è autenticato
 2. L'utente ha già inserito degli eventi
- Flusso eventi:*

1. L'utente modifica la descrizione dell'evento
 2. L'utente modifica l'orario dell'evento
 3. L'utente modifica la cadenza dell'evento
 4. L'utente modifica la priorità dell'evento
- Postcondizioni:* Reindirizzamento al calendario
- Eccezioni:* Sovrapposizione di un evento a priorità minore su un evento a priorità più alta
-

UC5.3	Eliminazione impegni personali
--------------	---------------------------------------

Descrizione: Eliminazione degli impegni personali dell'utente

Attori: Utente

Precondizioni:

1. L'utente è autenticato

2. L'utente ha già inserito degli eventi

Flusso eventi:

1. L'utente cancella l'evento

Postcondizioni: Reindirizzamento al calendario

Eccezioni: -

UC6.1	Scelta del corso di laurea
-------	----------------------------

Descrizione: Inserimento del corso di laurea a cui l'utente è iscritto

Attori: Utente

Precondizioni: L'utente è autenticato

Flusso eventi:

1. L'utente sceglie il corso di laurea

2. L'utente sceglie i corsi a scelta

Postcondizioni: Reindirizzamento al calendario

Eccezioni: -

3.3 Diagramma entità relazioni (ER)

Nel seguito viene mostrato il diagramma entità relazioni ottenuto nell'iterazione 3.

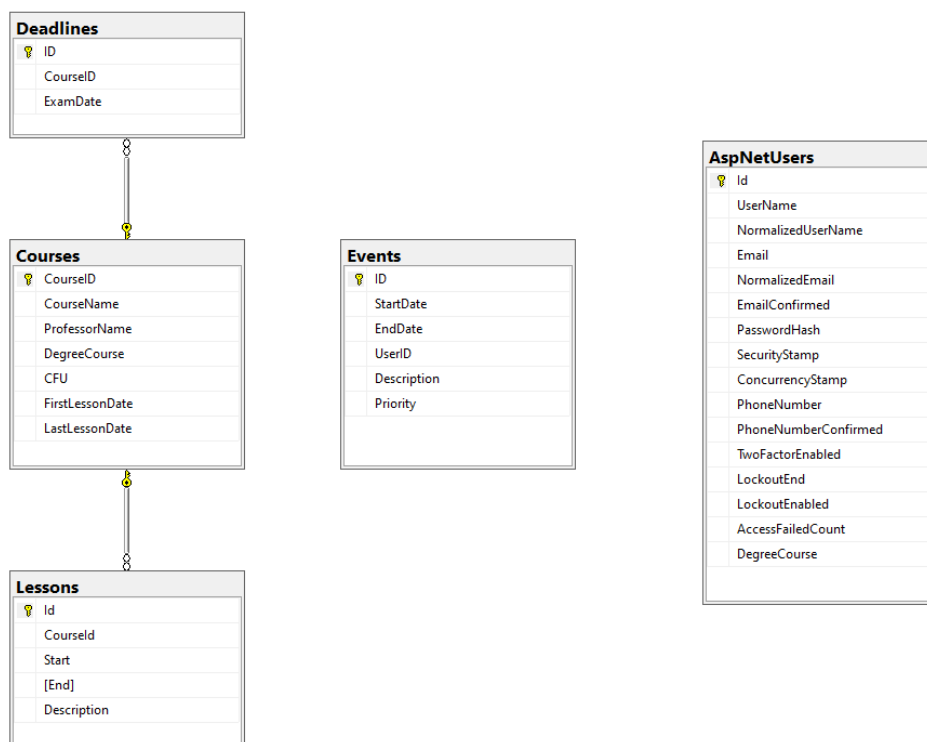


Figura 3.1: Diagramma entità relazioni

3.4 Diagramma delle classi (interfacce)

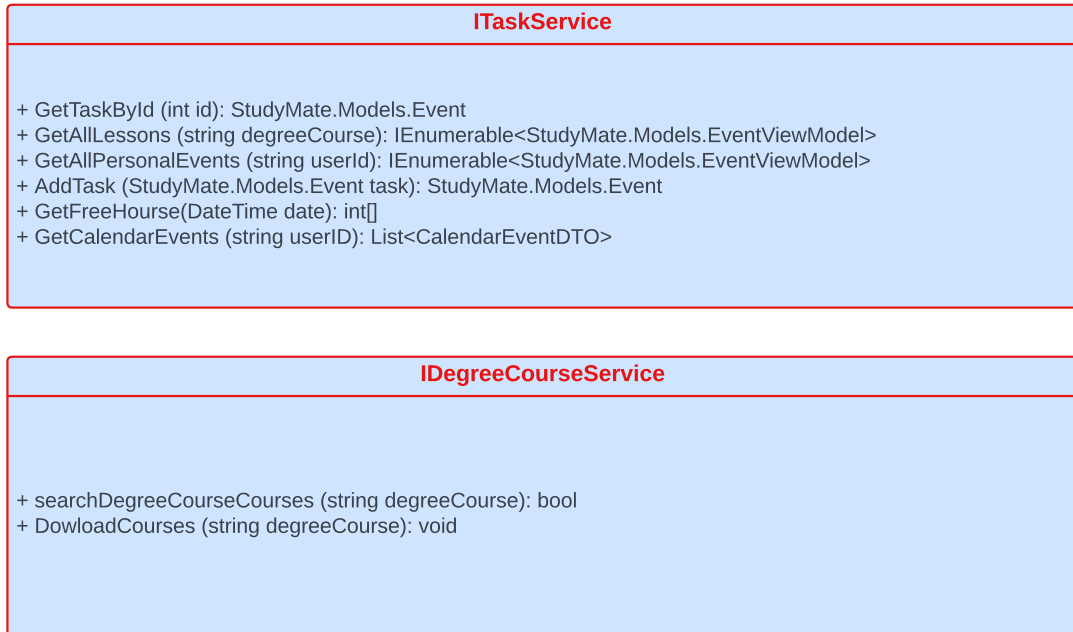


Figura 3.2: Diagramma delle classi (interfacce)

In questa iterazione, le interfacce *ITaskManagement* e *ITaskScheduling* sono state unite andando a formare un'unica interfaccia chiamata *ITaskService*.

È stata creata una nuova interfaccia chiamata *IDegreeCourseService*, che si occupa di prelevare dal database corsi specifici in base alle richieste.

3.5 Diagramma dei package

In questa iterazione si è resa necessaria l'aggiunta di un nuovo package chiamata *Services* con al suo interno i seguenti moduli:

- ITaskService
- TaskService
- IDegreeCourseService
- DegreeCourseService

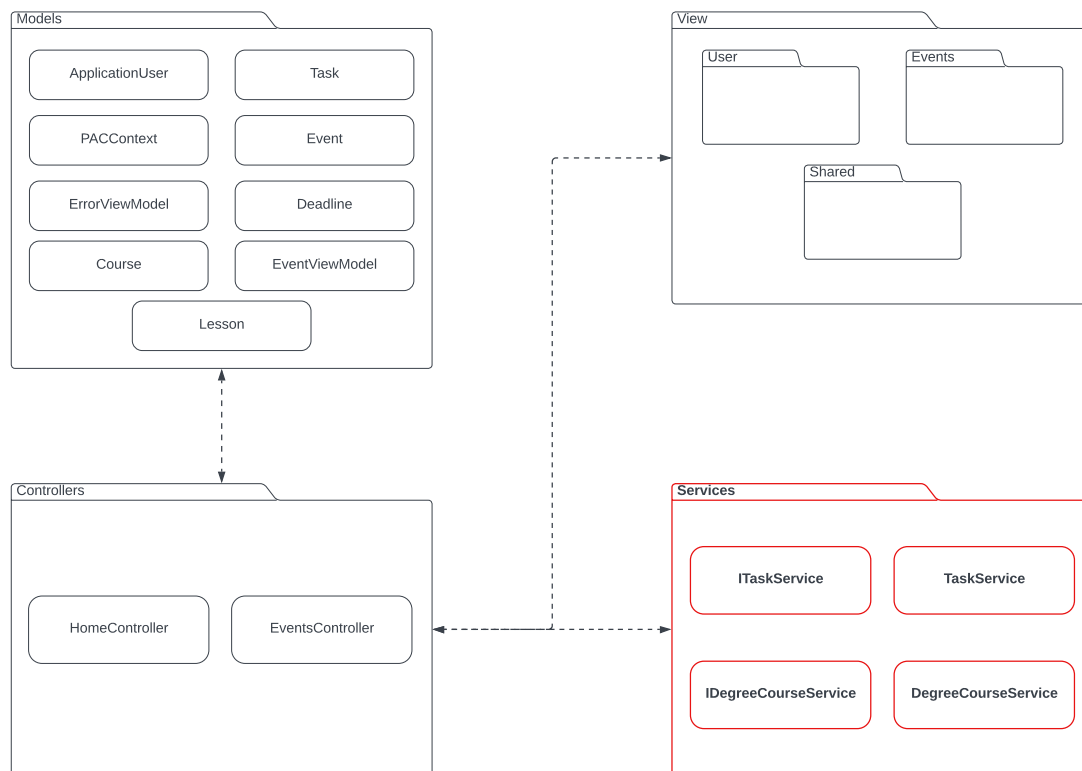


Figura 3.3: Diagramma dei package

3.6 API

In questa iterazione abbiamo sviluppato un API per simulare un servizio offerto dall'Università di Bergamo relativo alla fornitura dei corsi di laurea. Questa API permette agli sviluppatori di accedere a informazioni sui corsi di laurea dell'Università di Bergamo tramite richieste HTTP e di ottenere i risultati in formato JSON. Grazie a questa API, gli sviluppatori saranno in grado di inviare una richiesta specificando il corso di laurea desiderato come input e ricevere una risposta contenente i dettagli pertinenti in formato JSON. Nel seguito forniamo una panoramica completa delle funzionalità offerte dall'API, comprese le specifiche delle richieste, i parametri accettati e il formato dei dati restituiti.

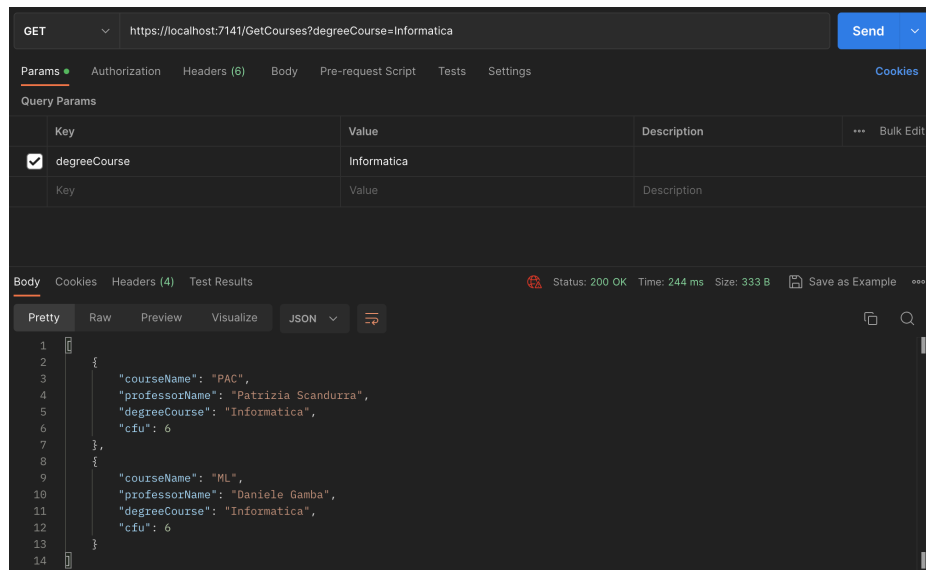


Figura 3.4: Test API Get Courses

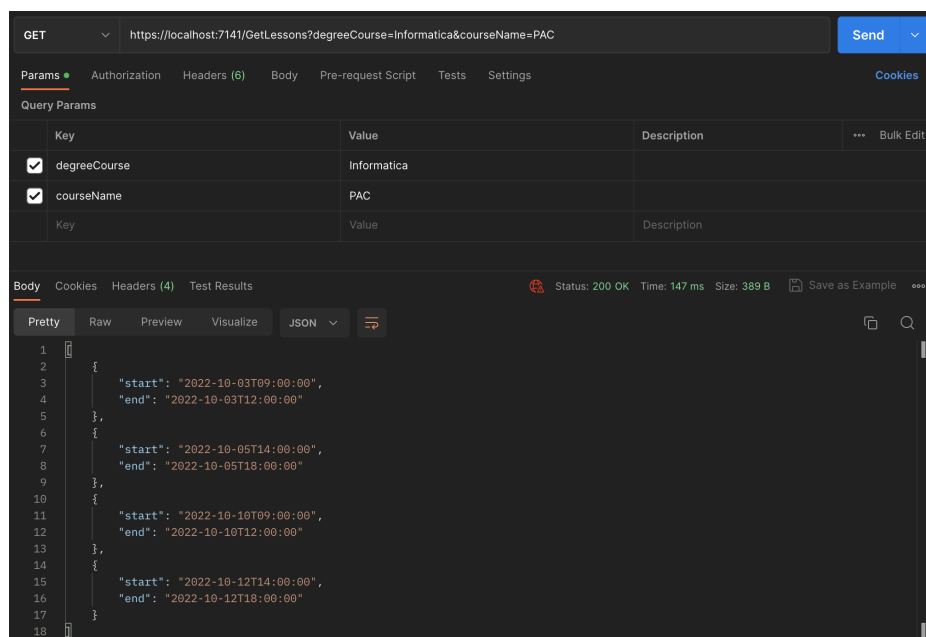


Figura 3.5: Test API Get Lessons

Nel seguito entriamo nel dettaglio delle chiamate sopra riportate.

3.6.1 Recupero dei corsi - GET Courses

Viene mostrato nel dettaglio una richiesta GET per ottenere i corsi relativi ad un indirizzo di studi.

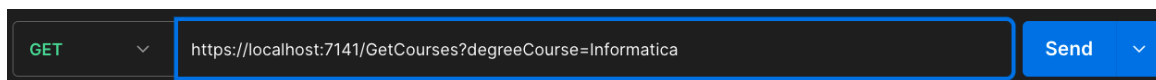


Figura 3.6: GET Courses

La risposta che si ottiene da questa chiamata è del tipo:

```
1  [
2    {
3      "courseName": "PAC",
4      "professorName": "Patrizia Scandurra",
5      "degreeCourse": "Informatica",
6      "cfu": 6
7    },
8    {
9      "courseName": "ML",
10     "professorName": "Daniele Gamba",
11     "degreeCourse": "Informatica",
12     "cfu": 6
13   }
14 ]
```

Figura 3.7: Risposta GET Courses

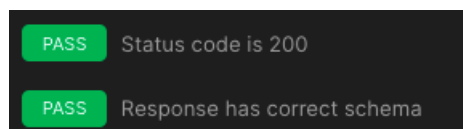


Figura 3.8: Verifica correttezza risposta GET Courses

Mostriamo nel seguito un esempio di chiamata non corretta nella quale diamo come dato di ingresso un indirizzo di studi non presente in memoria.

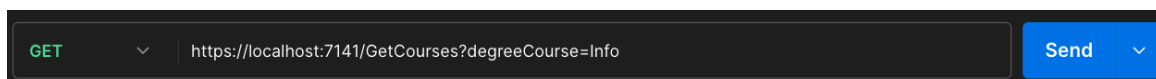


Figura 3.9: GET Courses errata

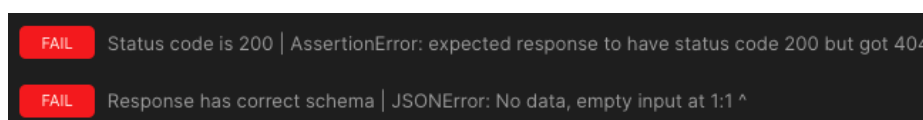


Figura 3.10: Verifica correttezza risposta GET Courses con chiamata errata

3.6.2 Recupero lezioni - GET Lessons

Viene mostrato nel dettaglio una richiesta GET per ottenere le lezioni relative ad un singolo corso.

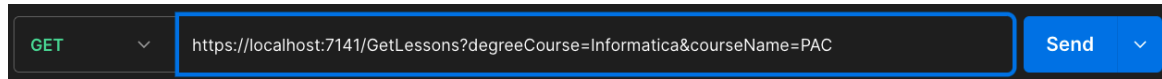


Figura 3.11: GET Lessons

La risposta che si ottiene da questa chiamata è del tipo:



Figura 3.12: Risposta GET Lessons

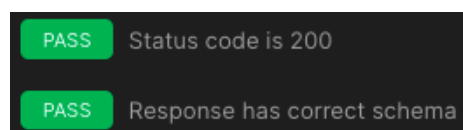


Figura 3.13: Verifica correttezza risposta GET Lessons

Mostriamo nel seguito un esempio di chiamata non corretta nella quale chiediamo le lezioni di un corso che non è presente in memoria.

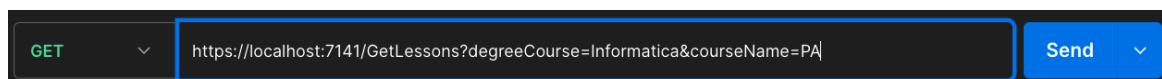


Figura 3.14: GET Lessons errata

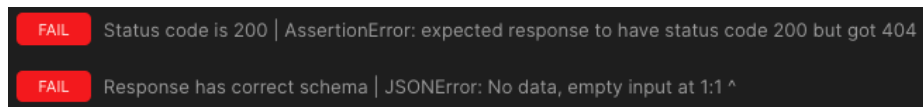


Figura 3.15: Verifica correttezza risposta GET Lessons con chiamata errata

3.7 Analisi statica del codice

Di seguito si riportano le metriche di qualità del codice fornite dallo strumento integrato di Visual Studio includendo le modifiche dell'iterazione 3

Gerarchia	Indice di manutenibilità	Complessità ciclomat...	Profondità dell'ereditari...	Accoppiamenti di classi	Righe di codice sorgente	Righe di codice eseguibile
StudyMate (Debug)	81	466	6	204	4.813	1.110
StudyMate.Data	100	1	6	3	10	0
StudyMate	49	3	1	42	61	26
StudyMate.Data.Migrations	37	4	2	32	751	208
StudyMate.Migrations	43	4	2	32	482	129
StudyMate.Controllers	85	17	3	26	154	32
StudyMate.Services	87	18	1	22	143	41
StudyMate.Areas.Identity.Pages.Account	81	47	2	54	302	68
StudyMate.Models	95	122	3	34	271	63
AspNetCore	82	250	4	56	2.639	543

Figura 3.16: Metriche di qualità del codice

Iterazione 4

4.1 Descrizione

In questa iterazione è stato sviluppato il caso d'uso relativo alla creazione del calendario (UC7).

4.2 Casi d'uso

4.2.1 UC7 - Generazione del calendario

UC7	Generazione del calendario
<i>Descrizione:</i>	Visualizzazione del calendario aggiornato
<i>Attori:</i>	Utente
<i>Precondizioni:</i>	
	1. L'utente è autenticato
<i>Flusso eventi:</i>	
	1. L'utente accede al suo account
	2. Viene caricata la pagina contenente il calendario
<i>Postcondizioni:</i>	Reindirizzamento al calendario
<i>Eccezioni:</i>	-

4.3 Diagramma entità relazioni (ER)

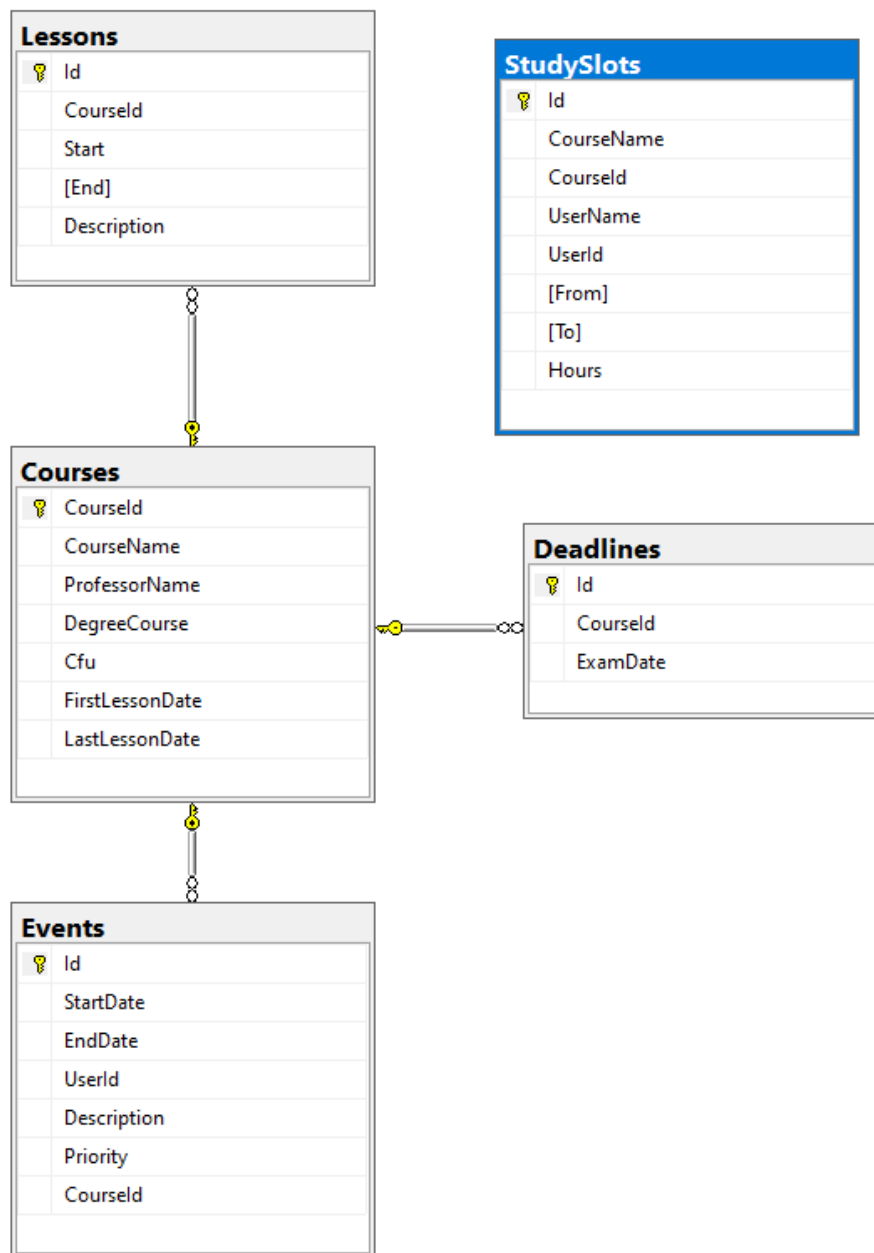


Figura 4.1: Diagramma entità relazioni

4.4 Diagramma delle classi

In questa iterazione è stata aggiunta la classe StudySlot.

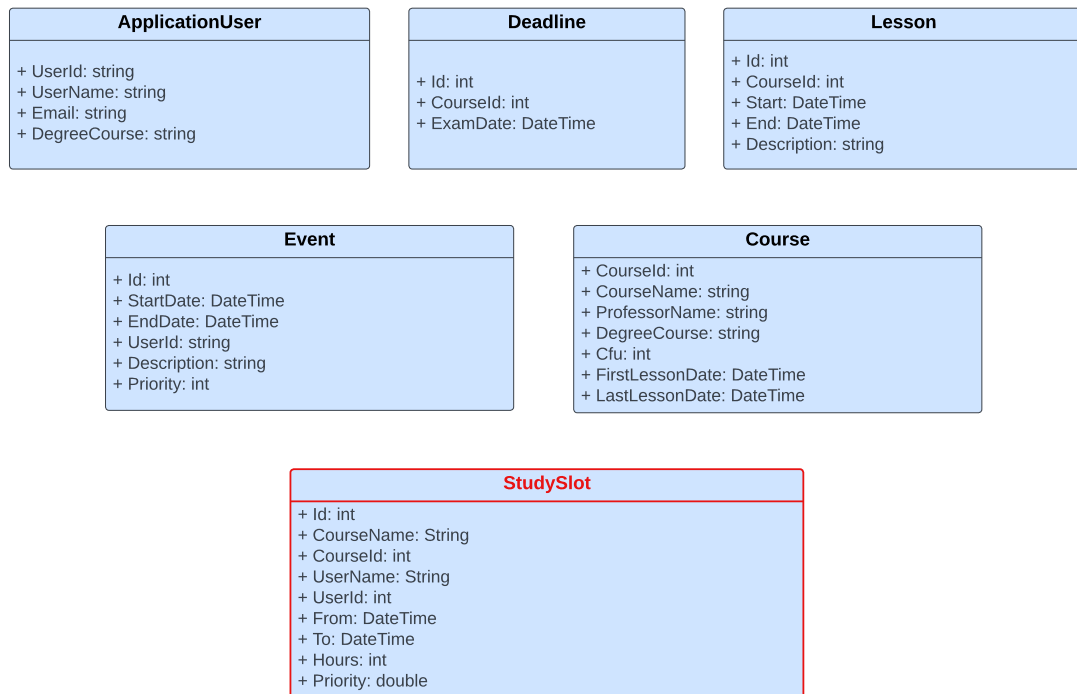


Figura 4.2: Diagramma delle classi (tipi di dati)

4.5 Diagramma delle interfacce

Lo sviluppo dell'algoritmo greedy ha reso necessaria l'implementazione di una nuova interfaccia. Mostriamo nel seguito l'aggiunta eseguita nella seguente iterazione.

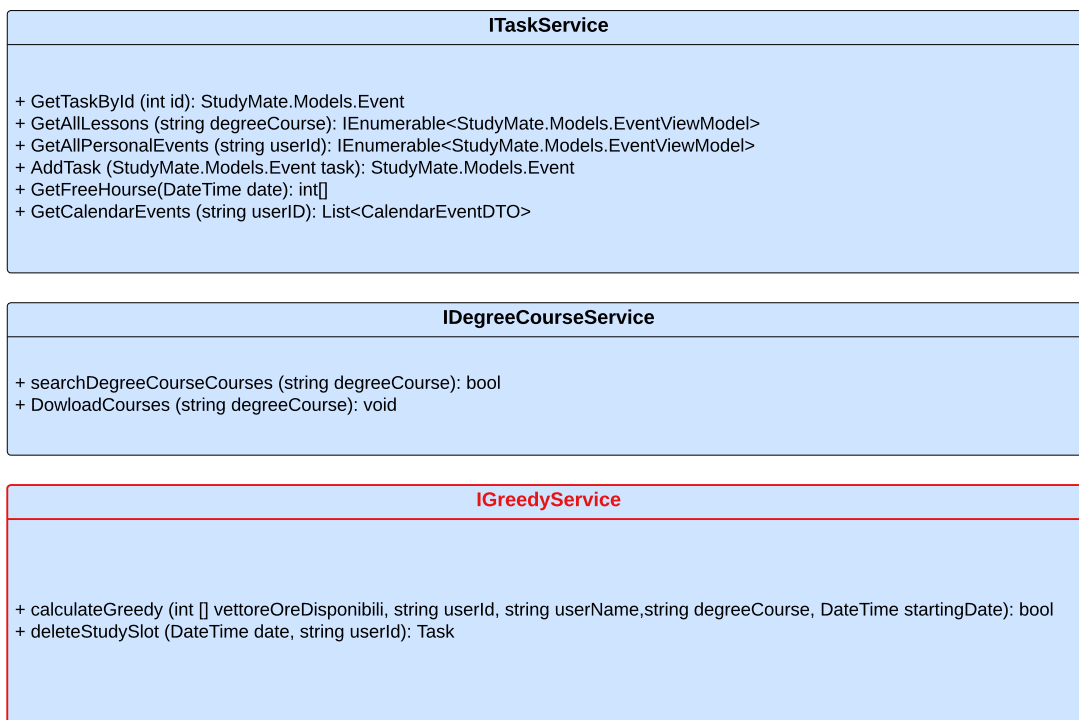


Figura 4.3: Diagramma delle classi (interfacce)

4.6 Diagramma dei package

Nel seguito viene mostrato come è stato modificato il diagramma dei package al seguito dello sviluppo effettuato nella seguente iterazione.

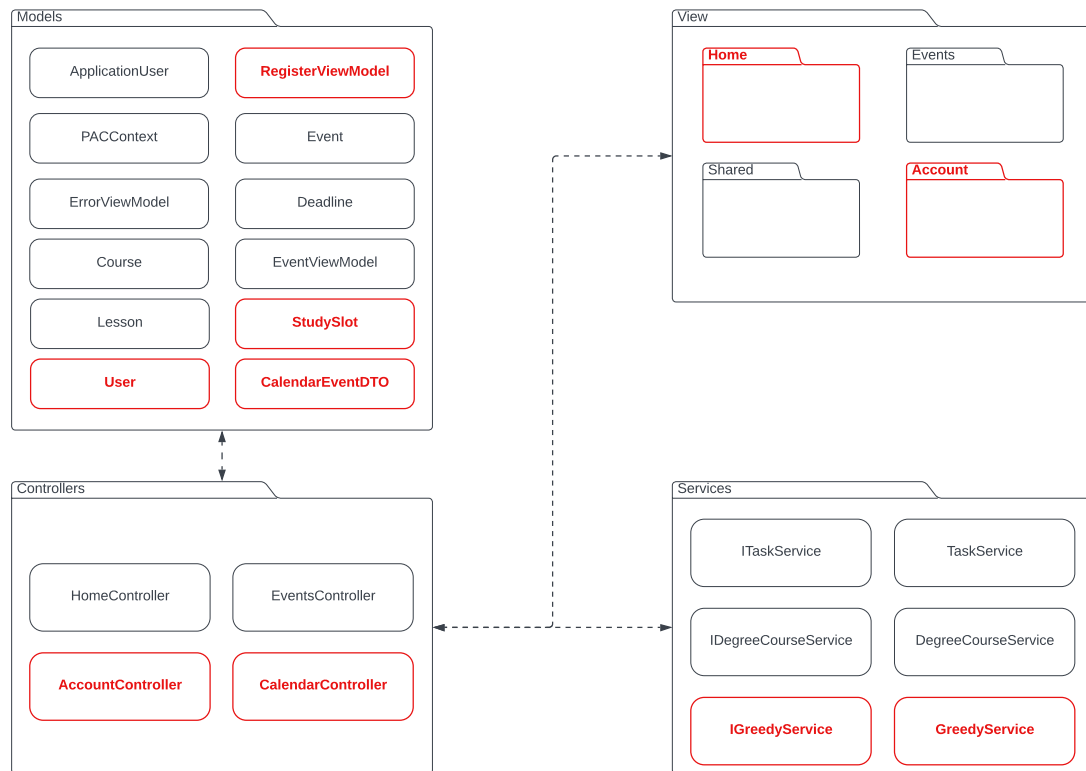


Figura 4.4: Diagramma dei package

4.7 Algoritmo di generazione del calendario

L'obiettivo del nostro algoritmo è quello di schedare le attività di studio di uno studente nell'arco di una settimana (7 giorni). L'algoritmo deve tenere conto, oltre che agli orari di riposo (dalle 23.00 alle 7.00) e agli orari di rifocillamento (12.00-14.00 e 19.00-21.00), anche delle lezioni frequentate dallo studente nelle quali non è possibile sovrapporre eventi. Si è deciso di gestire l'algoritmo di schedulazione con un approccio Greedy. Per semplicità si è deciso di operare esclusivamente con orari interi della durata minima di 1 ora (es. 9.00, 12.00 e non 9.36, 14.53 etc...). Mostriamo nel seguito lo pseudocodice dell'algoritmo implementato.

4.7.1 Funzione Greedy

Algoritmo 1 Pseudocodice funzione CalculateGreedy \rightarrow bool

Require: vettore delle ore disponibili, identificativo dell'utente e del corso di studi da lui frequentato, data di esecuzione dell'algoritmo.

```

1: candidati  $\leftarrow$  recuperiamo la lista dei candidati con l'apposita funzione SearchCandidates.
2: OreLibere  $\leftarrow$  recuperiamo il dizionario delle ore libere con la funzione SearchFreeHours.
3: soluzione  $\leftarrow$  lista contenente la soluzione inizialmente inizializzata vuota.
4: while non ho soluzione Ottimo AND ho ancora candidati da sottoporre do
5:   sl  $\leftarrow$  prendo il primo candidato dalla lista dei candidati
6:   tolgo dalla lista dei candidati il primo elemento
7:   if sl è Ammissibile then
8:     soluzione  $\leftarrow$  aggiungo sl a soluzione
9:   end if
10: end while
11: if ho raggiunto Ottimo then
12:   Restituisco True
13: else
14:   Restituisco False
15: end if

```

L'algoritmo greedy sopra riportato implementa delle chiamate a diverse funzioni non esplicitate. Le funzioni utilizzate sono:

- *SearchCandidates*
- *SearchFreeHours*
- *Ottimo*
- *Ammissibile*

Queste ultime vengono esplicitate nel seguito.

4.7.2 Funzione SearchCandidates

La funzione *SearchCandidates* restituisce una lista di candidati da sottoporre all'algoritmo greedy. Ogni candidato corrisponde ad uno slot temporale da dedicare allo studio di 3 o 1 ora corrispondente ad un determinato corso (es. PAC-3ore, TIT-3ore, AI-1ora, PAC-1ora etc...). La generazione della lista di candidati avviene secondo una serie di condizioni imposte in fase di analisi. Per ogni corso, sulla base dei suoi CFU, calcoliamo un monte ore di studio da dedicargli. Per ogni corso dividiamo il corrispettivo monte ore in slot temporali da 3 ore e da 1 ora. Calcoliamo una priorità basata sulla distanza tra la data attuale (ovvero quella nella quale viene eseguito l'algoritmo) e la data dell'esame del corso (deadline). Decidiamo inoltre di utilizzare come secondo criterio di priorità la durata dello slot prediligendo gli slot temporali dalla durata maggiore rispetto a quelli da 1 ora. Presenteremo infine in modo casuale (pur mantenendo l'ordine in base sulla base della durata dello slot temporale) i candidati all'algoritmo greedy.

Algoritmo 2 Pseudocodice funzione SearchCandidates → list

Require: string degreeCourse, string userName, string userId

- 1: *courses* ← preleva dal DB *Courses* i corsi relativi ad un determinato DegreeCourse
- 2: *studySlots* ← new empty List // Inizializzare una lista vuota di slot di studio
- 3: **for each** *course* **in** *courses* **do**
- 4: *alreadyStudyHours* ← ore di studio complessive già effettuate da un dato utente
- 5: *calc* ← (OreStudioNecessarie - *alreadyStudyHours*) / 12 // Inizializzare una lista vuota di slot di studio
- 6: *priority* ← 0

```

7:   threeHoursSlots ← ⌊OreStudioNecessarie/3⌋ // Numero di slots da 3 ore ap-
    prossimato per difetto
8:   oneHourSlots ← OreStudioNecessarie - 3*threeHoursSlots // Numero di slots
    da 1 ora
9:   deadline ← giorno dell'esame relativo ad uno specifico corso
10:  remainingDays ← deadline - Today
11:  priority ← (1/remainingDays) + ((Cfu * 25) - 100) * 1.1
12:  for index = 1, 2, ..., threeHoursSlots do
13:    sl ← crea nuova oggetto StudySlot e ne compilo i campi // Riempio tutti
    gli slots da 3 ore disponibili
14:  end for
15:  for index = 1, 2, ..., oneHoursSlots do // Riempio tutti gli slots da 1 ora
    disponibili
16:    sl ← crea nuova oggetto StudySlot e ne compilo i campi
17:  end for
18: end for
19: Shuffle(studySlots)
20: ordinaDecrescente(Hours) // Ordinamento in ordine decrescente delle ore, in .NET5
    viene utilizzato il Quicksort avente complessità media  $O(N \cdot \log N)$ 
21: return studySlots

```

4.7.3 Funzione SearchFreeHours

La funzione *SearchFreeHours* restituisce un dizionario contenente gli slot temporali liberi in cui è possibile inserire nuovi candidati. In ingresso riceve un vettore contenente l'informazione relativa agli impegni preesistenti di un utente. La funzione, una volta individuato uno slot libero, calcola il numero di ore libere immediatamente successive alla prima che è stata individuata. Viene salvata come chiave del dizionario un intero corrispondente all'orario di inizio di uno slot libero e come valore il numero di slot liberi ad esso direttamente connessi (es. [key:9,value:3] indica che alle 9.00 è presente uno slot della durata di 3 ore)

Algoritmo 3 Pseudocodice funzione SearchFreeHours → dizionario

Require: array[] *vettoreSettimana* // É un vettore contenente 0 (slot occupato) ed 1 (slot libero)

- 1: *dizionario* \leftarrow new empty Dictionary< *int*, *int* > // Dizionario per memorizzare le posizioni e le durate degli slot di tempo liberi.
- 2: *lunghezza* \leftarrow lunghezza del vettore passato
- 3: *posizione* \leftarrow -1 // Usato per tenere traccia della posizione iniziale di uno slot libero
- 4: *count* \leftarrow 0 // Indica la durata dello slot temporale libero
- 5: **for** *index* = 1, 2, ..., *lunghezza* **do**
- 6: **if** (*vettore*[*index*] == 1) **then**
- 7: **if** (*posizione* == -1) **then**
- 8: *posizione* \leftarrow *index*
- 9: **end if**
- 10: *count* \leftarrow *count* + 1
- 11: **else if** (*count* > 0) **then**
- 12: *dizionario*[*posizione*] \leftarrow *count*
- 13: *posizione* \leftarrow -1
- 14: *count* \leftarrow 0 // viene reimpostato a 0 per iniziare a contare la durata di un nuovo slot libero
- 15: **end if**
- 16: **end for**
- 17: **if** (*count* > 0) **then** // controllo finale per verificare se l'ultimo elemento del vettore è uno slot libero
- 18: *dizionario*[*posizione*] \leftarrow *count*
- 19: **end if**
- return** *dizionario*;

4.7.4 Funzione Ottimo

La funzione *Ottimo* restituisce vero se l'algoritmo greedy è stato in grado di riempire completamente tutti gli slot liberi, falso altrimenti.

Algoritmo 4 Pseudocodice funzione Ottimo \rightarrow bool

Require: Dictionary< *int*, *int* > freeHours

```

1: sum ← 0
2: for each kvp in freeHours do // kvp è una chiave del dizionario freeHours
3:   sum ← sum + kvp.Value // kvp.Value è il valore contenuto nel dizionario
4: end for
5: if sum == 0 then return true;
6: end if return false;

```

4.7.5 Funzione Ammissibile

La funzione *Ammissibile* verifica se un candidato può essere inserito nel primo slot di tempo disponibile.

Algoritmo 5 Pseudocodice funzione Ammissibile → **bool**

Require: StudySlot *sl*, Dictionary<int, int> freeHours, DateTime *startingDate*

```

1: for each kvp in freeHours do
2:   if (kvp.Value >= sl.Hours) then
3:     freeHours[kvp.Key] ← kvp.value - sl.Hours // Aggiorno le ore disponibili
        nel dizionario
4:     hr ← 0
5:     day ← 0
6:     if kvp.Key < 24 then // Significa che ci troviamo ancora al primo giorno
7:       hr ← kvp.Key
8:       day ← 0
9:     else
10:      number ← kvp.Key/24
11:      day ← [number] // Approssima per difetto
12:      hr ← kvp.Key mod 24
13:    end if
14:    sl.From ← startingDate.AddDays(day)
15:    TimeSpan ts ← new TimeSpan(hr + kvp.Value - sl.Hours, 0, 0)
16:    sl.From ← sl.From.Date + ts
17:    sl.To ← startingDate.AddDays(day)
18:    ts ← new TimeSpan(hr + kvp.Value, 0, 0)

```

```

19:         sl.To ← sl.To.Date + ts
20:         Inserimento degli studySlot nel database
           return true
21:     end if
22: end for return false

```

4.8 Analisi della complessità

Per procedere al calcolo della complessità dell'algoritmo greedy è necessario calcolare prima la complessità delle funzioni richiamate dall'algoritmo.

Iniziamo a calcolare la complessità della funzione *SearchCandidates*. Procederemo poi all'analisi delle funzioni *SearchFreeHours*, *Ottimo* e *Ammissibile*. Uniremo infine i risultati per ottenere il costo computazionale dell'algoritmo greedy implementato (*Divede et Impera*).

4.8.1 Complessità della funzione *SearchCandidates*

Procediamo all'analisi della funzione.

1. Il primo passo eseguito dalla funzione è il recupero di tutti i corsi relativi al corso di studi dello studente. Ogni corso recuperato viene memorizzato in una lista *courses* che li contiene tutti. Indichiamo con n il numero di corsi recuperati.
2. Per ogni corso andiamo in un primo momento ad eseguire una serie di operazioni elementari dal costo $\Theta(1)$. Successivamente, si presentano due cicli *for* all'interno del *foreach*. Il loro costo non è unitario. Questi cicli vengono ripetuti per il numero di slot (k) calcolato per ogni corso. In generale abbiamo che $n \ll k$, quindi considerando solo i cicli interni nel calcolo della complessità diventa $\Theta(n * k) \sim \Theta(k)$.
3. La funzione *Shuffle* disordina la lista *StudySlot*. Il suo costo computazione è proporzionale alla dimensione totale della lista, ovvero $\Theta(n * k) \sim \Theta(k)$. Il suo costo computazionale rimane $\Theta(k)$.
4. Abbiamo infine la funzione *OrderByDescending* che da documentazione ha costo computazionale $\Theta(k * \log k)$

La complessità complessiva della funzione *SearchCandidates* risulta quindi essere $\Theta(k * \log k)$ con k il numero dei candidati generati.

4.8.2 Complessità della funzione *SearchFreeHours*

La complessità dell'algoritmo dipende dalla lunghezza del vettore *vettoreSettimana*. Essa è, appunto, $\Theta(s)$, con s la lunghezza del vettore ore della settimana, in quanto le operazioni all'interno del ciclo *for* richiedono un costo unitario.

4.8.3 Complessità della funzione *Ottimo*

La complessità della funzione *Ottimo* ha complessità $\Theta(i)$ con i numero di ore libere rimaste nella settimana.

4.8.4 Complessità della funzione *Ammissibile*

La complessità della funzione *Ammissibile* ha complessità $\Theta(i)$ con i numero di ore libere rimaste nella settimana.

4.8.5 Complessità della funzione *Greedy*

Uniamo ora le complessità calcolate precedentemente per calcolare la complessità complessiva dell'algoritmo greedy.

1. La chiamata alla funzione *SearchCandidates* abbiamo visto che ha costo computazionale $\Theta(k * \log k)$
2. La chiamata alla funzione *SearchFreeHours* abbiamo visto che ha costo computazionale $\Theta(s)$
3. Il ciclo *While* viene ripetuto fino a quando, nel caso peggiore, il numero dei candidati non diventa zero, ovvero per k volte. Al suo interno viene eseguita una funzione (*RemoveAt*) che da documentazione ha costo computazionale $\Theta(k)$ con k il numero di candidati. Viene inoltre chiamata la funzione *Ammissibile* che, come abbiamo visto, ha costo computazionale $\Theta(i)$. Assumiamo che $k \gg i$ siccome i rappresenta il numero di ore libere che ho all'interno di una settimana

e quindi è limitata al più ad essere $24 * 7$. Il costo computazionale del ciclo risulta essere $\Theta(k^2)$.

- Viene infine eseguita la funzione ottimo che come abbiamo visto sopra ha costo computazionale $\Theta(i)$.

Possiamo allora concludere che il costo computazionale dell'algoritmo greedy è $\Theta(k^2)$.

4.9 Analisi statica del codice

Di seguito si riportano le metriche di qualità del codice fornite dallo strumento integrato di Visual Studio includendo le modifiche dell'iterazione 4

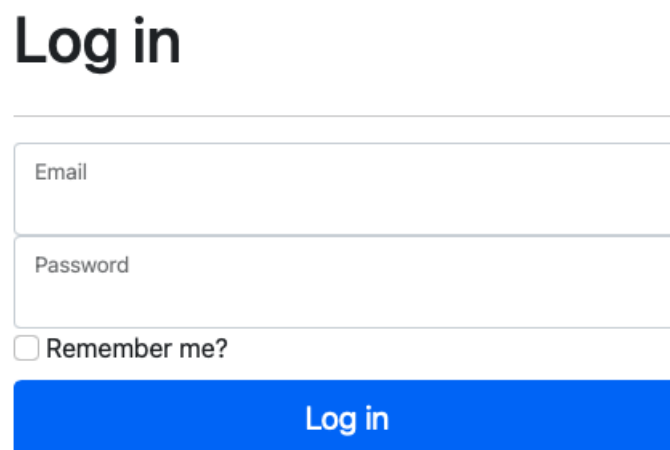
Gerarchia	Indice di manutenibilità	Complessità cicloma...	Profondità dell'ereditari...	Accoppiamenti di classi	Righe di codice sorgente	Righe di codice eseguibile
CourseLessons (Debug)	78	30	2	39	124	43
StudyMate (Debug)	81	596	6	213	5.965	1.389
StudyMate.Data	100	1	6	3	10	0
StudyMate	49	3	1	42	62	27
StudyMate.Data.Migrations	37	4	2	32	751	208
StudyMate.Migrations	43	4	2	32	482	129
StudyMate.Controllers	76	33	3	43	311	85
StudyMate.Areas.Identity.Pages.Account	81	47	2	54	302	68
StudyMate.Services	80	59	1	36	517	195
StudyMate.Models	98	136	3	26	298	21
AspNetCore	81	309	4	56	3.232	656

Figura 4.5: Metriche di qualità del codice

Manuale utente

La web-application, nonostante sia stata sviluppata con un framework di proprietà di Microsoft (.NET), è multiplatforma consentendo l'esecuzione su una vasta gamma di piattaforme, inclusi sistemi operativi come Windows, macOS, Linux e dispositivi mobili come smartphone e tablet indipendentemente dal sistema operativo utilizzato.

Nel seguito vengono mostrati una serie di screenshot rappresentativi del funzionamento dell'applicazione web creata.

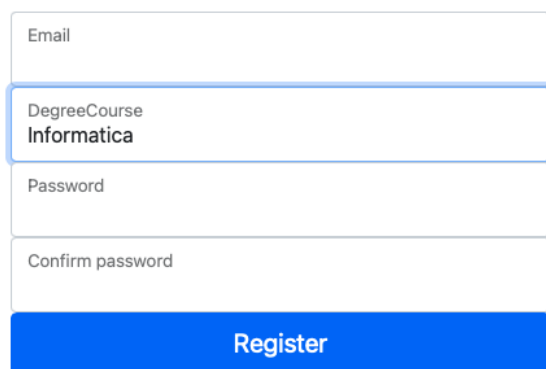


The screenshot displays a login interface. At the top, the text "Log in" is prominently displayed. Below this, there is a form with two input fields: "Email" and "Password". Under the "Password" field, there is a checkbox labeled "Remember me?". At the bottom of the form is a blue button with the text "Log in" in white.

Figura 5.1: Schermata Login

Register

Create a new account.

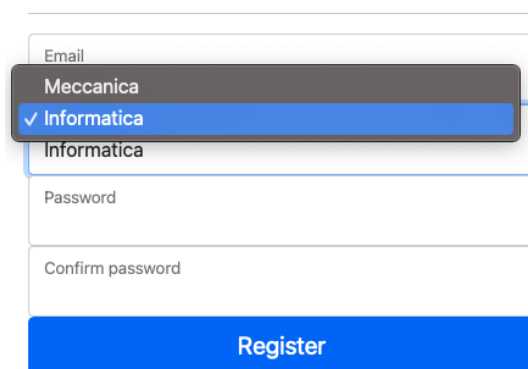


A screenshot of a registration form. It consists of four input fields stacked vertically: 'Email', 'DegreeCourse' (with 'Informatica' selected), 'Password', and 'Confirm password'. Below the fields is a blue button labeled 'Register'.

Figura 5.2: Schermata Register

Register

Create a new account.



A screenshot of the same registration form as in Figure 5.2, but with the 'DegreeCourse' dropdown menu open. The menu shows two options: 'Meccanica' and 'Informatica', with 'Informatica' selected and highlighted in blue. The 'Register' button remains at the bottom.

Figura 5.3: Dropdown scelta corso

StudyMate

Home

Calendar

Personal

Hello paoloolivieri1999@gmail.com! Logout

Calcola gli slot di studio

Today events list:

Elettronica

Start: 08:00:00

End: 10:00:00

PAC

Start: 10:00:00

End: 12:00:00

Elettronica

Start: 14:00:00

End: 17:00:00

PAC

Start: 17:00:00

End: 18:00:00

Elettronica

Start: 18:00:00

End: 19:00:00

Elettronica

Start: 21:00:00

End: 22:00:00

ML

Start: 22:00:00

End: 23:00:00

Figura 5.4: Schermata Home

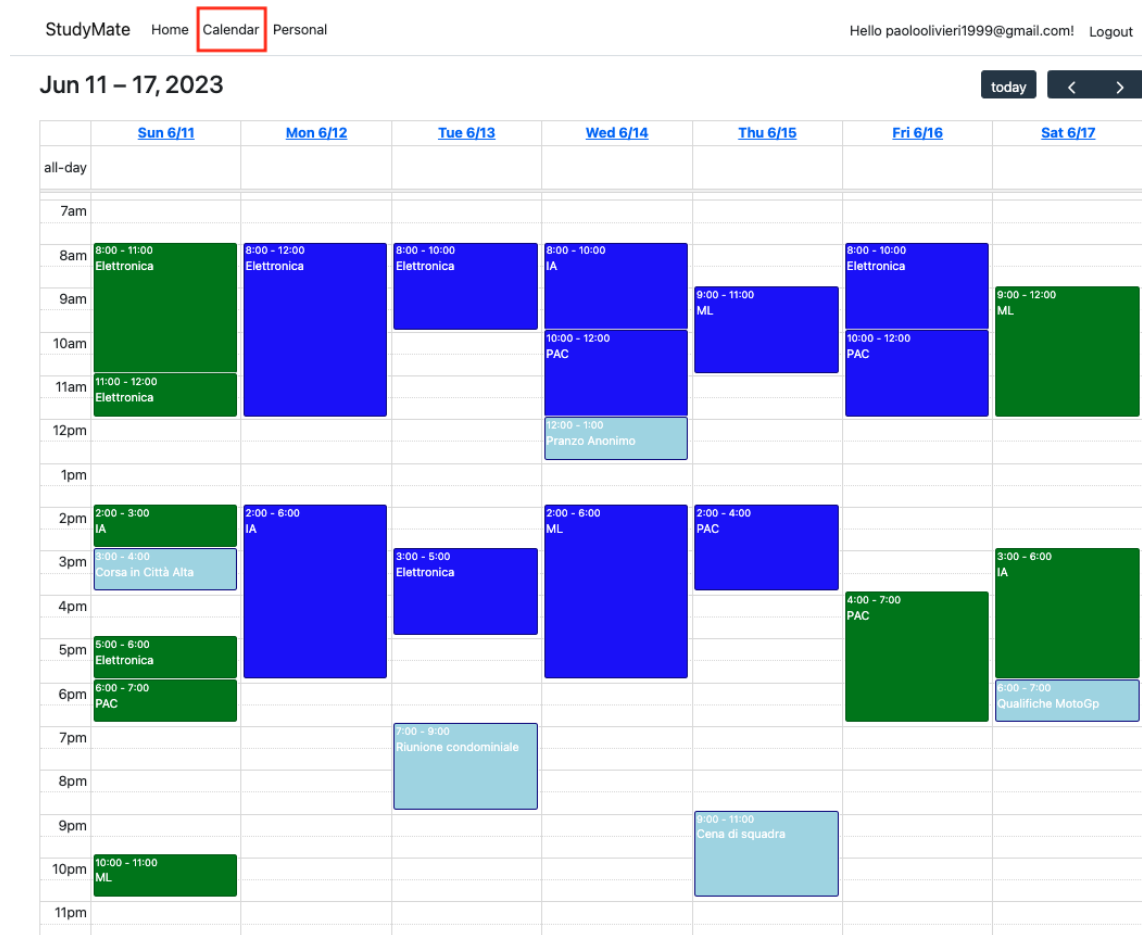


Figura 5.5: Schermata Calendar

Viene indicata la legenda dei colori utilizzati per gli eventi:

- Lezioni
- Study Slot
- Impegni personali
- Esami

StudyMate	Home	Calendar	Personal	Hello paoololivieri1999@gmail.com! Logout
<h2>Elenco impegni personali</h2> Create New				
Data	Ora inizio	Ora fine	Descrizione	
01/07/2023	11:00:00	12:00:00	Partita Calcetto	Edit Details Delete
01/07/2023	17:00:00	18:00:00	Corsa	Edit Details Delete
02/07/2023	17:00:00	18:00:00	Allenamento	Edit Details Delete
14/06/2023	12:00:00	13:00:00	Pranzo Anonimo	Edit Details Delete
11/06/2023	15:00:00	16:00:00	Corsa in Città Alta	Edit Details Delete
13/06/2023	19:00:00	21:00:00	Riunione condominiale	Edit Details Delete
17/06/2023	18:00:00	19:00:00	Qualifiche MotoGP	Edit Details Delete
15/06/2023	21:00:00	23:00:00	Cena di squadra	Edit Details Delete

Figura 5.6: Schermata Personal

Conclusioni

Sviluppi futuri

Non tutto il progetto è stato implementato.

Non è stata sviluppata la parte di progetto relativa ai seguenti use case:

- UC4.2 - Modifica degli orari di routine
- UC6.2 - Modifica corsi a scelta
- UC6.3 - Eliminazione corsi a scelta

Non sarà quindi possibile modificare gli orari di routine quali orari pranzo e cena, orari sonno etc... i quali vengono impostati di default dall'applicazione.

Non è stata implementata la possibilità di gestire i corsi a scelta di uno studente. Lo studente può selezionare solo il suo indirizzo di corso in fase di registrazione.

Inoltre abbiamo deciso di non permettere all'utente di selezionare le priorità dei singoli eventi. Abbiamo semplificato il tutto scegliendo di default di non sovrapporre mai nessun impegno generato dall'algoritmo. Il controllo sulla sovrapposizione di eventi viene eseguito solo per la parte relativa all'algoritmo, ovvero solo per la parte relativa all'assegnazione degli slot di studio. Per gli impegni personali e le lezioni non esiste un controllo specifico.

In generale quindi non viene impedita l'aggiunta di un impegno che si sovrappone ad un impegno già esistente. Eventualmente in fase di ricalcolo, se la sovrapposizione è su uno slot generato dall'algoritmo, viene data priorità agli impegni inseriti dall'utente.

Non sono state sviluppate API relative alla web-application. Tuttavia è stata sviluppata un API in modo da simulare un ipotetico servizio offerto dall'università di Bergamo il quale fornisce l'insieme dei corsi universitari relativi ad un particolare indirizzo di studi.

Per migliorare l'esperienza degli utenti, è possibile considerare l'integrazione del sistema di notifiche con i calendari cloud più comuni, come Google Calendar, Outlook Calendar o Apple Calendar. Ciò consentirebbe agli utenti di sincronizzare automaticamente gli eventi inseriti nella webapp Studymate con il proprio calendario preferito e ricevere notifiche attraverso tale calendario.

In un eventuale sviluppo futuro è possibile implementare quanto descritto sopra.