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

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.

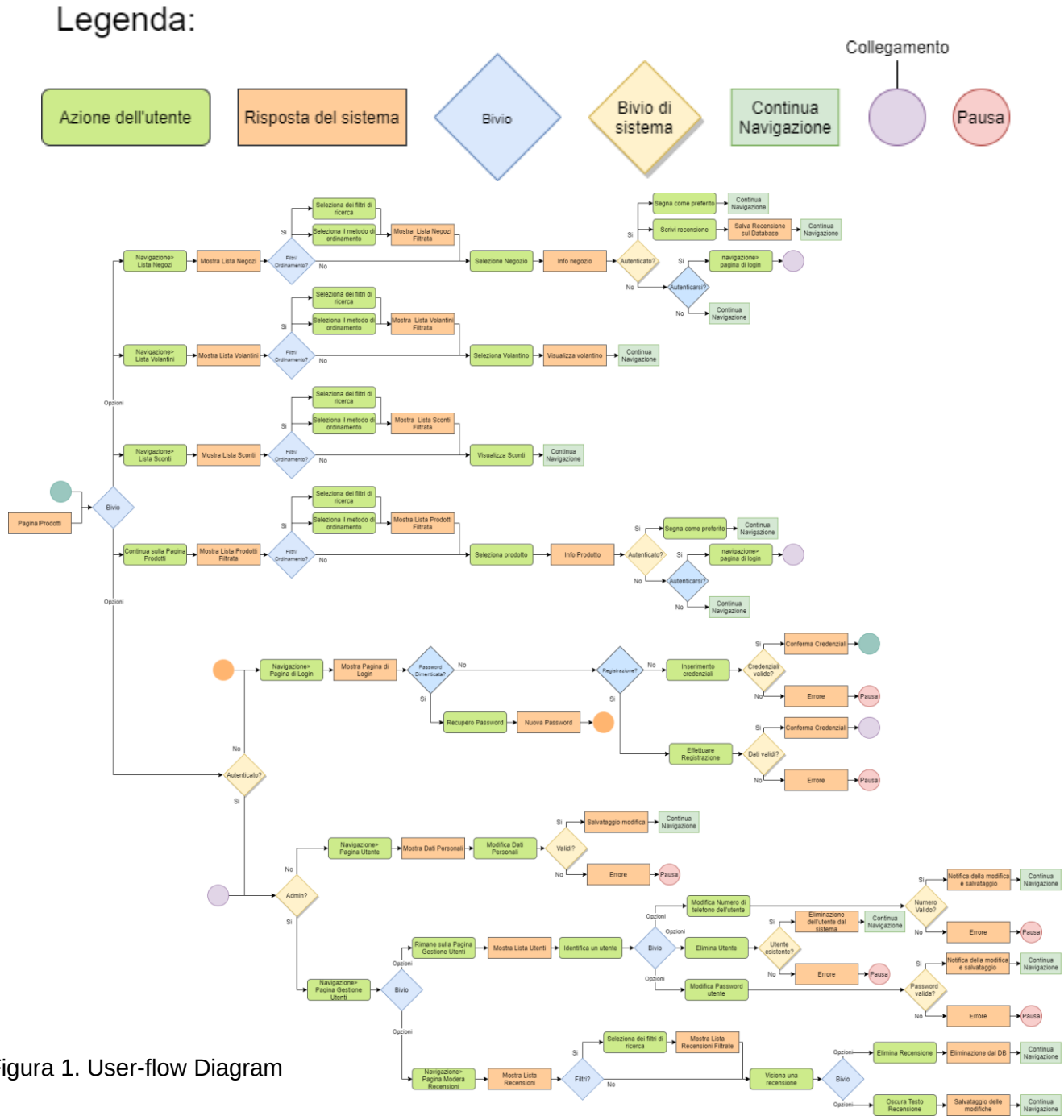
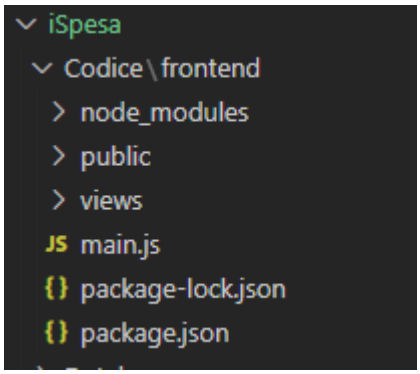


Figura 1. User-flow Diagram

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.

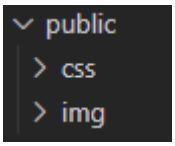
2.1 Struttura del progetto



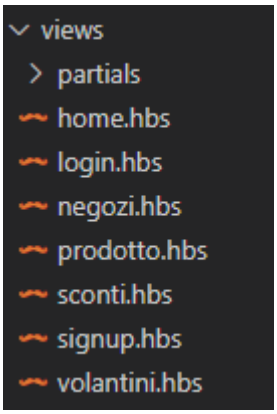
La seguente immagine riporta la struttura principale dell'applicazione. Come si può notare la cartella principale è la cartella *Codice_iSpesa* ed è così costituita:

- 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

```
CREATE TABLE `amministratore` (
  `Username` varchar(255) NOT NULL,
  `FotoProfilo` varchar(255) NOT NULL,
  `Email` varchar(255) NOT NULL,
  `Bloccato` tinyint(1) NOT NULL,
  `Password` varchar(255) NOT NULL
)

ALTER TABLE `amministratore`
  ADD PRIMARY KEY (`Username`),
  ADD UNIQUE KEY `Email` (`Email`);
```

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:

```
INSERT INTO `amministratore` (`Username`, `FotoProfilo`, `Email`, 'Bloccato','Password') VALUES
('AmministratoreDiSistema',
'https://lh3.googleusercontent.com/u/8/drive-viewer/AEYmBYRiEfJN3trP0-BAehuDx1jwiZ9lWtz6EYJDgm0xcfRaN0AiWx75jaBPOv-SJHwZaxmaGaMIpw4R2NeTuy9_rv7WQCc=w1920-h919',
'ispesasegnalazioni@gmail.com','0','Castoro1!')
```

4.2 Modello Utente Registrato

```
CREATE TABLE `utente_registrato` (
  `Username` varchar(255) NOT NULL,
  `FotoProfilo` varchar(255) NOT NULL,
  `2AF_attiva` tinyint(1) NOT NULL,
  `Email` varchar(255) NOT NULL,
  `Bloccato` tinyint(1) NOT NULL,
  `Telefono` int(12) NOT NULL,
  `Password` varchar(255) NOT NULL
)

ALTER TABLE `utente_registrato`
  ADD PRIMARY KEY (`Username`),
  ADD UNIQUE KEY `Email` (`Email`),
  ADD UNIQUE KEY `Telefono` (`Telefono`);
```

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:

```
INSERT INTO `utente_registrato` (`Username`, `FotoProfilo`,'"AF_attiva', `Email`, 'Bloccato','Telefono', 'Password') VALUES
('DavideGB',
'https://lh3.googleusercontent.com/u/8/drive-viewer/AEYmBYRiEfJN3trP0-BAehuDx1jwiZ9lWtz6EYJDgm0xcfRaN0AiWx75jaBPOv-SJHwZaxmaGaMIpw4R2NeTuy9_rv7WQCc=w1920-h919',
'0', 'davicolosimo23@gmail.com','0','3518497925','Banana14!')
```

4.3 Modello Categoria

```
CREATE TABLE `categoria` (
  `Categoria` varchar(255) NOT NULL
)

ALTER TABLE `categoria`
  ADD PRIMARY KEY (`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.

```
INSERT INTO `categoria` (`Categoria`) VALUES
('Bevande_alcoliche'),
('Bevande_analcoliche'),
('Carne'),
('Casa'),
('Colazione'),
('Cura_della_persona'),
('Formaggi'),
('Frutta'),
('Gastronomia'),
('Latte,Uova_e_Derivati'),
('Pane_e_Pasticceria'),
('Pasta'),
('Pesce'),
('Prodotti_alimentari'),
('Salumi'),
('Verdura');
```

4.4 Modello Negozio

```
CREATE TABLE `negozio` (
  `Ubicazione` varchar(255) NOT NULL,
  `Orari` varchar(255) NOT NULL,
  `Nome` varchar(255) NOT NULL,
  `Logo` varchar(255) NOT NULL,
  `IDNegozio` int(255) NOT NULL
)

ALTER TABLE `negozio`
  ADD PRIMARY KEY (`IDNegozio`),
  MODIFY `IDNegozio` int(255) NOT NULL AUTO_INCREMENT;
```

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:

```
INSERT INTO `negozio` (`Ubicazione`, `Orari`, `Nome`, `Logo`, `IDNegozio`) VALUES
('Trento_sud', '10:00-19:00', 'EuroSpin', 'https://lh3.google.com/u/0/d/1AZbWR1XsTIZuSVpLFeOB9kILg6H0Jv3f=w958-h890-iv2', 1),
('Trento', '9:00-21:00', 'Iperpoli', 'https://lh3.google.com/u/0/d/1-U7-8BsrmRqYduJn7L5ne7FNIIZovilb=w1607-h912-iv2', 2),
('Thiene', '8:00-20:00', 'Ipertosano', 'https://lh3.google.com/u/0/d/1JZDXZd906KERPtOAbKGFfzeo2z6IN8Fb=w958-h910-iv2', 3),
('Trento', '8:00-20:00', 'Lidl', 'https://lh3.google.com/u/0/d/1FxYA6gOHPeQcAzANj_oqMFLVhXNeomBX=w645-h910-iv2', 4);
```

4.5 Modello Negozi Preferiti

```
CREATE TABLE `negozipreferiti` (
  `Utente` varchar(255) NOT NULL,
  `Negozio` int(255) NOT NULL
)

ALTER TABLE `negozipreferiti`
  ADD PRIMARY KEY (`Utente`,`Negozio`),
  ADD KEY `negoziopref` (`Negozio`);
  ADD CONSTRAINT `negoziopref` FOREIGN KEY (`Negozio`) REFERENCES `negozio` (`IDNegozio`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `utente` FOREIGN KEY (`Utente`) REFERENCES `utente_registrato` (`Username`) ON DELETE CASCADE ON UPDATE CASCADE;
```

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

```
CREATE TABLE `prodotto` (
  `Nome` varchar(255) NOT NULL,
  `Immagine` varchar(255) NOT NULL,
  `Categoria` varchar(255) NOT NULL,
  `IDProdotto` int(255) NOT NULL,
  `NegozioProvenienza` int(255) NOT NULL
)

ALTER TABLE `prodotto`
  ADD PRIMARY KEY (`IDProdotto`),
  ADD KEY `categoria_del_prodotto` (`Categoria`),
  ADD KEY `prodotto_negozio` (`NegozioProvenienza`),
  MODIFY `IDProdotto` int(255) NOT NULL AUTO_INCREMENT,
  ADD CONSTRAINT `categoria_del_prodotto` FOREIGN KEY (`Categoria`) REFERENCES `categoria` (`Categoria`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `prodotto_negozio` FOREIGN KEY (`NegozioProvenienza`) REFERENCES `negozio` (`IDNegozio`) ON DELETE CASCADE ON UPDATE CASCADE;
```

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:

```
INSERT INTO `prodotto` (`Nome`, `Immagine`, `Categoria`, `IDProdotto`, `NegozioProvenienza`) VALUES
('Caramelle_Ricola', 'https://lh3.google.com/u/0/d/1j5eyUFBJb-zeulX5XZdSftlP62yeJdI9=w580-h910-iv1', 'Prodotti_alimentari', 0, 3);
```

4.6 Modello Prodotti Preferiti

```
CREATE TABLE `prodottipreferiti` (
  `Prodotto` int(255) NOT NULL,
  `Utente` varchar(255) NOT NULL
);


ALTER TABLE `prodottipreferiti`
  ADD PRIMARY KEY (`Prodotto`,`Utente`),
  ADD KEY `Utente-nomeutente` (`Utente`),
  ADD CONSTRAINT `Utente-nomeutente` FOREIGN KEY (`Utente`) REFERENCES `utente_registrato` (`Username`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `utente-prod` FOREIGN KEY (`Prodotto`) REFERENCES `prodotto` (`IDProdotto`) ON DELETE CASCADE ON UPDATE CASCADE;
```

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

```
CREATE TABLE `storicoprezzi` (
  `Prodotto` int(255) NOT NULL,
  `Prezzo` int(255) NOT NULL,
  `Data` date NOT NULL
);

ALTER TABLE `storicoprezzi`
  ADD PRIMARY KEY (`Prodotto`,`Data`),
  ADD CONSTRAINT `prodotto-prezzo` FOREIGN KEY (`Prodotto`) REFERENCES `prodotto` (`IDProdotto`) ON DELETE CASCADE ON UPDATE CASCADE;
```

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:

```
INSERT INTO `storicoprezzi` (`Prodotto`, `Prezzo`, `Data`) VALUES
(2, 1, '2023-12-21');
```

4.8 Modello Sconti

```
CREATE TABLE `sconti` (
  `CategoriaApplicabile` varchar(255),
  `Valore` int(11) NOT NULL,
  `IDSconto` int(255) NOT NULL,
  `Negozio` int(255) NOT NULL,
  'DataInizio' date NOT NULL,
  'DataFine' date NOT NULL
);

ALTER TABLE `sconti`
  ADD PRIMARY KEY (`IDSconto`),
  ADD KEY `sconti-cat` (`CategoriaApplicabile`),
  ADD KEY `sconti-neg` (`Negozio`),
  MODIFY `IDSconto` int(255) NOT NULL AUTO_INCREMENT,
  ADD CONSTRAINT `sconti-cat` FOREIGN KEY (`CategoriaApplicabile`) REFERENCES `categoria` (`Categoria`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `sconti-neg` FOREIGN KEY (`Negozio`) REFERENCES `negozio` (`IDNegozio`) ON DELETE CASCADE ON UPDATE CASCADE;
```

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. Sono presenti delle chiavi esterne, ovvero *CategoriaApplicabile* e *Negozio*, che hanno lo scopo di associare allo sconto una ***categoria di prodotti*** a cui si può applicare ed 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:

```
INSERT INTO `sconti` (`CategoriaApplicabile`, `Valore`, `IDSconto`, `Negozio`,'DataInizio', 'DataFine') VALUES
('Colazione', 10, 1, 2,'2024-02-01','2024-02-20');
```

4.9 Modello Validità Sconto Prodotto

```
CREATE TABLE `validita_sconto_prodotto` (
  `Sconto` int(255) NOT NULL,
  `prodotto` int(255) NOT NULL
);

ALTER TABLE `validita_sconto_prodotto`
  ADD PRIMARY KEY (`Sconto`,`prodotto`),
  ADD KEY `prod-scont` (`prodotto`),
  ADD CONSTRAINT `prod-scont` FOREIGN KEY (`prodotto`) REFERENCES `prodotto` (`IDProdotto`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `sconto-prod` FOREIGN KEY (`Sconto`) REFERENCES `sconti` (`IDSconto`) ON DELETE CASCADE ON UPDATE CASCADE;
```

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:

```
INSERT INTO `validita_sconto_prodotto` (`Sconto`, `prodotto`) VALUES
(1, 51);
```

4.10 Modello Recensione

```
CREATE TABLE `recensione` (
  `Titolo` varchar(255) NOT NULL,
  `Testo` varchar(255) NOT NULL,
  `N_stelle` int(5) NOT NULL,
  `Data_creazione` date NOT NULL,
  `Utente` varchar(255) NOT NULL,
  `IDRecensione` int(255) NOT NULL,
  `Negozio` int(255) NOT NULL
);

ALTER TABLE `recensione`
  ADD PRIMARY KEY (`IDRecensione`),
  ADD KEY `recensione-utente` (`Negozio`),
  ADD KEY `rec-utente` (`Utente`),
  MODIFY `IDRecensione` int(255) NOT NULL AUTO_INCREMENT,
  ADD CONSTRAINT `rec-utente` FOREIGN KEY (`Utente`) REFERENCES `utente_registrato` (`Username`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `recensione-negozio` FOREIGN KEY (`Negozio`) REFERENCES `negozio` (`IDNegozio`) ON DELETE CASCADE ON UPDATE CASCADE;
```

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:

```
INSERT INTO `recensione` (`Titolo`, `Testo`, `N_stelle`, `Data_creazione`,'Utente','IDRecensione','Negozio') VALUES
('Caratteristico','Il miglior supermercato di sempre', 5, '2023-12-17', 'DavideGB', 0,'Eurospin');
```

4.11 Modello Volantino

```
CREATE TABLE `volantino` (
  `Negozio` int(255) NOT NULL,
  `DataFine` date NOT NULL,
  `VolantinoFile` varchar(255) NOT NULL,
  `IDVolantino` int(255) NOT NULL
) ;

ALTER TABLE `volantino`
  ADD PRIMARY KEY (`IDVolantino`),
  ADD KEY `vol-neg` (`Negozio`),
  MODIFY `IDVolantino` int(255) NOT NULL AUTO_INCREMENT,
  ADD CONSTRAINT `vol-neg` FOREIGN KEY (`Negozio`) REFERENCES `negozio` (`IDNegozio`) ON DELETE CASCADE ON UPDATE CASCADE;
```

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:

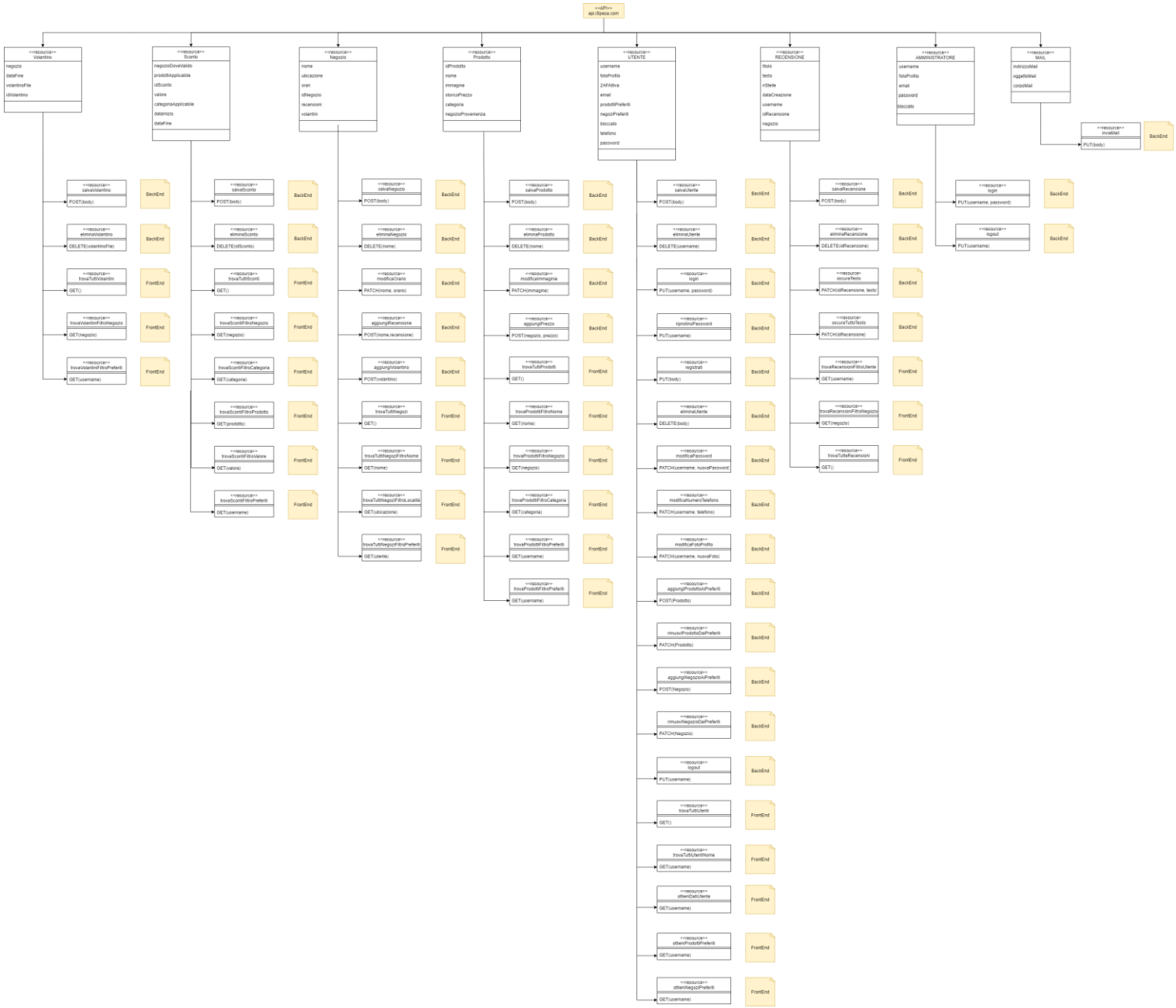
```
INSERT INTO `volantino` (`Negozio`, `DataFine`, `VolantinoFile`, `IDVolantino`) VALUES
(4, '2024-04-17', 'https://object.storage.eu01.onstackit.cloud/leaflets/pdfs/0d65a3c0-98f6-11ee-9ca4-fa163f3c89c9/LIDL-ATTUALE-S51-21-12-27-12-06.pdf', 1);
```

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