Gestione conti correnti

ATE

Lo scopo del prodotto finale è quello di gestire diversi conti correnti e tutte le operazioni annesse ad essi, quali “Prelievo”, “Deposito”, “Lista ultimi 5 movimenti”.

Per la realizzazione di questo prodotto ho pensato di costruirlo utilizzando dei moduli del framework Spring, come Spring Boot Starter web che permette di creare facilmente api RESTful, e applicazioni web utilizzando Spring MVC, Spring Boot Starter Data Jpa che permette invece di accedere e quindi modificare, archiviare e leggere dati da database relazionali e non. Io lo userò nello specifico per accedere facilmente a un database Mysql.

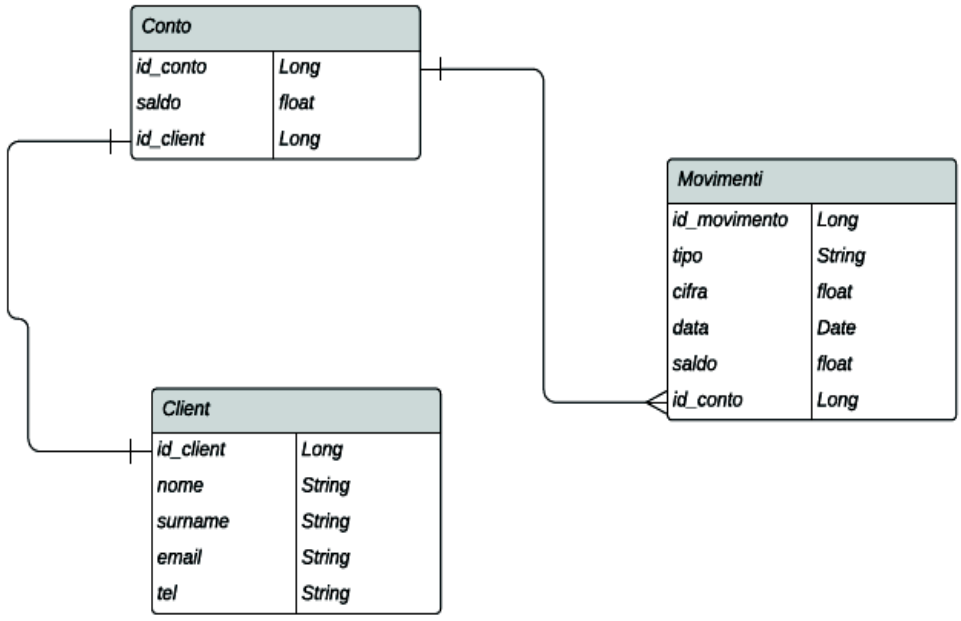
Il progetto è stato costruito per permettere di memorizzare dei clienti e quindi degli intestatari di conti correnti, i relativi dati dei conti correnti relazionati quindi con la tabella client tramite il campo “client", e tutti i movimenti effettuati nei conti per tenerne traccia.

```
-- Struttura della tabella `client`
CREATE TABLE `client` (
  `id_client` bigint(20) NOT NULL,
  `email` varchar(255) DEFAULT NULL,
  `name` varchar(255) DEFAULT NULL,
  `surname` varchar(255) DEFAULT NULL,
  `tel` varchar(255) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

-- Struttura della tabella `conto`
CREATE TABLE `conto` (
  `id_conto` bigint(20) NOT NULL,
  `saldo` float DEFAULT NULL,
  `client` bigint(20) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

```
-- Struttura della tabella `movimenti`
CREATE TABLE `movimenti` (
  `id_movimento` bigint(20) NOT NULL,
  `cifra` float DEFAULT NULL,
  `data` datetime DEFAULT NULL,
  `saldo` float DEFAULT NULL,
  `tipo` varchar(255) DEFAULT NULL,
  `id_conto` bigint(20) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Diagramma ER



Fatto ciò bisogna creare dei servizi che permettono di usufruire di tali operazioni.

Service:

ClientService, ContoService, MovimentiService

ClientService permetterà di: aggiungere un client tramite il metodo "addClient" il quale prenderà come parametro un'oggetto di tipo Client e di avere una lista di tutti i Client attualmente registrati nel DB con il metodo "getAllClient" che restituirà una List<Client>.

ContoService permetterà di: aggiungere un conto tramite il metodo "addConto" il quale prenderà come parametro un'oggetto di tipo Conto, di ricevere una lista di tutti i conti attualmente salvati nel DB con il metodo "getAllConto" che appunto restituirà una List<Conto>, di salvare delle modifiche apportate ad un'oggetto Conto tramite il metodo "saveUpdate" che prenderà come parametro appunto l'oggetto Conto già aggiornato, e di trovare un'oggetto a partire dal suo id_conto con il metodo "findById" che si aspetta come parametro un dato Long.

MovimentiService permettà di: aggiungere un nuovo movimento tramite il metodo "addMovimento" che prenderà come parametro un'oggetto di tipo Movimento, di avere la lista di tutti i movimenti attualmente presenti nel DB con il metodo "getAllMovimenti" che appunto restituirà una List<Movimenti>, e di avere la lista di tutti i movimenti relativi ad uno specifico Conto con il metodo "getAllMovimentiByConto" che prenderà come parametro un'oggetto di tipo Conto e ne restituirà una List<Movimenti>.

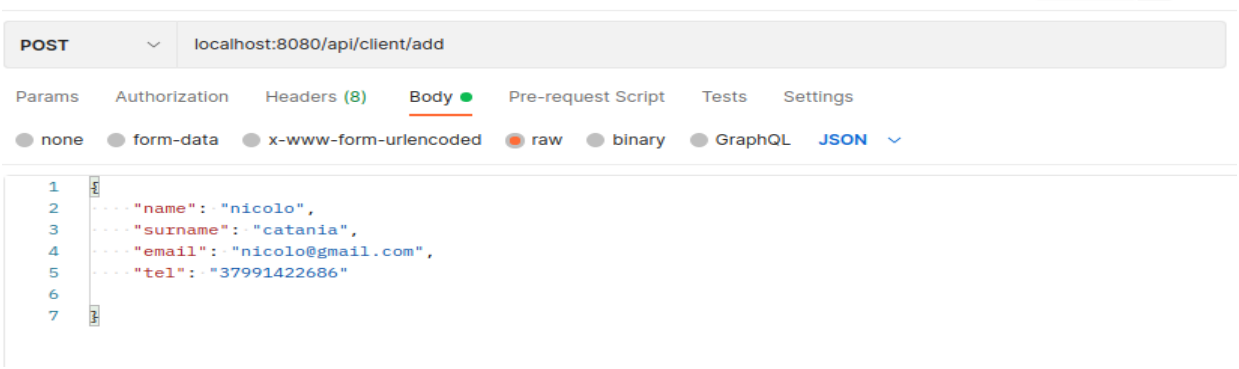
Per permettere di accedere dall'esterno a questi servizi creati bisogna creare dei controller di tipo Rest con relativi end-point creati.

Controller:

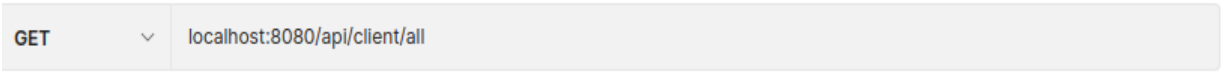
ClientController, ContoController, MovimentiController

ClientController che risponderà alle richieste col path=”/api/client”, avrà dunque due end-point:

addClient che risponderà alle richieste di tipo post, con un path: “/add”, permetterà di salvare un’oggetto Client. Di seguito un esempio di richiesta:

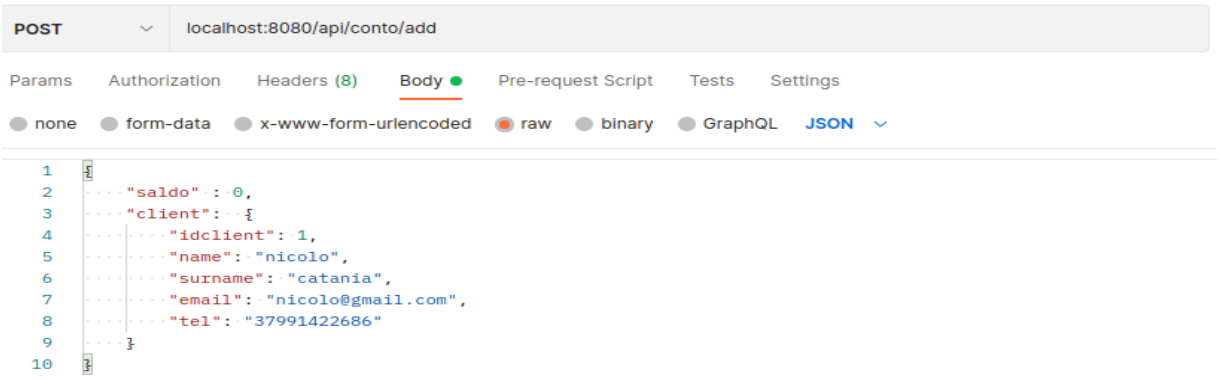


getAllClient che risponderà alle richieste di tipo get, con un path: “/all”, permetterà di ricevere una lista di tutti i clienti. Di seguito un esempio di richiesta:

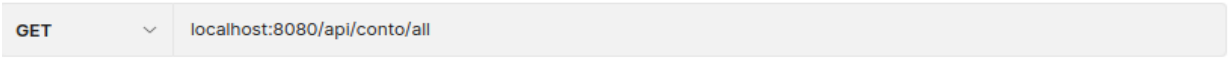


ContoController che risponderà alle richieste col path=”/api/conto”, avrà tre end-point:

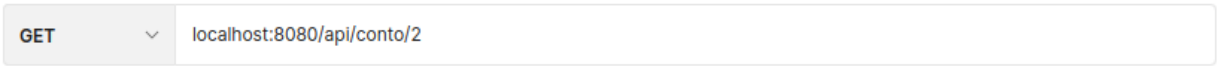
addConto che risponderà alle richieste di tipo post, con un path: “/add”, permetterà di salvare un’oggetto Conto. Di seguito un esempio di richiesta:



getAllConto che risponderà alle richieste di tipo get, con un path: “/all”, permetterà di ricevere una lista di tutti i conti. Di seguito un esempio di richiesta:

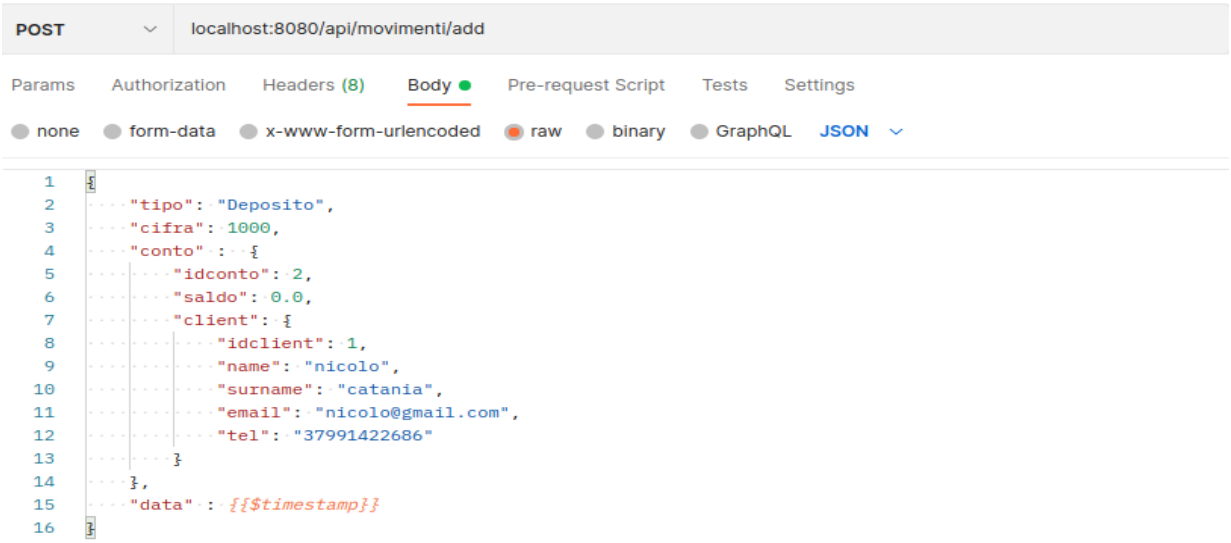


getContoById che risponderà alle richieste di tipo get, con un path: ”/{id}”, permetterà di ricevere l’oggetto di tipo Conto con l’id in questione. Di seguito un esempio di richiesta:

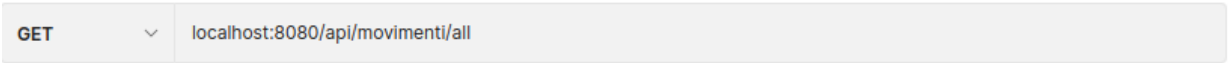


MovimentiController che risponderà alle richieste col path=”/api/movimenti”, avrà tre end-point:

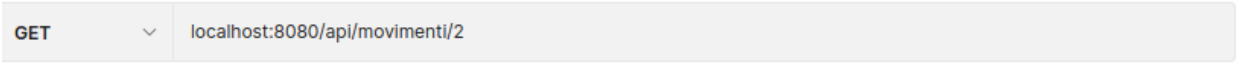
addMovimento che risponderà alle richieste di tipo post, con un path: “/add”, permetterà di salvare un’oggetto Movimenti e soprattutto di aggiornare il saldo del conto relativo al movimento effettuato. Di seguito un esempio di richiesta:



getAllMovimenti che risponderà alle richieste di tipo get, con un path: “/all”, permetterà di ricevere una lista di tutti i movimenti (di tutti i conti). Di seguito un esempio di richiesta:



GetMovimentiByConto che risponderà alle richieste di tipo get, con un path: ”/{id}”, permetterà di ricevere una lista degli ultimi 5 movimenti relativi al Conto con id passato nel path. Di seguito un esempio di richiesta:



Stime

Sviluppo	Stima(man/days)
Model/Struttura DB	0,2
ClientService	0,1
ContoService	0,1
MovimentiService	0,1
ClientController	0,2
ContoController	0,2
MovimentiController	0,2
Totale Stima	1,1
Contingency	0,5