

FRIENDLY NOTE

**TITOLO DOCUMENTO
SVILUPPO APPLICAZIONE**

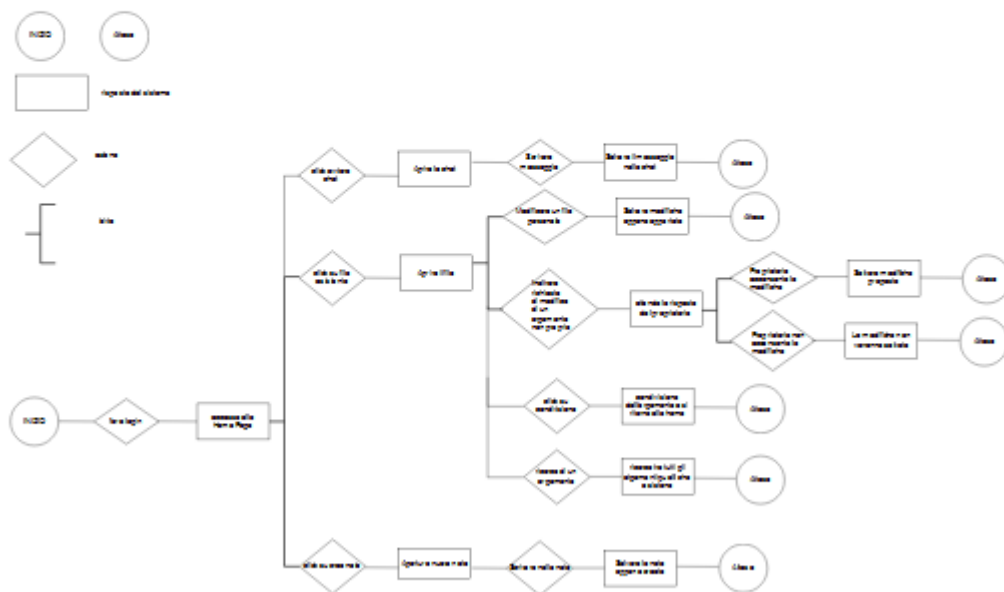
**INFORMAZIONI DEL DOCUMENTO
D1-DOCUMENTO DI SVILUPPO DELL'APPLICAZIONE**

INDICE

1. USERFLOWS
2. IMPLEMENTAZIONE E DOCUMENTAZIONE DELLE API
 - 2.1 STRUTTURA DEL BACKEND
 - 2.2 LINGUAGGI E DATABASE DEL PROGETTO
 - 2.2.1 LINGUAGGI USATI
 - 2.2.2 DATI NEL DATABASE
 - 2.3 SPECIFICA DELLE RISORSE E API
 - 2.3.1 DIAGRAMMA DELLE RISORSE
3. IMPLEMENTATIONE E DOCUMENTAZIONE DELLE API
4. DOCUMENTAZIONE API
5. IMPLEMENTAZIONE DEL FRONTEND
6. GITHUB REPOSITORY
7. TESTING

1. USERFLOWS

Lo userflow è un diagramma di flusso che mostra il percorso che l'utente compie nel sito per raggiungere un determinato obiettivo. In questo schema vengono rappresentate le azioni compiute dall'utente, dal momento in cui entra nel sito fino all'azione finale. Il diagramma di flusso che si considererà per friendly note è relativo all'utente.



2 IMPLEMENTAZIONE E DOCUMENTAZIONE DELLE API

2.1 STRUTTURA DEL BACKEND

All'interno della cartella controllers vengono definite le funzioni per gestire gli enclosure, note e gli utenti dei rispettivi file enclosureController.js, noteController.js e userController. Inoltre nel file tokenController.js, contiene le funzioni per generare e validare i token JWT. Ciascuno di questi file esporta un oggetto contenente le funzioni che possono essere richiamate da altri file. Nella cartella models vengono definiti tre diversi modelli di dati che possono essere utilizzati all'interno di una applicazione Node.js. Il primo file, enclosure.js, definisce uno schema per un modello di dati chiamato "Enclosure", che rappresenta un'area recintata e può contenere una serie di note. Il secondo file, note.js, definisce uno schema per un modello di dati chiamato "Note", che rappresenta una nota e può essere associato a un'area recintata tramite l'uso di un ObjectId. Il terzo file, token.js, definisce uno schema per un modello di dati chiamato "Token", che rappresenta un token di autenticazione e ha un tempo di scadenza. Quindi questi file forniscono uno schema per definire come i dati devono essere strutturati all'interno del database, consentendo all'applicazione di accedere e manipolare tali dati in modo coerente. Nella cartella router troveremo tutti i file JSON che si occupano di definire e esportare le routes associate ad un certo schema. La cartella test contiene tutti i file per i testing, mentre il file AuthHandler è importante per salvare il token durante il login, inoltre permette di salvare i dati relativi agli utenti durante l'accesso qualora servissero. All'interno di AuthHandler ci sono due funzioni principali getAuthcredential e setAuthcredential. Il primo permette di prelevare le informazioni dell'utente che ha fatto l'accesso tramite un token mentre il setAuthcredential permette di creare e salvare il token.

2.2 LINGUAGGI E BASI DI DATI DEL PROGETTO

2.2.1 LINGUAGGI USATI

I linguaggi che sono stati usati per realizzare questo progetto sono JSON ed in JSX (JavaScript XML), estensione di React che usa la sintassi di JavaScript ed è simile all'aspetto di HTML.

2.2.2 DATI NEL DATABASE

Durante la progettazione del sito abbiamo usato MongoDB, un database non relazionale orientato ai documenti. All'interno si possono trovare tre collection ovvero: user, enclosure e note.

```
_id: ObjectId('63eec8fb395a76a7abdefc7e')  
title: "Test Note"  
content: "Test Content"  
dateCreated: 2023-02-17T00:23:23.829+00:00  
__v: 0
```

Modello per le note

```
_id: ObjectId('63eec8fb395a76a7abdefc7d')  
name: "Test Enclosure"  
> notes: Array  
creator: ObjectId('546573742043726561746f72')  
dateCreated: 2023-02-17T00:23:23.827+00:00  
__v: 0
```

Modello per l'Enclosure

```
_id: ObjectId('63ee2b2e2f2347df0c1cd7eb')  
mail: "davide"  
password: "002"  
> EnclosurePossesed: Array  
__v: 0
```

Modello per l'user

2.3 SPECIFICA DELLE RISORSE E API

Le risorse principali per le API sono due Utente, Note ed Enclosure.

La risorsa Utente è costituita da:

- Email
- Password
- Id
- Token per l'utente

La risorsa Note è costituita da:

- Title
- Content
- Id
- Token per le note

La risorsa enclosure è costituita da:

- Nome
- Note
- Creator
- datecreated

API

<<resource>> newUser permette di creare un nuovo utente di registrarsi. Questa funzione viene svolta con una post e i parametri in ingresso sono e-mail e password.

<<resource>> login permette di accedere al sito tramite e-mail e password. Questa funzione viene svolta con una post.

<<resource>> deleteUser permette di eliminare un account, per far ciò è richiesto l'id dell'utente. Questa funzione viene svolta una delete.

<<resource>> getUserById permette tramite la richiesta get di poter ottenere le informazioni di un utente tramite il suo id.

<<resource>> newNote tramite la richiesta post vengono create nuove note. I dati richiesti sono titolo, contenuto e data di creazione.

<<resource>> getNoteById tramite la richiesta get vengono prelevate delle note tramite l'id di quest'ultime. Viene richiesto l'id.

<<resource>> getAllNoteByEnclosure tramite la richiesta get vengono prese tutte le note di un enclosure tramite l'id dell' enclosure.

<<resource>> deleteNote tramite la richiesta delete l'utente può decider quali note eliminare, dovrà essere inserito l'id.

<<resource>> updateNote tramite la richiesta post dovranno essere caricati nel server i vari aggiornamenti delle note. I parametri sono contenuto e id.

<<resource>> newEnclosure tramite la richiesta post crea una nuova enclosure all'interno del database. I parametri richiesti sono nome,note,creator e data di creazione.

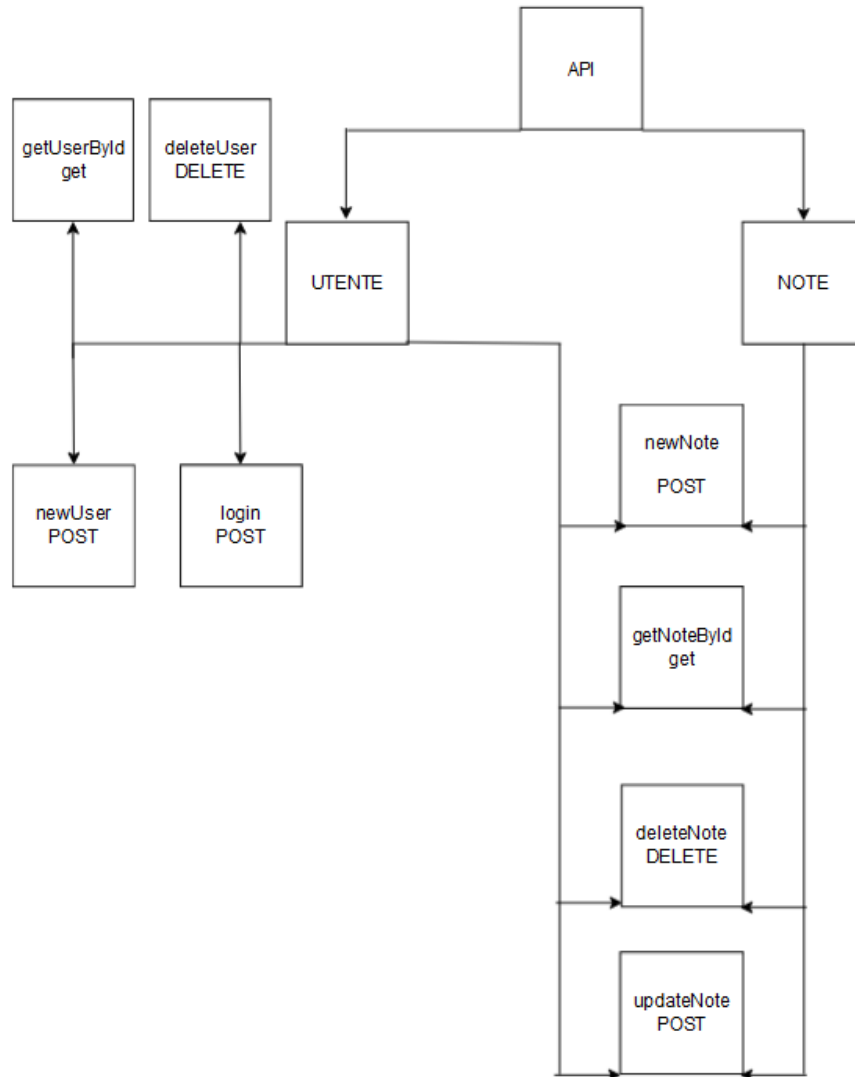
<<resource>> getEnclosureId tramite la richiesta get si preleva dal database l'enclosure con quell'id. Il parametro richiesto è l'id.

<<resource>> deleteEnclosure tramite la richiesta delete si elimina dal database l'enclosure con quell'id. Il parametro richiesto è l'id.

<<resource>> shareEnclosure tramite la richiesta put si condivide l'enclosure con altri utenti. Il parametro richiesto è l'id

<<resource>> getAllEnclosureByUser tramite la richiesta get si possono ottenere tutte le enclosure di un utente specifico. Il parametro richiesto è l'id.

2.3.1 DIAGRAMMA DELLE RISORSE

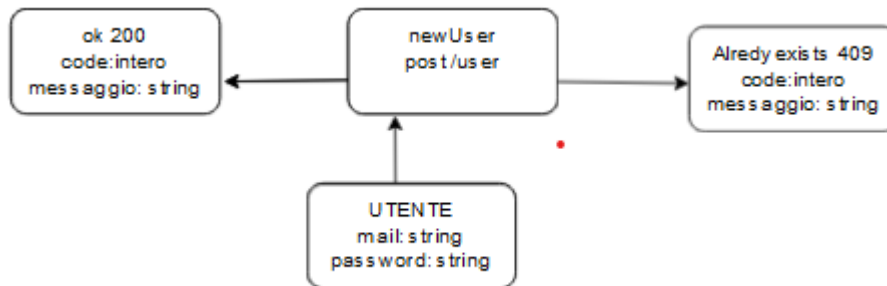


3. IMPLEMENTAZIONE E DOCUMENTAZIONE DELLE API

NewUser

Questa API ha indirizzo “/user” ed è caratterizzata dal metodo post che memorizza un nuovo utente all’interno del database. La request body è costituita da User che è costituito dagli attributi mail e password, entrambi di tipo string. Si possono avere due tipi di risposta a seconda del risultato ottenuto. Se la response body è data da code = 200 il nuovo utente è stato creato correttamente e restituisce lo user stesso

Se la response body è data da code = 409 vuol dire che l’utente già esiste e il messaggio user already exist



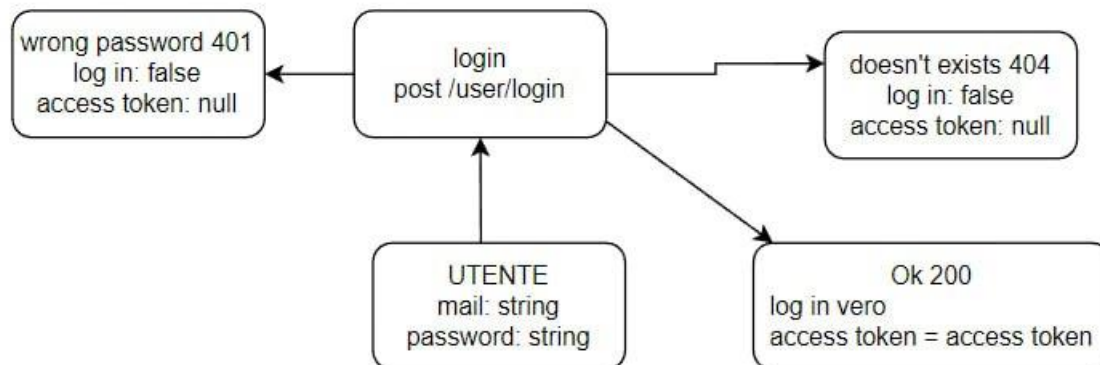
Login

Questa API ha indirizzo “/user/login” ed è caratterizzata dal metodo post e viene usata per far sì che l’utente possa accedere alla schermata home del proprio profilo. La request body della login è costituita da due campi email e password entrambi di tipo string. Si possono avere tre tipi di risposta a seconda del risultato ottenuto.

Se la response body è data da code = 401 la password inserita è sbagliata, otterremo un login = false e access token: null

Se la response body è data da code = 404 lo user non esiste, otterremo un login = false e access token: null

Se la response body è data da code = 200 lo user esiste, otterremo un login = true ed access token: access token

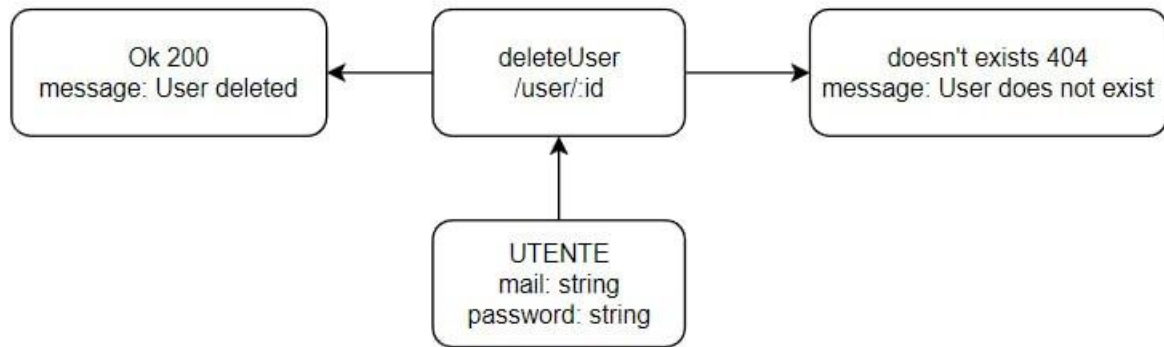


deleteUser

Questa API ha indirizzo “/user/:id” ed è caratterizzata dal metodo delete e viene usata per far sì che l’utente possa eliminare il proprio account. La request body della login è costituita dal campo id del profilo, di tipo ObjectId. Si possono avere 2 tipi di risposta a seconda del risultato ottenuto.

Se la response body è data da code = 200 l’utente viene eliminato correttamente e restituisce come messaggio: user deleted

Se la response body è data da code = 404 l’utente non esiste e restituisce come messaggio user doesn’t exist

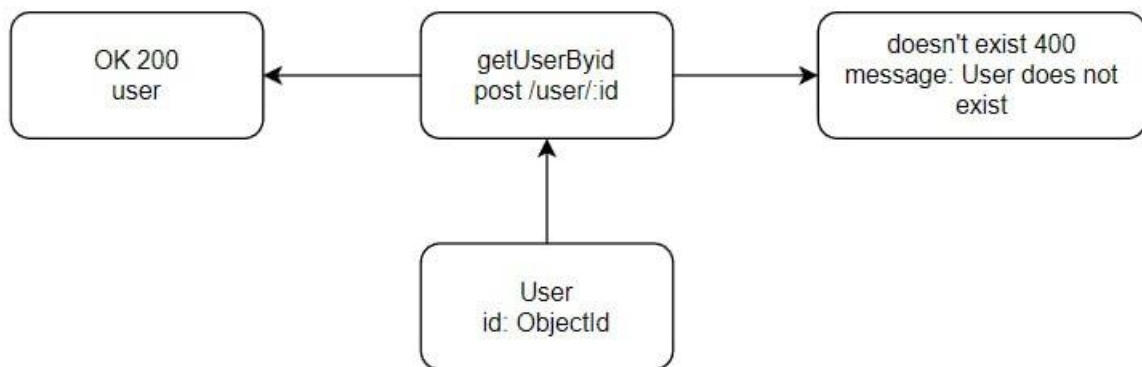


getUserById

Questa API ha indirizzo `/user/:id` ed è caratterizzata dal metodo `get` e viene usata per far sì che l'utente possa ottenere le informazioni di un utente tramite il suo id. La request body della `getUserById` è costituita dal campo `id` del profilo, di tipo `ObjectId`. Si possono avere due tipi di risposta a seconda del risultato ottenuto.

Se la response body è data da `code = 200` l'user esiste e restituisce come messaggio: `user`

Se la response body è data da `code = 400` l'utente non esiste e restituisce come messaggio `user doesn't exist`

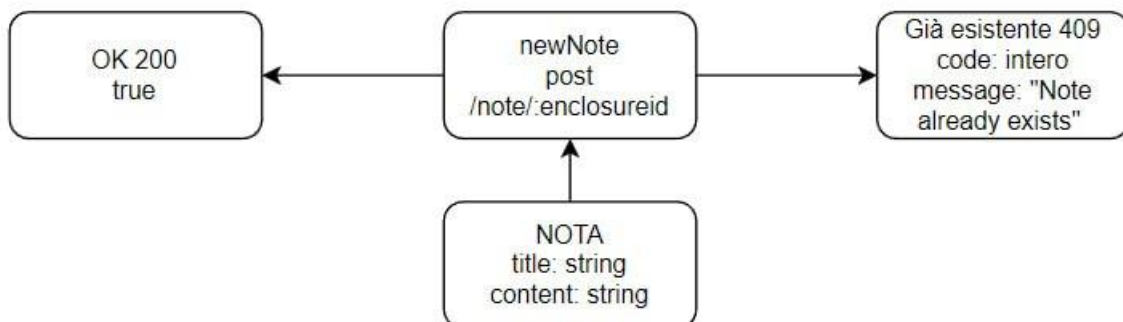


newNote

Questa API ha indirizzo `/note/:enclosureid` ed è caratterizzata dal metodo `post` e viene usata per far sì che l'utente possa creare una nuova nota. La request body di `newNote` è costituita dai campi `title`, `content` e `DateCreated`, di tipo `string` e `data`. Si possono avere due tipi di risposta a seconda del risultato ottenuto.

Se la response body è data da `code = 200` la nota esiste e restituisce come messaggio: `true`

Se la response body è data da `code = 409` la not già esiste e restituisce come messaggio `note alredy exist`

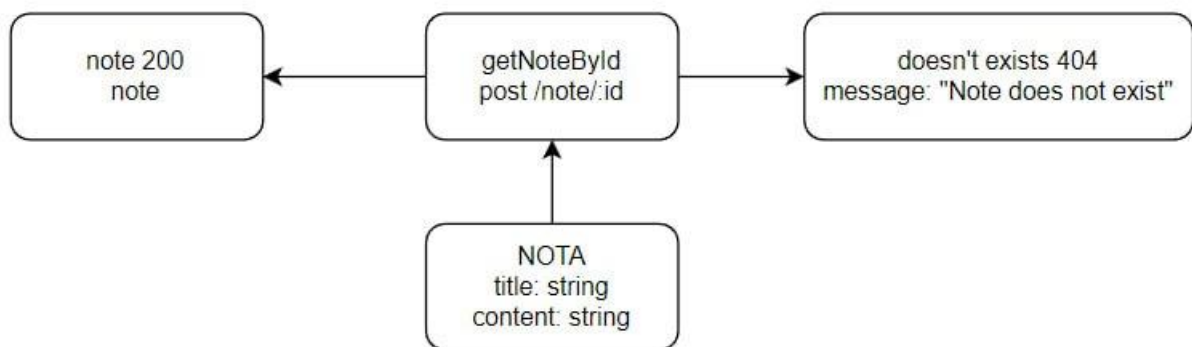


getNoteById

Questa API ha indirizzo “/note/:id” ed è caratterizzata dal metodo get e viene usata per far sì che l’utente possa prelevare delle note tramite l’id di quest’ultime. La request body di getNoteById è costituita dal campo id, di tipo ObjectId. Si possono avere due tipi di risposta a seconda del risultato ottenuto.

Se la response body è data da code = 200 restituisce la nota restituisce come messaggio: note

Se la response body è data da code = 404 la nota già esiste e restituisce come messaggio user doesn’t exist

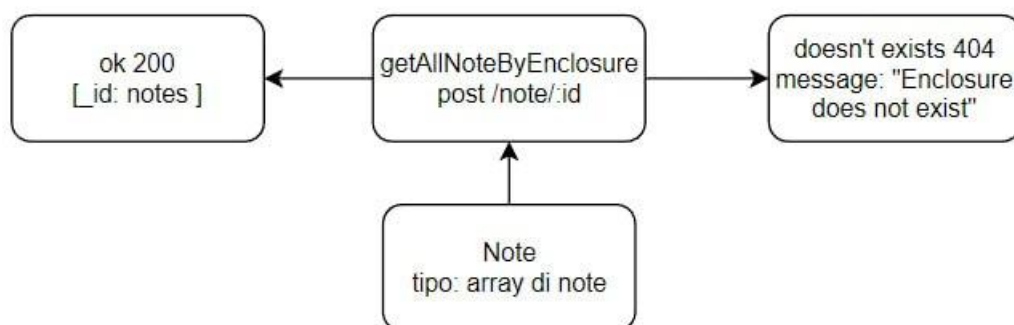


getAllNoteByEnclosure

Questa API ha indirizzo “/note/enclosure/:enclosure:id” ed è caratterizzata dal metodo get e viene usata per far sì che l’utente possa prelevare delle note tramite l’id di quest’ultime. La request body di getAllNoteByEnclosure è costituita dal campo id, di tipo ObjectId. Si possono avere due tipi di risposta a seconda del risultato ottenuto.

Se la response body è data da code = 200 restituisce tutte le note per quell’enclosure e restituisce come messaggio: [id:note]

Se la response body è data da code = 404 le note per quell’enclosure non esistono e restituisce come messaggio Enclosure doesn’t exist

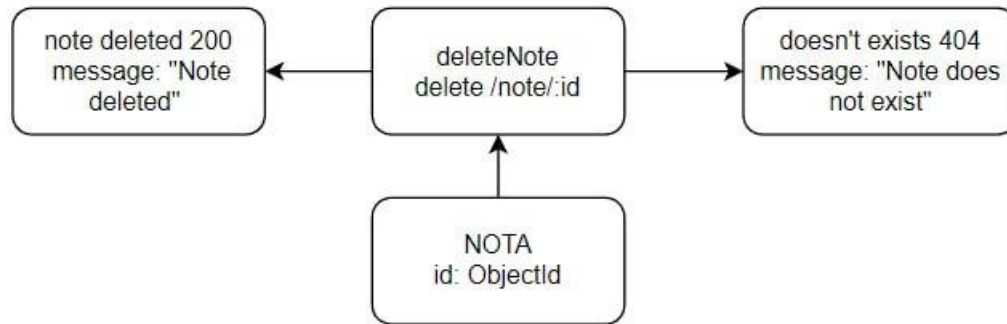


deleteNote

Questa API ha indirizzo “/note/:id” ed è caratterizzata dal metodo delete e viene usata per far sì che l’utente possa eliminare delle note tramite l’id di quest’ultime. La request body di deleteNote è costituita dal campo id, di tipo ObjectId. Si possono avere due tipi di risposta a seconda del risultato ottenuto.

Se la response body è data da code = 200 afferma che la nota è stata eliminata e restituisce come messaggio: Note deleted

Se la response body è data da code = 404 la nota non esiste e restituisce come messaggio note doesn’t exist

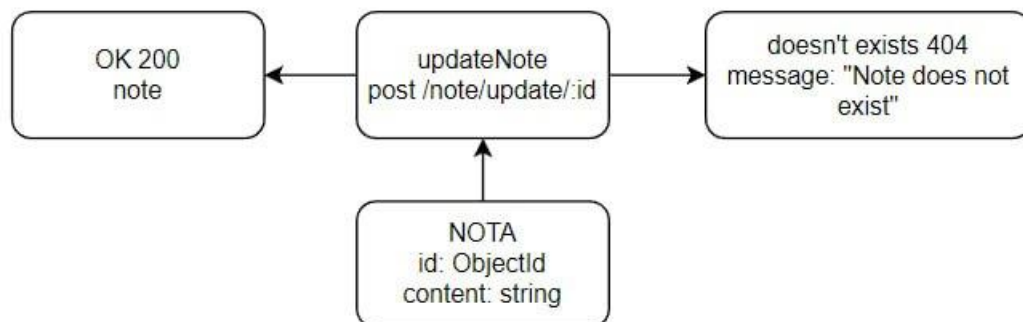


updateNote

Questa API ha indirizzo `"/note/:id"` ed è caratterizzata dal metodo `post` e viene usata per far sì che l'utente possa aggiornare determinate note tramite l'id di quest'ultime. La request body di `updateNote` è costituita dal campo `id`, di tipo `ObjectId`. Si possono avere **due** tipi di risposta a seconda del risultato ottenuto.

Se la response body è data da `code = 200` afferma che la nota è stata aggiornata e restituisce come messaggio: `Note`

Se la response body è data da `code = 404` la nota non esiste e restituisce come messaggio `note doesn't exist`

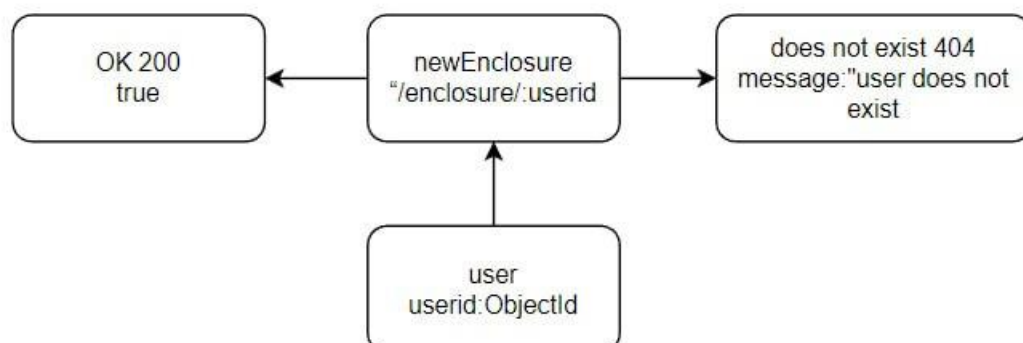


newEnclosure

Questa API ha indirizzo `"/enclosure/:userid"` ed è caratterizzata dal metodo `post` e viene usata per far sì che l'utente possa creare delle enclosure. La request body di `newEnclosure` è costituita dal nome, note, creator e data di creazione, di tipo `String` e `Data`. Si possono avere **due** tipi di risposta a seconda del risultato ottenuto.

Se la response body è data da `code = 200` afferma che l'enclosure esiste e restituisce come messaggio: `true`

Se la response body è data da `code = 404` l'enclosure non esiste e restituisce come messaggio `user doesn't exist`

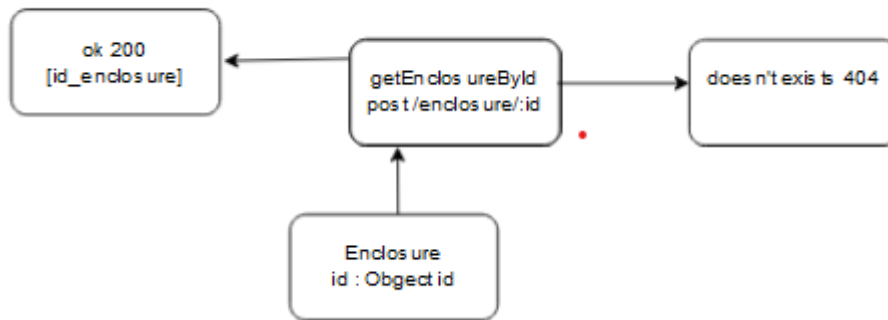


getEnclosureById

Questa API ha indirizzo `"/enclosure/:id"` ed è caratterizzata dal metodo `get` e viene usata per far sì che l'utente possa cercare delle enclosure per id. La request body di `getEnclosureById` è costituita dall'id, di tipo `ObjectId`. Si possono avere **due** tipi di risposta a seconda del risultato ottenuto.

Se la response body è data da code = 200 afferma che l'enclosure esiste e restituisce come messaggio il vettore di id dell'enclosure

Se la response body è data da code = 404 la enclosure non esiste e restituisce come messaggio user doesn't exist

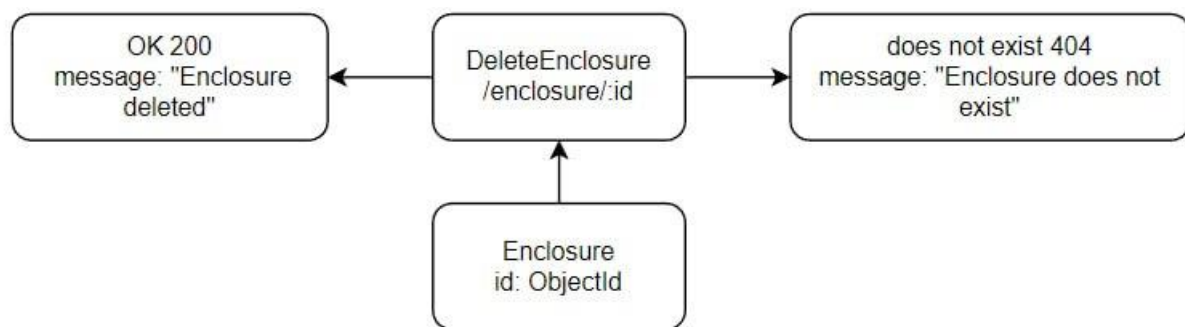


DeleteEnclosure

Questa API ha indirizzo `"/enclosure/:id"` ed è caratterizzata dal metodo `delete` e viene usata per far sì che l'utente possa eliminare delle enclosure per id. La request body di `deleteEnclosure` è costituita dall'id, di tipo `ObjectId`. Si possono avere due tipi di risposta a seconda del risultato ottenuto.

Se la response body è data da code = 200 afferma che l'enclosure è stato eliminato e restituisce come messaggio: `enclosure deleted`

Se la response body è data da code = 404 la enclosure non esiste e restituisce come messaggio `user doesn't exist`

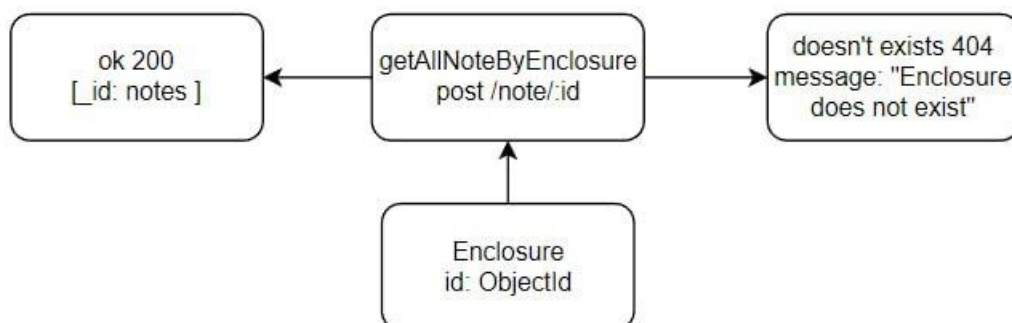


getAllEnclosureById

Questa API ha indirizzo `"/enclosure/user/:userid"` ed è caratterizzata dal metodo `get` e viene usata per far sì che l'utente possa ottenere tutte enclosure di un utente. La request body di `getAllEnclosureById` è costituita dall'id, di tipo `ObjectId`. Si possono avere due tipi di risposta a seconda del risultato ottenuto.

Se la response body è data da code = 200 afferma che tutte l'enclosure di un utente esistono e restituisce come messaggio: `[id:note]`

Se la response body è data da code = 404 che tutte l'enclosure di un utente non esistono e restituisce come messaggio `enclosure doesn't exist`



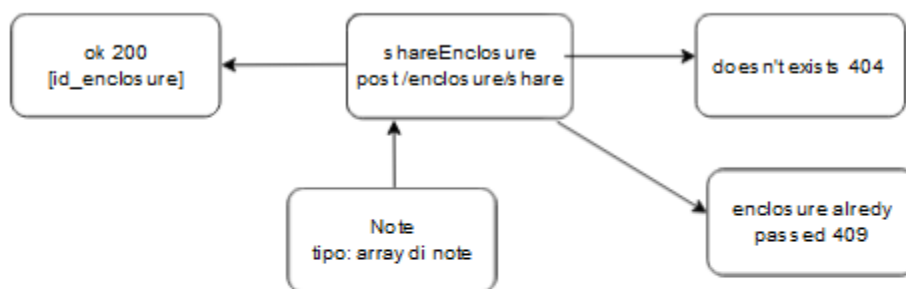
shareEnclosure

Questa API ha indirizzo “/enclosure/user/:userid” ed è caratterizzata dal metodo get e viene usata per far sì che l’utente possa ottenere tutte enclosure che hanno uno specifico id. La request body di shareEnclosure è costituita dall’id, di tipo ObjectId. Si possono avere **tre** tipi di risposta a seconda del risultato ottenuto.

Se la response body è data da code = 200 afferma che tutte l’enclosure di un utente esistono e restituisce come messaggio: [id_enclosure]

Se la response body è data da code = 404 che tutte l’enclosure di un utente non esistono e restituisce come messaggio enclosure doesn’t exist

Se la response body è data da code = 409 che tutte l’enclosure di un utente sono già state condivise e restituisce come messaggio enclosure already passed



4. DOCUMENTAZIONE API

La documentazione delle API (User, Note, Enclosure) è stata fatta tramite il modello `swagger-ui-express`, seguendo lo standard OpenAPI. Si riportano le immagini per la get, put, post e delete.

FriendlyNote CRUD ^{1.0.0}

[Base URL: localhost:8080/]

My User Project Application API

[MIT](#)

Schemes

HTTP

User API for users in the system

GET /user/:id Get user by ID

POST /user/login Logs in a user with email and password

Note API for notes in the system

POST /note/:enclosureid Create a new note

POST /note/:id Create a new note

POST /note/enclosure/:enclosureid Create a new note

PUT /note/update/:id Update an existing note

Enclosure

POST /enclosure/:id Create a new enclosure for a user

GET /enclosure/user/:userid Get all enclosures owned by a user

POST /enclosure/share Share an enclosure with a user

GET /enclosure/search/:id Search for enclosures by name

Users

GET /user Retrieve all users

POST /user Create a new user

POST /note/enclosure/:enclosureid Create a new note

Parameters

Try it out

Name	Description
enclosureid * required	The ID of the enclosure to which the note belongs

string (path)	enclosureid
------------------	-------------

note * required	The note object to create
-----------------	---------------------------

object (body)	Example Value Model
------------------	-----------------------

```
{
  "title": "My first note",
  "content": "This is the content of my first note",
  "dateCreated": "2023-02-17T12:34:56Z"
}
```

Parameter content type

application/json

Responses

Response content type application/json

Code	Description
200	OK

Example Value | Model

```
{
  "data": true
}
```

GET
/enclosure/user/:userid
Get all enclosures owned by a user

Parameters
Try it out

Name	Description
userid * required string (path)	ID of the user to get enclosures for <input type="text" value="userid"/>
page * required object (body)	The page number to get <div> Example Value Model </div> <pre> { "name": "string", "notes": [{ "title": "string", "body": "string", "author": "string", "createdAt": "2023-02-17T22:15:29.680Z", "updatedAt": "2023-02-17T22:15:29.680Z" }], "creator": {}, "dateCreated": "2023-02-17T22:15:29.680Z" } </pre> <div> Parameter content type application/json </div>

Responses
Response content type application/json

Code	Description
200	Successful operation
404	User not found

5. IMPLEMENTAZIONE FRONT END

Il frontend è un'interfaccia grafica interattiva che permette all'utente di eseguire diverse azioni grazie all'utilizzo delle API. Il frontend è stato realizzato tramite JSX e CSS di react. La schermata principale presenta due schede, una per il login e l'altra per la registrazione. Dopo l'accesso l'utente si ritroverà all'interno della home del sito. All'interno del sito, sulla sinistra, vi è un hamburger menù che permette la visualizzazione delle cartelle contenenti i propri appunti. A destra è presente una tendina che permette all'utente di fare il logout, condivisione argomenti e nuovo argomento.

Sign In



Email

Password

Non hai un account?
Registrati

Invia richiesta

Accesso con il proprio account

Sign up



Email

Password


hai già un account?
vai al log in

Invia richiesta

Creare un nuovo account

≡

Friendly Note

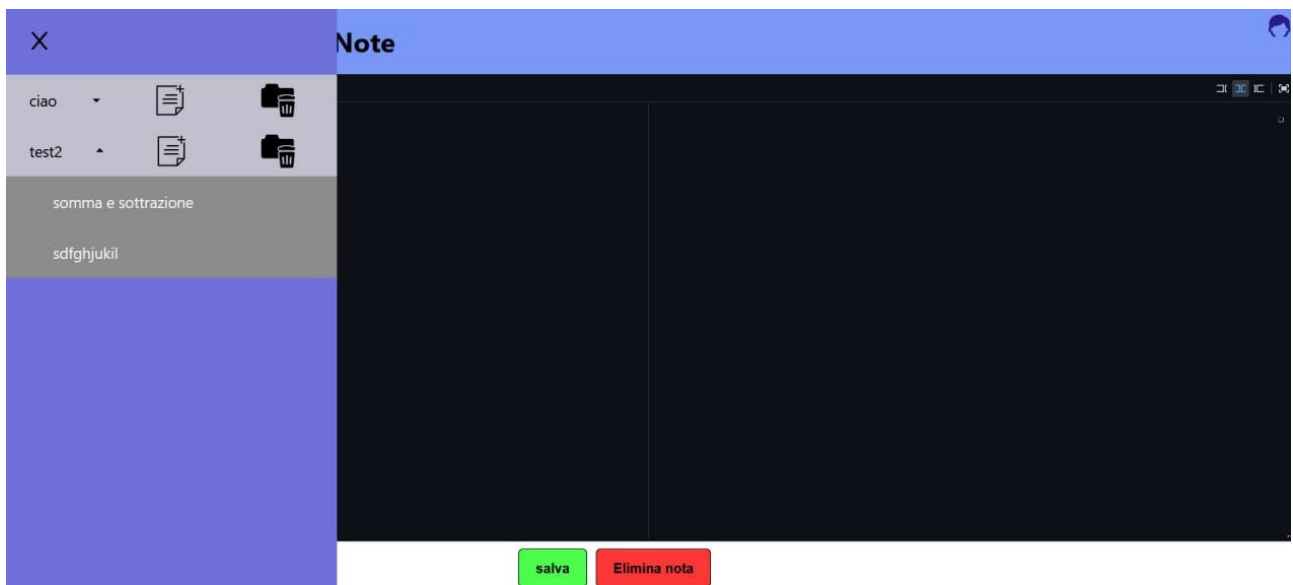




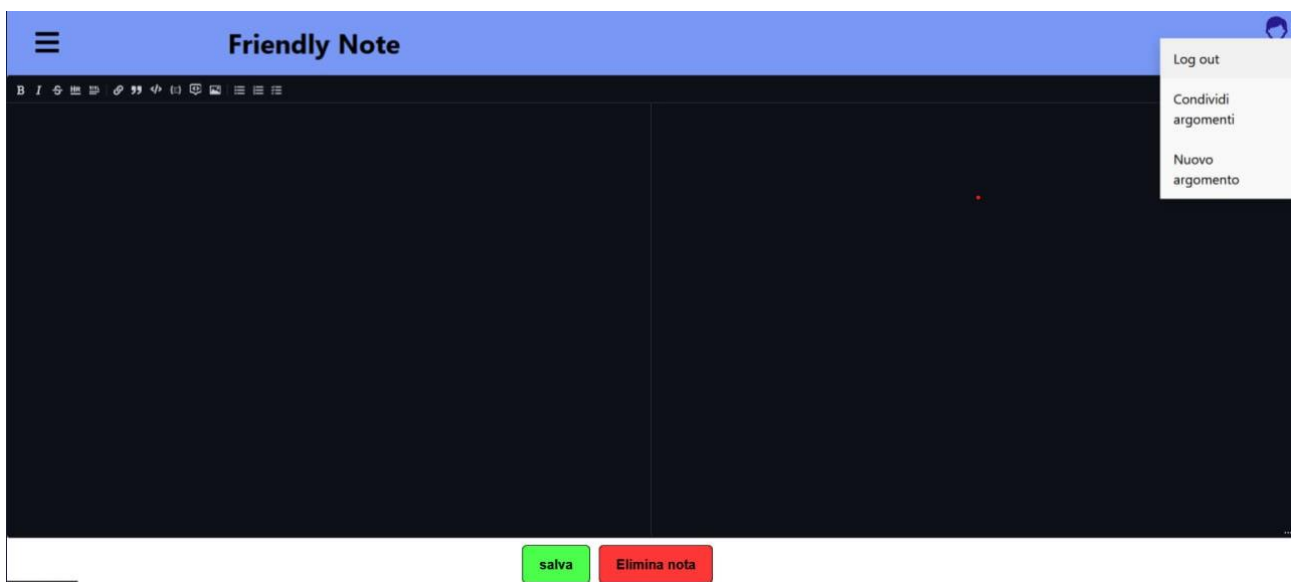
salva

Elimina nota

Pagina di home



Hamburger menù a sinistra con tutti gli argomenti che sono presenti nel database



Menù a cascata a destra con log out, condivisione argomenti e la possibilità di creare un nuovo argomento



Schermata della home con un test di scrittura

6. GITHUB REPOSITORY

Il progetto è stato caricato su GitHub. Sono presenti due depository, di cui due distinte per il frontend e il backend

7. TESTING

Qui vengono riportati alcuni screen di testing.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	57.7	20.68	24.24	58.46	
FriendlyNote	96.42	50	100	100	
server.js	96.42	50	100	100	32-41
...ndlyNote/controllers	38.37	18.51	16.66	38.69	
enclosure.js	20.75	0	0	20.75	...59,63-70,74-86
note.js	77.55	62.5	62.5	77.08	8,34-38,72-78
token.js	28	0	0	28	9-31,37,41-47
user.js	22.22	0	0	23.8	...58,64-69,76-80
FriendlyNote/models	100	100	100	100	
enclosure.js	100	100	100	100	
note.js	100	100	100	100	
token.js	100	100	100	100	
user.js	100	100	100	100	
FriendlyNote/routes	100	100	100	100	
enclosure.js	100	100	100	100	
note.js	100	100	100	100	
token.js	100	100	100	100	
user.js	100	100	100	100	
Test Suites: 1 passed, 1 total					
Tests: 8 passed, 8 total					
Snapshots: 0 total					
Time: 5.084 s					
Ran all test suites.					