Progetto:

**iSpesa**

Titolo del documento:

**Documento di architettura**

Informazioni Documento

|  |  |  |  |
| --- | --- | --- | --- |
| Nome Documento | *iSpesa\_Sviluppo\_Applicazione\_D4* | Numero  Documento | D4 |
| Descrizione | Il documento parla dello sviluppo dell’applicazione | | |

Indice

[Scopo del documento 5](#_Toc157959856)

[1. User-Flow 6](#_Toc157959857)

[2. Implementazione dell’Applicazione e Documentazione 7](#_Toc157959858)

[2.1 Struttura del progetto 8](#_Toc157959859)

[3. Dipendenze 8](#_Toc157959860)

[4. Modelli nel Database 9](#_Toc157959861)

[4.1 Modello Amministratore 9](#_Toc157959862)

[4.2 Modello Utente Registrato 9](#_Toc157959863)

[4.3 Modello Categoria 10](#_Toc157959864)

[4.4 Modello Negozio 10](#_Toc157959865)

[4.5 Modello Negozi Preferiti 10](#_Toc157959866)

[4.6 Modello Prodotto 11](#_Toc157959867)

[4.6 Modello Prodotti Preferiti 11](#_Toc157959868)

[4.7 Modello Storico Prezzi 11](#_Toc157959869)

[4.8 Modello Sconti 12](#_Toc157959870)

[4.9 Modello Validità Sconto Prodotto 12](#_Toc157959871)

[4.10 Modello Validità Sconto Categoria 13](#_Toc157959872)

[4.11 Modello Recensione 13](#_Toc157959873)

[4.12 Modello Volantino 14](#_Toc157959874)

[5. API del Progetto 15](#_Toc157959875)

[5.1 Estrazione delle risorse dal Diagramma delle Classi 15](#_Toc157959876)

[6. Diagrammi delle Risorse 17](#_Toc157959877)

[6.1 Diagramma delle Risorse 1: Prodotto 17](#_Toc157959878)

[6.2 Diagramma delle Risorse 2: Negozio 18](#_Toc157959879)

[6.3 Diagramma delle Risorse 3: Sconto 18](#_Toc157959880)

[6.4 Diagramma delle Risorse 4: Volantino 18](#_Toc157959881)

[6.5 Diagramma delle Risorse 5: Utente 19](#_Toc157959882)

[6.6 Diagramma delle Risorse 6: Recensioni 19](#_Toc157959883)

[6.7 Diagramma delle Risorse 7: Mail 20](#_Toc157959884)

[6.8 Diagramma delle Risorse 8: Categoria 20](#_Toc157959885)

[7. Sviluppo API 21](#_Toc157959886)

[7.1 API per il modello Utente 21](#_Toc157959887)

[7.1.1 Login 21](#_Toc157959888)

[7.1.2 Registrati 21](#_Toc157959889)

[7.1.3 Elimina Utente 21](#_Toc157959890)

[7.1.4 Ripristino Password 21](#_Toc157959891)

[7.1.5 Attiva2AF 21](#_Toc157959892)

[7.1.6 Modifica Password 21](#_Toc157959893)

[7.1.7 ModificaEmail 22](#_Toc157959894)

[7.1.8 Modifica Numero Telefono 22](#_Toc157959895)

[7.1.9 Modifica Foto Profilo 22](#_Toc157959896)

[7.1.10 Aggiungi Prodotto ai Preferiti 22](#_Toc157959897)

[7.1.11 Rimuovi Prodotto dai Preferiti 22](#_Toc157959898)

[7.1.12 Aggiungi Negozio ai Preferiti 22](#_Toc157959899)

[7.1.13 Rimuovi Negozio dai Preferiti 22](#_Toc157959900)

[7.1.14 Ottieni Dati utente 23](#_Toc157959901)

[7.1.15 Trova Tutti Utenti 23](#_Toc157959902)

[7.1.16 Ottieni Prodotti Preferiti 23](#_Toc157959903)

[7.1.17 Check Prodotto Preferito 23](#_Toc157959904)

[7.1.18 Ottieni Negozi Preferiti 23](#_Toc157959905)

[7.1.19 Check Negozi oPreferito 23](#_Toc157959906)

[7.2 API per il modello Volantino 23](#_Toc157959907)

[7.2.1 Salva Volantino 23](#_Toc157959908)

[7.2.2 Elimina Volantino 23](#_Toc157959909)

[7.2.3 Trova Tutti Volantini 24](#_Toc157959910)

[7.2.4 Trova Volantini Filtro Negozio 24](#_Toc157959911)

[7.3 API per il modello Sconto 24](#_Toc157959912)

[7.3.1 Salva Sconto 24](#_Toc157959913)

[7.3.2 Elimina Sconto 24](#_Toc157959914)

[7.3.3 Trova Tutti Sconti 24](#_Toc157959915)

[7.3.4 Trova Sconti con Categoria 24](#_Toc157959916)

[7.3.5 Trova Sconti con Categoria Filtro Negozio 24](#_Toc157959917)

[7.3.6 Trova Sconti con Prodotto 25](#_Toc157959918)

[7.3.5 Trova Sconti con Prodotto Filtro Negozio 25](#_Toc157959919)

[7.4 API per il modello Negozi 25](#_Toc157959920)

[7.4.1 Salva Negozio 25](#_Toc157959921)

[7.4.2 Elimina Negozio 25](#_Toc157959922)

[7.4.3 Modifica Orari 25](#_Toc157959923)

[7.4.4 Modifica Ubicazione 25](#_Toc157959924)

[7.4.5 Trova Tutti Negozi 26](#_Toc157959925)

[7.4.6 Trova Tutti Negozi Filtro Nome 26](#_Toc157959926)

[7.4.7 Trova Tutti Negozi Filtro Ubicazione 26](#_Toc157959927)

[7.4.8 Trova Tutti Negozi Filtro ID 26](#_Toc157959928)

[7.5 API per il modello Prodotti 26](#_Toc157959929)

[7.5.1 Salva Prodotto 26](#_Toc157959930)

[7.5.2 Elimina Prodotto 26](#_Toc157959931)

[7.5.3 Modifica Immagine 26](#_Toc157959932)

[7.5.4 Aggiungi Prezzo 27](#_Toc157959933)

[7.5.5 Trova Tutti Prodotti 27](#_Toc157959934)

[7.5.6 Trova Tutti Prodotti Scontati 27](#_Toc157959935)

[7.5.7 Trova Tutti Prodotti Scontati Filtro Categoria 27](#_Toc157959936)

[7.5.8 Trova Prodotti Filtro Nome 27](#_Toc157959937)

[7.5.9 Trova Prodotto Filtro ID 27](#_Toc157959938)

[7.5.10 Trova Prodotti Filtro Negozio 27](#_Toc157959939)

[7.5.11 Trova Prodotti Filtro Categoria 27](#_Toc157959940)

[7.6 API per il modello Recensioni 28](#_Toc157959941)

[7.6.1 Salva Recensione 28](#_Toc157959942)

[7.6.2 Elimina Recensione 28](#_Toc157959943)

[7.6.3 Trova Recensioni Filtro Utente 28](#_Toc157959944)

[7.6.4 Trova Recensioni Filtro Negozio 28](#_Toc157959945)

[7.6.5 Trova Tutte Recensioni 28](#_Toc157959946)

[7.7 API per il modello Mail 28](#_Toc157959947)

[7.7.1 Invia Mail 28](#_Toc157959948)

[7.8 API per il modello Categorie 28](#_Toc157959949)

[7.8.1 Categorie 29](#_Toc157959950)

# Scopo del documento

Il seguente documento riporta tutte le informazioni necessarie per descrivere lo sviluppo di una parte, abbastanza completa, dell’applicazione web iSpesa.

Nel primo capitolo viene riportato lo user flow, ovvero una descrizione tramite diagramma di tutte le azioni che si possono eseguire sulla parte implementata di iSpesa, descrivendo le varie richieste effettuabili a front-end in ogni pagina e le varie risposte possibili.

Successivamente rappresentiamo una struttura del codice realizzato, descrivendo le dipendenze installate, i modelli realizzati e le API implementate. Una attenta descrizione delle API implementate viene fatta con il diagramma delle risorse e il diagramma di estrazione delle risorse, in cui si individuano le risorse estratte a partire dal diagramma delle classi del documento D3.

Nel capitolo quattro si spiega ciò che si è fatto con Swagger per la documentazione delle API.

Successivamente viene fornita una breve descrizione per le pagine implementate e una descrizione del repository di GitHub con le istruzioni per effettuare il deployment.

Per finire mostriamo i vari casi di test realizzati per verificare il corretto funzionamento delle API.

# 1. User-Flow

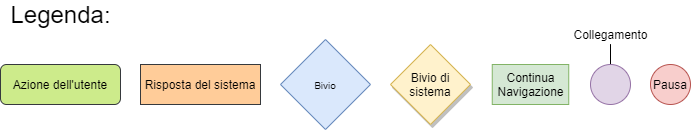
Riportiamo in seguito lo user-flow dell’applicazione, il quale descrive ciò che è possibile fare nell’implementazione descritta nel dettaglio nel seguente documento.

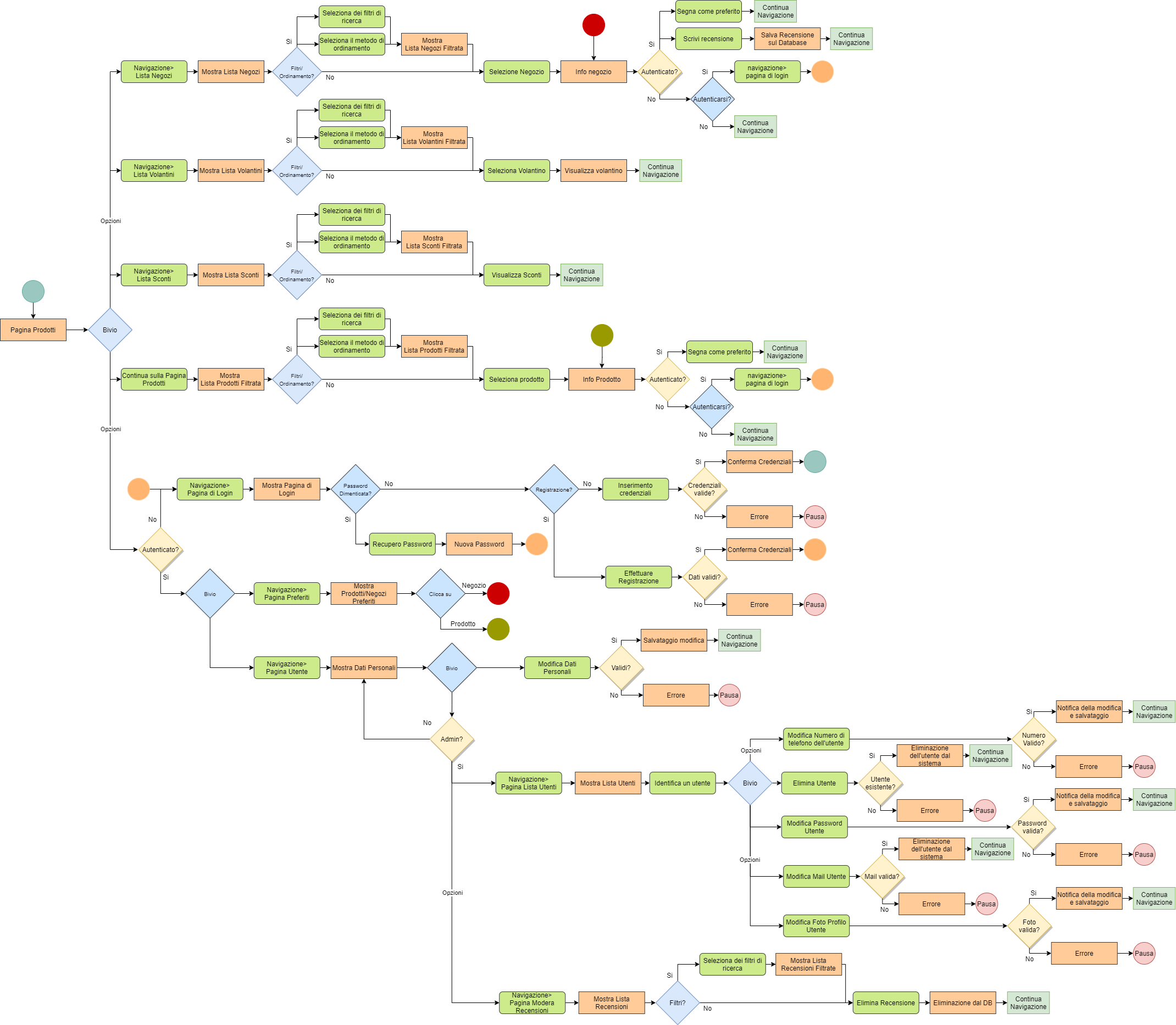
È stata riportata anche una didascalia dei vari componenti utilizzati nello user-flow.

Si può notare la presenza di due differenti tipi di bivi, quello semplice, che rappresenta una scelta dell’utente tra molteplici azioni fattibili in un determinato momento, e il bivio di sistema, che rappresenta differenti stati dell’applicazione (per esempio essere autenticato o meno) che fanno sì che si possano o non possano fare determinate azioni.

È presente, inoltre, un componente collegamento che ha lo scopo di collegare due zone molto distanti del diagramma. Infine vi è anche il componente “Continua Navigazione”, che indica un punto in cui l’utente può ritornare allo stato iniziale o andare in una qualunque delle pagine apribili nello stadio iniziale del sito.

Figura 1. User-flow Diagram





# 2. Implementazione dell’Applicazione e Documentazione

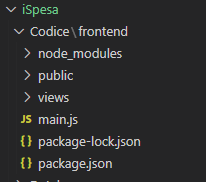
L’applicazione iSpesa è stata sviluppata utilizzando NodeJS per la parte di frontend. A livello di back-end per la memorizzazione dei vari dati abbiamo utilizzato MySQL.

Come si può notare dallo user-flow abbiamo sviluppato ogni aspetto dell’applicazione, tranne le parti relative ad Auth0. Purtroppo l’API esterna richiedeva un pagamento per utilizzare la funzione di 2FA quindi abbiamo implementato la parte di login e registrazione soltanto sul DB locale dell’applicazione. Abbiamo anche implementato una simulazione dell’acquisizione di dati da parte di un negozio.

Infine abbiamo anche utilizzato le API del sistema esterno GMail per gestire l’invio di email

agli utenti.

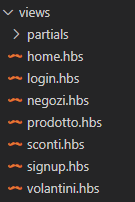
## 2.1 Struttura del progetto

La seguente immagine riporta la struttura principale dell’applicazione. Come si può notare la cartella principale è la cartella *Codice\_iSpesa* ed è così costituita:

* File *package.json* che rappresenta il file di configurazione generale del progetto.
* File *swagger.json*, dove sono descritte approfonditamente e con opportuni esempi le API implementate nel sistema. In esso è definita l’intera documentazione delle API da noi sviluppate.
* File *main.js*, che rappresenta il file principale del sistema, ovvero il file da eseguire per attivare la connessione verso il database MySQL e attivare il server all’indirizzo localhost:8080
* Cartella /*node\_modules*, dove sono scaricate le dipendenze utilizzate nel sistema
* Cartella /*public*, che contiene tutti quei file considerati “pubblici” (per esempio alcune immagini)
* Cartella /*views*, ovvero la cartella che contiene tutta la business logic del sistema dal lato front-end.

Nella cartella /*public* troviamo:

* La cartella /*css*, che contiene eventuali file .css utili per il front-end
* La cartella /*img*, che contiene tutte le immagini di dominio pubblico utili al sistema. Per esempio in essa sono contenute loghi, immagini del sito ed altro.

Nella cartella *views* sono presenti:

* Vari *file.hbs*, che sono il vero e proprio front-end dell’applicazione
* La cartella /*partials* che contiene vari file.hbs che vengono invocati su necessità.

# 3. Dipendenze

I moduli Node utilizzati e aggiunti al file package.json, nel campo dependencies, sono:

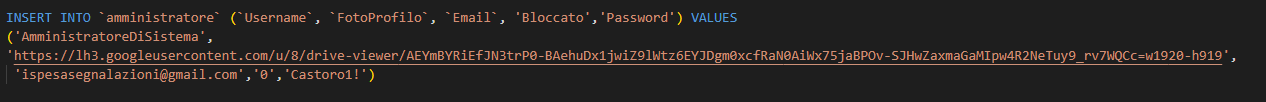
* *express*: framework che fornisce molte funzionalità per le web application come iSpesa, tra cui molte funzioni per creare e gestire le API;
* *express-handlebars*: un engine che ha lo scopo di far funzionare il framework express congli handlebars;
* *hbs*: un modulo usato per le handlebars (cheidere meglio Lanaro)
* *jest*: modulo usato per il testing delle API e delle funzioni nel back-end
* *supertest*: modulo usato per chiamare le API in fase di testing.

# 4. Modelli nel Database

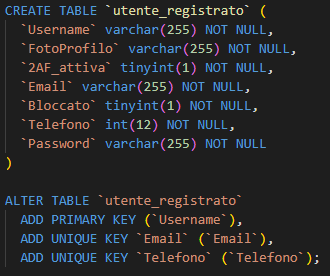
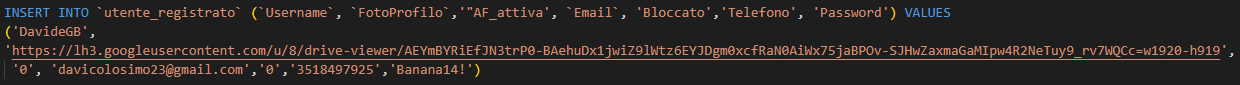
Per la gestione dei dati nell’applicazione abbiamo definito diversi modelli di dati partendo dalle classi sviluppate nel Diagramma delle Classi del Deliverable 3. Le risorse necessarie da gestire nel nostro sistema hanno portato alla definizione di cinque modelli, per ognuno dei quali è stata definita una precisa collezione nel database

## 4.1 Modello Amministratore

Per memorizzare i dati degli amministratori presenti nel nostro sito web abbiamo creato il modello ***amministratore***. Si pone necessaria la definizione degli attributi *Username*, *FotoProfilo*, *Email*, *Bloccato*, *Password* che hanno lo scopo di contenere tutti i dati degli amministratori come definito nel Diagramma delle Classi. Ovviamente la *Email* è unica e lo *Username* svolge il ruolo di chiave. Un esempio di elemento nella collezione nel database è il seguente:



## 4.2 Modello Utente Registrato

Per memorizzare i dati degli utenti registrati presenti nel nostro sito web abbiamo creato il modello ***utente\_registrato***. Si pone necessaria la definizione degli attributi *Username*, *FotoProfilo*, *Email*, *Bloccato*, *Telefono* e *Password* che hanno lo scopo di contenere tutti i dati degli Utenti come definito nel Diagramma delle Classi. Oltre che sulla chiave primaria, è presente un indice anche sull’attributo *Email* e sull’attributo *Telefono*. Ovviamente l’*Username* svolge il ruolo di chiave. Un esempio di elemento nella collezione nel database è il seguente:

## 4.3 Modello Categoria

Il modello ***categoria*** ha lo scopo di permettere il di associare ad ogni prodotto una categoria tra quelle predefinite. Si tratta, in pratica, di una sorta di enumerativo.

Si possono notare nell’inserimento tutte le categorie che riteniamo siano necessarie per l’applicazione. Nel caso in cui ne servano altre si espanderà semplicemente la lista.

## 4.4 Modello Negozio

Il modello ***negozio*** ha lo scopo di memorizzare i dati relativi ai vari negozi. Si pone necessaria la definizione degli attributi *Ubicazione*, *Orari*, *Nome*, *Logo* e *IDNegozio* che hanno lo scopo di contenere tutti i dati dei negozi come definito nel Diagramma delle Classi. Si può notare come la chiave sia l’*IDNegozio* che ha una funzione di Auto-Increment che permettere di usare senza problemi l’ID come chiave essendo sicuri della sua unicità. Un esempio di alcuni elementi nella collezione nel database è il seguente:

## 4.5 Modello Negozi Preferiti

Il modello ***negozipreferiti*** ha lo scopo di immagazzinare i dati relativi ai negozi preferiti di ogni utente. Si può notare, infatti, che è un’entità debole che dipende dalle entità ***negozio*** e ***utente\_registrato*** (ciò è evidenziato anche dalle regole ON DELETE CASCADE ON UPDATE CASCADE presenti nelle chiavi).

## 4.6 Modello Prodotto

Il modello ***prodotto*** ha lo scopo di contenere i dati relativi ai vari prodotti, seguendo quanto detto nel Diagramma delle Classi. Si possono identificare gli attributi *Nome*, *Immagine*, *Categoria*, *IDProdotto*e *NegozioProvenzienza*. L’attributo *IDProdotto*funge da chiave principale e viene incrementato automaticamente per ogni nuovo prodotto inserito nel database. Gli attributi *NegozioProvenienza*e *Categoria*sono chiavi esterne riferite ai modelli “***negozio***” e “***categoria***”. Nel caso in cui dovesse scomparire il negozio/la categoria del prodotto, anche il prodotto stesso verrebbe eliminato. Ecco un esempio di inserimento:

## 4.6 Modello Prodotti Preferiti

Il modello ***prodottipreferiti*** ha lo scopo di immagazzinare i dati relativi ai prodotti preferiti di ogni utente. Si può notare, infatti, che è un’entità debole che dipende dalle entità ***prodotto*** e ***utente*** (ciò è evidenziato anche dalle regole ON DELETE CASCADE ON UPDATE CASCADE presenti nelle chiavi).

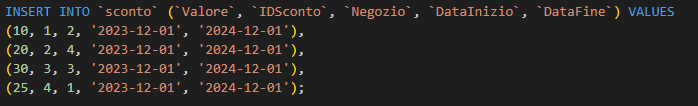
## 4.7 Modello Storico Prezzi

Il modello ***storicoprezzi*** ha lo scopo di immagazzinare i dati relativi al prezzo dei vari prodotti presenti nel database. Si può notare, infatti, che è un’entità debole che dipende dall’***entità prodotto*** (ciò è evidenziato anche dalle regole ON DELETE CASCADE ON UPDATE CASCADE presenti nella chiave). Il suo scopo principale è associare ad ogni prodotto una *data* ed un *prezzo*. Ecco un esempio di inserimento:



## 4.8 Modello Sconti

Il modello ***sconti*** ha lo scopo di immagazzinare i dati relativi agli sconti dei vari negozi integrati nel sistema iSpesa. Ogni sconto è identificabile da un *IDNegozio* che auto-incrementa e funge da chiave primaria. È presente una chiave esterna, ovvero *Negozio*, che ha lo scopo di associare allo sconto un ***negozio*** in cui esso è valido. Sono salvate sia la data d’inizio (*DataInizio*) che la data di scadenza (*DataFine*) dello sconto, mediante degli appositi attributi. Ovviamente nel caso in cui smettesse di esistere il negozio o la categoria a cui fanno riferimento, le entità di questo tipo verrebbero automaticamente eliminate Ecco un esempio di inserimento:



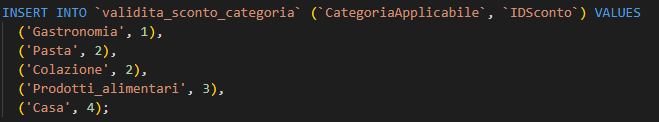
## 4.9 Modello Validità Sconto Prodotto

Il modello ***validità\_sconto\_prodotto*** ha lo scopo di salvare le informazioni relative a tutti quegli sconti che non si applicano ad una categoria di prodotti, ma a singoli prodotti (per esempio uno sconto soltanto sulla passata di pomodoro di una precisa marca). Si tratta di entità deboli, dipendenti totalmente dalle entità esterne ***sconto*** e ***prodotto*** che associano (ciò è evidenziato anche dalle regole ON DELETE CASCADE ON UPDATE CASCADE presenti nella chiave). Ecco un esempio di inserimento:



## 4.10 Modello Validità Sconto Categoria

Il modello ***validità\_sconto\_categoria*** ha lo scopo di salvare le informazioni relative a quale categoria di prodotti uno sconto si applica. Si tratta di entità deboli, dipendenti totalmente dalle entità esterne ***sconto*** e ***categoria*** che associano (ciò è evidenziato anche dalle regole ON DELETE CASCADE ON UPDATE CASCADE presenti nella chiave). Ecco un esempio di inserimento:



## 4.11 Modello Recensione

Il modello ***recensione*** è l’entità che ha lo scopo di salvare le recensioni all’interno del database. Ogni recensione è identificata da un attributo *IDRecensione* che viene incrementato automaticamente dal sistema. Come si può notare la recensione presenta gli attributi *Titolo*, *Testo*, N\_*stelle*, Data\_*creazione*, *Utente*, *Negozio* che ne caratterizzano ogni aspetto. È importante notare che nel caso in cui l’utente che ha scritto la recensione e/o il negozio da essa interessata venissero eliminati, allora anche la recensione verrebbe automaticamente eliminata. Ecco un esempio di inserimento:

## 4.12 Modello Volantino

Il modello ***volantino*** ha lo scopo di conservare nel database le informazioni relative ai volantini dei vari negozi. Sostanzialmente non fa altro che conservare il link al file del volantino che poi l’utente potrà visualizzare online. Ogni volantino è identificato dalla chiave *IDVolantino* che è viene incrementata autonomamente dal sistema. I volantini, inoltre, sono caratterizzati dagli attributi *Negozio*, *DataFine* e *VolantinoFile*. L’attributo *Negozio* è una chiave esterna e nel caso in cui il ***negozio*** a cui esso fa riferimento venisse eliminato, allora anche il volantino verrebbe autonomamente eliminato. Ecco un esempio di inserimento:

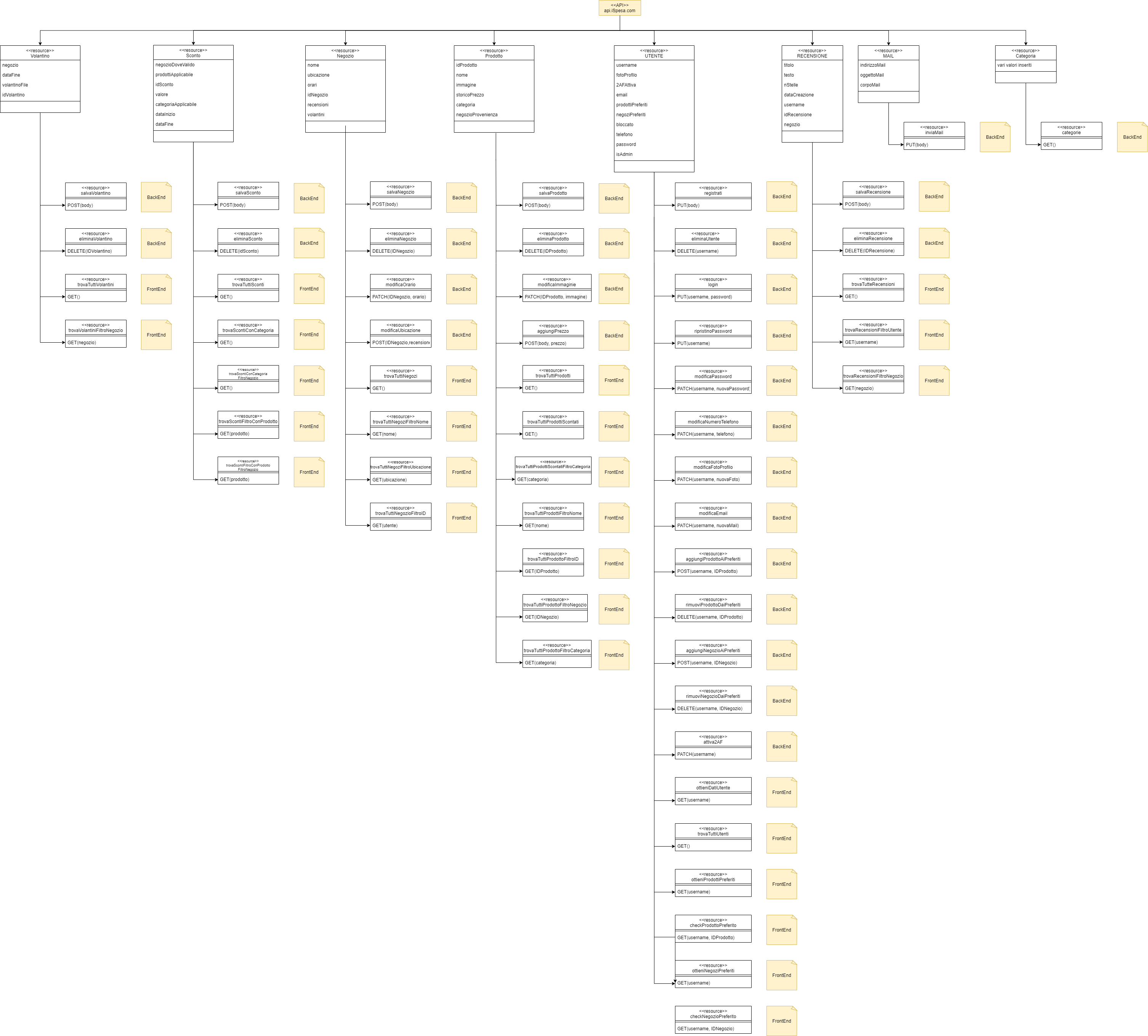
# 5. API del Progetto

In questa parte del documento vengono descritte le varie API implementate a partire dal diagramma delle classi del documento Deliverable 3. Useremo un diagramma per rappresentare l’estrazione delle risorse a partire dal diagramma delle classi e uno per rappresentare le risorse sviluppate.

## 5.1 Estrazione delle risorse dal Diagramma delle Classi

Questo diagramma mostra come abbiamo estratto le varie risorse sviluppate nel sistema a partire dal diagramma delle classi. Inizialmente le prime risorse che abbiamo individuato sono stati i modelli che abbiamo descritto nel capitolo Modelli nel Database. Alcuni dei modelli presenti sono stati riportati nel seguente diagramma come attributi dei modelli da cui dipendono. Per esempio non è presente il modello storico\_prezzi ma ogni prodotto ha un campo storicoPrezzo. A partire dalle classi presenti abbiamo individuato i “tipi di dato” che dovevano essere memorizzati nel database MySQL, preservandone gli attributi fondamentali. Infatti gli attributi specificati nel diagramma delle classi sono stati riportati anche nel diagramma delle risorse.

Successivamente abbiamo trasformato alcuni metodi delle classi in ulteriori risorse del nostro sistema. Le risorse a cui sono collegati i modelli sopra descritti sono nient’altro che le API che abbiamo sviluppato e coincidono con alcuni metodi delle classi. Non tutti i metodi sono chiaramente diventati API in quanto alcuni sono semplicemente delle piccole funzioni ausiliarie e di supporto alle API. Di queste risorse viene specificato il metodo (se si tratta di GET,PUT, POST, PATCH o DELETE a seconda del loro compito) e i parametri che richiedono per essere eseguiti (si è specificato body nel caso in cui servissero più di tre parametri in input per dire che quella data risorsa richiede tutti, o comunque molti, attributi del modello). Infine viene specificato anche se l’effetto di quella risorsa ha rilevanza nel front-end oppure nel back-end: per le varie risorse i tipo POST, PUT, PATCH e DELETE l’effetto è chiaramente sul back-end perché il loro compito è di salvare, modificare o eliminare una risorsa nel database, non fornendo informazioni in front-end oltre che un messaggio di conferma. Le risorse di tipo GET hanno invece un chiaro effetto sul front-end perché interrogano il database per chiedere alcune risorse e poi le mostrano nel front-end. Nella pagina seguente alleghiamo il diagramma che illustra quanto descritto, mentre una specifica più attenta delle API sarà fatta nel diagramma delle risorse.



# 6. Diagrammi delle Risorse

Nel seguente diagramma delle risorse rappresentiamo le varie API sviluppate nel progetto. Per permettere una visione più modulare e chiara, abbiamo diviso il diagramma delle risorse per modello.

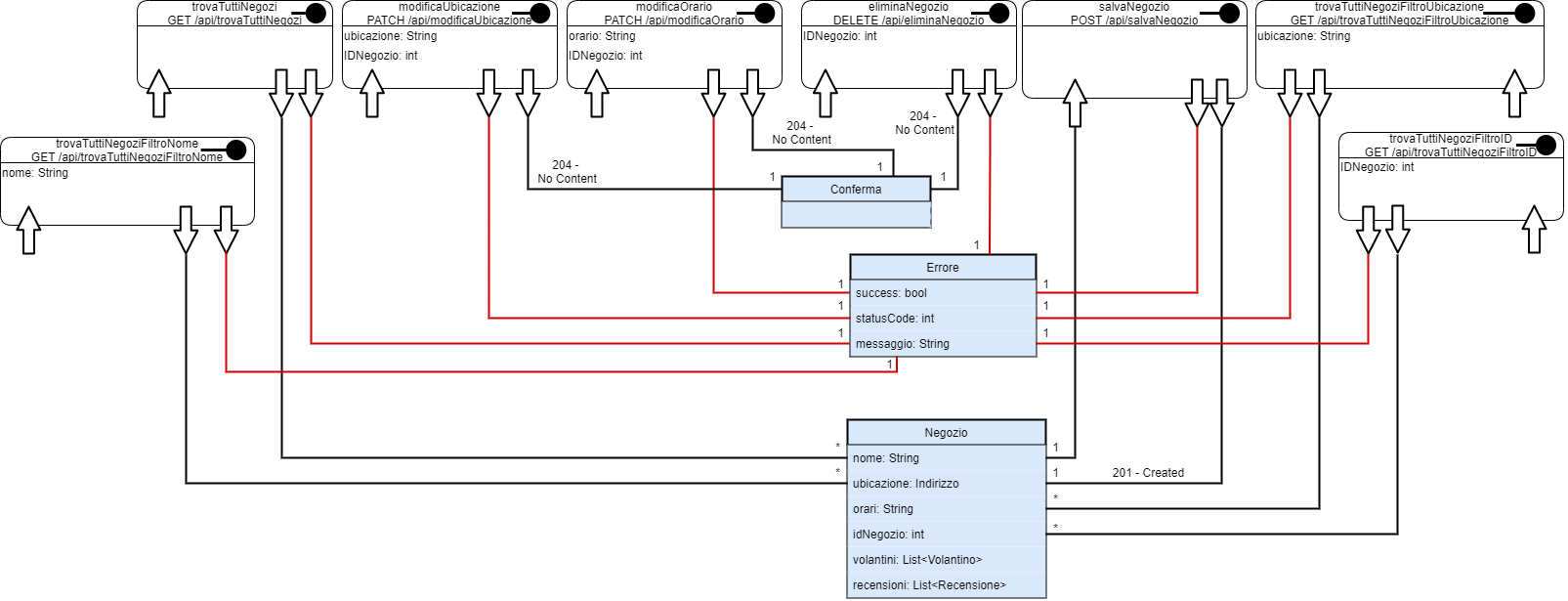
In tutti i diagrammi sono stati specificati gli input e gli output delle varie API. Le varie API possono ritornare errori di varia natura. Poiché non è possibile creare un output per ogni possibile errore, abbiamo decido di creare un output comune per tutti gli errori. In ogni API il messaggio di errore è stato standardizzato e comprende:

* un campo **success**, impostato a false;
* un campo **messaggio**, che spiega qual è stato il problema con l’esecuzione;
* un campo **statusCode**, che contiene il codice identificativo dell’errore.

Per le API che ritornano un codice **204 – No content** abbiamo deciso di creare un messaggio di conferma vuoto, in quanto appunto non ritornano nulla se non il codice 204 che conferma l’esecuzione dell’API con successo.

## 6.1 Diagramma delle Risorse 1: Prodotto

## 6.2 Diagramma delle Risorse 2: Negozio



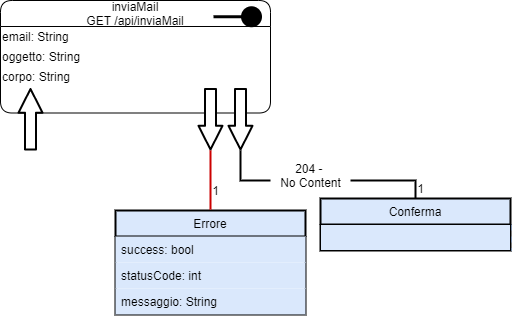
## 6.3 Diagramma delle Risorse 3: Sconto

## 6.4 Diagramma delle Risorse 4: Volantino

## 6.5 Diagramma delle Risorse 5: Utente

## 6.6 Diagramma delle Risorse 6: Recensioni

## 6.7 Diagramma delle Risorse 7: Mail



## 6.8 Diagramma delle Risorse 8: Categoria

# 7. Sviluppo API

In seguito descriveremo il funzionamento delle varie API sviluppate nel profetto e le funzion di MariaDB chiamate al loro interno. Non verrà allegato il codice in quanto sarebbe molto lungo può essere reperito facilmente nel file main.js.

# 7.1 API per il modello Utente

Di seguito descriviamo le API per gestire le varie azioni eseguibili sull’Utente

## 7.1.1 Login

L’API **login** ha lo scopo di validare le credenziali di un Utente per effettuare l’accesso sul sito. L’API riceve in input lo Username e la Password dell’Utente che desidera fare il login. Nel caso in cui le credenziali siano corrette restituisce come output un messaggio vuoto di conferma e modifica lo status dell’user. Nel caso in cui le credenziali siano errate oppure nel caso in cui ci siano degli errori, l’API restituisce dei messaggi d’errore.

## 7.1.2 Registrati

L’API **registrati** ha lo scopo di creare una nuova istanza della risorsa Utente e salvarla nel database. Riceve in input i dati dell’utente da memorizzare e ne controlla l’unicità. Nel caso in cui lo Username sia già presente nel database invia un errore come output, mentre se non ci sono problemi l’utente viene correttamente registrato nel sistema.

## 7.1.3 Elimina Utente

L’API **eliminaUtente** ha lo scopo di eliminare un’istanza della risorsa utente dal Database. Riceve come unico parametro di input lo Username dell’Utente da eliminare. Se non ci sono problemi l’API non ritornerà nulla mentre, se l’Utente che si sta cercando di eliminare non è presente sul database, ritornerà un messaggio d’errore con la spiegazione del problema.

## 7.1.4 Ripristino Password

L’API **ripristinoPassword** non è stata implementata poiché richiesto l’uso dell’API esterna Auth0 che non usiamo in questa implementazione come descritto in precedenza.

## 7.1.5 Attiva2AF

L’API **attiva2AF** non è stata pienamente implementata per via della mancanza dell’API esterna Auth0. Nella sua forma corrente l’API prende in input lo Username e la Mail dell’Utente desiderato e se i dati inseriti corrispondono a quelli di un utente esistente la 2FA verrà attivata per l’utente in questione. Nel caso in l’utente non sia presenta verrà inviato un messaggio d’errore.

## 7.1.6 Modifica Password

L’API **modificaPassword** ha lo scopo di modificare la password dell’utente desiderato. L’API prende in input lo Username dell’Utente, la Vecchia Password dell’Utente e la Nuova Password dell’Utente. Questa API funziona solo se l’utente che sta effettuando la richiesta di modificare è l’utente interessato dal cambiamento o un amministratore di sistema. Nel caso in cui la password vecchia inserita corrisponda a quella salvata sul database, la password dell’utente verrà aggiornata e non verrà data nessuna risposta. Nel caso in cui la password vecchia sia errata verrà inviato un messaggio d’errore 401 UNAUTHORIZED, mentre se lo Username non è presente nel database verrà mostrato un errore 404.

## 7.1.7 ModificaEmail

L’API **modificaEmail** ha la funzione di modificare l’Email di un Utente. Prende in input lo Username di un Utente e la Nuova Email da inserire al posto della precedenza. Questa API funziona solo se l’utente che sta effettuando la richiesta di modificare è l’utente interessato dal cambiamento o un amministratore di sistema. Nel caso in cui l’utente sia presente sul database l’Email verrà aggiornata, mentre se l’utente non è presente verrà inviato un messaggio d’errore.

## 7.1.8 Modifica Numero Telefono

L’API **modificaNumeroTelefono** ha la funzione di modificare il numero di telefono di un Utente. Prende in input lo Username di un Utente e il nuovo Numero di Telefono da inserire al posto del precedente. Questa API funziona solo se l’utente che sta effettuando la richiesta di modificare è l’utente interessato dal cambiamento o un amministratore di sistema. Nel caso in cui l’utente sia presente sul database il Numero di Telefono verrà aggiornato, mentre se l’utente non è presente verrà inviato un messaggio d’errore.

## 7.1.9 Modifica Foto Profilo

L’API **modificaFotoProfilo** ha la funzione di modificare la Foto Profilo di un Utente. Prende in input lo Username di un Utente e la nuova Foto Profilo (sotto forma di URL) da inserire al posto della precedente. Questa API funziona solo se l’utente che sta effettuando la richiesta di modificare è l’utente interessato dal cambiamento o un amministratore di sistema. Nel caso in cui l’utente sia presente sul database la Foto Profilo verrà aggiornata, mentre se l’utente non è presente verrà inviato un messaggio d’errore.

## 7.1.10 Aggiungi Prodotto ai Preferiti

L’API **aggiungiProdottoaiPreferiti** ha lo scopo di aggiungere un Prodotto alla lista preferiti di un Utente. Prende in input lo Username dell’Utente e l’ID del Prodotto in questione. Se lo Username non corrisponde a nessuno di quelli salvati sul database o se l’ID del Prodotto non corrisponde a nessun Prodotto, verrà mostrato un messaggio d’errore 404. Se non ci sono problemi la modifica avverrà senza restituire nulla.

## 7.1.11 Rimuovi Prodotto dai Preferiti

L’API **rimuoviProdottoDaiPreferiti** ha lo scopo di rimuovere un Prodotto dalla lista preferiti di un Utente. Prende in input lo Username dell’Utente e l’ID del Prodotto in questione. Se lo Username non corrisponde a nessuno di quelli salvati sul database o se l’ID del Prodotto non corrisponde a nessun Prodotto, verrà mostrato un messaggio d’errore 404. Se non ci sono problemi la modifica avverrà senza restituire nulla.

## 7.1.12 Aggiungi Negozio ai Preferiti

L’API **aggiungiNegozioAiPreferiti** ha lo scopo di aggiungere un Negozio alla lista preferiti di un Utente. Prende in input lo Username dell’Utente e l’ID del Negozio in questione. Se lo Username non corrisponde a nessuno di quelli salvati sul database o se l’ID del Negozio non corrisponde a nessun Negozio, verrà mostrato un messaggio d’errore 404. Se non ci sono problemi la modifica avverrà senza restituire nulla.

## 7.1.13 Rimuovi Negozio dai Preferiti

L’API **rimuoviNegozioDaiPreferiti** ha lo scopo di rimuovere un negozio dalla lista preferiti di un Utente. Prende in input lo Username dell’Utente e l’ID del Negozio in questione. Se lo Username non corrisponde a nessuno di quelli salvati sul database o se l’ID del Negozio non corrisponde a nessun Negozio, verrà mostrato un messaggio d’errore 404. Se non ci sono problemi la modifica avverrà senza restituire nulla.

## 7.1.14 Ottieni Dati utente

L’API **ottieniDatiUtente** ha lo scopo di restituire i dati personali dell’Utente desiderato. L’API prende in input lo Username dell’Utente e se corrisponde ad uno di quelli salvati nel database verranno restituiti i relativi dati. Nel caso in cui non esistano utenti con tale Username, verrà inviato un messaggio d’errore.

## 7.1.15 Trova Tutti Utenti

L’API **trovaTuttiUtenti** ha la funzione di restituire come output la lista degli Utenti presenti nel database. Non ha parametri di input e non può avere come output errori. Nel caso in cui nessun Utente sia presente la lista sarà semplicemente vuota.

## 7.1.16 Ottieni Prodotti Preferiti

L’API **ottieniProdottiPreferiti** ha lo scopo di restituire in output la lista dei Prodotti preferiti da un Utente. Prende in input lo Username dell’Utente e, se esso è presente nel database, restituisce la lista di Prodotti preferiti ad esso associati. Nel caso in cui non ci sia alcun utente con tale Username verrà mostrato un messaggio d’errore.

## 7.1.17 Check Prodotto Preferito

L’API **checkProdottoPreferito** ha lo scopo di controllare se un Prodotto è presente nella lista preferiti di un Utente. L’API prende in input lo Username dell’Utente interessato e l’ID del Prodotto in questione e restituisce in output il risultato del controllo. Nel caso in cui nessun Utente salvato sul database abbia lo Username desiderato, verrà mostrato un messaggio d’errore.

## 7.1.18 Ottieni Negozi Preferiti

L’API **ottieniNegoziPreferiti** ha lo scopo di restituire in output la lista dei Negozi preferiti da un Utente. Prende in input lo Username dell’Utente e, se esso è presente nel database, restituisce la lista di Negozi preferiti ad esso associati. Nel caso in cui non ci sia alcun utente con tale Username verrà mostrato un messaggio d’errore.

## 7.1.19 Check Negozi oPreferito

L’API **checkNegozioPreferito** ha lo scopo di controllare se un Negozio è presente nella lista preferiti di un Utente. L’API prende in input lo Username dell’Utente interessato e l’ID del Negozio in questione e restituisce in output il risultato del controllo. Nel caso in cui nessun Utente salvato sul database abbia lo Username desiderato, verrà mostrato un messaggio d’errore.

# 7.2 API per il modello Volantino

Di seguito descriviamo le API per gestire le varie azioni eseguibili sui Volantini.

## 7.2.1 Salva Volantino

L’API **salvaVolantino** ha lo scopo di creare una nuova istanza della risorsa Volantino e salvarla nel database. Riceve in input l’ID del Negozio a cui il Volantino da riferimento, la Dati di Scadenza del Volantino e un URL che contiene il file effettivo. Salvo errori del server, l’API crea il volantino, lo salva ed invia come output non messaggio 201 Created.

## 7.2.2 Elimina Volantino

L’API **eliminaVolantino** ha la funzione di eliminare un Volantino salvato sul database. Prende come input l’ID del Volantino e se non ci sono problemi lo elimina dal database. Nel caso in cui l’ID del Volantino non sia presente nel database, un messaggio d’errore 404 verrà inviato come output.

## 7.2.3 Trova Tutti Volantini

L’API **trovaTuttiVolantini** ha lo scopo di restituire la lista di tutti i Volantini presenti nel database. Non ha parametri di input e non può avere come output errori. Nel caso in cui nessun Volantino sia presente la lista sarà semplicemente vuota.

## 7.2.4 Trova Volantini Filtro Negozio

L’API **trovaVolantiniFiltroNegozio** permette di ottenere la lista di tutti i Volantini che si riferiscono ad un particolare Negozio. Prende in input l’ID di un Negozio e restituisce la lista dei Volantini ad esso associato. Se non ci sono Volantini associati la lista sarà semplicemente vuota.

# 7.3 API per il modello Sconto

Di seguito descriviamo le API per gestire le varie azioni eseguibili sugli Sconti.

## 7.3.1 Salva Sconto

L’API **salvaSconto** ha lo scopo di creare una nuova istanza della risorsa Sconto e salvarla nel database. Riceve in input il Valore dello Sconto, l’ID del Negozio a cui fa riferimento, la Data d’Inizio dello Sconto e la Data di Fine. L’API restituisce come output un messaggio 201 CREATED dopo aver creato e salvato lo Sconto sul database. Poiché l’ID dello Sconto è autoincrementale e gestito dal database non ci possono essere collisioni.

## 7.3.2 Elimina Sconto

L’API **eliminaSconto** ha la funzione di eliminare uno Sconto dal database. Prende come input l’ID dello Sconto desiderato e se è presente sul database lo elimina inviando un messaggio 204 No Content di conferma. Nel caso in cui non siano presenti Sconti con tale ID, un messaggio 400 Bad Request verrà inviato.

## 7.3.3 Trova Tutti Sconti

L’API **trovaTuttiSconti** ha lo scopo di restituire la lista di tutti gli Sconti presenti nel database. Non ha parametri di input e non può avere come output errori. Nel caso in cui nessuno Sconto sia presente la lista sarà semplicemente vuota.

## 7.3.4 Trova Sconti con Categoria

L’API **trovaScontiConCategoria** ha la funzione di ritornare tutti gli Sconti presenti nel database con la relativa Categoria associata. Non ha parametri di input e non può avere come output errori. Nel caso in cui nessuno Sconto sia presente la lista sarà semplicemente vuota.

## 7.3.5 Trova Sconti con Categoria Filtro Negozio

L’API **trovaScontiConCategoriaFiltroNegozio** ha la funzione di ritornare tutti gli Sconti presenti nel database con la relativa Categoria associata e che si riferiscono ad un Negozio in particolare. Ha come parametro di input l’ID del Negozio interessato e non può avere come output errori. Nel caso in cui nessuno Sconto sia presente la lista sarà semplicemente vuota.

## 7.3.6 Trova Sconti con Prodotto

L’API **trovaScontiConProdotto** ha la funzione di ritornare tutti gli Sconti presenti nel database con i relativi Prodotti associati. Non ha parametri di input e non può avere come output errori. Nel caso in cui nessuno Sconto sia presente la lista sarà semplicemente vuota.

## 7.3.5 Trova Sconti con Prodotto Filtro Negozio

L’API **trovaScontiConProdottoFiltroNegozio** ha la funzione di ritornare tutti gli Sconti presenti nel database con i relativi Prodotti associati e che si riferiscono ad un Negozio in particolare. Ha come parametro di input l’ID del Negozio interessato e non può avere come output errori. Nel caso in cui nessuno Sconto sia presente la lista sarà semplicemente vuota.

# 7.4 API per il modello Negozi

Di seguito descriviamo le API per gestire le varie azioni eseguibili sui Negozi.

## 7.4.1 Salva Negozio

L’API **salvaNegozio** ha lo scopo di creare una nuova istanza della risorsa Negozio e salvarla nel database. Riceve in input il Nome del Negozio, il suo Logo (sotto forma di URL), l’Ubicazione dello stesso e gli Orari di apertura. L’API restituisce come output un messaggio 201 CREATED dopo aver creato e salvato il Negozio sul database. Poiché l’ID del Negozio è autoincrementale e gestito dal database non ci possono essere collisioni.

## 7.4.2 Elimina Negozio

L’API **eliminaNegozio** ha la funzione di eliminare un Negozio dal database. Prende come input l’ID del Negozio desiderato e se è presente sul database lo elimina inviando un messaggio 204 No Content di conferma. Nel caso in cui l’Utente che provi ad eseguire l’azione non sia un Amministratore, un messaggio d’errore 403 Forbidden verrà mostrato. Nel caso in cui non vi siano Negozi salvati nel database con l’ID desiderato, un messaggio d’errore 400 Bad Request verrà inviato.

## 7.4.3 Modifica Orari

L’API **modificaOrari** ha lo scopo di modificare l’orario di un Negozio. Prende come input l’ID del Negozio desiderato e il nuovo Orario e, se non ci sono problemi, salva la modifica sul database e successivamente invia un messaggio 204 No Content di conferma. Nel caso in cui l’Utente che provi ad eseguire l’azione non sia un Amministratore, un messaggio d’errore 403 Forbidden verrà mostrato. Nel caso in cui non vi siano Negozi salvati nel database con l’ID desiderato o nel caso in cui l’Orario non sia valido, un messaggio d’errore 400 Bad Request verrà inviato.

## 7.4.4 Modifica Ubicazione

L’API **modificaUbicazione** ha lo scopo di modificare l’ubicazione di un Negozio. Prende come input l’ID del Negozio desiderato e la nuova Ubicazione e, se non ci sono problemi, salva la modifica sul database e successivamente invia un messaggio 204 No Content di conferma. Nel caso in cui l’Utente che provi ad eseguire l’azione non sia un Amministratore, un messaggio d’errore 403 Forbidden verrà mostrato. Nel caso in cui non vi siano Negozi salvati nel database con l’ID desiderato o nel caso in cui l’Ubicazione non sia valida, un messaggio d’errore 400 Bad Request verrà inviato.

## 7.4.5 Trova Tutti Negozi

L’API **trovaTuttiNegozi** ha lo scopo di restituire la lista di tutti i Negozi presenti nel database. Non ha parametri di input e non può avere come output errori. Nel caso in cui nessun Negozio sia presente la lista sarà semplicemente vuota.

## 7.4.6 Trova Tutti Negozi Filtro Nome

L’API **trovaTuttiNegoziFiltroNome** ha lo scopo di restituire la lista di tutti i Negozi presenti nel database con un nome specifico. L’unico parametro di input è il nome che si desidera cercare e non può avere come output errori. Nel caso in cui nessun Negozio col nome desiderato sia presente la lista sarà semplicemente vuota.

## 7.4.7 Trova Tutti Negozi Filtro Ubicazione

L’API **trovaTuttiNegoziFiltroUbicazione** ha lo scopo di restituire la lista di tutti i Negozi presenti nel database situati in un luogo specifico. L’unico parametro di input è l’ubicazione che si desidera cercare e non può avere come output errori. Nel caso in cui nessun Negozio ubicato nel luogo specificato sia presente la lista sarà semplicemente vuota.

## 7.4.8 Trova Tutti Negozi Filtro ID

L’API **trovaTuttiNegoziFiltroID** ha lo scopo di restituire i dati del Negozio con l’ID desiderato. L’unico parametro di input è l’ID che si desidera cercare e non si possono avere output d’errore. Nel caso in cui nessun Negozio abbia l’ID desiderato verranno semplicemente mostrati dati vuoti.

# 7.5 API per il modello Prodotti

Di seguito descriviamo le API per gestire le varie azioni eseguibili sui Prodotti.

## 7.5.1 Salva Prodotto

L’API **salvaProdotto** ha lo scopo di creare una nuova istanza della risorsa Prodotto e salvarla nel database. Riceve in input il Nome del Prodotto, la sua Immagine (sotto forma di URL), la Categoria dello stesso e l’ID del Negozio in cui è presente il Prodotto. L’API restituisce come output un messaggio 201 CREATED dopo aver creato e salvato il Prodotto sul database. Poiché l’ID del Prodotto è autoincrementale e gestito dal database non ci possono essere collisioni. Nel caso in cui l’Utente che provi ad eseguire l’azione non sia un Amministratore, un messaggio d’errore 403 Forbidden verrà mostrato. Nel caso in cui uno o più parametri non siano validi, un messaggio d’errore 400 Bad Request verrà inviato.

## 7.5.2 Elimina Prodotto

L’API **eliminaProdotto** ha la funzione di eliminare un Prodotto dal database. Prende come input l’ID del Prodotto desiderato e se è presente sul database lo elimina inviando un messaggio 204 No Content di conferma. Nel caso in cui non vi siano Prodotti salvati nel database con l’ID desiderato, un messaggio d’errore 400 Bad Request verrà inviato.

## 7.5.3 Modifica Immagine

L’API **modificaImmagine** ha lo scopo di modificare l’immagine di un Prodotto. Prende come input l’ID del Prodotto desiderato e la nuova Immagine e, se non ci sono problemi, salva la modifica sul database e successivamente invia un messaggio 204 No Content di conferma. Nel caso in cui non vi siano Prodotti salvati nel database con l’ID desiderato o nel caso in cui l’Immagine non sia valido, un messaggio d’errore 400 Bad Request verrà inviato.

## 7.5.4 Aggiungi Prezzo

L’API **aggiungiPrezzo** ha lo scopo di aggiungere un prezzo allo storico prezzo di un Prodotto. Prende come input l’ID del Prodotto desiderato e il prezzo e, se non ci sono problemi, salva la modifica sul database e successivamente invia un messaggio 204 No Content di conferma. Nel caso in cui non vi siano Prodotti salvati nel database con l’ID desiderato o nel caso in cui il prezzo non sia valido, un messaggio d’errore 400 Bad Request verrà inviato.

## 7.5.5 Trova Tutti Prodotti

L’API **trovaTuttiProdotti** ha lo scopo di restituire la lista di tutti i Prodotti presenti nel database. Non ha parametri di input e non può avere come output errori. Nel caso in cui nessun Prodotto sia presente la lista sarà semplicemente vuota.

## 7.5.6 Trova Tutti Prodotti Scontati

L’API **trovaTuttiProdottiScontati** ha lo scopo di restituire la lista di tutti i Prodotti presenti nel database su cui vi è uno sconto. Non ha parametri di input e non può avere come output errori. Nel caso in cui nessun Prodotto scontato sia presente la lista sarà semplicemente vuota.

## 7.5.7 Trova Tutti Prodotti Scontati Filtro Categoria

L’API **trovaTuttiProdottiScontatiFiltroCategoria** ha lo scopo di restituire la lista di tutti i Prodotti presenti nel database su cui vi è uno sconto e che appartengono ad una Categoria particolare. Ha come unico parametro di input la Categoria interessata e restituisce la lista dei Prodotti scontati che appartengo a tale Categoria. Nel caso in cui non ci siano Prodotti che rispondo ai requisiti, verrà visualizzata una lista vuota.

## 7.5.8 Trova Prodotti Filtro Nome

L’API **trovaProdottiFiltroNome** ha lo scopo di restituire la lista di tutti i Prodotti presenti nel database che posseggono un nome specifico. Ha come unico parametro di input il nome interessato e restituisce la lista dei Prodotti con tale nome. Nel caso in cui non ci siano Prodotti che rispondo ai requisiti, verrà visualizzata una lista vuota.

## 7.5.9 Trova Prodotto Filtro ID

L’API **trovaProdottoFiltroID** ha lo scopo di restituire i dati di un Prodotto con un determinato ID. Ha come unico parametro di input l’ID del Prodotto da cercare e restituisce come output i dati del Prodotto con tale ID. Nel caso in cui non ci siano Prodotti con l’ID desiderato verranno visualizzati dati vuoti

## 7.5.10 Trova Prodotti Filtro Negozio

L’API **trovaProdottiFiltroNegozio** ha lo scopo di restituire la lista dei Prodotti salvati nel database presenti in un determinato Negozio. Ha come unico valore di input l’ID del Negozio desiderato e restituisce come output la lista dei Prodotti presenti nel Negozio desiderato. Nel caso in cui non ci siano Prodotti che rispettano i requisiti verrà visualizzata una lista vuota.

## 7.5.11 Trova Prodotti Filtro Categoria

L’API **trovaProdottiFiltroCategoria** ha lo scopo di restituire la lista dei Prodotti salvati nel database che appartengono ad una determinata categoria. Ha come unico valore di input la Categoria desiderata e restituisce come output la lista dei Prodotti appartenenti a tale categoria. Nel caso in cui non ci siano Prodotti che rispettano i requisiti verrà visualizzata una lista vuota.

# 7.6 API per il modello Recensioni

Di seguito descriviamo le API per gestire le varie azioni eseguibili sulle Recensioni.

## 7.6.1 Salva Recensione

L’API **salvaRecensione** ha lo scopo di creare una nuova istanza della risorsa Recensione e salvarla nel database. Riceve in input lo Username dell’Utente che effettua la Recensione, il Titolo della Recensione, il Numero di Stelle della Recensione, il testo della Recensione e l’ID del Negozio a cui la Recensione si riferisce. Nel caso in cui non ci siano errori la Recensione verrà salvata sul database e un messaggio di conferma 201 CREATED verrà inviato. Nel caso in cui un la Recensione venga fatta da un Utente non registrato (ovvero con Username null) verrà inviato un messaggio d’errore 403 FORBIDDEN.

## 7.6.2 Elimina Recensione

L’API **eliminaRecensione** ha la funzione di eliminare una Recensione dal database. Prende come input l’ID della Recensione e, se è presente sul database, la elimina inviando un messaggio 204 No Content di conferma. Nel caso in cui non ci siano Recensioni con l’ID desiderato verrà inviato un messaggio d’errore 400 Bad Request. Nel caso in cui un Utente diverso dal creatore della Recensione o un Utente che non è Amministratore provi a cancellare la Recensione, un messaggio d’errore 403 Forbidden verrà inviato.

## 7.6.3 Trova Recensioni Filtro Utente

L’API **trovaRecensioniFiltroUtente** ha lo scopo di restituire la lista delle Recensioni inviate da un determinante Utente. Ha come parametro di input lo Username dell’Utente desiderato e non mostra messaggi d’errore. Nel caso in cui l’Utente desiderato non abbia mai fatto Recensioni, una lista vuota verrà mostrata.

## 7.6.4 Trova Recensioni Filtro Negozio

L’API **trovaRecensioniFiltroNegozio** ha lo scopo di restituire la lista delle Recensioni appartenenti ad un determinato Negozio. Ha come parametro di input l’ID del Negozio desiderato e non mostra messaggi d’errore. Nel caso in cui il Negozio desiderato non abbia Recensioni, verrà visualizzata una lista vuota.

## 7.6.5 Trova Tutte Recensioni

L’API **trovaTutteRecensioni** ha lo scopo di restituire la lista di tutte le Recensioni presenti sul database. Non ha parametri di input e non può inviare messaggi d’errore. Nel caso in cui non ci siano Recensioni, una lista vuota verrà inviata.

# 7.7 API per il modello Mail

Di seguito descriviamo le API per gestire le varie azioni eseguibili sulle Recensioni.

## 7.7.1 Invia Mail

L’API **inviaMail** ha lo scopo di inviare Mail agli Utenti. Non è stata implementata.

# 7.8 API per il modello Categorie

Di seguito descriviamo le API per gestire le varie azioni eseguibili sulle Categorie.

## 7.8.1 Categorie

L’API **categorie** ha lo scopo di ottenere tutte le Categorie presenti nel database. Non ha né parametri di input né output d’errore.

# 8 Implementazione del FrontEnd

In questa sezione vengono descritte in dettaglio le varie schermate del front end che abbiamo implementato con le rispettive azioni che possono essere eseguite su di esse.

## 8.1 Header

L’header è l’elemento che si trova sopra ad ogni pagina del sito, eccetto le pagine *login*, *signup* e *password\_dimenticata*.

Ci sono due tipi di header:

* **Header**: Elenca le varie pagine accessibili dall’utente non registrato, cioè le pagine *home* (accessibile attraverso il logo), *sconti*, *volantini*, *negozi*, *login*.
* **Header loggato**: Elenca le varie pagine accessibili dall’utente registrato, cioè le pagine *home*, *sconti*, *volantini*, *negozi*, *preferiti*, *area personale* e *logout*

## 8.2 Home

La pagina home è la prima pagina da cui si comincia la navigazione al sito.

In cima alla schermata si trova una slideshow dove sono mostrati tre prodotti scelti.

Sotto alla slideshow si trova una barra di navigazione contenente tutte le categorie dei prodotti presenti nel sito e una barra di ricerca per trovare il prodotto.

Sotto alla barra di navigazione si trovano un massimo di 18 prodotti. I prodotti scontati sono mostrati prima di quelli non scontati.

In ogni prodotto si può visualizzare il nome, prezzo prima dello sconto (se scontati) ed il prezzo attuale. Tutti i prezzi sono mostrati al kilo.

## 8.3 Login

Questa pagina è da dove l’utente non registrato potrà inserire le sue credenziali per il login (Username e Password) per accedere al sito. Se le credenziali inserite sono quelle di un admin, l’utente avrà accesso ad altre funzionalità del sito.

L’utente può accedere ad altre due pagine da questa schermata: la pagina di registrazione e la pagina per la password dimenticata.

Se le credenziali inserite sono errate, l’utente verrà notificato. Se le credenziali sono corrette verrà reindirizzato alla pagina home.

## 8.4 Password dimenticata

Fai te davide

## 8.5 Registrazione

Questa pagina è da dove l’utente potrà inserire le sue credenziali (Username, Email, Numero di telefono e Password) per registrarsi al sito.

Se le credenziali inserite non soddisfano i requisiti, l’utente verrà notificato. Se le credenziali inserite sono corrette l’account verrà creato e si verrà reindirizzati alla pagina di login.

L’utente può tornare alla pagina di login da questa pagina.

## 8.6 Sconti

In questa pagina vengono elencati i vari sconti del sito offerti dai negozi.

In cima alla pagina sconti viene fornita una barra di navigazione per filtrare i vari negozi. Vengono poi mostrati fino a 18 sconti offerti dai vari negozi, con la loro categoria e valore in percentuale.

## 8.7 Volantini

In questa pagina vengono elencati i vari volantini del sito offerti dai negozi.

In cima alla pagina volantini viene fornita una barra di navigazione per filtrare i vari negozi.

Vengono poi mostrati fino a 18 volantini offerti dai vari negozi e la loro data di scadenza. Si può decidere se aprire o scaricare il volantino.

## 8.8 Negozi

In questa pagina vengono elencati i vari negozi del sito.

In cima alla pagina negozi viene fornita una barra di ricerca per trovare il negozio.

Vengono poi mostrati i vari negozi usati dal sito con i loro orari. Cliccando su qualsiasi negozio si accederà alla propria pagina.

## 8.9 Negozio

In questa pagina vengono elencate le informazioni di un determinato negozio.

A sinistra della pagina si trova il logo del negozio. Se la pagina è accessa da un utente registrato si potrà anche visualizzare una form per inviare una recensione.

Nella recensione si potrà scrivere un titolo, testo ed elencare il numero di stelle.

In cima alla parte destra della pagina si elencano il nome del negozio, l’orario e l’ubicazione. Se la pagina è accessa da un utente registrato si potrà anche visualizzare una stella vuota o piena, per visualizzare se il negozio è tra i preferiti o meno. Cliccando sulla stella si potrà togliere o aggiungere il sito tra i preferiti.

Sotto a questa parte si trova l’elenco delle recensioni del negozio, dove si può visualizzare la foto profilo e il nome dell’utente che ha inviato la recensione, il numero di stelle, il titolo, il testo e quando è stata pubblicata.

## 8.10 Preferiti

In questa pagina vengono elencati i negozi o prodotti preferiti dell’utente.

Questa pagina è accessibile solamente da un utente registrato, senò si verrà reindirizzati alla pagina home.

Da questa pagina l’utente può cliccare sui prodotti o negozi preferiti per visualizzare la pagina collegata.

## 8.11 Area Personale

In questa pagina vengono elencate le varie informazioni dell’utente.

Questa pagina è accessibile solamente da un utente registrato, senò si verrà reindirizzati alla pagina home.

In questa pagina l’utente può visualizzare le proprie informazioni come: Nome utente, Email, Numero di telefono, Foto Profilo.

L’utente può modificare la propria Email, Numero di telefono e password.

L’utente può attivare l’autenticazione a due fattori da questa pagina.

Se l’utente registrato è un admin, verranno anche visualizzati due bottoni che lo reindirizzano alla vista admin relativa agli utenti e alle recensioni.

## 8.12 Vista admin utenti

In questa pagina vengono elencati gli utenti del sito.

Questa pagina è accessibile solamente da un admin, senò si verrà reindirizzati alla pagina home.

In cima alla pagina si trova una barra di ricerca per trovare un determinato utente.

L’admin vedrà tutte le informazioni relative ad un utente come: Username, FotoProfilo, 2AF\_attiva, Email, Telefono, Password.

L’admin può decidere se modificare FotoProfilo, Email, Telefono o Password.

L’admin può anche eliminare l’utente con un bottone situato in fondo alla riga relativa all’utente.

## 8.13 Vista admin recensioni

In questa pagina vengono elencate le recensioni inserite nel sito.

Questa pagina è accessibile solamente da un admin, senò si verrà reindirizzati alla pagina home.

In cima alla pagina si trova una barra di ricerca per trovare le recensioni di un determinato negozio.

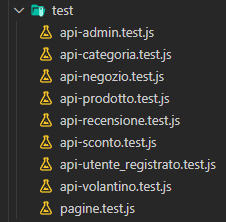
L’admin vedrà tutte le informazioni relative ad una recensionie come: IDRecensione, Negozio, Titolo, Testo, N\_stelle, Data.

L’admin può decidere se eliminare la recensione con un bottone situato in fondo alla riga relativa alla recensione.

# 9 Testing

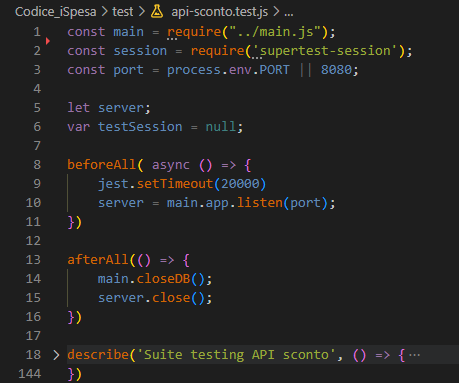
Per effettuare il testing abbiamo utilizzato la libreria Jest e il modulo supertest-session.

Tutto il testing effettuato è organizzato nella cartella test/ qua sotto descritta



In questa cartella ci sono 7 file .test.js che sono usati per testare le varie API del nostro sito. I file sono categorizzati in base ai test per i vari elementi che caratterizzano il sito iSpesa.

Vediamo come è strutturato un tipico file .test.js della nostra cartella, prendendo come esempio api-sconto.test.js



All’inizio importiamo il modulo supertest-session, che ci servirà per mantenere una sessione di login utente/admin nel sito durante il testing.

Nel metodo beforeAll() definiamo il tempo massimo che jest può aspettare per ogni test e apriamo un’istanza del server iSpesa da usare come test.

Nel metodo afterAll() invece chiudiamo il database dentro all’istanza del server iSpesa creata e poi chiudiamo l’istanza stessa.

Ogni API viene testata nella corrispettiva suite, definita dal metodo describe().

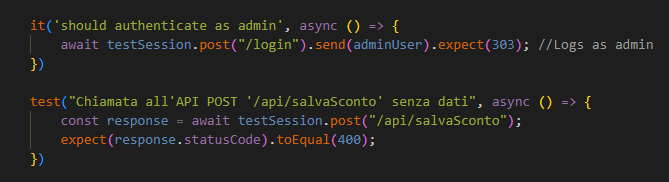


Dentro al metodo describe() per prima cosa definiamo l’inputBody, che in questo caso corrisponde allo scheletro dell’elemento Sconto.

Definiamo poi l’utente admin root, che viene usato per fare i test.

Viene poi avviata una sessione con il server attraverso il metodo session(URL).

Andiamo a vedere come viene testata una API.



Inanzitutto ci autentichiamo al server nel metodo it(), inviando i dati dell’admin utente all’api /login. La sessione porterà con sè l’autenticazione dell’admin, con la quale possiamo fare altre richieste.

Nel metodo test() chiamiamo l’API ‘/api/salvaSconto’ senza inviare i relativi dati. Da questa chiamata ci aspettiamo un messaggio di errore dal metodo ‘toEqual(400)’ equivalente a ‘400’.

## Risultati del testing

Per effettuare il testing abbiamo innanzitutto aggiunto il seguente script al file package.json: "test": "jest --coverage –detectOpenHandles --silent"

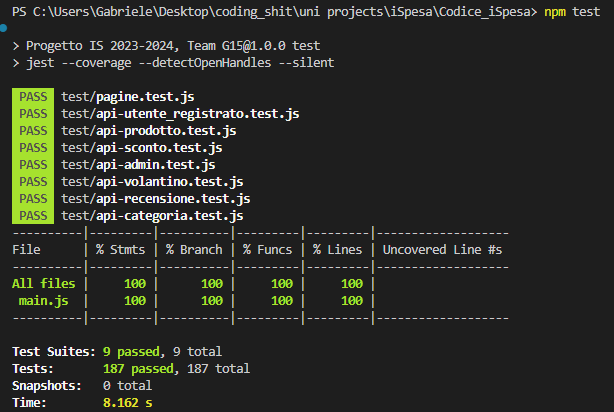
In questo con il comando npm test dalla root del progetto verranno eseguiti tutti i file .test.js dentro la cartella test.

Il flag --coverage serve per creare i report finali che testimoniano i risultati del testing.

Il flag --detectOpenHandles ha lo scopo di identificare se nel corso del testing viene chiamato qualcosa che impedisce al testing di terminare.

Il flag –silent ha lo scopo di evitare gli output delle API nella console del sito iSpesa, per visualizzare meglio la procedura del testing.

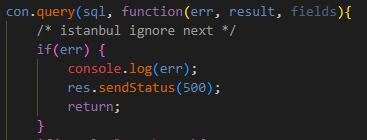
Ecco i risultati dei nostri test:



Tutte e 9 le suites sono passate, nelle quali sono presenti 187 test totali.

Dalle tabelle %Stmts (Statements), %Branch, %Funcs, %Lines possiamo visualizzare che ogni statement, branch, funzione e riga di codice dentro a main.js è stata testata e funziona correttamente.

Poichè questa procedura controlla anche righe di codice che raramente accadono, come errori del server SQL che possono essere difficilmente replicabili, abbiamo dovuto rimuoverli dai controlli della libreria Jest.



Usando l’Istanbul code coverage possiamo scrivere:

‘/\* istanbul ignore next \*/’

Per rimuovere delle funzioni dal controllo di codice di Jest. Questo è stato fatto per i vari errori che si possono presentare nel server SQL e per le schermate accessibili solamente attraverso browser e non attraverso API esterne.

