UNIVERSITÀ DI TRENTO

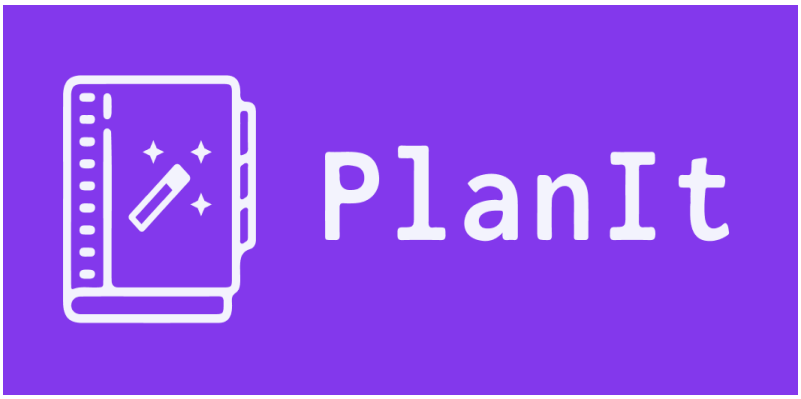
Dipartimento d'Ingegneria e
Scienze dell'informazione

Progetto:

PlanIt

Titolo del documento:

Sviluppo Applicazione



Gruppo T56
Gabriele Lacchin, Denis Lucietto ed Emanuele Zini

Indice

Scopo del documento

da sistemare, copia e incollato

Il presente documento riporta tutte le informazioni necessarie per lo sviluppo di una parte dell'applicazione PlanIt. In particolare, presenta tutti gli artefatti necessari per realizzare i servizi di creazione e modifica di eventi, calendari e utente autenticato nel sito. Partendo dalla descrizione degli user flows legati, soprattutto, al ruolo dell'utente autenticato dell'applicazione, il documento prosegue con la presentazione delle API necessarie (tramite l'API Model e il Modello delle risorse) per poter visualizzare, creare e modificare sia gli eventi che i calendari necessari per implementare l'applicativo PlanIt, che ricordiamo essere un applicativo di gestione calendari. Per ogni API realizzata, oltre ad una descrizione delle funzionalità fornite, il documento presenta la sua documentazione e i test effettuati. Per i test effettuatati, verrà anche fornito un documento html con il resoconto di questi. Infine una sezione e' dedicata alle informazioni del Git Repository e il deployment dell'applicazione stessa.
Di seguito sono presenti gli argomenti che verranno trattati in questo documento.

- User Flows;
- Application Implementation and Documentation;
- API Documentation;
- FrontEnd Implementation;
- GitHub Repository and DeploymentInfo;
- Testing;

1 User Flows

In questa sezione del documento di sviluppo riportiamo gli “user flows” riguardanti, soprattutto, il ruolo dell'utente autenticato nel nostro sito. Questa figura descrive gli "user flows" relativi alle funzionalità che abbiamo reso disponibili in questo prototipo dell'applicativo PlanIt. Come si può notare dall'"user flow", l'utente, una volta autenticato, visualizza la schermata "Calendario", schermata principale della nostra piattaforma. La procedura di autenticazione da parte dell'utente può contenere varie varianti; infatti l'utente ha la possibilità sia di accedere/registrarsi mediante proprie credenziali sia mediante un proprio account Google. Nel caso in cui si fosse registrato mediante credenziali e avesse dimenticato la propria password, l'utente ha la possibilità di fare il "reset password" per recuperare il proprio account indicando a quale indirizzo email inviare l'email di recupero.
Una volta effettuato l'accesso, l'utente, divenuto utente autenticato, visualizza la schermata "Calendario" da cui:

- può accedere alla seconda schermata implementata, ovvero "Eventi";
- può fare diverse funzionalità direttamente da questa schermata, come: visualizzare periodi temporali diversi nel calendario, filtrare i calendari che si visualizzano, creare eventi singoli e calendari direttamente da questa schermata, anziché andando nella schermata "Eventi", dove, però, c'è anche la possibilità di modificare ed eliminare quest'ultimi e creare eventi ripetuti, ovvero eventi che, già in tempo di compilazione di questi, sono definiti su più giorni.

Si specifica che per "compilazione evento", azione presente sia quando si modifica che crea un evento, si intende la definizione di vari campi, ovvero:

- titolo, campo da completare obbligatoriamente per far andare a buon fine la procedura di creazione o modifica evento;
- data, campo da completare obbligatoriamente per far andare a buon fine la procedura di creazione o modifica evento;
- priorità, campo facoltativo;
- difficoltà, campo facoltativo;
- calendario, ovvero calendario a cui appartiene l'evento che si sta modificando o creando. Questo è un campo da completare obbligatoriamente per far andare a buon fine la procedura di creazione o modifica evento;
- notifiche, ovvero quando ricevere la notifica di tale evento.

Infine per "compilazione calendario", azione presente sia nella "creazione" che "modifica" calendario, sin intende la definizione di vari campi, ovvero:

- persone, ovvero quali persone partecipano a tale evento.
- nome del calendario, campo che deve essere definito per forza per l'esito positivo della procedura di creazione o modifica;
- colore, campo che deve essere definito per forza per l'esito positivo della procedura di creazione o modifica; questo non è altro che il colore con cui saranno mostrati gli eventi appartenenti a tale calendario;
- fuso orario, campo facoltativo;
- impostazioni predefinite degli eventi, ovvero definizione di alcuni campi con cui si vanno a precompilare gli eventi appartenenti a quel calendario. Ovviamente, nella creazione o modifica di un evento, questi campi precompilati potranno essere modificati. Si specifica che nelle impostazioni predefinite degli eventi, i seguenti campi sono compresi: durata, priorità, difficoltà, luogo e notifiche.

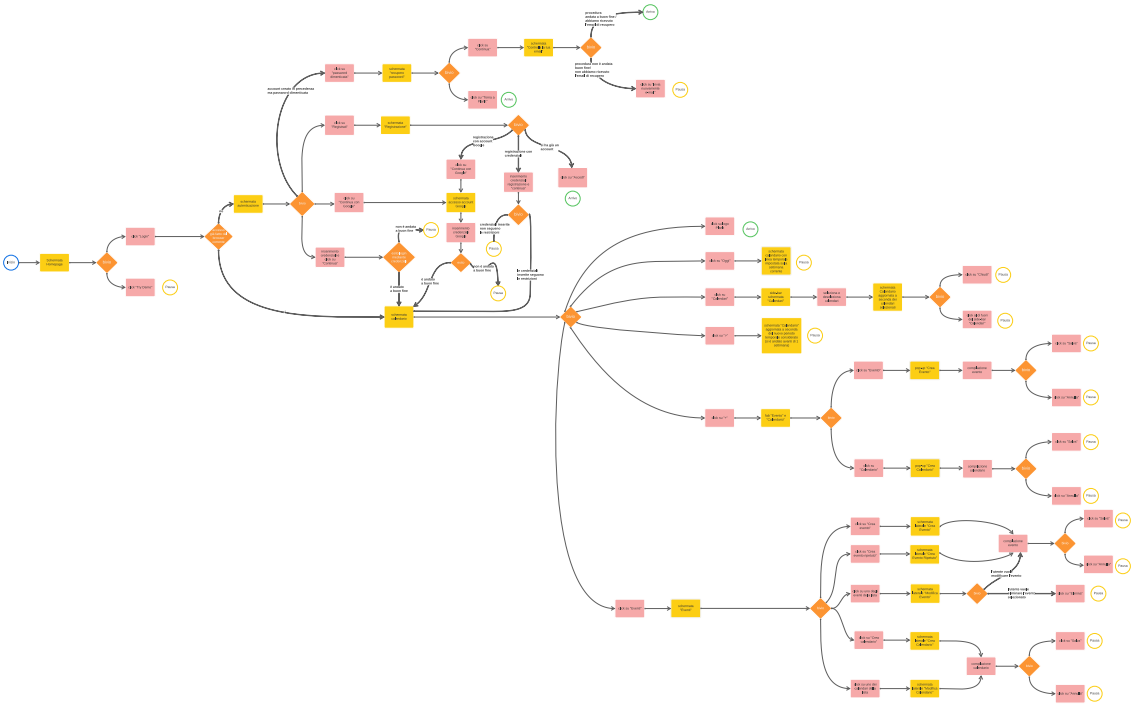
Come già detto precedentemente, dalla schermata "Calendario", si può passare alla schermata "Eventi", dove è possibile:

- creare eventi singoli, allo stesso modo che si può fare dalla schermata "Calendario";
- creare eventi ripetuti;

- modificare sia eventi singoli che ripetuti;
- eliminare sia eventi singoli che ripetuti;
- creare calendari;
- eliminare calendari.

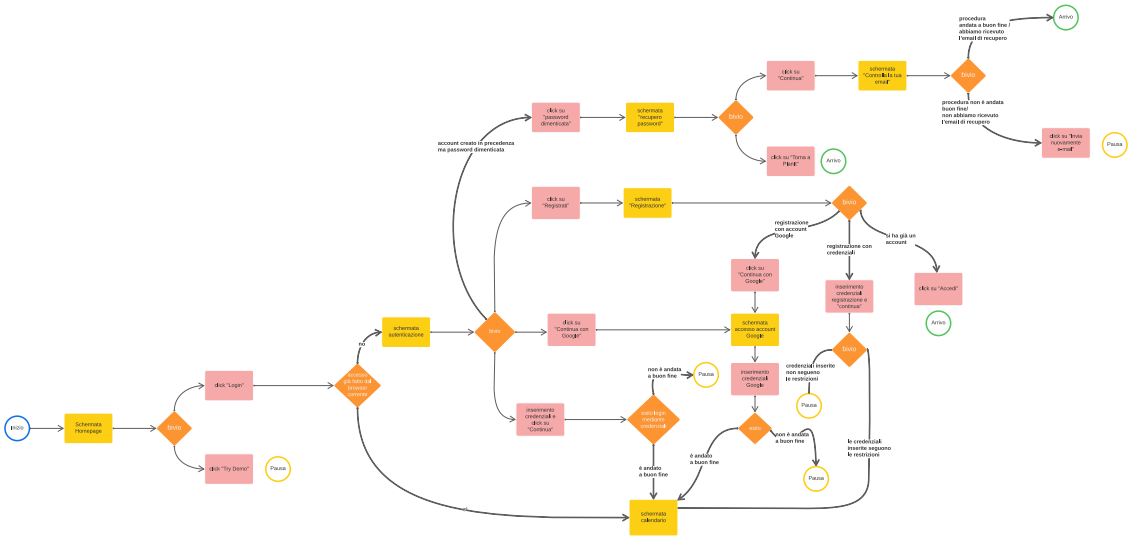
Nella prima foto sottostante è presente l'interno "user flow", ma per comodità, visto che è molto grande, abbiamo deciso d' inserire anche due "user flow" che non sono altro che il primo totale, ma diviso in due: uno che mostra l'"user flow" riguardo a tutto il processo di autenticazione, l'altro "user flow" riguarda a ciò che può fare l'utente una volta autenticato ed entrato effettivamente nel sito. In questo modo questi due "macro" azioni che sono percorribili dall'utente sono più facilmente visualizzabili. Ad ogni modo, come per tutte le foto, come nota a piè di pagina è presente il link alla repo GitHub, dove è possibile scaricare ciascuna immagine per poterla vedere meglio.
Infine, inseriamo anche la legenda con cui sono stati fatti questi "user flows".

Legenda

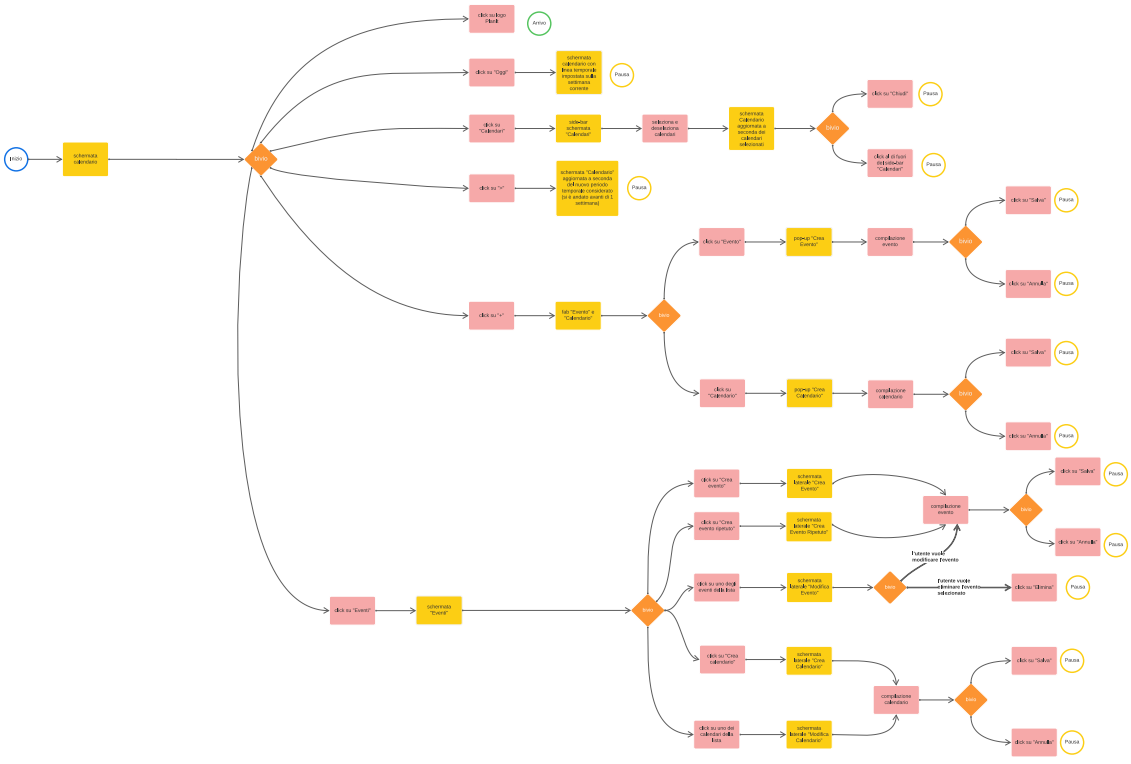


UserFlow completo

Immagine PNG/SVG Legenda
Immagine PNG/SVG User Flow completo



UserFlow autenticazione



UserFlow all'interno del sito dopo l'autenticazione

Immagine PNG/SVG User Flow - autenticazione
Immagine PNG/SVG User Flow all'interno del sito dopo l'autenticazione

2 Application Implementation and Documentation

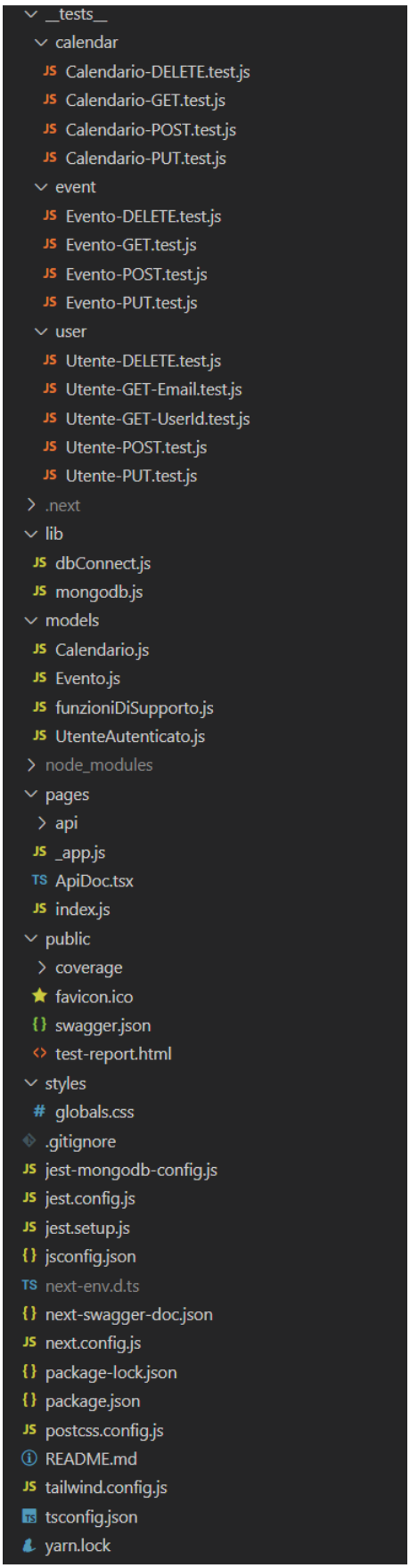
Nelle sezioni precedenti abbiamo individuato tutte le procedure e funzionalità che sono implementate nel prototipo del sito PlanIt e di come l'utente autenticato può utilizzarle nel suo flusso applicativo mostrando un diagramma di "user flows". L'applicazione è stata sviluppata utilizzando NextJS (anche TypeScript ma solo in piccola parte per la documentazione) per la parte di API e FrontEnd, invece per la parte di UI (user interface) sono stati utilizzati CSS e React. Per la gestione dei dati abbiamo utilizzato MongoDB e la libreria mongoose. Per la gestione di autenticazione all'interno del sito abbiamo utilizzato il sistema esterno Auth0. Per la produzione della documentazione delle APIs abbiamo usato la libreria Swagger per NextJS, invece per il testing, sempre delle APIs, è stato usato il framework di JavaScript Jest. Infine, per fare le chiamate delle APIs da FrontEnd usiamo la libreria di JavaScript Axios.

APD1 : **Project Structure**

La struttura del progetto è presentata nelle seguenti foto ed è composto di una cartella "api", dove sono presenti tutte le funzioni per la gestione delle APIs locali, divisi in base a quale resource gestiscono (ovvero "event", "calendar", "user"), di una cartella "pages", al cui interno oltre che la cartella api, sono presenti anche i file di estensione React indispensabili per comporre le interfacce grafiche di ciascuna schermata presente nel sito PlanIt, di una cartella "lib", in cui sono presenti i file usati per andare a stabilire la connessione con il database MongoDB, e, infine, di una cartella "models", dove ci sono i modelli di oggetti implementati per la gestione di dati del nostro sito. Si evidenzia che i modelli definiti sono presenti nel database MongoDB.

Sottolineiamo che le uniche "pages" che sono state sviluppate in maniera completa sono "calendario" e "eventi". Evidenziamo che le varie componenti che vanno a comporre queste schermate sono state sviluppate in React nella cartella "components", che può essere sempre visualizzata nella seguente foto.

Inoltre, nella cartella "styles" sono presenti i file "css" sviluppati per andare a definire i stili dei moduli presenti nell'interfaccia grafica che saranno visualizzabili dall'utente che entra nel sito PlanIt.



Struttura codice sorgente

Immagine PNG Struttura codice sorgente

APD2 : **Project Dependencies**

I seguenti moduli sono stati utilizzati e aggiunti al file "package.json", il cui contenuto è anche visualizzabile nella seguente foto:

- MongoDB, usato per fare il testing;
- Mongoose, libreria usata per connettersi al database MongoDB.
- NextJS, framework utilizzato per lo sviluppo delle APIs e del FrontEnd del sito PlanIt;
- Swagger, tool utilizzato per la documentazione delle APIs;
- React, libreria di JavaScript per l'implementazione dell'UI del sito;
- Jest, framework di JavaScript utilizzato per andare a fare il testing delle APIs;
- PostCSS, è uno strumento di sviluppo software che utilizza plug-in basati su JavaScript per automatizzare le operazioni CSS di routine e quindi lo sviluppo dei moduli presenti nel FrontEnd del nostro sito;
- TypeScript, estensione di JavaScript, usata solo in minima parte nello sviluppo delle APIs, ovvero nello sviluppo degli esempi per la documentazione di queste.

Project Dependencies

Immagine PNG Project Dependencies

APD3 : Project Data or DB

Per la gestione dei dati utili all'applicazione abbiamo definito tre principali strutture dati come illustrato nelle seguenti immagini. Infatti, le strutture dati che abbiamo deciso di avere in questo prototipo del nostro sito PlanIt, sono:

- "UtenteAutenticato", struttura dati che individua l'utente che si è autenticato ed accede al sito;
- "Evento", componente fondamentale del calendario, che sta alla base della logica del nostro sito ideato.
- "Calendario", contenitore di eventi, struttura dati sopra citata.

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
Calendario	1	371B	371B	36KB	1	36KB	36KB
Evento	2	859B	430B	36KB	1	36KB	36KB
UtenteAutenticato	3	447B	149B	36KB	1	36KB	36KB

Collections delle strutture dati usate nell'applicazione

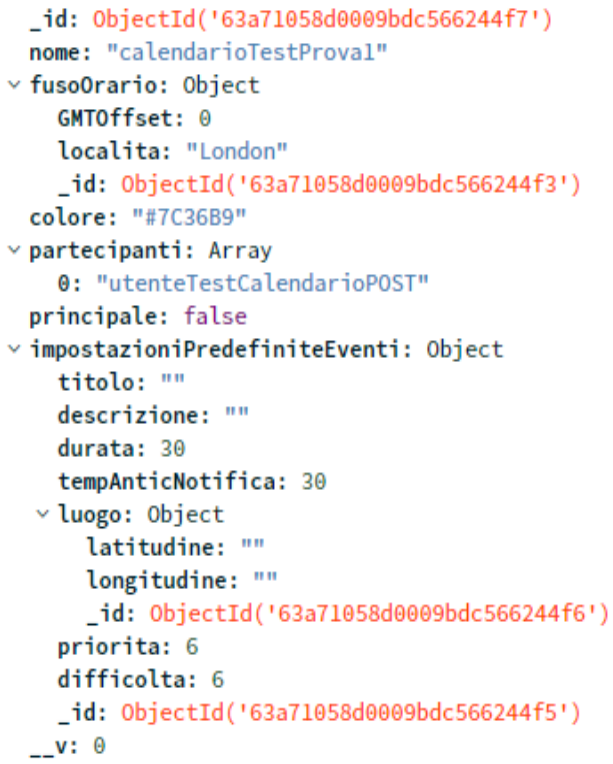
```
_id: ObjectId('63a709dadad816515f857b37')
userId: "User1"
email: "prova@example.com"
username: "Username1"
__v: 0
```

Tipo di dato "Utente Autenticato"

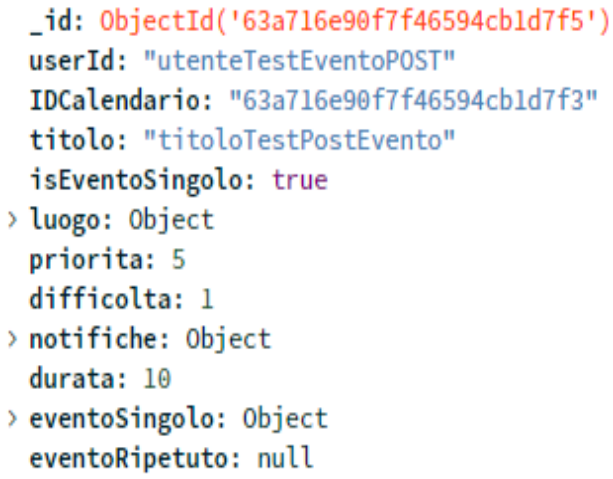
```
_id: ObjectId('63a71058d0009bdc566244f7')
nome: "calendarioTestProva1"
› fusoOrario: Object
colore: "#7C36B9"
› partecipanti: Array
principale: false
› impostazioniPredefiniteEventi: Object
__v: 0
```

Tipo di dato "Calendario"

Immagine PNG Collections
Immagine PNG Tipo di dato "Utente Autenticato"
Immagine PNG Tipo di dato "Calendario"



Tipo di dato "Calendario" con gli oggetti contenuti "esplorati"



Tipo di dato "Evento", in questo caso è un evento singolo, che quindi è presente solo in un giorno specifico

Immagine PNG Tipo di dato "Calendario" con gli oggetti contenuti "esplorati"
Immagine PNG Tipo di dato "Evento" singolo

Tipo di dato "Evento" singolo con gli oggetti contenuti "esplorati"

Tipo di dato "Evento", in questo caso è un evento ripetuto, che quindi è presente su più giorni

Immagine PNG Tipo di dato "Evento" singolo con gli oggetti contenuti "esplorati"
Immagine PNG Tipo di dato "Evento" ripetuto

Tipo di dato "Evento" ripetuto con gli oggetti contenuti "esplorati"

Immagine PNG Tipo di dato "Evento" ripetuto con gli oggetti contenuti "esplorati"

APD4 : Project APIs

In questo capitolo, verranno presentate le APIs che sono state sviluppate in questo prototipo del sito PlanIt. Infatti, si passerà dall'esposizione dell' Extract Diagram, diagramma che serve come collegamento tra il Class Diagram alla APIs, e Resources Models, diagramma in cui sono presenti tutte le risorse implementate, alla descrizione del vero e proprio codice delle APIs presentati in questi primi diagrammi.

4.1 : Resources Extraction from the Class Diagram

Questo diagramma si può dire che faccia da vero e proprio "ponte" tra Class Diagram, presentato nel documento D3, e le resources che andremo ad implementare nelle nostre APIs.
Come si può notare, abbiamo individuato dodici resources che riteniamo fondamentali per sviluppare almeno la logica che sta alla base del nostro sito PlanIt.

- Partendo dalla classe "UtenteAutenticato", presentata in D3 DCL1, abbiamo individuato tre resources, ovvero:
  - CreaAccount, di tipo POST. Risorsa che ha lo scopo di creare un account all'interno del database MongoDB e per questo motivo ha l'etichetta "BackEnd"; infatti l'effetto principale di questa API avviene nel "BackEnd" con la creazione dell'account nel database.
    L'account è una struttura dati già presenta nel capitolo precedente, ovvero "UtenteAutenticato", e l'input delineato è un "body" in quanto è formato da più di tre parametri, i quali verrano specificati e descritti nel diagramma Resources Model, presentato nel successivo capitolo.
  - ModificaAccount, di tipo PUT. Risorsa che ha al fine di dare la possibilità di andare a modificare un account già presente nel nostro database. L'etichetta "BackEnd" e l'input body sono preseti per gli stessi motivi definiti per la resource "CreaAccount".
  - GetDatiAccount, di tipo GET. API che ha lo scopo di ottenere i dati di un "UtenteAutenticato" conoscendo l'userId di tale utente. L'effetto ultima di questa resource è nel FrontEnd, in quanto i dati appariranno all'utente e quindi si è deciso di mettere come sua etichetta "FrontEnd"
  - GetDatiAccount_email, di tipo GET. Risorsa analoga a quella precedente, con la sola differenza che l'account, da cui estrarre i dati, non è definito mediante l' "userId", "id" e stringa che identifica univocamente un utente, ma mediante la sua email che ha utilizzato per registrarsi.

  Si specifica che gli attributi "link", "password" e "schema" sono necessari per poter effettuare la connessione con il database MongoDB, come specificato in D3 DCL13, e tutte le resources individuate riguardano a funzioni già presentate sempre in D3 DCL13.
- Dopo abbiamo individuato altre tre resources, partendo dalla classe "Evento", già presentata in D3 DCL6.2, ovvero:
  - creaEvento, di tipo POST. Resource che ha lo scopo di andare a creare un evento che viene salvato nel database MongoDB. L'input che viene preso da questa API è un insieme di parametri che vanno a formare un oggetto di tipo "Evento", struttura dati già definita nel capitolo precedente, che in quanto piuttosto complicata, non siamo andati a specificare in questo diagramma ma nel "Resources Model" e qua ci siamo fermati nel scrivere come input solo "body". In quanto ha un effetto sul "BackEnd", con il salvataggio dell'evento nel database, abbiamo indicato come etichetta "BackEnd".
  - modificaEvento, di tipo PUT; grazie alla sua esecuzione abbiamo la modifica di un determinato evento già presente al database appartenente ad un utente. In quanto ha un effetto finale nel database, gli abbiamo attribuito l'etichetta "BackEnd".
  - eliminaEvento, di tipo DELETE. Lo scopo di questa resource è quello di dare la possibilità di eliminare un "Evento" dato il suo "IDEvento" e l'"userId" dell'utente autenticato che ha questo determinato "Evento" che vuole eliminare. In quanto ha un effetto finale nel database, gli abbiamo attribuito l'etichetta "BackEnd".

- getEventi, di tipo GET. Risorsa mediante cui si ottengo gli eventi di un calendario, dato l'"userId" dell'utente che ha tale calendario e l' "IDCalendario" del calendario da cui vogliamo estrarre gli eventi. L'etichetta "FrontEnd" è stata messa in quanto questa API è fondamentale per mostrare all'utente finale nell'user interface gli eventi di un determinato calendario.

Si specifica che tutte le resources individuate riguardano a funzioni già presentate in D3 DCL6.2.

- Infine abbiamo sviluppato tre resources per la classe "Calendario", tipo di dato già presentato nel precedente capitolo, le cui funzionalità e attributi sono stati descritti in D3 DCL6.1. Le tre resources sono:
  - creaCalendario, di tipo POST. API che ha lo scopo di creare un calendario all'interno del database MongoDB, e per questo motivo ha come etichetta "BackEnd". In quanto, per andare a creare un "Calendario" all'interno del database servono gli attributi che formano un oggetto di tipo "Calendario" e l' "userId" dell'utente che vuole creare tale calendario, come input abbiamo indicato body. L'input verrà descritto maggiormente nel Resources Model che viene presentato nel successivo capitolo.
  - modificaCalendario, di tipo PUT; grazie alla sua esecuzione abbiamo la modifica di un determinato calendario già presente al database appartenente ad un utente. In quanto ha un effetto finale nel database, gli abbiamo attribuito l'etichetta "BackEnd".
  - eliminaCalendario, di tipo DELETE. Lo scopo di questa resource è quello di dare la possibilità di eliminare un "Calendario" dato il suo "IDCalendario" e l'"userId" dell'utente autenticato che ha questo determinato "Calendario" che vuole eliminare. In quanto ha un effetto finale nel database, gli abbiamo attribuito l'etichetta "BackEnd".
  - getCalendari, di tipo GET. API che ha lo scopo di ottenere tutti i calendari di un determinato "UtenteAutenticato", dato il suo "userId". Questa risorsa è fondamentale per andare a mostrare i calendari appartenenti all'utente, che potrà, quindi, visualizzare.

Si specifica che tutte le resources individuate riguardano a funzioni già presentate in D3 DCL6.1.

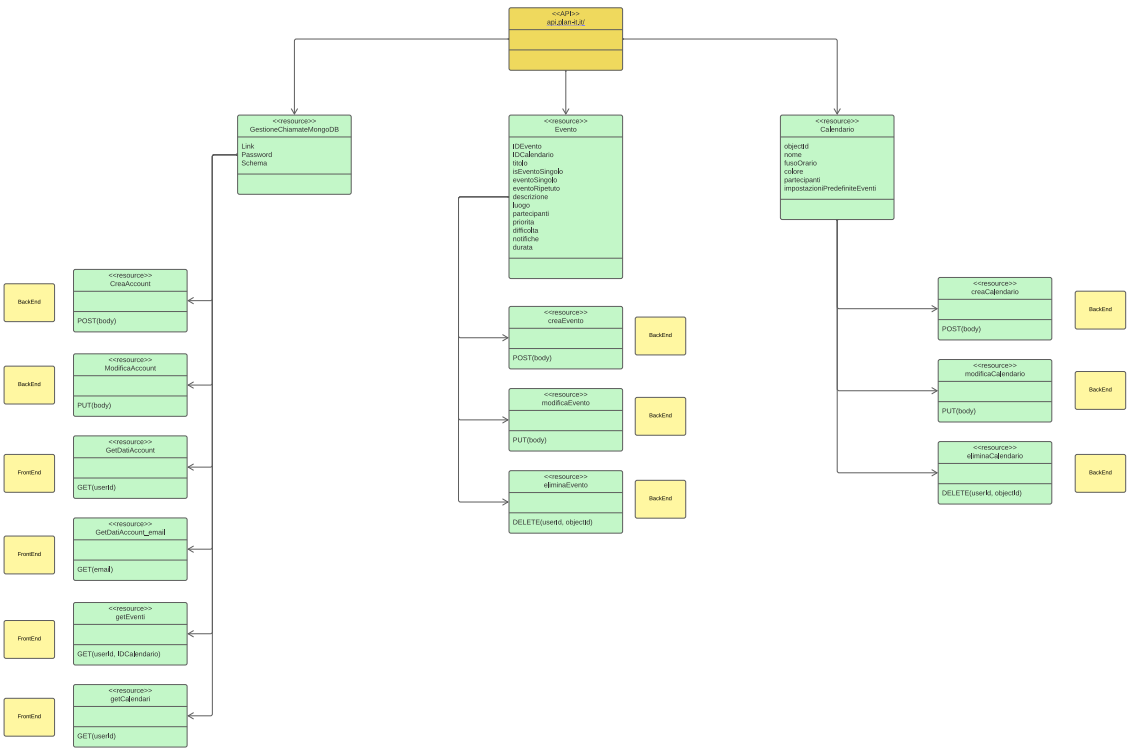


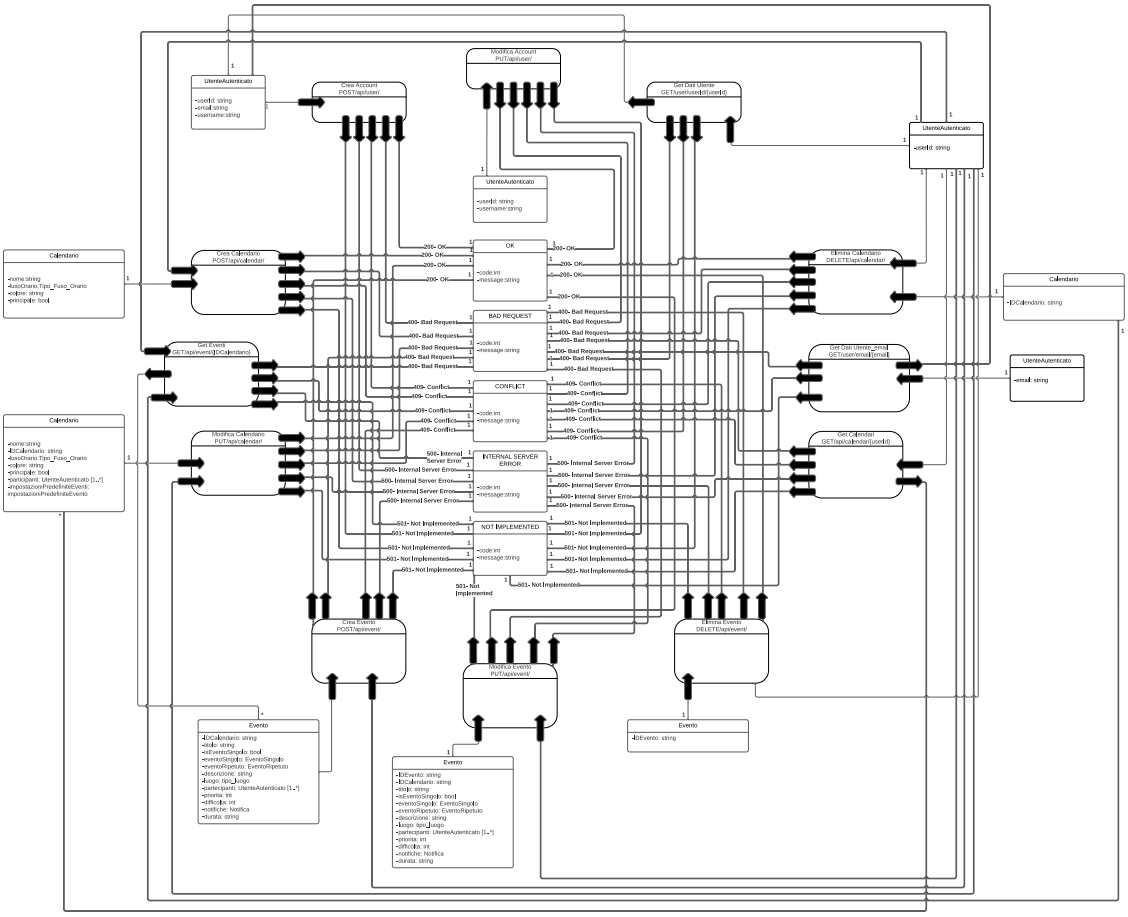
Immagine PNG/SVG Extract Diagram

4.2 : Resources Models

La seguente immagine mostra tutte le resources sviluppate nel nostro prototipo del sito PlanIt che sono state già presentate in parte nel precedente capitolo (APD4.1), ma in questo capitolo, e nelle seguenti figure, si va più nello specifico di come funzionano queste APIs, andando a specificare in maniera dettagliata gli input e gli output possibili di ciascuna risorsa.
Abbiamo deciso di non presentare solo il diagramma totale di tutti i resources models, ovvero quello successivo, ma anche altri tre diagrammi, con lo scopo di far visualizzare con più facilità tutte le risorse; infatti, si è andato a suddividere le resources presenti nel diagramma totale. Infatti in questi tre diagrammi sono presenti le resources, già presenti in quello totale, ma divise nei tre diversi diagrammi in base a come sono state già divise nell' Extract Diagram (APD4.1). C'è una cosa comune che si può notare prima di andare nel dettaglio della descrizione di ciascuna risorsa, ovvero che praticamente tutte le risorse danno le stesse possibile risposte/output, che sono:

- 200 OK. Serve ad indicare che la procedura effettuata dall'API è andata a buon fine e l'esecuzione è avvenuta con successo. Infatti "200 OK" è la risposta standard per le richieste HTTP andate a buon fine.
- 400 BAD REQUEST. Serve ad indicare che la richiesta non può essere soddisfatta a causa di errori di sintassi, quindi abbiamo ottenuto un errore dall'esecuzione dell' API. Per errori di sintassi si intendono errori che riguardano soprattutto l'input inserito, il quale ha un formato sbagliato e quindi la procedura che deve effettuare l'API non può andare a buon fine e si riceve un messaggio di errore accompagnato dal codice "400". Oppure, non si sono inseriti tutti i parametri necessari affinché l'API possa funzionare e quindi riceviamo sempre questo codice "400" con un messaggio di errore.
- 409 CONFLICT. Serve ad indicare che la richiesta non può essere portata a termine a causa di un conflitto con lo stato attuale della risorsa. Un caso che potrebbe portare ad un conflitto è ad esempio l'invio di un dato in input che è un duplicato di un dato già presente nel database, come un evento, calendario ed utente. Gli eventi, calendari e utenti duplicati inviati in input non possono essere creati e messi nel database, in quanto c'è il dato duplicato che crea conflitto. Questo errore lo otteniamo anche quando un determinato dato che stiamo passando non è presente nel database o non appartiene all'utente che lo sta inviando, questo può essere il caso in cui l' "userId" di un utente non esiste, oppure il "Calendario" o "Evento" passati con il loro "id" non appartengono a quell'utente identificato dall' "userId" inviato.
- 500 INTERNAL SERVER ERROR. Serve ad indicare un errore che riguarda il server interno. Questo errore non ha fatto andare a buon fine l'esecuzione dell' API, come ad esempio la sconnessione dal database MongoDB, dove sono inseriti i vari oggetti che vengono salvati.
- 501 NOT IMPLEMENTED. Questo errore indica che non si è stati in grado di soddisfare il metodo della richiesta. Quindi si può descrivere come un errore generico, che non specifica nessun dettaglio sul perché non sia andata a buon fine l'esecuzione dell' API.

Si specifica che da questo momento in poi, verranno scritti assieme i codici di risultato con i messaggi di risposta per pura comodità. Infatti invece che scrivere "Tale API invia "400" come codice di risposta e messaggio di errore "Wrong format for (something)", verrà scritto direttamente "Tale API invia come messaggio di errore "400 - Wrong format for (something)" ".



Resources Model completo

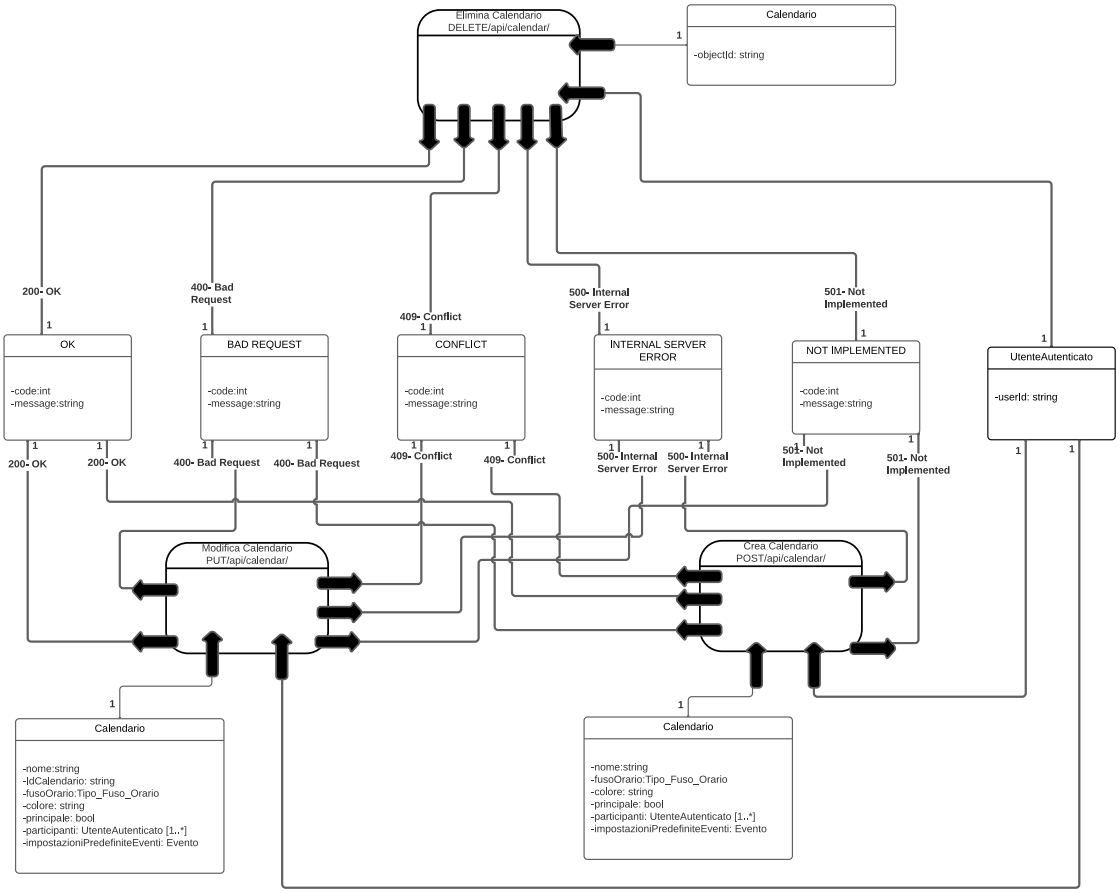
Immagine PNG/SVG Resources Model completo

4.2.1 : Resource Models "Calendario"

Nel seguente diagramma vengono presentate le APIs che riguardano la struttura dati "Calendario"; le risorse individuate per questa struttura dati, sono:

- "Elimina Calendario", resource di tipo DELETE. Questa API, già descritta in APD4.1, ha la funzione di eliminare un calendario specifico, dato il suo "IDCalendario" e l' "userId" dell'utente autenticato che vuole eliminare quel determinato calendario. Gli output sono quelli già descritti in precedenza, ma specifichiamo quali sono i casi in cui si ottiene l'errore "400". Otteniamo questo errore di risposta, quando manca un parametro (messaggio di errore "Parameter missing"), ovvero o "userId" o "IDCalendario". Invece, l'errore "409" si ottiene quando l' "userId" è un duplicato (messaggio di errore "There are too many users with that userId") o non esiste (messaggio di errore "There is no user with that userId") o l' "IDCalendario", del calendario che si vuole eliminare, non appartiene alla lista di calendari di quell'utente specificato con l' "userId" (messaggio di errore "You do not own the calendar").
- "Modifica Calendario", resource di tipo PUT. Questa API, già descritta in parte in APD4.1, ha la funzione di modificare un determinato calendario, dati tutti gli attributi che definiscono questa struttura dati modificata, e l' "userId" dell'utente che vuole modificare tale calendario. Specifichiamo, che si ottiene l'errore "400" ogniqualvolta non si è inserito uno degli attributi della struttura dati "Calendario", che si possono osservare nell'oggetto "Calendario" in input a questa API, con il messaggio "Parameter missing".
  Infine, questa resource ha anche gli errori che verranno specificati nella descrizione di "Crea Calendario", descritta qua sotto.
- "Crea Calendario", resource di tipo POST. Questa API, già descritta in parte in APD4.1, ha la funzione di creare e salvare nel database MongoDB il calendario formato dai parametri inviati in input per l'utente specificato con il parametro "userId". A differenza della resource PUT per il calendario, questa può avere anche dei campi vuoti in input senza che si ottenga un errore, ovvero questi campi: "fusoOrario", "colore", "principale" e "GestioneImpostazioniPredefiniteEventi". Invece i parametri "nome" e "userId" devono essere per forza non vuoti. Però, è da specificare che dagli attributi "fusoOrario", "colore" e "principali" si possono ottenere altri errori, ovvero:
  - errore "400" nel caso in cui il colore inserito avesse un formato sbagliato ("400" con messaggio di errore "Wrong format for color");
  - errore "400" nel caso in cui il fusoOrario inserito avesse un formato sbagliato; infatti "GMTOffset" deve stare tra -12 e +12 e l'attributo "localita" deve essere diverso da "null". Se così non fosse riceviamo il codice "400" con messaggio di errore "Wrong format for fusoOrario";
  - errore "409" nel caso in cui principale fosse uguale a "true", ovvero stiamo creando il calendario principale di un utente, e l'utente, specificato dall' "userId" inserito, avesse già un calendario principale. Infatti non si possono avere più di un calendario principale, quindi otteniamo come risposta: "409 - There are too many primary calendars";

  Infine, l'errore "409" si potrebbe ottenere anche quando l' "userId" dell'utente, che sta creando il calendario, è un duplicato o non esiste nel database. Si ottiene come messaggio di errore: "There are too many users with that userId" o "There is no user with that userId" rispettivamente.
- "Get Calendari", resource di tipo GET. Resource che ha lo scopo di ottenere tutti i calendari appartenenti ad un utente, dato il suo "userId": questo, ovviamente, deve essere non vuoto per non ricevere il messaggio di errore "400 - Parameter missing". Inoltre, nel caso in cui passassimo l' "userId" di un utente autenticato che non ha calendari, otteniamo il codice "409" con il messaggio di errore "There are no calendars with that userId". Gli altri errori e messaggi che si possono ricevere sono gli stessi già descritti in generale all'inizio del capitolo.



Resources Model riguardo solo il "Calendario"

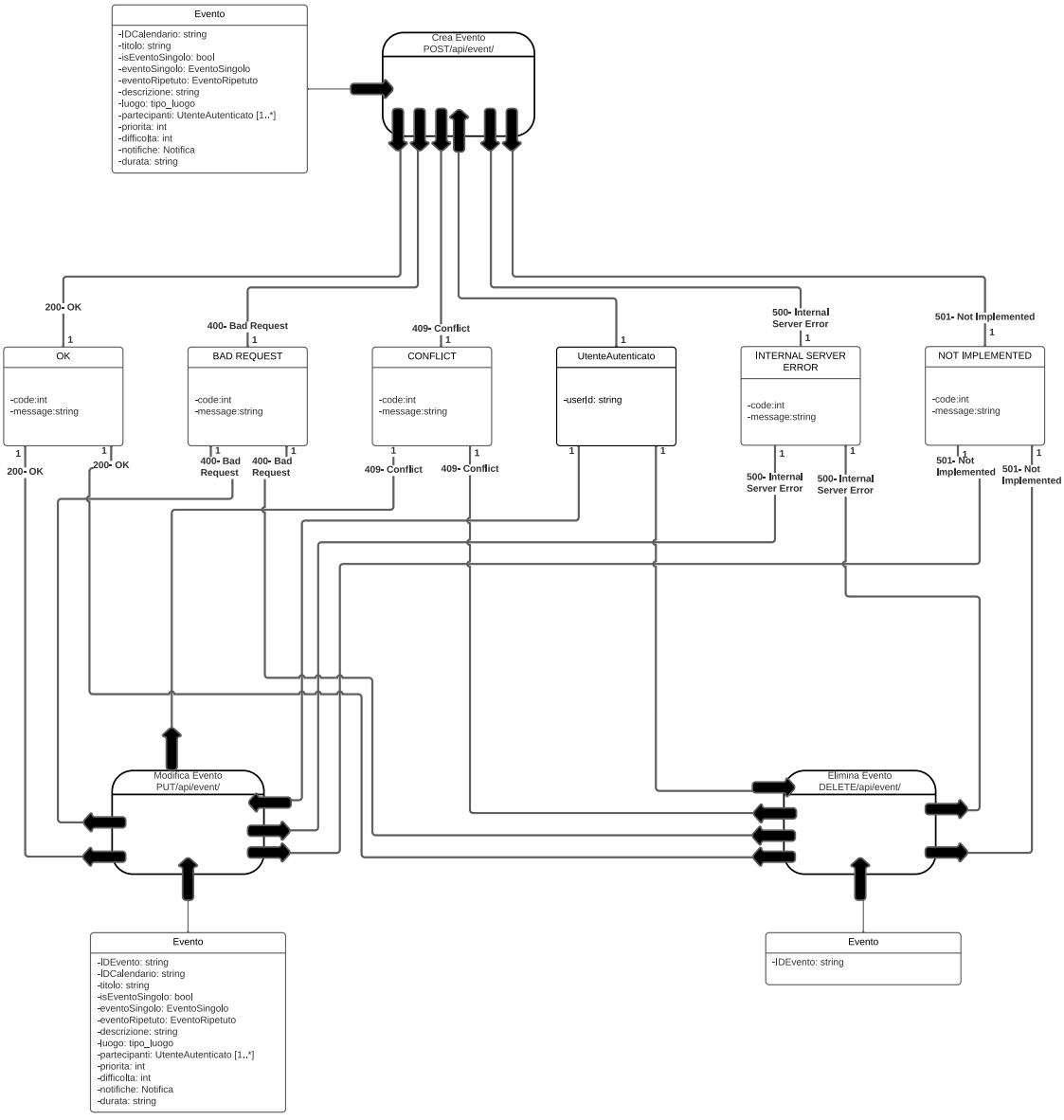
4.2.2 : Resource Models "Evento"

Nel seguente diagramma vengono mostrate le APIs che riguardano la struttura dati "Evento", già presentate in parte in APD4.1; le APIs individuate per "Evento" sono:

- "Crea Evento", resource di tipo POST. Questa API ha lo scopo di andare a salvare un evento nel database usando i parametri dati in input. Gli output sono sempre quelli citati in generale in precedenza, ma si specifica che i parametri obbligatori, per non ottenere l'errore "400 - IDCalendario or titolo or evento details missing", sono: "IDCalendario", id del calendario in cui stiamo aggiungendo questo evento, "titolo", titolo dell'evento, "userId", id dell'utente che sta creando l'evento, "isEventoSingolo", booleano che indica se l'evento che si sta creando è un evento singolo o ripetuto. Inoltre se "isEventoSingolo" è uguale a true si deve avere anche l'oggetto "eventoSingolo" diverso da "null", invece se questo è uguale a false, "eventoRipetuto" deve essere diverso da "null".
- "Modifica Evento", resource di tipo PUT. Questa resource ha lo scopo di andare a modificare un evento specifico appartenente ad un utente autenticato. A differenza dell' API "Crea Evento", questa API ha bisogno di tutti gli attributi che costituiscono la struttura dati "Evento" eccetto "eventoSingolo" o "eventoRipetuto", che possono essere uguali a "null", a seconda del valore del booleano isEventoSingolo, per i casi già citati per "Crea Evento". Dunque per non ricevere l'errore "400 - Parameter missing", questa resource ha bisogno di tutti gli attributi facenti parte di "Evento", che si possono osservare nel diagramma successivo in APD3. Gli altri errori che si possono ottenere da questa resource sono gli stessi già citati in precedenza in generale; ovviamente visto che i parametri da inserire obbligatoriamente sono di più, è più facile che si possa sbagliare e ottenere l'errore "400 - Wrong format for (inserted parameter uncorrectly)".

Immagine PNG/SVG Resources Model riguardo solo il "Calendario"

- "Elimina Evento", resource di tipo DELETE. Questa API ha lo scopo di eliminare un specifico evento di un utente autenticato, per questo motivo i parametri necessari sono: "IDEvento", "id" dell'evento che si vuole eliminare, e "userId", "id" dell'utente autenticato che vuole eliminare un suo evento. Se manca uno dei due parametri, ovviamente si ottiene l'errore "400 - Parameter missing". Gli altri output che si possono ottenere da questa API, sono quelli già descritti nella lista di possibile risposte di ciascuna API, con particolare attenzione al caso in cui viene inviato l' "IDEvento" di un evento che non appartiene a quel dato "userId"; in questo caso, si riceve l'errore "400 - You do not own the event".
- "Get Eventi", resource di tipo GET. API che ha la funzione di ritornare tutti gli eventi di un dato calendario. I parametri necessari, affinché questa resource possa effettuare il suo lavoro, sono l' "IDCalendario", "id" del calendario di cui vogliamo ottenere gli eventi, e l' "userId" dell'utente che ha tale calendario. Questi devono essere diversi da "null" per non ricevere il messaggio di errore "400 - Parameter missing". Inoltre, nel caso in cui passassimo l' "IDCalendario" di un calendario vuoto, ovvero senza eventi, viene ritornato il codice "409" con il messaggio di errore "There are no events with that userId and IDCalendario". Gli altri errori e messaggi che si possono ricevere sono gli stessi già descritti in generale all'inizio del capitolo.

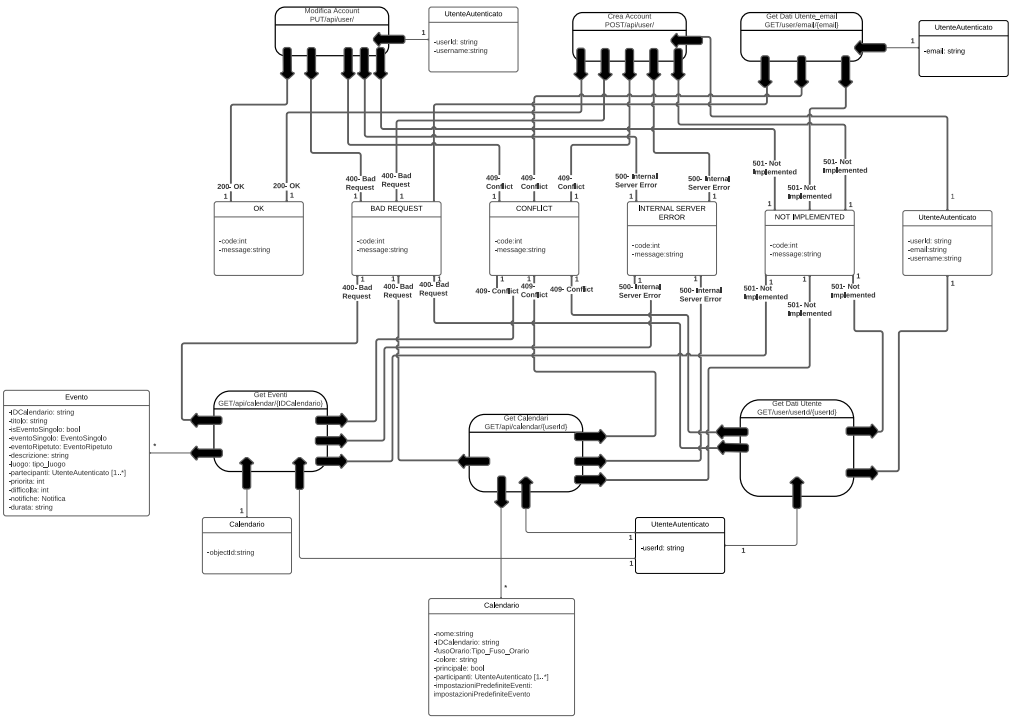


Resources Model riguardo solo l' "Evento"

4.2.3 : Resource Models "UtenteAutenticato"

Nel seguente diagramma vengono mostrate le APIs che riguardano la classe "UtenteAutenticato", già presentate in parte in APD4.1; le APIs individuate per "UtenteAutenticato" sono:

- "Modifica Account", resource di tipo PUT. La funzione di questa API è di modificare l' "username" di un utente autenticato, dato il suo "userId" e il nuovo "username"; ovviamente, per effettuare tale azione e non riceve il messaggio di errore "400 - Parameter missing", i parametri sopra citati devono essere sempre presenti. Gli altri errori che si possono riscontrare sono sempre i soliti presentati in generale all'inizio di questo capitolo.
- "Crea Account", resource di tipo POST. Lo scopo di questa resource è quella di creare un account all'interno del database, dato l' "email", l' "username" e l' "userId" dell'utente autenticato che si vuole creare; è importante specificare che tutti questi tre parametri devono essere diversi da "null" per non ottenere il messaggio di errore "400 - Parameter missing". Gli altri errori e risposte che si possono riscontrare sono gli stessi già citati in generale all'inizio del paragrafo.
- "Get Dati Utente" e "Get Dati Utente_email", APIs di tipo GET. Si descrivono assieme queste APIs, in quanto hanno la stessa funzione, ovvero ottenere i dati dell'account di un utente ("email", "userId", "username"), ma dando parametri diversi: per "Get Dati Utente" basta inviare l' "userId" dell'utente autenticato, invece per "Get Dati Utente_email" basta la propria "email". Si è deciso di mettere entrambi i GET, in quanto entrambi gli attributi "email" e "userId" dovrebbero univoci per ciascun utente autenticato. Ovviamente, per non ottenere il messaggio di errore "400 - Parameter missing", il parametro che deve essere sempre presente e diverso da "null" è l' "userId" o l' "email" rispettivamente. Gli altri errori e messaggi che si possono ricevere sono gli stessi già descritti in generale all'inizio del paragrafo.



Resources Model riguardo solo "UtenteAutenticato"

Immagine PNG/SVG Resources Model riguardo solo l' "Evento"
Immagine PNG/SVG Resources Model riguardo solo "UtenteAutenticato"

APD5 : Sviluppo API

In questo capitolo verranno presentate le API delle resources già sopra descritte, però andando più nello specifico mostrando il codice di come queste resources sono state implementate.

5.1 : Crea Calendario

Mediante questa API, l'applicativo salva nel database MongoDB un calendario con gli attributi uguali ai parametri ricevuti in input, per l'utente autenticato identificato dal suo "userId", parametro preso in input da questa funzione. Come già detto in precedenza nella descrizione della resource corrispondente in APD4.2.1, nel caso in cui mancasse l'attributo "nome" o l' "userId" non sarebbe possibile creare un calendario e quindi l'API manderebbe come messaggio di risposta il seguente errore: "400 - Name missing". Inoltre, come già detto in APD4.2.1, questa API controlla che il "colore" e "fusoOrario" inseriti rispettino degli standard affinché siano ritenuti accettabili: in caso non li rispettassero, riceveremmo rispettivamente i seguenti errori: "400 - Wrong format for color", "400 - Wrong format for fusoOrario".
Mediante la funzione "find()", l' API va a cercare nel database, gli utenti autenticati che hanno l' "userId" inserito; nel caso in cui trovasse più di un utente con questo "userId", si riceve un messaggio di errore del tipo "400 - There are too many users with that userId"; invece, nel caso in cui trovasse nessun utente autenticato con tale "userId" si riceve il messaggio di errore "400 - There is no user with that userId".
Infine, l'ultimo controllo, che si fa sugli attributi ricevuti in input, riguarda l'attributo "principale", booleano che indica se il calendario che si sta creando sia quello principale o meno. Infatti, ogni utente autenticato può avere al massimo un calendario principale, per questo motivo nel caso in cui si provasse a creare un calendario principale già esistente per quell' "userId", si riceve il messaggio di errore: "409 - There are too many primary calendars".
Grazie, alla funzione "create()", viene creato nel database un calendario con i valori dei campi che lo formano uguali ai parametri inseriti in input; nel caso ci fossero degli errori nell'esecuzione di questa funzione, otteniamo l'errore "500 - Not inserted". Se, invece l'esecuzione di questa funzione è andata a buon fine, otteniamo il messaggio di successo "200 - Calendar inserted correctly". C'è un eventuale altro controllo che serve per individuare errori generici durante l'esecuzione di creaCalendario, quindi otteniamo un messaggio di errore del tipo: "501 - Generic error".

```
export async function creaCalendario(req, res) {
  await dbConnect();
  try {
    const { nome, fusoOrario, colore, principale } = req.query;
    const { userId } = req.query;

    if (nome == null || userId == null) {
      res.status(400).json({ error: "Name missing" }); //TODO or userID
      return;
    }

    if (colore != null && !/^#([0-9a-f]{3}){1,2}$/i.test(colore)) {
      res.status(400).json({ error: "Wrong format for color" });
      return;
    }

    let tempFusoOrario

    if (fusoOrario != null) {
      try{
        tempFusoOrario = JSON.parse(fusoOrario)
      }catch{
        tempFusoOrario = fusoOrario
      }

      if (
        tempFusoOrario.GMTOffset == null ||
        tempFusoOrario.localita == null ||
        tempFusoOrario.GMTOffset > 12 ||
        tempFusoOrario.GMTOffset < -12
      ) {
        res.status(400).json({ error: "Wrong format for time zone" });
        return;
      }
    }

    const users = await UtenteAutenticato.find({
      userId: userId,
    });
    if (Object.keys(users).length == 0) {
      res.status(409).json({ error: "There is no user with that userId" });
      return;
    } else if (Object.keys(users).length > 1) {
      res
        .status(409)
        .json({ error: "There are too many users with that userId" });
      return;
    }

    if (principale === "true" || principale == true) {
      const calendariPrincipali = await Calendario.find({
        $and: [{ partecipanti: userId }, { principale: true }],
      });
      if (Object.keys(calendariPrincipali).length >= 1) {
        res.status(409).json({
          error: "There are too many primary calendars",
        });
        return;
      }
    }

    Calendario.create(
      {
        nome: nome,
        fusoOrario: fusoOrario == null ? undefined : tempFusoOrario,
        colore: colore == null ? undefined : colore,
        partecipanti: [userId],
        principale: principale == null ? false : principale,
        impostazioniPredefiniteEventi: undefined,
      },
      function (err, calendar) {
        if (err) {
          res.status(500).json({ error: "Not inserted" });
          return;
        }
      },
    );

    res.status(200).json({ success: "Calendar inserted correctly" });
    return;
  } catch (e) {
    console.error(e);
    res.status(501).json({ error: "Generic error" });
    throw new Error(e).message;
  }
}
```

API - creaCalendario

Immagine PNG API - creaCalendario

5.2 : **Modifica Calendario**

Grazie a questa API, già mostrata in parte in APD4.2.1, PlanIt riesce a modificare le impostazioni di un calendario già salvato per un utente autenticato. Per fare tale operazione, oltre all' "IDCalendario", "id" del calendario che si vuole modificare, e l' "userId", "id" dell'utente autenticato che vuole modificare un proprio calendario, sono necessari anche tutti gli altri attributi che costituiscono la struttura dati "Calendario", già molte volte mostrata come in (APD3), con i quali si andrà a modificare il calendario. Nel caso in cui mancasse uno di questi attributi, si riceve il messaggio di errore: "400 - Parameter missing".

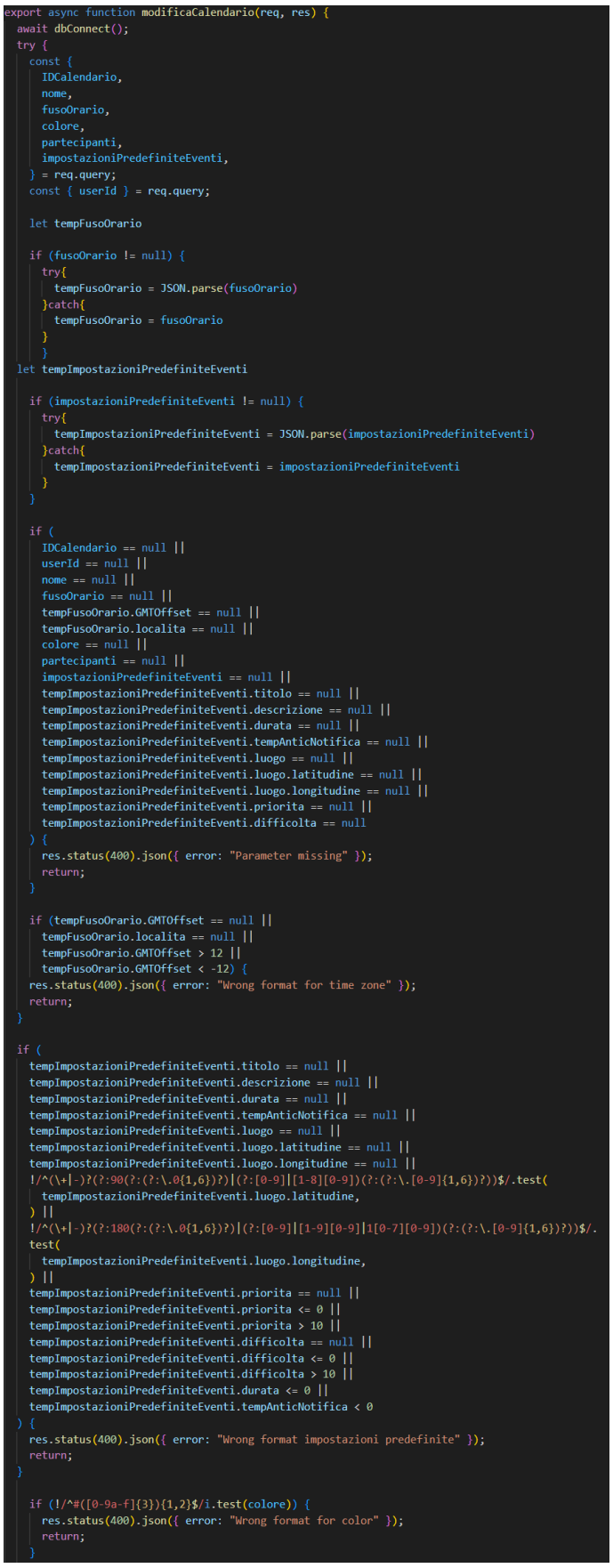
Come si può notare dal codice, in "modificaCalendario" sono presenti gli stessi controlli già descritti in "creaCalendario", ma con l'aggiunta di qualcuno di nuovo. Infatti per l'oggetto "impostazioniPredefiniteEventi", che ricordiamo essere l'insieme di valori con cui vengono precompilati gli eventi appartenenti ad un calendario specifico, valori che possono essere modificati sia durante la creazione che modifica di un evento in fase di compilazione, vengono fatti ulteriori accertamenti. In primo luogo, vengono controllati che ciascun campo che lo forma sia diverso da "null", inoltre:

- la "latitudine" e "longitudine" devono avere dei valori coerenti con la realtà (la longitudine deve essere nell'intervallo [-180,+180], invece la latitudine [-90,+90]);
- la "priorità" e "difficoltà" devono avere un valore tra 0 e 10;
- la "durata" deve essere maggiore di 0;
- "tempAnticNotifica", attributo che indica quanto prima inviare la notifica di un evento, deve essere maggiore e uguale a 0.

Nel caso in cui una delle restrizioni sopra citate non fossero rispettate per "impostazioniPredefiniteEventi", si riceverebbe il messaggio di errore "400 - Wrong format impostazioni predefinite".

Un altro controllo che viene fatto, è se il calendario, che si vuole modificare, appartenga o meno alla lista di calendari di quell'utente autenticato identificato dall' "userId". Questo controllo viene fatto grazie alla funzione "find()" che va a trovare nel database il calendario con l' "IDCalendario" inserito, e dopo aver ottenuto questo oggetto, viene controllato che il suo proprietario (primo utente presente nella lista di partecipanti al calendario, attributo che indica chi partecipa ad un calendario, ovvero ai suoi eventi) sia uguale all' "userId" inserito in input. Nel caso non lo fosse, si riceve il messaggio di errore: "409 - You do not own the calendar".

Infine, invece che usare la funzione "create()", usata per salvare un calendario nel database in "creaCalendario", viene usata la funzione "updataMany()" che va ad aggiornare gli attributi che costituiscono il calendario che si vuole modificare secondo i parametri ricevuti in input.



```
export async function modificaCalendario(req, res) {
  await dbConnect();
  try {
    const {
      IDCalendario,
      nome,
      fusoOrario,
      colore,
      partecipanti,
      impostazioniPredefiniteEventi,
    } = req.query;
    const { userId } = req.query;

    let tempFusoOrario

    if (fusoOrario != null) {
      try{
        tempFusoOrario = JSON.parse(fusoOrario)
      }catch{
        tempFusoOrario = fusoOrario
      }
      }
  let tempImpostazioniPredefiniteEventi

    if (impostazioniPredefiniteEventi != null) {
      try{
        tempImpostazioniPredefiniteEventi = JSON.parse(impostazioniPredefiniteEventi)
      }catch{
        tempImpostazioniPredefiniteEventi = impostazioniPredefiniteEventi
      }
    }

    if (
      IDCalendario == null ||
      userId == null ||
      nome == null ||
      fusoOrario == null ||
      tempFusoOrario.GMTOffset == null ||
      tempFusoOrario.localita == null ||
      colore == null ||
      partecipanti == null ||
      impostazioniPredefiniteEventi == null ||
      tempImpostazioniPredefiniteEventi.titolo == null ||
      tempImpostazioniPredefiniteEventi.descrizione == null ||
      tempImpostazioniPredefiniteEventi.durata == null ||
      tempImpostazioniPredefiniteEventi.tempAnticNotifica == null ||
      tempImpostazioniPredefiniteEventi.luogo == null ||
      tempImpostazioniPredefiniteEventi.luogo.latitudine == null ||
      tempImpostazioniPredefiniteEventi.luogo.longitudine == null ||
      tempImpostazioniPredefiniteEventi.priorita == null ||
      tempImpostazioniPredefiniteEventi.difficolta == null
    ) {
      res.status(400).json({ error: "Parameter missing" });
      return;
    }

    if (tempFusoOrario.GMTOffset == null ||
      tempFusoOrario.localita == null ||
      tempFusoOrario.GMTOffset > 12 ||
      tempFusoOrario.GMTOffset < -12) {
    res.status(400).json({ error: "Wrong format for time zone" });
    return;
  }

  if (
    tempImpostazioniPredefiniteEventi.titolo == null ||
    tempImpostazioniPredefiniteEventi.descrizione == null ||
    tempImpostazioniPredefiniteEventi.durata == null ||
    tempImpostazioniPredefiniteEventi.tempAnticNotifica == null ||
    tempImpostazioniPredefiniteEventi.luogo == null ||
    tempImpostazioniPredefiniteEventi.luogo.latitudine == null ||
    tempImpostazioniPredefiniteEventi.luogo.longitudine == null ||
    !/^(\+|-)?(?:90(?:(?:\.0{1,6})?)|(?:[0-9]|[1-8][0-9])(?:(?:\.[0-9]{1,6})?))$/.test(
      tempImpostazioniPredefiniteEventi.luogo.latitudine,
    ) ||
    !/^(\+|-)?(?:180(?:(?:\.0{1,6})?)|(?:[0-9]|[1-9][0-9]|1[0-7][0-9])(?:(?:\.[0-9]{1,6})?))$/.
    test(
      tempImpostazioniPredefiniteEventi.luogo.longitudine,
    ) ||
    tempImpostazioniPredefiniteEventi.priorita == null ||
    tempImpostazioniPredefiniteEventi.priorita <= 0 ||
    tempImpostazioniPredefiniteEventi.priorita > 10 ||
    tempImpostazioniPredefiniteEventi.difficolta == null ||
    tempImpostazioniPredefiniteEventi.difficolta <= 0 ||
    tempImpostazioniPredefiniteEventi.difficolta > 10 ||
    tempImpostazioniPredefiniteEventi.durata <= 0 ||
    tempImpostazioniPredefiniteEventi.tempAnticNotifica < 0
  ) {
    res.status(400).json({ error: "Wrong format impostazioni predefinite" });
    return;
  }

    if (!/^#([0-9a-f]{3}){1,2}$/i.test(colore)) {
      res.status(400).json({ error: "Wrong format for color" });
      return;
    }
```

API - modificaCalendario, 1a parte

Immagine PNG API - modificaCalendario

```
    const users = await UtenteAutenticato.find({
      userId: userId,
    });
    if (Object.keys(users).length == 0) {
      res.status(409).json({ error: "There is no user with that userId" });
      return;
    } else if (Object.keys(users).length > 1) {
      res
        .status(409)
        .json({ error: "There are too many users with that userId" });
      return;
    }
    var ObjectId = require("mongoose").Types.ObjectId;

    const calendariPosseduti = await Calendario.find({
      $and: [{ partecipanti: userId }, { _id: new ObjectId(IDCalendario) }],
    });
    if (
      Object.keys(calendariPosseduti).length == 0 ||
      calendariPosseduti[0].partecipanti[0] != userId
    ) {
      res.status(409).json({
        error: "You do not own the calendar",
      });
      return;
    }

    let tempPartecipanti = partecipanti.filter(item => item !== userId)

    Calendario.updateMany(
      { _id: new ObjectId(IDCalendario) },
      {
        partecipanti: [userId],
      },
      function (err, calendar) {
        if (err) {
          res.status(500).json({ error: "Not edited" });
          return;
        }
      },
    );
    Calendario.updateMany(
      { _id: new ObjectId(IDCalendario) },
      {
        nome: nome,
        fusoOrario: tempFusoOrario,
        colore: colore,
        $addToSet: {partecipanti: { $each: tempPartecipanti}},
        impostazioniPredefiniteEventi: tempImpostazioniPredefiniteEventi,
      },
      function (err, calendar) {
        if (err) {
          res.status(500).json({ error: "Not edited" });
          return;
        }
      },
    );

    res.status(200).json({ success: "Calendar updated correctly" });
    return;
  } catch (e) {
    console.error(e);
    res.status(501).json({ error: "Generic error" });
    throw new Error(e).message;
  }
}
```

API - modificaCalendario, 2a parte

5.3 : Elimina Calendario

Lo scopo di questa API, come già descritto in APD4.2.1, è quella di eliminare un calendario, dato il suo "IDCalendario" e l' "userId", per questo motivo questi due parametri non devono essere vuoti, sennò si riceve il messaggio di errore "400 - Parameter missing". Anche in questa API, come in "modificaCalendario" (APD5.2), viene controllato che l' "userId" inviato non sia un duplicato, che esista, che il calendario che si vuole eliminare appartenga alla lista di calendari dell'utente autenticato identificato dall' "userId".
Infine, per eliminare il calendario dal database viene usata la funzione "deleteMany()" che sfrutta solo l' "IDCalendario" inviato in input per identificare il calendario da eliminare. Nel caso in cui non fosse eliminato nessun calendario per qualche motivo, c'è

Immagine PNG API - modificaCalendario, 2a parte

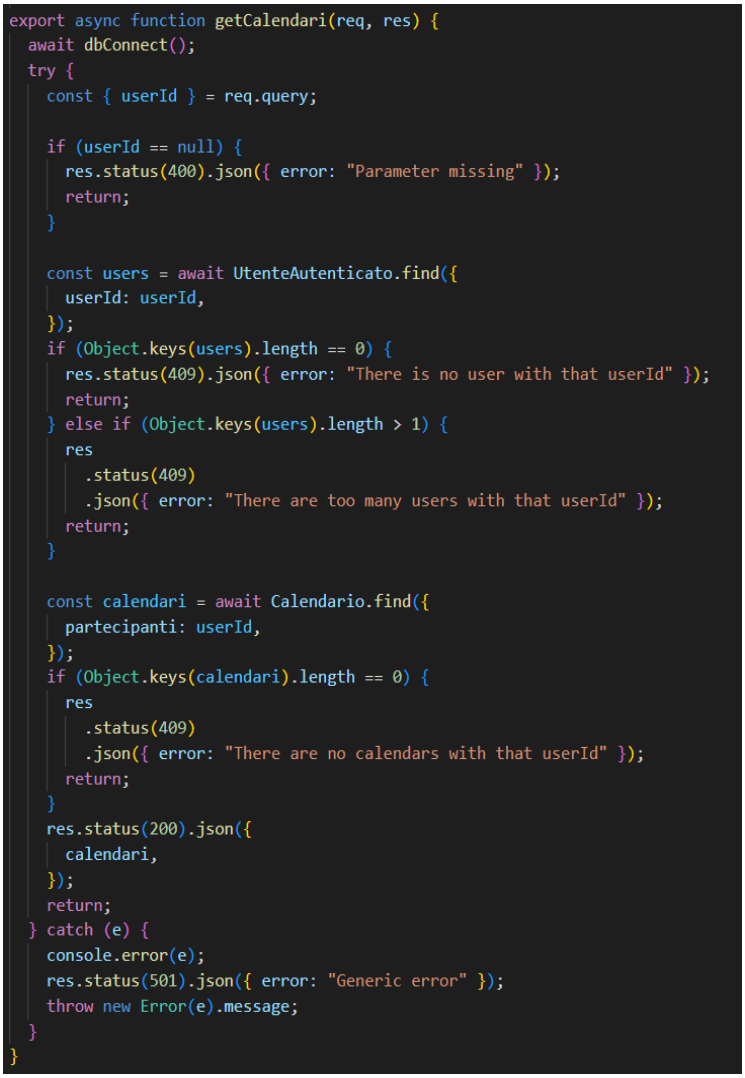
l'invio dell'errore "500 - Calendar not deleted"; un motivo che porta questo errore è la sconnessione improvvisa dal database MongoDB.

```
export async function eliminaCalendario(req, res) {
  await dbConnect();
  try {
    const { IDCalendario } = req.query;
    const { userId } = req.query;

    if (IDCalendario == null || userId == null) {
      res.status(400).json({ error: "Parameter missing" });
      return;
    }

    const users = await UtenteAutenticato.find({
      userId: userId,
    });
    if (Object.keys(users).length == 0) {
      res.status(409).json({ error: "There is no user with that userId" });
      return;
    } else if (Object.keys(users).length > 1) {
      res
        .status(409)
        .json({ error: "There are too many users with that userId" });
      return;
    }
    var ObjectId = require("mongoose").Types.ObjectId;

    const calendariPosseduti = await Calendario.find({
      $and: [{ partecipanti: userId }, { _id: new ObjectId(IDCalendario) }],
    });
    if (
      Object.keys(calendariPosseduti).length == 0 ||
      calendariPosseduti[0].partecipanti[0] != userId
    ) {
      res.status(409).json({
        error: "You do not own the calendar",
      });
      return;
    }

    const deleteCalendar = await Calendario.deleteMany({
      _id: new ObjectId(IDCalendario),
    });

    if (deleteCalendar.deletedCount >= 1) {
      res.status(200).json({ success: "Calendar deleted correctly" });
      return;
    } else {
      res.status(500).json({ error: "Calendar not deleted" });
      return;
    }
  } catch (e) {
    console.error(e);
    res.status(501).json({ error: "Generic error" });
    throw new Error(e).message;
  }
}
```

API - eliminaCalendario

5.4 : getCalendari

L'applicativo PlanIt usa questa API, come già detto in APD4.2.1, per ottenere tutti i calendari dell'utente autenticato identificato dall' "userId" ricevuto in input. Dunque l'unico parametro che deve essere presente, per non riceve l'errore "400 - Parameter missing", è l' "userId", su cui si fanno sempre i soliti controlli. Alla fine della procedura, se questa va a buon fine, si ottiene l'array di strutture dati "Calendario" (già mostrata in APD3) che appartengono all' utente autenticato che ha tale "userId".



```
export async function getCalendari(req, res) {
  await dbConnect();
  try {
    const { userId } = req.query;

    if (userId == null) {
      res.status(400).json({ error: "Parameter missing" });
      return;
    }

    const users = await UtenteAutenticato.find({
      userId: userId,
    });
    if (Object.keys(users).length == 0) {
      res.status(409).json({ error: "There is no user with that userId" });
      return;
    } else if (Object.keys(users).length > 1) {
      res
        .status(409)
        .json({ error: "There are too many users with that userId" });
      return;
    }

    const calendari = await Calendario.find({
      partecipanti: userId,
    });
    if (Object.keys(calendari).length == 0) {
      res
        .status(409)
        .json({ error: "There are no calendars with that userId" });
      return;
    }
    res.status(200).json({
      calendari,
    });
    return;
  } catch (e) {
    console.error(e);
    res.status(501).json({ error: "Generic error" });
    throw new Error(e).message;
  }
}
```

API - getCalendari

Immagine PNG API - eliminaCalendario
Immagine PNG API - getCalendari

5.5 : Crea Evento

Mediante questa API, l'applicativo PlanIt riesce a salvare nel database MongoDB un evento con gli attributi uguali ai parametri inviati in input, per l'utente autenticato identificato dall' "userId" indicato. Già in APD4.2.2, nella descrizione della risorsa "Crea Evento", siamo andati a presentare quali sono i casi in cui questa API invia come messaggio di errore "400 - IDCalendario or titolo or evento details missing", per questo motivo non andremo a ripeterli nuovamente.

Come si può notare per gli attributi "luogo", "prorita", "difficolta", "notifiche" e "durata" vengono fatti gli stessi controlli già presentati in "modificaCalendario" (APD5.2) per l'attributo "impostazioniPredefiniteEventi", in quanto quest'ultimo oggetto contiene gli attributi sopra citati, presenti anche durante la crazione e modifica di un evento, dunque non vengono riscritti tutti specificamente. Nel caso in cui uno di questi attributi non passasse un controllo, riceviamo un errore del tipo "400 - Wrong format (parameter)".

I controlli non presenti nelle altre API già descritte, riguardano gli attributi "eventoSingolo" ed "eventoRipetuto", oggetti che contengono informazioni fondamentali per creare un evento singolo, evento che poniamo solo in un giorno in fase di creazione, e un evento ripetuto, evento che poniamo su più giornate in fase di creazione o modifica. Per la struttura dati "eventoSingolo" andiamo a controllare che gli attributi "data" e "isScadenza" (questo attributo indica se l'evento che si sta mettendo con tale data sia una deadline di qualcosa oppure no) siano diversi da "null": se lo fossero riceviamo il messaggio di errore "400 - Wrong format for eventoSingolo". Invece, per la struttura dati "eventoRipetuto" andiamo ad controllare che "numeroRipetizione", attributo che indica quante volte si ripete tale evento, sia diverso da "null" e maggiore e uguale di 1, che "data", attributo che indica il giorno, mese, anno e orario dell'evento, sia diverso da "null" e che, infine, "giornidellaSettimana", attributo che indica in quali giorni si ripete tale evento, sia diverso da "null" e che non sia un array vuoto. Se questi attributi non rispettassero queste restrizioni, riceviamo l'errore "400 - Wrong format for eventoRipetuto".

Dopo sono presenti i soliti controlli già citati per il parametro "userId", ma evidenziamo la presenza della funzione "find()" che viene utilizzata per andare a vedere se il calendario, dove stiamo aggiungendo l'evento da che si sta creando, esista e appartenga all'utente autenticato identificato dall' "userId"; se così non fosse, otteniamo il messaggio di errore "409 - There is no calendar with that ID or you do not own the calendar".

Come per "creaCalendario" (APD5.1), viene usata la funzione "create()" per andare a creare nel database MongoDB un oggetto "Evento" con le caratteristiche dei parametri ricevuti in input.

```
export async function creaEvento(req, res) {
  await dbConnect();
  try {
    const {
      IDCalendario,
      titolo,
      descrizione,
      luogo,
      priorita,
      difficolta,
      partecipanti,
      notifiche,
      durata,
      isEventoSingolo,
      eventoSingolo,
      eventoRipetuto,
    } = req.query;
    const { userId } = req.query;

    if (
      IDCalendario == null ||
      titolo == null ||
      userId == null ||
      isEventoSingolo == null ||
      (isEventoSingolo == true && eventoSingolo == null) ||
      (isEventoSingolo == false && eventoRipetuto == null)
    ) {
      res
        .status(400)
        .json({ error: "IDCalendario or titolo or evento details missing" }); //TODO or userID
      return;
    }

    let tempLuogo;
    if (luogo != null) {
      try {
        tempLuogo = JSON.parse(luogo);
      } catch {
        tempLuogo = luogo;
      }
      if (
        tempLuogo.latitudine == null ||
        tempLuogo.longitudine == null ||
        !/^(\+|-)?(?:90(?:(?:\.0{1,6})?)|(?:[0-9]|[1-8][0-9])(?:(?:\.[0-9]{1,6})?))$/.test(
          tempLuogo.latitudine
        ) ||
        !/^(\+|-)?(?:180(?:(?:\.0{1,6})?)|(?:[0-9]|[1-9][0-9]|1[0-7][0-9])(?:(?:\.[0-9]{1,
        6})?))$/.test(
          tempLuogo.longitudine
        )
      ) {
        res.status(400).json({ error: "Wrong format for location" });
        return;
      }
    }
    if (priorita != null && (priorita <= 0 || priorita > 10)) {
      res.status(400).json({ error: "Wrong format for priorita" });
      return;
    }
    if (difficolta != null && (difficolta <= 0 || difficolta > 10)) {
      res.status(400).json({ error: "Wrong format for difficolta" });
      return;
    }
    let tempNotifiche;
    if (notifiche != null) {
      try {
        tempNotifiche = JSON.parse(notifiche);
      } catch {
        tempNotifiche = notifiche;
      }

      if (tempNotifiche.titolo == null || tempNotifiche.data == null) {
        res.status(400).json({ error: "Wrong format for notifiche" });
        return;
      }
    }
    if (durata != null && durata <= 0) {
      res.status(400).json({ error: "Wrong format for durata" });
      return;
    }

    let tempEvento;

    if (isEventoSingolo) {
      if (eventoSingolo != null) {
        try {
          tempEvento = JSON.parse(eventoSingolo);
        } catch {
          tempEvento = eventoSingolo;
        }
        if (tempEvento.data == null || tempEvento.isScadenza == null) {
          res.status(400).json({ error: "Wrong format for eventoSingolo" });
          return;
        }
      }
```

API - creaEvento, 1a parte

Immagine PNG API - creaEvento, 1a parte

```
if (isEventoSingolo) {
  if (eventoSingolo != null) {
    try {
      tempEvento = JSON.parse(eventoSingolo);
    } catch {
      tempEvento = eventoSingolo;
    }
    if (tempEvento.data == null || tempEvento.isScadenza == null) {
      res.status(400).json({ error: "Wrong format for eventoSingolo" });
      return;
    }
  }
} else {
  if (eventoRipetuto != null) {
    try {
      tempEvento = JSON.parse(eventoRipetuto);
    } catch {
      tempEvento = eventoRipetuto;
    }
    if (
      tempEvento.numeroRipetizioni == null ||
      tempEvento.impostazioniAvanzate == null ||
      tempEvento.impostazioniAvanzate.giorniSettimana == null ||
      tempEvento.impostazioniAvanzate.data == null ||
      tempEvento.numeroRipetizioni < 1 ||
      tempEvento.impostazioniAvanzate.giorniSettimana == []
    ) {
      res.status(400).json({ error: "Wrong format for eventoRipetuto" });
      return;
    }
  }
}

const users = await UtenteAutenticato.find({
  userId: userId,
});
if (Object.keys(users).length == 0) {
  res.status(409).json({ error: "There is no user with that userId" });
  return;
} else if (Object.keys(users).length > 1) {
  res
    .status(409)
    .json({ error: "There are too many users with that userId" });
  return;
}

var ObjectId = require("mongoose").Types.ObjectId;

const calendariPosseduti = await Calendario.find({
  $and: [{ partecipanti: userId }, { _id: new ObjectId(IDCalendario) }],
});
if (
  Object.keys(calendariPosseduti).length == 0 ||
  calendariPosseduti[0].partecipanti[0] != userId
) {
  res.status(409).json({
    error:
      "There is no calendar with that ID or you do not own the calendar",
  });
  return;
}

let tempPartecipanti = partecipanti != null && Array.isArray(partecipanti)? partecipanti.
filter((item) => item !== userId) : calendariPosseduti[0].partecipanti;

if (isEventoSingolo) {
  Evento.create(
    {
      IDCalendario: IDCalendario,
      titolo: titolo,
      descrizione: descrizione == null ? undefined : descrizione,
      luogo: luogo == null ? undefined : tempLuogo,
      priorita: priorita == null ? undefined : priorita,
      difficolta: difficolta == null ? undefined : difficolta,
      $addToSet: {partecipanti: { $each: tempPartecipanti}},
      notifiche: notifiche == null ? undefined : tempNotifiche,
      durata: durata == null ? undefined : durata,
      isEventoSingolo: true,
      eventoSingolo: eventoSingolo == null ? undefined : tempEvento,
    },
    function (err, calendar) {
      if (err) {
        res.status(500).json({ error: "Not inserted" });
        return;
      }
    }
  );
```

API - creaEvento, 2a parte

Immagine PNG API - creaEvento, 2a parte

```
    } else {
      Evento.create(
        {
          IDCalendario: IDCalendario,
          titolo: titolo,
          descrizione: descrizione == null ? undefined : descrizione,
          luogo: luogo == null ? undefined : tempLuogo,
          priorita: priorita == null ? undefined : priorita,
          difficolta: difficolta == null ? undefined : difficolta,
          $addToSet: {partecipanti: { $each: tempPartecipanti}},
          notifiche: notifiche == null ? undefined : tempNotifiche,
          durata: durata == null ? undefined : durata,
          isEventoSingolo: false,
          eventoRipetuto: eventoRipetuto == null ? undefined : tempEvento,
        },
        function (err, calendar) {
          if (err) {
            res.status(500).json({ error: "Not inserted" });
            return;
          }
        }
      );
    }

    res.status(200).json({ success: "Event inserted correctly" });
    return;
  } catch (e) {
    console.error(e);
    res.status(501).json({ error: "Generic error" });
    throw new Error(e).message;
  }
}
```

API - creaEvento, 3a parte

5.6 : Modifica Evento

Grazie a questa API, come già detto in APD4.2.2, è possibile modificare un "Evento" già salvato nel database MongoDB, con i vari parametri ricevuti in input, che non sono altro che gli attributi che formano la struttura dati "Evento", struttura dati già più volte mostrata (APD3), e l' "userId" dell'utente autenticato che vuole modificare tale evento. I controlli, presenti in questa API, sono gli stessi già descritti per "creaEvento" (APD5.5) e "modificaCalendario" (APD5.2); l'unica differenza è che per non ricevere il messaggio di errore "400 - Parameter missing", tutti gli attributi che andranno a formare l'oggetto "Evento" (APD3) e l' "userId" devono essere diversi da "null".
Alla fine della funzione, grazie alla funzione "updateMany()", viene aggiornato l'oggetto "Evento" presente nel database secondo i parametri ricevuti, come per "modificaCalendario" (APD5.2). Specifichiamo che a seconda se stiamo andando a modificare un "eventoSingolo" o "eventoRipetuto" abbiamo una diversa procedura di aggiornamento.

Immagine PNG API - creaEvento, 3a parte

```
export async function modificaEvento(req, res) {
  await dbConnect();
  try {
    const {
      IDEvento,
      IDCalendario,
      titolo,
      descrizione,
      luogo,
      priorita,
      difficolta,
      partecipanti,
      notifiche,
      durata,
      isEventoSingolo,
      eventoSingolo,
      eventoRipetuto,
    } = req.query;
    const { userId } = req.query;

    let tempLuogo;
    if (luogo != null) {
      try {
        tempLuogo = JSON.parse(luogo);
      } catch {
        tempLuogo = luogo;
      }
    }
    let tempNotifiche;
    if (notifiche != null) {
      try {
        tempNotifiche = JSON.parse(notifiche);
      } catch {
        tempNotifiche = notifiche;
      }
    }
    let tempEvento;
    if (isEventoSingolo) {
      if (eventoSingolo != null) {
        try {
          tempEvento = JSON.parse(eventoSingolo);
        } catch {
          tempEvento = eventoSingolo;
        }
      }
    } else {
      if (eventoRipetuto != null) {
        try {
          tempEvento = JSON.parse(eventoRipetuto);
        } catch {
          tempEvento = eventoRipetuto;
        }
      }
    }

    if (
      IDEvento == null ||
      IDCalendario == null ||
      titolo == null ||
      userId == null ||
      isEventoSingolo == null ||
      (isEventoSingolo == true && eventoSingolo == null) ||
      (isEventoSingolo == false && eventoRipetuto == null) ||
      descrizione == null ||
      luogo == null ||
      partecipanti == null ||
      priorita == null ||
      difficolta == null ||
      notifiche == null ||
      durata == null
    ) {
      res.status(400).json({ error: "Parameter missing" });
      return;
    }

    if (priorita <= 0 || priorita > 10) {
      res.status(400).json({ error: "Wrong format for priorita" });
      return;
    }
    if (difficolta <= 0 || difficolta > 10) {
      res.status(400).json({ error: "Wrong format for difficolta" });
      return;
    }

    if (durata <= 0) {
      res.status(400).json({ error: "Wrong format for durata" });
      return;
    }

    if (
      tempLuogo.latitudine == null ||
      tempLuogo.longitudine == null ||
      !/^(\+|-)?(?:90(?:(?:\.0{1,6})?)|(?:[0-9]|[1-8][0-9])(?:(?:\.[0-9]{1,6})?))$/.test(
        tempLuogo.latitudine
      ) ||
      !/^(\+|-)?(?:180(?:(?:\.0{1,6})?)|(?:[0-9]|[1-9][0-9]|1[0-7][0-9])(?:(?:\.[0-9]{1,6})?))
      $/.test(
        tempLuogo.longitudine
      )
    ) {
      res.status(400).json({ error: "Wrong format for location" });
      return;
```

API - modificaEvento, 1a parte

Immagine PNG API - modificaEvento, 1a parte

```
if (tempNotifiche.titolo == null || tempNotifiche.data == null) {
  res.status(400).json({ error: "Wrong format for notifiche" });
  return;
}

if (isEventoSingolo) {
  if (eventoSingolo != null) {
    if (tempEvento.data == null || tempEvento.isScadenza == null) {
      res.status(400).json({ error: "Wrong format for eventoSingolo" });
      return;
    }
  }
} else {
  if (eventoRipetuto != null) {
    if (
      tempEvento.numeroRipetizioni == null ||
      tempEvento.impostazioniAvanzate == null ||
      tempEvento.impostazioniAvanzate.giorniSettimana == null ||
      tempEvento.impostazioniAvanzate.data == null ||
      tempEvento.numeroRipetizioni < 1 ||
      tempEvento.impostazioniAvanzate.giorniSettimana == []
    ) {
      res.status(400).json({ error: "Wrong format for eventoRipetuto" });
      return;
    }
  }
}

const users = await UtenteAutenticato.find({
  userId: userId,
});
if (Object.keys(users).length == 0) {
  res.status(409).json({ error: "There is no user with that userId" });
  return;
} else if (Object.keys(users).length > 1) {
  res
    .status(409)
    .json({ error: "There are too many users with that userId" });
  return;
}
var ObjectId = require("mongoose").Types.ObjectId;

const evento = await Evento.find({
  _id: new ObjectId(IDEvento),
  partecipanti: userId,
});
if (
  Object.keys(evento).length == 0 ||
  evento[0].partecipanti[0] != userId
) {
  res.status(409).json({
    error: "You do not own the event",
  });
  return;
}

const calendario = await Calendario.find({
  _id: new ObjectId(IDCalendario),
  partecipanti: userId,
});
if (
  Object.keys(calendario).length == 0 ||
  calendario[0].partecipanti[0] != userId
) {
  res.status(409).json({
    error:
      "There is no calendar with that ID or you do not own the calendar",
  });
  return;
}

let tempPartecipanti = partecipanti.filter((item) => item !== userId);

Evento.updateMany(
  { _id: new ObjectId(IDEvento) },
  {
    partecipanti: [userId],
  },
  function (err, calendar) {
    if (err) {
      res.status(500).json({ error: "Not modified" });
      return;
    }
  }
);
```

API - modificaEvento, 2a parte

Immagine PNG API - modificaEvento, 2a parte

```
  if (isEventoSingolo) {
    Evento.updateMany(
      { _id: new ObjectId(IDEvento) },
      {
        IDCalendario: IDCalendario,
        titolo: titolo,
        descrizione: descrizione == null ? undefined : descrizione,
        luogo: luogo == null ? undefined : tempLuogo,
        priorita: priorita == null ? undefined : priorita,
        difficolta: difficolta == null ? undefined : difficolta,
        $addToSet: {partecipanti: { $each: tempPartecipanti}},
        notifiche: notifiche == null ? undefined : tempNotifiche,
        durata: durata == null ? undefined : durata,
        isEventoSingolo: true,
        eventoSingolo: eventoSingolo == null ? undefined : tempEvento,
      },
      function (err, calendar) {
        if (err) {
          res.status(500).json({ error: "Not modified" });
          return;
        }
      }
    );
  } else {
    Evento.updateMany(
      { _id: new ObjectId(IDEvento) },
      {
        IDCalendario: IDCalendario,
        titolo: titolo,
        descrizione: descrizione == null ? undefined : descrizione,
        luogo: luogo == null ? undefined : tempLuogo,
        priorita: priorita == null ? undefined : priorita,
        difficolta: difficolta == null ? undefined : difficolta,
        $addToSet: {partecipanti: { $each: tempPartecipanti}},
        notifiche: notifiche == null ? undefined : tempNotifiche,
        durata: durata == null ? undefined : durata,
        isEventoSingolo: false,
        eventoRipetuto: eventoRipetuto == null ? undefined : tempEvento,
      },
      function (err, calendar) {
        if (err) {
          res.status(500).json({ error: "Not modified" });
          return;
        }
      }
    );
  }

  res.status(200).json({ success: "Event edited correctly" });
  return;
} catch (e) {
  console.error(e);
  res.status(501).json({ error: "Generic error" });
  throw new Error(e).message;
}
}
```

API - modificaEvento, 3a parte

5.7 : Elimina Evento

Questa API, come già anticipato in APD4.2.2, ha lo scopo di andare ad eliminare un "Evento", dato il suo "IDEvento" e l' "userId" dell'utente autenticato che ha questo evento che vuole eliminare. Ovviamente, nel caso in cui non avessimo uno di questi due parametri, come già detto in APD4.2.2, otteniamo un messaggio di errore del tipo "400 - Parameter missing".

In seguito, vengono fatti sempre gli stessi controlli già citati in "EliminaCalendario" APD5.3, ma stavolta per gli eventi. Infatti si controlla che l'evento che si vuole eliminare faccia parte della lista degli eventi appartenenti all'utente autenticato indicato dall' "userId". Se così non fosse, l'API invia l'errore "409 - You do not own the event".

Infine, anche per questa funzione vengono usate le funzioni "find()" e "deleteMany()" per ottenere rispettivamente la lista di eventi di un utente autenticato e per eliminare l'evento identificato dall' "IDEvento".

Immagine PNG API - modificaEvento, 3a parte

```
export async function eliminaEvento(req, res) {
  await dbConnect();
  try {
    const { IDEvento } = req.query;
    const { userId } = req.query;

    if (IDEvento == null || userId == null) {
      res.status(400).json({ error: "Parameter missing" });
      return;
    }

    const users = await UtenteAutenticato.find({
      userId: userId,
    });
    if (Object.keys(users).length == 0) {
      res.status(409).json({ error: "There is no user with that userId" });
      return;
    } else if (Object.keys(users).length > 1) {
      res
        .status(409)
        .json({ error: "There are too many users with that userId" });
      return;
    }
    var ObjectId = require("mongoose").Types.ObjectId;

    const eventiPosseduti = await Evento.find({
      $and: [{ partecipanti: userId }, { _id: new ObjectId(IDEvento) }],
    });
    if (
      Object.keys(eventiPosseduti).length == 0 ||
      eventiPosseduti[0].partecipanti[0] != userId
    ) {
      res.status(409).json({
        error: "You do not own the event",
      });
      return;
    }

    const deleteEvent = await Evento.deleteMany({
      _id: new ObjectId(IDEvento),
    });

    if (deleteEvent.deletedCount >= 1) {
      res.status(200).json({ success: "Event deleted correctly" });
      return;
    } else {
      res.status(500).json({ error: "Event not deleted" });
      return;
    }
  } catch (e) {
    console.error(e);
    res.status(501).json({ error: "Generic error" });
    throw new Error(e).message;
  }
}
```

API - eliminaEvento

Immagine PNG API - eliminaEvento

5.8 : getEventi

Questa API, precedentemente descritta in APD4.2.2, viene usata per ottenere tutti gli eventi che appartengono ad un calendario di un utente autenticato. Questo calendario viene identificato dall' "IDCalendario" e l' "userId" ricevuti in input, i quali, per ottenere un successo dalla procedura dell'API, non devono essere vuoti. Sul parametro "userId" vengono fatti i soliti controlli già presenti nelle altre API. Invece, dato l' "IDCalendario" e l' "userId", grazie alla funzione "find()", "getEventi" controlla che tale calendario esista e che appartenga alla lista dei calendari dell'utente autenticato; se così non si fosse, si riceve il messaggio di errore "409 - There is no calendar with that ID or you are not part of it".
Infine, si controlla anche che questo calendario non sia vuoto, ovvero che abbia almeno un evento, in caso lo fosse si ottiene il messaggio di errore "409 - There are no events with that userId and IDCalendario", come già detto in APD4.2.2.

```
export async function getEventi(req, res) {
  await dbConnect();
  try {
    const { IDCalendario } = req.query;
    const { userId } = req.query;

    if (IDCalendario == null || userId == null) {
      res.status(400).json({ error: "Parameter missing" });
      return;
    }

    const users = await UtenteAutenticato.find({
      userId: userId,
    });
    if (Object.keys(users).length == 0) {
      res.status(409).json({ error: "There is no user with that userId" });
      return;
    } else if (Object.keys(users).length > 1) {
      res
        .status(409)
        .json({ error: "There are too many users with that userId" });
      return;
    }
    var ObjectId = require("mongoose").Types.ObjectId;

    const calendariPosseduti = await Calendario.find({
      $and: [{ partecipanti: userId }, { _id: new ObjectId(IDCalendario) }],
    });
    if (Object.keys(calendariPosseduti).length == 0) {
      res.status(409).json({
        error: "There is no calendar with that ID or you are not part of it",
      });
      return;
    }

    const eventi = await Evento.find({
      $and: [
        { partecipanti: userId },
        { IDCalendario: new ObjectId(IDCalendario) },
      ],
    });
    if (Object.keys(eventi).length == 0) {
      res.status(409).json({
        error: "There are no events with that userId and IDCalendario",
      });
      return;
    }
    res.status(200).json({
      eventi,
    });
    return;
  } catch (e) {
    console.error(e);
    res.status(501).json({ error: "Generic error" });
    throw new Error(e).message;
  }
}
```

API - getEventi

Immagine PNG API - getEventi

5.9 : Crea Account

Questa API (descritta in parte in ??apd:ResourcesModelUtenteAutenticato)) viene utilizzata per creare un "account", ovvero un oggetto di tipo "UtenteAutenticato", struttura dati già mostrata in precendenza in APD3, all'interno del database MongoDB, la prima volta che un utente accede autenticandosi nel sito PlanIt. I parametri, che devono essere presenti per non ottenere il messaggio di errore "400 - Parameter missing", sono: "userId", "email" e "username" dell'utente autenticato che si vuole creare.
I controlli che vengono effettuati sono gli stessi già descritti nelle altre API, si evidenzia la presenza della funzione "find()" che ha lo scopo di trovare se esistono nel database già utenti con l' "userId" o l' "email" inseriti in input; se ne esistessero otteniamo come errore "409 - There is already one user with that id or email".
Alla fine della funzione, viene usata la funzione "create()" per andare a creare nel database un oggetto "UtenteAutenticato" con gli attributi ricevuti come parametri.

```
export async function creaUser(req, res) {
  await dbConnect();
  try {
    const { userId, email, username } = req.query;

    if (userId == null || email == null || username == null) {
      res.status(400).json({ error: "Parameter missing" });
      return;
    }

    const users = await UtenteAutenticato.find({
      $or: [{ userId: userId }, { email: email }],
    });
    if (Object.keys(users).length >= 1) {
      res
        .status(409)
        .json({ error: "There is alrady one user with that id or email" });
      return;
    }

    UtenteAutenticato.create(
      {
        userId: userId,
        email: email,
        username: username,
      },
      function (err, user) {
        if (err) {
          res.status(500).json({ error: "Not inserted" });
          return;
        }
      },
    );
    res.status(200).json({ success: "User inserted correctly" });
    return;
  } catch (e) {
    res.status(501).json({ error: "Generic error" });
    return;
  }
}
```

API - creaAccount

Immagine PNG API - creaAccount

5.10 : Modifica Account

L'applicativo PlanIt sfrutta questa API, già anticipata in parte in (apd:ResourcesModelUtenteAutenticato), per andare a modificare l'account, più precisamente l' "username", di un oggetto "UtenteAutenticato" già presente nel database. Per andare a modificare l'oggetto "UtenteAutenticato" vengono utilizzati i parametri che sono ricevuti in input, che non sono altro che gli attributi che formano un oggetto di tipo "UtenteAutenticato" (guardare APD3). Per andare ad identificare l'utente da modificare viene usato l' "userId" ricevuto, invece l' "username" è l'attributo che viene modificato nell'oggetto "UtenteAutenticato" già presente nel database. I controlli effettuati, all'interno di questa API, sono gli stessi già descritti per altre API; infatti si controlla che l' "userId" inserito corrisponda ad un utente esistente e non duplicato e che la funzione "updateMany()" esegua con nessun tipo di errore.

```
export async function modificaUser(req, res) {
  await dbConnect();
  try {
    const { userId, username } = req.query;

    if (userId == null || username == null) {
      res.status(400).json({ error: "Parameter missing" });
      return;
    }

    const users = await UtenteAutenticato.find({
      userId: userId,
    });
    if (Object.keys(users).length == 0) {
      res.status(409).json({ error: "There is no user with that userId" });
      return;
    } else if (Object.keys(users).length > 1) {
      res
        .status(409)
        .json({ error: "There are too many users with that userId" });
      return;
    }

    const put = await UtenteAutenticato.updateMany(
      { userId: userId },
      { $set: { username: username } },
    );

    if (put.modifiedCount == 1) {
      res.status(200).json({ success: "User updated correctly" });
    } else {
      res.status(500).json({ error: "Not edited" });
    }
  } catch (e) {
    console.error(e);
    res.status(501).json({ error: "Generic error" });
    throw new Error(e).message;
  }
}
```

API - modificaAccount

Immagine PNG API - modificaAccount

5.11 : Get Dati Account - userId

Questa API viene utilizzata per ottenere in output un oggetto di tipo "UtenteAutenticato" che abbia l' "userId" uguale al parametro "userId" inserito in input. Nel caso in cui l' "userId" non fosse ricevuto in input o non corrispondesse a nessun "UtenteAutenticato" salvato nel database o corrispondesse a più utenti, riceviamo un messaggio di errore diverso per ciascuno dei casi, già descritti più volte precedentemente nelle altre APIs.

```
export async function getUser(req, res) {
  await dbConnect();
  try {
    const { userId } = req.query;

    if (userId == null) {
      res.status(400).json({ error: "Parameter missing" });
      return;
    }

    const users = await UtenteAutenticato.find({
      userId: userId,
    });
    if (Object.keys(users).length == 0) {
      res.status(409).json({ error: "There is no user with that userId" });
      return;
    } else if (Object.keys(users).length > 1) {
      res
        .status(409)
        .json({ error: "There are too many users with that userId" });
      return;
    }

    res.status(200).json({
      userId: users[0].userId,
      email: users[0].email,
      username: users[0].username,
    });
    return;
  } catch (e) {
    console.error(e);
    res.status(501).json({ error: "Generic error" });
    throw new Error(e).message;
  }
}
```

API - Get Dati Account_userId

Immagine PNG API - Get Dati Account_userId

5.12 : Get Dati Account - email

Questa API viene utilizzata per ottenere in output un oggetto di tipo "UtenteAutenticato" che abbia l' "email" uguale al parametro "email" inserito in input. Questa API fa la stessa procedura e ha lo stesso scopo dell'API descritta in APD5.11, ovvero quella precedente (Get Dati Account - userId), l'unica cosa che cambia è che viene utilizzata l'email per identificare univocamente un "UtenteAutenticato". Con entrambe le APIs, ad ogni modo, si dovrebbe ottenere lo stesso risultato in quanto sia l' "userId" che l' "email" dovrebbero identificare univocamente un "UtenteAutenticato".

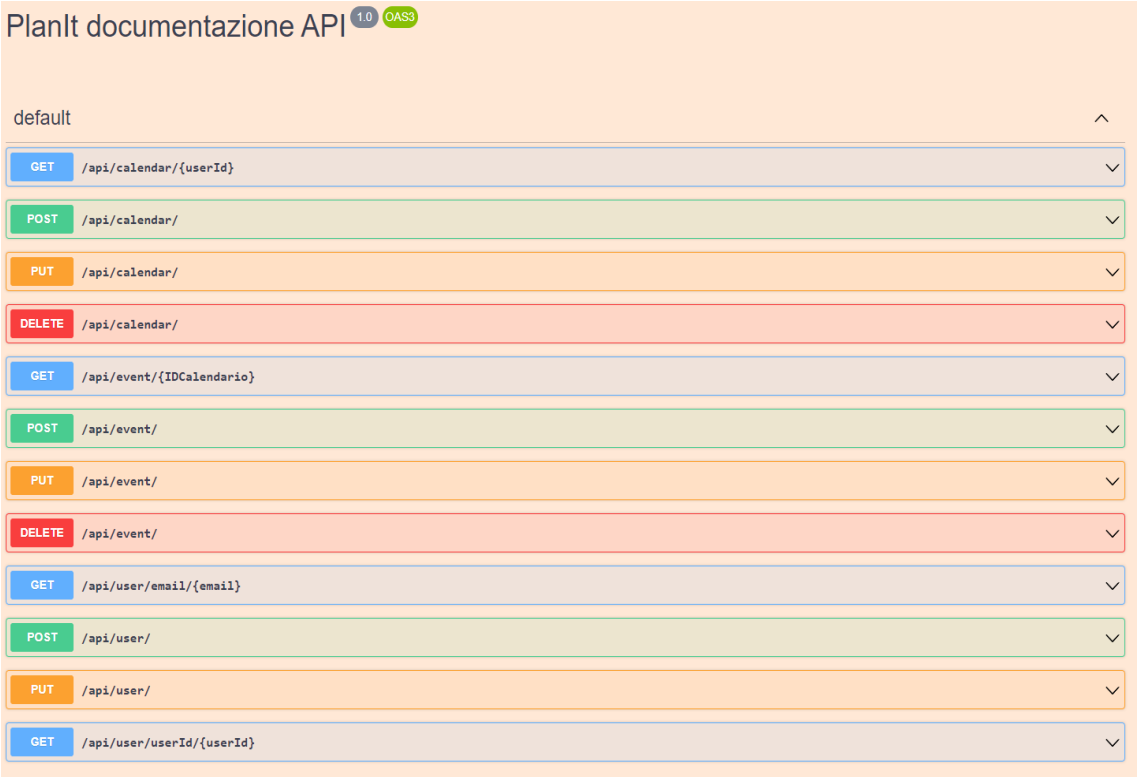
```
export async function getUser(req, res) {
  await dbConnect();
  try {
    const { email } = req.query;

    if (
      email == null ||
      !/^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/.test(email)
    ) {
      res.status(400).json({ error: "Parameter missing or malformed" });
      return;
    }

    const emailDB = await UtenteAutenticato.find({
      email: email,
    });

    if (Object.keys(emailDB).length == 0) {
      res.status(409).json({ error: "There is no user with that email" });
      return;
    } else if (Object.keys(emailDB).length > 1) {
      res
        .status(409)
        .json({ error: "There are too many users with that email" });
      return;
    }

    res.status(200).json({
      userId: emailDB[0].userId,
      email: emailDB[0].email,
      username: emailDB[0].username,
    });
    return;
  } catch (e) {
    console.error(e);
    res.status(501).json({ error: "Generic error" });
    throw new Error(e).message;
  }
}
```

API - Get Dati Account_email

Immagine PNG API - Get Dati Account_email

3 API documentation

Le API fornite dall'applicazione PlanIt e già descritte nella sezione precedente APD5 sono state documentate utilizzando il modulo di NextJS chiamato "next-swagger-doc". In questo modo la documentazione relativa alle API è direttamente disponibile a chiunque veda il codice sorgente.
Per poter generare l'endpoint dedicato alla presentazione delle API abbiamo utilizzato "swagger-ui-react" in quanto crea una pagina web secondo la documentazione scritta. In particolare, di seguito mostriamo la pagina web relativa alla documentazione che presenta le 12 API, di cui ce ne sono sia di tipo GET che POST che DELETE, essenziali per la gestione del prototipo del sito PlanIt che è stato sviluppato.
Nella pagina della documentazione fornita, abbiamo reso disponibile anche la possibilità di utilizzare degli esempi preparati "ad hoc" per poter vedere come funzionano tutte le API da noi implementate.
L'endpoint da invocare per raggiungere la seguente documentazione e': http://localhost:3000/ApiDoc oppure http://127.0.0.1:3000/ApiDoc Inoltre, la documentazione è possibile visionarla non solo, grazie al codice sorgente da noi sviluppato, ma anche direttamente al seguente link dove abbiamo effettuato l'hosting della documentazione: https://api.plan-it.it/apidoc. Infatti abbiamo usato il sito "netlify" per l'hosting di ciò che abbiamo implementato per questo deliverable D4, in modo tale che la documentazione e anche il FrontEnd, il cui link verrà passato successivamente, possano essere sempre raggiungibili anche senza dover avere per forza il codice sorgente da eseguire.

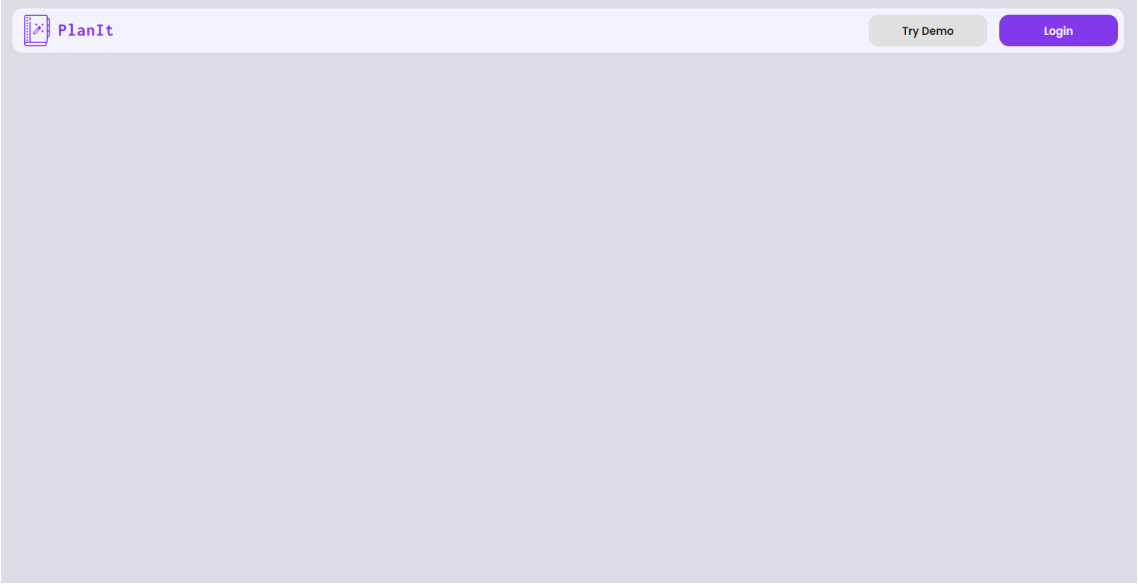


Documentazione

Immagine PNG Documentazione

4 FrontEnd Implementation

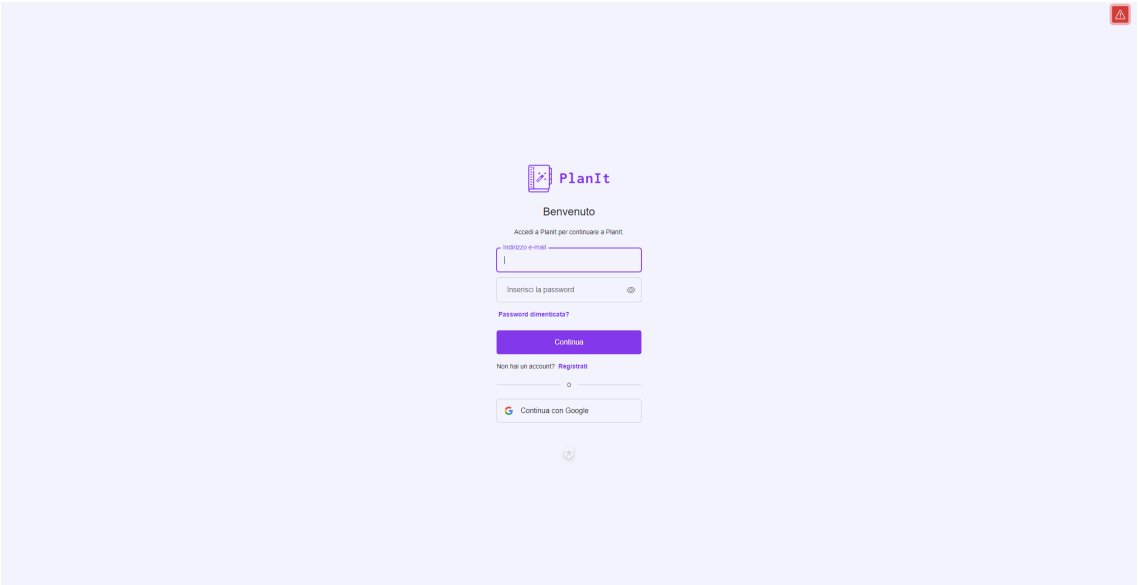
Il FrontEnd (visualizzabile al link https://frontend.plan-it.it) implementato in questo prototipo del sito PlanIt, da noi sviluppato, fornisce le funzionalità di visualizzazione, creazione, modifica e cancellazione di calendari ed eventi all'interno dell'applicativo; gli eventi che si possono gestire sono sia singoli, ovvero che in fase di creazione o modifica vengono definiti solo per una specifica data, oppure eventi ripetuti, ovvero che vengono definiti per più giornate in fase di creazione o modifica. Inoltre, dopo il primo accesso, è presente anche l'automatica creazione di un account per l'utente che ha fatto la registrazione. L'username attribuito a tale utente autenticato, che inizialmente corrisponde alla sua email, può anche esser modificato in fase di registrazione.
Prima di cominciare con la descrizione più specifica di ciascuna schermata, partiamo con il dire che le pagine che costituiscono questo prototipo del sito PlanIt, sono: "Homepage", "schermata login", "schermata registrazione", "schermata recupero password", "schermata Calendario" e "schermata Eventi".
In particolare partiamo da come appare il sito, quando ci si accede. La prima pagina che si visualizza, è l'Homepage, visualizzabile nell'immagine sottostante. Dall' "Homepage", l'utente può solo premere il tasto "Login", che lo indirizzerà alla pagina di autenticazione fornita da Auth0. Infatti tutta la fase di registrazione, login e anche recupero password sono gestite completamente da questo sistema esterno di autenticazione che integriamo nel sito. Il bottone "Try Demo", presente nell' "Homepage", non ha nessuna funzionalità, in quanto in questo prototipo del sito, si è sviluppato solo il sito per gli utenti che si autenticano.



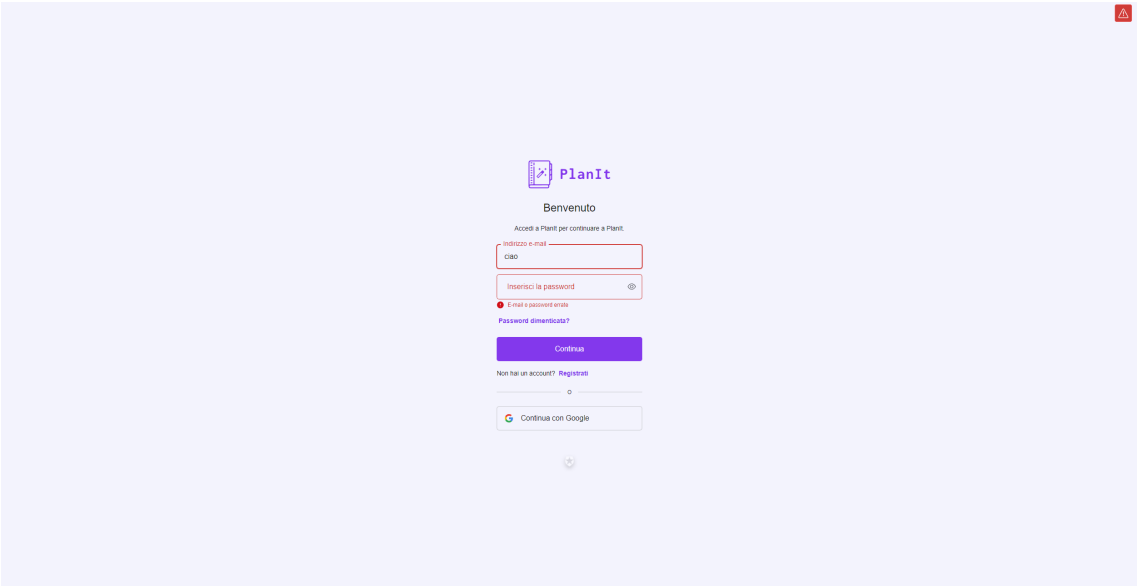
FrontEnd - Homepage

Quindi, dopo aver premuto il bottone "login", si è indirizzati nella pagina "Login", che si può osservare qua sotto. In questa pagina, se l'utente ha già un proprio account PlanIt, può accedere usando le sue credenziali, oppure l'autenticazione può esser fatta usando anche un account Google appartenente all'utente che vuole accedere al sito PlanIt come utente autenticato. Ovviamente, nel caso si provasse a fare l'accesso mediante credenziali, ma si inserissero delle credenziali errate, il sistema notifica l'utente dell'errore che si è verificato.

Immagine PNG FrontEnd - Homepage



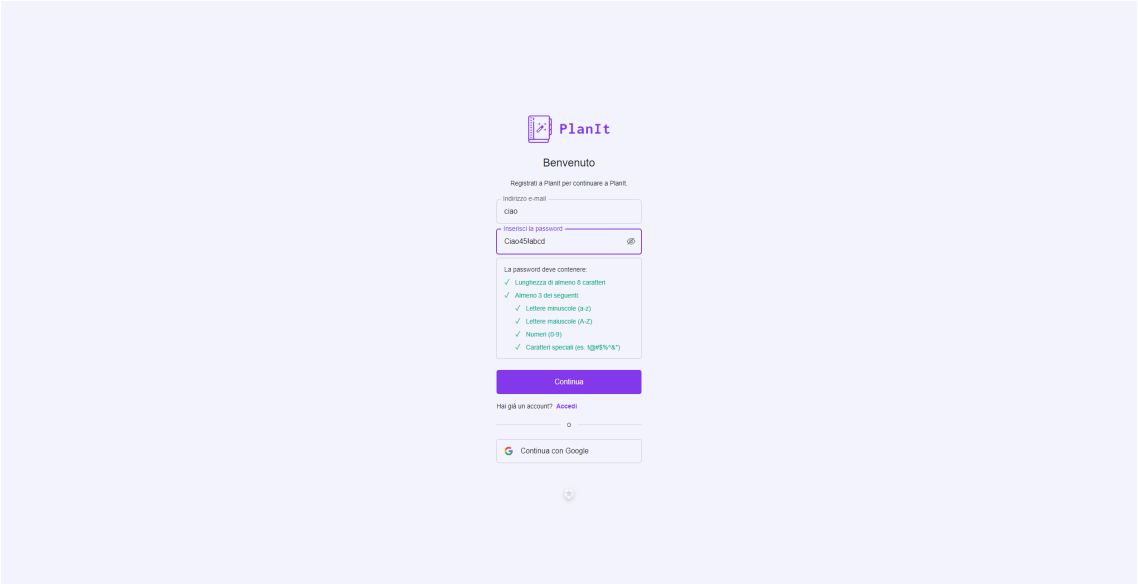
FrontEnd - Login



FrontEnd - Login errato

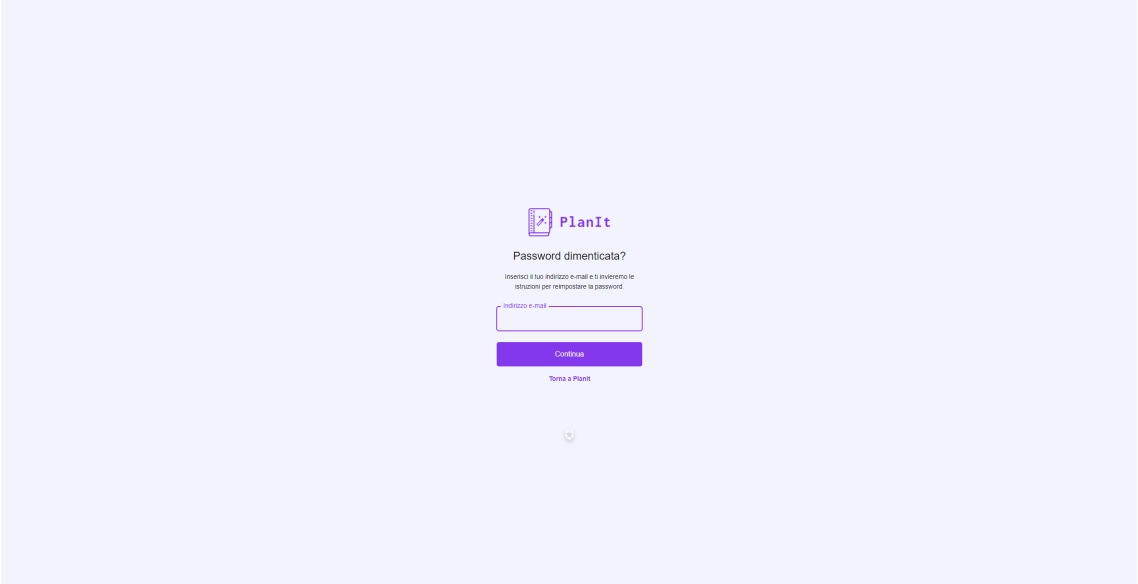
Quando l'utente non ha un proprio account PlanIt e non vuole accedere mediante Google, può indirizzarsi alla pagina di registrazione, mediante la scritta "Registrati". Da questa schermata, l'utente può tornare indietro alla pagina di "Login" nel caso in cui avesse sbagliato, può ancora "continuare con Google" oppure, ovviamente, inserire le proprie credenziali con cui si vuole registrare. Ovviamente, le credenziali da inserire devono rispettare delle restrizioni che sono imposte e che vengono mostrate all'utente.
Una volta inserite delle credenziali che soddisfano tali restrizioni, l'utente può terminare la procedura di registrazione premendo il tasto "Continua", che lo indirizza alla pagina "Calendario". Contemporaneamente, l'utente riceve anche una email per la validazione dell'email inserita per la registrazione; questa validazione è davvero importante, in quanto l'email verificata è quella che si dovrà utilizzare per il reset password.

Immagine PNG FrontEnd - Login
Immagine PNG FrontEnd - Login errato



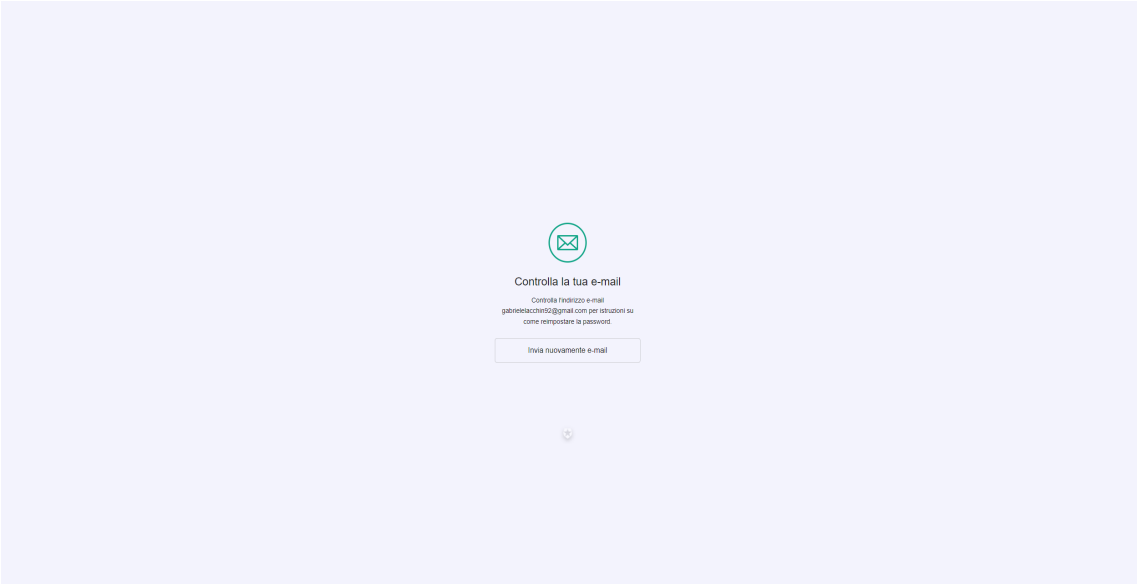
FrontEnd - Registrazione valida, segue le restrizioni che vengono imposte e mostrate

Nel caso in cui l'utente si fosse dimenticato la propria password, può schiacciare sulla scritta "Password dimenticata" dalla pagina "Login", che indirizza l'utente alla pagina d'inserimento dell'email con cui si è fatta la registrazione al sito. Il sistema automaticamente invierà un'email di recupero e reset password all'indirizzo email indicato.
E' da sottolineare che questa procedura di recupero password, è disponibile solo per gli utenti che hanno deciso di registrarsi al sito mediante credenziali e non con "Google".
Infine, dopo l'inserimento dell'email e averla confermata, l'utente visualizza la schermata di conferma che l'email di recupero password è stata inviata; dunque, bisogna controllare la propria casella di posta. Nel caso in cui non fosse stata ricevuta, si può richiedere l'invio di nuovo di tale email.



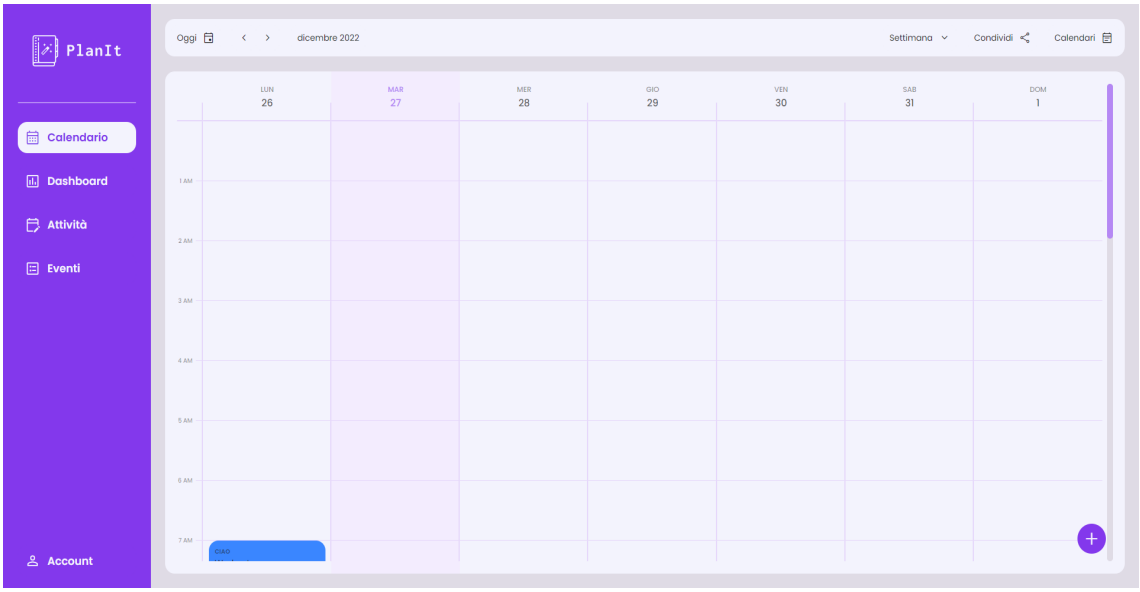
FrontEnd - Recupero password

Immagine PNG FrontEnd - Registrazione valida, segue le restrizioni che vengono imposte e mostrate
Immagine PNG FrontEnd - Recupero password



FrontEnd - Controlla email

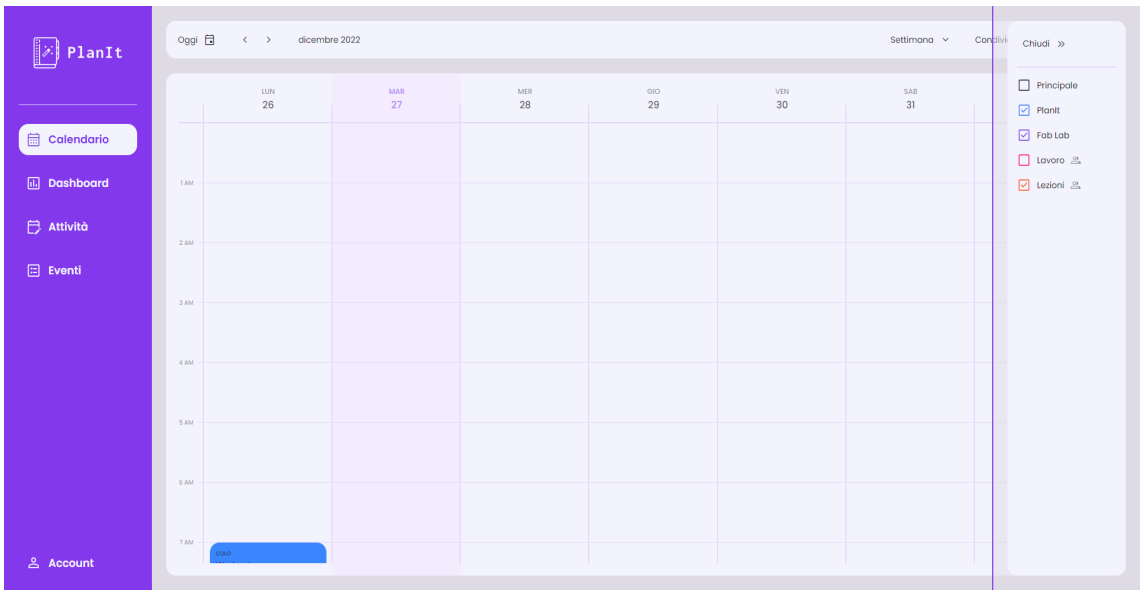
Dopo aver fatto l'autenticazione, l'utente autenticato visualizza la pagina "Calendario", da cui può passare a tutte le altre pagine (ricordiamo che oltre a "Calendario" è stata sviluppata solo la pagina "Eventi" in questo protitipo), può creare un evento e calendario, possibilità visualizzabile dopo aver premuto il "+" in baso a destra, può visualizzare la lista di calendari personali, e anche quelli condivisi, dal bottone "Calendari", che apre una side-bar in cui si possono selezionare i calendari da visualizzare nella schermata "Calendario, e infine può spostarsi temporalmente nel tempo grazie ai bottono "<", con cui si va indietro, e ">", con cui si va avanti, e tornare alla data corrente mediante il bottone "Oggi". Adesso andiamo un po' più nello specifico delle cose visualizzabili e che si possono fare in questa pagina.



FrontEnd - schermata "Calendario"

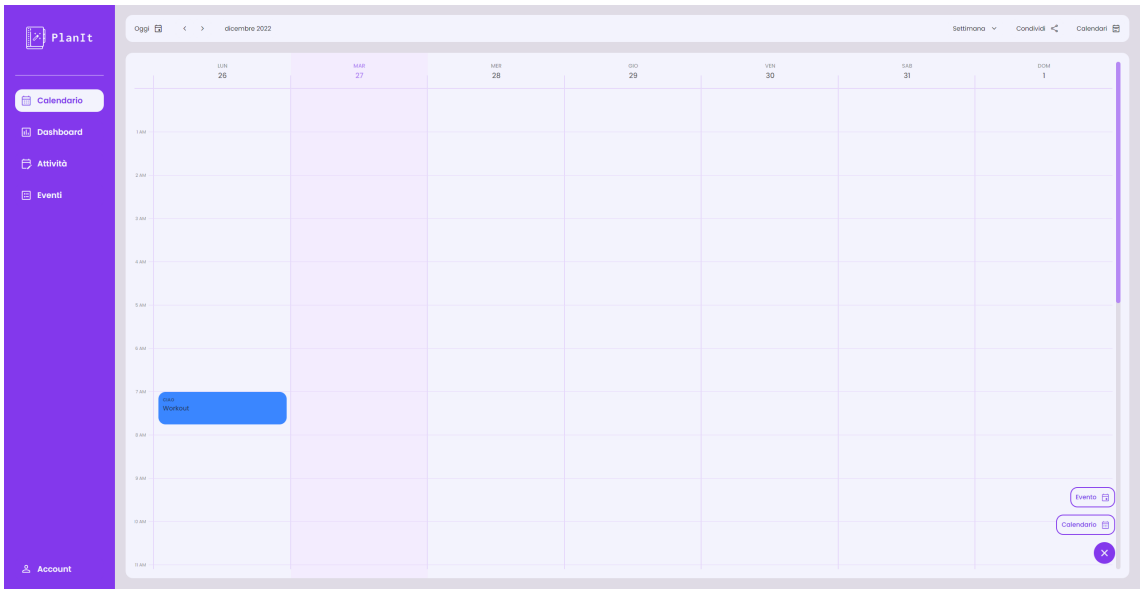
Dunque, premendo il bottone "Calendari" in alto a destra, appare questa side-bar, in cui è presente una lista di tutti i calendari appartenenti all'utente. I calendari condivisi sono distinguibili grazie allo stereotipo di "persone" che si trova a fianco di tali calendari. La funzionalità di questa side-bar, oltre che far visualizzare la lista di calendari personali, è anche quella di permettere all'utente di filtrare, spuntandoli, quale calendari visualizzare, e quindi i rispettivi eventi, nel nella pagina "Calendario".

Immagine PNG FrontEnd - Controlla email
Immagine PNG FrontEnd - schermata "Calendario"



FrontEnd - side-bar Calendari

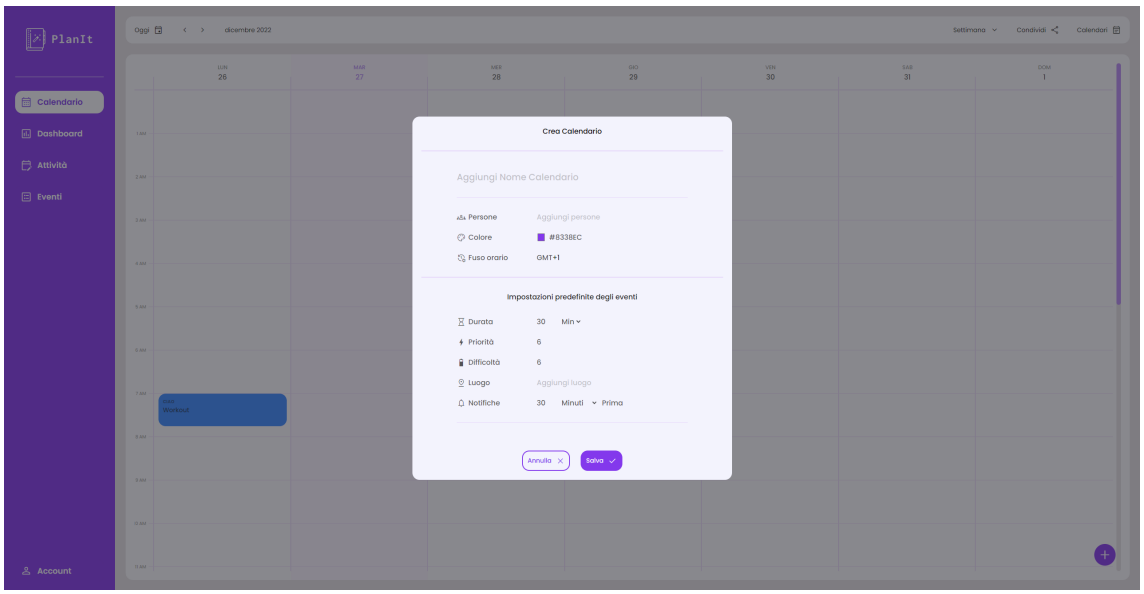
Dopo, premendo il tasto "+" in basso a destra è possibile aprire un fab, dove appare la scritta "Evento" e "Calendario". Grazie a questi due scelte, è possibile poter creare un evento, specifichiamo che in questa schermata si può creare solo quello singolo, a differenza della schermata "Eventi" in cui si possono creare anche quelli ripetuti, e/o un calendario. Dunque, premendo una delle tue scelte, appare un pop-up di creazione calendario o evento.



FrontEnd - Scelta creazione

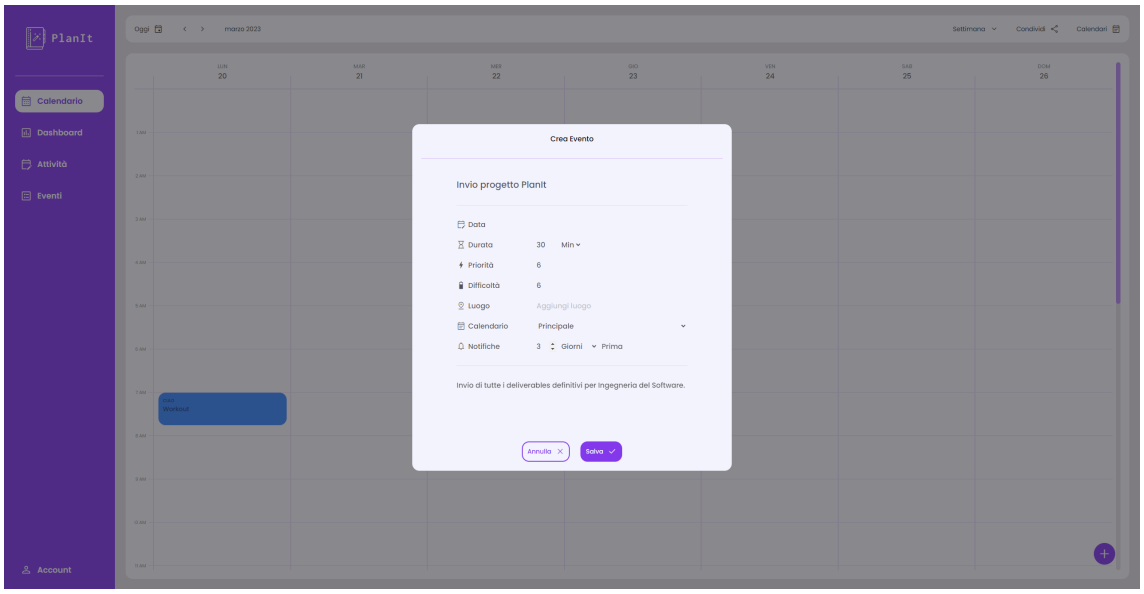
Il pop-up sottostante è quello di "Crea Calendario". Tutti i campi visualizzabili sono compilabili, eccetto per "Luogo", in quanto non abbiamo sviluppato la possibilità dell'utente d'indicare il luogo dove si trova l'evento. Dopo aver compilato a piacere i vari campi, ricordiamo che il nome del calendario è obbligatorio per poter creare un calendario, l'utente ha la possibilità di salvarlo, oppure anche di annullare l'azione.

Immagine PNG schermata "Calendario" - side-bar Calendari
Immagine PNG schermata "Calendario" - Scelta creazione



FrontEnd - Crea Calendario

Invece, il pop-up sottostante è quello che si ottiene quando l'utente vuole creare un evento. Tutti i campi sono compilabili, eccetto per il campo "Luogo", come già detto per "Crea Calendario". Dopo aver compilato a piacere i vari campi, ricordiamo che il titolo dell'evento e la sua dati devono essere compilati obbligatoriamente per poter creare un evento, l'utente ha la possibilità di salvarlo, oppure anche di annullare l'azione.

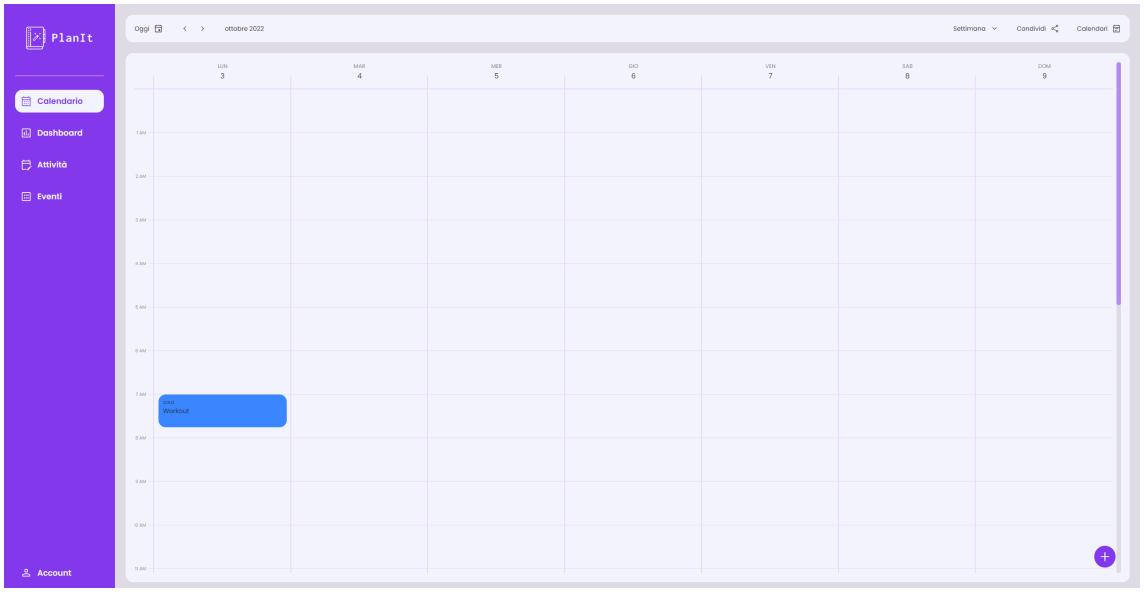


FrontEnd - Crea Evento

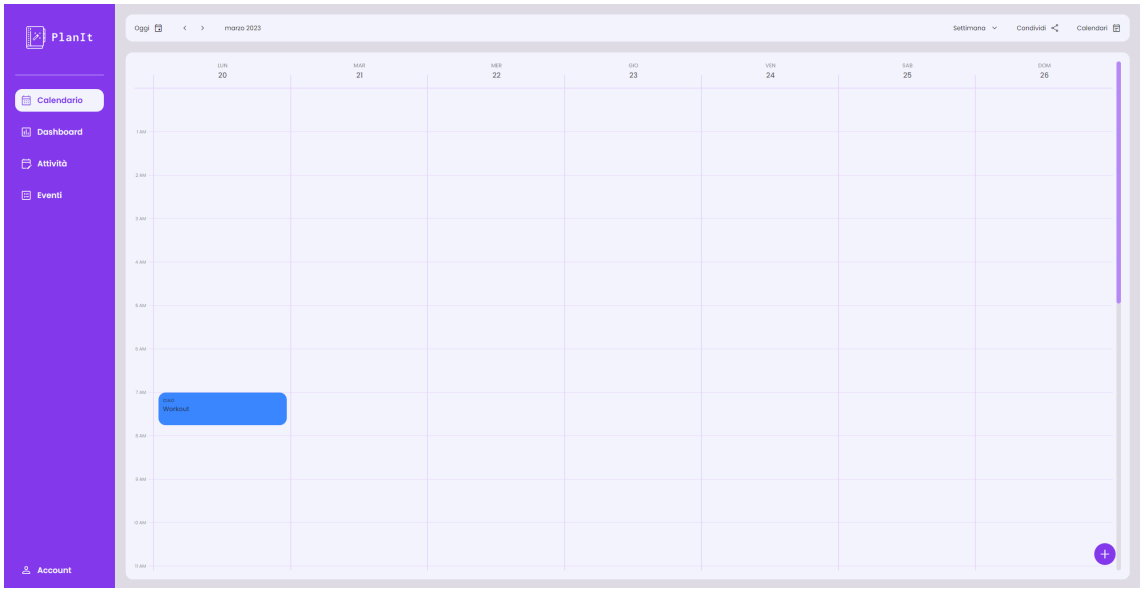
Come già detto in precedenza, mediante i bottoni "<", ">" e "Oggi" l'utente può rispettivamente:

- andare indietro di settimane nel calendario, si guardi la prima immagine sottostante a questa lista;
- andare avanti di settimane nel calendario, si guardi la seconda immagine sottostante a questa lista;
- tornare alla settimana presente.

Immagine PNG schermata "Calendario" - Crea Calendario
Immagine PNG schermata "Calendario" - Crea Evento



FrontEnd - Andare indietro di settimane



FrontEnd - Andare avanti di settimane

Dalla schermata "Calendario", come citato precedentemente, l'utente autenticato può passare alla schermata "Eventi". Da quest'ultima schermata, oltre a poter creare nuovi eventi singoli e nuovi calendari, mediante gli stessi pop-up ottenibili nella pagina "Calendario", è possibile anche modificare sia gli eventi che i calendari salvati e creare e modificare eventi ripetuti. Adesso andiamo più nello specifico di ciò che può visualizzare e fare l'utente da questa schermata.

Nel caso in cui l'utente volesse creare nuovi calendari e nuovi eventi singoli, premendo i tasti in alto a sinistra "Crea Calendario" e "Crea Evento", può visualizzare nella parte destra della schermata i rispettivi form di creazione, identici a quelli che si possono ottenere nella pagina "Calendario".

Dopo, quando l'utente vuole creare un evento ripetuto, ovvero un evento che definiamo presente su più giorni, già in fase di creazione dell'evento, basta che usi il bottone "Crea Evento Ripetuto" per far apparire il form di creazione "Crea Evento Ripetuto". In questo form, tutti i form sono compilabili, eccetto per "Luogo". Ricordiamo che i campi "nome" e almeno una data su cui cade tale evento devono essere sempre presenti, per far andare a buon fine l'esito di creazione dell'evento ripetuto. L'utente, mediante le "Impostazioni avanzate" può andare a specificare maggiormente

Immagine PNG schermata "Calendario" - Andare indietro di settimane
Immagine PNG schermata "Calendario" - Andare avanti di settimane

quando cadono tali eventi e in quali orari. Con l'attributo "Durata", l'utente indica per quale periodo di tempo, questo evento ripetuto vale e quindi deve essere presente nel calendario.

L'utente, inoltre, in questa schermata "Eventi" visualizza la lista di calendari. Avendo la possibilità di esplorarli, ovvero visualizzare quali eventi ne fanno parte, può selezionare uno dei suoi eventi per aprire il form di "Modifica Evento". Questo form è uguale a quello di "Crea Evento", ovvero ha gli stessi campi, eccetto che l'utente ha la possibilità di eliminare tale evento selezionato mediante il bottone "Elimina".

Infine, selezionando uno dei calendari della lista, si apre il form di "Modifica Calendario", che è identico a quello di "Crea Evento", eccetto per la presenza del bottono "Elimina", che l'utente può utilizzare per eliminare un proprio calendario.

5 GitHub Repository and Deployment Info

Riguardo alla descrizione del repo Git, che nel nostro caso si chiama "Codice" (repo dove è presente tutto il codice da noi sviluppato per l'implementazione del sito PlanIt, sia BackEnd che FrontEnd), molto è già descritto in APD1. Ma in questa capitolo andiamo a descrivere altre cartelle presenti nel nostro git "Codice" (link: https://github.com/Life-planner/Codice). Infatti in questa cartella git è presente anche la cartella "_tests_" dove sono presenti tutti i test effettuati per ciascuna API che è stata implementata, divisi a seconda di quale "resource" (guardare Extract Diagram APD4.1) riguardano. Inoltre, in ciascuno file della cartella api, è presente la documentazione relativa alle funzioni presenti in tale file. E' stato fatto in questo modo, ovvero la presenza della documentazione nel relativo file, in quando abbiamo usato una libreria apposita di Swagger per NextJS che richiede tale procedura per andare a formare la documentazione. Infatti, grazie "next-swagger-doc", basta fornire la documentazione in ciascun file e automaticamente viene compilato un file "swagger.json" dove è scritta tutta la documentazione in formato ".json". Questo file è contenuto nella cartella "public", dove sono presenti anche tutti gli "asset" del progetto.
In questo capitolo presentiamo anche le informazioni riguardo al deployment e al link per eseguire il prototipo PlanIt da noi implementato. Per l'hosting del nostro sito abbiamo deciso di utilizzare il sito di hosting netlify.
Questi sono i link da noi fatti da cui si può usufruire di ciò che abbiamo implementato:

- main branch: https://plan-it.it;
- api branch: https://api.plan-it.it;
- frontend branch: https://frontend.plan-it.it;
- documentazione: https://api.plan-it.it/apidoc;
- testing, prima opzione: https://plan-it.it/test-report.html
- testing, seconda opzione: https://api.plan-it.it/coverage/index.html

6 Testing

In questo capitolo sono descritti i casi di testing effettuati sulle API da noi sviluppate. La parte di testing è fondamentale per poter individuare eventuali problemi sulle API sviluppate e accertarsi che tutti funzioni secondo quello voluto.
Prima di andare con la descrizione dei vari casi di testing effettuati, si specifica che per comodità siamo andati a suddividere i testing in base a quella resource riguardassero, quindi utente, calendario ed evento e in base alla tipologia di API da testare, ovvero POST, PUT e GET. Inoltre, prima di ciascuna procedura di testing per ogni API, sono presenti delle procedure necessarie affinché si possa effettuare con successo un certo testing per una API. Ad esempio, prima di tutto, nel nella funzione "beforeAll()" di ciascun file, si sono creati degli oggetti che ci servivano per poter testare con successo le APIs, come "userId" duplicati, "userId" (parametro essenziale, che deve essere sempre presente per far funzionare un' API) corretti, calendari da cui ottenere l' "IDCalendario" (essenziale, come vedremo, per il POST e il PUT dell'evento e per il PUT del calendario), eventi da cui ottenere l' "IDEvento" (essenziale, come vedremo, per il PUT evento), calendari pieni e vuoti di eventi (essenziale per l'api getEventi di un calendario), ecc.. E dopo, alla fine della procedura di testing per una certa API, abbiamo sempre inserito nell' "afterAll()" la pulizia di tutto il database MongoDB, in modo tale che gli altri testing non avessero conflitti con quello fatto in altri file di testing.
Nelle seguenti pagine, si può ottenere un resoconto dei testing effettuati: https://plan-it.it/test-report.html, https://api.plan-it.it/coverage/index.html. Nel secondo link, è presente il documento html, che è stato mostrato anche in aula, che ci è stato consigliato aggiungere. Aprendo quest'ultimo link, si può notare che non tutte le linee sono state coperte per ciascun file esaminato, perché? I motivi sono molto semplici; infatti tutte le linee non coperte riguardano o la funzione "handle()", presente in ciascun file delle APIs, che è una funzione che viene invocata quando si vuole invocare un'API, ma per i casi di test abbiamo deciso di invocare direttamente le APIs, invece che gli handler, in quanto questi, nei test, davano dei problemi, o pezzi di codice in cui sono presenti il controllo ad errori da noi non verificabili. Per l'appunto, in tutti i file, le parti di codice in cui sono presenti il "throw" del codice "500" e "501" non sono da noi verificabili, in quanto sono errori derivanti da errori generici che spuntano durante l'esecuzione dell'API, oppure dalla sconnessione improvvisa dal server MongoDB; sono tutte condizioni da noi non replicabili nei test. Quindi, queste sono le ragioni per cui, nel secondo link riportato, e nella foto sottostante, la percentuale coperta di "lines" per ciascuna cartella è, quasi in tutti i casi, diversa dal 100%.

Resoconto testing, secondo link dato

Abbiamo deciso anche di produrre un secondo resoconto, reperibile a questo link, in quanto quest'ultimo ci sembrava migliore per mostrare tutti i test cases effettuati, con la loro relativa breve descrizione per far intendere a cosa ci riferissimo. Per questo motivo, useremo questo resoconto più chiaro nelle prossime immagini.

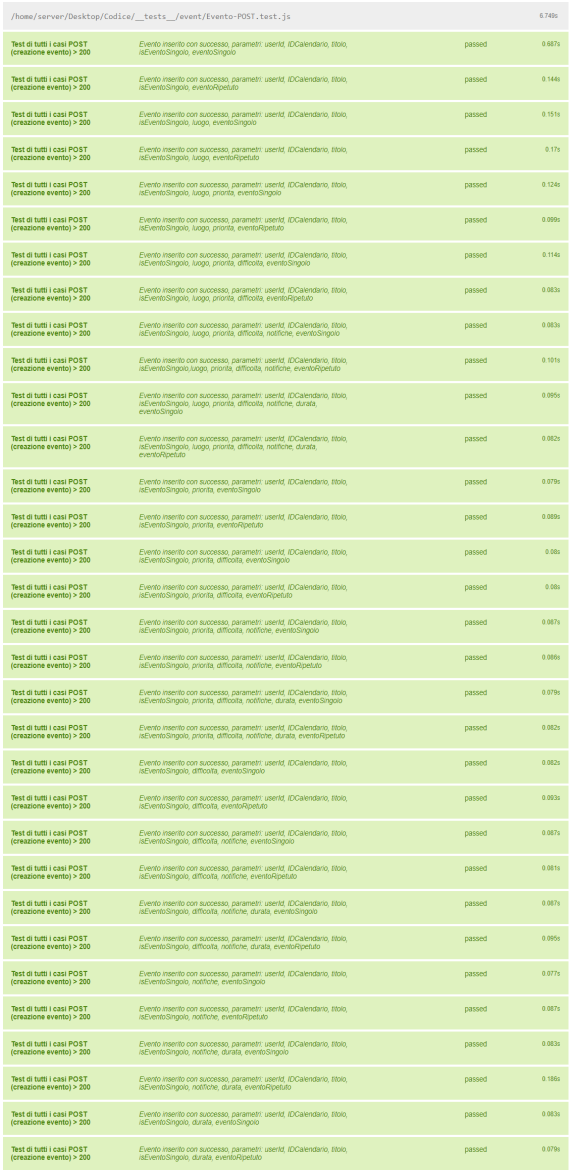
qua posso inserire anche lo screen di tutta la pagina html intera, divisa in parti, oppure a fine del capitolo

TS1 : **Evento - POST**

Come già detto precedentemente in APD5.5, i parametri che sono obbligatori per non ottenere l'errore "400 - IDCalendario or titolo or evento details missing" da questa API sono "userId", "IDCalendario", "titolo", "isEventoSingolo" e "eventoSingolo" o "eventoRipetuto" ("eventoSingolo" quando "isEventoSingolo" = true, "eventoRipetuto" se "isEventoSingolo" = false). Per questo motivo i primi due test effettuati, sono stati fatti mandando in input solo questi parametri scritti con il loro giusto formato, in modo tale di non ottenere l'errore "409 - Wrong format for (parameter with wrong format)" e ottenere il buon fine dall'esecuzione dell'API con l'arrivo del codice "200" e il messaggio "Event inserted correctly", come è avvenuto infatti. Sono stati citati "due test" iniziali, in quanto il primo è stato fatto con "eventoSingolo" != null e "isEventoSingolo" = true, e il secondo invece con "eventoRipetuto"

Immagine PNG Resoconto testing, secondo link dato

!= null e "isEventoSingolo" = false. Questi due test sono i test con il minor numero possibile di parametri che si possono avere per avere un possibile buon esito dall'esecuzione delle API, sempre se anche gli altri controlli citati in APD5.5 vengono superati. A partire da questi due test, sono stati fatti tutti gli altri test, in cui mano a mano si sono inseriti anche gli altri parametri opzionali, con formati corretti, che possono essere presenti come input per l'API "creaEvento" (APD5.5) nelle loro varie combinazioni osservando se si ottenesse sempre il successo ("200 - event inserted correctly"). I parametri opzionali inseriti nelle varie combinazioni sono "luogo", "priorita", "difficolta", "notifiche" e "durata". Si è deciso di fare tutti i possibili casi con le varie combinazioni, anche sapendo da altre API passate, che il risultato sarebbe stato corretto, per poter osservare che tutto andasse come aspettato con tutte le varie possibilità e anche poter identificare più velocemente le cause di errore, nel caso in cui i test non fossero andati a buon fine. Infatti in tutti i primi test, ciò che volevamo ottenere era il buon esito dell'esecuzione dell' API "creaEvento" con "200 - Event inserted correctly".



Casi 200, POST evento

Dal test numero 32 al 38 ("Test di tutti i casi POST (creazione evento) > 400", "Manca uno o piu parametri – IDCalendario, parametri presenti: userId, titolo, isEventoSingolo, eventoSingolo"), abbiamo formulato test con lo scopo di ottenere errori del tipo "400 - Mancano uno o più parametri – parametro mancante, parametri presenti: ..." per poter osservare se i controlli presenti all'interno dell'API fossero corretti e identificassero che tutti i parametri

Immagine PNG Casi 200, POST Evento

necessari affinché l'API possa eseguire con successo, fossero presenti. Come si può notare in questa pagina, tutti questi casi sono stati passati con successo dal codice dell'API "creaEvento" per tutti i possibili parametri che possono obbligatori, ovvero: "userId", "titolo", "isEventoSingolo", "eventoSingolo" o "eventoRipetuto".

Test di tutti i casi POST (creazione evento) > 400	*Manca uno o piu parametri -- IDCalendario, parametri presenti: userId, titolo, isEventoSingolo, eventoSingolo*	passed	0.029s
Test di tutti i casi POST (creazione evento) > 400	*Manca uno o piu parametri -- userId, parametri presenti: IDCalendario, titolo, isEventoSingolo, eventoSingolo*	passed	0.03s
Test di tutti i casi POST (creazione evento) > 400	*Manca uno o piu parametri -- titolo, parametri presenti: userId, IDCalendario, isEventoSingolo, eventoSingolo*	passed	0.029s
Test di tutti i casi POST (creazione evento) > 400	*Manca uno o piu parametri -- isEventoSingolo, parametri presenti: userId, IDCalendario, eventoSingolo*	passed	0.028s
Test di tutti i casi POST (creazione evento) > 400	*Manca uno o piu parametri -- eventoSingolo, parametri presenti; userId, IDCalendario, isEventoSingolo*	passed	0.031s
Test di tutti i casi POST (creazione evento) > 400	*Manca uno o piu parametri -- eventoRipetuto, parametri presenti; userId, IDCalendario, isEventoSingolo*	passed	0.03s

Casi 400 "Missing parameter", POST Evento

Dal test case numero 39, si è voluto controllare che i controlli riguardo al formato dei parametri inseriti funzionassero. Infatti nel caso in cui, uno dei parametri inseriti avesse un formato scorretto, si deve ottenere l'errore "400 - Wrong format for (parameter)". Questi controlli riguardo al formato scorretto sono fatti per i seguenti parametri: "luogo", "priorita", "difficolta", "notifiche", "durata", "eventoSingolo" ed "eventoRipetuto". Come si può notare dalla lista di test effettuati, sono stati fatti più test anche sullo stesso parametro sempre ottenendo questo errore, per poter osservare se l'API identificasse sempre l'errore con i vari casi con cui un parametro può assumere un valore erroneo.

Immagine PNG Casi 400 "Missing parameter", POST Evento

Test di tutti i casi POST (creazione evento) > 400	*Formato parametro luogo non corretto*	passed	0.071s
Test di tutti i casi POST (creazione evento) > 400	*Formato parametro luogo non corretto*	passed	0.078s
Test di tutti i casi POST (creazione evento) > 400	*Formato parametro luogo non corretto*	passed	0.072s
Test di tutti i casi POST (creazione evento) > 400	*Formato parametro luogo non corretto*	passed	0.075s
Test di tutti i casi POST (creazione evento) > 400	*Formato parametro priorita non corretto*	passed	0.084s
Test di tutti i casi POST (creazione evento) > 400	*Formato parametro difficolta non corretto*	passed	0.076s
Test di tutti i casi POST (creazione evento) > 400	*Formato parametro notifiche non corretto*	passed	0.074s
Test di tutti i casi POST (creazione evento) > 400	*Formato parametro notifiche non corretto*	passed	0.085s
Test di tutti i casi POST (creazione evento) > 400	*Formato parametro notifiche non corretto*	passed	0.071s
Test di tutti i casi POST (creazione evento) > 400	*Formato parametro durata non corretto*	passed	0.079s
Test di tutti i casi POST (creazione evento) > 400	*Formato parametro eventoSingolo non corretto*	passed	0.07s
Test di tutti i casi POST (creazione evento) > 400	*Formato parametro eventoRipetuto non corretto*	passed	0.086s

Casi 400 "Wrong format for (parameter)", POST Evento

Invece, dal test case numero 51, si è controllato che i casi in cui si inserisse un "userId" scorretto, ovvero che o non esistesse nel database MongoDB o fosse un duplicato, fossero individuati e fosse mandato il rispettivo messaggio di errore. Infatti, come si può notare dal codice presente nella cartella "_tests_" della repo "Codice" (cartella "tests"), prima dell'esecuzione di tutti i vari test per "creaEvento" è stato creato un "userId" corretto ("utenteTestEventoPOST"), che è stato usato in tutti i vari casi in cui non volevamo ottenere un errore ottenuto da questo parametro, e un duplicato ("utenteTestEventoPOSTDuplicato"). Quest'ultimo è stato utilizzato come per parametro erroneo, per vedere se l'API lo individuasse e inviasse l'errore "409 - There are too many users with that userId", come è successo. Invece, è stato usato l' "userId" "UtenteNonEsiste", ma andava bene un qualsiasi altro "userId" che non fosse salvato nel database, per vedere se il controllo riguardo se l' "userId" esistesse o meno funzionasse e in questo caso ottenessimo l'errore "409 - There is no user with that userId", come è successo.

Test di tutti i casi POST (creazione evento) > 409	*Ci sono più di un utente con l'userId dato*	passed	0.051s
Test di tutti i casi POST (creazione evento) > 409	*Utente non esistente*	passed	0.054s
Test di tutti i casi POST (creazione evento) > 409	*IDcalendario non esistente o userId non possiede tale IDCalendario*	passed	0.07s

Casi 409, POST Evento

Immagine PNG Casi 400 "Wrong format for (parameter)", POST Evento
Immagine PNG Casi 409, POST Evento

TS2 : **Evento - PUT**

Questa API, come già anticipato in APD5.6, ha bisogno di tutti gli attributi che formano la struttura dati "Evento" come parametri per iniziare la sua procedura. Nel caso in cui mancasse uno dei seguente attributi, ovvero "IDEvento" ("id" dell'evento che si sta modificando), "userId", "titolo", "isEventoSingolo", "luogo", "priorita", "difficolta", "notifiche", "data", "durata", "eventoSingolo" e "eventoRipetuto", si ottiene l'errore "400 - Parameter missing". Quindi, nei primi due test effettuati, che hanno l'obiettivo di ottenere il codice di successo "200" con messaggio "Event edited correctly", si è mandato in input gli stessi parametri, con l'unica differenza che nel primo abbiamo messo "isEventoSingolo" = true, dunque l'evento è un evento singolo, invece nel secondo "isEventoSingolo" = false, dunque l'evento è un evento ripetuto. In quanto per ottenere "200", questi sono gli unici due casi distinti, abbiamo fatto questi, e abbiamo ottenuto, come sperato, "200" con messaggio "Event edited correctly", come si può notare nella foto sottostante o in questa pagina.

In seguito, abbiamo fatto tredici test che avevano lo scopo di ottenere il codice "400" con il messaggio "Parameter missing"; infatti in questi tredici test, siamo andati a togliere in ciascun test, un diverso parametro in modo tale da osservare se l'API individuasse correttamente la mancanza di ciascun parametro. In questa pagina, si può notare che i test sono stati effettuati con successo e che l'API ha notato la mancanza del parametro da noi appositamente tolto, inviando in output il codice "400" con il messaggio di errore "Parameter missing".

inserire qua i casi 200

Dal test numero 15, siamo andati a fare degli input appositi, che avevano lo scopo di ottenere il codice "400" con il messaggio di errore "Wrong format for (parameter)"; infatti gli attributi "luogo", "priorita", "difficolta", "notifiche", "durata", "eventoSingolo" ed "eventoRipetuto", possono essere inviati con dei formati sbagliati, ovvero possono avere dei valori o sintassi sbagliata. Dunque, in questi sette test, che si possono guardare dalla foto sottostante o da questa pagina, siamo andati a inviare in input un parametro, scelto una alla volta tra gli attributi sopra citati, con formato erroneo. Come era aspettato, l'API si è comportata correttamente, inviandoci in output il codice "400" con il messaggio di errore "Wrog format for (parameter)".

inserire qua i casi 400, parameter missing

In due test, siamo andati a testare che con l'invio di un "userId" sbagliato, ovvero o non esistente o duplicato, l'API lo notasse e inviasse un codice di errore "409" con messaggio "There is no user with that userId" oppure "There are too many users with that userId" rispettivamente. Come si può notare, l'API si è comportata correttamente inviandoci gli output sopra citati. Sottolineiamo che questi controlli fatti sull' "userId" sono gli stessi presenti in TS1, quindi questi test non sono altro che una conferma che questi controlli funzionano correttamente.

inserire qua i casi 409, wrong format

Nei restanti test, siamo andati a testare i casi in cui l' "IDCalendario" o l' "IDEvento" fossero sbagliati, ovvero o non esistenti o non appartenenti a l' "userId" inviato in input. Nel primo test che si può leggere nella foto sottostante, abbiamo inviato un "IDCalendario" che non facesse parte della lista di calendari di quel "userId" indicato, e come aspettato, abbiamo ricevuto il codice "409" con il messaggio di errore "There is no calendar with that ID or you do not own the calendar". Nell'ultimo test, abbiamo inviato un "IDEvento" che non apparteneva alla lista di eventi di un "userId" e abbiamo ricevuto, come aspettato, il codice "409" con il messaggio "You do not own the event". Dunque, tutti i test effettuati hanno avuto successo anche per questa API.

foto 409 userId sbagliati

mettere gli ultimi casi 409

TS3 : **Evento - DELETE**

Questa API ha bisogno (vedere anche APD5.7), dell' "IDEvento" dell'evento che si vuole eliminare con l' "userId" dell'utente che ha tale evento. Dunque, nel primo test, in cui volevamo ottenere il codice "200" con messaggio di successo "Event deleted correctly", abbiamo inserito questi parametri scritti correttamente, che quindi potessero superare anche gli altri controlli presenti nelle API. Come previsto, da questo test abbiamo ottenuto il codice e messaggio sopra citati.

inserire foto 200 unico caso

Nei seguenti casi, siamo andati a controllare che nel caso in cui, non inserissimo uno dei parametri sopra citati, l'API lo notasse e ci inviasse il codice "400" con messaggio di errore "Parameter missing", cosa che è avvenuta sia nel caso in cui mancava "userId" sia nel caso in cui mancava "IDEvento".

mettere i due casi 400

In seguito, abbiamo fatto, come in TS1, i due test in cui inseriamo un "userId" scorretto, ovvero o duplicato o non esiste, aspettandoci il codice "409" con i messaggi "There are too many users with that userId" o "There is no user with that userId" rispettivamente. E, come avvenuto in TS1, i due test sono avvenuti con successo, ottenendo i codici e messaggi sopra

citati.
L'ultimo caso testato è quello in cui si inserisce un "IDEvento" che non esiste tra gli "id" degli eventi appartenenti all' "userId" dato. In questo caso, prevediamo di ottenere il codice "409" con il messaggio di errore "You do not own the event", cosa che è avvenuta; quindi, anche questo test è andato a buon fine.

mettere foto due casi userID scorretto

inserire qua ultimo caso 409

TS4 : **Calendario - POST**

Come già anticipato in APD5.1, lo scopo di questa API è creare e salvare un calendario nel database MongoDB e come già scritto in APD5.1, i parametri minimi necessari affinché questa API possa iniziare la sua procedura sono "userId", "nome" del calendario. Per questo motivo, il primo caso di testing effettuato prende in input solo questi due parametri, scritti in modo tale che siano corretti e che possano superare anche tutti gli altri controlli, e come aspettato, da tale testing otteniamo il codice di risposta "200" con il messaggio "Calendar inserted correctly". Gli altri testing che succedono, che hanno sempre l'obiettivo di ottenere come codice di risposta "200" e "Calendar inserted correctly", non sono altro che dei test in cui abbiamo aggiunto gli altri parametri opzionali, ovvero "fusoOrario", "colore" e "principale", in tutte le varie possibili combinazioni. Come si può notare in questa pagina, anche tutti questi test danno il risultato aspettato per tutti i possibili casi. Abbiamo deciso di fare tutte le possibili combinazioni, anche se non erano del tutto necessario, per fare un'analisi più approfondita, in modo tale che tutti i casi possibili fossero coperti e che nel caso ci fosse un errore, questo sarebbe individuato più velocemente.

Dal caso numero 9 al numero 12, i test che andiamo a fare hanno l'obiettivo di ottenere il codice "400" con il messaggio di errore del tipo "Name missing". Per questo motivo, in questi casi in combinazione mancano i parametri "userId" e "nome", parametri, che come detto precedentemente, devono essere sempre presenti affinché questa API possa procedere nell'esecuzione, e in tutti questi casi otteniamo come aspettato il codice "400" con il messaggio di errore "Name missing".

mettere casi 200

Dal caso numero 13 al caso numero 15, abbiamo fatto dei test, che hanno l'obiettivo di ottenere il codice "400" con il messaggio di errore del tipo "Wrong format for (parameter)". Infatti, i seguenti attributi potrebbero avere dei formati sbagliati, ovvero "colore" e "fusoOrario". Questi potrebbero avere dei formati sbagliati e per questo motivo devono essere controllati. Quindi, in questi casi, sono messi tutti i possibili formati erronei per questi parametri, in modo tale di poter osservare che l'API li individui, cosa che succede come aspettato e che si può osservare nella foto sottostante e da questa pagina.

mettere casi 400, i primi

Infine, negli ultimi casi di test si controlla che nel caso in cui si inserisse un "userId" non esistente o duplicato, questo venga individuato e sia inviato il codice "409" con i messaggi "There is no user with that userId" e "There is too many users with that userId" rispettivamente. Dati "userId" duplicati o non esistenti otteniamo questi output come aspettato e come ne abbiamo già parlato in TS1.

mettere wrong format 400

Invece, un altro caso in cui otteniamo il codice "409", è quello in cui l'utente prova a creare un calendario principale, quando ne ha già uno principale; infatti, ricordiamo che un utente può avere al massimo un calendario principale alla volta. Per questo motivo in questo test siamo andati a creare in primo luogo un calendario principale per un dato utente esiste, e questa procedura va a buon fine, e ripetiamo la stessa cosa una seconda volta. La seconda volta, come aspettato, otteniamo il codice "409" con messaggio di errore "There are too many primary calendars".

mettere i casi 409

TS5 : **Calendario - PUT**

Questa API, per iniziare la propria di procedura, ha bisogno come parametri in input di tutti gli attributi che formano una struttura dati "Calendario", ovvero "IDCalendario", "id" del calendario che stiamo per modificare, "nome", nome del calendario, "fusoOrario", "colore", "partecipanti", persone che partecipano agli eventi facenti parte di questo calendario, "principale", booleano che indica se il calendario modificato è quello principale. Quindi nel primo test, in cui avevamo l'obiettivo di ottenere il codice "200" con il messaggio di successo "Calendar modified correctly", abbiamo inserito come parametri tutti questi attributi, in modo che fossero corretti e superassero anche gli altri controlli, e come aspettato, abbiamo ottenuto come codice e messaggio quelli sopra citati.

inserire casi 200 fatti

Nei seguente diciannove test avevamo l'obiettivo di ottenere dall'API il codice "400" con il messaggio "Parameter missing"; infatti, siamo andati a togliere, uno alla volta, i parametri sopra citati, andando anche più nello specifico e togliendo una alla volta anche gli attribu-

ti che formavano gli oggetti "luogo" e "impostazioniPredefiniteEventi", per notare se l'API notasse queste mancanze e ci inviasse il codice e messaggio sopra citati. Come si può notare dalla foto sottostante e da questa pagina, così è stato!

mettere qua i casi 400

Nei seguenti otto test, siamo andati ad osservare se nel caso in cui inviassimo dei parametri, una alla volta, con un formato sbagliato, l'API lo notasse e ci inviasse in ritorno il codice "400" con messaggio "Wrong format for (parameter)". I parametri che possono avere un formato sbagliato sono: "fusoOrario", "colore", attributi "latitudine" e "longitudine" appartenenti all'oggetto "luogo", gli attributi "priorita", "difficolta", "durata", "tempoAnticNotifica" appartenenti all'oggetto "impostazioniPredefiniteEventi". Dunque, abbiamo fatto dei test in cui abbiamo inviato, una alla volta, i parametri sopra citati con dei formati sbagliati e, come aspettato, abbiamo ottenuto il codice "400" con messaggio "Wrong format for (parameter)"

inserire qui immagine dei casi 400

Nei seguenti due casi siamo andati a controllare che l'API notasse, come sempre, i casi in cui inviassimo degli "userId" errati, ovvero duplicati o non esistenti, e come negli altri casi abbiamo ottenuto il codice "409" con i messaggi corrispondenti, ovvero "There is no user with that userId" oppure "There are too many users with that userId" rispettivamente.

inserire qua i due casi di userId foto

Nell'ultimo caso di test, abbiamo inserito un "IDCalendario" errato, ovvero o non esistente o che non facesse parte della lista di calendari appartenenti a un dato "userId", aspettandoci di ottenere il codice "409" con il messaggio di errore "You do not own the calendar", come è successo. .

inserire foto ultimo caso di test

TS6 : **Calendario - DELETE**

Questa API ha bisogno dell' "IDCalendario" del calendario che si vuole eliminare con l' "userId" dell'utente che ha tale calendario. Dunque, nel primo test, in cui volevamo ottenere il codice "200" con messaggio di successo "Event deleted correctly", abbiamo inserito questi parametri scritti correttamente, che quindi potessero superare anche gli altri controlli presenti nelle API. Come previsto, da questo test abbiamo ottenuto il codice e messaggio sopra citati.

inserire foto 200 unico caso

Nei seguenti casi, siamo andati a controllare che nel caso in cui, non inserissimo uno dei parametri sopra citati, l'API lo notasse e ci inviasse il codice "400" con messaggio di errore "Parameter missing", cosa che è avvenuta sia nel caso in cui mancava "userId" sia nel caso in cui mancava "IDCalendario" e sia nel caso in cui mancano entrambi.

mettere i tre casi 400

In seguito, abbiamo fatto, come in TS1, i due test in cui inseriamo un "userId" scorretto, ovvero o duplicato o non esiste, aspettandoci il codice "409" con i messaggi "There are too many users with that userId" o "There is no user with that userId" rispettivamente. E, come avvenuto in TS1, i due test sono avvenuti con successo, ottenendo i codici e messaggi sopra citati.

mettere foto due casi userID scorretto

L'ultimo caso testato è quello in cui si inserisce un "IDCalendario" che non esiste tra gli "id" dei calendari appartenenti all' "userId" dato. In questo caso, prevediamo di ottenere il codice "409" con il messaggio di errore "You do not own the calendar", cosa che è avvenuta; quindi, anche questo test è andato a buon fine.

inserire ultimo caso

TS7 : **Utente - POST**

Come già anticipato in APD5.9, questa API, per iniziare il suo processo di creazione di un utente nel database, ha bisogno dei seguenti parametri: "userId", "email" ed "username". Dunque, nel primo test abbiamo messo questi parametri corretti in input, aspettandoci di ottenere il codice "200" con messaggio "User inserted correctly", cosa, che come si può notare dalla foto sottostante e da questa pagina, è avvenuta.

mettere foto 200 unico caso

Nei seguenti casi, siamo andati a testare i casi in cui mancassero uno o più parametri di quelli sopra citati, in tutte le possibili combinazioni, in modo tale che potessimo notare che l'API si comportasse in ogni caso nel modo corretto. Ci aspettavamo, in ciascun caso, il codice "400" con il messaggio di errore "Parameter missing", cosa che è successa per tutti questi sette casi.

mettere casi 400

Nell'ultimo caso controllato per questa API, abbiamo messo dei parametri che corrispondessero ad un utente già esistente, ovvero l' "userId" era già esistente nel database MongoDB. Nel caso in cui si provasse a creare un utente già esistente, questa API deve rispondere con codice "409" e messaggio "There is already one user with that id or email", cosa che è avvenuta come previsto.

mettere qua foto ultimo caso 409

TS8 : **Utente - PUT**

Questa API, come già scritto in APD5.10, ha lo scopo di modificare l' "username" di un utente identificato dal suo "userId". Per questo motivo, i parametri che devono essere presenti

quando si invoca questa API, sono i due attributi sopra citati; dunque, il primo test, in cui volevamo ottenere il codice "200" e il messaggio di successo "User update correctly", abbbiamo inserito questi due parametri in modo che superassero anche gli altri controllo presenti nell'API e come previsto, abbiamo ottenuto il codice e messaggio scritti precedentemente.
Nei seguenti casi, siamo andati a controllare che nel caso in cui, non inserissimo uno dei parametri sopra citati, l'API lo notasse e ci inviasse il codice "400" con messaggio di errore "Parameter missing", cosa che è avvenuta sia nel caso in cui mancava "userId" sia nel caso in cui mancava "username" e sia nel caso in cui mancano entrambi.
Negli ultimi due casi siamo andati a controllare che l'API notasse, come sempre, i casi in cui inviassimo degli "userId" errati, ovvero duplicati o non esistenti, e come negli altri casi abbiamo ottenuto il codice "409" con i messaggi corrispondenti, ovvero "There is no user with that userId" oppure "There are too many users with that userId" rispettivamente.

mettere foto questo caso 200

mettere i due casi 400, parameter missing

mettere foto ultimi 2 casi

TS9 : **Utente - DELETE**

Questa API, come già anticipato in **??**, va ad l'eliminare un utente dal database, dato il suo "userId". Dunque, nel primo test abbiamo inserito come parametro l' "userId" di un utente che sapevamo fosse esistente, volendo ottenere il codice "200" e il messaggio di successo "User deleted correctly", cosa che è successa; dunque questo test è stato passato.
Nel seguente caso, siamo andati a testare il caso in cui chiamassimo l'API senza l'aggiunta del parametro "userId". Come previsto, l'API ci ritorna il codice "400" con il messaggio di errore "Parameter missing".
In due test, siamo andati a testare che con l'invio di un "userId" sbagliato, ovvero o non esistente o duplicato, l'API lo notasse e inviasse un codice di errore "409" con messaggio "There is no user with that userId" oppure "There are too many users with that userId" rispettivamente. Come si può notare, l'API si è comportata correttamente inviandoci gli output sopra citati. Sottolineiamo che questi controlli fatti sull' "userId" sono gli stessi presenti in TS1, quindi questi test non sono altro che una conferma che questi controlli funzionano correttamente.

mettere foto 200 successo

mettere foto caso 400

foto 409 userId sbagliati

TS10 : **Utente - GET-Email**

Questa API ha bisogno di un' "email" per poter identificare un utente all'interno del database MongoDB, dunque l' "email" è l'unico parametro necessario. Nel primo test, abbiamo inserito un' "email" corretta aspettandoci di ottenere, come ritorno, il codice "200" e l'oggetto "UtenteAutenticato" dell'utente che ha tale "email", cosa che è avvenuta.
Il secondo test effettuato, è quello in cui non inseriamo nessun parametro, ovvero non mettiamo l' "email". Dunque l'API non può procedere, ma prevediamo invii il codice "400" con il messaggio di errore "Parameter missing", cosa che accede.
Gli ultimi due casi sono quelli in cui inseriamo un' "email" scorretta, ovvero non esistente o duplicata; dunque ci aspettiamo il codice "409" con il messaggio di errore "There is no user with that email" e "There are too many users with that email" rispettivamente. Come possiamo notare nella foto sottostante, o da questa pagina, anche questi test hanno esito positivo.

inserire caso 200

inserire caso 400 parameter missing

mettere foto ultimi 2 casi 409

TS11 : **Utente - GET-UserId**

Per questa API sono stati fatti gli test già presentati sopra in "Utente - GET-Email" (TS10), con l'unica differenza che al posto dell' "email", abbiamo l' "userId". Dunque non ripetiamo la descrizione dei casi, in quanto sono gli stessi, ma con l' "userId".

mettere foto di tutti i casi

TS12 : **Eventi - GET**

Come già detto in APD5.8, lo scopo di questa API è di ritornare l'array di eventi che fanno parte di un calendario. Dunque, i parametri necessari in quanto questa procedura possa andare, sono "IDCalendario" del calendario di cui si vogliono ottenere i suoi eventi e l' "userId" dell'utente che ha tale calendario. Per questo motivo, nel primo caso di test, in cui avevamo l'obiettivo di ottenere "200" con il ritorno dell'array di eventi, abbiamo inserito i parametri sopra citati corretti, in modo tale che superassero anche gli altri controlli. Come aspettato, abbiamo ottenuto il codice sopra citato e il test è andato a buon fine.
Nei successivi due test, siamo andati a togliere una alla volta i parametri "userId" e "IDCalendario" dai parametri in input, in modo tale da controllare che l'API notasse tale mancanza e ce la segnalasse con un codice di ritorno "400" e il messaggio "Parameter missing", cosa

inserire unico 200

che è successa in entrambi; cosa che si può notare dalla foto sottostante o da questa pagina. Come sempre, abbiamo messo i due casi in cui l' "userId" era scorretto, e anche questi sono stati gestiti come aspettato; non aggiungiamo altro, in quanto sono i soliti test cases ripetuti per tutte le API, quindi per approfondire guardare TS1.
Nel penultimo caso siamo andati a testare il caso in cui l' "IDCalendario" non esiste o non appartiene alla lista di calendari dell' "userId" indicato. Da tale caso, bisogna ottenere il codice "409" e il messaggio "There is no calendar with that ID or you are not part of it". Dunque abbiamo inserito un "IDCalendario" inesistente e abbiamo ottenuto, come previsto, il codice e messaggio sopra citati. Invece, nell'ultimo caso abbiamo inserito l' "IDCalendario" di un calendario vuoto, che quindi non ci può tornare alcuna lista di eventi, in quanto non l'ha. Dunque, abbiamo creato un calendario vuoto, ottenuto il suo "id" che abbiamo usato come parametro, invocato l'API, aspettandoci che ci ritornasse il codice "409" con il messaggio "There are no events with that userId and IDCalendario", come è successo.

mettere i due casi 400

inserire test cases userId scorretto

inserire foto penultimo caso

mettere foto ultimo caso

TS13 : **Calendari - GET**

Già anticipato in precedenza in APD5.4, lo scopo di questa API è ottenere tutti i calendari di un utente autenticato. Dunque, l'unico parametro che deve essere sempre presente, affinché l'API possa avere successo, se passa anche i successivi controlli, è l' "userId" dell'utente autenticato. Per questo motivo, sono stati effettuati solo due test, da cui si aspetta di ottenere successo con codice "200" e l'array di calendari, in cui il primo è stato effettuato passando l' "userId" di un utente che ha solo un calendario personale, invece il secondo che ha due calendari personali. Come si può notare dalla foto sottostante a da questa pagina, questi due test hanno avuto il risultato aspettato.
In seguito, abbiamo testato che l'API notasse il caso in cui non inserissimo l'unico parametro richiesto, ovvero "userId". Come aspettato, come si può notare dalla foto sottostante e anche da questa pagina, l'API ha dato la risposta aspettata, inviando come output il codice "400" con messaggio "Parameter missing".
Dopo si è passati a controllare che nel caso in cui se inserissimo il parametro "userId" incorretto, ovvero che non esiste nel database o è duplicato, l'API individua tale errore e restituisce il codice "409" con il relativo messaggio. Infatti, nel caso in cui abbiamo inserito l' "userId" di un utente non esistente, l'API ha individuato tale errore restituendo "409 - There is no user with that userId". Invece, nel secondo caso (abbiamo sempre creato un userId duplicato all'inizio della procedura di testing come fatto in TS2) con l'inserimento di un "userId" duplicato abbiamo ottenuto come ci si aspettava, l'errore "409 - There are too many users with that userId".

inserire qua i casi 200

inserire qua l'unico cosa 400

inserire qua i casi 409

7 Legenda Riferimenti

riferimenti

Application Implementation And Documentation	APD
Testing	TS