

# PipeNetwork L.T.D.

Mucciacito Maria

Infobasic  
Via Tirino, 99  
65129 Pescara PE

## *Documentazione Progetto*

### *Sistema CRM*

**21 dicembre 2022**

### Panoramica

L'obiettivo di questa documentazione è quello di descrivere l'implementazione dell'applicazione CRM (Customer Relationship Management) della PipeNetwork I.t.d., azienda specializzata nella vendita di software ingegneristico in ambito meccanico. Un sistema CRM aiuta l'azienda a rimanere in contatto con i clienti, a semplificare i processi e a migliorare la redditività. Le funzionalità dell'applicazione sono le seguenti:

- Gestione dei Contatti
- Gestione dei Lead (clienti potenziali)

## Sommario

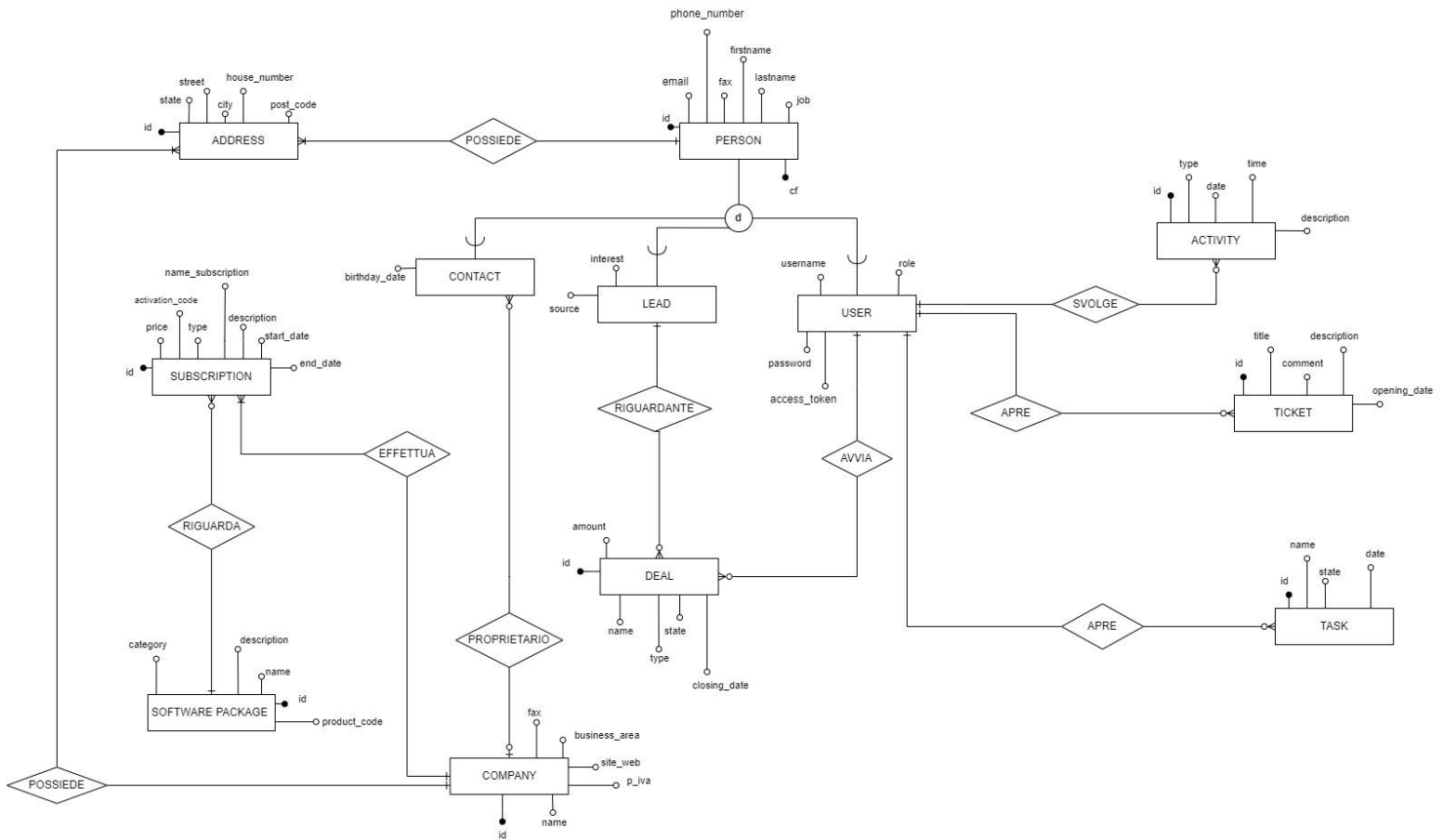
1. Acronimi e Abbreviazioni
2. Modello dei dati
3. UML classi
4. Api Rest
5. Diagramma dei Casi d'Uso
6. Struttura dell'applicazione
7. JWT
8. Tecnologie utilizzate

## Acronimi e Abbreviazioni

Terms	Definition
PERSON	Identifica a grandi linee una persona fisica
CONTACT	Identifica il nominativo di un cliente dell'azienda
LEAD	Identifica il nominativo di un potenziale cliente, ovvero una persona che ha già espresso un interesse per i nostri prodotti
USER	Identifica le credenziali di un utente che è loggato nel sistema e può quindi accedere alla dashboard del crm
ADDRESS	Identifica l'insieme delle indicazioni corrispondenti a un determinato domicilio
SUBSCRIPTION	Identifica un abbonamento ad un pacchetto software
SOFTWARE PACKAGE	Identifica un prodotto software venduto dall'azienda
DEAL	Identifica una trattativa che ha lo scopo di trasformare un potenziale cliente in uno effettivo
COMPANY	Identifica il nominativo di un'azienda
TASK	Identifica un to-do, ovvero un compito già pianificato che va eseguito
TICKET	Identifica una richiesta di assistenza
ACTIVITY	Identifica tutti i meeting, le telefonate e le attività effettuate da un utente

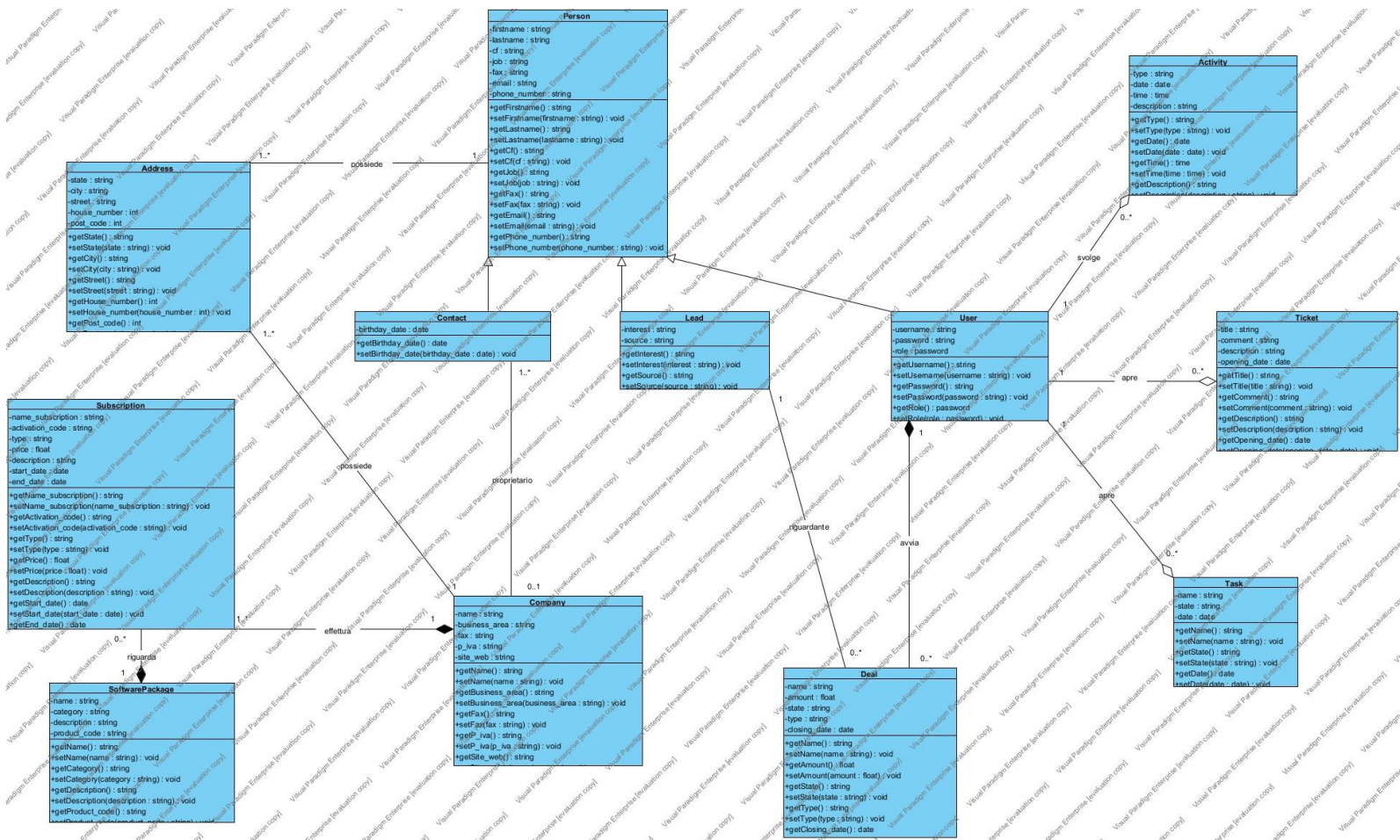
# Modello Dei Dati

Il modello concettuale consente di visualizzare il modo in cui le entità all'interno di un database si relazionano tra loro e con gli attributi di ciascun entità, attraverso lo schema ER (Entity Relationship).



# UML Classi

UML (Unified Modeling Language) è un linguaggio grafico e semi formale, utilizzato per visualizzare, costruire e documentare gli artefatti di un sistema software.



# Api Rest

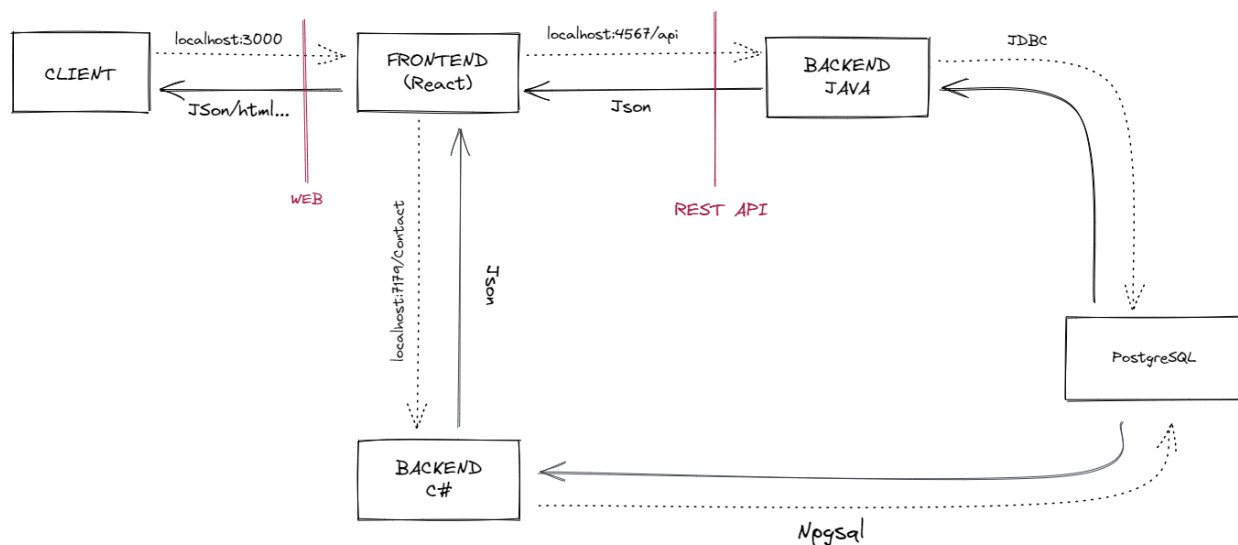
## 1.1. Descrizione

Le Api REST (acronimo di REpresentational State Transfer) sono un insieme di definizioni e protocolli con i quali vengono realizzati e integrati software applicativi.

L'API funge quindi da elemento di intermediazione tra gli utenti o i clienti e le risorse o servizi web che questi intendono ottenere. È anche un mezzo con il quale un'organizzazione può condividere risorse e informazioni assicurando al contempo sicurezza, controllo e autenticazione, poiché stabilisce i criteri di accesso.

Nel progetto vengono infatti utilizzate per la comunicazione tra frontend e backend.

## 1.2. Architettura dell'applicazione



Le API sono state realizzate per la maggior parte in C# per il CRUD dei modelli (entità del database), invece per i grafici sulla dashboard, che permettono di fare valutazioni su determinati aspetti dell'azienda, è stato utilizzato java. Di seguito gli endpoint realizzati:

<b>WebApiPipeNetwork</b> <small>1.0 OAS3</small>	
<small><a href="https://localhost:7176/swagger/v1/swagger.json">https://localhost:7176/swagger/v1/swagger.json</a></small>	
<b>Activity</b> ^	
GET	/api/Activity
POST	/api/Activity
GET	/api/Activity/{id}
PUT	/api/Activity/{id}
DELETE	/api/Activity/{id}
<b>Address</b> ^	
GET	/api/Address
POST	/api/Address
GET	/api/Address/{id}
PUT	/api/Address/{id}
DELETE	/api/Address/{id}

Company		^
GET	/api/Company	▼
POST	/api/Company	▼
GET	/api/Company/{id}	▼
PUT	/api/Company/{id}	▼
DELETE	/api/Company/{id}	▼
Contact		^
GET	/api/Contact	▼
POST	/api/Contact	▼
GET	/api/Contact/{id}	▼
PUT	/api/Contact/{id}	▼
DELETE	/api/Contact/{id}	▼
Deal		^
GET	/api/Deal	▼
POST	/api/Deal	▼
GET	/api/Deal/{id}	▼
PUT	/api/Deal/{id}	▼
DELETE	/api/Deal/{id}	▼
Lead		^
GET	/api/Lead	▼
POST	/api/Lead	▼
GET	/api/Lead/{id}	▼
PUT	/api/Lead/{id}	▼
DELETE	/api/Lead/{id}	▼

Person		^
GET	/api/Person	▼
POST	/api/Person	▼
GET	/api/Person/{id}	▼
PUT	/api/Person/{id}	▼
DELETE	/api/Person/{id}	▼
SoftwarePackage		^
GET	/api/SoftwarePackage	▼
POST	/api/SoftwarePackage	▼
GET	/api/SoftwarePackage/{id}	▼
PUT	/api/SoftwarePackage/{id}	▼
DELETE	/api/SoftwarePackage/{id}	▼
Subscription		^
GET	/api/Subscription	▼
POST	/api/Subscription	▼
GET	/api/Subscription/{id}	▼
PUT	/api/Subscription/{id}	▼
DELETE	/api/Subscription/{id}	▼
Task		^
GET	/api/Task	▼
POST	/api/Task	▼
GET	/api/Task/{id}	▼
PUT	/api/Task/{id}	▼
DELETE	/api/Task/{id}	▼



### Ticket

GET

/api/Ticket

POST

/api/Ticket

GET

/api/Ticket/{id}

PUT

/api/Ticket/{id}

DELETE

/api/Ticket/{id}

### User

GET

/api/User

POST

/api/User

GET

/api/User/{id}

PUT

/api/User/{id}

DELETE

/api/User/{id}

### UserLogin

POST

/api/UserLogin/login

### Contact

GET

/api/Contact

Parameters

No parameters

Try it out

Responses

Code	Description	Links
200	Success	No links

Media type

text/plain

Controls Accept header.

Example Value | Schema

```
{
  "firstName": "string",
  "lastName": "string",
  "fav": "string",
  "job": "string",
  "cpr": "string",
  "id": 0,
  "company": 0,
  "birthdayDate": {
    "year": 0,
    "month": 0,
    "day": 0,
    "dayOfWeek": 0,
    "dayOfYear": 0,
    "dayNumber": 0
  },
  "email": "string",
  "phoneNumber": "string",
  "addresses": [
    {
      "state": "string",
      "street": "string",
      "city": "string",
      "houseNumber": 0,
      "postCode": 0,
      "company": 0
    }
  ]
}
```

http://localhost:4567/leadbysource


**GET** ⌵ http://localhost:4567/leadbysource

Params Authorization Headers (6) **Body** Pre-request Script Tests Settings

☒ none ☐ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

This request does not have a body

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ⌵ 

```
1 {  
2   "call": 2,  
3   "email": 1  
4 }
```

http://localhost:4567/dealbystate


**GET** ⌵ http://localhost:4567/dealbystate

Params Authorization Headers (6) **Body** Pre-request Script Tests Settings

☒ none ☐ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

This request does not have a body

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON ⌵ 

```
1 {  
2   "closure lost": 1,  
3   "closure win": 1,  
4   "sales ready": 1  
5 }
```

http://localhost:4567/taskformonth

GET http://localhost:4567/taskformonth

Params Authorization Headers (6) Body Pre-request Script Tests Settings

☒ none ☐ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

This request does not have a body

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "12": 1,
3   "8": 1,
4   "9": 2
5 }
```

http://localhost:4567/taskbystate

GET http://localhost:4567/taskbystate

Params Authorization Headers (6) Body Pre-request Script Tests Settings

☒ none ☐ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

This request does not have

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

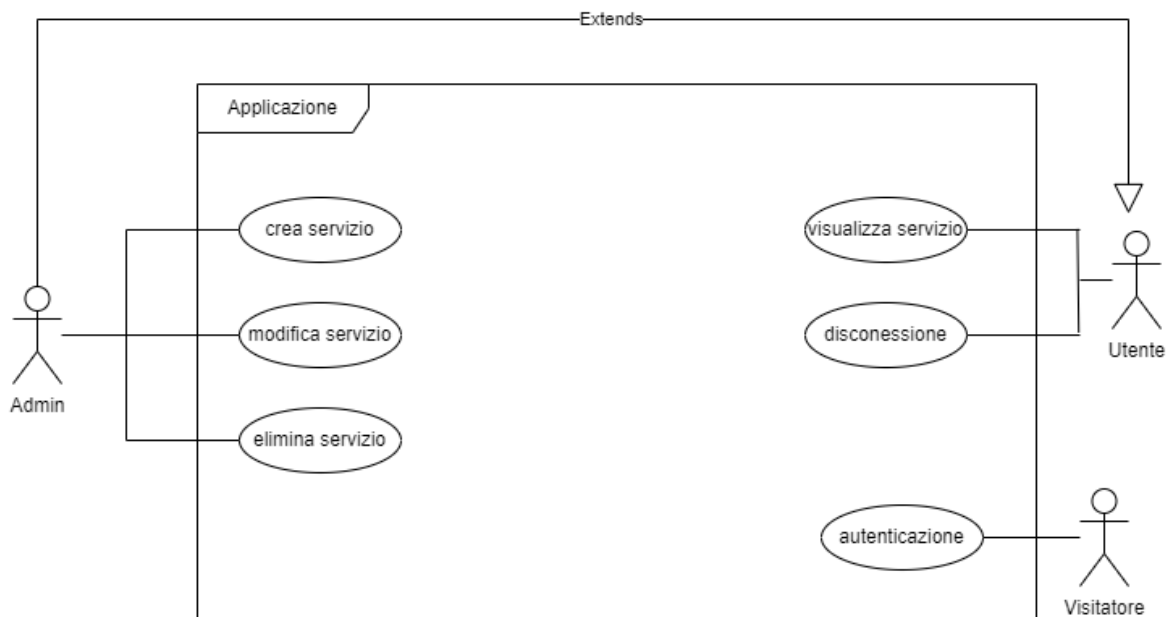
```
1 {
2   "waiting": 1,
3   "completed": 3
4 }
```

## Diagramma dei Casi d'Uso

Si tratta di un diagramma o di una descrizione che esamina un sistema o una sua parte. Individua chi ha a che fare con il sistema (attori) e che cosa gli attori possono fare (casi d'uso). Spiega quindi il funzionamento desiderato dal sistema.

Nello schema seguente abbiamo tre tipi di attori:

- l'admin che ha i permessi per modificare, eliminare e creare microservizi, oltre ad avere quelli dell'utente
- l'utente che ha i permessi per visualizzare i servizi
- il visitatore che effettua l'autenticazione

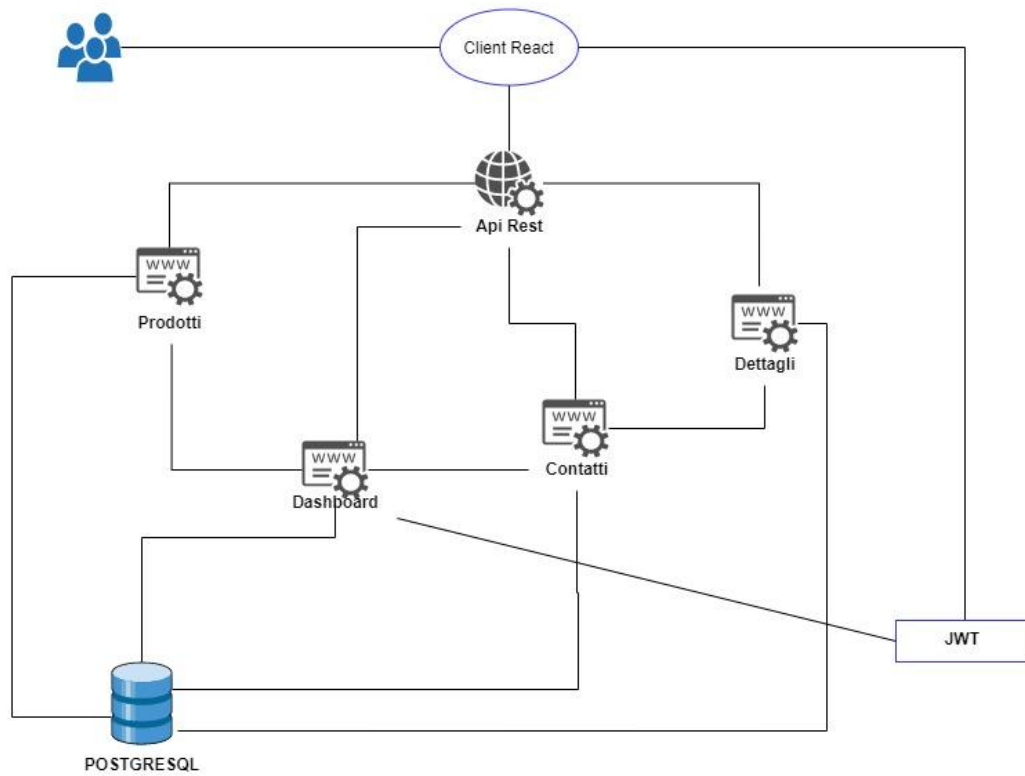


## Struttura dell'applicazione

La struttura dell'applicazione è la seguente :

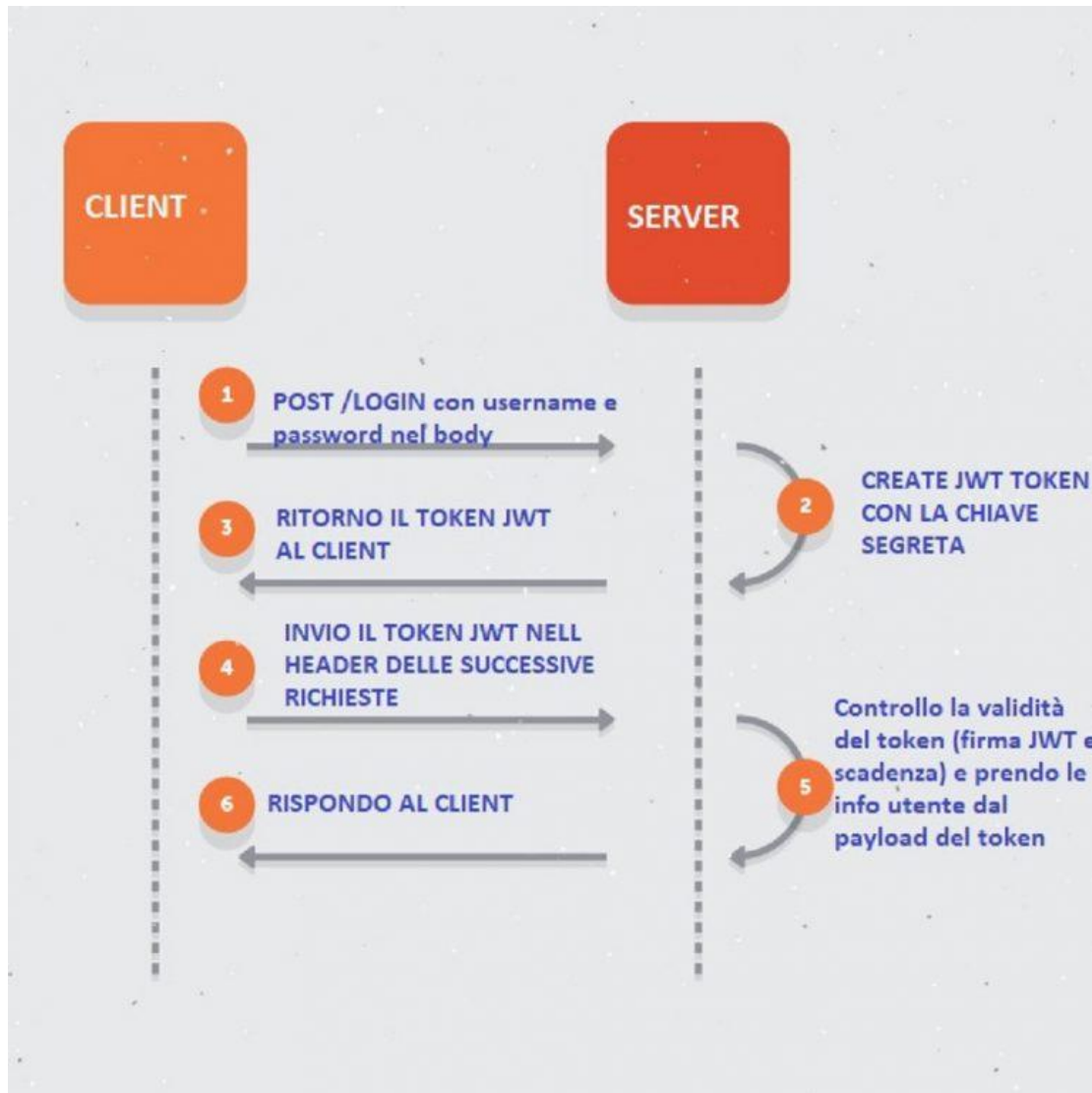
I visitatori del sito web si connettono ad esso, tramite un'interfaccia grafica realizzata in reactjs, e dopo aver effettuato il login tramite JWT ed essersi autenticati, diventano utenti e posso navigare nell'applicazione per svolgere diverse attività, attività che possono riguardare uno dei microservizi, come dashboard, contatti, dettagli di essi o i prodotti venduti.

Questi servizi sono resi disponibili grazie alle api, realizzate nel backend in java e c#, che permettono di leggere e modificare i dati, salvati all'interno di un database Postgresql.



# JWT

JWT, acronimo di JSON Web Token, è un sistema di cifratura e di contatto in formato [JSON](#) per lo scambio di informazioni tra i vari servizi di un server. Si genera così un token che può essere cifrato e firmato tramite una chiave disponibile solo a colui che lo ha effettivamente generato.



# Tecnologie Utilizzate

Le principali tecnologie utilizzate per il progetto sono le seguenti:

- Java, per il backend
- C#, per il backend
- Entity Framework, che permette di realizzare un modello concettuale e di interrogare direttamente le entità del dominio senza dover realizzare query SQL sul modello logico
- Swagger, per documentare le API REST, poiché è in grado di mappare quasi tutti i servizi web e le informazioni relative all'interfaccia
- JDB (Java Database Connectivity) è un' interfaccia completamente Java utilizzata per eseguire istruzioni SQL
- Spark Java, per creare le api rest
- React JS, per la creazione dell'interfaccia utente
- Axios, una libreria Javascript usata per fare richieste HTTP
- JWT
- React Router Dom , un pacchetto npm che consente di implementare il routing dinamico in un'app web
- Postman , per testare le api
- ElephantDb per creare l'istanza del database postgresql