Università degli Studi di Milano - Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea in Informatica

Integrazione di un sistema di gestione meeting per una piattaforma di recruiting

Relatore: Prof.ssa Daniela Micucci

Tesi di Laurea di:
Simone Lesinigo
Matricola 899540

Anno Accademico 2023-2024

Ai miei genitori, per non avermi mai fatto mancare niente
e per avermi permesso di frequentare l'università,
sostenendomi sempre nelle mie scelte.

Ai miei amici Angela, Federico, Mattia, Maurizio e Vito,
che mi accompagnano fin dall'adolescenza, sempre pronti
a sostenermi e ad aiutarmi nei momenti di difficoltà.

A tutti i professori che mi hanno formato nel mio percorso di studi,
donandomi preziosi insegnamenti.

Ai miei compagni di corso, in particolare Luca e Simone,
con cui ho condiviso questi anni di università.

A me stesso, per aver sempre creduto in me
e aver raggiunto gli obiettivi che mi ero prefissato.

Crescat scientia vita excolatur.

Indice

Elenco delle figure

Introduzione

Il campo del recruiting è in continua evoluzione, con una crescente necessità di strumenti che facilitino in modo semplice ed efficiente l'incontro tra domanda e offerta di lavoro. In questo contesto, la gestione dei colloqui di lavoro rappresenta una fase cruciale e complessa del processo di selezione.

L'obiettivo di questa esperienza di stage, svoltasi presso l'azienda Nesecom SRLS, è stato quello di integrare un sistema per la gestione dei colloqui di lavoro tramite meeting all'interno di un sito web in sviluppo, RisUma.

RisUma, acronimo di RISorse UMAne, è pensato per essere una piattaforma di recruiting intuitiva, che consente agli utenti e alle aziende di trovare efficacemente le opportunità di lavoro e le figure professionali richieste, gestendo l'intero processo di selezione su un'unica piattaforma.

Questo progetto è ancora nelle fasi iniziali del suo sviluppo, ma punta a eguagliare e superare le più famose piattaforme di recruiting grazie alle sue comode funzionalità.

0.1 Stato dell'arte

Capitolo 1

Tecnologie utilizzate

Nel corso dello sviluppo del progetto sono state impiegate diverse tecnologie moderne, per garantire un'applicazione robusta, scalabile e facilmente manutenibile. La parte del progetto a me assegnata ha richiesto uno sviluppo full-stack, coinvolgendo sia il frontend che il backend.

1.1 Frontend

Il frontend rappresenta la parte dell'applicazione con cui l'utente interagisce direttamente. È stato sviluppato utilizzando **React** insieme a **TypeScript**, con l'obiettivo di garantire un'esperienza utente fluida e intuitiva.
Per gestire i pacchetti e le dipendenze, è stato utilizzato **npm** (Node Package Manager). Inoltre, il design dell'interfaccia utente è stato standardizzato utilizzando il **MUI Theme** di Material-UI.
Per la comunicazione tra frontend e backend, è stata utilizzata la libreria **Axios**, che consente di effettuare richieste HTTP in modo semplice ed efficiente.

- **React**: React è una libreria JavaScript open source, sviluppata da Facebook, per la costruzione di interfacce utente. Utilizzando un approccio basato sui componenti, React consente di creare interfacce utente modulari e riutilizzabili. La sua capacità di aggiornare e mostrare efficientemente solo i componenti necessari in risposta ai cambiamenti rende React particolarmente adatto per lo sviluppo di applicazioni interattive e ad alte prestazioni. Inoltre, React permette di gestire lo stato delle applicazioni in modo prevedibile e scalabile, facilitando lo sviluppo di applicazioni complesse. [1]

- **TypeScript**: TypeScript è un linguaggio di programmazione open source sviluppato da Microsoft, che estende JavaScript aggiungendo tipi statici. L'adozione di TypeScript permette di ridurre significativamente gli errori durante lo sviluppo, grazie al controllo statico dei tipi che individua potenziali problemi prima ancora dell'esecuzione del codice. Inoltre, TypeScript facilita la manutenzione del codice in progetti di grandi dimensioni, migliorando la leggibilità e la documentazione attraverso la tipizzazione esplicita. [2]

- **MUI Theme (Material-UI)**: Material-UI è una libreria di componenti React che implementa le linee guida del **Material Design** di Google. Utilizzare Material-UI permette di sviluppare interfacce utente coerenti e professionali senza dover creare e stilizzare i componenti da zero. La libreria offre una vasta gamma di componenti pre-stilizzati e altamente personalizzabili, che facilitano la creazione di un design uniforme e accattivante. Inoltre, Material-UI supporta nativamente la creazione di temi, consentendo di personalizzare l'aspetto dell'applicazione in modo centralizzato.

- **npm**: Node Package Manager è il gestore di pacchetti predefinito per l'ecosistema JavaScript, utilizzato per installare e gestire le dipendenze necessarie per lo sviluppo delle applicazioni. Npm facilita l'integrazione di librerie e strumenti di terze parti, permettendo agli sviluppatori di accedere rapidamente a un vasto repository di pacchetti open source. Questo strumento è fondamentale per mantenere aggiornate le dipendenze del progetto e per gestire le versioni delle librerie utilizzate.

- **Axios**: Axios è una popolare libreria JavaScript per effettuare richieste HTTP dal browser e da Node.js. Grazie alla sua interfaccia basata su Promises, Axios consente di inviare richieste al backend in modo semplice. Supporta tutte le operazioni HTTP principali, come GET, POST, PUT e DELETE, e offre funzionalità avanzate come l'intercettazione delle richieste e delle risposte e la gestione automatica degli errori. Queste caratteristiche lo rendono uno strumento potente e flessibile per la comunicazione tra frontend e backend. [3]

1.2 Backend

Il backend rappresenta la parte dell'applicazione che gestisce la logica di business, l'elaborazione dei dati e la comunicazione con il database. È responsabile del funzionamento lato server dell'applicazione, elaborando le richieste degli utenti e restituendo le risposte appropriate al frontend.
Il backend dell'applicazione è stato sviluppato utilizzando **Java** con il framework **Spring Boot**. Inoltre, viene utilizzato **Maven** come strumento di gestione delle dipendenze.
Per la gestione della persistenza dei dati è stato utilizzato **Java Persistence API (JPA)**, che permette di interfacciarsi con il database in modo efficiente e strutturato. Il database utilizzato si basa su **PostgreSQL 14**.

- **Java**: Java è un linguaggio di programmazione ad alto livello, orientato agli oggetti e a tipizzazione statica, sviluppato da Sun Microsystems (ora di proprietà di Oracle). È noto per la sua robustezza, sicurezza e portabilità, grazie alla Java Virtual Machine (JVM) che permette di eseguire il codice Java su qualsiasi piattaforma. Java è ampiamente utilizzato nello sviluppo di applicazioni enterprise, sistemi embedded, applicazioni mobili e web. La sua vasta ecosistema di librerie e strumenti lo rende una scelta ideale per lo sviluppo backend. [4]

- **Spring Boot**: Spring Boot è un framework open source per lo sviluppo di applicazioni Java basato su Spring, che semplifica il processo di creazione

di app web e microservizi. Spring Boot offre una configurazione automatica dei componenti, riducendo significativamente il tempo necessario per avviare un progetto. Fornisce anche una serie di funzionalità integrate, come la gestione della sicurezza e il logging che facilitano lo sviluppo e la manutenzione delle applicazioni. [5]

- **Maven**: Maven è uno strumento di build automation e gestione delle dipendenze per progetti Java sviluppato dalla Apache Software Foundation. Utilizzando un modello basato su pom.xml (Project Object Model), Maven facilita la gestione del ciclo di vita del progetto, dall'inizializzazione alla distribuzione. Consente di integrare facilmente le librerie di terze parti e di gestire le versioni delle dipendenze, migliorando la coerenza e la riproducibilità del build. [6]

- **Java Persistence API (JPA)**: La Java Persistence API (JPA) è una specifica standard per la gestione della persistenza dei dati nelle applicazioni Java. JPA offre un modo standardizzato per mappare le classi Java agli oggetti del database, permettendo di eseguire operazioni CRUD (Create, Read, Update, Delete) in modo semplice e intuitivo.
  Inoltre fornisce un linguaggio per effettuare query SQL, chiamato **JPQL** (Java Persistence Query Language), che è indipendente dal DBMS utilizzato.
  Utilizzando JPA, gli sviluppatori possono concentrarsi sulla logica di business senza doversi preoccupare dei dettagli specifici dell'interazione con il database, migliorando la produttività e la manutenibilità del codice. [7]

- **PostgreSQL**: PostgreSQL è un sistema open source di database relazionale a oggetti sviluppato da Michael Stonebraker. È famoso per la sua affidabilità e per l'integrità dei dati, oltre alla community che continua a sostenerlo continuando a sviluppare nuove soluzioni. [8]

Capitolo 2

Approccio Proposto

2.1 Problema e Obiettivo

Il problema principale affrontato riguarda:

- Per le **aziende**: ottimizzazione dell'efficienza nella gestione dei meeting per i colloqui di lavoro, fornendo uno strumento adeguato e completo di tutte le funzionalità necessarie. L'obiettivo principale è migliorare l'esperienza dei recruiter, che potrebbero trovarsi a gestire un elevato numero di meeting.
- Per gli **utenti**: eliminare la necessità per i candidati di segnare manualmente gli appuntamenti, offrendo loro uno strumento che ricordi in modo comodo e automatico tutti i colloqui programmati, facilitandone l'accesso.

2.1.1 Caratteristiche dei Meeting

Un ulteriore obiettivo riguarda le proprietà che devono caratterizzare un meeting:

- Non è richiesto avere un account per partecipare al meeting, ma sarà sufficiente inserire un nome a propria scelta al momento dell'accesso.
- Nessuno può accedere al meeting prima di un quarto d'ora rispetto all'orario di inizio prestabilito o dopo che sia trascorso l'orario di fine.

2.2 Metodologia

Di seguito sono riportati i passi seguiti per la realizzazione di questo progetto. Si vuole far notare che RisUma è un'idea del direttore aziendale, e pertanto non vi è un cliente con cui confrontarsi. Inoltre, poiché non sono state disponibili direttive scritte ma solo alcune indicazioni orali, tutte le scelte sono state prese a mia completa discrezione.

1. **Servizio**: il primo passo è stato scegliere quale servizio di terze parti utilizzare per le chiamate online. Si cercava un servizio che:

(a) avesse le caratteristiche richieste

(b) si integrasse bene con il progetto (API disponibili, compatibilità con le tecnologie utilizzate...)

(c) fosse facilmente usabile anche dagli utenti meno esperti

Per tali motivi sono state prese in considerazione le piattaforme più note, quali Cisco Webex, Google Meet e Zoom.

2. **Interfaccia utente**: è stato scelto il design della pagina frontend. Basandosi anche sulle 10 euristiche di Jakob Nielsen, si sono perseguiti i seguenti obiettivi:

   (a) Adattare il sistema al mondo reale. (Euristica 2)

   (b) La pagina doveva essere facile da navigare, riducendo al minimo il carico cognitivo per l'utente. (Euristica 6)

   (c) Optare per uno stile minimalista e intuitivo. (Euristica 7)

   (d) Implementare funzionalità di feedback per guidare l'utente in caso di errori o problemi durante l'utilizzo della pagina. (Euristiche 8 e 9)

3. **Creazione della tabella a database**: è stata progettata e creata la tabella nel database aziendale, includendo tutte le colonne necessarie, al fine di garantire il corretto funzionamento della pagina web.

4. **Integrazione delle API**: sono state integrate le API per consentire le operazioni CRUD sui meeting, sviluppando contemporaneamente le parti di frontend e backend secondo il seguente ordine:

   (a) **Creazione**: implementazione della funzionalità per creare nuovi meeting, gestendo i parametri necessari come data, ora e invitati.

   (b) **Recupero**: realizzazione della logica per recuperare e visualizzare i meeting di un utente.

   (c) **Modifica**: implementazione della possibilità dei modificare i parametri dei meeting già esistenti.

   (d) **Eliminazione**: implementazione della funzionalità per annullare meeting programmati.

   Questa parte è stata cruciale nel corso della mia esperienza di stage in quanto ha costituito il nucleo essenziale della pagina che ho progettato e integrato.

5. **Test**: Il sistema è stato sottoposto da parte mia e di alcuni colleghi a dei test per garantire che tutte le funzionalità operassero correttamente.

6. **Deploy**: È stato eseguito un deploy su un server per testare le prestazioni in un ambiente più realistico rispetto a quello locale.

2.3 Possibili sfide

Nel corso dello sviluppo del progetto sono emerse due principali sfide:

1. **Consistenza tra i database**: assicurare la consistenza tra il database aziendale e quello del servizio di terze parti durante le operazioni di creazione, modifica o eliminazione di un meeting. Una eventuale inconsistenza porterebbe a gravi problemi legati allo svolgimento dei meeting, compromettendo l'affidabilità e le funzionalità del sistema.

2. **Creazione dell'interfaccia utente**: la realizzazione di un'interfaccia utente piacevole e funzionale ha rappresentato un'altra grande sfida. Nel corso dello sviluppo è stata più volte modificata sulla base dei feedback dei colleghi.

Capitolo 3

Struttura del progetto

3.1 Frontend

Il frontend del progetto è stato strutturato in modo da massimizzare la leggibilità del codice e sfruttare al meglio la modularità offerta da React.

- **Pagina principale**: la pagina principale costituisce il nucleo dell'interfaccia utente, funge da contenitore per i vari componenti che insieme andranno a formare la pagina visualizzata.
- **Componenti**: ogni componente è progettato per gestire una specifica parte dell'interfaccia utente. Questo favorisce la modularità di React, consentendo di riutilizzare facilmente le stesse interfacce in diverse parti dell'applicazione, oltre a facilitare la manutenzione del codice e a migliorarne la leggibilità.
- **Metodi**: tutti i metodi utilizzati nella logica del frontend sono posti in file separati, suddivisi per semantica. Questa struttura migliora la leggibilità e la comprensione del codice.
- **useEffect**: uno `useEffect` è un hook di React che viene eseguito in risposta a cambiamenti specifici nello stato dell'applicazione o al caricamento dei componenti nel DOM [9]. Anche in questo caso tutti gli useEffect sono stati divisi in file in base alla loro semantica.
- **Centralizzazione degli `useState`**: uno `useState` è un hook di React che permette di aggiungere una variabile di stato locale a un componenete [10]. Questi sono stati centralizzati utilizzando la Context API di React. La pagina principale viene wrappata con un contesto, permettendo ai componenti figli di accedere e modificare lo stato globale senza dover passare props attraverso vari livelli di componenti.

3.2 Backend

Il backend del progetto è stato progettato seguendo principi che favoriscono la manutenibilità e l'efficienza delle operazioni di business. È organizzato secondo un'architettura a tre strati, che include Controller, Service e Repository.

- **Controller**: i Controller costituiscono il punto di ingresso delle richieste API provenienti dal frontend. Gestiscono la ricezione delle richieste HTTP, validazione dei dati e routing delle operazioni verso il Service appropriato. Questo livello fornisce un'interfaccia verso l'esterno dell'applicazione, separando la logica di gestione delle richieste dalla logica di business sottostante.

- **Service**: il Service è responsabile della logica di business dell'applicazione. Ogni Service si occupa di una specifica area funzionale dell'applicazione, coordinando le operazioni necessarie per soddisfare le richieste ricevute dai Controller. Il Service coordina le interazioni con i Repository.

- **Repository**: i Repository si interfacciano direttamente con le fonti di dati, che possono essere il database o API esterne di servizi di terze parti. Forniscono un'astrazione dell'accesso ai dati, permettendo al Service di accedere e manipolare i dati in modo uniforme, indipendentemente dalla specifica implementazione del sistema di persistenza dei dati. Questo approccio facilita la sostituzione o l'aggiornamento delle tecnologie di accesso ai dati senza modificare la logica di business.

- **Gestione delle chiamate API esterne**: nel caso in cui le operazioni richiedano l'interazione con API esterne, i Repository sono progettati per gestire queste comunicazioni. Questo include la configurazione delle richieste HTTP, la gestione delle risposte e la gestione degli errori.

Capitolo 4

Sviluppo

In questa sezione sono riportati in dettaglio tutti gli step che si sono seguiti per la realizzazione del progetto.

4.1 L'idea

L'idea alla base del progetto è che l'azienda presso cui si è svolto lo stage assuma il ruolo di organizzatrice di tutti i meeting, garantendone il pieno controllo.
Gli utenti di RisUma dovranno semplicemente partecipare come ospiti.

Un altro punto cruciale è impedire alle aziende di mettersi in contatto con i candidati tramite mezzi alternativi a RisUma almeno fino al momento del meeting. Questo obbliga le aziende e gli utenti a utilizzare esclusivamente il sito, garantendo che tutte le comunicazioni avvengano tramite la piattaforma.

4.2 Scelta del servizio

Come riportato precedentemente, il primo passo è stato selezionare il servizio di terze parti da utilizzare per effettuare le chiamate online.
Le possibili opzioni considerate sono state:

- Cisco Webex
- Google Meet
- Zoom

Dopo un'attenta lettura della documentazione e delle funzionalità offerte dai vari servizi, la scelta è ricaduta su **Cisco Webex** per i seguenti motivi:

- **Facilità di integrazione**: Cisco Webex consente di integrare facilmente la propria applicazione nel loro sito, permettendo di abilitare tutte le funzionalità necessarie in modo comodo e rapido.
- **Gestione tramite API RESTful**: la gestione dei meeting avviene interamente attraverso API RESTful, le quali possono essere chiamate da qualsiasi ambiente, garantendo una notevole flessibilità.

- **Completezza delle API**: Cisco Webex offre una vasta gamma di API per soddisfare qualsiasi esigenza, ognuna delle quali è altamente personalizzabile in base alle specifiche necessità del progetto.
- **Documentazione chiara ed efficace**: la documentazione fornita da Cisco è ben strutturata e di facile comprensione, facilitando il processo di sviluppo.
- **Ambiente di prova delle API**: Cisco Webex offre la possibilità di testare le API in un ambiente dedicato senza la necessità di effettuare il login, utilizzando esempi predefiniti o personalizzati.
- **Supporto rapido ed efficiente**: in caso di necessità, il supporto fornito da Cisco è tempestivo e competente.

4.3 Configurazione di Webex

Per utilizzare i servizi offerti da Cisco Webex, è necessario compiere una serie di passaggi preliminari per ottenere un **access token**, che verrà utilizzato per autenticare tutte le chiamate alle API.

1. **Creazione di un account**: il primo passo consiste nel creare un account sul sito Webex for Developers, che permette l'accesso agli strumenti necessari per lo sviluppo.
2. **Creazione dell'integrazione**: successivamente si deve creare l'integrazione per l'applicazione sul proprio account. Durante questo processo sono stati specificati:
   - Il **nome dell'applicazione**
   - Una **descrizione dell'applicazione**
   - Un'**icona rappresentativa**
   - Gli **scopes**: di fondamentale importanza sono gli scopes. Questi definiscono tutte le operazioni che il nostro account potrà andare ad eseguire quando si effettuano chiamate alle API. È necessario selezionare tutti gli scopes necessari alle finalità del progetto, con la possibilità di abilitarli o disabilitarli in qualsiasi momento.
3. **Richiesta di un account sandbox**: per effettuare test senza limitazioni durante lo sviluppo, è necessario richiedere un account Sandbox tramite il proprio account creato in precedenza. Una volta ottenute le credenziali, si può usufruire di questo account speciale per accedere come amministratore di un'organizzazione e gestire tutti i meeting e le relative impostazioni attraverso una comoda interfaccia, Webex Control Hub. [11]
4. **Recupero dell'access token**: l'ultimo passaggio è stato ottenere l'access token necessario per autenticare le chiamate alle API. Bisogna effettuare una richiesta POST all'API `https://webexapis.com/v1/access_token` con il seguente body:

```
{
    "grant_type": "authorization_code",
    "client_id": "1234567890abcdef123456",
    "client_secret": "abcdef1234567890abcdef1234567890",
    "code":"12345678abcdef12345678abcdef"
}
```

I valori di `client_id`, `client_secret`, e `code` sono stati reperiti dalla pagina di integrazione dell'applicazione sul proprio account Webex for Developers. Se tutti i dati passati sono corretti, l'access token viene restituito come risposta.

4.4 Interfaccia Utente

L'obiettivo dell'interfaccia utente era fornire una visualizzazione chiara e intuitiva dei meeting programmati. A tal fine, è stato scelto un calendario come elemento principale dell'interfaccia.

In particolare, si è deciso di utilizzare *FullCalendar* [12], una famosa libreria Javascript open source che si integra perfettamente con React.

Grazie al **MUI Theme**, l'utente ha la possibilità di personalizzare i colori principali del sito a proprio piacimento tramite una pratica interfaccia laterale:

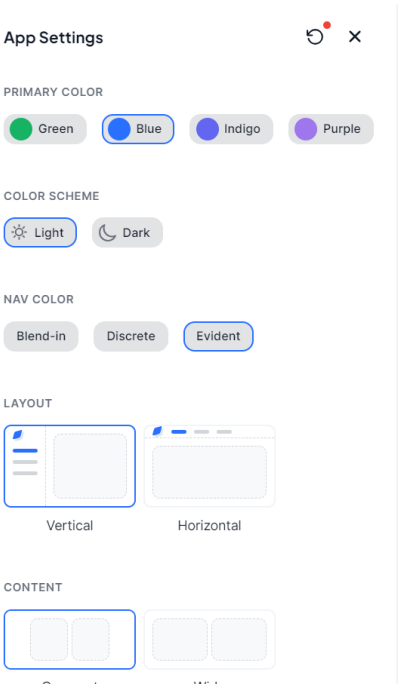


Figura 4.1: Tema chiaro

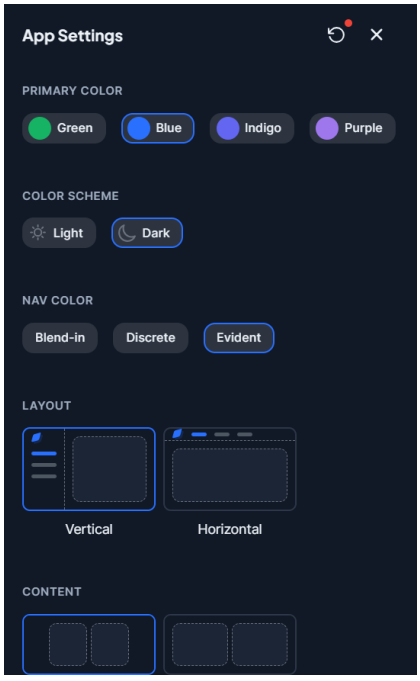


Figura 4.2: Tema scuro

Un esempio di visualizzazione della pagina utilizzando entrambi i temi:

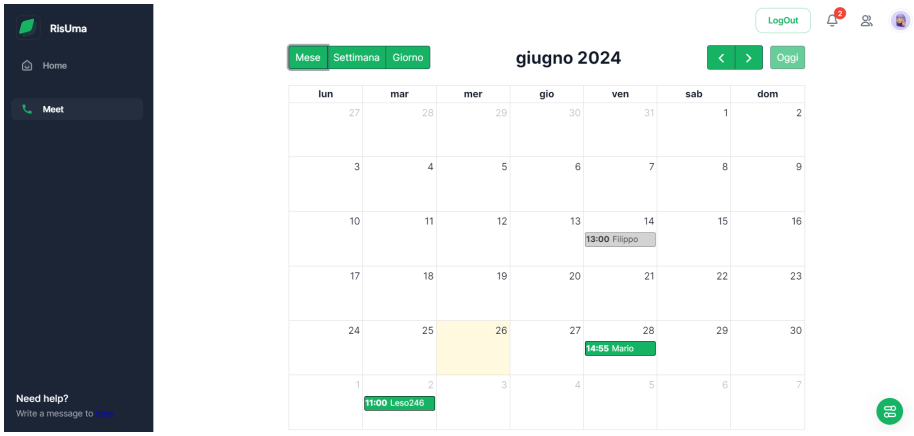


Figura 4.3: Visualizzazione Calendario - tema chiaro

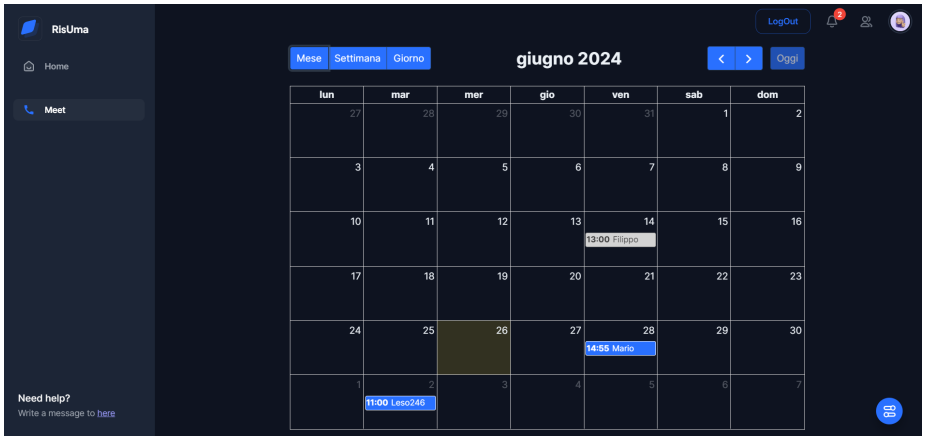


Figura 4.4: Visualizzazione Calendario - tema scuro

Per far capire che un evento è concluso, viene mostrato con uno sfondo grigio. Nel caso siano presenti più eventi in un giorno nella visualizzazione mensile del calendario, è possibile utilizzare un pratico popup per avere una visione migliore:

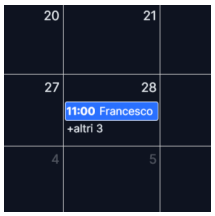


Figura 4.5: + altri meet

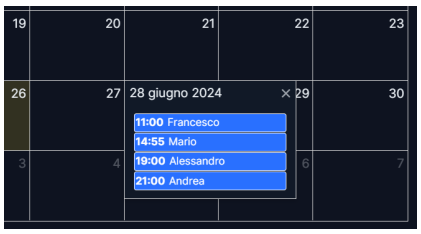


Figura 4.6: popup meet

È anche disponibile una visualizzazione settimanale e giornaliera:

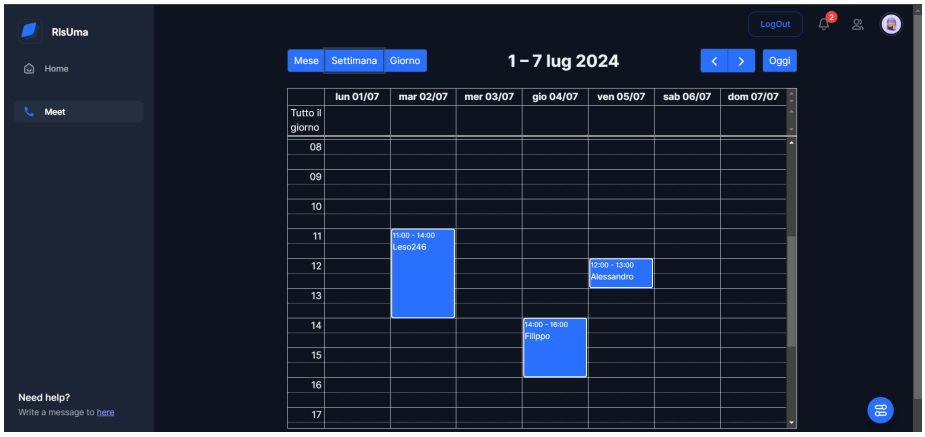


Figura 4.7: Visualizzazione Settimanale - tema scuro

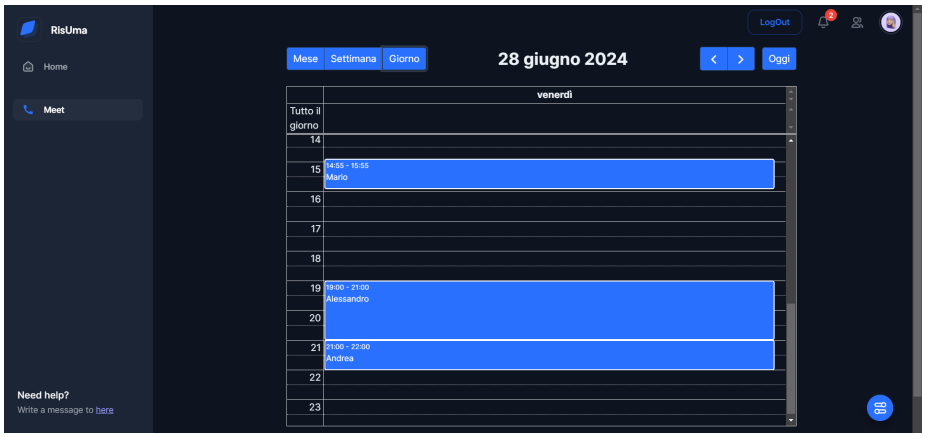


Figura 4.8: Visualizzazione Giornaliera - tema scuro

In seguito si entrerà nel dettaglio dei restanti elementi dell'interfaccia utente.

4.5 Creazione della tabella a database

Per garantire un'efficace gestione dei meeting, è stato necessario creare una tabella dedicata nel database aziendale. Questa tabella è stata progettata per memorizzare tutte le informazioni rilevanti relative ai meeting, le quali saranno utilizzate da `FullCalendar` [12] per permettere la corretta visualizzazione e gestione da parte degli utenti.

Per collegare un evento nel database aziendale con il corrispondente meeting su Webex, viene memorizzato l'ID univoco assegnato da Webex al momento della creazione del meeting. Inoltre, per ragioni di efficienza e praticità, viene salvato anche il link del meeting, generato anch'esso al momento della creazione.

Viene riportata di seguito la query con cui è stata generata la tabella:

```
CREATE TABLE meet (
    id BIGSERIAL PRIMARY KEY,
    data_inizio TIMESTAMP NOT NULL,
    data_fine TIMESTAMP NOT NULL,
    id_utente BIGINT NOT NULL,
    id_azienda BIGINT NOT NULL,
    link VARCHAR(200) NOT NULL,
    webex_id VARCHAR(200) NOT NULL,
    FOREIGN KEY (id_utente) REFERENCES utente(id_utente),
    FOREIGN KEY (id_azienda) REFERENCES aziende(id_azienda)
);
```

id	data_inizio	data_fine	id_utente	id_azienda	link	webex_id
50	2024-06-28 15:00:00	2024-06-28 16:00:00	73	51	webex.com/meet/ex1	93d7d864bd9b4
51	2024-07-13 14:30:00	2024-07-13 16:30:00	34	62	webex.com/meet/ex2	34421188b307b9

Tabella 4.1: Tabella a database di esempio

Nota: i timestamp a database vengono automaticamente visualizzati in un formato leggibile da un umano.

Segue una descrizione dettagliata delle colonne:

1. **id**:
   - **Tipo**: BIGSERIAL
   - **Descrizione**: chiave primaria della tabella. È un identificatore univoco generato automaticamente per ogni record.
2. **data_inizio**:
   - **Tipo**: TIMESTAMP
   - **Descrizione**: indica la data e l'ora di inizio del meeting. Questo campo non può essere nullo.
3. **data_fine**:
   - **Tipo**: TIMESTAMP
   - **Descrizione**: indica la data e l'ora di fine del meeting. Questo campo non può essere nullo.
4. **id_utente**:
   - **Tipo**: BIGINT
   - **Descrizione**: identificatore dell'utente associato al meeting. Questo campo è una chiave esterna che fa riferimento alla colonna `id_utente` della tabella `utente`. Non può essere nullo.

5. **id_azienda**:
    - **Tipo**: BIGINT
    - **Descrizione**: identificatore dell'azienda associato al meeting. Questo campo è una chiave esterna che fa riferimento alla colonna `id_azienda` della tabella `azienda`. Non può essere nullo.
6. **link**:
    - **Tipo**: VARCHAR(200)
    - **Descrizione**: contiene il link univoco del meeting generato da Webex.
7. **webex_id**:
    - **Tipo**: VARCHAR(200)
    - **Descrizione**: id univoco del meeting generato da Webex. Viene utilizzato per gestire tutte le operazioni che si effettuano sul meeting tramite le API di Webex.

Dato l'utilizzo di **JPA**, si è resa necessaria la mappatura della tabella a backend:

```
@Entity
@Table(name = "meet")
@Data
public class Meet {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  @Column(name = "id", nullable = false)
  private long id;

  @Column(name = "data_inizio", nullable = false)
  private Timestamp dataInizio;

  @Column(name = "data_fine",nullable = false)
  private Timestamp dataFine;

  @ManyToOne
  @JoinColumn(name = "id_utente", referencedColumnName = "id_utente", nullable = false)
  private Utente utente;

  @ManyToOne
  @JoinColumn(name = "id_azienda", referencedColumnName = "id_azienda", nullable = false)
  private Azienda azienda;

  @Column(name = "link", nullable = false)
  private String link;

  @Column(name = "webex_id", nullable = false)
  private String webexId;
}
```

Segue una breve spiegazione del codice:

- **@Entity**: indica che questa classe è un'entità JPA, mappata a una tabella del database.
- **@Table(name = "meet")**: specifica il nome della tabella a database a cui è mappata questa entità.

- **@Data**: un'annotazione di Lombock che genera automaticamente getter, setter, toString, equals, e hashCode per la classe.
- **@Id**: definisce il campo id come la chiave primaria della tabella.
- **@GeneratedValue(strategy = GenerationType.IDENTITY)**: delega al database la generazione dei valori della chiave primaria.
- **@Column(name="", nullable=false)**: definisce il nome della colonna nel database e imposta che il valore del campo non può essere NULL.
- **@ManyToOne**: indica una relazione molti-a-uno, dove ogni Meet è associato a un singolo Utente e una singola Azienda, ma ogni Utente e Azienda possono essere associati a più Meet.
- **@JoinColumn(name = "", referencedColumnName = "", nullable = false)**: specifica la colonna da utilizzare per la join con un'altra entità, definendo il nome della colonna locale e la colonna di riferimento dell'altra entità.

Il tipo e il nome definiti sotto ogni annotazione **@Column** sono quelli utilizzati per gestire l'entità. Ad esempio, se si dispone di un oggetto `Meet` denominato `meet` e si desidera ottenere il timestamp che indica la data di inizio, è sufficiente invocare `meet.getDataInizio()`. Questo illustra la potenza di JPA, che semplifica l'accesso e la manipolazione degli attributi delle entità.

4.6 Creazione dei meeting

4.6.1 Frontend

Attualmente, non è implementato alcun metodo per la creazione dei meeting attraverso il frontend del sistema. L'idea di base è che gli utenti possano cercare annunci di lavoro creati dalle aziende utilizzando una barra di ricerca dedicata. Gli utenti potrebbero inserire parole chiave, filtri di posizione o settore, e altre specifiche per trovare gli annunci che corrispondono ai loro interessi e competenze. Allo stesso modo, le aziende potrebbero cercare utenti basandosi sulle informazioni presenti nei loro profili, come competenze, esperienze lavorative, certificazioni, e altre informazioni pertinenti.

Dopo aver individuato un annuncio di interesse o un utente potenziale, è possibile richiedere un incontro mediante un apposito pulsante, attraverso il quale si forniscono le proprie disponibilità per il meeting. L'utente o l'azienda potrebbe specificare una serie di fasce orarie e date preferite, offrendo così flessibilità alla controparte. Nel caso in cui la controparte accetti l'invito scegliendo una data tra quelle disponibili, il sistema dovrebbe procedere automaticamente con la creazione del meeting, aggiornando i calendari delle parti coinvolte.

Tuttavia, sussiste un problema fondamentale: al momento, manca l'implementazione della logica di ricerca degli annunci e degli utenti, la creazione degli annunci da parte delle aziende e il completamento dei profili da parte degli utenti, rendendo impossibile anche la relativa visualizzazione. Di conseguenza, la funzionalità per la creazione automatica dei meeting non è stata implementata.

4.6.2 Backend

La creazione di un meeting nel backend viene gestita tramite una chiamata POST all'API `/addMeet` con il seguente body:

```
{
    "data_inizio": "2024-06-25T16:00:00.000Z",
    "data_fine": "2024-06-25T17:00:00.000Z",
    "azienda": "techSPA@gmail.com",
    "utente": "mario@gmail.com",
    "invitati": [azienda1@outlook.com, azienda2@virgilio.it]
}
```

- **azienda**: l'email con cui l'azienda si è registrata su RisUma.
- **utente**: l'email con cui l'utente si è registrata su RisUma.
- **invitati**: un array contenente le email di ulteriori invitati, nel caso più persone dell'azienda volessero partecipare al colloquio. Questo attributo è opzionale; può essere vuoto o assente, e la creazione del meeting andrà comunque a buon fine.

Controller

```
1  @PostMapping(value = "/addMeet", consumes = "application/json")
2  public ResponseEntity<?> addMeeting(@RequestBody ObjectNode obj) {
3    try {
4     Mono<Boolean> resultMono = serviceMeet.addMeet(obj);
5     Boolean result = resultMono.toFuture().get();
6
7     if (result) {
8       return new ResponseEntity<>(result, HttpStatus.OK);
9     } else {
10      return new ResponseEntity<>(result, HttpStatus.INTERNAL_SERVER_ERROR);
11    }
12   } catch (Exception e) {
13    e.printStackTrace();
14    return new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
15   }
16 }
```

- **Riga 1**: l'annotazione indica che il metodo risponde a richieste HTTP POST sull'URL /addMeet e che il body della richiesta deve essere in formato JSON.
- **Riga 2**: questo è il metodo che gestisce la richiesta HTTP.
  - **ResponseEntity<?>**: `ResponseEntity` è una classe di Spring che rappresenta l'intera risposta HTTP inviata al client. Il tipo generico "<?>" indica che il corpo della risposta può essere di qualsiasi tipo. In altre parole, il metodo può restituire una risposta che può contenere dati di qualsiasi tipo.
  - **@RequestBody ObjectNode obj**: è un'annotazione di Spring che indica al framework di utilizzare il corpo della richiesta HTTP per popolare `obj`, `ObjectNode` è un oggetto fornito dalla libreria **Jackson** utile a rappresentare JSON in Java.

- **Riga 4**: all'interno del blocco try, viene chiamato il **service** attraverso `serviceMeet.addMeet(obj)`, che restituisce un oggetto `Mono<Boolean>`.
- **Riga 5**: il risultato del Mono viene convertito in un Future e poi ottenuto. Questo metodo blocca l'esecuzione finché il risultato non è disponibile.
- **Riga 7**: se il risultato è `true`, viene restituita una risposta HTTP con lo stato OK (200), altrimenti viene restituita una risposta con lo stato INTERNAL_SERVER_ERROR (500).
- **Riga 12**: se si verifica un'eccezione durante l'esecuzione, l'eccezione viene stampata nello stack trace e viene restituita una risposta con lo stato INTERNAL_SERVER_ERROR (500), contenente il messaggio dell'eccezione.

Service

```
1  public Mono<Boolean> addMeet(ObjectNode obj) {
2    try {
3      Azienda azienda = aziendaService.trovaPerEmail(obj.get("azienda").asText());
4      Utente utente = utenteService.trovaPerEmail(obj.get("utente").asText());
5
6      String usernameAzienda = azienda.getAnagraficaAz().getNome();
7      String usernameUtente = utente.getUsername();
8
9      return CreateMeeting
10       .addWebexMeet(webexAPI, obj, usernameAzienda, usernameUtente)
11       .flatMap(response -> {
12
13        JsonNode jsonResponse = (JsonNode) response;
14        String webLink = jsonResponse.get("webLink").asText();
15        String meetingID = jsonResponse.get("id").asText();
16
17        CreateMeeting.addMeetToDatabase(obj, azienda, utente, repo, webLink, meetingID);
18
19        return Mono.just(true);
20     }).doOnError(error -> {
21         System.err.println("Errore durante la creazione del meeting su Webex: "
22         + error.getMessage());
23     }).onErrorReturn(false);
24   } catch (Exception e) {
25     e.printStackTrace();
26     return Mono.just(false);
27     }
28 }
```

- **Righe 3-7**: vengono recuperati dal database il nome dell'azienda e lo username dell'utente invitati al meeting. Questo passaggio è fondamentale perché, come precedentemente menzionato, si intende limitare la comunicazione tra candidati e aziende esclusivamente attraverso RisUma. Poiché Webex invia una mail di notifica alla creazione del meeting, e non si voleva disabilitare questa funzione, nella mail verranno mostrati i rispettivi username al posto degli indirizzi email degli invitati.
- **Riga 10**: viene invocato il metodo responsabile della creazione del meeting sul database di Webex.
- **Righe 11-19**: se la creazione del meeting su Webex ha avuto successo, il flusso d'esecuzzione passa al blocco `flatmap`. In questo blocco, vengono recuperati l'`ID` e il `link` della riunione dal JSON di risposta e viene

chiamato il metodo per aggiungere il meeting anche nel database aziendale.

Nota: è fondamentale che, se la creazione del meet su Webex non ha avuto successo, il metodo ***`addMeetToDatabase`*** *non venga eseguito. Altrimenti, si rischierebbe di mostrare a frontend riunioni inesistenti. Per tale motivo nel caso in cui venga generato un errore da Webex il flusso d'esecuzione non viene passatto al blocco* ***`flatmap`****.*

Repository

Metodo per creare il meet su Webex:

```
1  public static Mono<JsonNode> addWebexMeet(
2      WebClient webexAPI,
3      ObjectNode obj,
4      String usernameAzienda,
5      String usernameUtente) {
6
7    String requestBody = createAddWebexMeetBody(obj, usernameAzienda, usernameUtente);
8
9    return webexAPI.post()
10     .uri(MeetingAPI.ADD_WEBEX_MEET)
11     .header(HttpHeaders.AUTHORIZATION, "Bearer " + MeetingAPI.TOKEN)
12     .contentType(MediaType.APPLICATION_JSON)
13     .bodyValue(requestBody)
14     .retrieve()
15     .bodyToMono(String.class).map(responseBody -> {
16
17         ObjectMapper objectMapper = new ObjectMapper();
18
19         try {
20             JsonNode jsonResponse = objectMapper.readTree(responseBody);
21             return jsonResponse;
22         } catch (JsonProcessingException e) {
23             throw new RuntimeException("Errore durante il parsing della risposta JSON", e);
24         }
25     }).onErrorMap(e ->
26                     new RuntimeException("Errore durante la chiamata all'API di Webex", e));
27 }
```

- **Riga 7**: `createAddWebexMeetBody` è un metodo di utilità che permette di creare il body della richiesta per l'API di Webex. Esempio:

```
{
    "title": "Riunione tra Mario e Tech SPA",
    "start": "2024-06-25T16:00:00",
    "end": "2024-06-25T17:00:00",
    "invitees": [
        {
            "email": "techSPA@gmail.com",
            "displayName": "Tech SPA",
            "coHost": false
        },
        {
            "email": "mario@gmail.com",
            "displayName": "Mario",
            "coHost": false
        }
    ],
    "timezone": "Europe/Rome",
    "enabledJoinBeforeHost": true,
    "enableConnectAudioBeforeHost": true,
    "joinBeforeHostMinutes": 15,
    "scheduledType": "meeting",
    "enableAutomaticLock": true,
    "automaticLockMinutes": 60
}
```

- **title**: titolo della riunione.
- **start**: data e ora d'inizio della riunione, formattata secondo lo standard ISO 8601.
- **end**: data e ora di fine della riunionw, formattata secondo lo standard ISO 8601.
- **invitees**: lista degli invitati alla riunione. Webex invierà una mail agli indirizzi specificati, il nome visualizzato è il valore contenuto in `displayName`.
- **timezone**: il fuso orario in cui si terrà il meet.
- **enabledJoinBeforeHost**: permette ai partecipanti di unirsi alla riunione anche senza la presenza dell'host. Questo attributo è fondamentale poiché l'azienda è l'organizzatrice di tutti i meeting ma non parteciperà a nessuno di essi.
- **enableConnectAudioBeforeHost**: permette ai partecipanti di utilizzare l'audio anche in assenza dell'host.
- **joinBeforeHostMinutes**: indica quanti minuti rispetto alla data di inizio della riunione i partecipanti possono unirsi anche senza la presenza dell'host.
- **scheduledType**: indica il tipo di riunione, in questo caso è un meeting.
- **enableAutomaticLock**: dopo un certo periodo di tempo, nessuno può partecipare al meeting senza essere accettato dall'host.
- **automaticLockMinutes**: indica il periodo di tempo dopo il quale la riunione verrà automaticamente bloccata. Questo valore è calcolato come la differenza tra `end` e `start`, in modo che, dopo la data di fine prefissata, nessuno possa più partecipare.

- **Righe 9-15**: viene effettuata una chiamata POST all'API `https://webexapis.com/v1/meetings`, utilizzando il token d'autorizzazione generato e passando come body il JSON precedentemente creato.
- **Righe 20-21**: se la richiesta è andata a buon fine, viene restituito come risposta il JSON fornito dall'API di Webex.

Metodo per creare il meet a database:

```
1  public static void addMeetToDatabase(
2      ObjectNode obj,
3      Azienda azienda,
4      Utente utente,
5      MeetJPA repo,
6      String link,
7      String webexId) {
8
9    try {
10
11     Meet meet = new Meet();
12
13     Timestamp dataInizio =
           Timestamp.valueOf(UtiltyMeet.transformUTCIntoRomeTimezone(obj.get("data_inizio").asText()));
14     Timestamp dataFine =
           Timestamp.valueOf(UtiltyMeet.transformUTCIntoRomeTimezone(obj.get("data_fine").asText()));
15
```

```
16      meet.setDataInizio(dataInizio);
17      meet.setDataFine(dataFine);
18      meet.setAzienda(azienda);
19      meet.setUtente(utente);
20      meet.setLink(link);
21      meet.setWebexId(webexId);
22
23      repo.save(meet);
24
25    } catch (Exception e) {
26      throw new RuntimeException("Errore durante la creazione del meeting al database", e);
27    }
28  }
```

Questo metodo utilizza JPA per creare facilmente un record nella tabella **meet** del database.

Nota: il frontend fornisce le date nel fuso orario UTC+0. Pertanto, è stato creato un metodo `transformUTCIntoRomeTimezone` *per convertire le date nel fuso orario di Roma. Questo garantisce che le date siano corrette rispetto alla localizzazione del sistema.*

4.7 Recupero dei meeting

4.7.1 Frontend

Ogni volta che la pagina viene caricata, viene chiamato uno `useEffect`.

```
1  useEffect(() => {
2      const jwtToken = TokenAuth.getTokenAuth();
3
4      const getMeet = async () => {
5          try {
6              const response = await ritornaMeeting(jwtToken);
7              setEvents(response.data);
8          } catch (error) {
9              showPopup('Errore durante il caricamento dei colloqui', false);
10             console.error('Errore durante il recupero degli eventi:', error);
11         }
12     };
13     getMeet();
14 }, []);
```

- **Riga 2**: viene recuperato il `jwtToken` dell'utente. Abbreviazione di **JSON Web Token**, il `jwtToken` è uno standard (RFC 7519) per trasmettere in modo sicuro dei dati sotto forma di oggeto JSON. [13] Viene generato quando l'utente effettua il login e contiene informazioni utili riguardo a esso.
- **Righe 4-11**: viene definita la funzione `getMeet` che chiama il backend per il recupero degli eventi
  - **API**:

```
1  const api_url =
2    import.meta.env.MODE === 'development'
3      ? import.meta.env.VITE_API_GESTUT_URL_DEV
4      : import.meta.env.VITE_API_GESTUT_URL_PROD;
5
6  const get_meet = api_url + 'meet/getMeet?jwtToken=';
7
```

```
8  export const ritornaMeeting = (jwtToken: string) => {
9    return axios.get(get_meet + jwtToken);
10 };
```

- **Riga 7**: `setEvents` è uno `useState`. In questo caso è usato per impostare gli eventi di `Fullcalendar` in modo da poterli visualizzare. `Fullcalendar` accetta un array di eventi così formattati:

```
[
    {
        "id": 50,
        "start": "2024-06-28T15:00:00",
        "end": "2024-06-28T16:00:00",
        "editable": true|false,
        "eventStartEditable": true|false,
        "extendedProps": {
            "nomeUtente": "Mario",
            "nomeAzienda": "Tech SPA",
            "link": "webex.com/meet/ex1",
            "webexId": "93d7d864bd9b4"
        }
    },
]
```

  - **editable** ed **eventStartEditable**: sono attributi degli eventi di `Fullcalendar` che permettono di trascinare un evento sul calendario per modificarne la data e/o l'ora. Questo comportamento verrà discusso nella sezione dedicata alla modifica dei meeting.
  - **extendedProps**: in questo JSON è possibile inserire tutte le coppie di chiave-valore che risultano utili allo sviluppo. In questo caso, di particolare importanza sono il `link` del meeting e il suo `webexId`.

- **Righe 9-10**: nel caso in cui il caricamento dei colloqui fallisca, utilizzando nuovamente degli `useState` viene mostrato un popup all'utente contenente il messaggio di errore *"Errore durante il caricamento dei colloqui"*.

Una volta che gli eventi sono stati caricati correttamente, vengono visualizzati come precedentemente mostrato all'interno del calendario.

Quando si clicca su un evento, viene visualizzato un `Dialog` contenente i dettagli del meet recuperati dalla funzione `getMeet`. Questo `Dialog` cambia a seconda che l'utente loggato sia un candidato o un'azienda, sfruttando la renderizzazione condizionale di React. [14]

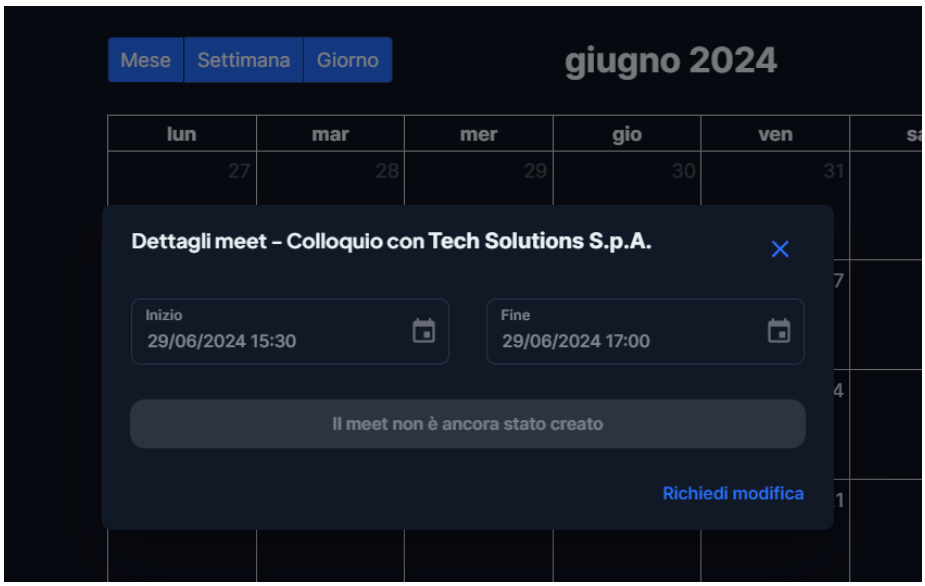


Figura 4.9: Visualizzazione candidato - evento non iniziato

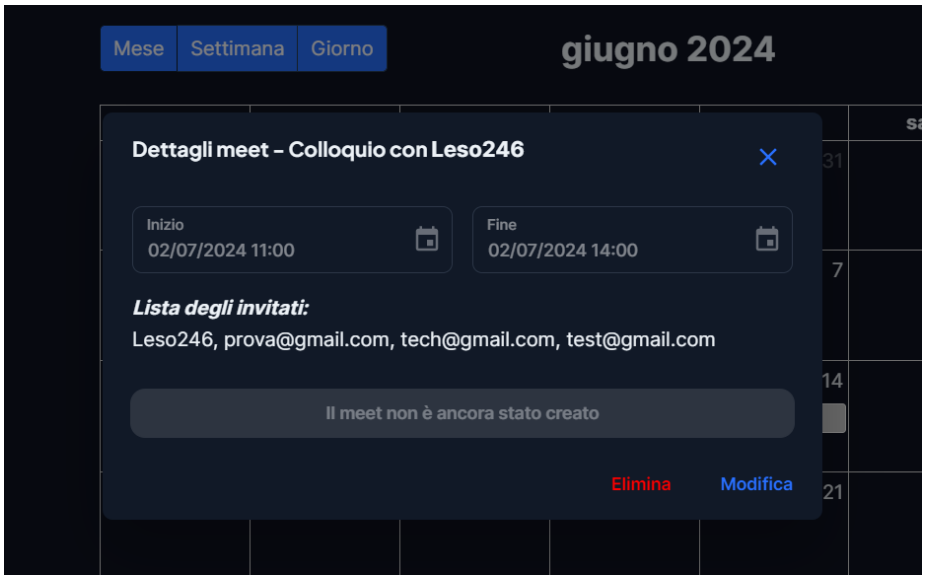


Figura 4.10: Visualizzazione azienda - evento non iniziato

Quando mancano meno di 10 minuti all'inizio del meet, il pulsante per partecipare al meet viene abilitato. Cliccandoci, si viene reinderizzati al meet su Webex. In questo stato, l'evento non è più modificabile né eliminabile.

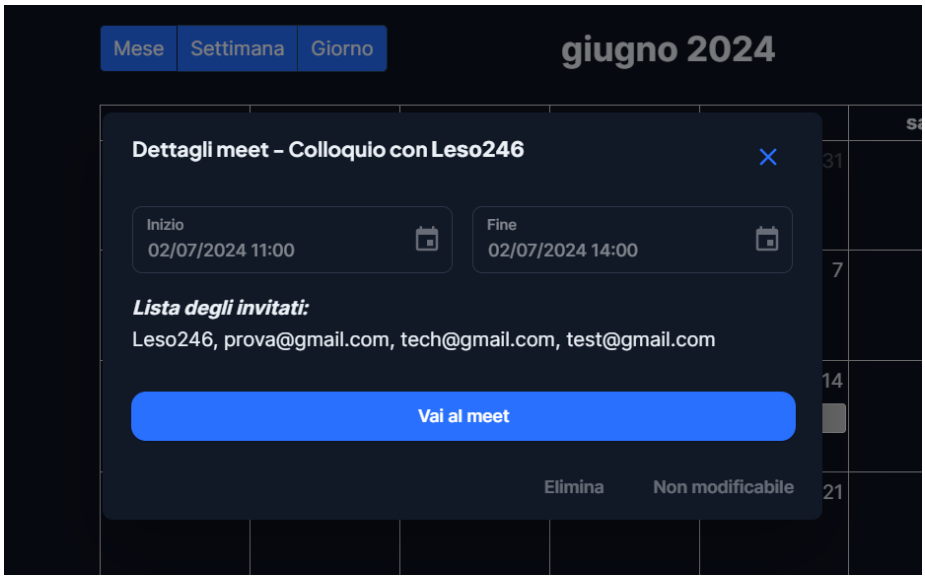


Figura 4.11: Visualizzazione azienda - evento iniziato

Se l'evento è già concluso, è comunque possibile visualizzarne i dettagli per avere uno storico dell'incontro. In questo caso, non è possibile effettuare nessun'altra operazione sul meet.

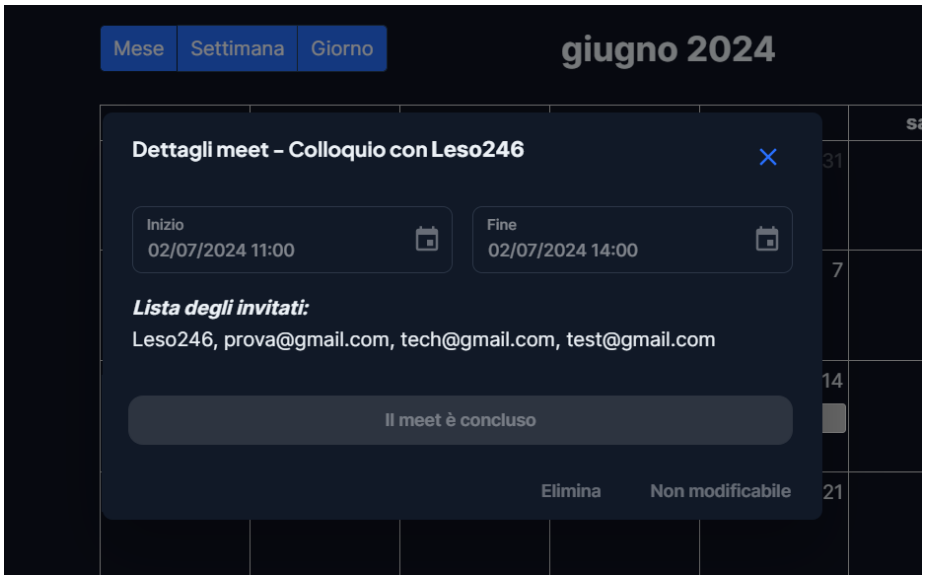


Figura 4.12: Visualizzazione azienda - evento concluso

La logica per abilitare/disabilitare i pulsanti è gestita interamente a frontend attraverso l'utilizzo di uno `useEffect`: ogni volta che un evento viene selezionato e viene aperto il relativo `Dialog`, il sistema verifica la data attuale rispetto alla data di inizio e di fine del meeting. Questo controllo determina se mancano più di 10 minuti all'inizio del meeting, se il meeting è già iniziato o se è concluso. Sulla base di questi valori, i pulsanti vengono abilitati o disabilitati di conseguenza.

È importante notare che al momento della creazione del meeting, l'orario di accesso per gli utenti è impostato 15 minuti prima della data di inizio. Questo permette di abilitare il pulsante "Vai al meet" in modo che gli utenti possano accedere direttamente senza attese o conferme.

```
useEffect(() => {
    if (!selectedEvent) return;

    const { start, end } = selectedEvent;
    const now = new Date();

    const eventStartDate = new Date(start);
    const eventFinishDate = new Date(end);

    const minutesUntilStart = differenceInMinutes(eventStartDate, now);
    const isLessThan10Minutes = minutesUntilStart < 10;
    const isEventFinished = now > eventFinishDate;

    setIsLessThan10Minutes(isLessThan10Minutes);
    setIsEventFinished(isEventFinished);
}, [selectedEvent]);
```

Recupero degli invitati

Per quanto riguarda gli invitati a un meet, questi sono visualizzabili **solo** dall'azienda. Poiché non vengono salvati nel database, è necessario fare una chiamata all'API di Webex per recuperarli. Considerato troppo oneroso effettuare una chiamata per ogni singolo meet durante il caricamento della pagina, l'API `/getMeetInvitees` viene chiamata ogni volta che si clicca su un evento. Sono stati effettuati 30 test e il tempo di risposta medio è risultato essere di 863 ms, motivo per cui questa soluzione è stata ritenuta adeguata. Prima che gli invitati vengano caricati, viene visualizzata una semplice scritta *"Caricamento..."* di fianco alla dicitura *"Lista degli invitati:"* del `Dialog`.

```
const handleEventClick = async (arg: any) => {
    setInvitees([]); // Resetta l'array di invitati
    setSelectedEvent(arg.event);
    setOpen(true); // Apre il Dialog
    if (isAzienda) {
        const payload = {
        id: arg.event.id,
        webexId: arg.event.extendedProps.webexId,
        };
        try {
            const response = await getMeetInvitees(payload);
            if (response.status === 200) {
              setInvitees(response.data);
            } else {
              console.error('Errore nel recupero degli invitati:', response.statusText);
            }
          } catch (error) {
            console.error('Errore nel recupero degli invitati:', error);
          }
        } ...
```

4.7.2 Backend

Il recupero dei meeting nel backend viene gestita tramite una chiamata GET all'API `/getMeet` con il parametro `jwtToken`.

Controller

```
1  @GetMapping(value = "/getMeet")
2  public ResponseEntity<?> getMeeting(@RequestParam("jwtToken") String jwtToken) {
3    try {
4       String email = gestioneToken.getEmailFromToken(jwtToken);
5       boolean isAzienda = gestioneToken.isAzienda(jwtToken);
6
7       Long idUtente = 0l;
8
9       if (isAzienda) {
10         idUtente = serviceAzienda.trovaPerEmail(email).getId();
11      } else {
12         idUtente = serviceUtente.trovaPerEmail(email).getId();
13      }
14
15      List<Map<String, Object>> meets = serviceMeet.getMeet(isAzienda, idUtente);
16
17      return new ResponseEntity<>(meets, HttpStatus.OK);
18
19    } catch(Exception e) {
20      e.printStackTrace();
21      return new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
22    }
23 }
```

- **Riga 1**: l'annotazione indica che il metodo risponde a richieste HTTP GET sull'URL /getMeet.
- **Riga 2**: questo è il metodo che gestisce la richiesta HTTP.
  - **@RequestParam("jwtToken") String jwtToken**: è un'annotazione di Spring che indica che la richiesta deve contenere un parametro `jwtToken`, il quale assegnato alla variabile di tipo `String` denominata `jwtToken`.
- **Righe 4-13**: viene recuperata l'email dal token e verifica se il token appartiene a un'azienda o a un utente. Una volta determinato il tipo di utente, viene recuperato l'ID corrispondente nella rispettiva tabella. Questo passaggio è cruciale perché è necessario mostrare solo i meet specifici di quell'utente e l'ID, che non è salvato nel `jwtToken`, è utilizzato come riferimento.
- **Riga 15**: viene chiamato il service per recuperare i meet da database.
- **Righe 19-21**: se si verifica un'eccezione durante l'esecuzione, l'eccezione viene stampata nello stack trace e viene restituita una risposta con lo stato INTERNAL_SERVER_ERROR (500), contenente il messaggio dell'eccezione.

Service

```
1  @Autowired
2  private MeetJPA repo;
3
4  public List<Map<String, Object>> getMeet(boolean isAzienda, Long idUtente) {
5    List<MeetDTO> meets;
6
7     if (isAzienda) {
8        meets = repo.getMeetAzienda(idUtente);
9     } else {
10       meets = repo.getMeetUtente(idUtente);
11    }
12
13   List<Map<String, Object>> transformedMeets =
14       meets.stream()
15         .map(meet -> UtiltyMeet.transformMeetDTO(meet, isAzienda))
16         .collect(Collectors.toList());
17
18   return transformedMeets;
19 }
```

- **Righe 1-2**: questa annotazione indica a Spring di iniettare automaticamente un'istanza del bean `MeetJPA` nella variabile `repo` quando viene creata un'istanza di `MeetService`. [15] In questo contesto, la classe `MeetService` ha bisogno di accedere al repository `MeetJPA` per effettuare operazioni CRUD sui meet.
- **Righe 8-12**: a seconda che l'utente sia un'azienda o un candidato, vengono recuperati i meet associati.
- **Righe 14-17**: metodo di utilità che itera sull'array di JSON restituito dal database e formatta ogni elemento in modo che possa essere letto correttamente da `FullCalendar`.

Repository

```
1  @Repository
2  public interface MeetJPA extends JpaRepository<Meet, Long> {
3      public Meet findMeetById(Long id);
4
5      @Query(value = "select new dto.MeetDTO("
6          + "m.id, "
7          + "m.dataInizio, "
8          + "m.dataFine, "
9          + "m.utente.username, "
10         + "m.azienda.anagraficaAz.nome, "
11         + "m.link, "
12         + "m.webexId) "
13         + "from Meet m "
14         + "where m.azienda.id =:id_azienda")
15     public List<MeetDTO> getMeetAzienda(Long id_azienda);
16
17     @Query(value = "select new dto.MeetDTO("
18         + "m.id, "
19         + "m.dataInizio, "
20         + "m.dataFine, "
21         + "m.utente.username, "
22         + "m.azienda.anagraficaAz.nome, "
23         + "m.link, "
24         + "m.webexId) "
25         + "from Meet m "
26         + "where m.utente.id =:id_utente")
27     public List<MeetDTO> getMeetUtente(Long id_utente);
28 }
```

- **Riga 1**: questa annotazione indica che l'interfaccia è una repository Spring. La classe fornisce operazioni CRUD su un oggetto. [16]
- **Riga 3**: questo metodo consente di recuperare un'istanza di `Meet` a database dato il suo id. Spring Data JPA genera automaticamente l'implementazione di questo metodo basandosi sul suo nome. Verrà utilizzato per il recupero degli invitati.
- **Righe 4-26**: queste due query personalizzate recuperano tutti i meet associati all'ID dell'utente, sono un esempio di utilizzo del linguaggio **JPQL**.

È interessante notare l'uso della sintassi `new dto.MeetDTO` nelle query JPQL: è stato necessario creare una classe `MeetDTO` in quanto, se si fosse recuperata l'intera istanza di `Meet`, il database avrebbe restituito come attributi di `utente` e di `azienda` le due istanze nella loro interezza, al posto dei due soli nomi, che sono invece di nostro interesse. Ovviamente ciò avrebbe fatto sorgere un problema di sicurezza, in quanto sarebbero stati restituiti a frontend dati sensibili riguardo le due utenze.

DTO, acronimo di Data Transfer Object, rappresenta un oggetto progettato appositamente per trasferire dati tra processi all'interno di un sistema, in questo caso tra il database e il backend [17]. L'implementazione di `MeetDTO` consente di selezionare solo i campi pertinenti da `Meet` necessari per l'interesse dell'applicazione, escludendo informazioni non richieste e sensibili. Questo approccio non solo migliora le prestazioni dell'applicazione evitando il trasporto di dati non necessari, ma anche rafforza la sicurezza limitando l'esposizione di informazioni sensibili.

```
1  @Data
2  @NoArgsConstructor
3  public class MeetDTO {
4      private long id;
5      private Timestamp start;
6      private Timestamp end;
7      private String nomeUtente;
8      private String nomeAzienda;
9      private String link;
10     private String webexId;
11
12     public MeetDTO(
13         long id,
14         Timestamp start,
15         Timestamp end,
16         String nomeUtente,
17         String nomeAzienda,
18         String link,
19         String webexId) {
20
21         super();
22         this.id = id;
23         this.start = start;
24         this.end = end;
25         this.nomeUtente = nomeUtente;
26         this.nomeAzienda = nomeAzienda;
27         this.link = link;
28         this.webexId = webexId;
29     }
30 }
```

Quando vengono recuperati i vari attributi della query JPQL, viene utilizzato il costruttore della classe `MeetDTO` per creare un oggetto di questa classe che viene poi aggiunto alla `List` che il metoodo restituisce.

Recupero degli invitati

In relazione al recupero degli invitati a backend, si presenta il seguente problema: recuperando le mail di tutti gli invitati al meet da Webex, viene recuperato anche il contatto del candidato. Se questo venisse mostrato a frontend, verrebbe meno l'obiettivo prefissato, ovvero non consentire alle aziende di contattare i candidati al di fuori della piattaforma, almeno fino al primo colloquio. Per risolvere questo problema, una volta ottenuto l'array contenente le mail degli invitati dall'API `https://webexapis.com/v1/meetingInvitees/?meetingId=` di Webex, viene recuperata l'istanza `Utente` del candidato associato al `Meet` grazie al metodo `findMeetById`, successivamente la mail del candidato viene sostituita con il suo nome utente. Si vuole far notare che la mail dell'utente è obbligatoriamente presente, poiché è stata utilizzata per creare il meeting. Se così non fosse, la creazione del meeting non sarebbe andata a buon fine.

```
private static ArrayList<String> removeRisumaUserEmail(
  ObjectNode obj,
  MeetJPA repo,
  ArrayList<String> invitees) {
   try {
      Long meetId = obj.get("id").asLong();
      Utente utente = repo.findMeetById(meetId).getUtente();
      String emailToRemove = utente.getEmail();
      String username = utente.getUsername();

      invitees.remove(emailToRemove);
      invitees.add(username);

      return invitees;
   } catch (Exception e) {
      e.printStackTrace();
      return null;
   }
}
```

4.8 Modifica dei meeting

4.8.1 Frontend

La modifica dei meeting è disponibile unicamente per le aziende. È possibile:

- Modificare data e ora del colloquio.
- Aggiungere ulteriori invitati, inserendo le rispettive email.

Per poter accedere alla sezione di modifica di un evento, è necessario aprirne i dettagli e cliccare su *"Modifica"*. Il `Dialog` caricato è lo stesso utilizzato per visualizzare i dettagli, sfruttando ancora una volta la renderizzazione condizionale di React . [14]

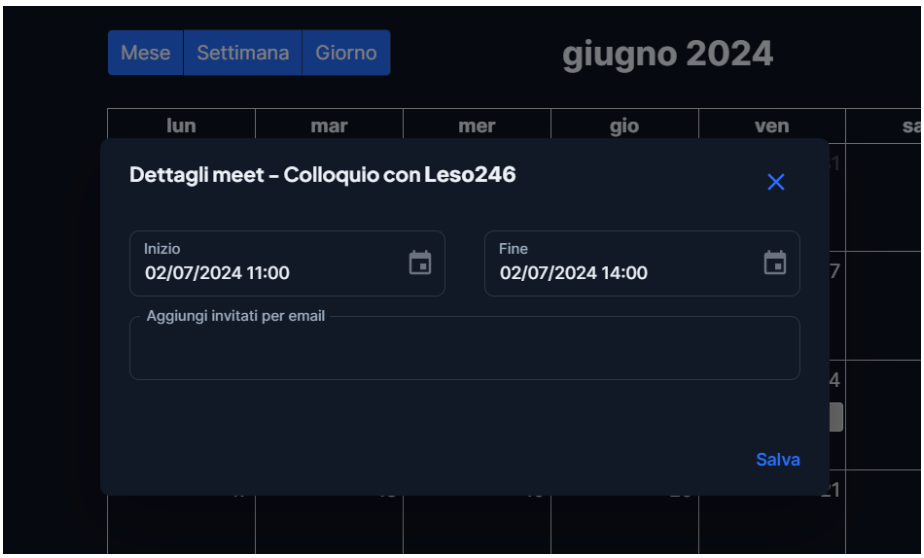


Figura 4.13: Visualizzazione Modifica

Modifica di data e ora

Per cambiare la data e l'ora di un meeting, sono disponibili due modalità:

- selezionare la data e l'ora dai rispettivi `DateTimePicker`. Tutte le date passate sono disabilitate e quindi non selezionabili.

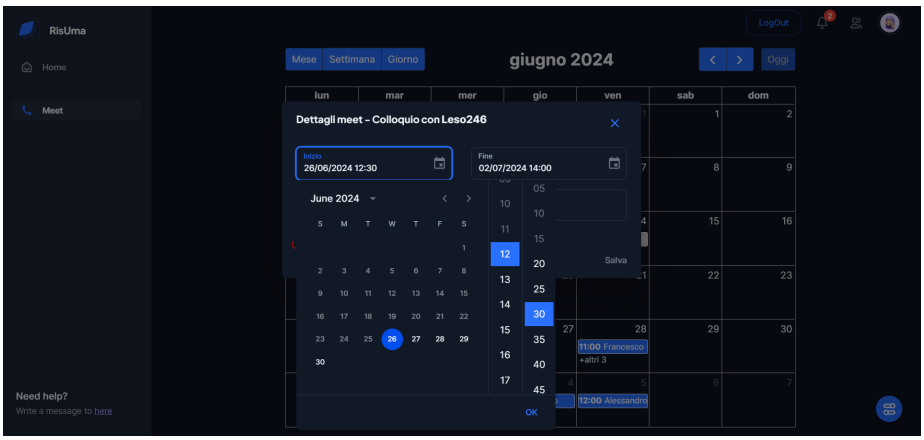


Figura 4.14: Modifica di data e ora

Ci sono tre condizioni da rispettare, le ultime due imposte da Webex:

- la data di fine non può essere antecedente alla data di inizio.
- un meet non può avere una durata inferiore ai 10 minuti.
- un meet non può avere una durata superiore alle 24 ore.

Nel caso si verifichi una di queste tre condizioni, viene visualizzato un messaggio di errore e il pulsante per salvare viene disabilitato.

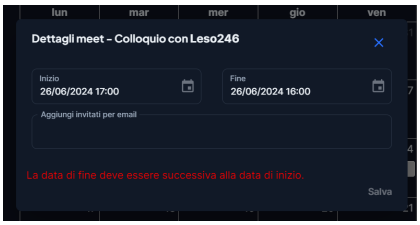


Figura 4.15: errore data fine

Figura 4.16: errore 10 minuti

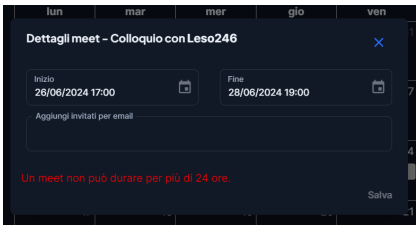


Figura 4.17: errore 24 ore

- Trascinando gli eventi a calendario, rilasciandoli sulla data e l'ora desiderate:
  - Nella visualizzazione mensile, è possibile cambiare solo la data e non l'ora.
  - Nella visualizzazione settimanale, è possibile cambiare la data e l'ora con una precisione di mezz'ora rispetto all'orario di inizio del meeting.
  - Nella visualizzazione giornaliera è possibile cambiare unicamente l'ora, sempre con una precisione di mezz'ora.

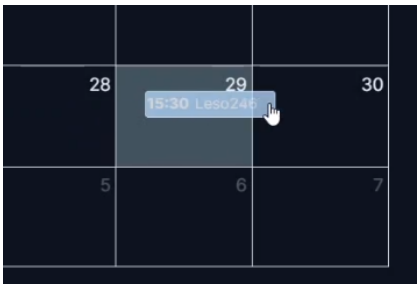


Figura 4.18: trascinamento evento

Figura 4.19: successo nella modifica

Nel caso in cui si cerchi di spostare un evento in una data passata, questo viene riportato alla sua posizione originale e l'utente viene avvisato dell'errore tramite un popup.

Figura 4.20: Data passata selezionata

Anche quando viene riscontrato un errore nella modifica, l'utente viene avvisato attraverso un popup e l'evento riportato alla posizione originale.

Aggiunta degli invitati

Per aggiungere invitati a un meet tramite email, è sufficiente digitare gli indirizzi all'interno dell'apposito `TextField`. Quando l'utente inserisce un'email e sposta il focus su un altro elemento, oppure digita uno tra i seguenti caratteri:

$$\text{Spazio} \mid \text{Invio} \mid , \mid ; \mid .$$

viene effettuato un controllo sull'input digitato. Se l'input viene riconosciuto, attraverso una regex, come un'email valida, questa viene visualizzata all'interno di una card per indicare che è stata correttamente riconosciuta e viene contemporaneamente aggiunta all'array *inviteesList*, uno `useState`, che verrà inviato al backend.

Figura 4.21: Aggiunta degli invitati

Se si desidera modificare un'email, basta cliccarci sopra per ritornare alla visualizzazione del testo. Per eliminarla, è possibile cliccare sulla *X*.
Sorge un problema: se l'utente digita una sola email e, senza spostare il focus

dal `TextField`, clicca sul pulsante *Salva*, l'email digitata non sarà aggiunta come card e quindi non sarà inclusa nell'array di email da inviare al backend. Per tale motivo, al momento del salvataggio, è necessario controllare l'input rimasto all'interno del `TextField` e, se questo risulta essere un'email valida, aggiungerla all'array di invitati.

Salvataggio delle modifiche

Quando si salvano le modifiche, viene chiamata la funzione `modifyMeet`:

```
1  const modifyMeet = async () => {
2
3      const payload = {
4          evento: isDragged ? modifiedDraggedEvent : eventDetails,
5          invitati: inviteesList,
6      };
7
8      try {
9          const response = await modifyMeeting(payload);
10         if (response.status == 200) {
11             showPopup('Modifica avvenuta con successo', true);
12
13             if (!isDragged) {
14                 const updatedEvents = events.map((event) =>
15                 event.id == eventDetails.id ? { ...event, ...eventDetails } : event
16             );
17             setEvents(updatedEvents);
18             }
19         } else
20 ...
```

- **Righe 3-6**: viene costruito l'oggetto da inviare a backend. Questo include il JSON dell'evento modificato e l'array di stringhe contenente le email da aggiungere agli invitati. `isDragged` è un `boolean` che indica se l'evento è stato modificato dall'intrefaccia del `Dialog` o se è stato trascinato, in base al suo valore viene passato l'evento corretto.
- **Righe 13-17**: se l'operazione di modifica dell'evento dall'interfaccia del `Dialog` ha avuto successo, gli eventi del `Fullcalendar` vengono aggiornati manualmente in modo da visualizzare la modifica senza che si debba ricaricare la pagina.

Inoltre, tutte le persone invitate al meeting ricevono una email di notifica riguardante la modifica apportata.

4.8.2 Backend

Controller

```
1  @PostMapping(value = "/modifyMeet", consumes = "application/json")
2  public ResponseEntity<?> modifyMeeting(@RequestBody ObjectNode obj) {
3      try {
4          Mono<Boolean> resultMono = serviceMeet.modifyMeet(obj);
5          Boolean result = resultMono.toFuture().get();
6
7          if (result) {
8              return new ResponseEntity<>(result, HttpStatus.OK);
9          } else {
10             return new ResponseEntity<>(result, HttpStatus.INTERNAL_SERVER_ERROR);
11         }
```

```
12      } catch (Exception e) {
13          e.printStackTrace();
14          return new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
15      }
16  }
```

- **Riga 1**: l'annotazione indica che il metodo risponde a richieste HTTP POST sull'URL /addMeet e che il body della richiesta deve essere in formato JSON.
- **Riga 2**: questo è il metodo che gestisce la richiesta HTTP.
  - **@RequestBody ObjectNode obj**: è un'annotazione di Spring che indica al framework di utilizzare il corpo della richiesta HTTP per popolare `obj`.
- **Riga 4**: all'interno del blocco try, viene chiamato il **service** attraverso `serviceMeet.modifyMeet(obj)`, che restituisce un oggetto `Mono<Boolean>`.
- **Riga 5**: il risultato del Mono viene convertito in un Future e poi ottenuto. Questo metodo blocca l'esecuzione finché il risultato non è disponibile.

Service

```
1   public Mono<Boolean> modifyMeet(ObjectNode obj) {
2       try {
3           JsonNode evento = obj.get("evento");
4
5           JsonNode invitati = obj.get("invitati");
6           List<String> invitatiList = new ArrayList<>();
7           if (invitati != null && invitati.isArray()) {
8               for (JsonNode invitato : invitati) {
9                   invitatiList.add(invitato.asText());
10              }
11          }
12
13          return modifyWebexMeet(webexAPI, evento, invitatiList).flatMap(success -> {
14              if (success) {
15                  ModifyMeeting.modifyDatabaseMeet(repo, evento);
16                  return Mono.just(true);
17              } else {
18                  return Mono.just(false);
19              }
20          }).doOnError(error -> {
21              System.err.println(
22                  "Errore durante la modifica del meeting su Webex: " + error.getMessage());
23          }).onErrorReturn(false);
24
25      } catch (Exception e) {
26          e.printStackTrace();
27          return Mono.just(false);
28      }
29  }
```

- **Righe 5-11**: viene estratto dal body della richiesta l'oggetto *invitati*, contenente le email degli invitati da aggiungere al meeting. Questo viene successivamente convertito in un array di stringhe.
- **Riga 13**: viene invocato il metodo `modifyWebexMeet`, che si trova anch'esso nel Service. Questo metodo gestisce la logica necessaria per effettuare

le due chiamate API a Webex: una per aggiornare l'orario e l'altra per aggiungere gli invitati.

- **Riga 15**: se entrambe le chiamate a Webex hanno esito positivo, viene chiamato il metodo per aggiornare l'orario del meeting nel database.

Repository

Per modificare l'orario di un meeting, viene effettuata una richiesta PATCH all'API `https://webexapis.com/v1/meetings/webexId`.

```
1  public static Mono<Boolean> modifyWebexMeetDate(WebClient webexAPI, JsonNode evento) {
2     try {
3        ObjectNode requestBody = UtiltyMeet.createModifyWebexMeetDateBody(evento);
4        String webexId = evento.get("extendedProps").get("webexId").asText();
5
6        String uri = MeetingAPI.MODIFY_WEBEX_MEET_DATE + webexId;
7
8        return webexAPI
9           .patch()
10          .uri(uri)
11          .header(HttpHeaders.AUTHORIZATION, "Bearer " + MeetingAPI.TOKEN)
12          .contentType(MediaType.APPLICATION_JSON)
13          .bodyValue(requestBody)
14          .retrieve()
15          .toBodilessEntity()
16          .thenReturn(true).onErrorMap(e -> new RuntimeException(
17             "Errore durante la chiamata all'API di Webex per la modifica di un evento", e));
18
19    } catch (Exception e) {
20       e.printStackTrace();
21       return Mono.just(false);
22    }
23 }
```

- **Riga 3**: metodo di utilità per costruire il body della richiesta

```
{
   "start": "2024-07-22T10:00:00",
   "end": "2024-07-22T11:00:00",
   "timezone": "Europe/Rome"
}
```

- **Righe 8-17**: chiamata all'API.

Per aggiungere gli invitati a un meeting, viene effettuata una richiesta POST all'API `https://webexapis.com/v1/meetingInvitees/bulkInsert`.

```
1  public static Mono<Boolean> modifyWebexMeetInvitees(WebClient webexAPI, JsonNode evento,
       List<String> invitatiList) {
2     try {
3        if (invitatiList.size() != 0) {
4           ObjectNode requestBody = createModifyWebexInviteesBody(evento, invitatiList);
5
6           return webexAPI.post().uri(MeetingAPI.MODIFY_WEBEX_MEET_INVITEES)
7              .header(HttpHeaders.AUTHORIZATION, "Bearer " + MeetingAPI.TOKEN)
8              .contentType(MediaType.APPLICATION_JSON).bodyValue(requestBody).retrieve().toBodilessEntity()
9              .thenReturn(true).onErrorResume(e -> {
10                System.err.println(
11                   "Errore durante la chiamata all'API di Webex per l'aggiunta degli invitati
                          a un meet: "
12                   + e.getMessage());
13
14                return Mono.error(e);
15             });
```

```
16        } else {
17           return Mono.just(true);
18        }
19     } catch (Exception e) {
20        e.printStackTrace();
21        return Mono.error(e);
22     }
23  }
```

- **Riga 3**: la richiesta viene eseguita solo se ci sono invitati da aggiungere. In caso contrario, viene restituito `true` e il controllo torna al service.
- **Riga 4**: metodo di utilità per costruire il body della richiesta. `hostEmail` rappresenta l'email dell'account che crea tutti i meeting, mentre `items` è un array di JSON contenente le email degli invitati. È possibile specificare ulteriori attributi per ogni invitato, ma in questo caso non è stato necessario.

```
{
    "meetingId":"abcde123456",
    "hostEmail":"admin@tiscali.it",
    "items":[
        {
            "email":"test@gmail.com"
        },
        {
            "email":"popi@gmail.com"
        }
    ]
}
```

- **Righe 6-15**: viene chiamata l'API.

La modifica del meeting nel database prevede l'aggiornamento delle date di inizio e fine, poiché gli invitati non vengono salvati nel database ma vengono recuperati direttamente da Webex ogni volta.

```
1     public static void modifyDatabaseMeet(MeetJPA repo, JsonNode evento) {
2       try {
3          Meet meet = repo.findMeetById(evento.get("id").asLong());
4
5          Timestamp dataInzio =
6                Timestamp.valueOf(UtiltyMeet.transformUTCIntoRomeTimezone(evento.get("start").asText()));
7          Timestamp dataFine =
8                Timestamp.valueOf(UtiltyMeet.transformUTCIntoRomeTimezone(evento.get("end").asText()));
9
10         meet.setDataInizio(dataInzio);
11         meet.setDataFine(dataFine);
12
13         repo.save(meet);
14
15       } catch (Exception e) {
16          e.printStackTrace();
17          throw new RuntimeException("Errore durante la modifica del meeting a database", e);
18       }
19    }
```

- **Riga 3**: ricerca dell'oggetto `Meet` nel database utilizzando il suo ID.
- **Righe 5-8**: calcolo dei `Timestamp` nel fuso orario di Roma.
- **Righe 10-11**: aggiornamento delle date di inizio e fine dell'oggetto `Meet`.
- **Riga 13**: salvataggio della modifica nel database.

4.9 Eliminazione dei meeting

4.9.1 Frontend

Per eliminare un meeting, è necessario aprire il relativo `Dialog` per visualizzarne i dettagli. A questo punto, è presente il pulsante *elimina*. L'immagine seguente mostra il pulsante quando vi si passa sopra con il mouse.

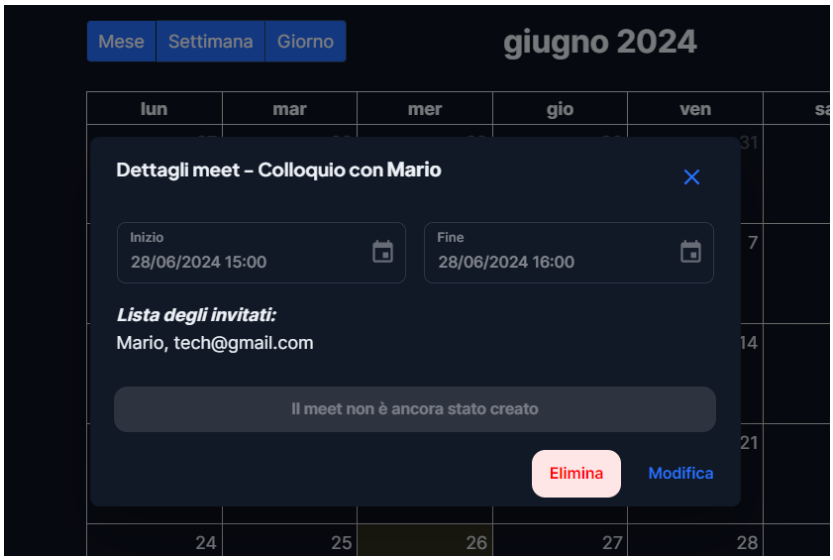


Figura 4.22: Eliminazion meet - hover

Se si clicca sul pulsante, viene visualizzato un popup di conferma:

Figura 4.23: Eliminazion meet - popup di conferma

Proseguendo con l'eliminazione, viene chiamata la funzione `handleDelete`

```
1  const handleDelete = async () => {
2      setOpen(false);
3      setOpenDeleteMeetDialog(false);
4
5      const payload = {
6          id: eventDetails.id,
7          webexId: eventDetails.extendedProps.webexId,
8      };
9
10     try {
11         const response = await deleteMeeting(payload);
12
13         if (response.status == 200) {
14           const updatedEvents = events.filter((event) => event.id != eventDetails.id);
15           setEvents(updatedEvents);
16
17           showPopup('Eliminazione avvenuta con successo', true);
18         } else {
19           showPopup("Errore nell'eliminazione del meeting", false);
20         }
21     } catch (error) {
22         console.error("Errore durante l'eliminazione di un evento:", error);
23         showPopup("Errore nell'eliminazione del meeting", false);
24     }
25 };
```

- **Righe 6-7**: viene creato il payload da inviare al backend, contenente i due ID necessari per l'eliminazione del meeting: l'ID del database e l'ID di Webex.
- **Righe 14-15**: se l'eliminazione è andata a buon fine, l'array degli eventi associati a `Fullcalendar` viene aggiornato manualmente. Questo avviene creando un nuovo array tramite la funzione `filter`, che rimuove l'evento con lo stesso ID di quello eliminato. In questo modo, il cambiamento viene immediatamente visualizzato nel calendario senza bisogno di ricaricare la pagina.

Se il meet viene eliminato con successo, viene mostrato un popup di avviso:

Figura 4.24: Eliminazion meet - popup di successo

4.9.2 Backend

Controller

```
1  @DeleteMapping(value = "/deleteMeet")
2      public ResponseEntity<?> deleteMeeting(@RequestBody ObjectNode obj) {
3        try {
4           Mono<Boolean> resultMono = serviceMeet.deleteMeet(obj);
5           Boolean result = resultMono.toFuture().get();
6
7           if (result) {
8              return new ResponseEntity<>(result, HttpStatus.OK);
9           } else {
10             return new ResponseEntity<>(result, HttpStatus.INTERNAL_SERVER_ERROR);
11          }
12
13       } catch (Exception e) {
14          e.printStackTrace();
15          return new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
16       }
17 }
```

- **Riga 1**: l'annotazione indica che il metodo risponde a richieste HTTP DELETE sull'URL /deleteMeet.
- **Righe 4-5**: viene chiamato il service che restituisce un oggetto `Mono<Boolean>`. L'esecuzione viene sospesa fino a quando il risultato non viene ottenuto tramite il metodo `toFuture().get()`.
- **Righe 7-15**: se l'operazione di eliminazione è riuscita, viene restituita una risposta con stato HTTP 200 (OK). In caso contrario, viene restituita una risposta con stato HTTP 500 (Internal Server Error).

Service

```
1  public Mono<Boolean> deleteMeet(ObjectNode obj) {
2      try {
3          Long idDatabase = obj.get("id").asLong();
4          String webexId = obj.get("webexId").asText();
5
6          return DeleteMeeting.deleteWebexMeet(webexAPI, webexId).flatMap(success -> {
7              if (success) {
8                  DeleteMeeting.deleteDatabaseMeet(repo, idDatabase);
9                  return Mono.just(true);
10             } else {
11                 return Mono.just(false);
12             }
13         }).doOnError(error -> {
14             System.err.println("Errore durante la modifica del meeting su Webex: " +
                   error.getMessage());
15         }).onErrorReturn(false);
16
17     } catch (Exception e) {
18         e.printStackTrace();
19         return Mono.just(false);
20     }
21 }
```

- **Righe 3-4**: vengono recupeati i campi `id` e `webexId` associati al meet passati dal frontend.

- **Righe 6-8**: viene chiamato il metodo per eliminare il meet su Webex. Se l'operazione va a buon fine, viene chiamato il metodo per eliminare il record del meeting a database.

Repository

Metodo per eliminare il meet su Webex: viene effettuata una richiesta DELETE all'API `https://webexapis.com/v1/meetings/webexId`. L'unico parametro necessario è il `WebexId`, insieme al token per autenticare l'API.

```
1  public static Mono<Boolean> deleteWebexMeet(WebClient webexAPI, String webexId) {
2    String uri = MeetingAPI.DELETE_WEBEX_MEET + webexId;
3
4    return webexAPI.delete()
5        .uri(uri)
6        .header(HttpHeaders.AUTHORIZATION, "Bearer " + MeetingAPI.TOKEN)
7        .exchangeToMono(response -> {
8            if (response.statusCode().is2xxSuccessful()) {
9                return Mono.just(true);
10           } else {
11               return response.createException().flatMap(Mono::error);
12           }
13       })
14       .onErrorResume(e -> Mono.just(false));
15 }
```

Metodo per eliminare il record del meet a database: viene recuperata l'istanza del `Meet` grazie al suo ID utilizzando il metodo `findMeetById`. Successivamente viene rimosso dal database.

```
1  public static void deleteDatabaseMeet(MeetJPA repo, Long id) {
2      try {
3          Meet meetDaEliminare = repo.findMeetById(id);
4          repo.delete(meetDaEliminare);
5      } catch (Exception e) {
6          e.printStackTrace();
7          throw new RuntimeException(
8              "Errore durante l'eliminazione del meeting al database", e
9          );
10     }
11 }
```

Capitolo 5

Idee future

5.1 Autenticazione delle chiamate API

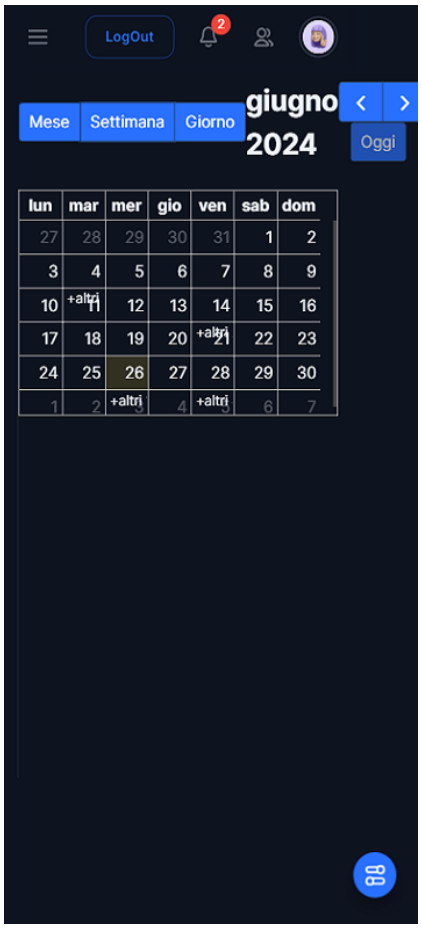
Al momento, nel progetto non è ancora stato implementato un metodo per autenticare le chiamate alle API definite. Questo passaggio è fondamentale per evitare che malintenzionati possano accedere, modificare o eliminare dati sensibili. Inoltre, l'autenticazione delle chiamate API è necessaria per conformarsi alle normative sulla protezione dei dati, come il GDPR, garantendo che le informazioni personali siano trattate con la massima riservatezza e sicurezza.

5.2 Integrazione con Google Calendar

Se l'utente acconsente a collegare il proprio account Google a RisUma, sarebbe opportuno sincronizzare automaticamente il calendario di RisUma con quello di Google ogni volta che viene fissato un colloquio. In questo modo, la gestione degli appuntamenti risulterebbe ancora più semplice ed efficiente. L'integrazione con Google Calendar permetterebbe agli utenti di visualizzare e gestire i propri appuntamenti direttamente dal loro calendario personale, senza dover accedere separatamente a RisUma. Ciò migliorerebbe significativamente l'esperienza utente, riducendo la probabilità di sovrapposizioni di appuntamenti e dimenticanze. Inoltre, l'integrazione potrebbe includere funzionalità avanzate come notifiche automatiche, promemoria via email o SMS e la gestione delle disponibilità, rendendo il processo di pianificazione dei colloqui più fluido e integrato con gli strumenti già utilizzati dagli utenti nella loro vita quotidiana.

5.3 Miglioramento della visualizzazione mobile

È necessario ottimizzare la pagina del sito per la visualizzazione su smartphone:



L'adattamento della pagina del sito alla visualizzazione mobile è fondamentale per garantire un'esperienza utente ottimale su tutti i dispositivi. In particolare, la visualizzazione del calendario richiede una riprogettazione completa della barra di controllo superiore, magari introducendo una dropdown list per la scelta della visualizzazione e ridisponendo completamente le scritte e i pulsanti per muoversi nel calendario. Sarebbe utile implementare la funzionalità di swipe per passare da un mese all'altro senza dover necessariamente cliccare sui bottoni. Anche la visualizzazione del calendario deve essere ripensata per utilizzare meglio lo spazio verticale, estendendolo sia in altezza che in larghezza.

I due `Dialog` per i dettagli e la modifica di un evento si adattano già abbastanza bene a una visualizzazione da smartphone, ma si potrebbe sfruttare meglio lo spazio verticale aumentando lo spazio tra i vari componenti per rendere l'interfaccia ancora più pulita e godibile.

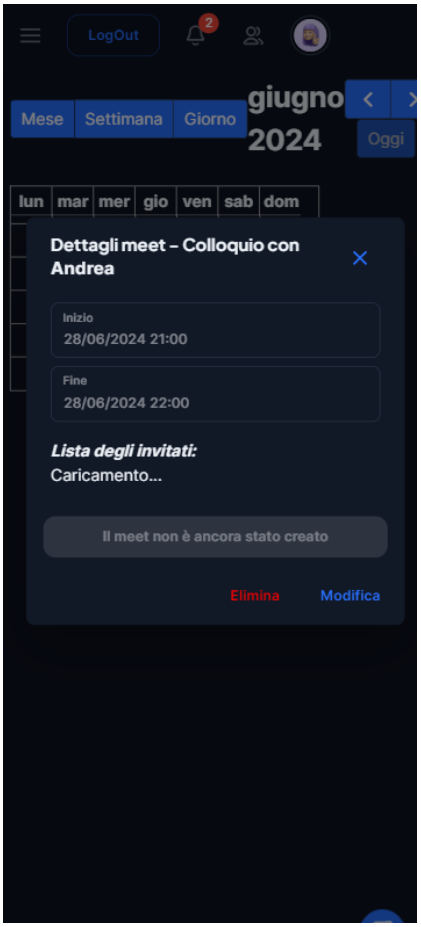


Figura 5.1: Dettagli meet - mobile

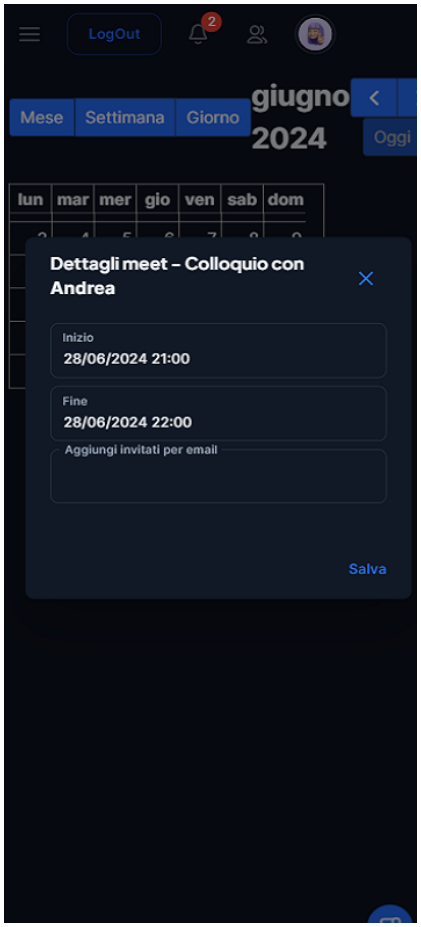


Figura 5.2: Modifica meet - mobile

Il `DateTimePicker` cambia automaticamente il tipo di visualizzazione in base al dispositivo utilizzato, il che elimina la necessità di ulteriori modifiche per questa specifica funzionalità.

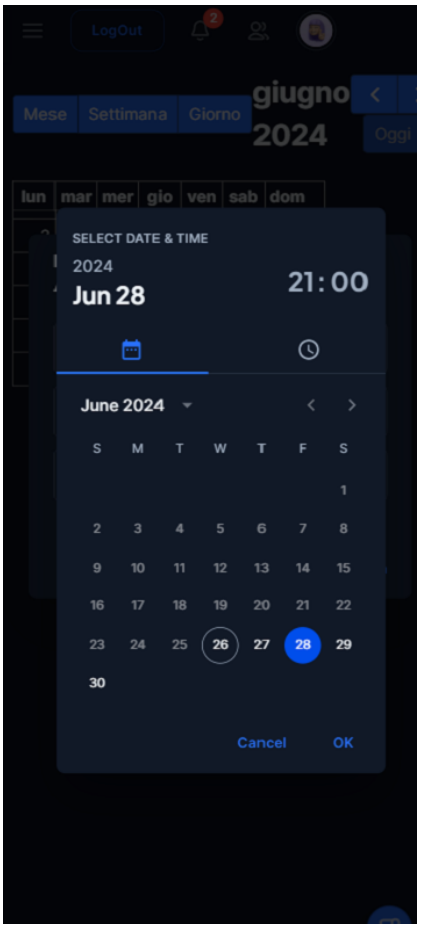


Figura 5.3: Seleziona data - mobile

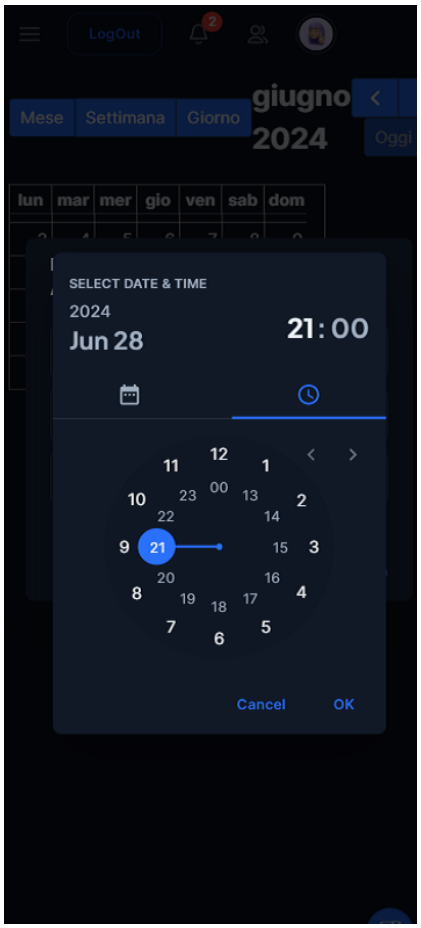


Figura 5.4: Seleziona ora - mobile

Capitolo 6

Conclusioni

Bibliografia

[1] Wikipedia. «React (web framework).» (), indirizzo: https://it.wikipedia.org/wiki/React_(web_framework). (Data di accesso: 25.06.2024).

[2] Wikipedia. «TypeScript.» (), indirizzo: https://it.wikipedia.org/wiki/TypeScript. (Data di accesso: 25.06.2024).

[3] Geekandjob. «Axios.» (), indirizzo: https://www.geekandjob.com/wiki/axios. (Data di accesso: 25.06.2024).

[4] Wikipedia. «Java (linguaggio di programmazione).» (), indirizzo: https://it.wikipedia.org/wiki/Java_(linguaggio_di_programmazione). (Data di accesso: 24.06.2024).

[5] Microsoft. «Che cos'è Spring Boot di Java?» (), indirizzo: https://azure.microsoft.com/it-it/resources/cloud-computing-dictionary/what-is-java-spring-boot. (Data di accesso: 25.06.2024).

[6] A. Gatu. «Cos'è Maven, a cosa serve e come si usa.» (2024), indirizzo: https://www.nextre.it/cose-maven-si-usa/. (Data di accesso: 25.06.2024).

[7] V. Racca. «Introduzione a JPA e mapping delle relazioni.» (2020), indirizzo: https://www.vincenzoracca.com/blog/framework/jpa/jpa-reletions/. (Data di accesso: 25.06.2024).

[8] Geekandjob. «Cos'è PostgreSQL.» (), indirizzo: https://www.geekandjob.com/wiki/postgresql. (Data di accesso: 25.06.2024).

[9] React. «useEffect.» (), indirizzo: https://react.dev/reference/react/useEffect. (Data di accesso: 03.07.2024).

[10] React. «useState.» (), indirizzo: https://react.dev/reference/react/useState. (Data di accesso: 03.07.2024).

[11] Webex. «Developer Sandbox.» (), indirizzo: https://developer.webex.com/docs/developer-sandbox-guide. (Data di accesso: 27.06.2024).

[12] A. Arshaw. «FullCalendar.» (), indirizzo: https://fullcalendar.io/. (Data di accesso: 27.06.2024).

[13] Okta. «JSON Web Tokens.» (), indirizzo: https://auth0.com/docs/secure/tokens/json-web-tokens. (Data di accesso: 03.07.2024).

[14] React. «Conditional Rendering.» (), indirizzo: https://react.dev/learn/conditional-rendering. (Data di accesso: 03.07.2024).

[15] Baeldung. «Guide to Spring @Autowired.» (2024), indirizzo: https://www.baeldung.com/spring-autowire. (Data di accesso: 03.07.2024).

[16] Pankaj. ≪Spring @Repository Annotation.≫ (2022), indirizzo: https://www.digitalocean.com/community/tutorials/spring-repository-annotation. (Data di accesso: 03.07.2024).

[17] Okta. ≪Data Transfer Object DTO Definition and Usage.≫ (2023), indirizzo: https://www.okta.com/identity-101/dto/. (Data di accesso: 03.07.2024).

[18] S. Lesinigo, *Appunti del corso di Interazione Uomo-Macchina*, Università degli Studi di Milano-Bicocca, Docente: Cabitza Federico Antonio Niccolò Amedeo, Anno Accademico 2023-2024.

Appendice A

Euristiche di Nielsen

Delineate da Jakob Nielsen in collaborazione con Rolf Morich nel 1994, sono un insieme di principi generali ampaimente utilizzati ancora oggi che supportano la progettazione di sistemi interattivi usabili [18]:

1. Visibilità dello stato del sistema
2. Corrispondenza tra sistema e mondo reale
3. Controllo e libertà dell'utente
4. Consistenza e standard
5. Riconoscimento piuttosto che ricordo
6. Flessibilità ed efficienza d'uso
7. Estetica e minimalismo
8. Prevenzione degli errori
9. Aiuto nel rilevare e correggere gli errori
10. Aiuto e documentazione