TeamSync

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 necessità 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 necessità un utente user
Sotto requisiti	
001	Si necessita la possibilità di creare degli sprint
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 necessità 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

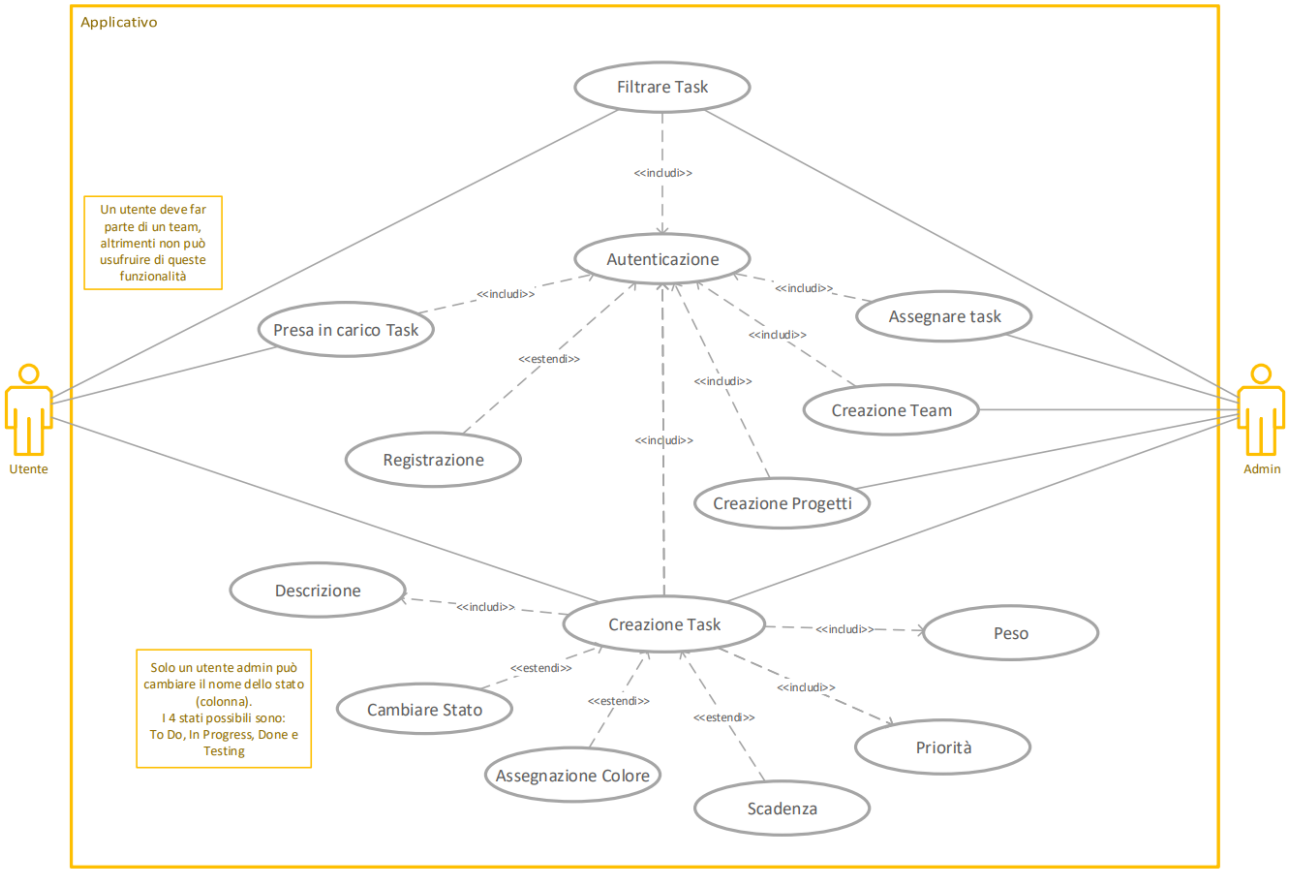


Figura 1 - Use Case

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:

	Modalità attività	Nome attività	Lavoro	Inizio	Fine	Predecessori
1		**TeamSync**	**80 h**	**lun 05.05.25**	**ven 16.05.25**	
2		**Analisi**	**8 h**	**lun 05.05.25**	**lun 05.05.25**	
3		Discussione col formatore	3 h	lun 05.05.25	lun 05.05.25	
4		Analisi delle tecnologie	3 h	lun 05.05.25	lun 05.05.25	
5		Analisi del dominio	2 h	lun 05.05.25	lun 05.05.25	
6		**Progettazione**	**9 h**	**mar 06.05.25**	**mer 07.05.25**	**2**
7		Diagramma Gantt	1 h	mar 06.05.25	mar 06.05.25	
8		UML	2 h	mar 06.05.25	mar 06.05.25	
9		Diagrammi di flusso	1 h	mar 06.05.25	mar 06.05.25	
10		Design delle interfacce	2 h	mar 06.05.25	mar 06.05.25	
11		Requisiti funzionali	2 h	mar 06.05.25	mar 06.05.25	
12		Diagramma ER	1 h	mer 07.05.25	mer 07.05.25	
13		**Implementazione**	**35 h**	**mer 07.05.25**	**mar 13.05.25**	
14		**Backend**	**19 h**	**mer 07.05.25**	**ven 09.05.25**	**6**
15		**DB**	**3 h**	**mer 07.05.25**	**mer 07.05.25**	
16		Implementazione DB	3 h	mer 07.05.25	mer 07.05.25	
17		**Autenticazione**	**4 h**	**mer 07.05.25**	**mer 07.05.25**	**15**
18		Login e registrazione	2 h	mer 07.05.25	mer 07.05.25	
19		Gestione ruoli	2 h	mer 07.05.25	mer 07.05.25	
20		**Funzionalità**	**12 h**	**gio 08.05.25**	**ven 09.05.25**	**17**
21		Creazione task	8 h	gio 08.05.25	gio 08.05.25	
22		Dettagli task	4 h	ven 09.05.25	ven 09.05.25	
23		**GUI**	**4 h**	**ven 09.05.25**	**ven 09.05.25**	
24		Creazione GUI	4 h	ven 09.05.25	ven 09.05.25	
25		**Test**	**12 h**	**lun 12.05.25**	**mar 13.05.25**	**14**
26		Unit test	4 h	lun 12.05.25	lun 12.05.25	
27		Debug	4 h	lun 12.05.25	lun 12.05.25	
28		Correzioni	4 h	mar 13.05.25	mar 13.05.25	
29		**Documentazione**	**28 h**	**lun 05.05.25**	**ven 16.05.25**	
30		Documentazione DB	4 h	mar 13.05.25	mar 13.05.25	
31		Documentazione progetto	22 h	lun 05.05.25	ven 16.05.25	
32		Poster	2 h	ven 16.05.25	ven 16.05.25	

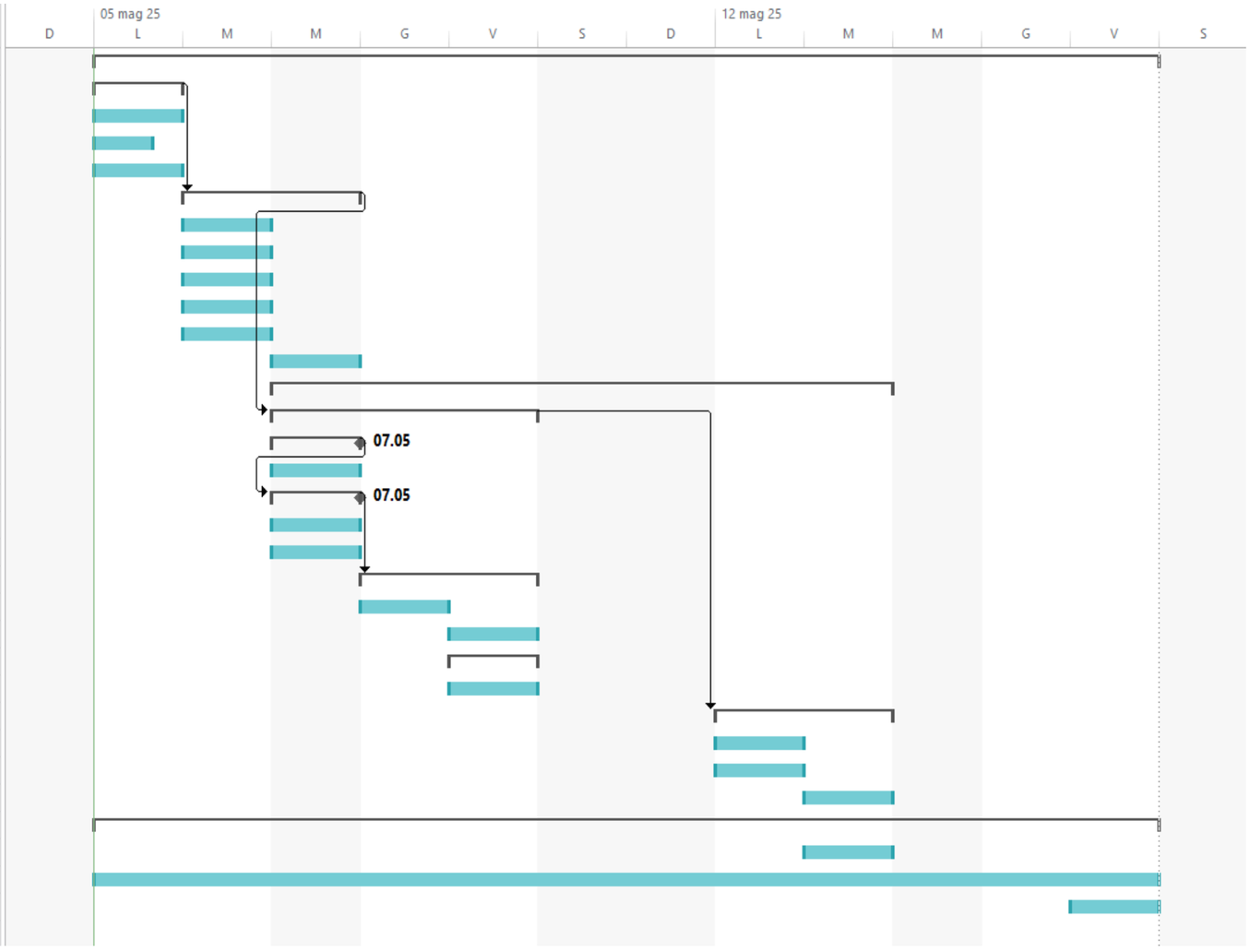


Figura 2 - Gantt

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:

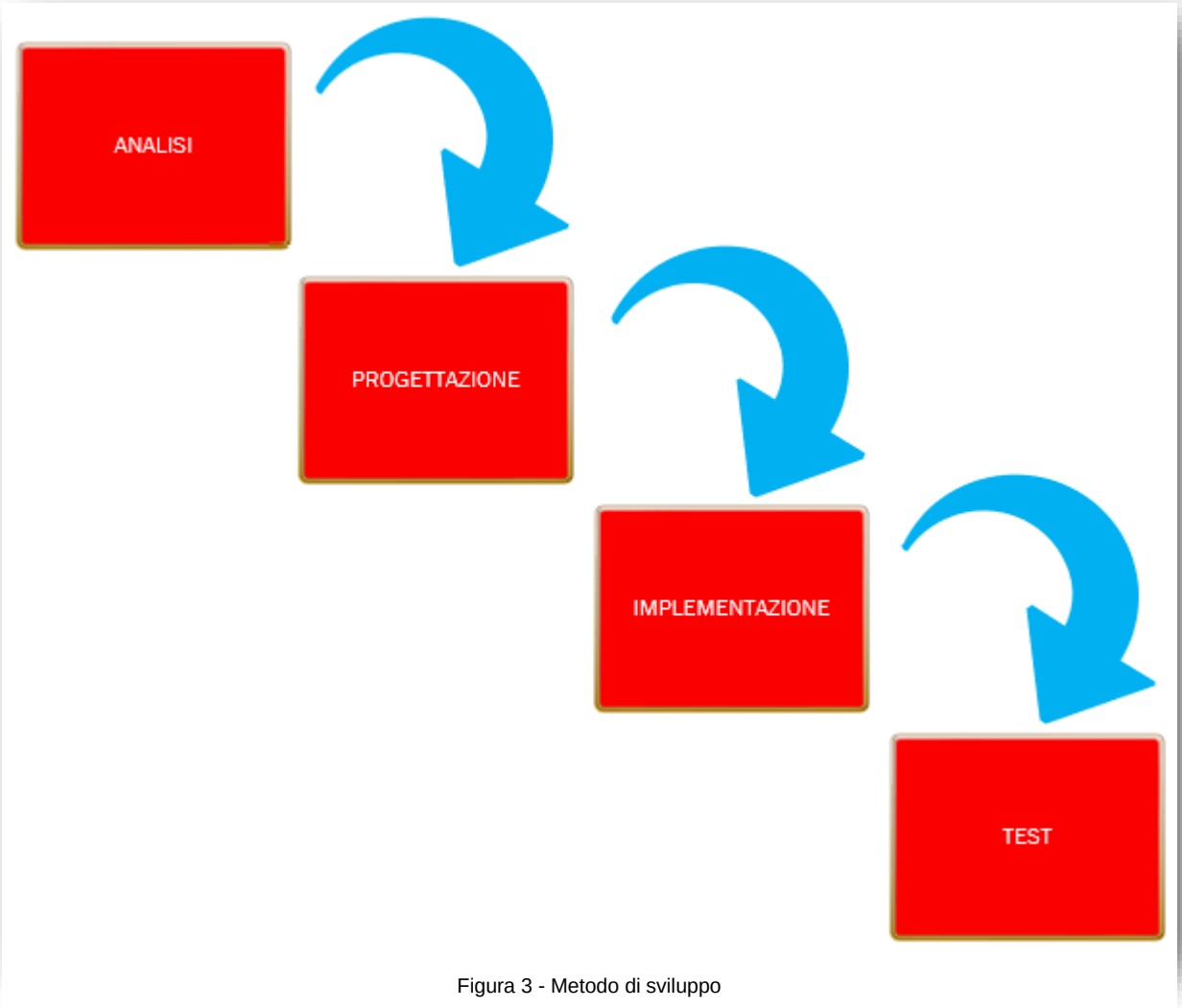


Figura 3 - Metodo di sviluppo

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 Edge116.0.1938.81, browser per testare l'applicativo.
- Micorsoft 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.

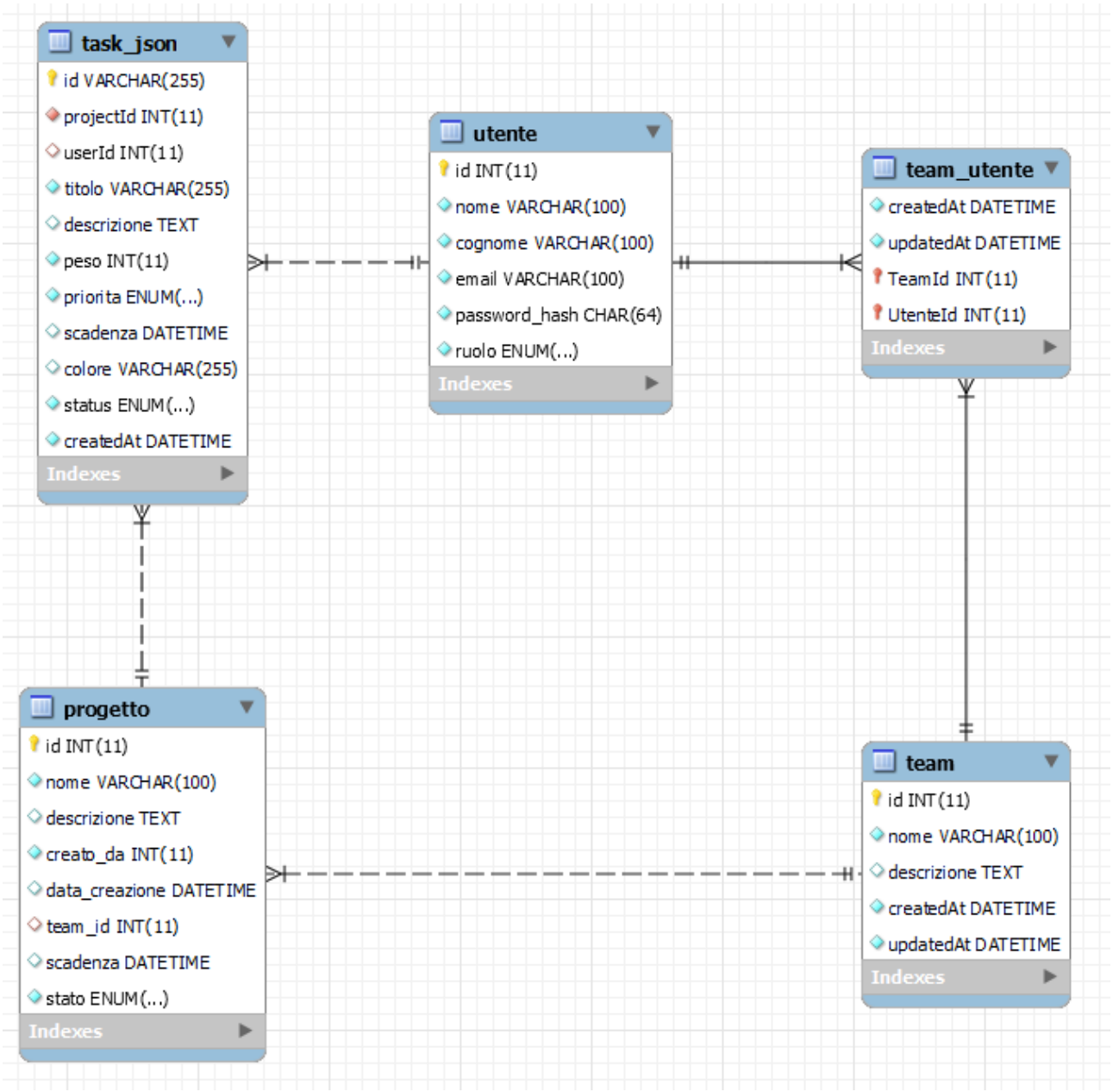


Figura 4 - Diagramma ER

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 priorita, peso, status e colore 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 creato_da 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 Design delle interfacce

Queste interfacce grafiche sono state realizzate con l'ausilio dell'intelligenza artificiale tramite la piattaforma *https://v0.dev/.*

3.4.1 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.

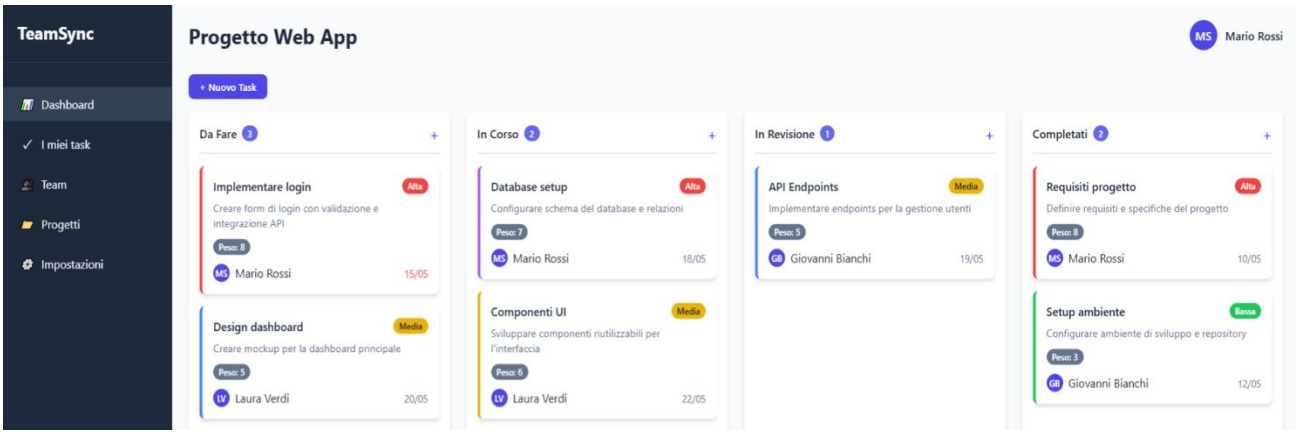


Figura 5 - Design dashboard user

3.4.2 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.

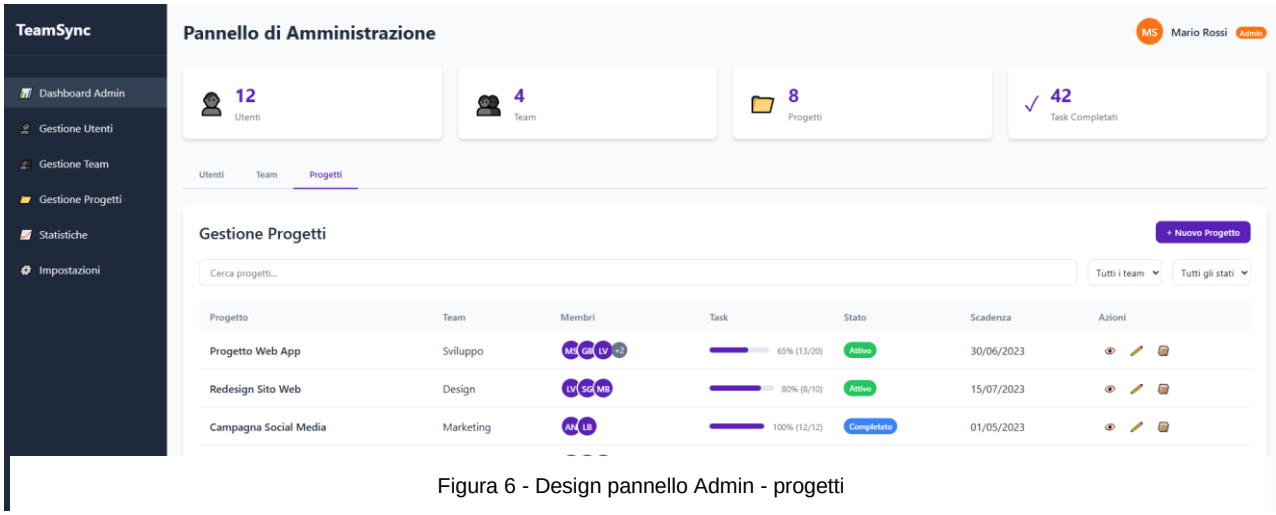


Figura 6 - Design pannello Admin - progetti

3.4.3 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.

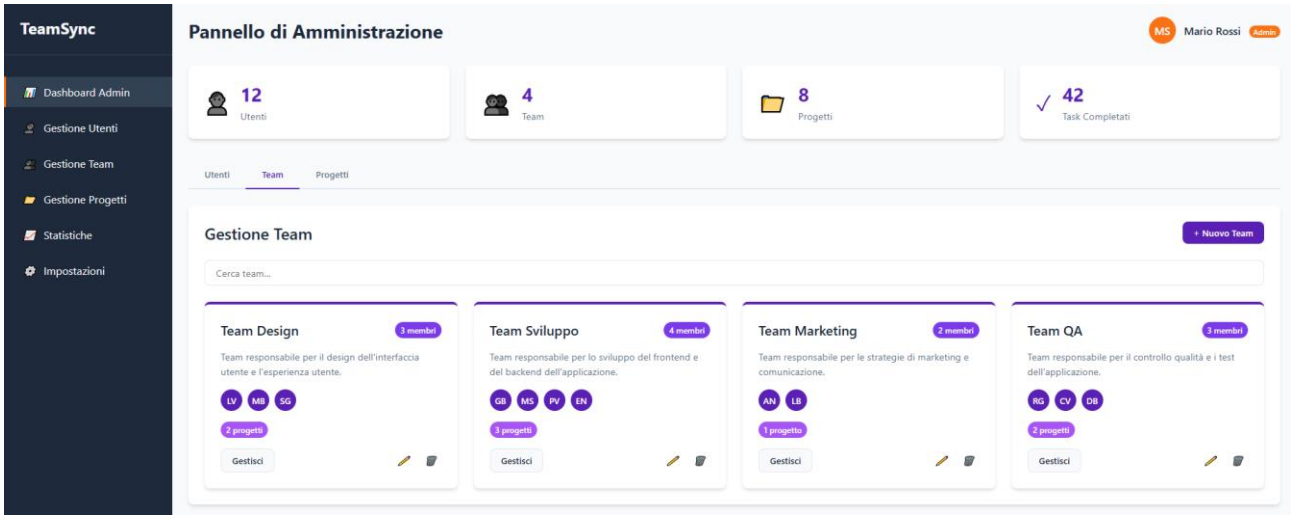


Figura 7 - Design pannello Admin - team

3.4.4 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.

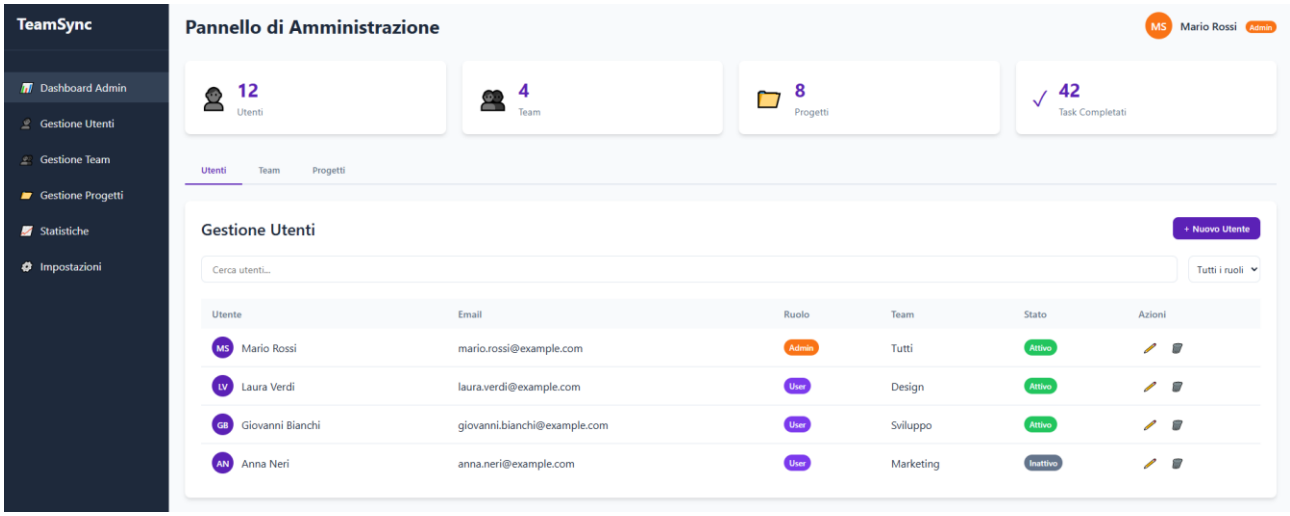


Figura 8 – Design pannello Admin – utenti

3.5 Design procedurale

3.5.1 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

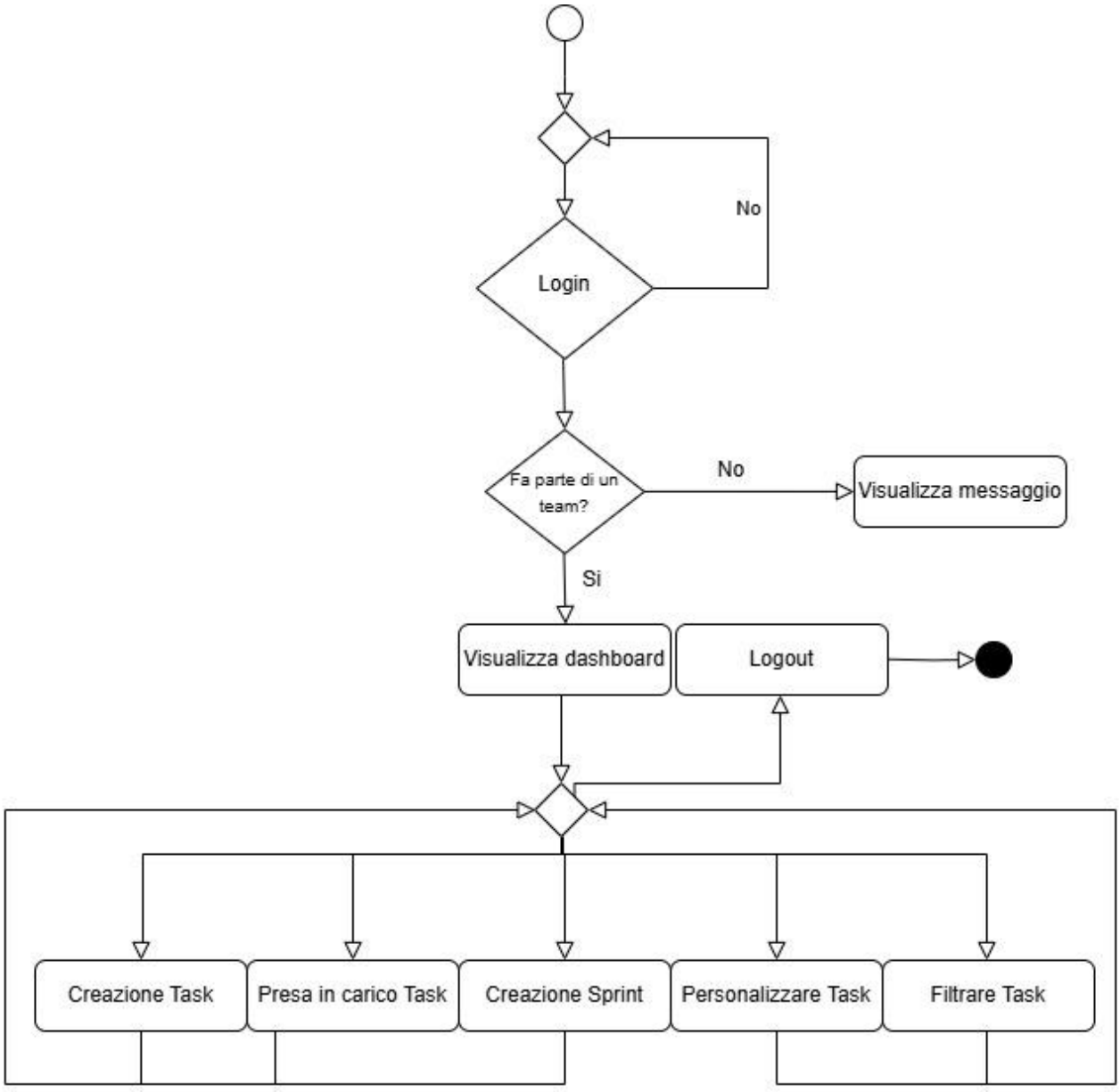


Figura 9 - Diagramma di flusso user

3.6 Utente admin

I 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.

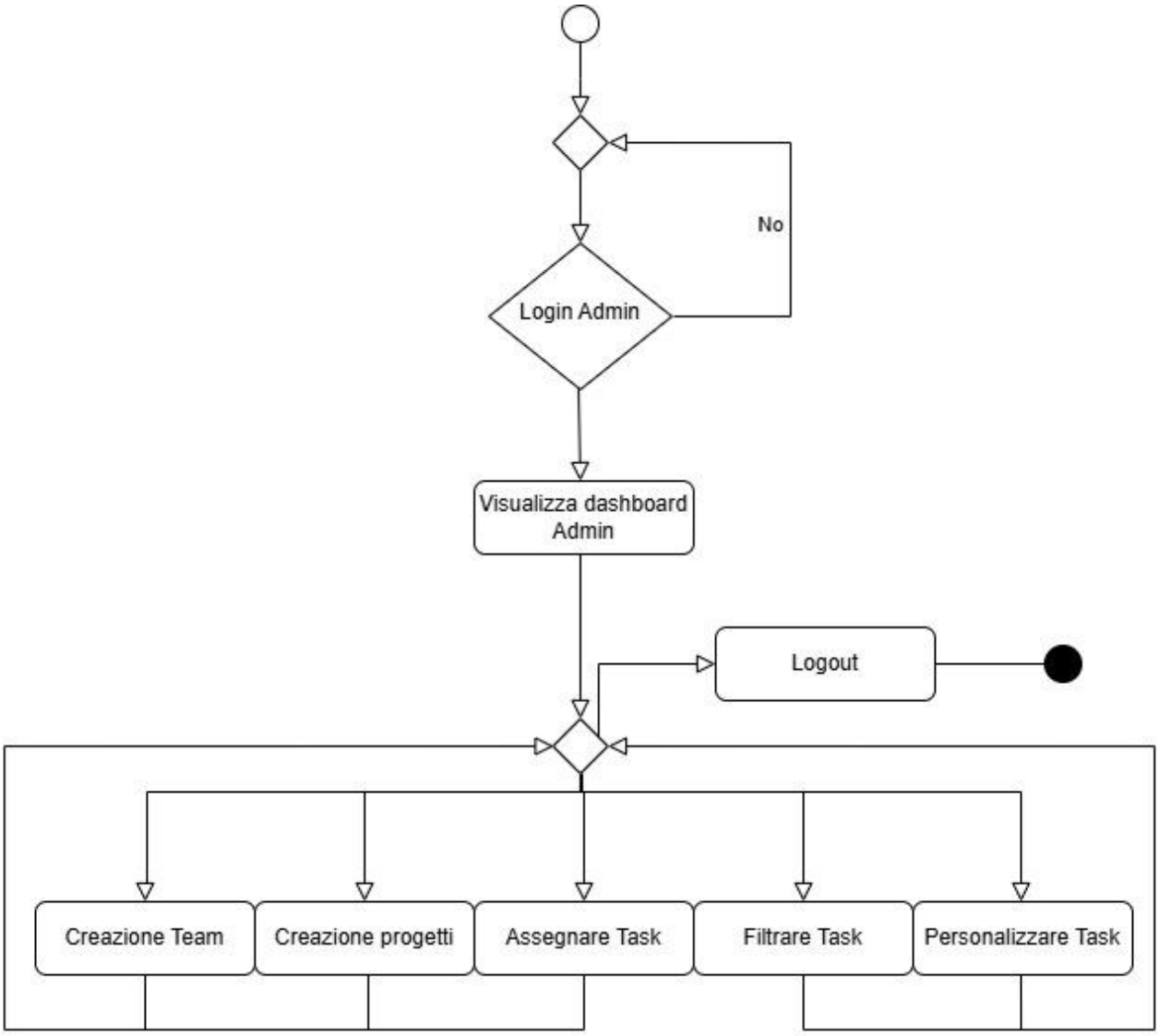


Figura 10 - Diagramma di flusso admin

4 Implementazione

4.1 Index.js

4.2 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'));
```

Nel file index.js, il motore di template Handlebars viene configurato per il rendering delle pagine HTML lato server. In particolare, la configurazione avviene tramite il pacchetto express-handlebars, che permette di separare la logica di presentazione dalla logica di business. Viene utilizzata l'estensione .hbs per i file template, che sono salvati nella cartella views. La configurazione disabilita l'uso di un layout predefinito (defaultLayout: false), permettendo così un controllo manuale e più flessibile del layout per ogni singola pagina. Inoltre, vengono definiti dei partial, ovvero blocchi di template riutilizzabili come header, footer, ecc., memorizzati nella cartella views/partials.

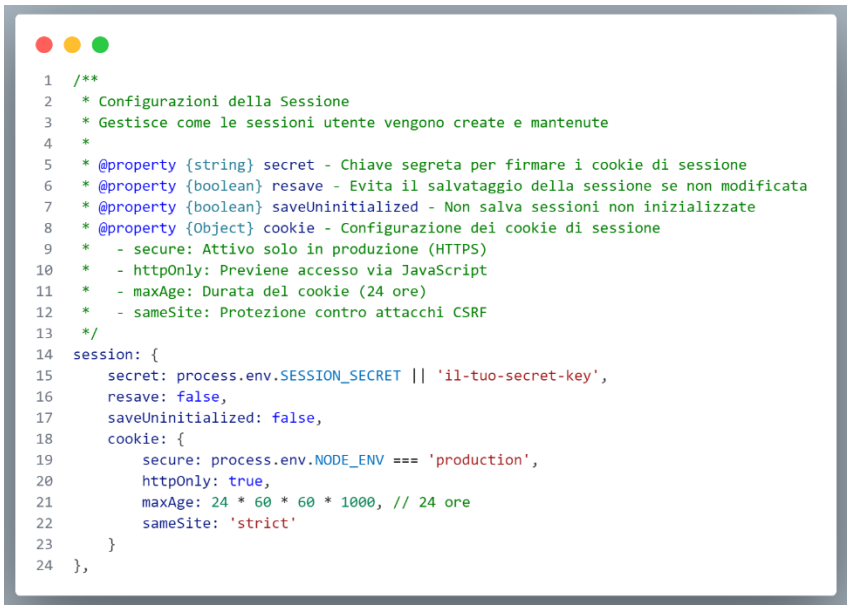
4.3 Configurazioni di Sicurezza

```
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);
```

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 lagestione sicura delle sessioni utente, viene utilizzato il pacchetto "*express-session*".

4.4 Security.js

4.4.1 Configurazione della sessione



```
/**
 * Configurazioni della Sessione
 * Gestisce come le sessioni utente vengono create e mantenute
 *
 * @property {string} secret - Chiave segreta per firmare i cookie di sessione
 * @property {boolean} resave - Evita il salvataggio della sessione se non modificata
 * @property {boolean} saveUninitialized - Non salva sessioni non inizializzate
 * @property {Object} cookie - Configurazione dei cookie di sessione
 *   - secure: Attivo solo in produzione (HTTPS)
 *   - httpOnly: Previene accesso via JavaScript
 *   - maxAge: Durata del cookie (24 ore)
 *   - sameSite: Protezione contro attacchi CSRF
 */
session: {
    secret: process.env.SESSION_SECRET || 'il-tuo-secret-key',
    resave: false,
    saveUninitialized: false,
    cookie: {
        secure: process.env.NODE_ENV === 'production',
        httpOnly: true,
        maxAge: 24 * 60 * 60 * 1000, // 24 ore
        sameSite: 'strict'
    }
},
```

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

```
/**
 * Configurazioni Helmet
 * Imposta gli header HTTP di sicurezza
 *
 * Content Security Policy (CSP):
 * - defaultSrc: Risorse predefinite solo dal proprio dominio
 * - scriptSrc: JavaScript dal proprio dominio e inline
 * - styleSrc: CSS dal proprio dominio, inline e Google Fonts
 * - fontSrc: Font dal proprio dominio e Google Fonts
 * - imgSrc: Immagini dal proprio dominio, data URLs e HTTPS
 * - connectSrc: Connessioni WebSocket/AJAX solo dal proprio dominio
 */
helmet: {
    contentSecurityPolicy: {
        directives: {
            defaultSrc: ["'self'"],
            scriptSrc: ["'self'", "'unsafe-inline'"],
            styleSrc: ["'self'", "'unsafe-inline'", "https://fonts.googleapis.com"],
            fontSrc: ["'self'", "https://fonts.gstatic.com"],
            imgSrc: ["'self'", "data:", "https:"],
            connectSrc: ["'self'"]
        }
    }
}
```

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 Rate Limiter

```
/**
 * Configurazioni Rate Limiter
 * Protegge da attacchi DDoS e abusi delle API
 *
 * @property {number} windowMs - Finestra temporale per il conteggio (1 minuto)
 * @property {number} max - Numero massimo di richieste nella finestra
 * @property {string} message - Messaggio di errore per limite superato
 * @property {boolean} standardHeaders - Abilita header standard di rate limit
 * @property {boolean} legacyHeaders - Disabilita header legacy
 */
rateLimit: {
    windowMs: 60 * 1000,
    max: 1000,
    message: 'Troppe richieste, riprova più tardi.',
    standardHeaders: true,
    legacyHeaders: false
}
};
```

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

```
/**
 * Verifica se l'utente è autenticato
 * Protegge le rotte che richiedono autenticazione
 *
 * @middleware
 * @param {Object} req - Request object
 * @param {Object} res - Response object
 * @param {Function} next - Next middleware
 * @redirects {string} /login - Se l'utente non è autenticato
 */
const isAuthenticated = (req, res, next) => {
    if (req.session.user) {
        next();
    } else {
        res.redirect('/login');
    }
```

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

```
 * Verifica:
  * - Esistenza del progetto
  * - Autorizzazione dell'utente
  * - Appartenenza al team
  *
  * @route GET /dashboard/:id
  * @access Private
  * @param {string} id - ID del progetto
  * @renders dashboard
  */
router.get('/dashboard/:id', isAuthenticated, async (req, res) => {
    try {
        const progetto = await Progetto.findByPk(req.params.id, {
            include: [{
                model: Team,
                include: [{
                    model: Utente,
                    attributes: ['id', 'nome', 'cognome']
                }]
            }]
        });

        if (!progetto) {
            return res.status(404).redirect('/home');
        }

        // Verifica che l'utente sia autorizzato a vedere il progetto
        const isCreator = progetto.creato_da === req.session.user.id;
        const isTeamMember = progetto.Team && progetto.Team.Utentes.some(u => u.id === req.session.user.id);

        if (!isCreator && !isTeamMember) {
            return res.status(403).redirect('/home');
        }

        res.render('dashboard', {
            user: req.session.user,
            progetto: progetto.get({ plain: true }),
            progettoId: progetto.id,
            activePage: 'dashboard'
        });
    } catch (error) {
        logger.error(`Errore nel caricamento della dashboard: ${error.message}`);
        res.status(500).redirect('/home');
    }
});
```

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.

4.6 authController.js

4.6.1 Registrazione di un Nuovo Utente

```
 * @param {Object} req - Request object con i dati dell'utente:
 *   - nome: Nome dell'utente
 *   - cognome: Cognome dell'utente
 *   - email: Email dell'utente
 *   - password: Password non criptata
 * @param {Object} res - Response object
 *
 * Flusso di esecuzione:
 * 1. Validazione e formattazione input
 * 2. Verifica email duplicata
 * 3. Hashing password
 * 4. Creazione utente nel database
 * 5. Risposta con conferma
 */
exports.register = async (req, res) => {
    try {
        let { nome, cognome, email, password } = req.body;
        logger.info(`Tentativo di registrazione per l'email: ${email}`);

        // Formatta nome e cognome
        nome = formatName(nome);
        cognome = formatName(cognome);

        // Validazione input
        validateEmail(email);
        validatePassword(password);

        // Verifica se l'email esiste già
        const existingUser = await Utente.findOne({ where: { email } });
        if (existingUser) {
            throw new AppError(messages.email.exists, 400);
        }

        // Genera salt e hash della password
        const salt = await bcrypt.genSalt(12);
        const passwordHash = await bcrypt.hash(password, salt);

        // Crea nuovo utente
        const user = await Utente.create({
            nome,
            cognome,
            email,
            password_hash: passwordHash,
            ruolo: 'user'
        });
```

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

```
/**
 * Valida la password secondo i requisiti di sicurezza
 * Verifica che la password soddisfi tutti i criteri richiesti
 *
 * @param {string} password - Password da validare
 * @throws {AppError} Se la password non soddisfa uno o più requisiti:
 * - Lunghezza minima
 * - Presenza di maiuscole
 * - Presenza di minuscole
 * - Presenza di numeri
 * - Presenza di caratteri speciali
 */
const validatePassword = (password) => {
    const errors = [];

    if (password.length < securityConfig.auth.passwordMinLength) {
        errors.push(messages.password.invalid.details.length);
    }
    if (!/[A-Z]/.test(password)) {
        errors.push(messages.password.invalid.details.uppercase);
    }
    if (!/[a-z]/.test(password)) {
        errors.push(messages.password.invalid.details.lowercase);
    }
    if (!/[0-9]/.test(password)) {
        errors.push(messages.password.invalid.details.number);
    }
    if (!/[!@#$%^&*]/.test(password)) {
        errors.push(messages.password.invalid.details.special);
    }

    if (errors.length > 0) {
        throw new AppError({
            message: messages.password.invalid.message,
            details: errors,
            example: messages.password.invalid.example
        }, 400);
    }
};
```

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

```
/**
 * Valida il formato dell'email
 * Verifica che l'email rispetti il formato standard
 *
 * @param {string} email - Email da validare
 * @throws {AppError} Se l'email non rispetta il formato standard:
 * - Deve contenere @
 * - Deve avere un dominio valido
 * - Non può contenere spazi
 */
const validateEmail = (email) => {
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(email)) {
        throw new AppError(messages.email.invalid, 400);
    }
};
```

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

```
/**
 * Recupera tutte le task associate a un progetto specifico
 * Include i dati dell'utente assegnato (nome e cognome)
 *
 * @param {Object} req - Request object contenente:
 *   - projectId: ID del progetto nei parametri URL
 * @param {Object} res - Response object
 *
 * Caratteristiche:
 * - Ordinamento: Dalla più recente alla più vecchia
 * - Include dati utente associato
 * - Filtraggio per progetto
 */
const getTasks = async (req, res) => {
    try {
        const { projectId } = req.params;

        const tasks = await TaskJson.findAll({
            where: { projectId },
            include: [{
                model: Utente,
                attributes: ['nome', 'cognome']
            }],
            order: [['createdAt', 'DESC']]
        });

        res.json({
            success: true,
            data: tasks
        });
    } catch (error) {
        logger.error('Errore durante il recupero delle task:', error);
        res.status(500).json({
            success: false,
            message: 'Errore durante il recupero delle task',
            error: error.message
        });
    }
};
```

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

```
/**
 * Elimina una task dal sistema
 *
 * @param {Object} req - Request object contenente:
 *   - id: ID della task da eliminare nei parametri URL
 * @param {Object} res - Response object
 *
 * Validazioni:
 * - Verifica esistenza task
 * - Conferma eliminazione
 */
const deleteTask = async (req, res) => {
    try {
        const { id } = req.params;

        // Verifica esistenza task
        const task = await TaskJson.findByPk(id);
        if (!task) {
            return res.status(404).json({
                success: false,
                message: 'Task non trovata'
            });
        }

        // Eliminazione task
        await task.destroy();

        res.json({
            success: true,
            message: 'Task eliminata con successo'
        });
    } catch (error) {
        logger.error('Errore durante l\'eliminazione della task:', error);
        res.status(500).json({
            success: false,
            message: 'Errore durante l\'eliminazione della task',
            error: error.message
        });
    }
};
```

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.

4.8 Protocollo di test

Test Case: **Riferimento**:	TC-001 REQ-001	**Nome:**	**Registrazione utente**
Descrizione:	Verifica che un utente si possa registrare nell'applicativo		
Prerequisiti:	• -		
Procedura:	1. -		
Risultati attesi:	Verrà caricata una pagina con un messaggio positivo dell'avvenuta registrazione.		

Test Case: **Riferimento**:	TC-002 REQ-002	**Nome:**	**Login utente**
Descrizione:	Verifica che un utente si possa effettuare l'accesso all'applicativo		
Prerequisiti:	• Essersi registrati		
Procedura:	1. Aprire il browser e recarsi nel seguente percorso: "*https://looplab.labosamt.ch/*". 2. Inserire i dati richiesti (email e password). 3. 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:	• Aver già testato il test case numero 001.		
Procedura:	1. -		
Risultati attesi:	Verrà mostrata la parte admin dell'applicativo.		

4.9 Risultati test

Test Case	Risultato	Descrizione	Data Test
001	Passato	**Risultato atteso:** **Risultato effettivo:** **Foto:**	XX.XX.XX

4.10 Riassunto Test

ID Test-Case	Risultato	Commento / Note	Data
TC-001	-		xx.xx.xx
TC-002	-		xx.xx.xx
TC-003	-		xx.xx.xx
TC-004	-		xx.xx.xx
TC-005	-		xx.xx.xx
TC-006	-		xx.xx.xx
TC-007	-		xx.xx.xx
TC-008	-		xx.xx.xx

4.11 Mancanze/limitazioni conosciute

Descrizione con motivazione di eventuali elementi mancanti o non completamente implementati, al di fuori dei test case. Non devono essere riportati gli errori e i problemi riscontrati e poi risolti durante il progetto.

5 Consuntivo

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

6 Conclusioni

Quali sono le implicazioni della mia soluzione? Che impatto avrà? Cambierà il mondo? È un successo importante? È solo un'aggiunta marginale o è semplicemente servita per scoprire che questo percorso è stato una perdita di tempo? I risultati ottenuti sono generali, facilmente generalizzabili o sono specifici di un caso particolare? ecc.

6.1 Sviluppi futuri

Migliorie o estensioni che possono essere sviluppate sul prodotto.

6.2 Considerazioni personali

Cosa ho imparato in questo progetto? ecc.

7 Bibliografia

7.1 Sitografia

- https://www.example.com - 03.02.2025

7.2 Sommario delle immagini

8 Allegati

Elenco degli allegati, esempio:

- Diari di lavoro
- Codici sorgente/documentazione macchine virtuali
- Istruzioni di installazione del prodotto (con credenziali di accesso) e/o di eventuali prodotti terzi
- Documentazione di prodotti di terzi
- Eventuali guide utente / Manuali di utilizzo
- Mandato e/o QdC
- Prodotto
- …