**Scopo del documento**

Il presente documento riporta tutte le informazioni necessarie per lo sviluppo di una parte dell’applicazione Skupply.

In particolare, presenta le funzioni necessarie per il servizio di autenticazione (login e registrazione), le funzioni per la visualizzazione del carrello e della wishlist, la funzionalità della ricerca degli articoli e la loro visualizzazione.

Partendo dalla descrizione degli user flow dell’applicazione, il documento prosegue con la presentazione delle API necessarie (tramite l’API Model e il Modello delle risorse)

per le funzionalità appena elencate.

Per ogni API realizzata, oltre ad una descrizione delle funzionalità fornite, il documento

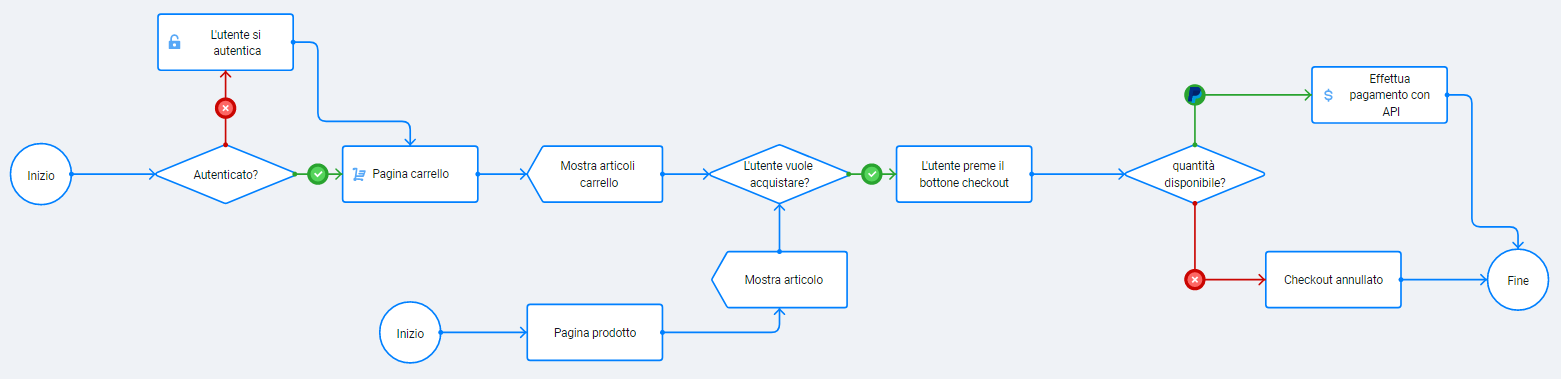
presenta la sua documentazione e i test effettuati. Infine, una sezione e’ dedicata alle

informazioni del Git Repository e il deployment dell’applicazione stessa.

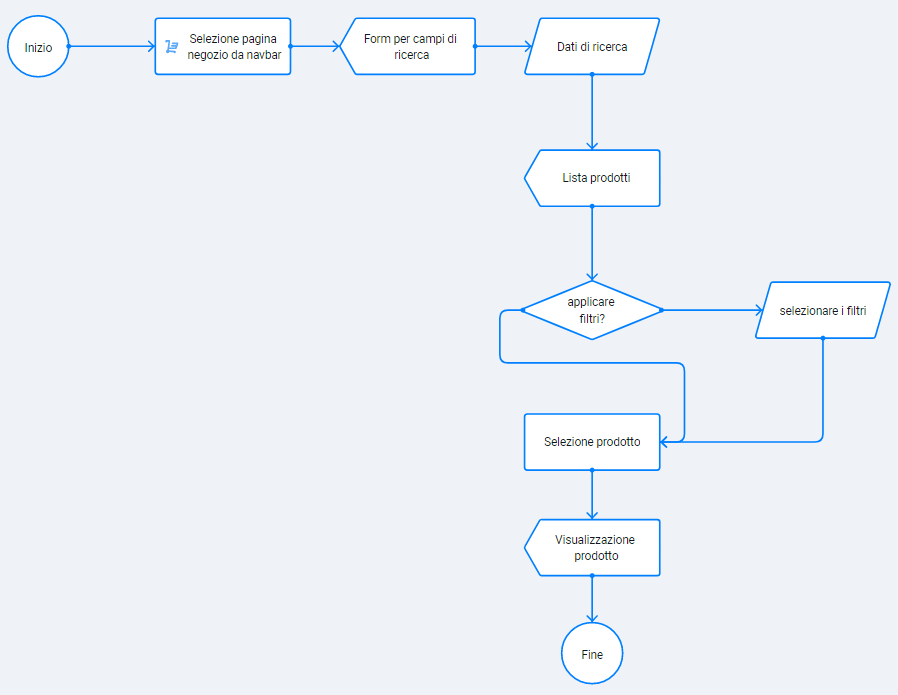
**1. User Flows**

In questa sezione del documento di sviluppo riportiamo gli “user flows” per il pagamento di un articolo/i, per l’autenticazione, per la ricerca di un prodotto, per la creazione di un annuncio, la visualizzazione del carrello e della wishlist e della chat.

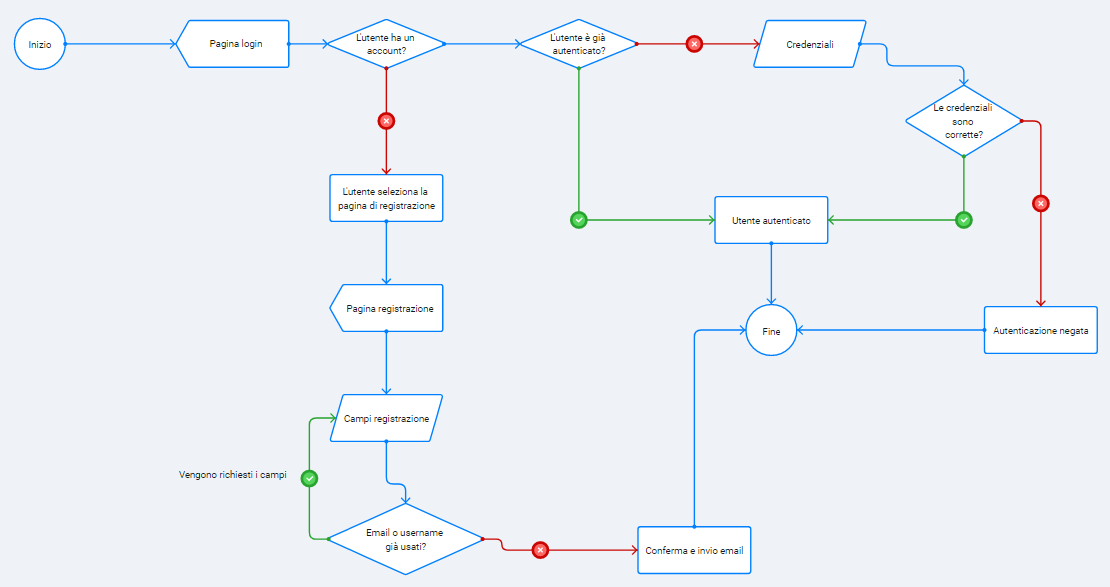
1. User flow per pagamento articolo/i



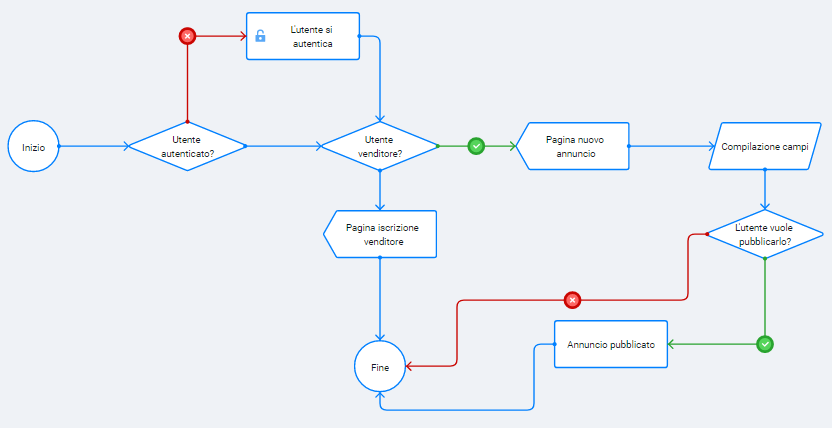
1. User flow per la ricerca di un prodotto



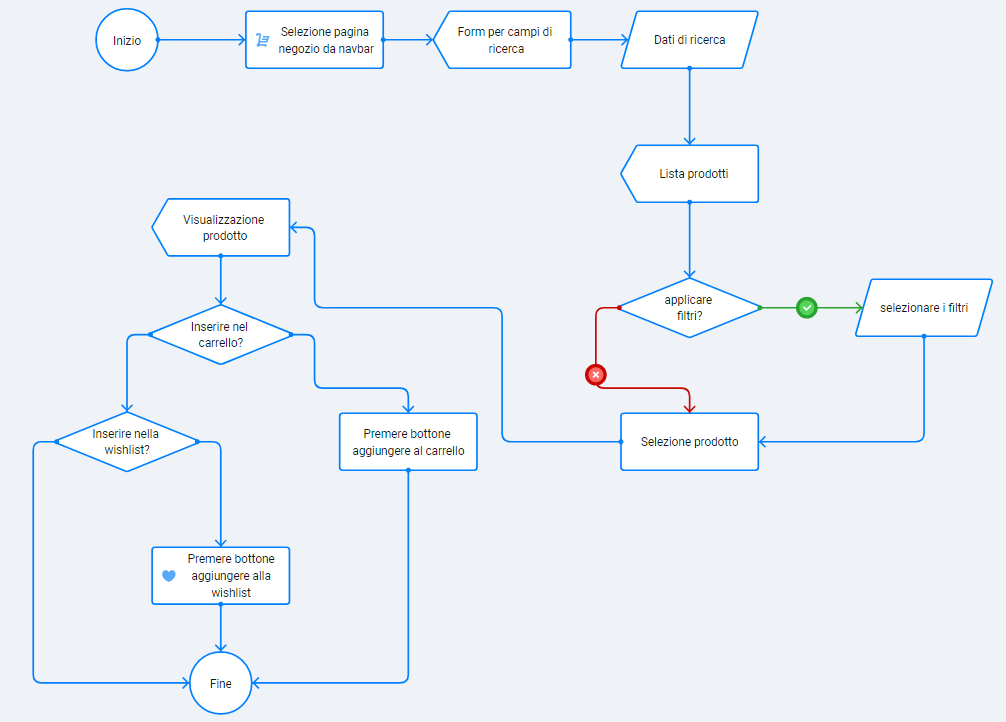
1. User flow per l’autenticazione



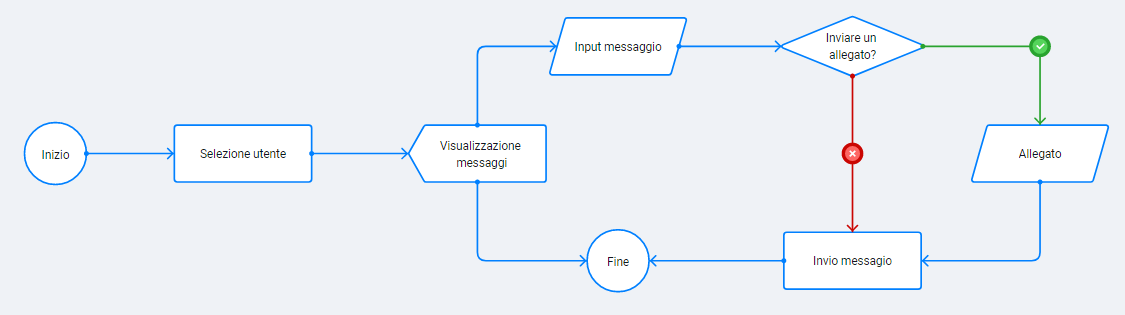
1. Use flow per la creazione di un annuncio



1. Use flow per inserire un prodotto all’interno del carrello e/o la wishlist



1. Use flow per utilizzo della chat



**2. Application Implementation and**

**Documentation**

Nelle sezioni precedenti abbiamo identificato alcune funzionalità che devono essere implementate all’interno della mostra applicazione Skupply e di come l’utente può utilizzarle.

L’applicazione è stata sviluppata utilizzando NodeJs e VueJS mentre per la gestione dei dati abbiamo utilizzato MongoDB;

**2.1. Project Structure**

La struttura del progetto è presentata in figura nella pagina successiva.

Tale progetto è suddiviso in 2 cartelle separate, frontend e backend che sono autoesplicativi.

La cartella backend è composta dalle seguenti cartelle:

* Controllers: per la gestione delle api
* Models: per la creazione dei modelli del db
* Routes: per la gestione degli endpoints delle api
* Test: per le funzioni di test delle api
* Varie cartelle per i moduli
* Utils: per funzioni di verifica di credenziali e indirizzi e-mail

E dai seguenti file:

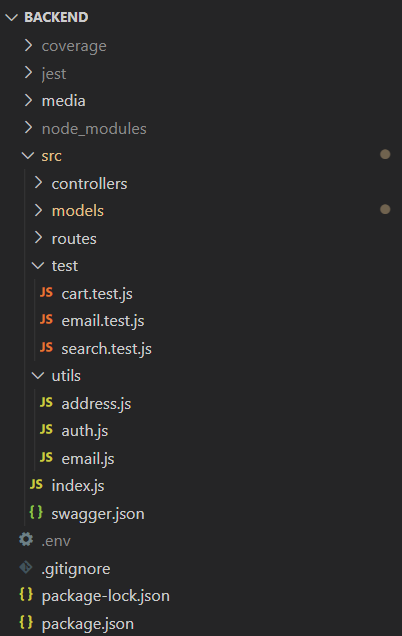
* Index: il file che inizializza il server
* Swagger: il file per la documentazione delle API

La cartella frontend è composta dalle seguenti cartelle:

* Route: per la gestione degli endpoints delle api
* Stores: per la gestione della sessione
* Views: per la gestione delle interfacce grafiche
* Components: per la gestione dei componenti grafici utilizzati
* Assets: per la gestione dei font

E dai seguenti file:

* Main: file di inizializzazione

Immagine che contiene testo

Descrizione generata automaticamente

**2.2. Project Dependencies**

I seguenti moduli Node sono stati utilizzati e aggiunti al file Package.Json

• Express

• MongoDB

• Cors

• Geoapify

• Vicons

• Pinia

• Vue

• Email validator

• Jest

• Jsonwebtoken

• Supertest

• Multer

**2.3. Project Data or DB**

Per la gestione dei dati utili all’applicazione abbiamo definito le seguenti strutture dati:

Immagine che contiene tavolo

Descrizione generata automaticamente

E i seguenti tipi di dati in ordine come rappresentato nell’immagine:

Articles

Immagine che contiene testo

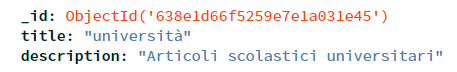
Descrizione generata automaticamente

Buyers

Immagine che contiene testo

Descrizione generata automaticamente

Categories



Chats

Immagine che contiene tavolo

Descrizione generata automaticamente

Messages

Immagine che contiene testo

Descrizione generata automaticamente

Orders

Immagine che contiene testo

Descrizione generata automaticamente

Proposals

Immagine che contiene testo

Descrizione generata automaticamente

Reviews

Immagine che contiene testo

Descrizione generata automaticamente

Sellers

Immagine che contiene testo

Descrizione generata automaticamente

Users



**2.4. Project APIs**

**2.4.1. Resources Extraction from the Class**

**Diagram**

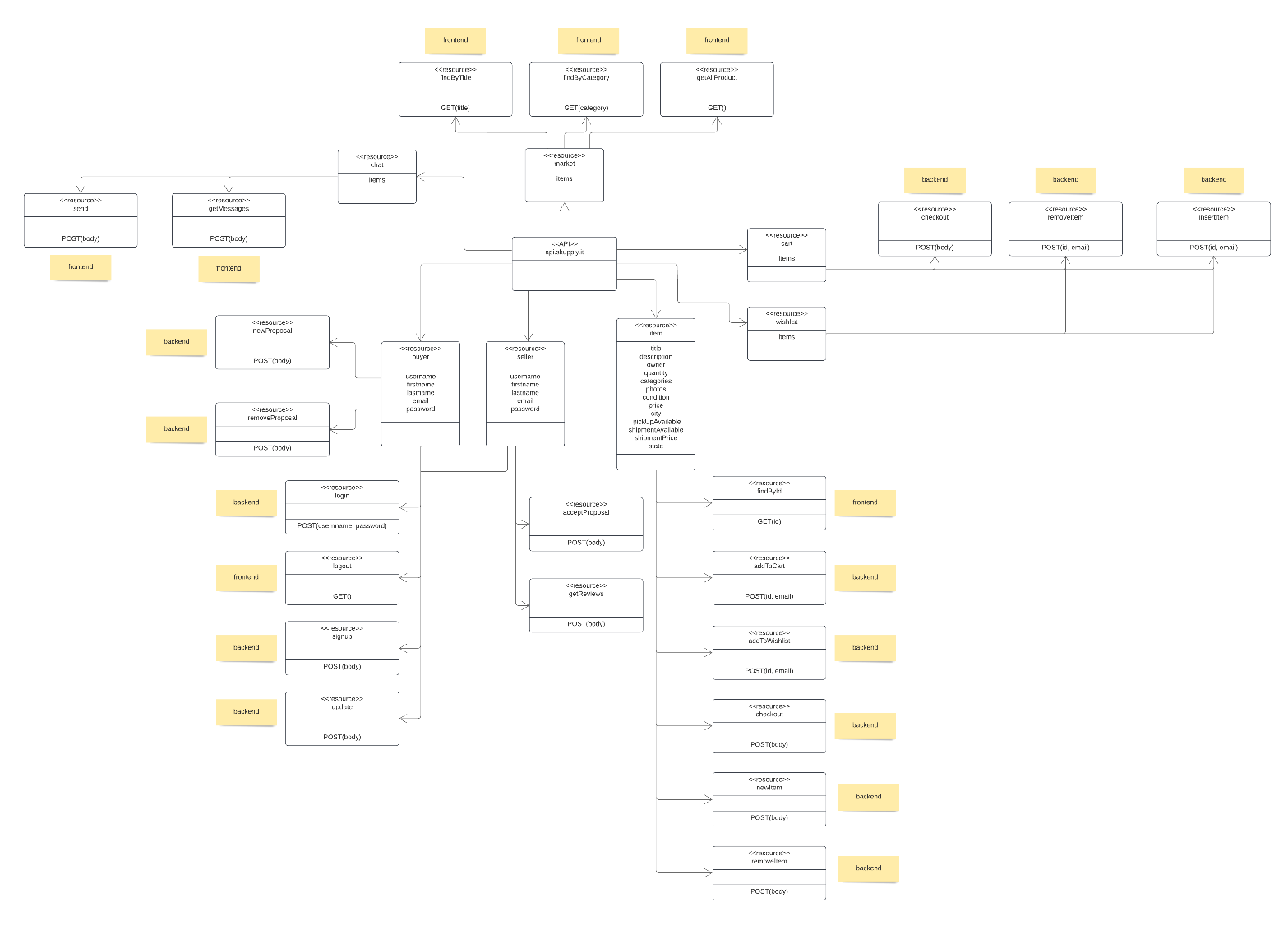
Nel seguente diagramma vengono visualizzate le risorse estratte dal diagramma delle classi descritto precedentemente nel D3 e se le funzioni hanno un esito nel frontend o nel backend.

Qui vengono elencate le api che l’utente può utilizzare all’interno della piattaforma.

Per le entità cart e wishlist vengono utilizzate per entrambe le stesse funzioni come definito nel diagramma delle classi.

Per l’entità market, che opera su una lista di elementi (gli annunci), vengono definite le funzioni di ricerca disponibili.

Come specificato nel D3, l’utente viene diviso in due possibili entità, buyer e seller e anche per loro sono state definite delle api.

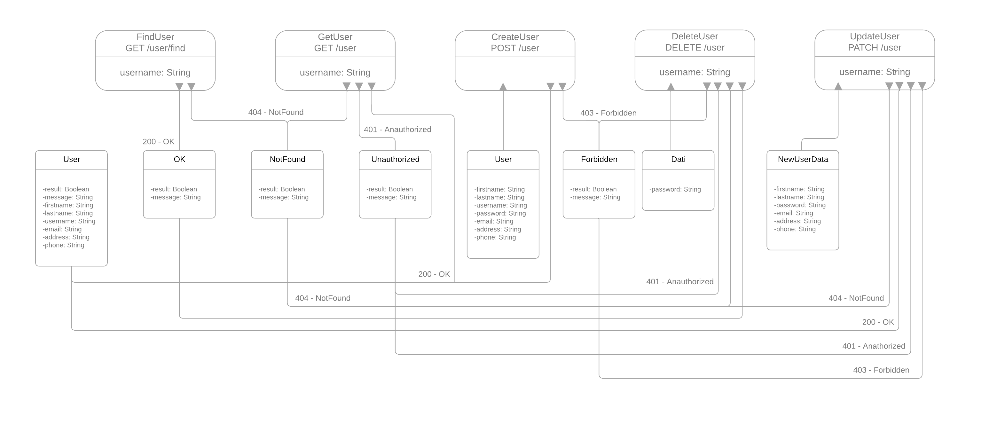


**2.4.2. Resources Models**

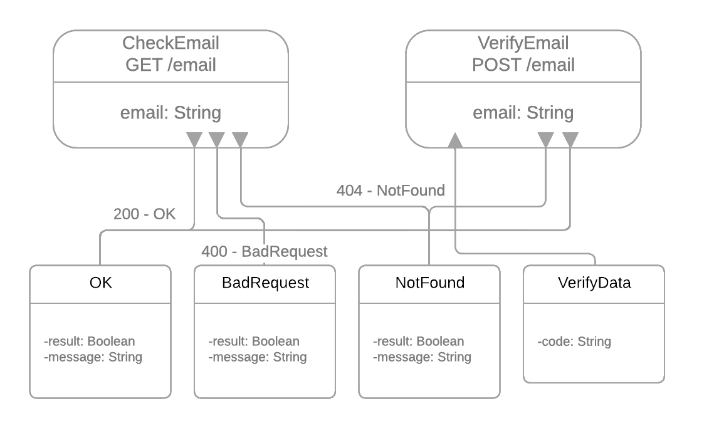
Il resources models presenta lo sviluppo tramite diagrammi delle api che sono state definite nel resources extraction.

Di seguito verranno elencate le api definite per ogni risorsa con una immagine illustrativa.

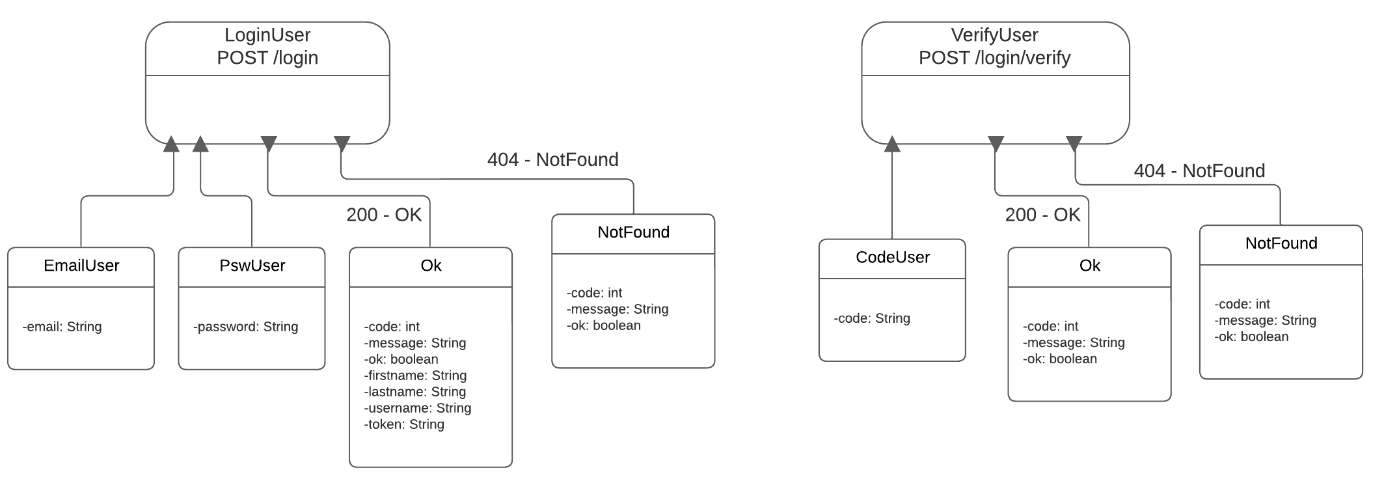
User resource model



Email resource model

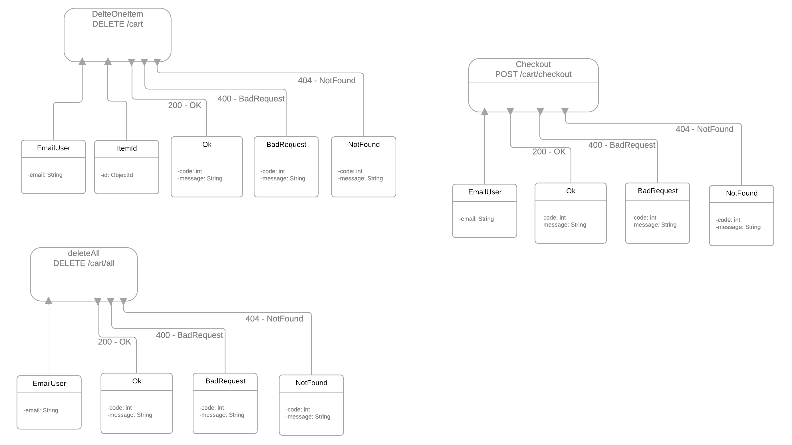


Login resource model

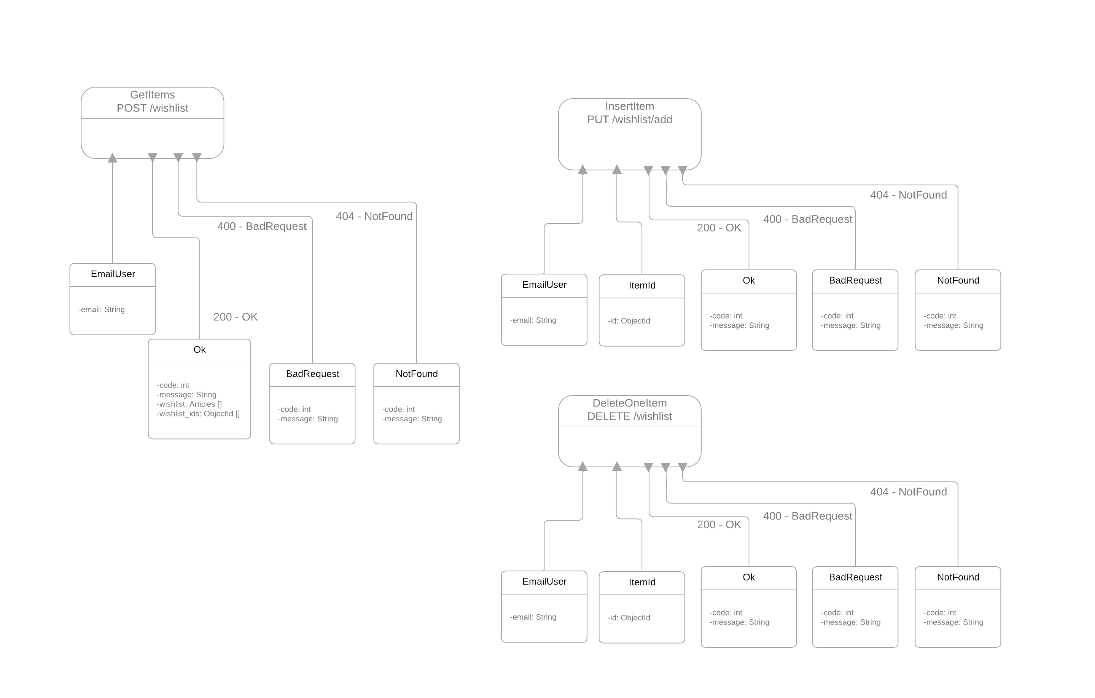


Cart resource model

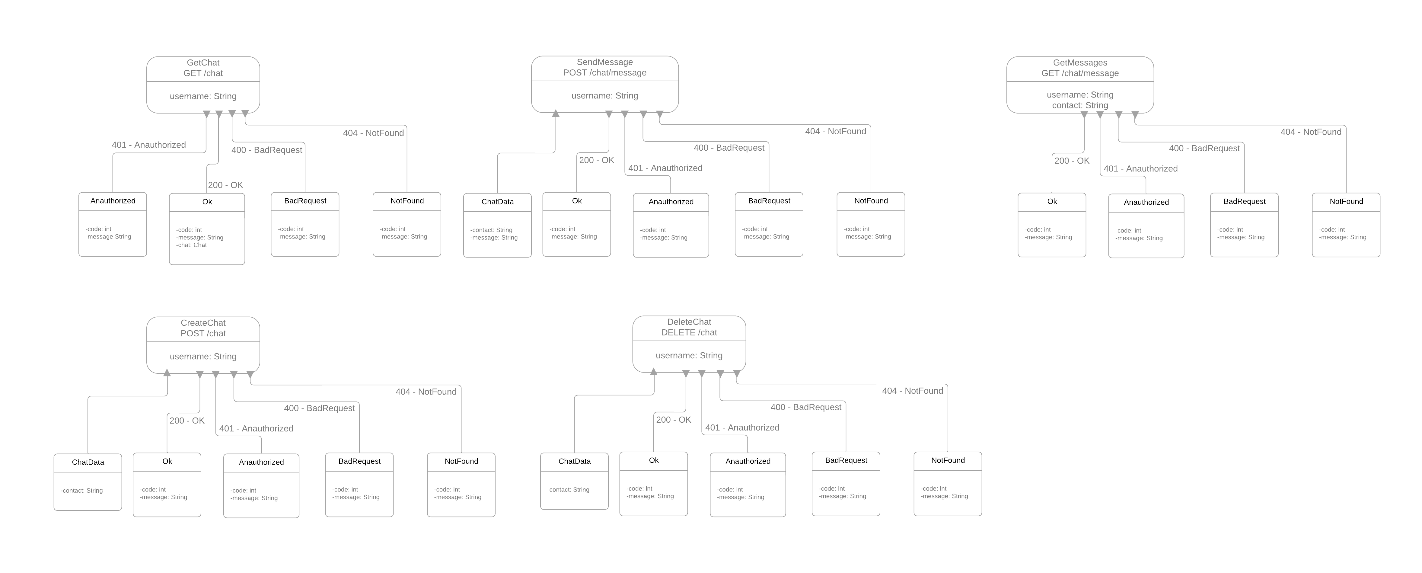
Immagine che contiene testo, nero, schermo

Descrizione generata automaticamente

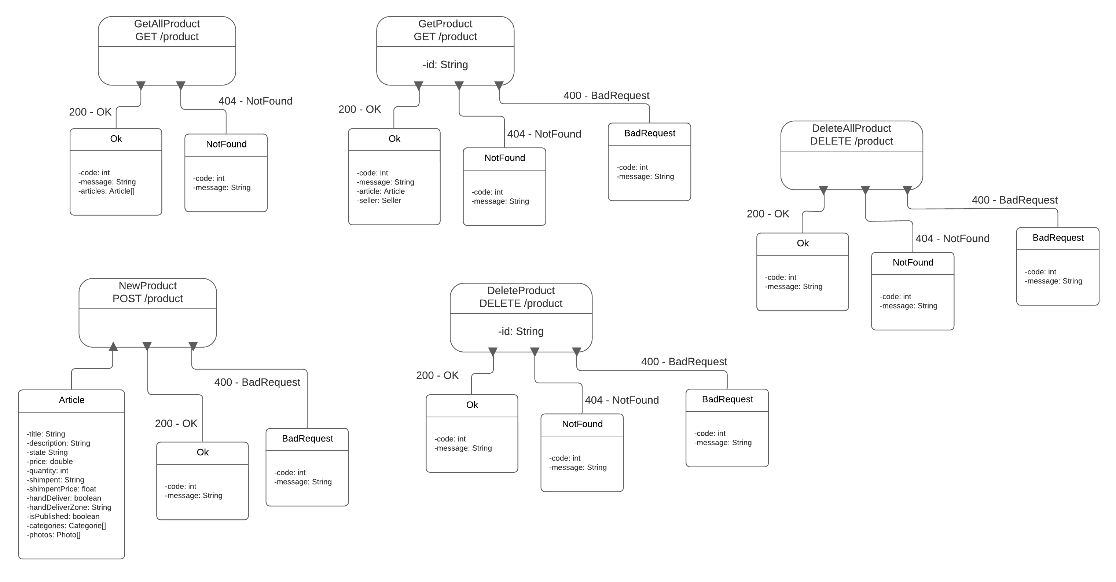
Wishlist resource model



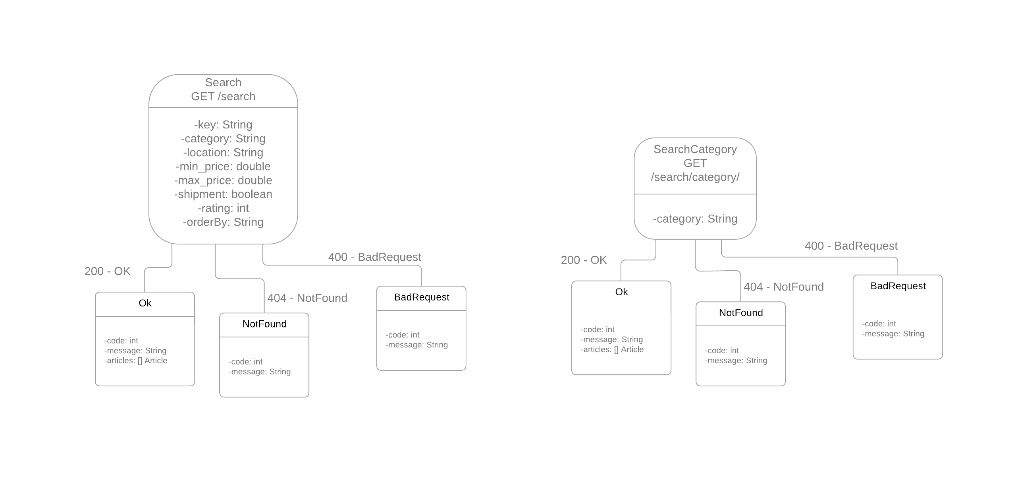
Chat resource model



Product resource model

****

Search resource model

****

**2.5. Sviluppo API**

Nel paragrafo seguente verranno descritte le api che sono state sviluppate e delle immagini illustrative

User API

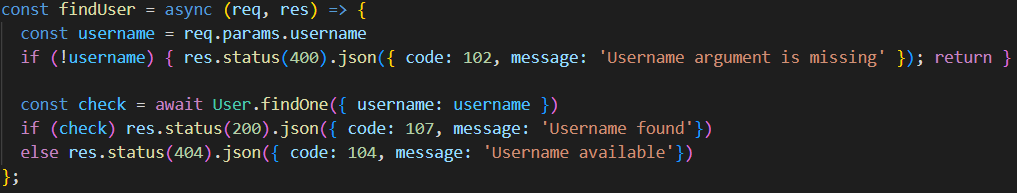
Api per la creazione di un nuovo user

Immagine che contiene testo

Descrizione generata automaticamente

Tale api prende dal body della richiesta POST i dati relativi all’utente e, utilizzando lo schema definito per lo user, salva nel database utilizzando la mongoose dependencies.

Api per trovare uno user con un dato username



Tale api utilizza lo username passato come parametro della richiesta e verifica se tale username è già in uso verificando se nel database esiste un User con tale username.

Email API

Api per verificare l’esistenza di una email

Immagine che contiene testo, monitor, interni, screenshot

Descrizione generata automaticamente

Tale api prende dalla query il campo email e verifica se tale email è già stata usata.

Api per verifica account

Immagine che contiene testo

Descrizione generata automaticamente

Tale api prendendo l’email e il codice di verifica, convalida la verifica dell’account.

Login API

Cart API

Wishlist API

Chat API

Product API

Search API

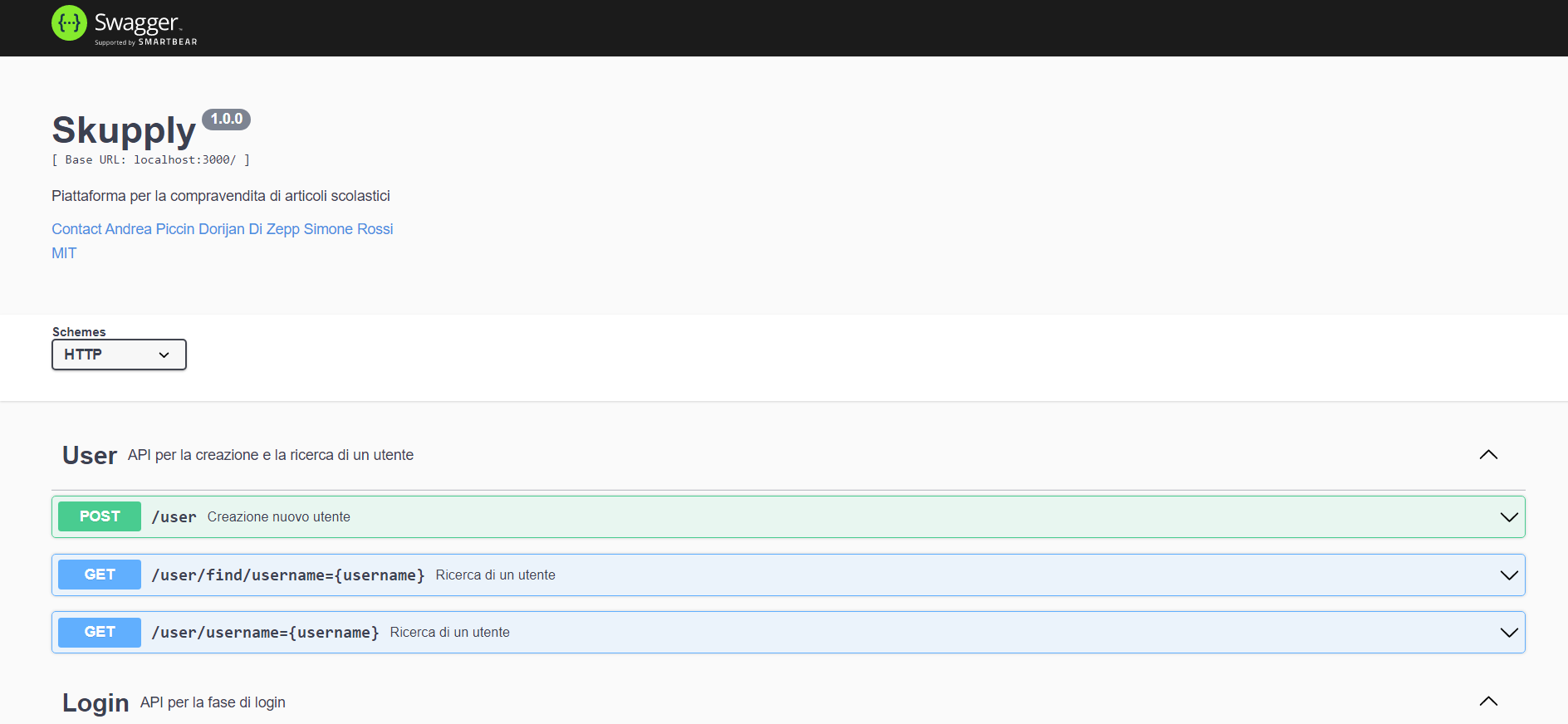
Proposal API

**3. API documentation**

Le API locali fornite dall’applicazione Skupply e descritte nella sezione precedente sono state documentate utilizzando il modulo NodeJS chiamato Swagger UI Express. In questo modo la documentazione relativa alle API è direttamente disponibile a chiunque veda il codice sorgente.

L’endpoint da invocare per raggiungere la seguente documentazione è:

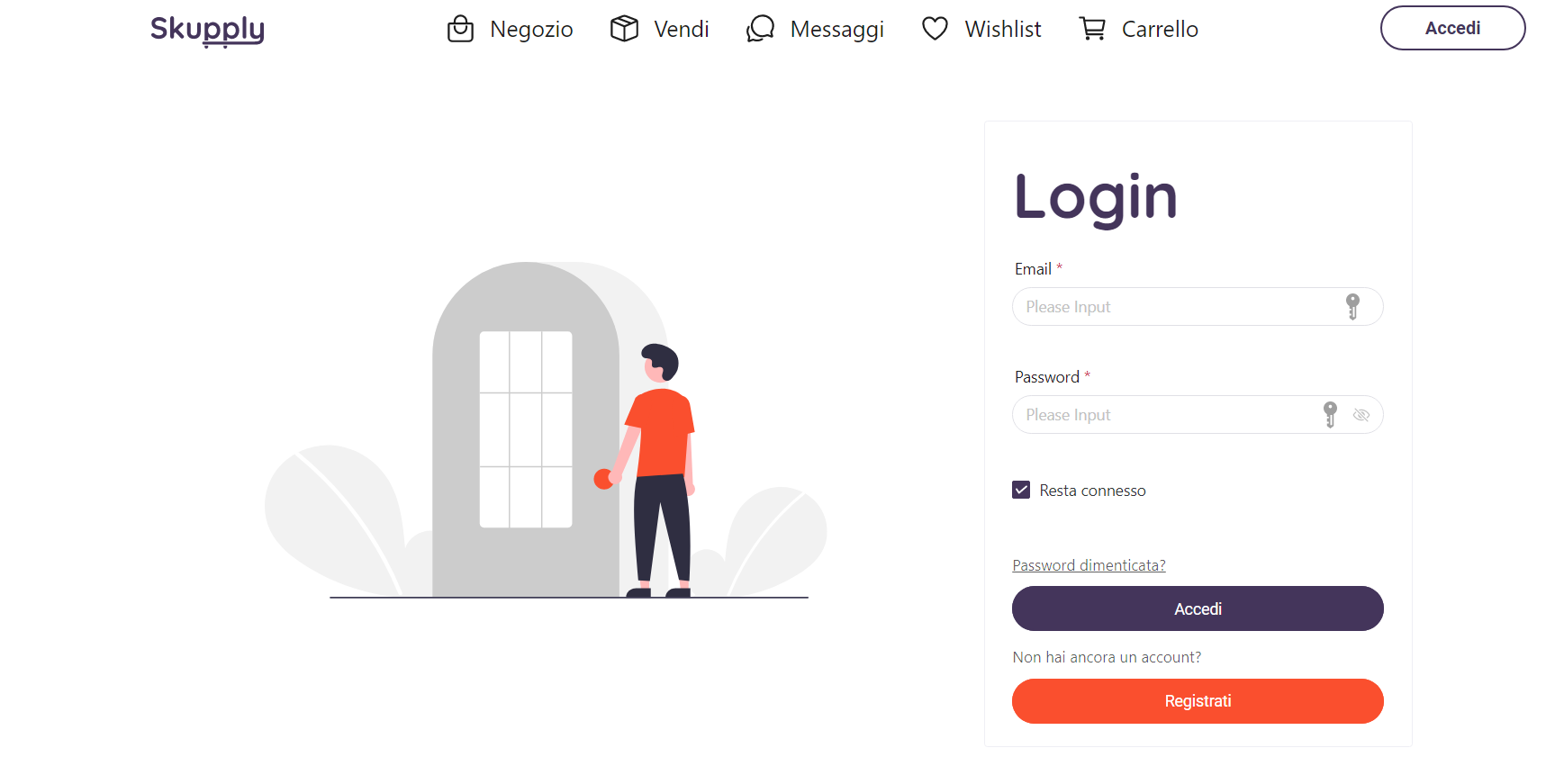
[**http://localhost:8080/api-docs**](http://localhost:8080/api-docs)

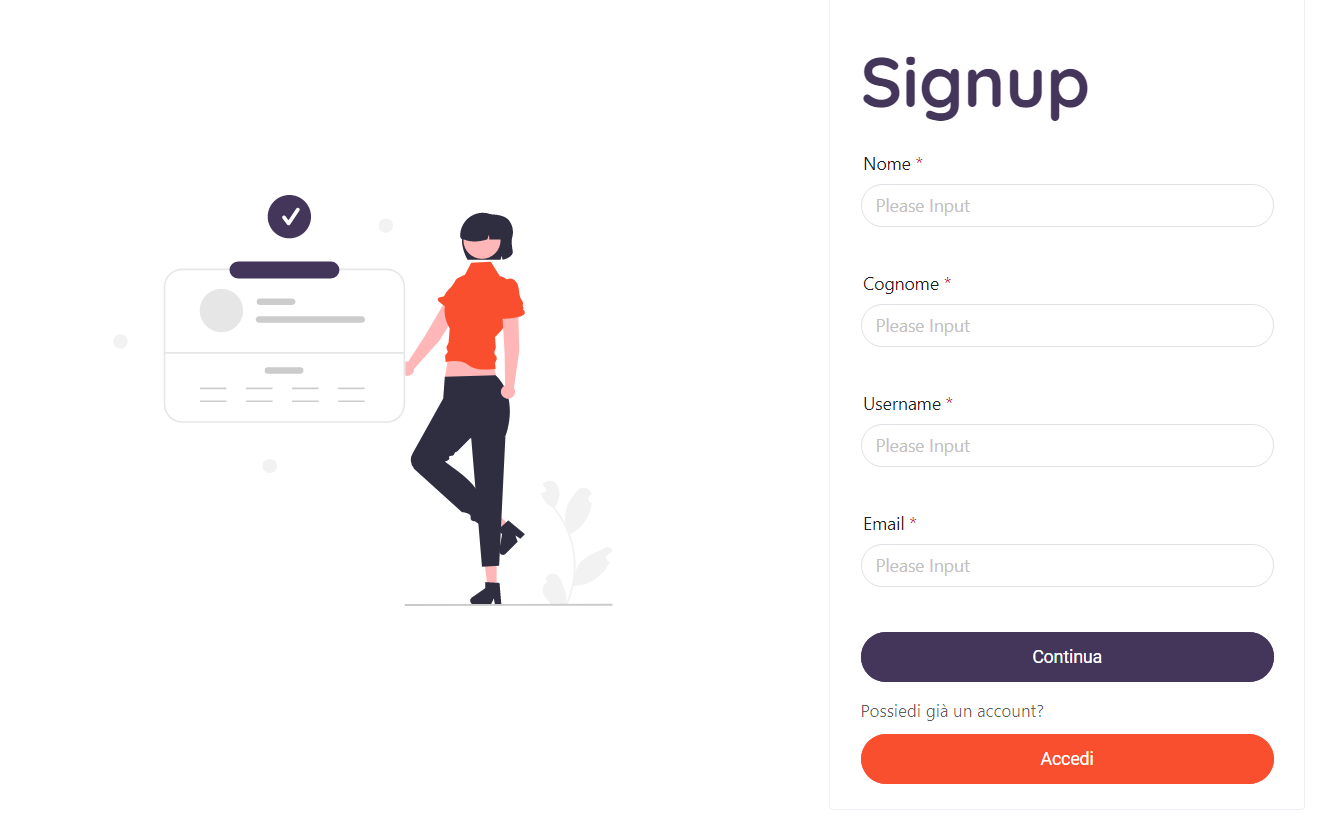


**4. FrontEnd Implementation**

In questo paragrafo viene descritta l’implementazione delle varie pagine frontend disponibili all’interno della piattaforma, descrivendo brevemente alcune funzionalità come già fatto nel D2 e con l’inserimento di alcune immagini illustrative del codice e del rendering della pagina.

Pagina di login e registrazione





**5. GitHub Repository and Deployment Info**

La Repo Git è strutturata su 4 sotto repo:

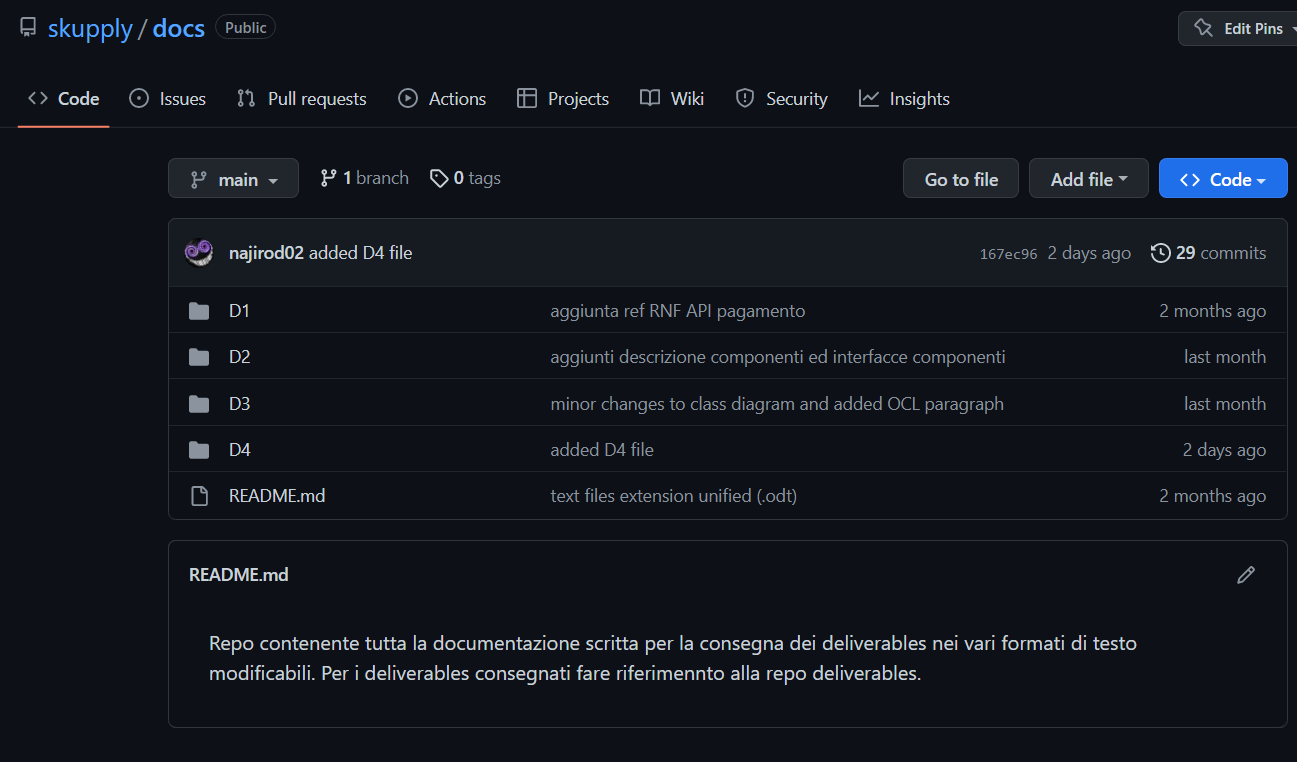
* Docs, per la fase di sviluppo dei documenti
* Deliverables, per archiviare le versioni finali dei documenti
* Frontend, per la fase di sviluppo della parte grafica e di richiesta alle api
* Backend, per la fase di gestione e sviluppo delle api.

Immagine che contiene testo, monitor, screenshot, schermo

Descrizione generata automaticamente

Di seguito alcune immagini in dettaglio delle repo.

Per la repo docs, abbiamo creato una cartella per ogni deliverable così da mantenere separato e ordinato la fase di sviluppo e revisione dei documenti



Per la repo deliverables non abbiamo pensato ad una struttura a cartelle come per la repo docs ma abbiamo usato la struttura lineare di base.

Immagine che contiene testo, screenshot, monitor, schermo

Descrizione generata automaticamente

La repo frontend e in modo analogo la repo backend, rispecchiano la struttura descritta nel paragrafo 2.1.

**Immagine che contiene testo, monitor, screenshot, schermo

Descrizione generata automaticamente**

**Immagine che contiene testo, monitor, screenshot, schermo

Descrizione generata automaticamente**

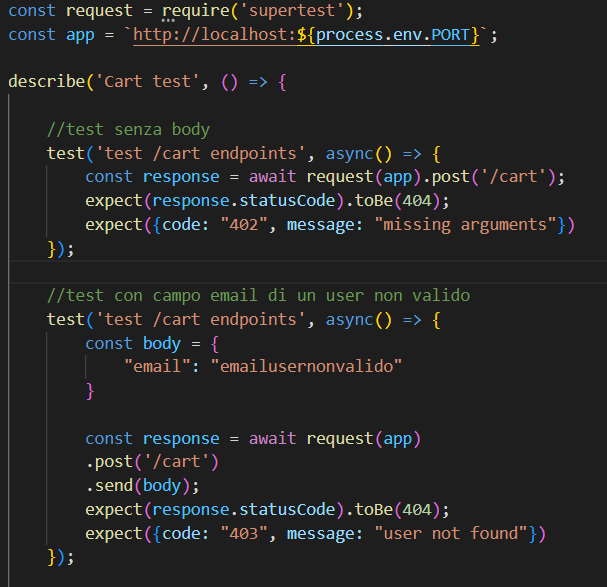
**6. Testing**

Per la fase di testing è stato fatto uso delle librerie jest e supertest.

Di seguito verranno descritti alcuni casi di test e del loro esito.

Da sottolineare che durante lo sviluppo delle api è stato deciso di includere per ogni messaggio di risposta un codice e un messaggio relativo che indicasse nel modo più semplice e preciso l’esito della richiesta, sia se è andata a buon fine sia se è avvenuto qualche errore.

Cart test :



Questi sono i casi di test che verificano la corretta verifica dei parametri passati nel body della richiesta utilizzando la libreria supertest. Come indicato nei commenti, la prima richiesta è effettuata senza parametri e la seconda con un campo email che non fa riferimento a nessun profilo.

Entrambi i test restituiscono un codice di errore 404 ma nel corpo della risposta restituiscono due codici differenti e di conseguenza anche i messaggi.

Immagine che contiene testo

Descrizione generata automaticamente

In questo caso di test, ci si aspetta una corretta restituzione degli articoli del profilo, vengono fatti alcuni controlli sulla presenza di alcuni parametri come l’array contenente gli oggetti articolo e che le quantità di ogni articolo sia maggiore o uguale a 0.

Email test

Immagine che contiene testo

Descrizione generata automaticamente

In questo test, impiegando la libreria jet, vengono verificate le api per la verifica degli indirizzi email in particolare, se l’email è valida, se è raggiungibile e se l’indirizzo email è già stato impiegato per un account.

Search test

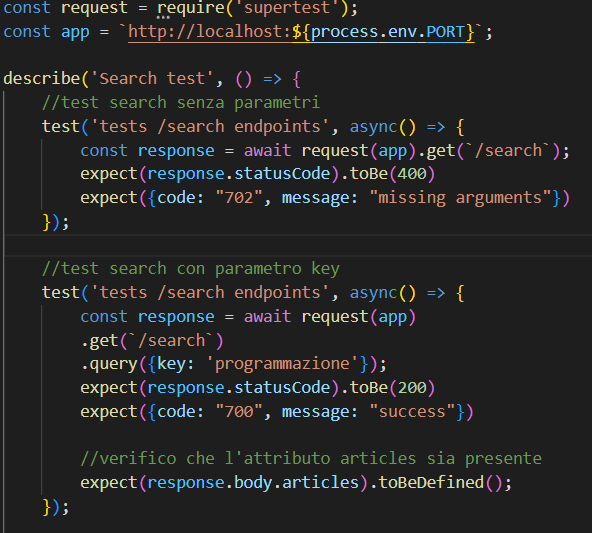


Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

La fase di test per gli endpoints della ricerca degli articoli è più articolata rispetto agli altri questo perché l’api permette il passaggio di molti parametri che possono essere combinati fra di loro.

Come visualizzato nelle immagini, i primi casi si sono incentrati sulla presenza dei parametri base necessari key e category. Successivamente sulla base di uno o entrambi i parametri base si sono costruiti gli altri casi di test come il prezzo minimo degli articoli o l’appartenenza ad una certa categoria.

<<parte sul deployment con heroku o travic cli>>