	SAMT – Sezione Informatica	Pagina 1 di 64
	TeamSync	

TeamSync



1	Introduzione	6
1.1	Informazioni sul progetto	6
1.2	Abstract	6
1.3	Scopo	6
2	Analisi	7
2.1	Analisi del dominio	7
2.2	Analisi e specifica dei requisiti	7
2.3	Use case	10
2.4	Pianificazione	11
2.4.1	Metodo di sviluppo	12
2.5	Analisi dei mezzi.....	13
2.5.1	Software	13
2.5.2	Hardware.....	13
3	Progettazione	14
3.1	Design dei dati e database.....	14
3.2	Struttura delle tabelle e motivazioni	15
3.2.1	Tabella utente	15
3.2.2	Tabella team	15
3.2.3	Tabella progetto	15
3.2.4	Tabella task_json	16
3.3	Relazioni fra tabelle.....	16
3.4	ORM (Object-Relational Mapping)	16
3.4.1	Sintassi.....	17
3.4.2	Relazioni	17
3.4.3	Inizializzazione database	18
3.5	Design delle interfacce	19
3.5.1	Dashboard user	19
3.5.2	Pannello admin (progetti).....	20
3.5.3	Pannello admin (team).....	20
3.5.4	Pannello admin (utenti)	21
3.5.5	Form di login	21
3.5.6	Form di registrazione	22
3.5.7	Nota design interfacce	22
3.6	Design procedurale	23
3.6.1	Utente user.....	23
3.6.2	Utente admin.....	24
4	Implementazione	25
4.1	Linguaggi e framework utilizzati	25
4.2	Struttura del progetto	25
4.3	Index.js	26
4.3.1	Configurazione del Template Engine (Handlebars).....	26
4.3.2	Configurazioni di Sicurezza	26
4.4	Security.js.....	27
4.4.1	Configurazione della sessione	27
4.4.2	Configurazioni Helmet.....	28
4.4.3	Configurazioni di Sicurezza	28
4.5	pageRoutes.js	29
4.5.1	Middleware di Autenticazione	29
4.5.2	Accesso alla Dashboard del Progetto.....	30
4.6	authController.js	31
4.6.1	Registrazione di un Nuovo Utente	31
4.6.2	Validazione della Password	32
4.6.3	Validazione dell'Email	33
4.7	taskController.js.....	34
4.7.1	Recupero delle Task di un Progetto	34
4.7.2	Eliminazione di una Task	35
4.7.3	Spostamento task	36
5	Test.....	37

5.1	Protocollo di test.....	37
5.2	Risultati test.....	41
5.2.1	Riassunto Test	47
5.3	Unit test	48
5.3.1	Configurazione	48
5.3.2	Sintassi.....	49
5.4	Risultati.....	49
5.4.1	utils.test.js	49
5.4.2	project.test.js	50
5.4.3	admin.test.js	50
5.4.4	users.test.js	50
5.4.5	modal.test.js	51
5.4.6	authController.test.js	51
5.4.7	taskController.test.js	52
5.4.8	Riassunto unit test	52
5.5	Mancanze e limitazioni conosciute.....	53
5.5.1	Mancanze.....	53
5.5.2	Limitazioni	53
5.6	Implementazioni extra	53
5.6.1	Gestione profilo utente.....	53
5.6.2	Invio delle email	54
5.6.3	Reset della password se dimenticata	55
5.6.4	Form di contatto	56
5.6.5	Calendario attività	57
6	Consuntivo.....	58
6.1	Confronto pianificazione preventiva e consuntiva.....	59
7	Conclusioni	60
7.1	Sviluppi futuri.....	60
7.2	Considerazioni personali	61
8	Glossario	62
9	Bibliografia	63
9.1	Sitografia	63
10	Allegati	63
11	Note	64

Indice delle figure:

Figura 1 - Use Case.....	10
Figura 2 - Gantt.....	11
Figura 3 - Metodo di sviluppo	12
Figura 4 - Diagramma ER.....	14
Figura 5 - Sintassi Sequelize	17
Figura 6 - Relazioni (ORM).....	17
Figura 7 - Inizializzazione DB	18
Figura 8 - Design dashboard user	19
Figura 9 - Design pannello Admin - progetti.....	20
Figura 10 - Design pannello Admin - team	20
Figura 11 – Design pannello Admin – utenti.....	21
Figura 12 - Design form Login	21
Figura 13 - Design form registrazione	22
Figura 14 - Diagramma di flusso user	23
Figura 15 - Diagramma di flusso admin.....	24
Figura 16 - Struttura progetto	25
Figura 17 - Configurazione Handlebars.....	26
Figura 18 - Configurazioni di sicurezza	26
Figura 19 - Configurazione sessione.....	27
Figura 20 - Configurazioni Helmet.....	28
Figura 21 - Configurazioni di sicurezza	28
Figura 22 - Middleware autenticazione.....	29
Figura 23 - Accesso alla Dashboard del Progetto	30
Figura 24 - Registrazione di un Nuovo Utente	31
Figura 25 - Validazione della password.....	32
Figura 26 - Validazione Email.....	33
Figura 27 - Recupero task di un progetto	34
Figura 28 - Eliminazione di una Task	35
Figura 29 - Spostamento Task	36
Figura 30 - Configurazione Jest	48
Figura 31 - Sintassi unit test	49
Figura 32 - unit test utils.js.....	49
Figura 33 - unit test project.js	50
Figura 34 - unit test admin.js	50
Figura 35 - unit test users.js	50
Figura 36 - unit test modal.js	51
Figura 37 - unit test modal.js	51
Figura 38 - unit test taskController.js.....	52
Figura 39 - riassunto unit test	52
Figura 40 - Form gestione profilo utente	53
Figura 41 - Mail registrazione	54
Figura 42 - Mail progetto.....	54
Figura 43 - Mail cancellazione account	55
Figura 44 - Mail codice di reset password	55
Figura 45 - Form reset password	56
Figura 46 - Form di contatto	56
Figura 47 - Calendario	57
Figura 48 - Gantt consuntivo	58

1 Introduzione

1.1 Informazioni sul progetto

- Allievo: Eros Marucchi
 - Docente Responsabile: Maurizio Di Florio
 - Perito 1: Fabrizio Valsangiacomo
 - Perito 2: Daniel Pagano
 - Data di inizio: 05.05.2025
 - Data di fine: 16.05.2025
 - Luogo di sviluppo: Scuola Arti e Mestieri di Trevano
 - Sezione scolastica: Informatica 4° anno
-

1.2 Abstract

The aim of the TeamSync project is to develop an interactive and intuitive web application that supports teams in organising work according to agile methodologies. The application aims to simplify collaborative task management by improving task visibility and communication between team members.

Through a configurable dashboard, each team can monitor the progress of its projects, create and assign tasks flexibly, define priorities and deadlines, and adapt the status of activities to operational needs. The system provides two distinct roles admin and user to differentiate access levels and responsibilities within the team and application.

The interface allows users to customise their tasks by assigning colours, thus facilitating visual identification within the dashboard. Each user can manage their own tasks independently, while admins can also manage the tasks of others, as well as create and configure projects and teams.

TeamSync aims to provide a practical, flexible and user-centred tool that promotes productivity and coordination in development teams.

1.3 Scopo

L'obiettivo del progetto TeamSync è sviluppare un applicativo web interattiva e intuitiva che supporti i team nell'organizzazione del lavoro secondo metodologie agili. L'applicazione si propone di semplificare la gestione collaborativa delle attività, migliorando la gestione delle task e la comunicazione nel team.

Attraverso una dashboard configurabile, ogni team può monitorare l'avanzamento dei propri progetti, creare e assegnare task in modo flessibile, definire priorità e scadenze, e adattare lo stato delle attività alle esigenze operative. Il sistema prevede due ruoli distinti admin e user per differenziare i livelli di accesso e le responsabilità all'interno del team e dell'applicativo.

L'interfaccia consente agli utenti di personalizzare i propri task mediante l'assegnazione di colori, facilitando così l'identificazione visiva all'interno della dashboard. Ogni utente può gestire autonomamente i propri task, mentre gli admin hanno la possibilità di gestire anche quelli altrui, oltre a creare e configurare progetti e i team.

TeamSync mira a fornire uno strumento pratico, flessibile e centrato sull'utente, che favorisca la produttività e il coordinamento nei team di sviluppo.

2 Analisi

2.1 Analisi del dominio

L'applicazione TeamSync si inserisce nel dominio della gestione di progetti e task collaborativi, un settore ampiamente sviluppato e supportato da numerosi strumenti già presenti sul mercato. L'ambito di utilizzo principale è costituito da team di sviluppo software che adottano metodologie agili come Scrum o Kanban, ma l'applicazione può risultare utile anche in contesti più generici di lavoro di gruppo.

Durante l'analisi del dominio, sono stati presi in esame alcuni dei principali competitor, come Trello, ClickUp, Monday.com e Notion. Questi strumenti offrono funzionalità avanzate per la pianificazione, l'assegnazione e il monitoraggio delle task, spesso accompagnate da interfacce complesse e una grande varietà di opzioni configurabili. Tuttavia, molti di questi strumenti possono risultare eccessivi per team più piccoli o per chi cerca un'esperienza più focalizzata sulle necessità essenziali della gestione agile.

TeamSync si propone come un'alternativa leggera, mirata e didattica, pensata per rendere la gestione delle attività chiara e accessibile, offrendo solo le funzionalità fondamentali in un'interfaccia semplice e configurabile. L'obiettivo non è competere con piattaforme enterprise, ma fornire uno strumento funzionale, educativo e immediato, adatto a contesti scolastici o a team che vogliono concentrarsi sull'essenziale della collaborazione.

2.2 Analisi e specifica dei requisiti

Spiegazione elementi tabella dei requisiti:

ID: identificativo univoco del requisito

Nome: breve descrizione del requisito

Priorità: indica l'importanza di un requisito nell'insieme del progetto, definita assieme al committente.

Versione: indica la versione del requisito. Ogni modifica del requisito avrà una versione aggiornata.

Sulla documentazione apparirà solamente l'ultima versione.

Note: eventuali osservazioni importanti o riferimenti ad altri requisiti.

Sotto requisiti: elementi che compongono il requisito.

ID: REQ-001	
Nome	Registrazione
Priorità	1
Versione	1.0
Note	Si necessita un'interfaccia grafica semplice ed intuitiva per ogni tipo di utente.
Sotto requisiti	
001	Si necessita una maschera di registrazione
002	Si necessitano gli input per inserire le informazioni richieste
003	Si necessita di una mail di conferma

ID: REQ-002	
Nome	Login
Priorità	1
Versione	1.0
Note	Si necessita un'interfaccia grafica semplice ed intuitiva per ogni tipo di utente.
Sotto requisiti	
001	Si necessita una maschera di login
002	Si necessita della possibilità di rimpostare la password

ID: REQ-003	
Nome	User e Admin
Priorità	1
Versione	1.0
Note	
Sotto requisiti	
001	Si necessita il tipo di utente user
002	Si necessita il tipo di utente admin

ID: REQ-004	
Nome	Creazione team e progetti
Priorità	1
Versione	1.0
Note	Si necessita un utente admin
Sotto requisiti	
001	Si necessita la possibilità di creare dei team
002	Si necessita la possibilità di creare dei progetti

ID: REQ-005	
Nome	Creazione sprint
Priorità	1
Versione	1.0
Note	Si necessita un utente user
Sotto requisiti	
002	Si necessita la possibilità di creare delle task
003	Si necessita la possibilità di inserire, descrizione, stato, colore, priorità, scadenza e peso ad una task.
004	Si necessita la possibilità di filtrare le task

ID: REQ-006	
Nome	Assegnazione task e stato.
Priorità	1
Versione	1.0
Note	Si necessita un utente admin
Sotto requisiti	
001	Si necessita che un utente admin abbia la possibilità di poter assegnare delle task.
002	Si necessita che un utente admin possa cambiare lo stato delle task di tutti.

ID: REQ-007	
Nome	Presa in carico delle task
Priorità	1
Versione	1.0
Note	
Sotto requisiti	
001	Si necessita che qualsiasi tipo di utente possa prendersi in carico le task libere

2.3 Use case

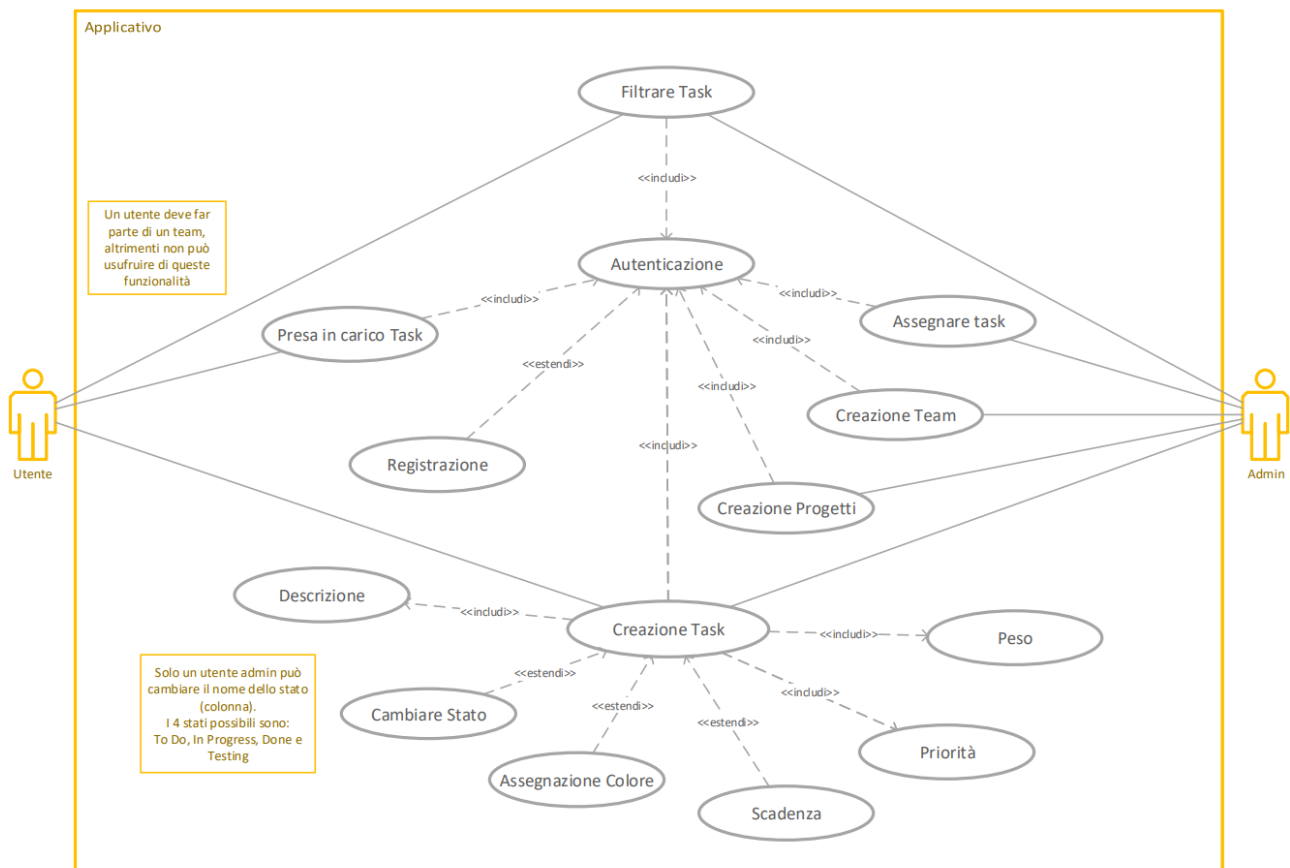


Figura 1 - Use Case

Spiegazione:

Il sistema applicativo per la gestione delle task coinvolge due attori principali: l'Utente e l'Admin. Entrambi devono effettuare l'autenticazione per accedere alle funzionalità dell'applicazione, e un utente può operare solo se è membro di un team. L'Utente ha la possibilità di registrarsi, autenticarsi, prendere in carico task, crearli (specificando descrizione, colore, scadenza, priorità e peso), filtrare le task e modificare lo stato dei propri. L'Admin, oltre a disporre delle stesse funzionalità, possiede privilegi aggiuntivi: può creare team e progetti, assegnare task ad altri utenti e modificare lo stato delle task indipendentemente dal proprietario. Le task possono assumere quattro stati: To Do, In Progress, Testing e Done.

2.4 Pianificazione

Ecco la pianificazione preventiva del mio progetto TeamSync:

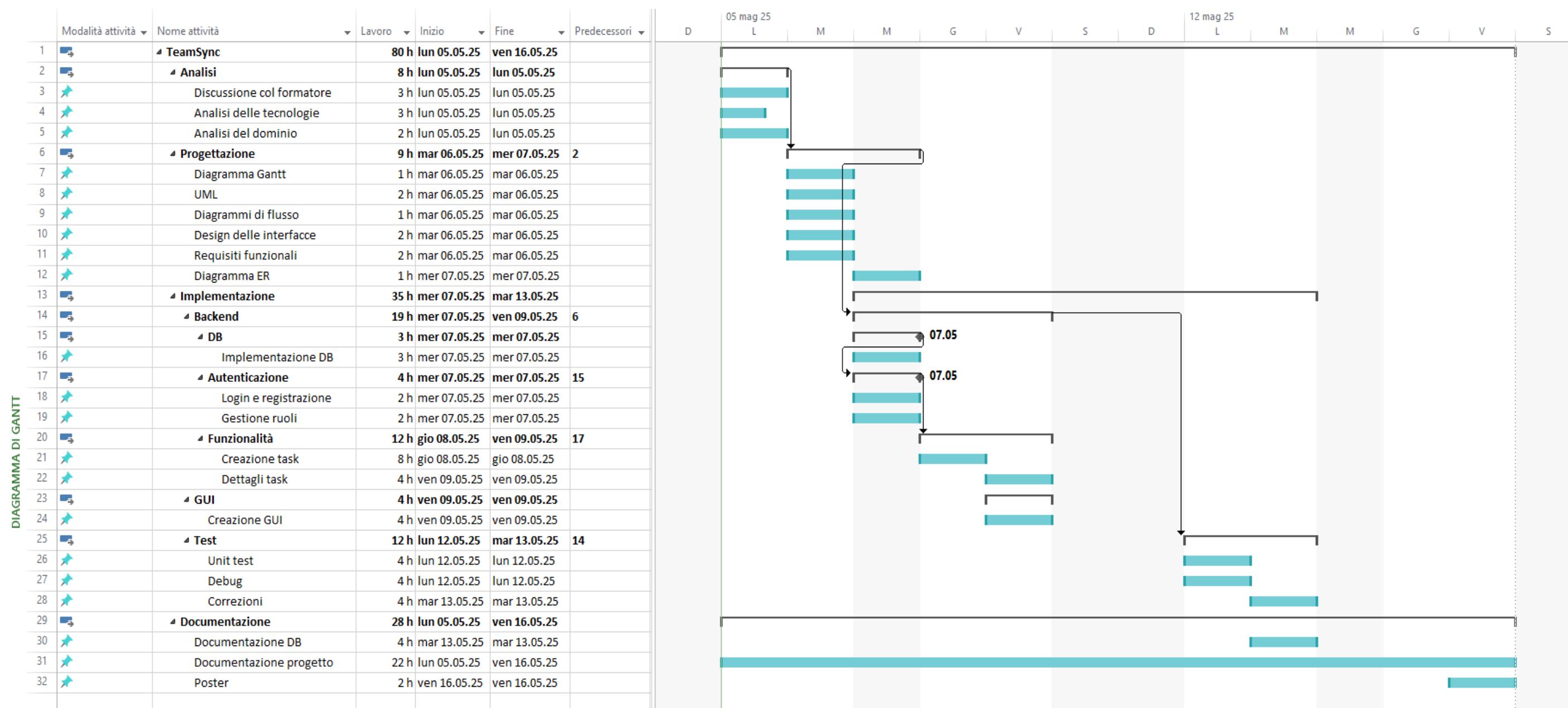


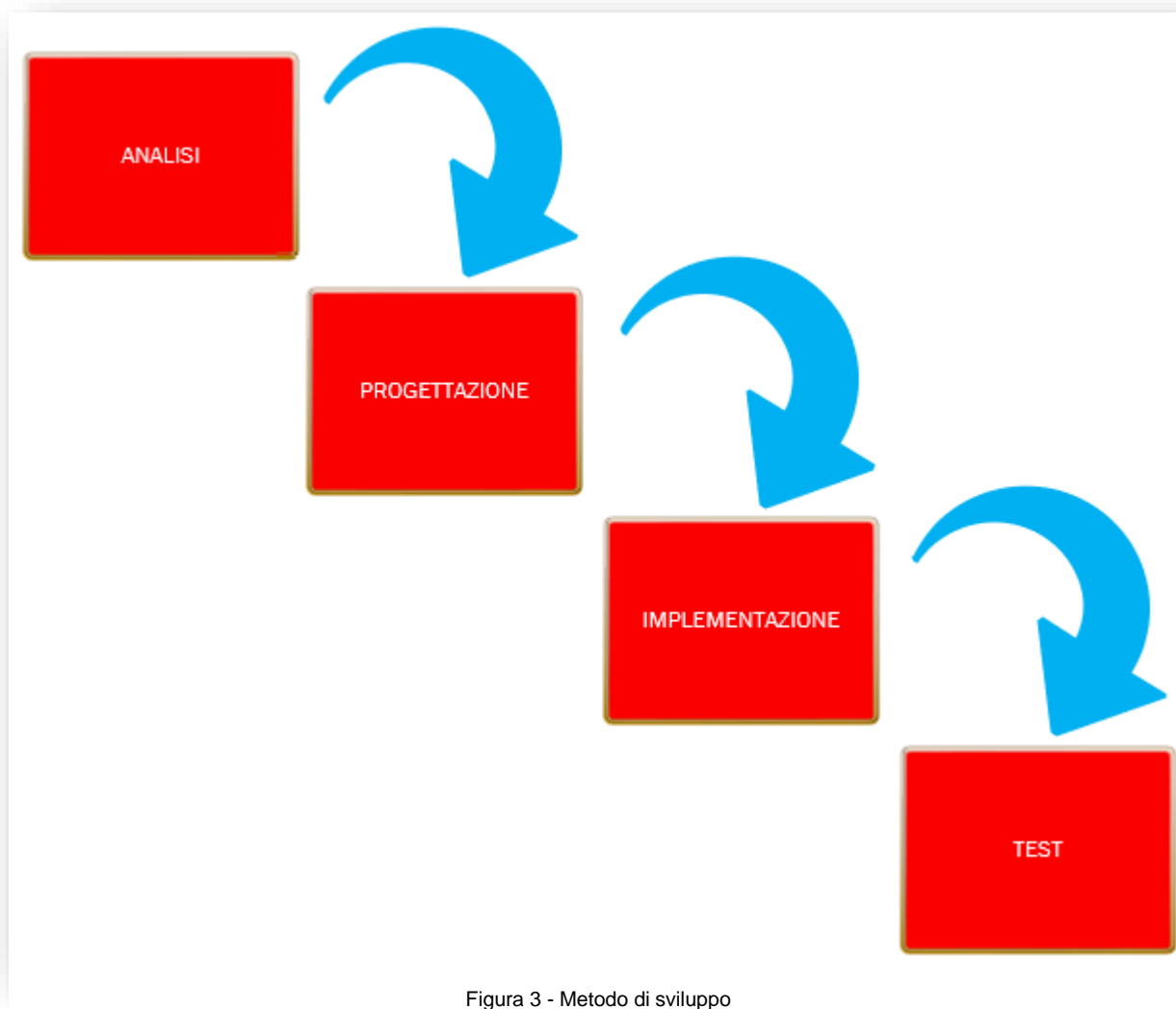
Figura 2 - Gantt

Il diagramma di Gantt presentato segue il modello Waterfall, ovvero un approccio sequenziale allo sviluppo del progetto, in cui ogni fase inizia solo dopo il completamento di quella precedente. La pianificazione è suddivisa in diverse macrofasi ben definite: Analisi, Progettazione, Implementazione, Test e Documentazione, ciascuna composta da attività specifiche ordinate cronologicamente.

2.4.1 Metodo di sviluppo

Ho scelto di utilizzare il modello a cascata perché, secondo me, semplifica lo sviluppo del progetto suddividendo i compiti in modo chiaro.

Il metodo di sviluppo è rappresentato con il seguente schema:



Spiegazione dei vari punti:

- **Analisi:** comprendere e definire le specifiche e le funzionalità richieste dal sistema.
- **Progettazione:** creare un piano dettagliato che definisce come e quando verranno implementate le funzionalità del sistema.
- **Implementazione:** sviluppare le funzionalità specificate nel piano di progettazione utilizzando un linguaggio di programmazione.
- **Test:** verificare che il sistema funzioni correttamente e che soddisfi le specifiche e i requisiti identificati durante l'analisi.

2.5 Analisi dei mezzi

Per fare questo progetto mi è stato fornito un PC scolastico.

2.5.1 Software

- Visual Studio Code 1.78.2, editor di testo per sviluppare l'applicativo web.
 - XAMPP 3.3.0, per il servizio di MySQL.
 - Google Chrome 116.0.5845.188, browser per testare e testare l'applicativo.
 - FireFox 110.0, browser per testare l'applicativo.
 - Microsoft Edge 116.0.1938.81, browser per testare l'applicativo.
 - Microsoft Project, per fare il GANTT.
 - Microsoft Visio Professional 2019, per fare lo Use Case
-

2.5.2 Hardware

- Nome dispositivo: 427-19
 - Nome completo del dispositivo: 427-19.CPT.local
 - Processore: Intel Core i7-9700 CPU 3.00 GHz
 - RAM installata: 32.0 GB
 - Scheda Video: NVIDIA GeForce RTX 2060
 - Tipo sistema: Sistema operativo a 64 bit, processore basato su x64
 - Versione di Windows: Windows 10 Enterprise, versione 22H2
-

3 Progettazione

3.1 Design dei dati e database

Il l'immagine seguente descrive la struttura logica del database destinato alla gestione di progetti, attività (task), team di lavoro e utenti. Lo schema è stato progettato per supportare una piattaforma collaborativa nella quale gli utenti possono essere organizzati in team, assegnati a progetti e incaricati di svolgere task specifici.

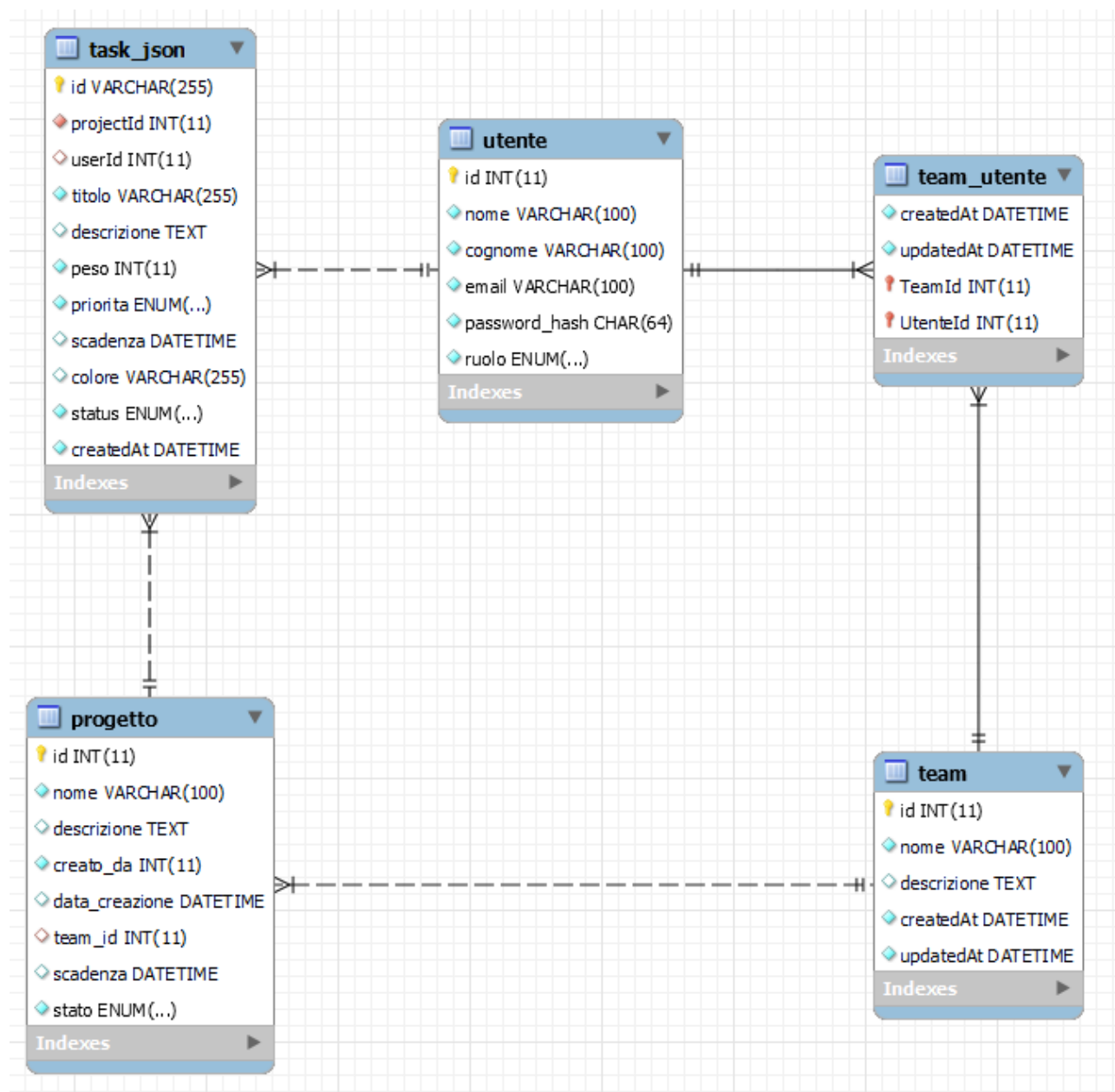


Figura 4 - Diagramma ER

3.2 Struttura delle tabelle e motivazioni

3.2.1 Tabella utente

Contiene le informazioni anagrafiche e di autenticazione degli utenti.

Motivazioni progettuali:

- La separazione tra nome, cognome ed email consente ricerche, ordinamenti e filtri più flessibili, soprattutto in contesti con numerosi utenti.
- Il campo ruolo (di tipo ENUM) permette di distinguere tra i diversi livelli di accesso (ad esempio, admin, membro, manager) direttamente a livello database, facilitando il controllo degli accessi.
- password_hash garantisce la sicurezza dei dati sensibili, memorizzando un hash sicuro della password senza conservarla in chiaro.

Accessi tipici:

- Autenticazione dell'utente tramite email e verifica della password.
 - Ricerca per nome o cognome.
 - Filtraggio degli utenti in base al ruolo
-

3.2.2 Tabella team

Rappresenta i team di lavoro cui gli utenti possono appartenere.

Motivazioni progettuali:

- Il campo descrizione permette di fornire informazioni aggiuntive utili a distinguere i team o a descriverne gli obiettivi.
- I timestamp createdAt e updatedAt abilitano la gestione della cronologia (audit log), rendendo possibile l'ordinamento per data di creazione o la visualizzazione delle modifiche più recenti.

Accessi tipici:

- Ricerca dei team per nome.
-

3.2.3 Tabella progetto

Contiene i progetti che vengono creati e gestiti dai team.

Motivazioni progettuali:

- La distinzione tra autore del progetto (creato_da) e team responsabile (team_id) consente maggiore flessibilità, ad esempio permettendo la creazione da parte di utenti esterni al team.
- I campi scadenza e stato permettono una gestione avanzata del ciclo di vita del progetto (es. pianificazione, tracciamento, completamento).

Accessi tipici:

- Elenco dei progetti associati a un team specifico.
- Ricerca dei progetti filtrati per stato (attivo, completato, ecc.) o data di scadenza.

3.2.4 Tabella task_json

Contiene le attività (task) assegnate agli utenti nell'ambito dei progetti.

Motivazioni progettuali:

- Il campo `userId` permette di assegnare task specifici a singoli utenti, favorendo responsabilità chiare.
- Il collegamento al progetto tramite `projectId` mantiene l'organizzazione delle attività in contesti di progetto.
- L'utilizzo di campi come `priority`, `weight`, `status` e `color` abilita una gestione dinamica e personalizzabile, utile anche per interfacce grafiche (es. bacheche Kanban).
- Il campo `createdAt` consente il tracciamento temporale per monitorare l'andamento del lavoro.

Accessi tipici:

- Visualizzazione delle task assegnati a un utente.
- Visualizzazione delle task relativi a un progetto.
- Filtro per priorità, stato o scadenza.

3.3 Relazioni fra tabelle

Il database prevede le seguenti relazioni tra entità principali:

- **Utente – Team** (relazione molti-a-molti):
Un utente può appartenere a più team e viceversa.
Relazione implementata attraverso la tabella ponte `team_utente`.
- **Team – Progetto** (relazione uno-a-molti):
Ogni progetto è assegnato a un solo team, tramite il campo `team_id`.
- **Utente – Progetto** (relazione uno-a-molti):
Il campo `created_by` specifica quale utente ha avviato il progetto.
- **Progetto – Task** (relazione uno-a-molti):
Ogni task è legata a un solo progetto, tramite il campo `projectId`.
- **Utente – Task** (relazione uno-a-molti):
Le task possono essere assegnate a un utente specifico tramite il campo `userId`.

3.4 ORM (Object-Relational Mapping)

Per la creazione del database, ho scelto di utilizzare l'ORM Sequelize per semplificare la gestione delle operazioni SQL attraverso un'interfaccia JavaScript.

Ho creato quattro modelli: `Projects.js`, `TaskJson.js`, `Team.js` e `Utente.js`, ciascuno dei quali rappresenta una tabella del database. La definizione delle colonne all'interno di ogni modello segue la sintassi specifica di Sequelize.

3.4.1 Sintassi

```

1  /**
2  * Definizione del modello Progetto utilizzando Sequelize
3  * Ogni proprietà rappresenta una colonna nella tabella 'progetto'
4  */
5  const Progetto = sequelize.define('Progetto', {
6    /**
7     * ID del progetto
8     * Chiave primaria auto-incrementale per identificazione univoca
9     */
10   id: {
11     type: DataTypes.INTEGER,
12     primaryKey: true,
13     autoIncrement: true
14   },

```

Figura 5 - Sintassi Sequelize

Nel codice viene definito un modello chiamato “*Progetto*” tramite la funzione “*sequelize.define()*”, che accetta come primo parametro il nome del modello e come secondo un oggetto che descrive le colonne della tabella. In questo caso, viene definita la colonna “*id*” con diverse proprietà: “*type: DataTypes.INTEGER*” indica il tipo di dato, “*primaryKey: true*” la imposta come chiave primaria della tabella, e “*autoIncrement: true*” fa sì che il valore venga incrementato automaticamente a ogni nuovo inserimento. Ogni colonna viene quindi configurata come una proprietà dell’oggetto, specificando tipo e vincoli, in modo da rappresentare correttamente la struttura della tabella nel database.

3.4.2 Relazioni

```

1  /**
2  * Definizione delle relazioni many-to-many
3  * Configura le associazioni bidirezionali tra Team e Utenti
4  *
5  * Relazioni stabilite:
6  * - Un team può avere più utenti
7  * - Un utente può appartenere a più team
8  * - TeamUtente funge da tabella di join
9  */
10 Team.belongsToMany(Utente, { through: TeamUtente });
11 Utente.belongsToMany(Team, { through: TeamUtente });

```

Figura 6 - Relazioni (ORM)

In questo frammento di codice viene definita una relazione *many-to-many* (multi-a-molti) tra i modelli Team e Utente. La funzione “*belongsToMany()*” viene utilizzata per indicare che un’istanza di un modello può essere associata a più istanze di un altro modello, e viceversa.

Nel dettaglio, “*Team.belongsToMany(Utente, { through: TeamUtente })*” indica che ogni team può avere più utenti associati, mentre “*Utente.belongsToMany(Team, { through: TeamUtente })*” stabilisce che ogni utente può far parte di più team. L’opzione “*{ through: TeamUtente }*” specifica che questa relazione è mediata da una tabella di join chiamata “*TeamUtente*”, che funge da ponte tra le due tabelle principali (Team e Utente). Questa tabella contiene tipicamente le chiavi esterne di entrambe le entità e può includere anche ulteriori informazioni, come il ruolo dell’utente all’interno del team o la data di assegnazione.

3.4.3 Inizializzazione database

```

1  async function initDatabase() {
2    try {
3      const isFirstRun = !await isDatabaseInitialized();
4
5      await createDatabaseIfNotExists();
6      await sequelize.authenticate();
7      logger.info('Connessione al database stabilita con successo.');

```

Figura 7 - Inizializzazione DB

La funzione `initDatabase()` è una funzione asincrona che gestisce l'intera inizializzazione del database utilizzando *Sequelize*. Inizia verificando se si tratta della prima esecuzione del database e, in caso affermativo, crea il database se non esiste. Successivamente, autentica la connessione al database, importa i modelli necessari (*Utente*, *Team*, *Progetto*, *TaskJson*), stabilisce le relazioni tra di essi tramite metodi come `hasMany` e `belongsToMany`, e sincronizza i modelli con il database. Poi crea un utente admin di default e, se è la prima esecuzione, esegue i *seeders* per popolare il database con i dati iniziali. La funzione restituisce l'istanza di *Sequelize* configurata, mentre eventuali errori vengono catturati e loggati, interrompendo l'esecuzione del processo.

(Mi sono fatto aiutare dal seguente modello di intelligenza artificiale: <https://chatgpt.com/>)

3.5 Design delle interfacce

Questi design sono stati realizzati con l'ausilio dell'intelligenza artificiale tramite la piattaforma <https://v0.dev/>.

Tutte le interfacce sono progettate per garantire un'esperienza ottimale sia su dispositivi desktop che mobile, grazie a un design completamente responsive.

3.5.1 Dashboard user

Questa è la schermata di gestione delle attività del progetto, organizzata secondo la metodologia Kanban. In alto a sinistra è presente un pulsante per creare un nuovo task, mentre al centro la pagina è suddivisa in quattro colonne: Da Fare, In Corso, In Revisione e Completati, che rappresentano lo stato di avanzamento delle attività. Ogni task include informazioni rilevanti come il titolo, una breve descrizione, il responsabile assegnato, la priorità e la data di scadenza. I colori e le etichette aiutano a distinguere rapidamente il livello di urgenza e il peso delle attività. Sulla sinistra, un menu laterale consente di accedere alla dashboard, ai propri task, al team, ai progetti e alle impostazioni.

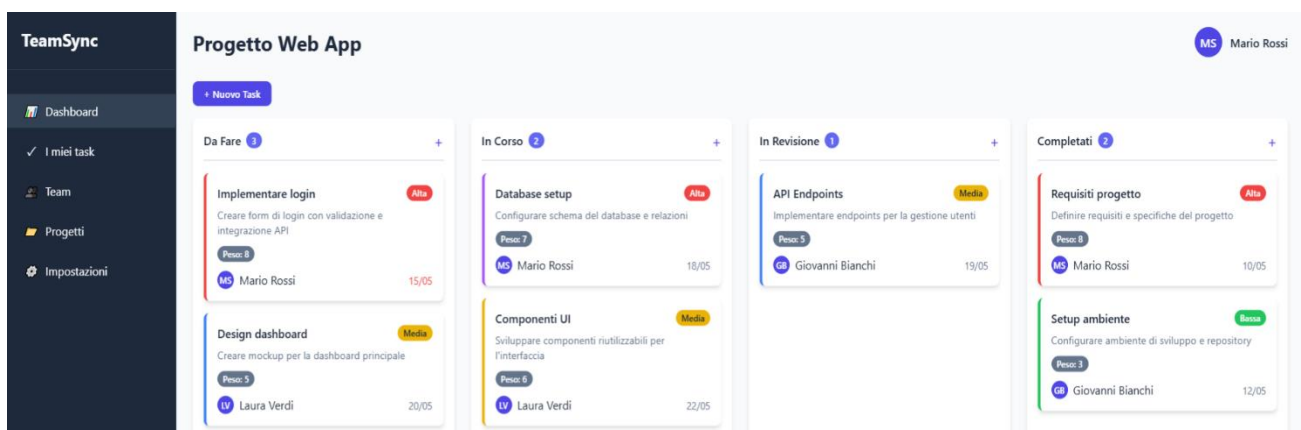


Figura 8 - Design dashboard user

3.5.2 Pannello admin (progetti)

Questa è l'interfaccia progettata per il pannello di amministrazione di TeamSync, focalizzata sulla gestione dei progetti. La schermata presenta una sidebar per la navigazione tra le sezioni e un'area centrale che mostra riepiloghi (utenti, team, progetti, task completati) e una tabella dei progetti con dettagli su team, membri, avanzamento, stato, scadenza e azioni rapide.

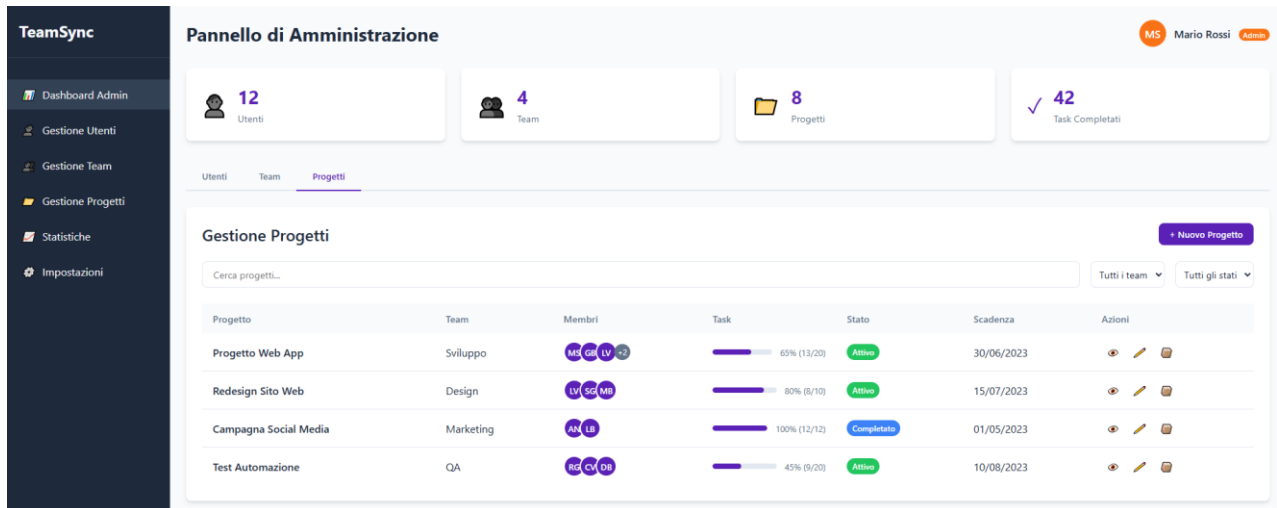


Figura 9 - Design pannello Admin - progetti

3.5.3 Pannello admin (team)

Questa schermata fa parte del pannello di controllo dell'amministratore, con un focus specifico sulla gestione dei team. L'interfaccia consente di visualizzare l'elenco dei team e, tramite il pulsante "Gestisci" o le icone delle azioni rapide, permette di modificarne la composizione, assegnare membri e coordinare le attività.

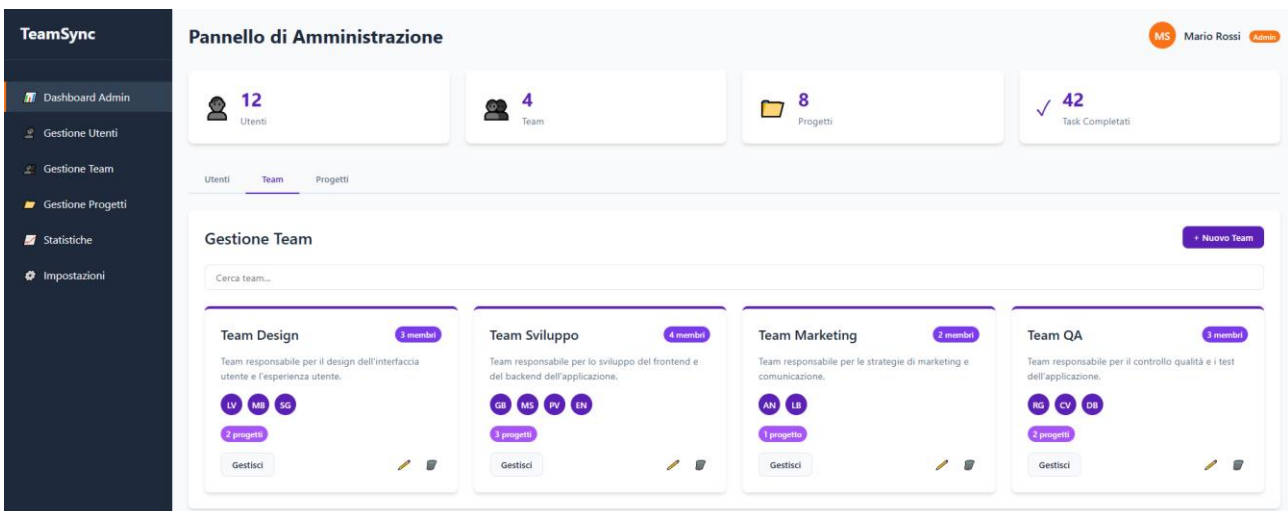


Figura 10 - Design pannello Admin - team

3.5.4 Pannello admin (utenti)

Questa schermata è dedicata alla gestione degli utenti all'interno del pannello di controllo dell'amministratore. L'interfaccia permette di creare nuovi utenti e, attraverso la tabella sottostante, di visualizzare e modificare quelli già esistenti utilizzando le azioni rapide disponibili.

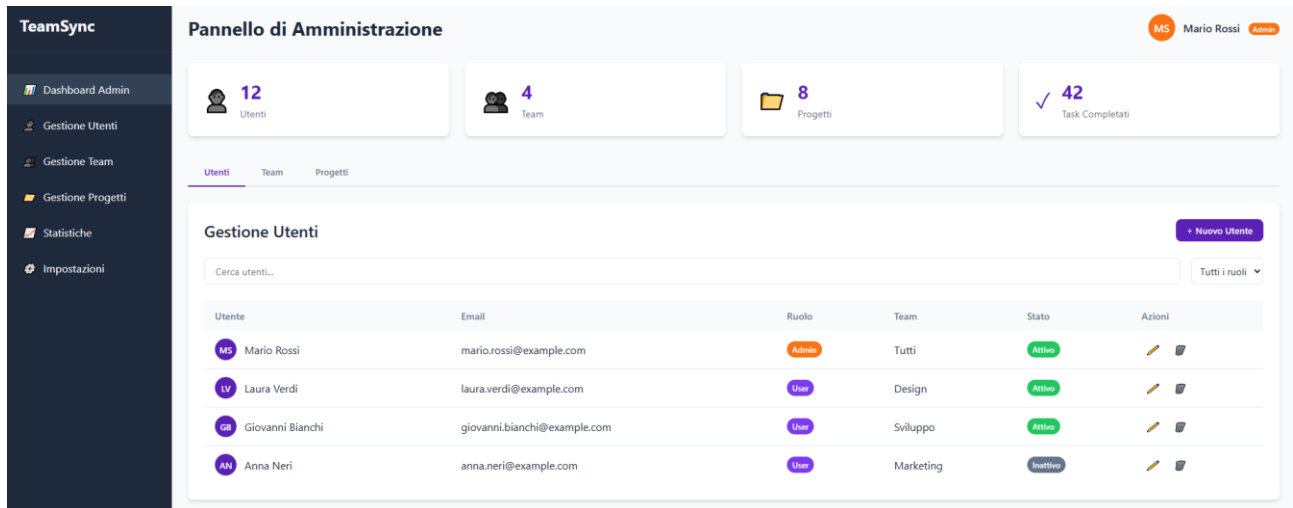


Figura 11 – Design pannello Admin – utenti

3.5.5 Form di login

Questa schermata rappresenta il form di login. L'interfaccia, semplice e moderna, consente agli utenti registrati di accedere al proprio account attraverso due campi principali: uno per l'email, accompagnato da un'icona a forma di busta, e uno per la password, con icona a forma di lucchetto. Sono presenti anche un link per il recupero della password e un pulsante evidenziato in blu con la dicitura "Accedi", che avvia il processo di autenticazione. In alto, l'utente può scegliere tra le schede "Accedi" e "Registrati", mentre in basso è presente un link testuale che invita alla registrazione nel caso non si disponga ancora di un account.

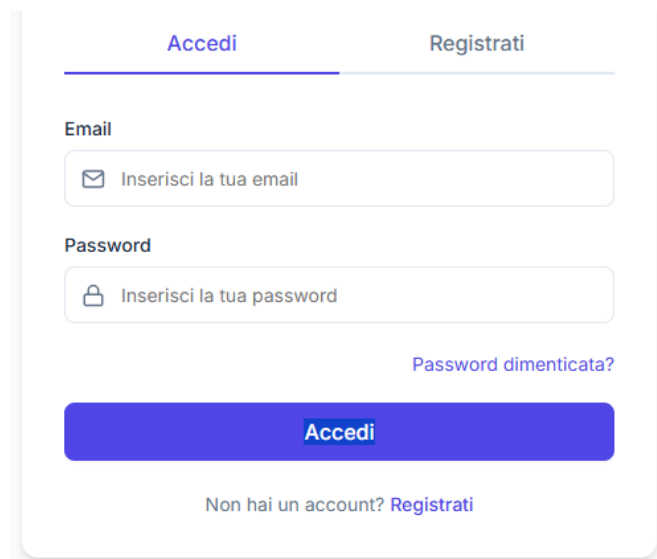


Figura 12 - Design form Login

3.5.6 Form di registrazione

Questa schermata rappresenta il form di registrazione. L'interfaccia, chiara e moderna, consente ai nuovi utenti di creare un account compilando i campi richiesti: nome, cognome, email e password, ciascuno accompagnato da un'icona descrittiva. È inoltre presente una casella di spunta per accettare i termini e le condizioni, necessaria per completare la registrazione. Il pulsante blu "Registrati" consente di inviare i dati e creare l'account.

The image shows a registration form with two tabs: "Accedi" and "Registrati". The "Registrati" tab is active. The form contains the following elements:

- Nome**: A text input field with a person icon and placeholder text "Inserisci il tuo nome".
- Cognome**: A text input field with a person icon and placeholder text "Inserisci il tuo cognome".
- Email**: A text input field with an envelope icon and placeholder text "Inserisci la tua email".
- Password**: A text input field with a lock icon and placeholder text "Crea una password".
- Accetto i termini e le condizioni**: A checkbox.
- Registrati**: A large blue button.
- Hai già un account? Accedi**: A link below the registration button.

Figura 13 - Design form registrazione

3.5.7 Nota design interfacce

Il design delle interfacce potrebbe aver avuto qualche piccola modifica nel corso dello sviluppo del progetto.

3.6 Design procedurale

3.6.1 Utente user

L'utente inizia effettuando il login; se non è autenticato, viene bloccato. Dopo il login, il sistema verifica se l'utente fa parte di un team: se la risposta è negativa, viene mostrato un messaggio informativo e il processo termina. Se l'utente appartiene a un team, accede alla dashboard da cui può svolgere diverse operazioni: creare task, prendere in carico task, creare sprint, personalizzare task e filtrare task. In qualsiasi momento può effettuare il logout, che conclude il flusso.

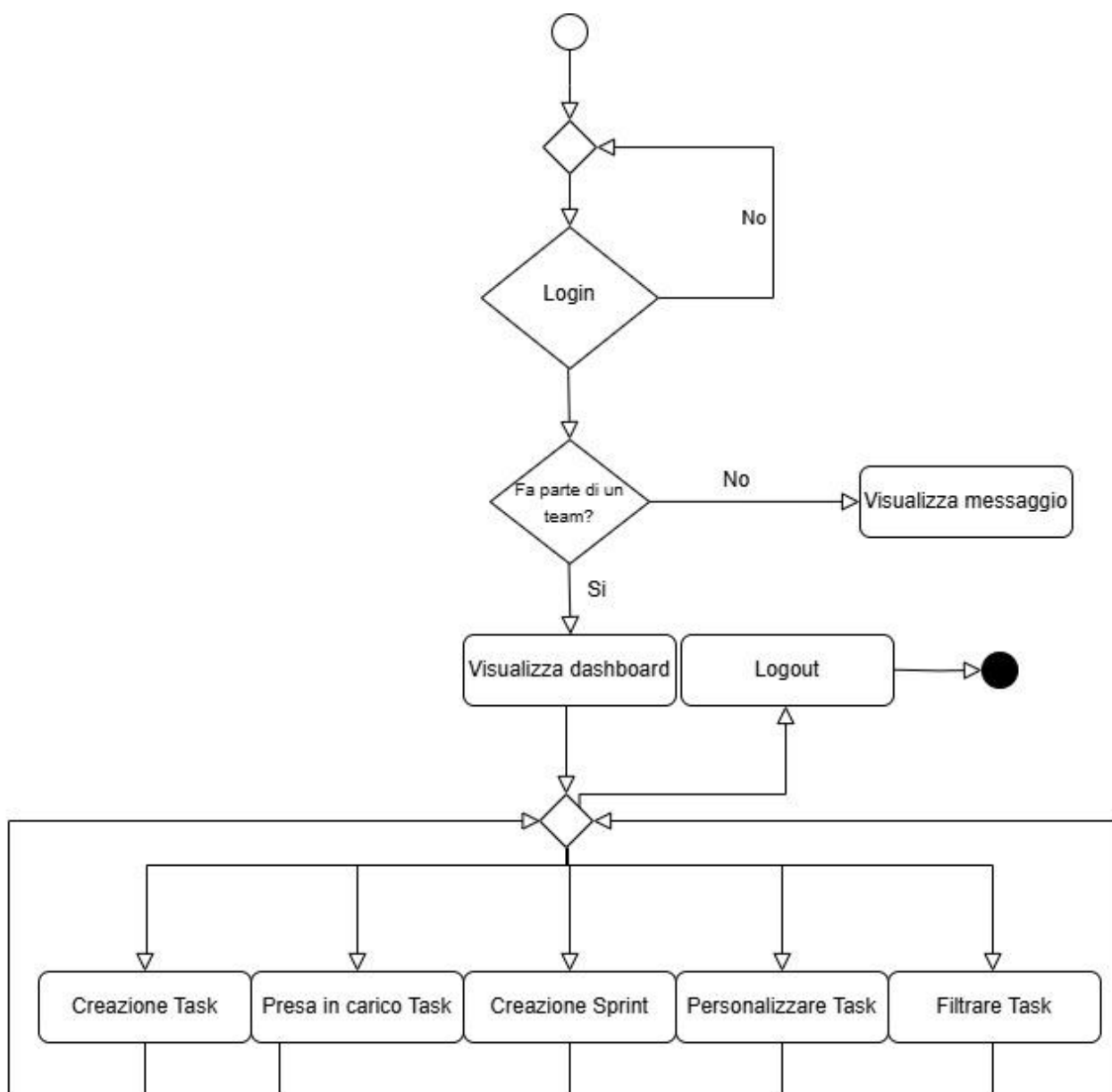


Figura 14 - Diagramma di flusso user

3.6.2 Utente admin

Il processo inizia con il login dell'amministratore; se le credenziali non sono corrette, l'accesso viene negato. Una volta autenticato, l'Admin accede alla propria dashboard, da cui può gestire varie funzionalità di livello superiore. Le azioni disponibili includono: creazione di team, creazione di progetti, assegnazione delle task agli utenti, filtraggio delle task e personalizzazione delle task. In qualsiasi momento, l'amministratore può eseguire il logout, che conclude il flusso.

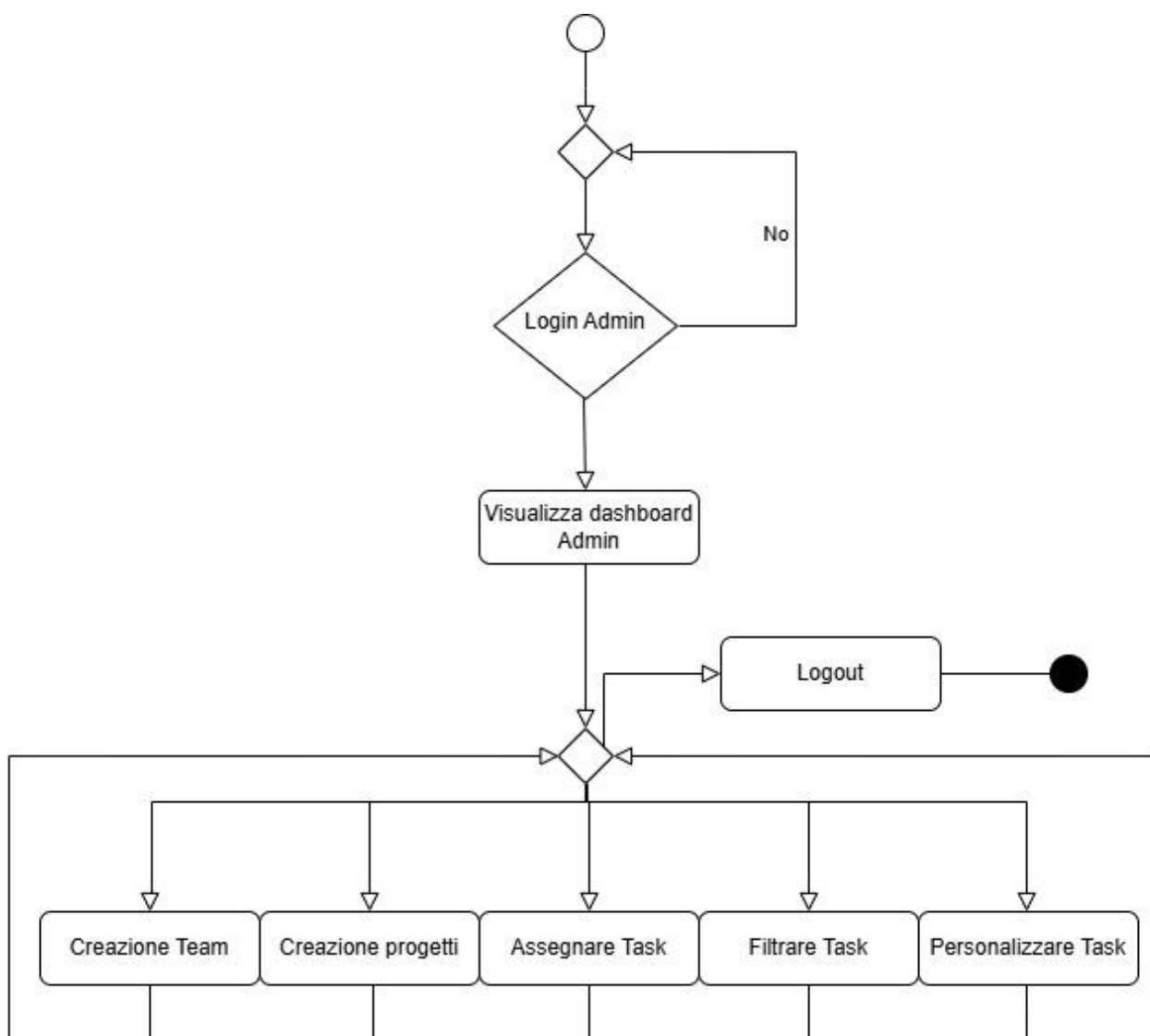


Figura 15 - Diagramma di flusso admin

4 Implementazione

4.1 Linguaggi e framework utilizzati

L'applicazione è stata sviluppata seguendo un approccio *full-stack*, con una chiara separazione tra logica lato server e interfaccia utente.

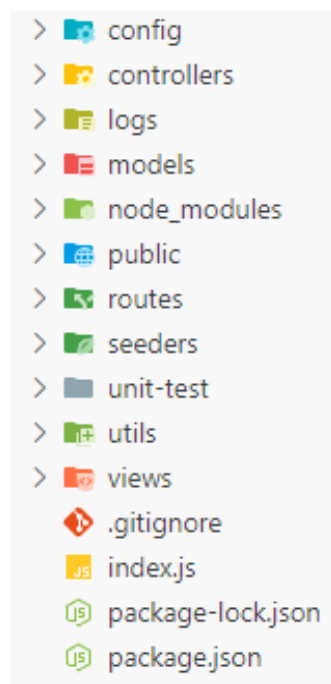
Il backend è stato sviluppato utilizzando *Node.js*, una piattaforma *JavaScript* lato server che consente di costruire applicazioni web performanti e non bloccanti grazie al suo modello asincrono e orientato agli eventi. *Node.js* si integra perfettamente con il vasto ecosistema di moduli disponibili tramite il gestore di pacchetti *npm*, agevolando lo sviluppo e l'aggiunta di nuove funzionalità.

A supporto della struttura del server è stato impiegato *Express.js*, un *framework* minimalista e flessibile che semplifica la gestione delle rotte, delle richieste *HTTP* e delle middleware. Questo ha permesso di organizzare il codice in modo chiaro e modulare, favorendo una più agevole manutenzione dell'applicazione.

Per quanto riguarda la generazione dinamica delle pagine *HTML*, è stato scelto *Handlebars* come motore di template. Questo sistema consente di separare la logica di presentazione dalla logica applicativa, permettendo di creare viste dinamiche e facilmente riutilizzabili. Grazie a *Handlebars*, l'interfaccia può essere popolata con dati provenienti dal server mantenendo una struttura pulita e leggibile.

Sul fronte *client*, lo sviluppo dell'interfaccia utente è stato realizzato attraverso l'impiego congiunto di *HTML*, *CSS* e *JavaScript*.

4.2 Struttura del progetto



Il progetto è organizzato secondo l'architettura *MVC* (*Model-View-Controller*), un modello che favorisce la separazione delle responsabilità e contribuisce a rendere il codice più leggibile, manutenibile e scalabile. All'interno della struttura, i controller si trovano nella cartella *controllers* e sono responsabili della gestione delle richieste in ingresso, dell'invocazione della logica applicativa e della restituzione delle risposte appropriate. I modelli, collocati nella directory *models*, rappresentano le entità principali dell'applicazione e gestiscono l'interazione con il database, incapsulando la logica di accesso ai dati.

Le viste, all'interno della cartella *views*, rappresentano la componente dell'interfaccia utente e sono utilizzate nel caso di rendering server-side. La cartella *routes* definisce i vari endpoint dell'applicazione, collegando ciascuna rotta al controller corrispondente. Le configurazioni globali dell'applicazione sono gestite all'interno della directory *config*, dove risiedono i file relativi, ad esempio, alla connessione al database o ad altre impostazioni ambientali. Per quanto riguarda i contenuti statici come immagini, fogli di stile e script client-side, questi si trovano nella cartella *public*, accessibile direttamente dal browser. La directory *utils* contiene moduli e funzioni di utilità riutilizzabili in vari punti dell'applicazione. I file di log sono archiviati in *logs*, utili per tracciare il comportamento dell'applicazione, monitorarne l'andamento o analizzare eventuali errori.

Figura 16 - Struttura progetto

La cartella *seeders* raccoglie script utilizzati per il popolamento iniziale del database, particolarmente utili in fase di sviluppo o testing. I test automatici sono contenuti nella cartella *unit-test*, dove vengono verificati il comportamento e la correttezza delle singole componenti dell'applicazione. La directory *node_modules*, generata automaticamente da *npm*, contiene tutte le dipendenze necessarie al progetto.

Il file *index.js* rappresenta il punto di ingresso dell'applicazione: qui vengono inizializzati il server, le configurazioni principali e le *route*. I file *package.json* e *package-lock.json* descrivono le dipendenze del progetto e ne garantiscono la coerenza durante l'installazione, mentre *.gitignore* specifica quali file o cartelle devono essere esclusi dal versionamento tramite *Git*.

4.3 Index.js

4.3.1 Configurazione del Template Engine (Handlebars)

```

1  /**
2   * Configurazione Handlebars come template engine
3   * - Estensione file: .hbs
4   * - Layout: Disabilitato (gestito manualmente)
5   * - Cartella partials: views/partials
6   * - Helper personalizzati per la logica di template
7   */
8  app.engine('hbs', exphbs.engine({
9    extname: '.hbs',
10   defaultLayout: false,
11   partialsDir: path.join(__dirname, 'views/partials'),
12   helpers: {
13     eq: function (v1, v2) { return v1 === v2; }
14   }
15 }));
16 app.set('view engine', 'hbs');
17 app.set('views', path.join(__dirname, 'views'));

```

Figura 17 - Configurazione Handlebars

Nel file `index.js`, *Handlebars* è configurato tramite *express-handlebars* per rendere pagine *HTML* lato server. I template, con estensione `.hbs`, si trovano nella cartella `views`. L'opzione `defaultLayout: false` disabilita il layout predefinito, offrendo maggiore flessibilità. I *partial* riutilizzabili (es. *header*, *footer*) sono definiti nella cartella `views/partials`.

4.3.2 Configurazioni di Sicurezza

```

1  /**
2   * ===== CONFIGURAZIONE SICUREZZA =====
3   * Implementazione di multiple misure di sicurezza:
4   * 1. Rate Limiting: Previene attacchi brute force
5   * 2. Helmet: Headers HTTP di sicurezza
6   * 3. CORS: Gestione delle richieste cross-origin
7   * 4. Cookie Parser: Gestione sicura dei cookie
8   * 5. Session: Gestione delle sessioni utente
9   */
10 const limiter = rateLimit(securityConfig.rateLimit);
11
12 app.use(helmet(securityConfig.helmet));
13 app.use(cors(securityConfig.cors));
14 app.use(cookieParser(process.env.COOKIE_SECRET));
15 app.use(express.json());
16 app.use(express.urlencoded({ extended: true }));
17 app.use(session(securityConfig.session));
18 app.use(limiter);

```

Figura 18 - Configurazioni di sicurezza

Possiamo vedere che vengono utilizzati i pacchetti: “*Helmet*”, “*Rate Limiting*”, “*CORS*”, e sessioni sicure. “*Helmet*” è configurato per impostare correttamente le intestazioni “*http*” e proteggere l'app da vulnerabilità come “*XSS*”, “*clickjacking*” e altre tecniche di attacco tramite *header* malformati. Il “*Rate Limiting*” è implementato per evitare attacchi “*DoS*” (Denial of Service) e *brute-force*, limitando il numero di richieste che un client può effettuare in un dato intervallo di tempo. La configurazione “*CORS*” gestisce le politiche di accesso *cross-origin*, permettendo solo le richieste provenienti da origini specifiche. Invece per la gestione sicura delle sessioni utente, viene utilizzato il pacchetto “*express-session*”.

4.4 Security.js

4.4.1 Configurazione della sessione

```

1  /**
2   * Configurazioni della Sessione
3   * Gestisce come le sessioni utente vengono create e mantenute
4   *
5   * @property {string} secret - Chiave segreta per firmare i cookie di sessione
6   * @property {boolean} resave - Evita il salvataggio della sessione se non modificata
7   * @property {boolean} saveUninitialized - Non salva sessioni non inizializzate
8   * @property {Object} cookie - Configurazione dei cookie di sessione
9   *   - secure: Attivo solo in produzione (HTTPS)
10  *   - httpOnly: Previene accesso via JavaScript
11  *   - maxAge: Durata del cookie (24 ore)
12  *   - sameSite: Protezione contro attacchi CSRF
13  */
14  session: {
15    secret: process.env.SESSION_SECRET || 'il-tuo-secret-key',
16    resave: false,
17    saveUninitialized: false,
18    cookie: {
19      secure: process.env.NODE_ENV === 'production',
20      httpOnly: true,
21      maxAge: 24 * 60 * 60 * 1000, // 24 ore
22      sameSite: 'strict'
23    }
24  },

```

Figura 19 - Configurazione sessione

La configurazione della sessione gestisce come vengono create e mantenute le sessioni utente nell'applicazione. In particolare, viene definita una chiave segreta per firmare i cookie di sessione, assicurando che i dati non vengano alterati durante il trasferimento. L'opzione *resave* impedisce il salvataggio della sessione se non è stata modificata, mentre *saveUninitialized* evita di salvare sessioni vuote, riducendo il consumo di memoria. La configurazione dei cookie include impostazioni per garantire una maggiore sicurezza: i cookie sono inviati solo tramite *HTTPS*, non sono accessibili tramite JavaScript (*httpOnly*), hanno una durata di 24 ore (*maxAge*) e sono protetti da attacchi *CSRF* grazie alla politica *sameSite: 'strict'*, che limita l'invio del cookie solo dal dominio di origine. Queste misure contribuiscono a rendere le sessioni sicure e resistenti a potenziali vulnerabilità.

4.4.2 Configurazioni Helmet

```

1  /**
2   * Configurazioni Helmet
3   * Imposta gli header HTTP di sicurezza
4   *
5   * Content Security Policy (CSP):
6   * - defaultSrc: Risorse predefinite solo dal proprio dominio
7   * - scriptSrc: JavaScript dal proprio dominio e inline
8   * - styleSrc: CSS dal proprio dominio, inline e Google Fonts
9   * - fontSrc: Font dal proprio dominio e Google Fonts
10  * - imgSrc: Immagini dal proprio dominio, data URLs e HTTPS
11  * - connectSrc: Connessioni WebSocket/AJAX solo dal proprio dominio
12  */
13  helmet: {
14    contentSecurityPolicy: {
15      directives: {
16        defaultSrc: ["'self'"],
17        scriptSrc: ["'self'", "'unsafe-inline'"],
18        styleSrc: ["'self'", "'unsafe-inline'", "https://fonts.googleapis.com"],
19        fontSrc: ["'self'", "https://fonts.gstatic.com"],
20        imgSrc: ["'self'", "data:", "https:"],
21        connectSrc: ["'self'"]
22      }
23    }
24  }

```

Figura 20 - Configurazioni Helmet

La configurazione di *Helmet* con “*Content Security Policy*” stabilisce delle direttive per controllare da quali origini le risorse possono essere caricate, contribuendo a proteggere l'applicazione da varie tipologie di attacchi. La direttiva “*defaultSrc*” limita il caricamento delle risorse al solo dominio del sito (*'self'*). Le risorse JavaScript sono permesse solo dal dominio stesso e, in aggiunta, è consentito l'uso di script inline tramite la direttiva “*scriptSrc*”. Per i fogli di stile “CSS”, vengono consentite risorse dal proprio dominio e da “*Google Fonts*”, mentre per i font è permesso caricare risorse da *'self'* e da “*Google Fonts*”. La direttiva “*imgSrc*” permette il caricamento di immagini dal proprio dominio, da URL sicuri (HTTPS) e anche da URL *data*: Infine, “*connectSrc*” limita le connessioni “*WebSocket*” e “*AJAX*” al solo dominio del sito.

4.4.3 Configurazioni di Sicurezza

```

1  /**
2   * Configurazioni Rate Limiter
3   * Protegge da attacchi DDoS e abusi delle API
4   *
5   * @property {number} windowMs - Finestra temporale per il conteggio (1 minuto)
6   * @property {number} max - Numero massimo di richieste nella finestra
7   * @property {string} message - Messaggio di errore per limite superato
8   * @property {boolean} standardHeaders - Abilita header standard di rate limit
9   * @property {boolean} legacyHeaders - Disabilita header legacy
10  */
11  ratelimit: {
12    windowMs: 60 * 1000,
13    max: 1000,
14    message: 'Troppe richieste, riprova più tardi.',
15    standardHeaders: true,
16    legacyHeaders: false
17  }
18  };

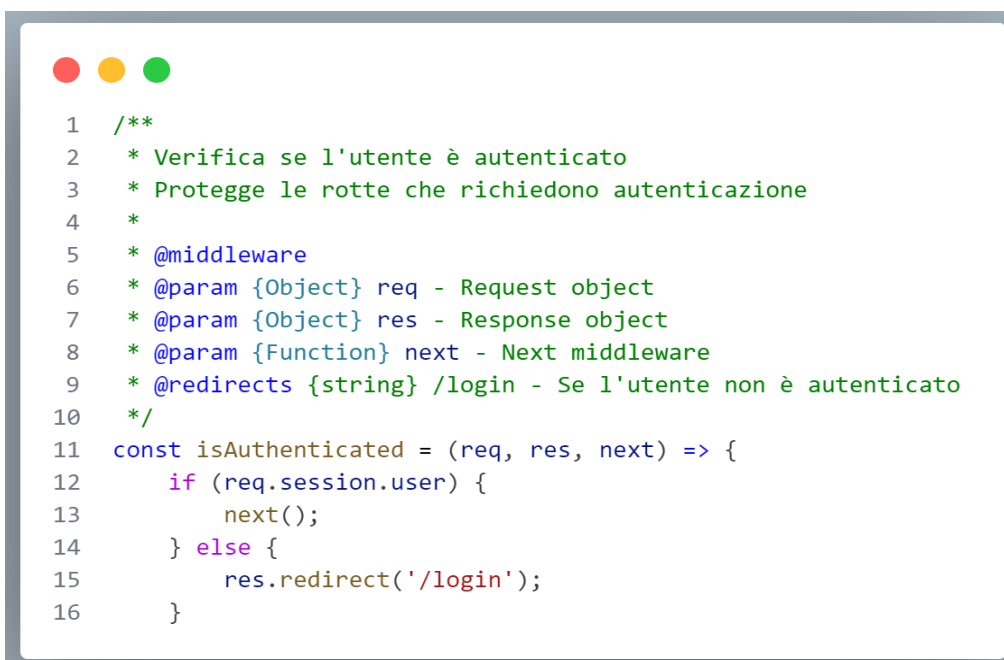
```

Figura 21 - Configurazioni di sicurezza

La configurazione del *Rate Limiter* serve a proteggere l'applicazione da abusi, sovraccarichi o attacchi di tipo *DoS* limitando il numero di richieste che un client può effettuare in un dato intervallo di tempo. In particolare, viene impostata una finestra temporale di 1 minuto (*windowMs*), durante la quale un massimo di 1000 richieste per IP sono consentite. Se questo limite viene superato, l'utente riceve un messaggio di errore personalizzato che lo invita a riprovare più tardi. L'opzione "*standardHeaders*" abilita gli "*header*" standard *HTTP* per comunicare al client lo stato del "*rate limiting*", mentre "*legacyHeaders*" è disattivata per evitare l'invio di "*header*" obsoleti. Questa configurazione è fondamentale per garantire la stabilità e l'affidabilità del servizio, soprattutto in ambienti pubblici o con alti volumi di traffico.

4.5 pageRoutes.js

4.5.1 Middleware di Autenticazione



```

1  /**
2   * Verifica se l'utente è autenticato
3   * Protegge le rotte che richiedono autenticazione
4   *
5   * @middleware
6   * @param {Object} req - Request object
7   * @param {Object} res - Response object
8   * @param {Function} next - Next middleware
9   * @redirects {string} /login - Se l'utente non è autenticato
10  */
11  const isAuthenticated = (req, res, next) => {
12    if (req.session.user) {
13      next();
14    } else {
15      res.redirect('/login');
16    }
17  }

```

Figura 22 - Middleware autenticazione

Il middleware "*isAuthenticated*" ha la funzione di proteggere le rotte che richiedono l'accesso autenticato da parte dell'utente. Viene utilizzato per verificare la presenza di una sessione attiva con un oggetto utente valido (*req.session.user*). Se l'utente è autenticato, l'esecuzione della richiesta prosegue normalmente tramite la funzione "*next()*". In caso contrario, l'utente viene automaticamente reindirizzato alla pagina di login (*/login*). Questo meccanismo garantisce che le risorse riservate non siano accessibili a utenti anonimi, offrendo una barriera di sicurezza fondamentale per tutte le sezioni private dell'applicazione, come *dashboard*, gestione progetti o pannelli amministrativi. L'approccio è semplice ma efficace e viene comunemente utilizzato in architetture "*session-based*" per applicazioni *Express*.

4.5.2 Accesso alla Dashboard del Progetto

```

1  * Verifica:
2  * - Esistenza del progetto
3  * - Autorizzazione dell'utente
4  * - Appartenenza al team
5  *
6  * @route GET /dashboard/:id
7  * @access Private
8  * @param {string} id - ID del progetto
9  * @renders dashboard
10 *
11 router.get('/dashboard/:id', isAuthenticated, async (req, res) => {
12   try {
13     const progetto = await Progetto.findById(req.params.id, {
14       include: [{
15         model: Team,
16         include: [{
17           model: Utente,
18           attributes: ['id', 'nome', 'cognome']
19         }]
20       }]
21     });
22   }
23   if (!progetto) {
24     return res.status(404).redirect('/home');
25   }
26
27   // Verifica che l'utente sia autorizzato a vedere il progetto
28   const isCreator = progetto.creato_da === req.session.user.id;
29   const isTeamMember = progetto.Team && progetto.Team.Utentes.some(u => u.id === req.session.user.id);
30
31   if (!isCreator && !isTeamMember) {
32     return res.status(403).redirect('/home');
33   }
34
35   res.render('dashboard', {
36     user: req.session.user,
37     progetto: progetto.get({ plain: true }),
38     progettoId: progetto.id,
39     activePage: 'dashboard'
40   });
41 } catch (error) {
42   logger.error('Errore nel caricamento della dashboard: ${error.message}');
43   res.status(500).redirect('/home');
44 }
45 });

```

Figura 23 - Accesso alla Dashboard del Progetto

La rotta GET /dashboard/:id consente agli utenti autenticati di accedere alla dashboard di un progetto specifico. È protetta dal “*middleware isAuthenticated*”, che impedisce l'accesso agli utenti non loggati, reindirizzandoli alla pagina di login.

Una volta autenticato, il server tenta di recuperare il progetto corrispondente all'ID fornito nella URL, includendo i dati del team e dei relativi membri. Se il progetto non esiste, l'utente viene reindirizzato alla home page con un codice 404.

Segue una doppia verifica di autorizzazione per garantire che l'utente abbia i diritti necessari:

È l'autore del progetto (*creato_da*). Oppure è un membro del team associato (*Team.Utentes*)

Se l'utente non rientra in nessuna di queste due categorie, l'accesso è negato (403) e viene effettuato un reindirizzamento alla home.

Se tutte le condizioni sono soddisfatte, viene renderizzata la vista dashboard, con i dati del progetto e dell'utente correntemente autenticato. In questo modo, l'accesso è limitato esclusivamente a chi ha legittima autorizzazione, garantendo protezione e riservatezza.

(Mi sono fatto aiutare dal seguente modello di intelligenza artificiale: <https://chatgpt.com/>)

4.6 authController.js

4.6.1 Registrazione di un Nuovo Utente

```

1  * @param {Object} req - Request object con i dati dell'utente:
2  *   - nome: Nome dell'utente
3  *   - cognome: Cognome dell'utente
4  *   - email: Email dell'utente
5  *   - password: Password non criptata
6  * @param {Object} res - Response object
7  *
8  * Flusso di esecuzione:
9  * 1. Validazione e formattazione input
10 * 2. Verifica email duplicata
11 * 3. Hashing password
12 * 4. Creazione utente nel database
13 * 5. Risposta con conferma
14 */
15 exports.register = async (req, res) => {
16   try {
17     let { nome, cognome, email, password } = req.body;
18     logger.info(`Tentativo di registrazione per l'email: ${email}`);
19
20     // Formatta nome e cognome
21     nome = formatName(nome);
22     cognome = formatName(cognome);
23
24     // Validazione input
25     validateEmail(email);
26     validatePassword(password);
27
28     // Verifica se l'email esiste già
29     const existingUser = await Utente.findOne({ where: { email } });
30     if (existingUser) {
31       throw new AppError(messages.email.exists, 400);
32     }
33
34     // Genera salt e hash della password
35     const salt = await bcrypt.genSalt(12);
36     const passwordHash = await bcrypt.hash(password, salt);
37
38     // Crea nuovo utente
39     const user = await Utente.create({
40       nome,
41       cognome,
42       email,
43       password_hash: passwordHash,
44       ruolo: 'user'
45     });

```

Figura 24 - Registrazione di un Nuovo Utente

La funzione *register* gestisce l'intero processo di registrazione di un nuovo utente nell'applicazione. Inizia con la validazione dei dati di input forniti dall'utente, che include la verifica del formato corretto dell'email e la validazione della password secondo i requisiti di sicurezza. Successivamente, verifica se l'email è già associata a un utente esistente nel database, evitando duplicazioni. Se l'email è disponibile, la funzione prosegue con la generazione di un *salt* e l'*hashing* della password per garantirne la sicurezza. Infine, crea un nuovo record nel database per l'utente e restituisce una risposta con la conferma della registrazione, includendo l'*ID* dell'utente appena creato. Questo processo assicura che le credenziali dell'utente siano sicure e correttamente registrate nel sistema.

4.6.2 Validazione della Password

```

1  /**
2   * Valida la password secondo i requisiti di sicurezza
3   * Verifica che la password soddisfi tutti i criteri richiesti
4   *
5   * @param {string} password - Password da validare
6   * @throws {AppError} Se la password non soddisfa uno o più requisiti:
7   * - Lunghezza minima
8   * - Presenza di maiuscole
9   * - Presenza di minuscole
10  * - Presenza di numeri
11  * - Presenza di caratteri speciali
12  */
13  const validatePassword = (password) => {
14      const errors = [];
15
16      if (password.length < securityConfig.auth.passwordMinLength) {
17          errors.push(messages.password.invalid.details.length);
18      }
19      if (!/[A-Z]/.test(password)) {
20          errors.push(messages.password.invalid.details.uppercase);
21      }
22      if (!/[a-z]/.test(password)) {
23          errors.push(messages.password.invalid.details.lowercase);
24      }
25      if (!/[0-9]/.test(password)) {
26          errors.push(messages.password.invalid.details.number);
27      }
28      if (!/[!@#$$%^&*]/.test(password)) {
29          errors.push(messages.password.invalid.details.special);
30      }
31
32      if (errors.length > 0) {
33          throw new AppError({
34              message: messages.password.invalid.message,
35              details: errors,
36              example: messages.password.invalid.example
37          }, 400);
38      }
39  };

```

Figura 25 - Validazione della password

La funzione “*validatePassword*” esegue una serie di controlli di sicurezza su una password per assicurarsi che soddisfi determinati criteri minimi. Prima di tutto, verifica che la password abbia una lunghezza sufficiente, come definito nella configurazione di sicurezza. Successivamente, controlla la presenza di almeno una lettera maiuscola, una lettera minuscola, un numero e un carattere speciale, come richiesto dalle politiche di sicurezza.

Se uno o più di questi criteri non sono soddisfatti, viene generato un errore con un messaggio che specifica quali requisiti sono stati violati.

L'errore è gestito tramite l'oggetto “*AppError*”, che fornisce dettagli su quale regola non è stata rispettata e un esempio di password corretta.

4.6.3 Validazione dell'Email

```

1  /**
2   * Valida il formato dell'email
3   * Verifica che l'email rispetti il formato standard
4   *
5   * @param {string} email - Email da validare
6   * @throws {AppError} Se l'email non rispetta il formato standard:
7   * - Deve contenere @
8   * - Deve avere un dominio valido
9   * - Non può contenere spazi
10  */
11  const validateEmail = (email) => {
12      const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
13      if (!emailRegex.test(email)) {
14          throw new AppError(messages.email.invalid, 400);
15      }
16  };

```

Figura 26 - Validazione Email

La funzione “*validateEmail*” verifica che l'indirizzo email fornito rispetti un formato valido. Utilizzando una regular “*expression*” (*regex*), la funzione controlla che l'email contenga il carattere “@” per separare il nome utente dal dominio, che il dominio sia valido (composto da un dominio di secondo livello e un dominio di primo livello, ad esempio “example.com”), e che l'email non contenga spazi. Se l'email non soddisfa questi criteri, viene generato un errore tramite un oggetto “*AppError*”, che restituisce un messaggio di errore dettagliato.

4.7 taskController.js

4.7.1 Recupero delle Task di un Progetto

```

1  /**
2   * Recupera tutte le task associate a un progetto specifico
3   * Include i dati dell'utente assegnato (nome e cognome)
4   *
5   * @param {Object} req - Request object contenente:
6   *   - projectId: ID del progetto nei parametri URL
7   * @param {Object} res - Response object
8   *
9   * Caratteristiche:
10  * - Ordinamento: Dalla più recente alla più vecchia
11  * - Include dati utente associato
12  * - Filtraggio per progetto
13  */
14  const getTasks = async (req, res) => {
15    try {
16      const { projectId } = req.params;
17
18      const tasks = await TaskJson.findAll({
19        where: { projectId },
20        include: [{
21          model: Utente,
22          attributes: ['nome', 'cognome']
23        }],
24        order: [['createdAt', 'DESC']]
25      });
26
27      res.json({
28        success: true,
29        data: tasks
30      });
31    } catch (error) {
32      logger.error('Errore durante il recupero delle task:', error);
33      res.status(500).json({
34        success: false,
35        message: 'Errore durante il recupero delle task',
36        error: error.message
37      });
38    }
39  };

```

Figura 27 - Recupero task di un progetto

La funzione “*getTasks*” gestisce la logica per recuperare tutte le task associate a un progetto specifico. Utilizzando l'ID del progetto fornito nei parametri URL (*projectId*), la funzione esegue una *query* sul database per ottenere tutte le task correlate a quel progetto, ordinandole dalla più recente alla più vecchia tramite il campo “*createdAt*”. Inoltre, per ogni task recuperata, vengono inclusi anche i dati dell'utente assegnato, specificamente il nome e il cognome dell'utente tramite un'operazione di join con il modello *Utente*. Questo permette di associare ogni task a informazioni pertinenti sull'utente che è stato incaricato di completarla. In caso di successo, la funzione restituisce i dati delle task in formato *JSON*. Se si verifica un errore durante l'esecuzione della *query*, viene generato un log dell'errore e restituito un messaggio di errore con stato 500.

4.7.2 Eliminazione di una Task

```

1  /**
2   * Elimina una task dal sistema
3   *
4   * @param {Object} req - Request object contenente:
5   *   - id: ID della task da eliminare nei parametri URL
6   * @param {Object} res - Response object
7   *
8   * Validazioni:
9   * - Verifica esistenza task
10  * - Conferma eliminazione
11  */
12  const deleteTask = async (req, res) => {
13    try {
14      const { id } = req.params;
15
16      // Verifica esistenza task
17      const task = await TaskJson.findByPk(id);
18      if (!task) {
19        return res.status(404).json({
20          success: false,
21          message: 'Task non trovata'
22        });
23      }
24
25      // Eliminazione task
26      await task.destroy();
27
28      res.json({
29        success: true,
30        message: 'Task eliminata con successo'
31      });
32    } catch (error) {
33      logger.error('Errore durante l\'eliminazione della task:', error);
34      res.status(500).json({
35        success: false,
36        message: 'Errore durante l\'eliminazione della task',
37        error: error.message
38      });
39    }
40  };

```

Figura 28 - Eliminazione di una Task

La funzione “*deleteTask*” si occupa dell’eliminazione di una task dal sistema. Il processo inizia recuperando l’ID della task da eliminare, passato nei parametri della richiesta (id). Prima di procedere con l’eliminazione, la funzione verifica che la task esista nel database mediante il metodo “*findByPk*”. Se la task non viene trovata, la funzione restituisce una risposta JSON con codice di stato 404, indicando che la task non esiste. Se la task è presente, la funzione la elimina utilizzando il metodo “*destroy*”. Una volta completata l’eliminazione, viene restituita una risposta JSON con un messaggio di successo. In caso di errore, viene loggato un messaggio di errore e restituito un codice di stato 500 con i dettagli dell’errore. Questo flusso assicura che l’eliminazione avvenga in modo sicuro e con il controllo dell’esistenza della task.

4.7.3 Spostamento task

```

1  /**
2   * Gestisce lo spostamento di una task tra colonne
3   * Aggiorna lo stato della task sul server
4   *
5   * @async
6   * @param {string} taskId - ID della task spostata
7   * @param {string} newStatus - Nuovo stato della task
8   * @throws {Error} Se lo spostamento fallisce
9   */
10 async handleTaskMove(taskId, newStatus) {
11     try {
12         const updatedTask = await this.updateTask(taskId, { status: newStatus });
13
14         // Aggiorna la task localmente
15         const taskIndex = this.tasks.findIndex(t => t.id === taskId);
16         if (taskIndex !== -1) {
17             this.tasks[taskIndex] = updatedTask;
18             this.renderTasks();
19         }
20
21         // Emetti l'evento di aggiornamento
22         this.socket.emit('taskUpdated', updatedTask);
23     } catch (error) {
24         this.renderTasks();
25         throw error;
26     }
27 }

```

Figura 29 - Spostamento Task

Il metodo “*handleTaskMove*” gestisce lo spostamento di una task tra colonne aggiornandone lo stato sia localmente che sul server. È una funzione asincrona che, dato l’ID della task e il nuovo stato (“*newStatus*”), invia una richiesta di aggiornamento al server tramite “*updateTask*”. Se l’operazione ha successo, aggiorna la task nella lista locale e ne rinfresca la visualizzazione con “*renderTasks*”. Infine, notifica gli altri client emettendo un evento “*taskUpdated*” attraverso un *socket*. In caso di errore, ripristina comunque l’interfaccia chiamando “*renderTasks*” e rilancia l’eccezione.

(Mi sono fatto aiutare dal seguente modello di intelligenza artificiale: <https://chatgpt.com/>)

5 Test

5.1 Protocollo di test

Test Case: Riferimento:	TC-001 REQ-001	Nome:	Registrazione utente
Descrizione:	Verifica che un utente si possa registrare nell'applicativo		
Prerequisiti:	<ul style="list-style-type: none"> - 		
Procedura:	<ol style="list-style-type: none"> 1. Inserire I dati richiesti (nome, cognome, mail e password) 2. Spuntare il checkbox per accettare I termini e le condizioni 3. Cliccare il bottone con voce "Registrati" 		
Risultati attesi:	Verrà mostrato un messaggio dell'avvenuta registrazione e si verrà renderizzati alla pagina di login.		

Test Case: Riferimento:	TC-002 REQ-002	Nome:	Login utente
Descrizione:	Verifica che un utente si possa effettuare l'accesso all'applicativo		
Prerequisiti:	<ul style="list-style-type: none"> • Essersi registrati 		
Procedura:	<ol style="list-style-type: none"> 1. Inserire i dati richiesti (email e password). 2. Cliccare il bottone con voce "Login". 		
Risultati attesi:	Verrà effettuato l'accesso all'applicativo e sarà renderizzata la pagina home.		

Test Case: Riferimento:	TC-003 REQ-003	Nome:	User e Admin
Descrizione:	Verifica che ci siano due tipi di utente		
Prerequisiti:	<ul style="list-style-type: none"> Aver già testato il test case numero 002. 		
Procedura:	<ol style="list-style-type: none"> Eseguire il login con l'utente admin di default (credenziali: admin@example.com Admin\$00). Cliccare il bottone con voce "Login". 		
Risultati attesi:	Verrà effettuato l'accesso all'applicativo e sarà renderizzata la pagina home.		

Test Case: Riferimento:	TC-004 REQ-004	Nome:	Creazione Team
Descrizione:	Verifica che si possano creare team di lavoro		
Prerequisiti:	<ul style="list-style-type: none"> Aver eseguito l'accesso nella parte admin dell'applicativo. 		
Procedura:	<ol style="list-style-type: none"> Recarsi nella tab con voce "Team". Cliccare il pulsante con voce "+ Nuovo Team". Inserire i dati obbligatori richiesti nel form (nome team e membri del team). 		
Risultati attesi:	Verrà mostrato un messaggio di successo e creato il team di lavoro.		

Test Case: Riferimento:	TC-005 REQ-004	Nome:	Creazione Progetti
Descrizione:	Verifica che si possano creare dei progetti		
Prerequisiti:	<ul style="list-style-type: none"> Aver eseguito l'accesso nella parte admin dell'applicativo. 		
Procedura:	<ol style="list-style-type: none"> Recarsi nella tab con voce "Progetti". Cliccare il pulsante con voce "+ Nuovo Progetto". Inserire i dati obbligatori richiesti nel form (nome progetto, descrizione) 		
Risultati attesi:	Verrà mostrato un messaggio di successo e creato il progetto.		

Test Case: Riferimento:	TC-006 REQ-005	Nome:	Creazione Task
------------------------------------	-------------------	--------------	-----------------------

Descrizione:	Verifica che si possano creare delle task
Prerequisiti:	<ul style="list-style-type: none"> Aver eseguito l'accesso nella parte user dell'applicativo con un utente con almeno un progetto assegnato.
Procedura:	<ol style="list-style-type: none"> Cliccare il pulsante "Visualizza". Cliccare il pulsante con voce "Aggiungi Task". Inserire i dati obbligatori richiesti nel form (titolo, peso e priorità) Cliccare il pulsante con voce "Crea Task".
Risultati attesi:	Verrà mostrato un messaggio di successo e la task verrà creata correttamente.

Test Case: Riferimento:	TC-007 REQ-005	Nome:	Filtrare Task
Descrizione:	Verifica che si possano filtrare le task		
Prerequisiti:	<ul style="list-style-type: none"> Aver eseguito l'accesso nella parte user dell'applicativo con un utente con almeno un progetto assegnato e essere all'interno di un progetto con almeno una task creata. 		
Procedura:	<ol style="list-style-type: none"> Scegliere la priorità delle task da visualizzare tramite il select con voce "Filtra per priorità". Scegliere l'assegnatario delle task da visualizzare tramite il select con voce "Filtra per assegnatario". 		
Risultati attesi:	Le task verranno filtrate nel modo corretto.		

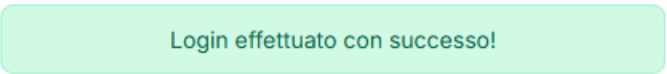
Test Case: Riferimento:	TC-008 REQ-006	Nome:	Assegnare Task
Descrizione:	Verifica che tramite un utente admin si possano assegnare le task a un qualsiasi membro del team di quel progetto.		
Prerequisiti:	<ul style="list-style-type: none"> Aver eseguito l'accesso nella parte admin dell'applicativo e visualizzare una dashboard di un progetto con almeno una task creata. 		
Procedura:	<ol style="list-style-type: none"> Cliccare su una task Scegliere l'assegnatario della task tramite il select all'interno del form. Cliccare il bottone con voce "Crea Task". 		
Risultati attesi:	La task verrà assegnata correttamente all'utente selezionato		

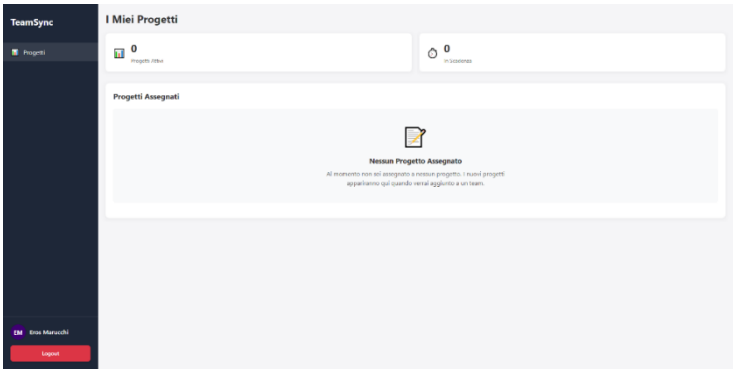
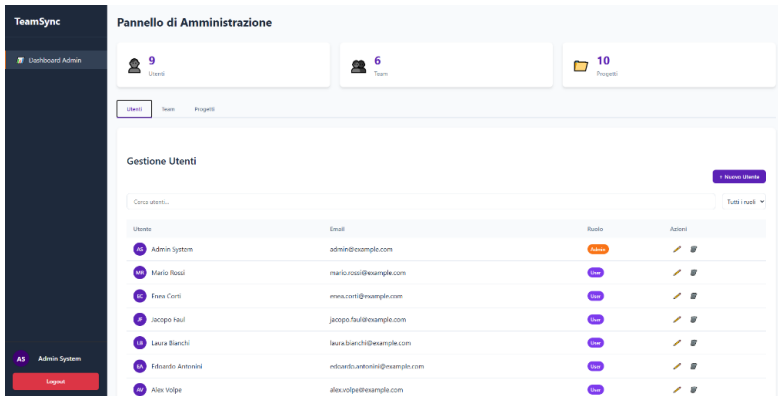
Test Case: Riferimento:	TC-009 REQ-006	Nome:	Cambiare stato alle task
Descrizione:	Verifica che tramite un utente admin si possano spostare le task di un qualsiasi membro del team di quel progetto, da una colonna all'altra.		
Prerequisiti:	<ul style="list-style-type: none"> Aver eseguito l'accesso nella parte admin dell'applicativo e visualizzare una dashboard di un progetto con almeno una task creata. 		
Procedura:	<ol style="list-style-type: none"> Eseguire un drag and drop su una task assegnata ad un utente che non sia l'utente con cui si ha eseguito il login, per spostare la task da una colonna all'altra. 		
Risultati attesi:	La task si potrà spostare per far avvenire il cambiamento di stato.		

Test Case: Riferimento:	TC-010 REQ-007	Nome:	Presa in carico delle task
Descrizione:	Aver es		
Prerequisiti:	<ul style="list-style-type: none"> Aver eseguito l'accesso nella parte user dell'applicativo con un utente con almeno un progetto assegnato e essere all'interno di un progetto con almeno una task creata non assegnata a nessuno. 		
Procedura:	<ol style="list-style-type: none"> Cliccare su una task non assegnata a nessuno. Cliccare il checkbox con voce "Assegna a me stesso" Cliccare il bottone con voce "Crea Task". 		
Risultati attesi:	La task si potrà spostare per far avvenire il cambiamento di stato.		

5.2 Risultati test

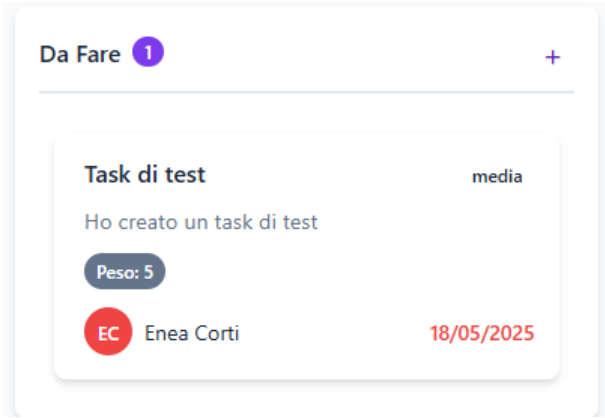
Test Case	Risultato	Descrizione	Data Test
001	Passato	<p>Risultato atteso: Deve essere possibile registrarsi all'interno dell'applicativo.</p> <p>Risultato effettivo: È possibile registrarsi all'interno dell'applicativo.</p> <p>Foto:</p> 	09.05.25


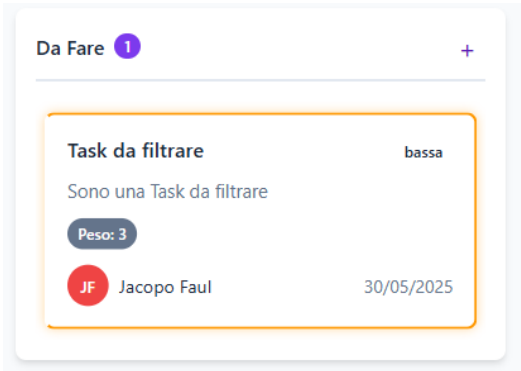
Test Case	Risultato	Descrizione	Data Test
002	Passato	<p>Risultato atteso: Deve essere possibile eseguire il login nell'applicativo</p> <p>Risultato effettivo: È possibile effettuare il login all'interno dell'applicativo</p> <p>Foto:</p> 	09.05.25

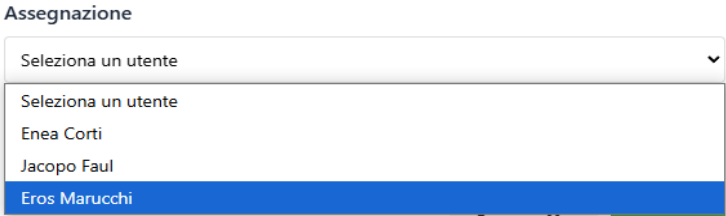
Test Case	Risultato	Descrizione	Data Test
003	Passato	<p>Risultato atteso:</p> <p>Deve essere possibile fare l'accesso con due tipi di utente diversi (user e admin).</p> <p>Risultato effettivo:</p> <p>È possibile effettuare il login con entrambi i tipi di utente.</p> <p>Foto:</p> <p>User:</p>  <p>Admin:</p> 	09.05.25

Test Case	Risultato	Descrizione	Data Test
004	Passato	<p>Risultato atteso: Deve essere possibile team di lavoro tramite un utente admin</p> <p>Risultato effettivo: È possibile creare un team di lavoro tramite un utente admin</p> <p>Foto:</p>	12.05.25

Test Case	Risultato	Descrizione	Data Test
005	Passato	<p>Risultato atteso: Deve essere possibile team di lavoro tramite un utente admin</p> <p>Risultato effettivo: È possibile creare un team di lavoro tramite un utente admin</p> <p>Foto:</p>	12.05.25

Test Case	Risultato	Descrizione	Data Test
006	Passato	<p>Risultato atteso: Deve essere possibile creare delle task all'interno di un progetto.</p> <p>Risultato effettivo: È possibile creare delle task all'interno di un progetto.</p> <p>Foto:</p> 	12.05.25

Test Case	Risultato	Descrizione	Data Test
007	Passato	<p>Risultato atteso: Deve essere filtrare le task</p> <p>Risultato effettivo: È possibile filtrare le task</p> <p>Foto:</p> <p>Filtro:</p>  <hr/> <p>Task:</p> 	13.05.25

Test Case	Risultato	Descrizione	Data Test
008	Passato	<p>Risultato atteso: Tramite utente admin deve essere possibile assegnare una task a un qualsiasi membro del team.</p> <p>Risultato effettivo: È possibile assegnare una task a un qualsiasi membro del team tramite un utente admin.</p> <p>Foto</p> 	13.05.25

Test Case	Risultato	Descrizione	Data Test
009	Passato	<p>Risultato atteso: Tramite utente admin deve essere possibile spostare una task di un qualsiasi utente di un team da una colonna all'altra.</p> <p>Risultato effettivo: È possibile spostare una task di un qualsiasi utente di un team da una colonna all'altra tramite un utente admin.</p> <p>Foto:</p> 	13.05.25

Test Case	Risultato	Descrizione	Data Test
010	Passato	<p>Risultato atteso: Tramite un utente user deve essere possibile prendere in carico una task non assegnata.</p> <p>Risultato effettivo: È possibile prendere in carico una task non assegnata.</p> <p>Foto:</p> <p>Assegnazione</p> <p><input checked="" type="checkbox"/> Assegna a me stesso</p>	13.05.25

5.2.1 Riassunto Test

ID Test-Case	Risultato	Commento / Note	Data
TC-001	Passato	-	09.05.25
TC-002	Passato	-	09.05.25
TC-003	Passato	-	09.05.25
TC-004	Passato	-	12.05.25
TC-005	Passato	-	12.05.25
TC-006	Passato	-	12.05.25
TC-007	Passato	-	13.05.25
TC-008	Passato	-	13.05.25
TC-009	Passato	-	13.05.25
TC-010	Passato	-	13.05.25

5.3 Unit test

Nel mio progetto ho implementato i test unitari, in quanto rappresentavano un requisito tecnico stabilito. Per questo scopo ho utilizzato Jest come framework di testing. Ho realizzato sette file di test, per un totale di 72 test, concentrandomi sulle funzionalità più rilevanti dell'applicativo. In accordo con il mio formatore, è stato chiarito che non era necessario coprire tutte le funzionalità, ma solo quelle considerate più significative. Di conseguenza, ho selezionato e testato i moduli che ritenevo fondamentali per il corretto funzionamento del progetto. Tutti i test da me sviluppati hanno un esito positivo.

Il comando per eseguire tutti i test è: `npm test -- --config unit-test/config/jest.config.js`.

(I test sono stati revisionati da: <https://chatgpt.com/>)

5.3.1 Configurazione

```

1  module.exports = {
2    testEnvironment: 'jsdom',
3    transform: {
4      '^.+\\.jsx?$': ['babel-jest', { presets: ['@babel/preset-env'] }]
5    },
6    moduleFileExtensions: ['js', 'json'],
7    roots: ['../'],
8    transformIgnorePatterns: [
9      '/node_modules/(?!(sequelize|uuid))'
10   ],
11   verbose: true,
12 };

```

Figura 30 - Configurazione Jest

Il file `jest.config.js` serve a configurare il comportamento dei test. Imposta `jsdom` come ambiente di test, utile per simulare il DOM. I file `.js` vengono trasformati con `babel-jest` usando il preset `@babel/preset-env`, così da supportare le funzionalità moderne di *JavaScript*. Specifica quali estensioni di file considerare (`js`, `json`), da dove partire nella ricerca dei test (`../`), e quali moduli ignorare o includere nella trasformazione. L'opzione `verbose: true` abilita un output dettagliato dei test eseguiti.

5.3.2 Sintassi

```

1  describe('getUserName', () => {
2    beforeEach(() => {
3      global.fetch = jest.fn();
4    });
5    it('restituisce il nome completo se utente trovato', async () => {
6      global.fetch.mockResolvedValue({ ok: true, json: async () => ({ nome: 'Anna', cognome: 'Verdi' }) });
7      const name = await getUserName('user456');
8      expect(name).toBe('Anna Verdi');
9    });
10   it('restituisce "Utente" se utente non trovato', async () => {
11     global.fetch.mockResolvedValue({ ok: false });
12     const name = await getUserName('user404');
13     expect(name).toBe('Utente');
14   });
15   it('restituisce "Utente" se fetch lancia errore', async () => {
16     global.fetch.mockRejectedValue(new Error('Errore'));
17     const name = await getUserName('user500');
18     expect(name).toBe('Utente');
19   });
20 });

```

Figura 31 - Sintassi unit test

Il codice definisce una serie di test unitari per la funzione “*getUserName*”, raggruppati all'interno del blocco “*describe*”. La funzione “*beforeEach*” viene utilizzata per reimpostare “*global.fetch*” come funzione simulata prima di ogni test, così da poter controllare le risposte della fetch. Il primo test verifica che, se la richiesta ha esito positivo e restituisce un oggetto con “*nome*” e “*cognome*”, la funzione ritorni il nome completo. Il secondo test controlla che, in caso di risposta non valida (“*ok: false*”), venga restituita la stringa “*Utente*”. Il terzo test simula un errore lanciato da “*fetch*”, ad esempio per problemi di rete, e verifica che anche in questo caso il risultato sia “*Utente*”. In sintesi, i test coprono i casi principali in cui la funzione deve comportarsi correttamente sia in presenza di dati validi, sia in caso di errore o risposta non valida.

5.4 Risultati

5.4.1 utils.test.js

Ecco i test e i risultati relativi al file *utils.js*:

```

PASS unit-test/utils.test.js
  getColorByPriority
    ✓ restituisce il colore corretto per priorità alta (5 ms)
    ✓ restituisce il colore corretto per priorità media
    ✓ restituisce il colore corretto per priorità bassa (1 ms)
    ✓ restituisce bianco per priorità sconosciuta
  getInitials
    ✓ restituisce le iniziali corrette se utente trovato (3 ms)
    ✓ restituisce XX se utente non trovato
    ✓ restituisce XX se fetch lancia errore (1 ms)
  getUserName
    ✓ restituisce il nome completo se utente trovato (1 ms)
    ✓ restituisce "Utente" se utente non trovato (1 ms)
    ✓ restituisce "Utente" se fetch lancia errore (1 ms)
  updateTaskCounters
    ✓ aggiorna i contatori delle colonne Kanban (35 ms)

```

Figura 32 - unit test utils.js

5.4.2 project.test.js

Ecco i test e i risultati relativi al file project.js:

```
PASS unit-test/projects.test.js
ProjectManager
  ✓ apre il modale per nuovo progetto (39 ms)
  ✓ chiude il modale e resetta il form (7 ms)
  ✓ valida il form: nome vuoto non valido (7 ms)
  ✓ valida il form: nome valido (4 ms)
  ✓ aggiorna la riga progetto nella tabella (9 ms)
  ✓ mostra un messaggio con showMessage (4 ms)
  ✓ filtra i progetti tramite ricerca (9 ms)
  ✓ gestisce la submit del form con dati validi (7 ms)
```

Figura 33 - unit test project.js

5.4.3 admin.test.js

Ecco i test e i risultati relativi al file admin.js:

```
PASS unit-test/admin.test.js
Admin Manager principali
  ✓ AuthManager: inizializza e setta event listener logout (30 ms)
  ✓ TeamManager: inizializza e setta riferimenti DOM (17 ms)
  ✓ ProjectManager: inizializza e setta riferimenti DOM (43 ms)
  ✓ UserManager: inizializza e setta riferimenti DOM (19 ms)
  ✓ SearchManager: inizializza senza errori (6 ms)
```

Figura 34 - unit test admin.js

5.4.4 users.test.js

Ecco i test e i risultati relativi al file users.js:

```
PASS unit-test/users.test.js
UserManager
  ✓ apre il modale per nuovo utente (36 ms)
  ✓ chiude il modale senza modifiche (8 ms)
  ✓ chiude il modale con modifiche e conferma (5 ms)
  ✓ gestisce la submit del form per nuovo utente (6 ms)
  ✓ gestisce la submit del form per modifica utente (4 ms)
  ✓ gestisce errore nella submit del form (4 ms)
  ✓ gestisce la modifica di un utente (initEditButtons) (6 ms)
  ✓ gestisce la cancellazione di un utente (initDeleteButtons) (5 ms)
  ✓ gestisce errore nella cancellazione utente (13 ms)
```

Figura 35 - unit test users.js

5.4.5 modal.test.js

Ecco i test e i risultati relativi al file modal.js:

```
PASS unit-test/modal.test.js
TaskModal
  ✓ resetForm resetta il form e lo stato (41 ms)
  ✓ close nasconde il modal e resetta il form (8 ms)
  ✓ populateUserSelect popola la select con i membri del team (10 ms)
  ✓ loadTeamMembers restituisce i membri del team (mock fetch) (5 ms)
  ✓ loadTeamMembers gestisce errori e restituisce array vuoto (5 ms)
  ✓ getProject restituisce i dati del progetto (mock fetch) (47 ms)
  ✓ getProject lancia errore se fetch fallisce (44 ms)
  ✓ openForEdit (admin) popola il form e la select (18 ms)
  ✓ openForEdit (user) seleziona assegnata_a_me se userId coincide (4 ms)
```

Figura 36 - unit test modal.js

5.4.6 authController.test.js

Ecco i test e i risultati relativi al file authController.js:

```
PASS unit-test/authController.test.js
authController.register
  ✓ registra un nuovo utente con dati validi (9 ms)
  ✓ risponde con errore se la validazione fallisce (1 ms)
  ✓ risponde con errore se l'email è già registrata (1 ms)
authController.login
  ✓ fa il login con credenziali valide (1 ms)
  ✓ risponde con errore se la validazione fallisce (1 ms)
  ✓ risponde con errore se utente non trovato
  ✓ risponde con errore se la password è errata (1 ms)
  ✓ risponde con errore 500 in caso di eccezione (43 ms)
authController.logout
  ✓ fa il logout con successo
  ✓ risponde con errore se destroy fallisce (1 ms)
  ✓ risponde con errore 500 in caso di eccezione (1 ms)
```

Figura 37 - unit test modal.js

5.4.7 taskController.test.js

Ecco i test e i risultati relativi al file taskController.js:

```
PASS unit-test/taskController.test.js
taskController.createTask
  ✓ crea una task con dati validi (8 ms)
  ✓ risponde con errore se la validazione fallisce
  ✓ risponde con errore se utente non admin assegna task ad altri (1 ms)
  ✓ risponde con errore 500 in caso di eccezione (45 ms)
taskController.getTasks
  ✓ restituisce le task di un progetto
  ✓ risponde con errore se manca projectId
  ✓ risponde con errore 500 in caso di eccezione (2 ms)
taskController.updateTask
  ✓ aggiorna una task esistente (1 ms)
  ✓ risponde con errore se la task non esiste
  ✓ risponde con errore se utente non admin modifica task non sua (1 ms)
  ✓ risponde con errore se la validazione fallisce (1 ms)
  ✓ risponde con errore 500 in caso di eccezione (2 ms)
taskController.deleteTask
  ✓ elimina una task esistente (1 ms)
  ✓ risponde con errore se la task non esiste
  ✓ risponde con errore 500 in caso di eccezione (1 ms)
taskController.moveTask
  ✓ sposta una task in uno stato valido
  ✓ risponde con errore se status non valido
  ✓ risponde con errore se la task non esiste
  ✓ risponde con errore 500 in caso di eccezione (2 ms)
```

Figura 38 - unit test taskController.js

5.4.8 Riassunto unit test

Ecco il riassunto di tutti gli unit test e i relativi risultati:

```
Test Suites: 7 passed, 7 total
Tests:      72 passed, 72 total
Snapshots: 0 total
Time:      2.792 s, estimated 3 s
Ran all test suites.
```

Figura 39 - riassunto unit test

5.5 Mancanze e limitazioni conosciute

5.5.1 Mancanze

Una mancanza del mio applicativo è quella che l'utente admin possa modificare il nome delle quattro colonne degli stati delle *task*.

5.5.2 Limitazioni

Una limitazione attuale del mio applicativo riguarda la gestione dei colori assegnati alle task, una funzionalità prevista dalle specifiche del QDC. Attualmente, è possibile assegnare un colore a ciascuna task individualmente. Tuttavia, una soluzione probabilmente più efficace sarebbe stata quella di associare un colore a ciascun utente, in modo che tutti le task creati da quell'utente condividessero automaticamente lo stesso colore.

Per implementare questa soluzione, sarebbe stato necessario aggiungere una colonna aggiuntiva nella tabella Utente del database, in cui salvare il colore scelto dall'utente. Tale scelta avrebbe potuto essere effettuata, ad esempio, nella sezione "Profilo" dell'applicativo.

5.6 Implementazioni extra

5.6.1 Gestione profilo utente

Ho implementato la sezione per la gestione del profilo utente da parte di un utente user. Ora un utente user può andare nella sezione utente e modificare: nome, cognome, *email* e *password*. Ovviamente è tutto controllato per far sì che vengano inseriti solo dati corretti e non ci siano duplicati.

Figura 40 - Form gestione profilo utente

5.6.2 Invio delle email

Ho implementato la funzionalità di invio email all'interno dell'applicativo. A tal fine, ho creato l'indirizzo email teamsyncbot@gmail.com, che è stato integrato nel sistema.

Attualmente, vengono inviate email automatiche in tre casi principali:

Registrazione: al completamento della registrazione, l'utente riceve una mail all'indirizzo specificato nel campo "Email".

Assegnazione a un progetto: quando un utente viene aggiunto a un nuovo progetto, riceve una mail contenente tutte le informazioni rilevanti del progetto.

Eliminazione account: se un utente viene rimosso da un amministratore, il sistema invia una notifica via email per informarlo della cancellazione dell'account.



Figura 41 - Mail registrazione

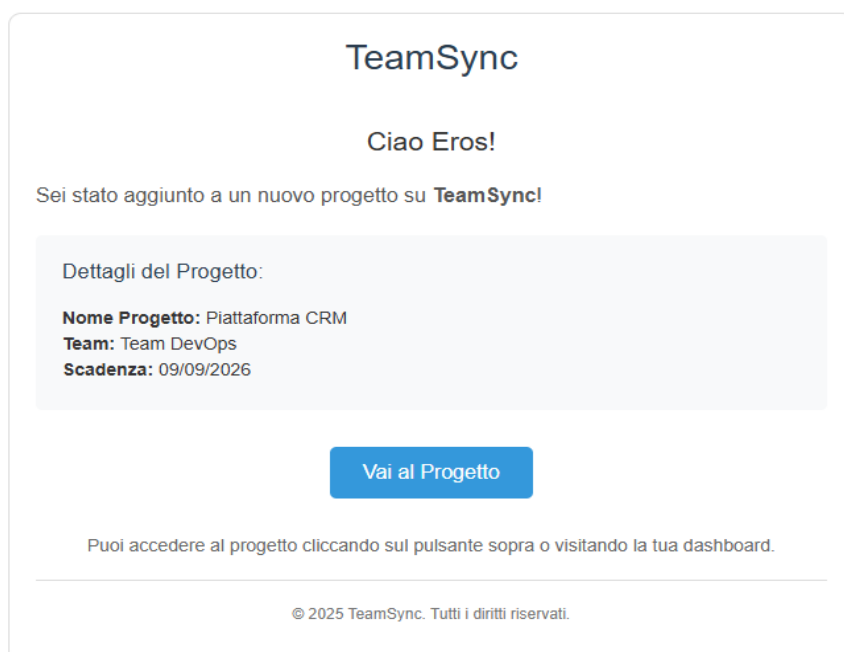


Figura 42 - Mail progetto



Figura 43 - Mail cancellazione account

5.6.3 Reset della password se dimenticata

Ritengo che la possibilità per un utente di reimpostare la propria password in caso di smarrimento sia una funzionalità fondamentale.

Attualmente, se un utente dimentica la password, può avviare la procedura di reset cliccando su "Password dimenticata?" nel form di login. Verrà quindi richiesto di inserire l'indirizzo email utilizzato durante la registrazione.

Se l'indirizzo fornito è valido, il sistema invierà una mail contenente un codice di verifica a 4 cifre. Questo codice dovrà essere inserito nel form successivo (il codice ha una validità massima di 5 minuti. Oppure è possibile richiedere più codici cliccando "Richiedi un nuovo codice"). Una volta confermata la correttezza del codice, l'utente potrà impostare una nuova password compilando i due campi dedicati.



Figura 44 - Mail codice di reset password

TeamSync

Inserisci e conferma la tua nuova password per completare il reset.

Nuova password

Inserisci la nuova password

Conferma password

Conferma la nuova password

Cambia password

Figura 45 - Form reset password

5.6.4 Form di contatto

Ho aggiunto una sezione “*Contatti*” all'interno dell'applicativo. Quando un utente accede a questa sezione, viene mostrato un *form* che consente di inviare un messaggio all'amministrazione dell'app, utile per segnalare eventuali problemi o richiedere supporto.

Nome

Il tuo nome

Email

La tua email

Soggetto

Seleziona un soggetto

Messaggio

Scrivi il tuo messaggio...

Annulla **Invia**

Figura 46 - Form di contatto

5.6.5 Calendario attività

Ho introdotto una nuova sezione “Calendario” all’interno dell’applicativo. Questo permette agli utenti di visualizzare in modo chiaro tutte le scadenze relative alle proprie task e ai progetti assegnati. Inoltre, cliccando su una task o su un progetto nel calendario, si viene reindirizzati direttamente alla relativa pagina di dettaglio.

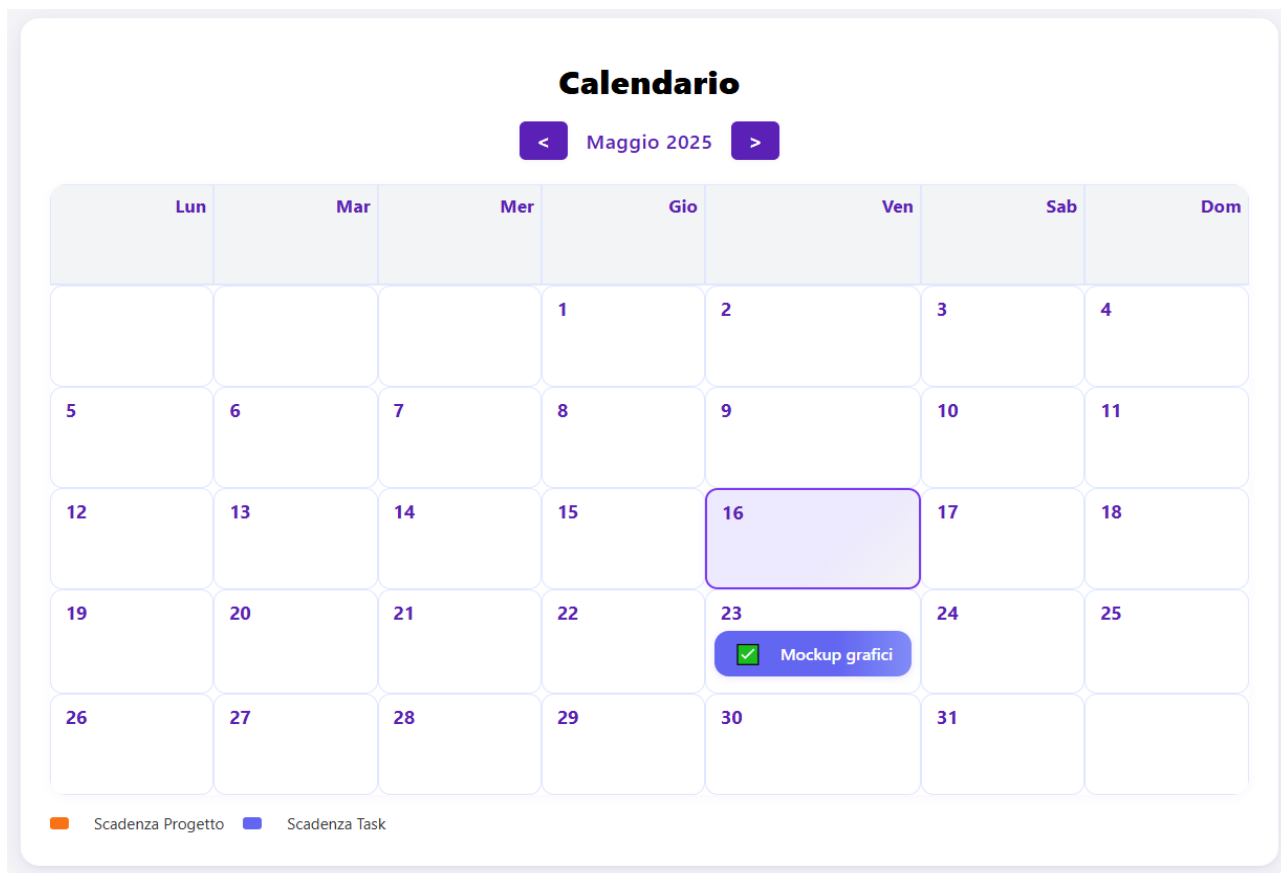


Figura 47 - Calendario

6 Consuntivo

Qui è riportato il diagramma di Gantt consuntivo del progetto, che rappresenta l'effettivo svolgimento delle attività durante lo sviluppo:

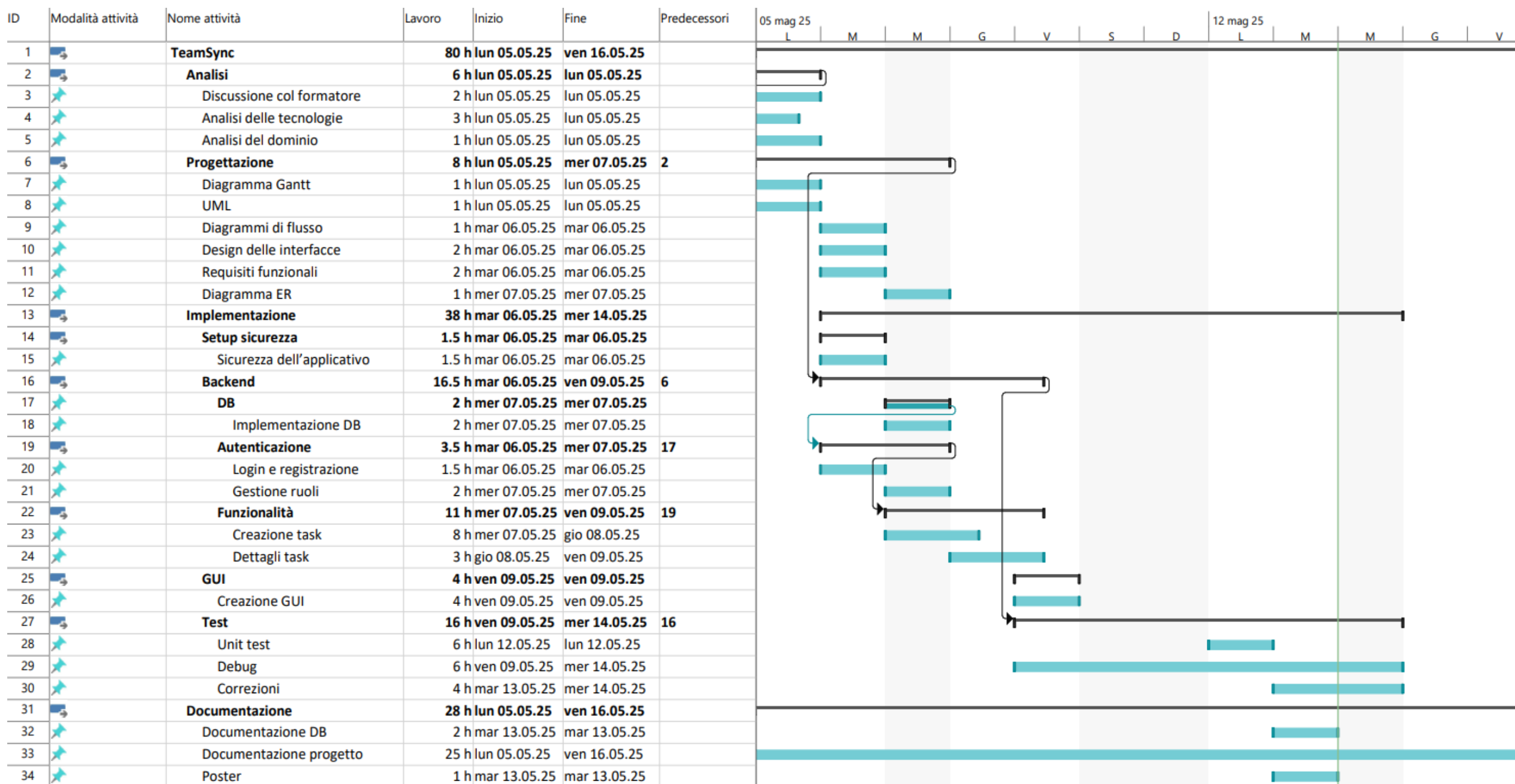


Figura 48 - Gantt consuntivo

6.1 Confronto pianificazione preventiva e consuntiva

Una delle principali difficoltà che ho incontrato durante lo sviluppo del progetto è stata il tempo impiegato nel debugging e nella creazione dei test per le funzioni implementate. Spesso mi sono trovato a dover individuare e correggere errori che rallentavano il progresso, rendendo il processo più complesso del previsto.

Riflettendo sulla pianificazione, mi sono reso conto di aver commesso alcuni errori nella sequenza delle attività previste. Durante la stesura del diagramma di Gantt, avrei potuto strutturare meglio l'ordine delle operazioni. In diverse occasioni, mi sono trovato nella situazione in cui, secondo il piano, avrei dovuto completare una determinata attività, ma in realtà era necessario svolgerne prima un'altra per poter procedere correttamente. Questo ha causato alcuni problemi organizzativi e mi ha costretto a rivedere e riadattare alcune fasi del lavoro.

Nonostante queste difficoltà, sono riuscito a portare a termine il progetto con successo. Questa esperienza mi ha insegnato quanto sia fondamentale una pianificazione accurata ma allo stesso tempo flessibile, in grado di adattarsi alle esigenze reali dello sviluppo. In futuro, cercherò di dedicare maggiore attenzione alla fase di progettazione e organizzazione, per evitare di incorrere in problematiche simili.

7 Conclusioni

TeamSync ha un grande potenziale nel migliorare l'efficienza e la collaborazione all'interno dei team, semplificando la gestione dei progetti e delle attività attraverso un'interfaccia visiva e interattiva. Il sistema consente di monitorare il progresso delle task in modo chiaro e intuitivo, anche per gli utenti con competenze tecniche limitate. La sua struttura consente una gestione dinamica e flessibile dei progetti, adattandosi alle diverse esigenze operative di team di varia dimensione. Con l'implementazione di funzionalità come la personalizzazione delle task tramite colori e la gestione delle priorità, *TeamSync* non solo ottimizza l'organizzazione delle attività, ma stimola anche la produttività e la comunicazione tra i membri del team.

Un altro aspetto fondamentale del progetto è la separazione dei ruoli, con la possibilità di assegnare permessi differenti tra amministratori e utenti. Gli amministratori hanno la possibilità di gestire e configurare progetti, team e task, mentre gli utenti possono concentrarsi sulle proprie attività in modo autonomo. Questa gestione gerarchica dei permessi consente una chiara suddivisione delle responsabilità, evitando conflitti e migliorando la collaborazione all'interno del team. Inoltre, l'applicativo facilita l'assegnazione di priorità e scadenze, dando ai membri del team la visibilità necessaria per organizzare al meglio il proprio lavoro e rispettare le scadenze.

TeamSync non è solo uno strumento per la gestione delle task, ma anche una piattaforma che promuove l'inclusività e l'accessibilità. La sua interfaccia *user-friendly* e il sistema di personalizzazione permettono agli utenti di concentrarsi sull'aspetto creativo e organizzativo del lavoro, senza doversi preoccupare di strumenti complessi. Questa caratteristica rende *TeamSync* ideale per un'ampia gamma di utenti, da team di sviluppo software a gruppi educativi o professionisti che desiderano un sistema di gestione del lavoro agile e semplice da usare.

7.1 Sviluppi futuri

Una possibile miglioria per *TeamSync* potrebbe essere l'introduzione di funzionalità che ottimizzano ulteriormente la collaborazione in tempo reale. Sebbene sia già possibile lavorare insieme su un progetto simultaneamente, l'aggiunta di strumenti avanzati per la gestione delle notifiche in tempo reale potrebbe migliorare l'esperienza. Ad esempio, notifiche istantanee per aggiornamenti o modifiche da parte di altri membri del team consentirebbero agli utenti di restare sempre aggiornati sugli sviluppi del progetto, senza dover monitorare manualmente le modifiche. Inoltre, l'inclusione di uno strumento di chat o commenti in-linea permetterebbe una comunicazione diretta e immediata, facilitando il confronto sulle task e il *feedback* tra i membri del team, senza dover uscire dall'applicazione.

Un'altra possibile estensione potrebbe riguardare l'introduzione di report di avanzamento delle task, che fornirebbero una visione chiara dei progressi e permetterebbero di identificare rapidamente eventuali blocchi o ritardi. Questi strumenti di monitoraggio delle performance potrebbero essere personalizzabili, consentendo ad ogni team di adattarli alle proprie esigenze specifiche. Inoltre, sarebbe utile integrare *TeamSync* con altre piattaforme di project management o strumenti esterni già utilizzati dal team, facilitando così la sincronizzazione del lavoro e migliorando l'efficienza complessiva.

Infine, per migliorare ulteriormente l'interazione e la produttività, l'implementazione di funzionalità di personalizzazione avanzata della dashboard e delle task renderebbe l'applicazione ancora più adattabile alle esigenze specifiche di ciascun team. Con queste migliorie, *TeamSync* potrebbe evolversi in una piattaforma ancora più completa e versatile, offrendo un'esperienza di lavoro collaborativo più fluida ed efficiente.

7.2 Considerazioni personali

Questo progetto è stato un'esperienza estremamente gratificante e formativa, e sono molto soddisfatto del risultato finale. Durante lo sviluppo, ho avuto l'opportunità di crescere sia a livello tecnico che personale. Lavorare su un progetto così interessante e stimolante mi ha permesso di affrontare diverse sfide che hanno ampliato le mie competenze, ma soprattutto mi ha dato la possibilità di mettermi alla prova e migliorare le mie capacità di *problem-solving* e gestione del progetto. Sin dall'inizio, l'obiettivo di sviluppare un'applicazione in grado di gestire team e progetti in modo agile mi ha entusiasmato, e la realizzazione di ogni funzionalità mi ha motivato ad andare avanti.

Dal punto di vista tecnico, ho avuto l'opportunità di lavorare con Node.js per il backend e JavaScript per il *frontend*, tecnologie che conosco abbastanza bene, ma che mi hanno permesso di approfondire aspetti nuovi, come l'implementazione di *unit* test automatizzati tramite *Jest*. L'adozione di *Jest* mi ha permesso di migliorare la qualità del codice e di sviluppare una consapevolezza maggiore sull'importanza dei test nel ciclo di vita di un'applicazione. La scrittura di test è stata una fase fondamentale del progetto, che mi ha aiutato a garantire la robustezza del sistema, evitando regressioni e migliorando la sicurezza del software.

Inoltre, un altro aspetto che ho affinato durante lo sviluppo del progetto è stata la gestione del tempo. Anche se non ci sono riuscito al 100% a rispettare il piano di lavoro è stato un aspetto cruciale per completare il progetto nei tempi stabiliti. Questo mi ha permesso di sviluppare una maggiore capacità di pianificazione, organizzazione e gestione delle risorse, competenze che si rivelano fondamentali anche in contesti lavorativi reali. La capacità di bilanciare il tempo tra le diverse fasi dello sviluppo, dalla scrittura del codice alla fase di test, è stata determinante per il buon esito del progetto.

Guardando al futuro, questo progetto rappresenta un'importante opportunità di crescita professionale. Mi ha permesso di mettere in pratica le conoscenze acquisite durante tutti i quattro anni di scuola e di affrontare nuove sfide che mi hanno spinto a migliorarmi ulteriormente. Le competenze tecniche acquisite, insieme all'intera gestione di un progetto, mi hanno dato una solida preparazione per affrontare future opportunità lavorative. Questo progetto ha arricchito il mio bagaglio professionale, rafforzando le mie capacità sia a livello tecnico che organizzativo, e sono convinto che questa esperienza mi abbia preparato al meglio per le sfide che mi aspettano nel mondo del lavoro.

8 Glossario

Termine	Significato
AJAX	Tecnica di sviluppo web che consente di aggiornare parti di una pagina web senza ricaricarla completamente, utilizzando richieste HTTP asincrone.
Attacchi CSRF	(Cross-Site Request Forgery) Attacchi che inducono un utente autenticato a compiere azioni non volute su un'applicazione web.
Backend	Parte di un'applicazione web che gestisce la logica, l'elaborazione dei dati e la comunicazione con il database.
CORS	(Cross-Origin Resource Sharing) Meccanismo che controlla l'accesso alle risorse web da domini diversi rispetto a quello dell'applicazione.
DoS	(Denial of Service) Attacco che mira a rendere un servizio non disponibile sovraccaricando il sistema con richieste.
Express	Framework web minimalista per Node.js, utilizzato per costruire applicazioni web e API.
Framework	Insieme di strumenti e librerie che forniscono una struttura di base per sviluppare applicazioni software.
Frontend	Parte visibile di un'applicazione web con cui l'utente interagisce, sviluppata con tecnologie come HTML, CSS e JavaScript.
Handlebars	Motore di template JavaScript che consente di generare HTML in modo dinamico a partire da dati.
Hashing	Tecnica crittografica che trasforma dati in una stringa di lunghezza fissa, usata per proteggere password e verificare integrità.
HTTP	(HyperText Transfer Protocol) Protocollo usato per la comunicazione tra client e server web.
HTTPS	Variante sicura di HTTP, in cui i dati sono cifrati tramite SSL/TLS.
Javascript	Linguaggio di programmazione usato principalmente per aggiungere interattività alle pagine web.
Jest	Framework di testing per JavaScript sviluppato da Facebook, utilizzato per testare applicazioni frontend e backend, con funzionalità come test unitari, mock e copertura del codice.
Json	(JavaScript Object Notation) Formato leggero per lo scambio di dati, leggibile sia da esseri umani che da macchine.
Kanban	Metodo di gestione dei flussi di lavoro che utilizza bacheche visive per rappresentare il progresso delle attività.
MVC	(Model-View-Controller) Architettura software che separa l'applicazione in tre componenti: Model (gestione dei dati e logica), View (interfaccia utente), e Controller (gestione delle interazioni e collegamento tra Model e View).
Nodejs	Ambiente di esecuzione JavaScript lato server basato sul motore V8 di Chrome.
ORM Sequelize	Libreria ORM (Object-Relational Mapping) per Node.js che permette di interagire con database SQL usando oggetti JavaScript.
Rate Limiting	Tecnica per limitare il numero di richieste che un utente può fare a un server in un determinato intervallo di tempo, utile per prevenire abusi.
URL	(Uniform Resource Locator) Indirizzo utilizzato per identificare una risorsa su Internet.
XSS	(Cross-Site Scripting) Tipo di vulnerabilità che consente a un attaccante di iniettare script maligni in pagine web visualizzate da altri utenti.

9 Bibliografia

9.1 Sitografia

Guida introduttiva a Node.js con esempi pratici su server, moduli, file e altro:	
• https://www.w3schools.com/nodejs	- 05.05.2025
Documentazione ufficiale di Express, framework web per applicazioni Node.js:	
• https://expressjs.com/	- 05.05.2025
Strumenti online e risorse per scrivere, testare e ottimizzare CSS:	
• https://www.cssportal.com/	- 06.05.2025
Documentazione ufficiale di Sequelize, ORM per database SQL in Node.js:	
• https://sequelize.org/	- 07.05.2025
Guida rapida per iniziare con Sequelize: installazione, configurazione e primi passi:	
• https://sequelize.org/docs/v6/getting-started/	- 07.05.2025
Documentazione su come usare Sequelize con il database MySQL:	
• https://sequelize.org/docs/v7/databases/mysql/	- 07.05.2025
Sito ufficiale di Socket.IO, libreria per comunicazione in tempo reale via WebSocket:	
• https://socket.io/	- 09.05.2025
Documentazione ufficiale di Socket.IO con guide e API per la comunicazione in tempo reale:	
• https://socket.io/docs/v4/	- 09.05.2025
Tutorial ufficiale di Node.js su come usare WebSocket per comunicazioni bidirezionali:	
• https://nodejs.org/en/learn/getting-started/websocket	- 09.05.2025
Sito ufficiale di Jest, framework di testing per JavaScript:	
• https://jestjs.io/	- 12.05.2025
Guida introduttiva di Jest per configurare e scrivere test in JavaScript:	
• https://jestjs.io/docs/getting-started	- 12.05.2025
Pagina ufficiale del pacchetto express-rate-limit per limitare le richieste in Express.js:	
• https://www.npmjs.com/package/express-rate-limit	- 13.05.2025
Articolo su come usare Winston per il logging in applicazioni Node.js:	
• https://last9.io/blog/winston-logging-in-nodejs/	- 13.05.2025
Guida pratica su come fare database seeding in Node.js per popolare dati iniziali:	
• https://dev.to/larswaechter/database-seeding-in-node-js-518d	- 13.05.2025

10 Allegati

Gli allegati si possono trovare nel seguente repository:

http://gitsam.cpt.local/2024_2025_lpi/marucchi-teamsync

- **QDC** (1_QDC\LPI25_QdC_MaDi_Eros_Marucchi.pdf).
- **Abstract** (2_Abstract\ Abstract_LPI_Marucchi_TeamSync.pdf).
- **Diario di lavoro** (4_Diari\).
- **Applicativo** (5_Applicativo\).
- **Database** (6_Database\teamsync.sql).
- **Piano preventivo e consuntivo** (7_Allegati\gant\1_preventivo\ Preventivo_LPI_Marucchi_TeamSync e 7_Allegati\2_consuntivo\ Consuntivo_LPI_Marucchi_TeamSync).
- **UseCase** (7_Allegati\ Uml_LPI_Marucchi_TeamSync.pdf).
- **Diagramma ER** (7_Allegati\diagramma_er\ diagramma_er_LPI_Marucchi_TeamSync.png).
- **Diagrammi di flusso** (7_Allegati\diagrammi_flusso\admin\ diagrammaFlusso_admin.png e 7_Allegati\diagrammi_flusso\user\ diagrammaFlusso_user.png).
- **Design delle interfacce** (7_Allegati\design_interfacce\admin e 7_Allegati\design_interfacce\admin).

11 Note

Questo progetto, e la sua documentazione, sono stati realizzati con il supporto dell'intelligenza artificiale. In particolare, sono stati utilizzati i seguenti modelli:

- <https://chatgpt.com/>
- <https://claude.ai/>
- <https://v0.dev/>