



# UNIVERSITA' DEGLI STUDI DI MESSINA

## Progetto Programmazione II

---

Studente: Mobilia Alessio  
Professore: Salvatore Distefano  
Dipartimento: MIFT  
Anno: 2019/2020

---

<b>Introduzione</b>	<b>4</b>
<b>Framework: Ninja</b>	<b>6</b>
Concetti di base	6
Route	6
Controller	7
View	7
Flow	7
Filter	8
<b>Token</b>	<b>9</b>
Header	9
Payload	10
Signature	10
Implementazione	10
<b>ThingSpeak e Contatore</b>	<b>11</b>
ThingSpeak	11
Dispositivo	12
<b>Bing API</b>	<b>13</b>
Static Map	13
Esempio di richiesta	13
Location	15
Esempio	16
<b>Librerie</b>	<b>17</b>
Java JWT	17
Jackson	18
MySQL Connector/J	18
Java.*	18
ninja	18
JFreeChart	18
<b>File di configurazione</b>	<b>18</b>
Banca	19
config/DBinfo.xml	19
config/KEYS.xml	19
config/RSASERVER.xml	19
config/RSATOKEN.xml	20
config/ServerInfo.xml	20
Core Server	20
config/config.json	20
config/DBinfo.xml	21

config/KEYS.xml	21
config/RSACLIENT.xml	22
config/RSATOKEN.xml	22
config/ServerInfo.xml	22
Server Token	22
config/DBinfo.xml	22
config/RSA.xml	23
config/RSACLIENT.xml	23
config/RSASERVER.xml	23
config/token.xml	24
Client Principale	24
config/coords.json	24
config/KEYS.xml	24
config/RSA.xml	25
config/RSABANK.xml	25
config/RSASERVER.xml	25
config/RSATOKEN.xml	25
config/ServerInfo.xml	26
Client Log	26
config/KEYS.xml	26
config/RSA.xml	26
config/RSABANK.xml	27
config/RSASERVER.xml	27
config/RSATOKEN.xml	27
config/ServerInfo.xml	27
<b>Route HTTP</b>	<b>28</b>
Server Token	28
Server Banca	29
Core Server	30
<b>Sequence Diagram</b>	<b>33</b>
Login	33
Richiesta mappa	34
Richiesta Upgrade	35
Connessione Access Point	36
<b>Class Diagram</b>	<b>37</b>
Server Token	37
conf	37
controllers	37
mysql	38
utility	38
crittography	39

POJO	40
Server Banca	41
conf	41
controllers	41
crittography	41
filters	43
mysql	45
utility	45
POJO	47
Core Server	48
conf	48
controllers	48
crittography	50
filters	51
mysql	53
utility	54
POJO	58
Client	62
GUI	62
Server	66
TokenServer	67
Request	67
crittography	68
POJO	69
Client Log	70
GUI	70
Server	75
TokenServer	76
Request	76
crittography	77
POJO	80
<b>Database</b>	<b>81</b>
bank	81
core	81
nothings	84
<b>GUI</b>	<b>84</b>
Client principale	84
Login e Registrazione	84
Mappa	86
Connessione Access Point	87
Gestione Contatore	88

Gestione Access Point	89
Registrazione carta di credito	90
Client log	91
Login e Registrazione	91
Log Connessioni	92
Log Pagamenti	92
Grafico a torta	93
Grafico a bolle	94

## Introduzione

il progetto si pone l'obiettivo di fornire agli utenti la possibilità di connettersi a diversi punti di accesso wifi sparsi nella città in cambio di crediti.

I crediti possono essere guadagnati condividendo l'accesso della propria connessione agli altri utenti.

Un utente può avere un dispositivo fisico che mostra a display il numero dei crediti in suo possesso. Questi devono essere precedentemente configurati sulla piattaforma online ThingSpeak e possono essere aggiunti e modificati sul client.

Un utente può ottenere l'accesso illimitato a tutte le reti wifi per un periodo limitato di tempo pagando una quota e diventando così utente premium.

Il progetto è diviso in 5 programmi, **tre server e due client**:

- **il client utente** permette agli utenti, previa registrazione, di utilizzare il servizio al pieno delle possibilità. Viene visualizzata la mappa della zona in cui si trova l'utente con indicati gli Access Point più vicini. L'utente ne può selezionare uno e spendere dei crediti per ricevere una password temporanea.  
L'utente può aggiungere e gestire i propri Access Point e guadagnare 100 crediti per ogni inserimento e 2 crediti per ogni ora di accesso richiesto da un altro utente.  
Invece di spendere crediti per potersi connettere ad un access point, un utente può effettuare un pagamento con carta di credito per diventare utente premium e poter usufruire gratuitamente di qualsiasi access point.
- **il client dei log** permette ad un utente con permessi di amministratore di visualizzare i log delle transazioni effettuate dagli utenti per eseguire l'upgrade a premium e i log delle connessioni agli Access Point.

Inoltre fornisce due grafici utili ad una migliore monitoraggio sull'uso degli access point.

- **Il server banca** gestisce le informazioni degli utenti relative ai pagamenti, come la carta di credito. Gestisce inoltre le transazioni necessarie ad un utente per effettuare l'upgrade ad utente premium.
- **Il server token** gestisce il database degli utenti e genera un token che autentica l'utente negli altri server.
- **Il core server** gestisce gli access point degli utenti, ne controlla l'accesso, tiene traccia delle connessioni e gestisce il saldo dei crediti degli utenti, si occupa anche di aggiornare il valore del contatore sul server ThingSpeak.

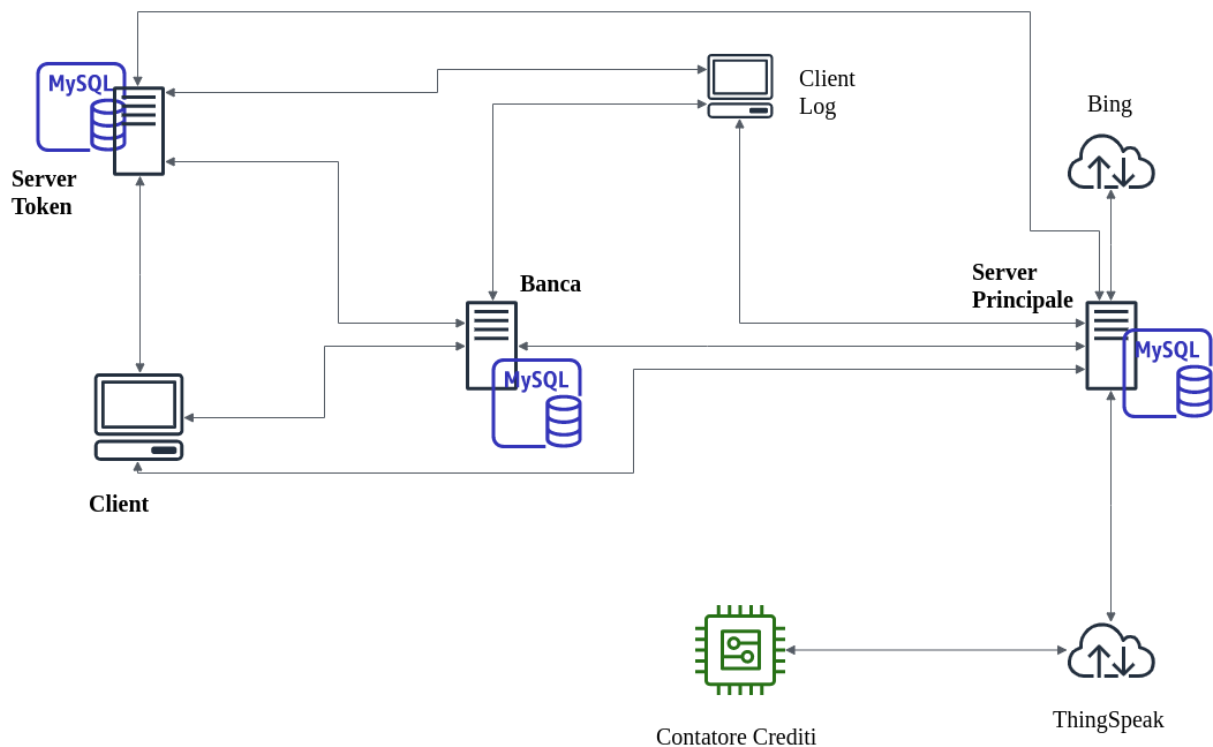


Figure 1: Connessioni di rete del progetto

Il progetto è quindi un'implementazione dell'architettura client/server che comunica su protocollo HTTP attraverso l'utilizzo di json.

I server sono implementati utilizzando il Framework Ninja.

I client presentano una GUI sviluppata grazie al framework swing.

Vi sono diversi file di configurazione in xml che permettono di definire le chiavi utili agli algoritmi di crittografia, le informazioni sugli altri server e le informazioni per la generazione dei token.

L'infrastruttura è stata progettata per utilizzare RSA per codificare il token e AES256 per codificare il body delle richieste http e delle risposte.

Vi sono 3 diversi database mysql, uno per ogni server.

Il core server utilizza la API fornite da Bing e ThingSpeak. Quelle di Bing per fornire le mappe e il servizio di traduzione da indirizzo a coppia latitudine/longitudine. Quelle di ThingSpeak per controllare i dispositivi (il contatore dei token).

Per la gestione migliore delle librerie è stato utilizzato maven.

## Framework: Ninja

Per la realizzazione del progetto ho deciso di utilizzare il web framework ninja per java. In quanto è un framework solido, veloce e produttivo, con una buona documentazione ed una filosofia open source. Ha un'architettura restfull e permette un'organizzazione del codice pulita.

### Concetti di base

#### Route

Il file "Routes" è il punto di accesso di ogni richiesta HTTP. Permette di mappare un URL ad un metodo all'interno dell'applicazione.

Per convenzione ogni applicazione Ninja contiene il file "conf/Routes.java"

#### Esempio base di file Routes.java

```
public class Routes implements ApplicationRoutes {  
  
    @Override  
    public void init(Router router) {  
  
        router.GET().route("/").with(ApplicationController::index);  
  
    }  
}
```

## Controller

I controller sono semplici classi java presenti all'interno del package "controllers". I metodi del controller restituiscono sempre il tipo "Result". Che sono dei tipi che contengono tutte le informazioni che il framework restituirà come risposta alla chiamata http.

Results è una classe che aiuta a creare l'oggetto di tipo Result.

### Esempio base di controller

```
package controllers;

@Singleton
public class ApplicationController {

    public Result index() {
        return Results.html();
    }
}
```

## View

Le viste sono dichiarate dentro il package "view". Per convenzione sono sempre salvate in "views/CONTROLLER\_NAME/METHOD\_NAME"

Le viste permettono di avere dei modi di per formattare delle pagine html in modo standard. Non sono state utilizzate nel progetto in quanto le chiamate al server restituiscono sempre un json o un messaggio di errore.

## Flow

Il flusso di base delle operazioni in un'applicazione Ninja è **Route => Controller => View**.

Vi sono molti altre cose da dire sul framework Ninja, ma queste sono una panoramica sufficiente per comprendere il progetto sviluppato.

## Filter

I filtri sono un potente strumento del framework. Non sono considerati come strumenti essenziali, ma ne ho fatto largo uso nel progetto.



I filtri se indicati nel file di route vengono chiamati prima del controller e possono essere utilizzati per controllare la validità dei dati.

### Esempio di filtro nel file Route

```
router.GET().route("/index").filters(SecureFilter.class).with(AppController::index);
```

Ho utilizzato i filtri per controllare la validità del token, per controllare se l'utente è autorizzato a eseguire l'operazione richiesta e per decifrare il body delle richieste http.

### Esempio di filtro

```
public class SecureFilter implements Filter {  
  
    /** If a username is saved we assume the session is valid */  
    public final String USERNAME = "username";  
  
    @Override  
    public Result filter(FilterChain chain, Context context) {  
  
        // if we got no cookies we break:  
        if (context.getSession() == null  
            || context.getSession().get(USERNAME) == null) {  
  
            return  
Results.forbidden().html().template("/views/forbidden403.ftl.html");  
  
        } else {  
            return chain.next(context);  
        }  
  
    }  
}
```

Il filtro restituisce `chain.next(context)` nel caso non ci siano problemi, ovvero il prossimo elemento nella catena dei filtri. Se invece il filtro riscontra qualche problema, restituisce un oggetto `Result` che verrà direttamente restituito all'utente senza che il controller venga eseguito.

Il parametro `Context context` permette di passare variabili tra i filtri e tra i filtri e il controller.

```
context.setAttribute("foo", "bar");
```

```
String foo = context.getAttribute("foo");
```

Esistono filtri predefiniti come il “BasicAuthFilter”, ma non sono stati utilizzati nel progetto.

## Token

Per l'autenticazione si è deciso di usare il json web token, ovvero un metodo standard (RFC 7519) per rappresentare le richieste tra due parti.

Il json web token consiste in una stringa divisa in tre parti da un punto: **Header, Payload e Signature**.

Tipicamente un token ha questo aspetto:

```
xxxxx.yyyyy.zzzzz
```

## Header

L'header tipicamente consiste in due parti: il tipo di token e l'algoritmo usato per la firma.

### esempio di header

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

poi il json viene codificato in Base64 per formare la prima parte del token.

## Payload

La seconda parte del token è il payload, che contiene i “claims”. I Claims sono delle informazioni su un'entità, di solito l'utente.

Vi sono dei claim predefiniti che non sono obbligatori, ma è raccomandato inserire, come ad esempio: iss (issuer), exp (expiration time), sub (subject), aud (audience)

### esempio di payload

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Successivamente il json è convertito in Base64 e forma la seconda parte del token

## Signature

La firma prende l'header codificato, il payload codificato, un stringa segreta e firma il tutto con l'algoritmo specificato nell'header

Il tutto viene convertito in Base64 e aggiunto al token.

## Implementazione

Nel progetto il server dedicato genera il token e lo restituisce all'utente che lo utilizzerà per autenticarsi sugli altri server. E' stata utilizzata la libreria open source "Java JWT" per generare i token e per verificarli.

**Esempio di token** che rappresenta l'utente "ciao" identificato dall'id "45":

```
eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI0NSIsInN1YiI6ImNpYW8iLCJpc3MiOiJub1RoYW5nc1Rva2VuIiwiaWF0IjoxLCJpYXQiOiJlNjE1Njk0MTE5NjQsImV4cCI6MTU2OTQxMjk2NH0.gUq02U9iyutv3e_nop2E0B0Nu-SEodMI_B1r3BB14JM
```

Header:

```
{
  "alg": "HS256"
}
```

Payload:

```
{
  "jti": "45",
  "sub": "ciao",
  "iss": "noThingsToken",
  "adm": 1,
  "iat": 1569411964,
}
```

```
"exp": 1569412964  
}
```

Signature:

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  
) secret base64 encoded
```

## ThingSpeak e Contatore

### ThingSpeak

ThingSpeak è un'applicazione open source per Internet of Things che permette di conservare e recuperare dati da oggetti smart usando il protocollo HTTP o/e MQTT attraverso internet o LAN.

ThingSpeak dà la possibilità di analizzare e visualizzare i dati con MATLAB oltre che impostare delle azioni da eseguire quando si verificano determinate condizioni.

Questo servizio mette a disposizione di ogni utente diversi canali e, per ognuno di questi, al massimo 5 campi. L'associazione di un dispositivo ad un determinato canale ci permette di conservare il valore dei crediti residui da visualizzare.

Grazie alle API di ThingSpeak è quindi possibile ricevere o inviare le informazioni nei diversi campi, il cui stato è conservato nello storico.

Una limitazione della piattaforma è data dal fatto che può ricevere un aggiornamento del canale solamente ogni 15 secondi. Limite imposto per evitare il flood dei server.

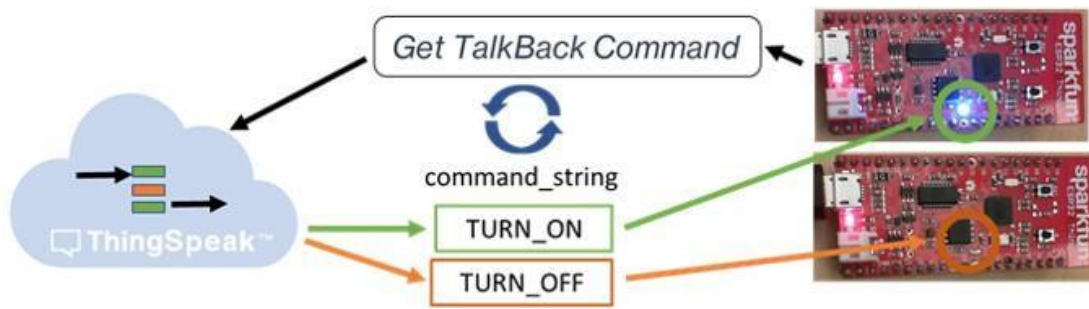


Figure 2: Esempio di comunicazione tra un dispositivo e ThingSpeak

## Dispositivo

Per visualizzare i crediti residui ho utilizzato un dispositivo NodeMCU con ESP8266, quindi un dispositivo economico dotato di un'interfaccia di rete wireless e con una capacità computazionale più che sufficiente allo scopo.

Ho programmato il dispositivo in modo tale da effettuare continuamente delle richieste HTTPS alla piattaforma ThingSpeak, con lo scopo di chiedere il valore del campo stabilito in precedenza. Il valore letto viene successivamente visualizzato sul display LCD.

In questo modo è il dispositivo a contattare il server e questo non sarà direttamente raggiungibile dall'esterno, quindi meno soggetto a possibili attacchi informatici.

Per semplicità ho "clonato" la libreria open source ThingSpeak-Arduino e l'ho utilizzata per effettuare le richieste HTTPS al server dal dispositivo.

Mentre sta tentando di connettersi al wifi, il dispositivo non potrà effettuare nessun'altra operazione. Per segnalare questo stato si illuminerà il led rosso.

Per ogni utente è possibile creare un dispositivo ed associarlo ad un nuovo canale ThingSpeak. Le informazioni del canale possono poi essere autonomamente inserite dall'utente interessato attraverso il client.

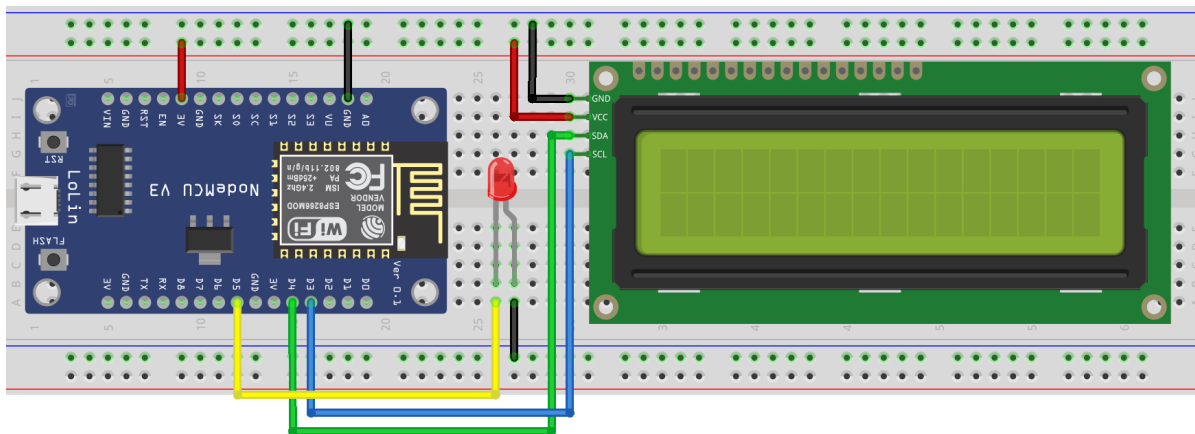


Figure 3: Schema elettrico dispositivo contatore

# Bing API

Microsoft rende disponibile le api di bing agli utenti registrati. Questi possono essere utilizzate gratuitamente fino a 150.000 richieste all'anno. Le api messe a disposizione sono molte e sono realizzate secondo l'architettura rest.

## Static Map

Le API Static Map di Bing permettono di richiedere l'immagine di una mappa. Questa può mostrare un percorso o dei segnalini. Questi segnalini possono essere richiesti di diversa tipologia e possono includere delle etichette.

Nel progetto ho utilizzato questa API per mostrare gli Access Point nelle vicinanze dell'utente. I pin che rappresentano gli Access Point sono rappresentati con delle etichette che permettono all'utente di identificarli univocamente e velocemente. Così da richiedere l'accesso all'AP desiderato.

E' possibile richiedere tramite richiesta GET una mappa con al massimo 10 segnalini, perciò ho scelto di utilizzare la richiesta POST che permette di segnare 100 segnalini differenti.

## Esempio di richiesta

### Http POST Header

```
Content-Length: LengthOfHTTPBody  
Content-Type: text/plain; charset=utf-8
```

### Body

```
pp=38.1939,15.5526;64;  
pp=38.1938,15.5542;48;1  
pp=38.1953,15.5539;48;2  
pp=38.1999,15.5528;48;3
```

## Risultato

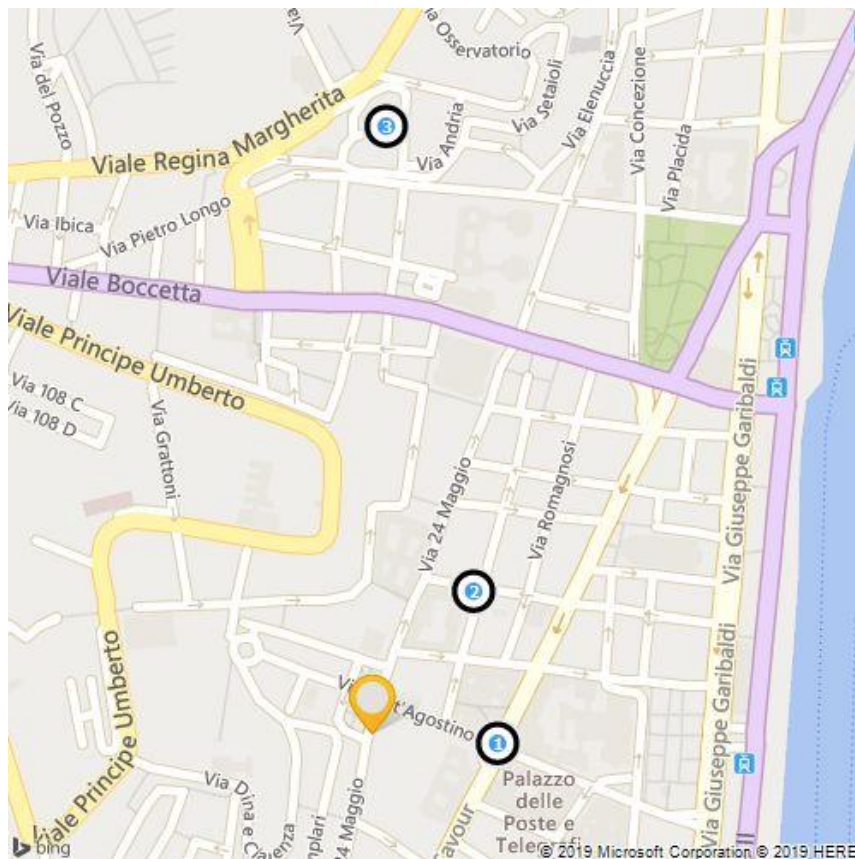


Figure 4: Risultato di una richiesta di mappa alle api di Bing

## Location

Le Location API di Bing permettono di risalire alla latitudine e longitudine di un luogo. In particolare io ho utilizzato le api “Find a Location by Query”, che permettono di richiedere latitudine e longitudine non solamente di un indirizzo ben preciso, ma anche di un luogo, come ad esempio “Università di messina”.

L'utilizzo di questa api nel progetto permette all'utente di inserire il proprio indirizzo, in modo da poter visualizzare nella mappa i dispositivi vicini all'utente. Inoltre è utilizzata anche nel momento in cui l'utente vuole inserire il proprio Access Point nel sistema.

In alternativa a questa api vi è la possibilità di utilizzare la geolocalizzazione, grazie alla libreria GeoLite2, di cui parlerò in seguito, ma è molto meno precisa.



## Esempio

### url

```
dev.virtualearth.net/REST/v1/Locations?query=Universit%C3%A0%20di%20messina&maxResults=1&key={api_key}
```

### Result

```
{
  "authenticationResultCode": "ValidCredentials",
  "brandLogoUri":
  "http://dev.virtualearth.net/Branding/logo_powered_by.png",
  "copyright": "Copyright (c) 2019 Microsoft and its suppliers. All rights reserved. This API cannot be accessed and the content and any results may not be used, reproduced or transmitted in any manner without express written permission from Microsoft Corporation.",
  "resourceSets": [
    {
      "estimatedTotal": 1,
      "resources": [
        {
          "__type":
          "Location:http://schemas.microsoft.com/search/local/ws/rest/v1",
          "bbox": [
            38.054580688476562,
            15.408063888549805,
            38.301528930664062,
            15.652946472167969
          ],
          "name": "Messina, Sic., Italy",
          "point": {
            "type": "Point",
            "coordinates": [
              38.178371429443359,
              15.553079605102539
            ]
          },
          "address": {
            "adminDistrict": "Sic.",
            "adminDistrict2": "Messina",
            "countryRegion": "Italy",
            "formattedAddress": "Messina, Sic., Italy",
            "locality": "Messina"
          },
        },
      ],
    },
  ],
}
```

```

"confidence": "Medium",
"entityType": "PopulatedPlace",
"geocodePoints": [
  {
    "type": "Point",
    "coordinates": [
      38.178371429443359,
      15.553079605102539
    ],
    "calculationMethod": "Rooftop",
    "usageTypes": [
      "Display"
    ]
  }
],
"matchCodes": [
  "UpHierarchy"
]
}
],
"statusCode": 200,
"statusDescription": "OK",
"traceId": "6ab88df1e891467ca2febd0c15f2d46d|DU00000D78|7.7.0.0|Ref
A: B4D93982574D4EE2950CDEFFBAA66037 Ref B: DB3EDGE0708 Ref C:
2019-09-22T15:31:07Z"
}

```

## Librerie

### Java JWT

JJWT mira ad essere la libreria più facile da capire ed utilizzare per creare e verificare JSON Web Tokens su JVM e android.

La libreria è open source e può essere trovata su github.

## **Jackson**

Jackson fornisce molti modi per lavorare con i json, come la possibilità di convertire POJO (Plain Old Java Object) in JSON e viceversa. Jackson fornisce anche un ampio set di notazioni per il mapping.

## **MySQL Connector/J**

MySQL Connector/J è un connettore fornito dalla Oracle che permette a Java di comunicare con i database MySQL.

## **Java.\***

E' necessario l'utilizzo delle librerie standard come java.util, java.io, java.net, java.awt.image. In queste librerie vi è tutto il necessario per gestire l'URLEncoding, fare il parsing da/in Base64, gestire lo stream di dati, le eccezioni, immagini e molto altro.

## **ninja**

Il framework ninja prevede una vasta libreria con molte classi e tipi utili ad utilizzare il framework stesso.

## **JFreeChart**

JFreeChart è una libreria opensource che permette di rappresentare graficamente dei dati. In particolar modo è stato utilizzato nel progetto il grafico a torta e il grafico a bolle.

## **File di configurazione**

L'utilizzo dei file di configurazione permette di rendere il software molto flessibile. Quindi invece di utilizzare delle variabili hardcoded, molte informazioni sono state salvate in dei file esterni sia in formato xml che json.

# Banca

## config/DBinfo.xml

questo file conserva in formato xml tutte le informazioni per accedere al database della banca

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<DBInfo>
  <IP>localhost</IP>
  <Port>3306</Port>
  <Username>...</Username>
  <Password>...</Password>
</DBInfo>
```

## config/KEYS.xml

questo file conserva in formato xml le informazioni relative alle chiavi rsa e aes dell server banca

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<KEYS>

<PrivateKey>...</PrivateKey>

<PublicKey>...</PublicKey>

<AES>ChiaveSegretaPerCrittografareInAES</AES>

<AES_salt>Salt_...</AES_salt>

</KEYS>
```

## config/RSASERVER.xml

questo file conserva in formato xml le informazioni relative alle chiavi rsa del server principale

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><RSA>
<PrivateKey>NON NECESSARIA</PrivateKey>
<PublicKey>....</PublicKey>
```

```
</RSA>
```

### config/RSATOKEN.xml

questo file conserva in formato xml le informazioni relative alle chiavi rsa del token

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><RSA>  
<PrivateKey>NON NECESSARIA</PrivateKey>  
<PublicKey>....</PublicKey>  
</RSA>
```

### config/ServerInfo.xml

questo file conserva in formato xml le informazioni relative all'indirizzo e alle porte dei server.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>  
<ServerInfo>  
  <ServerIP>localhost</ServerIP>  
  <ServerPort>8080</ServerPort>  
  <ServerTokenIP>localhost</ServerTokenIP>  
  <ServerTokenPort>8081</ServerTokenPort>  
  <ServerBankIP>localhost</ServerBankIP>  
  <ServerBankPort>8082</ServerBankPort>  
</ServerInfo>
```

## Core Server

### config/config.json

questo file conserva importanti informazioni riguardanti l'effettiva esecuzione delle operazioni del server.

- range indica il range in km dal punto in cui si trova l'utente entro il quale mostrare degli access point sulla mappa.
- rewardNewAp indica la quantità di crediti da dare ad un utente quando inserisce un nuovi access point
- costConnection indica il costo in crediti per ogni ora di connessione ad un Access Point
- timeUpgrade indica la durata dell'upgrade a premium in mesi.
- expirationToken indica il tempo di validità di un token sul server principale in minuti

```
{
  "range": 1,
  "rewardNewAP": 100,
  "costConnection": 4,
  "timeUpgrade":12,
  "expirationToken":10
}
```

## config/DBInfo.xml

questo file conserva in formato xml tutte le informazioni per accedere al database della banca

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<DBInfo>
  <IP>localhost</IP>
  <Port>3306</Port>
  <Username>...</Username>
  <Password>...</Password>
</DBInfo>
```

## config/KEYS.xml

questo file conserva in formato xml le informazioni relative alle chiavi rsa e aes del server principale

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<KEYS>

<PrivateKey>...</PrivateKey>

<PublicKey>...</PublicKey>

<AES>ChiaveSegretaPerCrittografareInAES</AES>

<AES_salt>Salt_...</AES_salt>

</KEYS>
```

### config/RSACLIENT.xml

questo file conserva in formato xml le informazioni relative alle chiavi rsa del client

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><RSA>  
<PrivateKey>NON NECESSARIA</PrivateKey>  
<PublicKey>....</PublicKey>  
</RSA>
```

### config/RSATOKEN.xml

questo file conserva in formato xml le informazioni relative alle chiavi rsa del token

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><RSA>  
<PrivateKey>NON NECESSARIA</PrivateKey>  
<PublicKey>....</PublicKey>  
</RSA>
```

### config/ServerInfo.xml

questo file conserva in formato xml le informazioni relative all'indirizzo e alle porte dei server.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>  
<ServerInfo>  
  <ServerIP>localhost</ServerIP>  
  <ServerPort>8080</ServerPort>  
  <ServerTokenIP>localhost</ServerTokenIP>  
  <ServerTokenPort>8081</ServerTokenPort>  
  <ServerBankIP>localhost</ServerBankIP>  
  <ServerBankPort>8082</ServerBankPort>  
</ServerInfo>
```

## Server Token

### config/DBInfo.xml

questo file conserva in formato xml tutte le informazioni per accedere al database del server token

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<DBInfo>
  <IP>localhost</IP>
  <Port>3306</Port>
  <Username>...</Username>
  <Password>...</Password>
</DBInfo>
```

### config/RSA.xml

questo file conserva in formato xml le informazioni relative alle chiavi rsa del server token

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<RSA>

<PrivateKey>...</PrivateKey>

<PublicKey>...</PublicKey>

</RSA>
```

### config/RSACLIENT.xml

questo file conserva in formato xml le informazioni relative alle chiavi rsa del client

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<RSA>
<PrivateKey>NON NECESSARIA</PrivateKey>
<PublicKey>....</PublicKey>
</RSA>
```

### config/RSASERVER.xml

questo file conserva in formato xml le informazioni relative alle chiavi rsa del server principale

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><RSA>
<PrivateKey>NON NECESSARIA</PrivateKey>
<PublicKey>....</PublicKey>
</RSA>
```



## config/token.xml

questo file conserva in formato xml le informazioni relative alla generazione e all'autenticazione dei token

- key indica la chiave con cui viene firmato ogni token
- expiration indica la durata di ogni token
- issuer indica il creatore del token

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<token>
  <key>h8SNv28rWd4ajnlXSS4zdiyo2dr0vQ3mAAumystV+20=</key>
  <expiration>999999</expiration>
  <issuer>noThingsToken</issuer>
</token>
```

## Client Principale

### config/coords.json

questo file conserva le coordinate di default. All'avvio il client richiederà una mappa con centro queste coordinate.

```
{
  "latitude": "38.190750",
  "longitude": "15.554190"
}
```

### config/KEYS.xml

questo file conserva in formato xml le informazioni relative alle chiavi aes

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<KEYS>

<AES>ChiaveSegretaPerCrittografareInAES</AES>

<AES_salt>Salt_sdondapfijo2134dcn2</AES_salt>

</KEYS>
```

### config/RSA.xml

questo file conserva in formato xml le informazioni relative alle chiavi rsa del client principale

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><RSA>  
<PrivateKey>...</PrivateKey>  
  
<PublicKey>...</PublicKey></RSA>
```

### config/RSABANK.xml

questo file conserva in formato xml le informazioni relative alle chiavi rsa del server banca

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><RSA>  
<PrivateKey>NON NECESSARIA</PrivateKey>  
<PublicKey>....</PublicKey>  
</RSA>
```

### config/RSASERVER.xml

questo file conserva in formato xml le informazioni relative alle chiavi rsa del server principale

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><RSA>  
<PrivateKey>NON NECESSARIA</PrivateKey>  
<PublicKey>....</PublicKey>  
</RSA>
```

### config/RSATOKEN.xml

questo file conserva in formato xml le informazioni relative alle chiavi rsa del token

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><RSA>  
<PrivateKey>NON NECESSARIA</PrivateKey>  
<PublicKey>....</PublicKey>  
</RSA>
```

## config/ServerInfo.xml

questo file conserva in formato xml le informazioni relative all'indirizzo e alle porte dei server.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<ServerInfo>
  <ServerIP>localhost</ServerIP>
  <ServerPort>8080</ServerPort>
  <ServerTokenIP>localhost</ServerTokenIP>
  <ServerTokenPort>8081</ServerTokenPort>
  <ServerBankIP>localhost</ServerBankIP>
  <ServerBankPort>8082</ServerBankPort>
</ServerInfo>
```

## Client Log

### config/KEYS.xml

questo file conserva in formato xml le informazioni relative alle chiavi aes

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<KEYS>
  <AES>ChiaveSegretaPerCrittografareInAES</AES>
  <AES_salt>Salt_sdondapfijo2134dcn2</AES_salt>
</KEYS>
```

### config/RSA.xml

questo file conserva in formato xml le informazioni relative alle chiavi rsa del client di log

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><RSA>
  <PrivateKey>...</PrivateKey>
  <PublicKey>...</PublicKey></RSA>
```

### config/RSABANK.xml

questo file conserva in formato xml le informazioni relative alle chiavi rsa del server banca

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><RSA>  
<PrivateKey>NON NECESSARIA</PrivateKey>  
<PublicKey>....</PublicKey>  
</RSA>
```

### config/RSASERVER.xml

questo file conserva in formato xml le informazioni relative alle chiavi rsa del server principale

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><RSA>  
<PrivateKey>NON NECESSARIA</PrivateKey>  
<PublicKey>....</PublicKey>  
</RSA>
```

### config/RSATOKEN.xml

questo file conserva in formato xml le informazioni relative alle chiavi rsa del token

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><RSA>  
<PrivateKey>NON NECESSARIA</PrivateKey>  
<PublicKey>....</PublicKey>  
</RSA>
```

### config/ServerInfo.xml

questo file conserva in formato xml le informazioni relative all'indirizzo e alle porte dei server.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>  
<ServerInfo>  
  <ServerIP>localhost</ServerIP>  
  <ServerPort>8080</ServerPort>  
  <ServerTokenIP>localhost</ServerTokenIP>  
  <ServerTokenPort>8081</ServerTokenPort>
```

```
<ServerBankIP>localhost</ServerBankIP>
<ServerBankPort>8082</ServerBankPort>
</ServerInfo>
```

## Route HTTP

I server per ogni richiesta http prevista restituiranno in caso di successo HTTP code 200 + payload come indicato nelle tabelle sotto. In caso di errori restituiranno i codici di errore 500 o 401 e come payload "-1" oppure un messaggio di errore personalizzato.

In caso di successo payload saranno crittografati in AES, mentre i token saranno sempre crittografata con RSA.

Nessuna richiesta GET prevede payload, mentre tutte le altre lo necessitano.

## Server Token

METODO	PATH	OPERAZIONE
PUT	/signin	registra l'utente con le informazioni inserite nel payload e restituisce un token
POST	/login	restituisce le informazioni dell'utente registrato sotto forma di token
GET	/verify/{token}	se il token è valido restituisce le informazioni dell'utente.

## Server Banca

METODO	PATH	OPERAZIONE
PUT	<code>/{token}/registerCard</code>	Registra le informazioni della carta di credito inserite nel payload. Se vi è già una carta registrata per l'utente verrà sostituita.
GET	<code>/{token}/getCard</code>	restituisce gli ultimi 3 numeri della carta registrata dall'utente
GET	<code>/{token}/pay</code>	Effettua il pagamento per effettuare l'upgrade a premium. Utilizzerà le informazioni della carta di credito registrate per l'utente.
POST	<code>/verify</code>	Controlla i dati della transazione inseriti nel payload e ne restituisce la data di esecuzione
GET	<code>/{token}/getLog</code>	restituisce tutti i log delle transazioni
GET	<code>/{token}/getLog/{user_id}</code>	restituisce tutti i log delle transazioni dell'utente con id "user_id"
GET	<code>/{token}/getLog/{start}/{end}</code>	restituisce tutti i log delle transazioni nel periodo di tempo indicato

GET	<code>/token/getLog/{user_id}/{start}/ {end}</code>	restituisce tutti i log delle transazioni dell'utente con id "user_id" nel periodo di tempo indicato
-----	---	--

## Core Server

METODO	PATH	OPERAZIONE
POST	<code>/token/upgrade</code>	Verifica che il codice della transazione nel payload sia valido e effettua l'upgrade a premium dell'utente
GET	<code>/token/isPremium</code>	restituisce 1 se l'utente è premium, altrimenti 0
PUT	<code>/token/addAP</code>	registra le informazioni dell'AccessPoint (indicando le coordinate geografiche) e aggiunge dei crediti all'utente
PUT	<code>/token/addAPQuery</code>	registra le informazioni dell'AccessPoint (indicando la posizione in forma di via o di località) e aggiunge dei crediti all'utente
DELETE	<code>/token/removeAP/{ssid}</code>	rimuove l'AP con l'ssid indicato se appartiene all'utente che effettua la richiesta
GET	<code>/token/connectAP/{ssid}/{time}</code>	se l'utente ha abbastanza crediti (o è premium) restituisce la password valida per il tempo indicato

GET	<code>/{token}/geCost/{ssid}/{time}</code>	restituisce il costo della connessione all'ap per il tempo indicato. Se l'utente è premium restituisce 0
GET	<code>/{token}/getMyAP</code>	restituisce l'elenco degli AP dell'utente
GET	<code>/{token}/getListAP/{latitude}/{longitude}</code>	restituisce gli AP nel range indicato dai file di configurazione del server
GET	<code>/{token}/getListAP/{query}</code>	restituisce gli AP nel range indicato dai file di configurazione del server
GET	<code>/{token}/getMap/{latitude}/{longitude}</code>	restituisce la mappa con indicati gli AP nel range definito dal file di configurazione del server
GET	<code>/{token}/getMap/{query}</code>	restituisce la mappa con indicati gli AP nel range definito dal file di configurazione del server
PUT	<code>/{token}/addCounter</code>	aggiunge o sostituisce le informazioni del contatore dell'utente
DELETE	<code>/{token}/removeCounter</code>	rimuove le informazioni sul counter dell'utente



GET	/token/getInfoCounter	restituisce le informazioni del contatore dell'utente
GET	/token/getBalance	restituisce il saldo dei crediti dell'utente
GET	/token/getUsers	restituisce tutti gli utenti (solo agli utenti admin)
GET	/token/getLog	restituisce il log delle connessioni agli AP (solo agli utenti admin)
GET	/token/getLog/{user_id}	restituisce il log delle connessioni agli AP effettuate dell'utente indicato (solo agli utenti admin)
GET	/token/getLog/{start}/{end}	restituisce il log delle connessioni agli AP effettuate nel tempo indicato (solo agli utenti admin)
GET	/token/getLog/{user_id}/{start}/{end}	restituisce il log delle connessioni agli AP effettuata dall'utente indicato nel tempo indicato (solo agli utenti admin)
GET	/token/getAPStats	restituisce le statistiche sull'uso degli AP utili ad effettuare il grafico a bolle (solo agli utenti admin)

# Sequence Diagram

## Login

Si ipotizza che l'utente abbia già effettuato una registrazione al server token

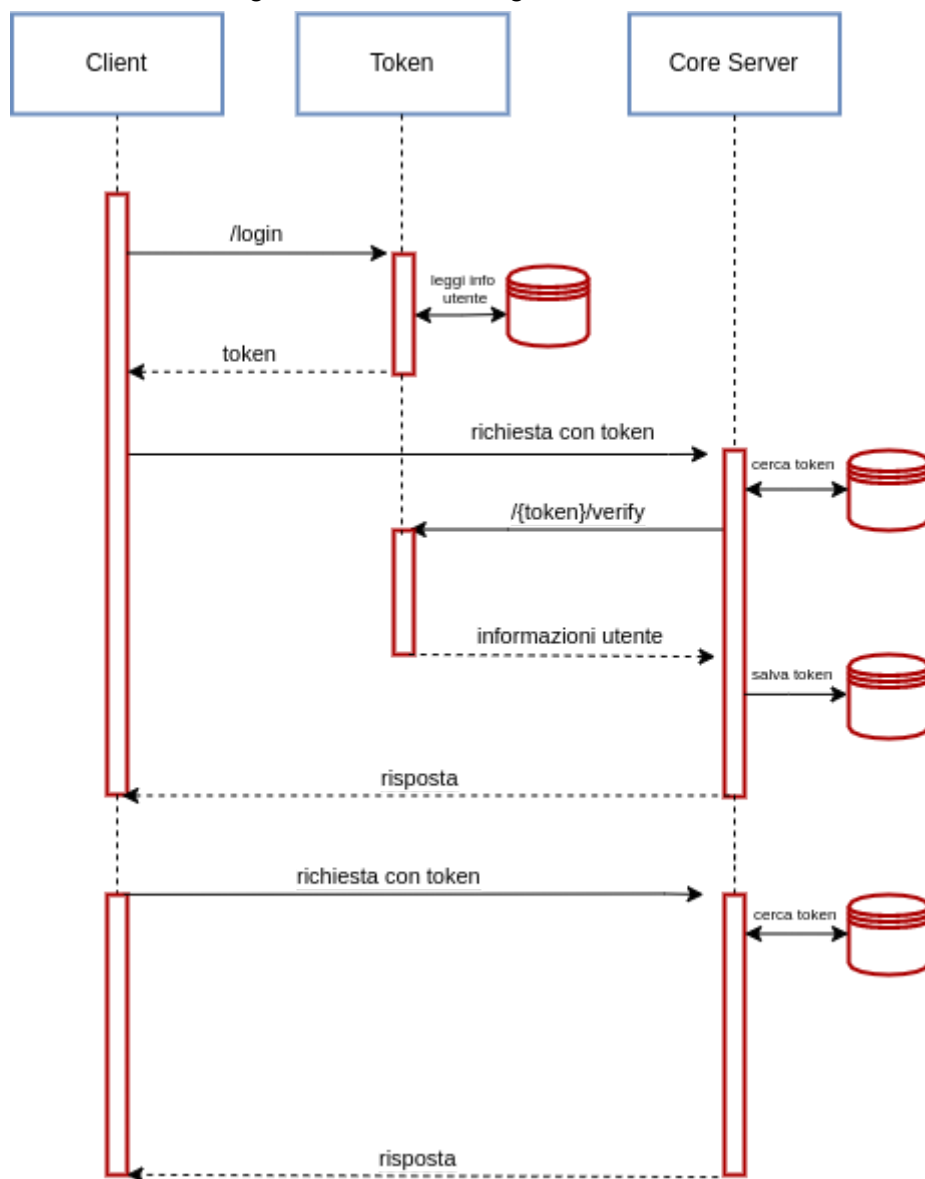


Figure 5: Diagramma di una sequenza di login

## Richiesta mappa

Si ipotizza che l'utente abbia già effettuato il login e che nel Core Server ci siano già le informazioni relative all'utente e al token.

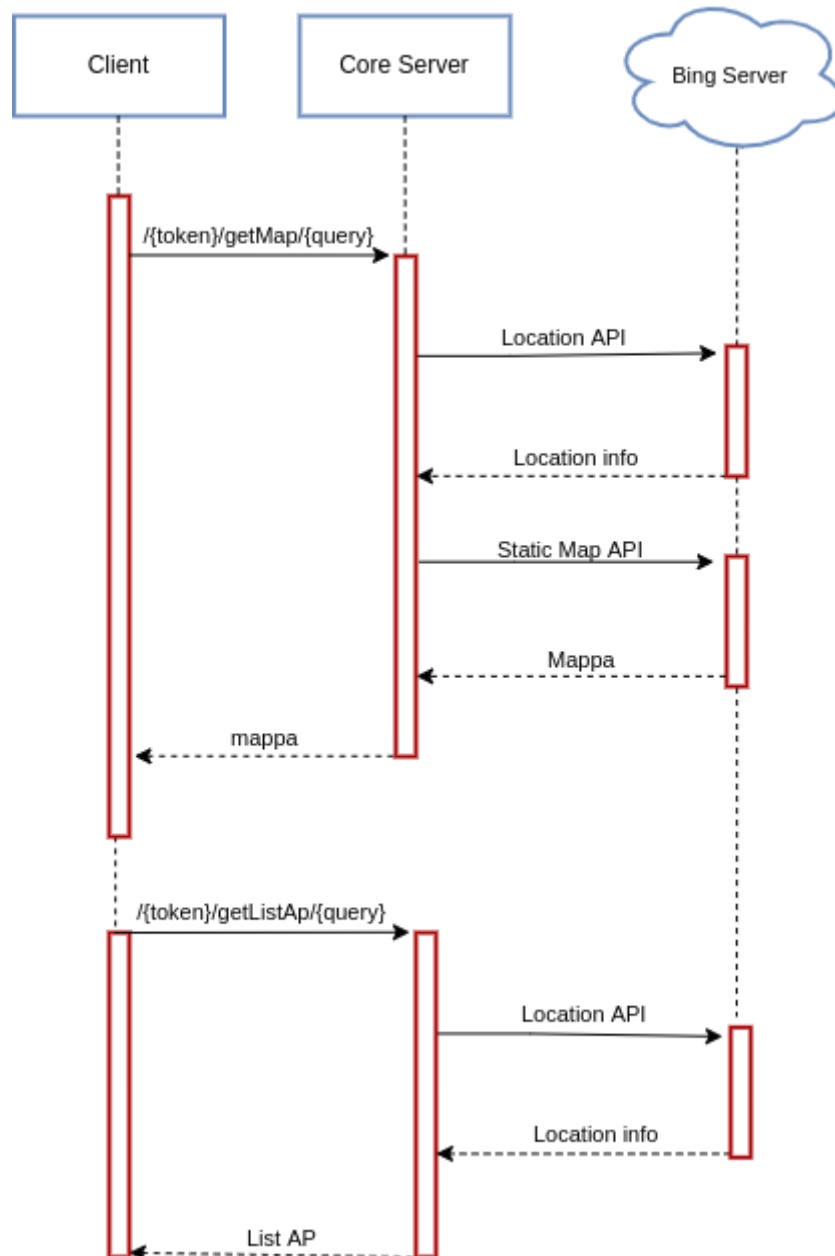


Figure 6: Diagramma di una sequenza di richiesta mappa tramite indirizzo

## Richiesta Upgrade

Si ipotizza che l'utente abbia già effettuato il login e che nel Core Server ci siano già le informazioni relative all'utente e al token.

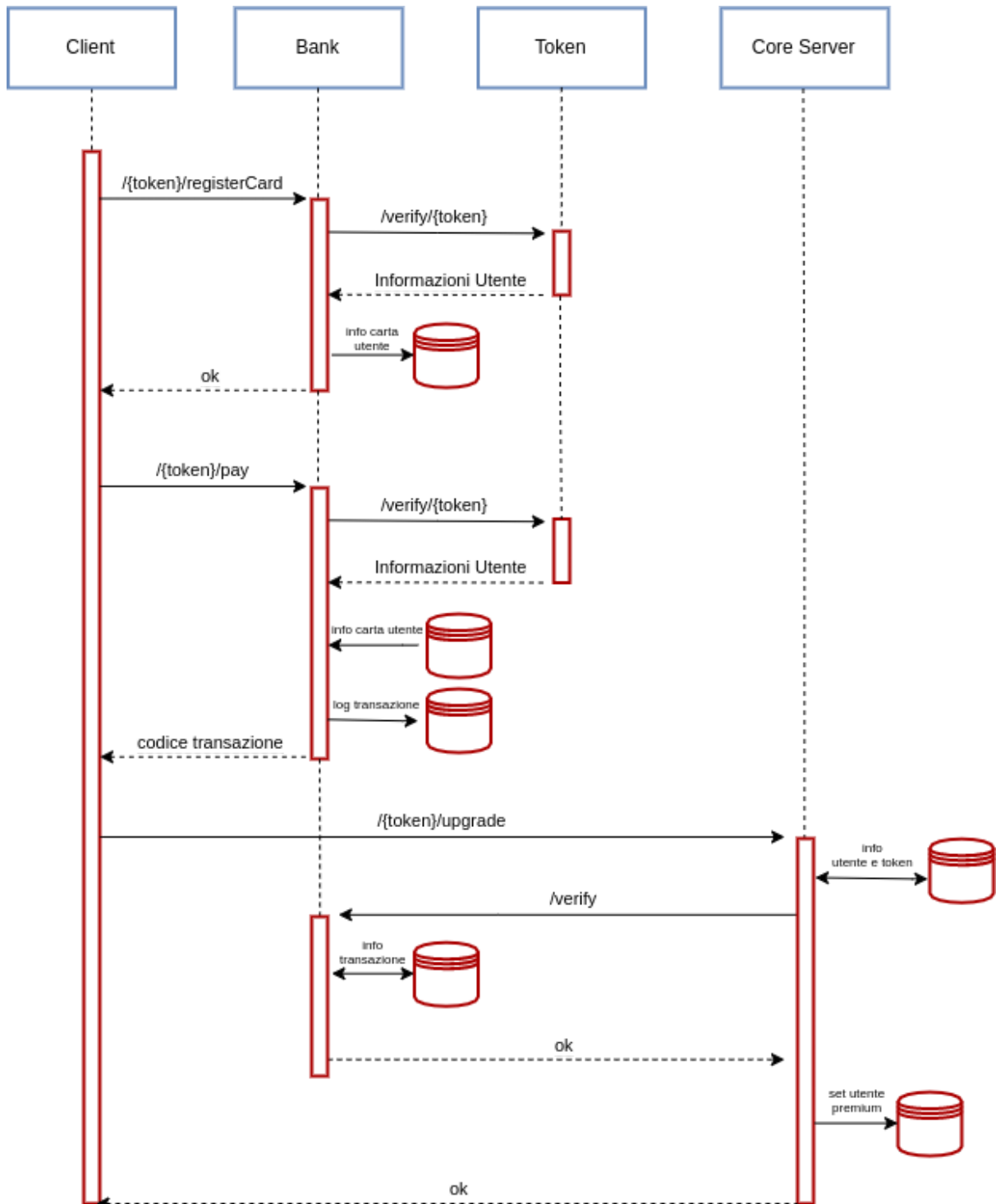


Figure 7: Diagramma della sequenza di una richiesta di upgrade a premium

## Connessione Access Point

Si ipotizza che l'utente non sia un utente premium e abbia già effettuato il login, che nel Core Server ci siano già le informazioni relative all'utente e al token e che abbia già precedentemente registrato sul Core Server le informazioni riguardo il contatore.

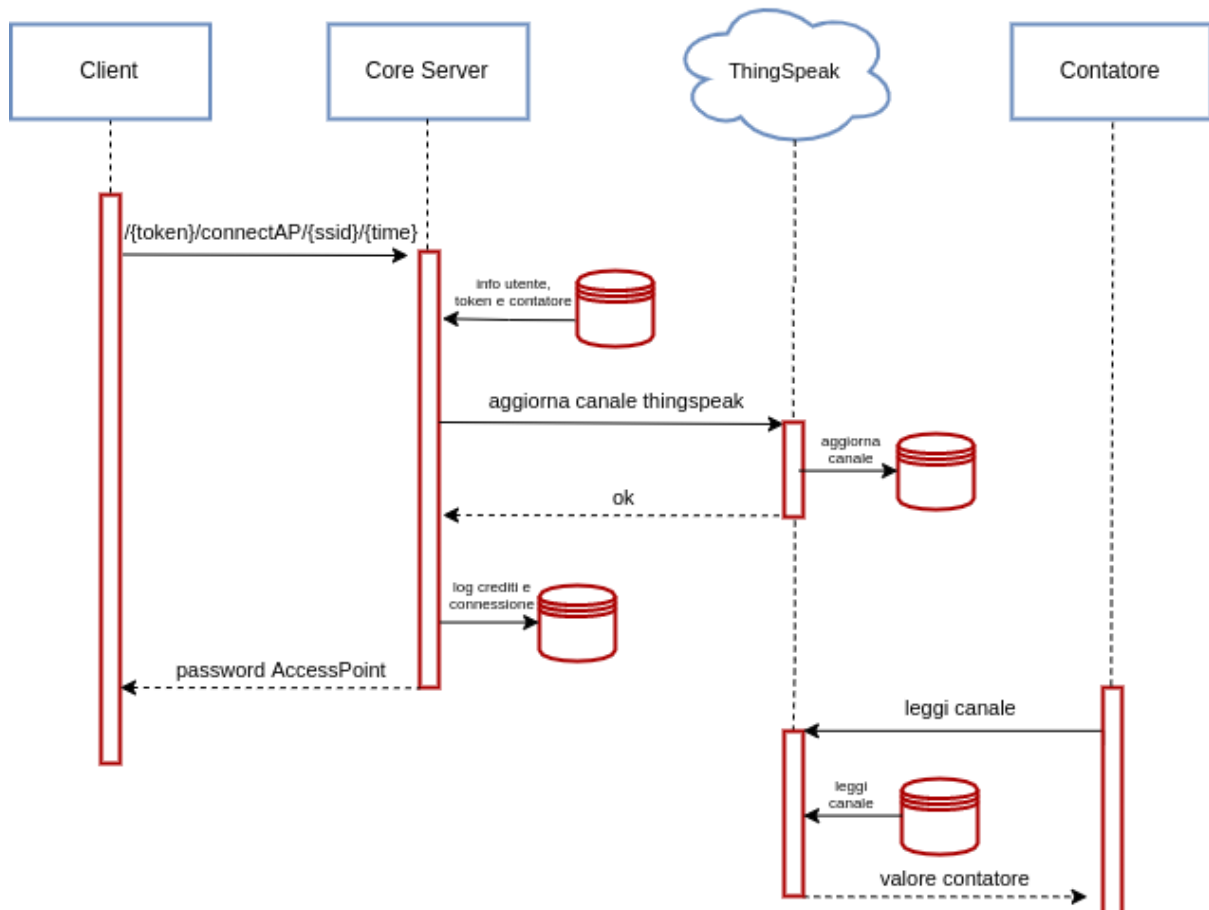


Figure 8: Diagramma di una sequenza di connessione ad un AccessPoint

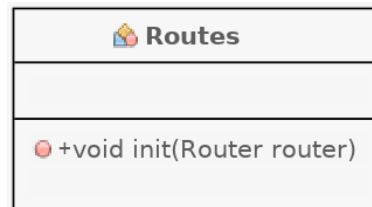
# Class Diagram

Ogni paragrafo di questa sezione descrive le classi del relativo progetto java. Ogni paragrafo è ulteriormente suddiviso seguendo la suddivisione interna dei package del progetto.

## Server Token

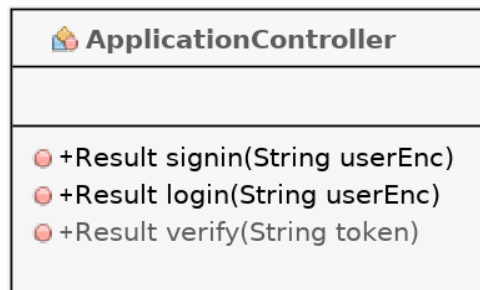
### conf

queste classi sono necessarie al funzionamento di un qualsiasi progetto ninja. La classe route conserva tutte le route http nel metodo init.



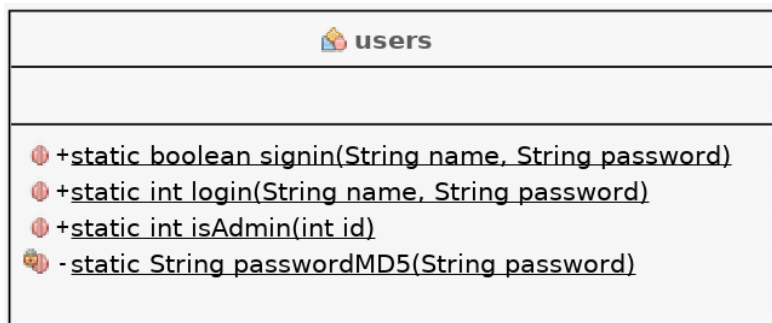
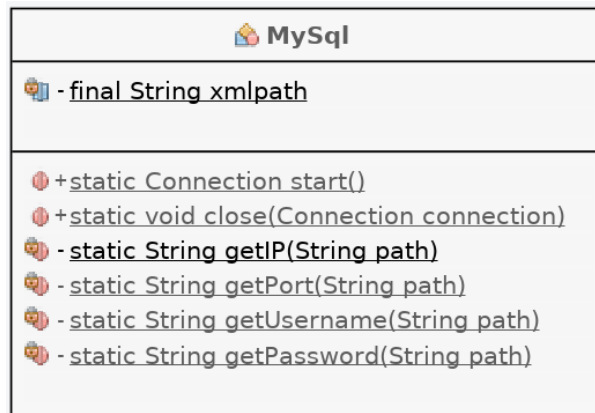
### controllers

le classi classi controller contengono i metodi che vengono richiamati dalle route http



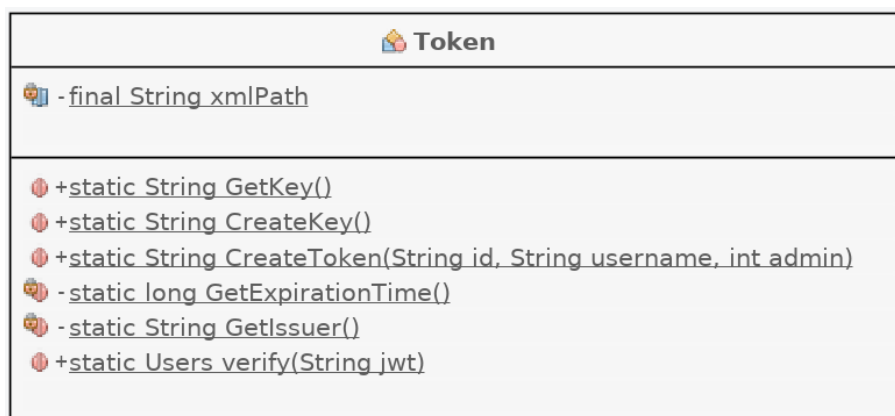
## mysql

queste classi si occupano di prendere informazioni dal database o inserirle.



## utility

la classe token contiene tutti i metodi per che restituiscono le informazioni all'interno dei file di configurazione relativi ai token.

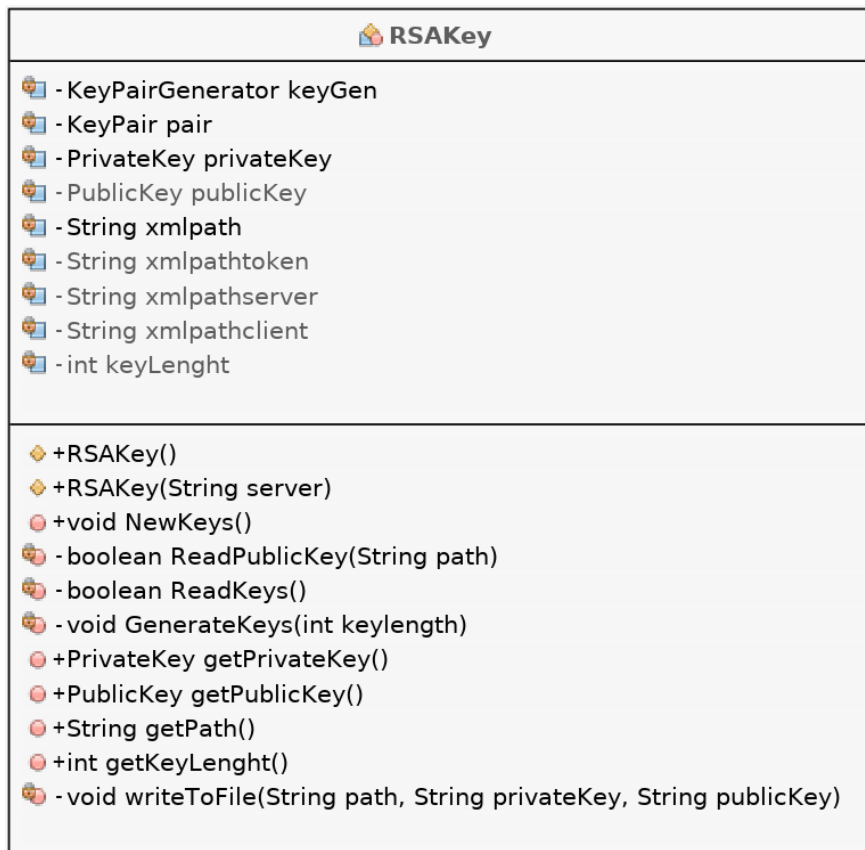


## crittography

le classi in questo package servono per ciò che riguarda la crittografia.

**RSAKey:** contiene tutti i metodi per leggere dai file di configurazione le chiavi.

- RSAKey() inizializza le variabili private con le proprie chiavi e se non trova il proprio file di chiavi rsa ne genera di nuove
- RSAKey(String server) inizializza le variabili private con le chiavi relative al progetto richiesto in argomento (client, bank, server, token).

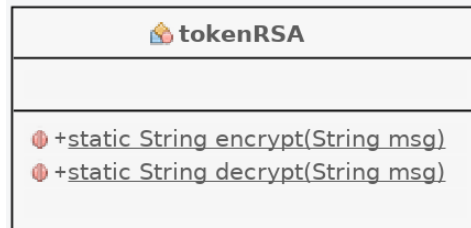


**RSA:** contiene i metodi utili a cifrare e decifrare un messaggio con una chiave data.



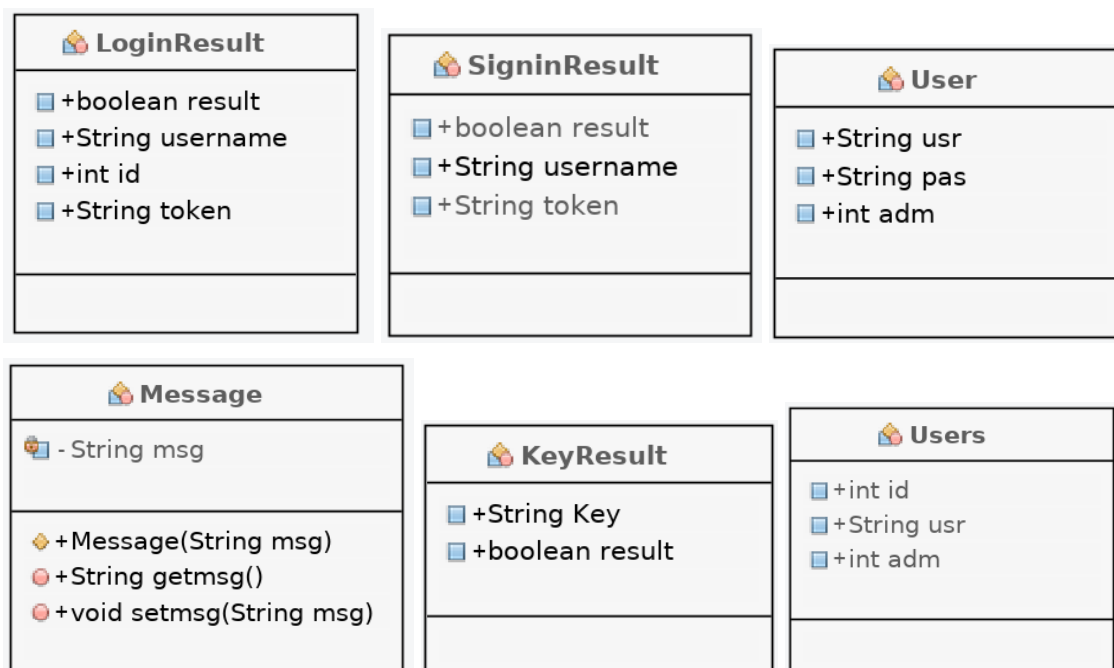


**tokenRSA:** contiene i metodi utili a cifrare con la chiave pubblica del client e a decifrare con la propria chiave privata.



## POJO

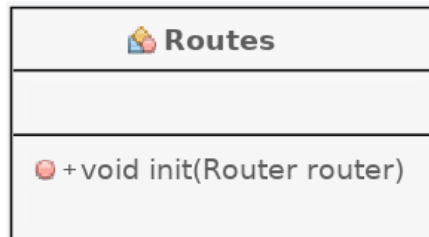
Queste Classi sono utilizzate solamente per rappresentare i dati inviati e/o ricevuti in formato json. Per definizione non sono legati ad alcuna restrizione diversa da quelle costrette dalla specifica del linguaggio Java.



# Server Banca

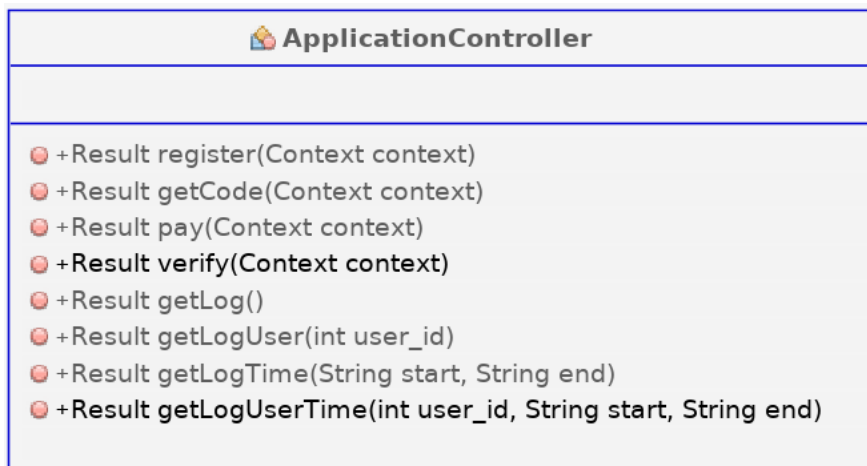
## conf

queste classi sono necessarie al funzionamento di un qualsiasi progetto ninja. La classe route conserva tutte le route http nel metodo init.



## controllers

le classi controller contengono i metodi che vengono richiamati dalle route http

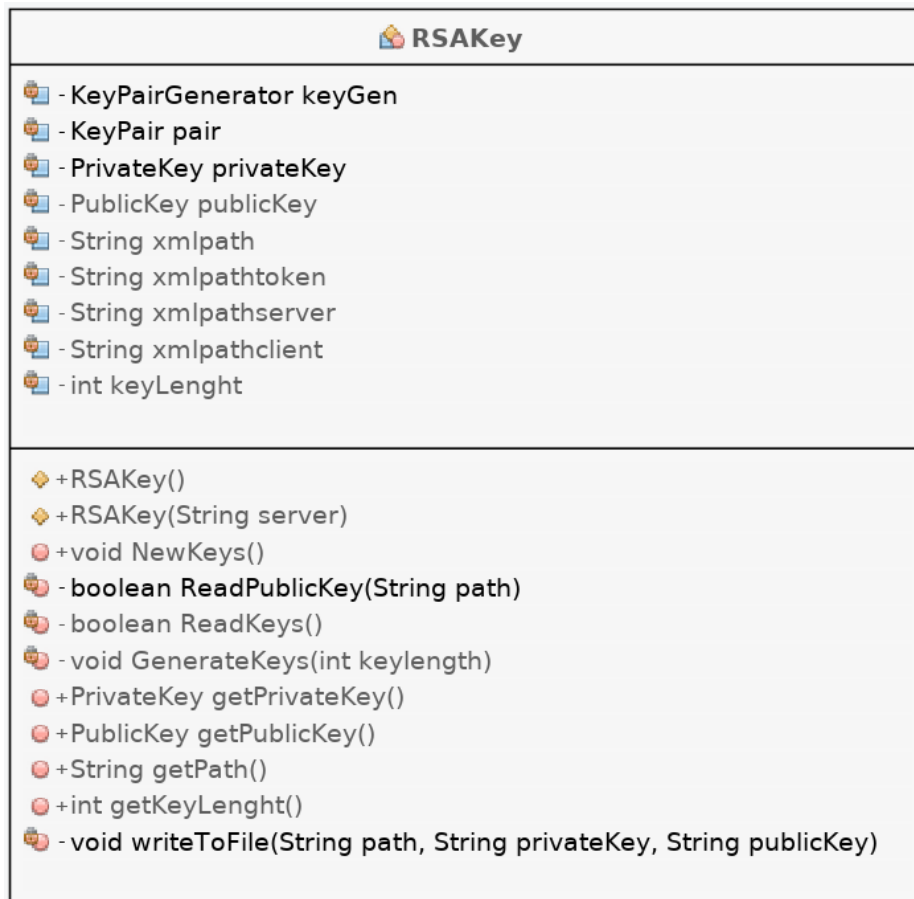


## crittography

le classi in questo package servono per ciò che riguarda la crittografia.

**RSAPKey:** contiene tutti i metodi per leggere dai file di configurazione le chiavi.

- RSAPKey() inizializza le variabili private con le proprie chiavi e se non trova il proprio file di chiavi rsa ne genera di nuove.
- RSAPKey(String server) inizializza le variabili private con le chiavi relative al progetto richiesto in argomento (client, bank, server, token).



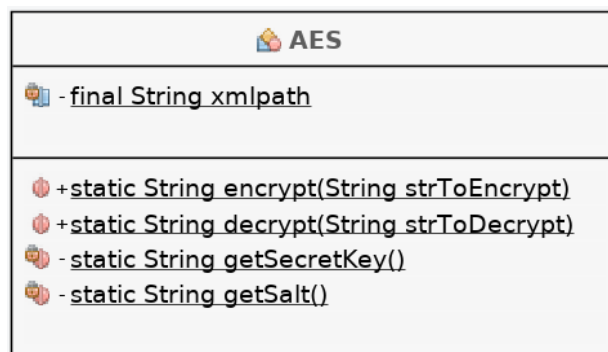
**RSA:** contiene i metodi utili a cifrare e decifrare un messaggio con una chiave data.



**serverRSA:** contiene i metodi utili a cifrare con la chiave pubblica del client e a decifrare con la propria chiave privata.



**AES:** contiene i metodi utili a crittografare e decrittografare una stringa con la relativa chiave presente nel file di configurazione



### filters

in questo package vi sono i filtri, classi con solamente un metodo che possono essere applicati alle route prima che venga chiamato il controller.

**AES\_decode\_card:** controlla il body della richiesta, la decifra con l’algoritmo di AES256, ed estrae le informazioni riguardanti la carta di credito. Si ipotizza che siano state formattate in json secondo la classe POJO.Card

Se dovesse riscontrare qualche problema restituisce errore 401.





**AES\_decode\_transaction:** controlla il body della richiesta, la decifra con l’algoritmo di AES256, ed estrae le informazioni riguardanti le transazioni. Si ipotizza che siano state formattate in json secondo la classe POJO.Transaction.

Se dovesse riscontrare qualche problema restituisce errore 401.

 <b>AES_decode_transaction</b>
 +Result filter(FilterChain chain, Context context)



**RSA\_tokenVerify:** decifra con l'algorithmo RSA il token, controlla sul db se è già stato precedentemente utilizzato da un utente e se non è scaduto. Se è così prende le informazioni dell'utente. Se il token non si trova nel db richiede al server token di verificarlo. Questo restituirà le informazioni dell'utente. Quindi se non è presente nel db verrà aggiunto. In ogni caso verrà inserito il token nel db e calcolata la data di scadenza secondo i parametri inseriti nei file di configurazione.

Se dovesse riscontrare qualche problema restituisce errore 401.

 <b>RSA_tokenVerify</b>
 +Result filter(FilterChain chain, Context context)

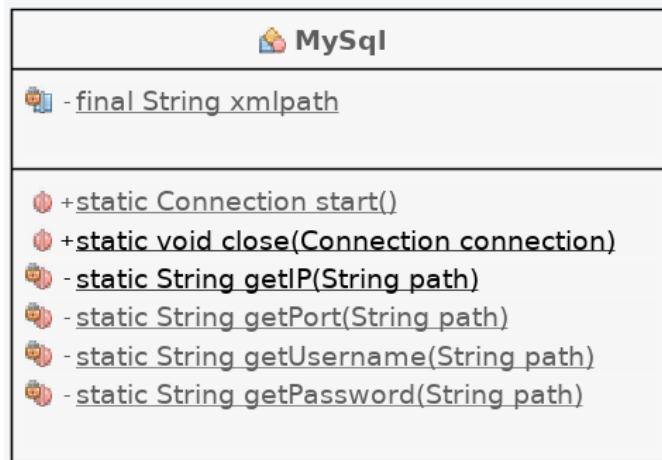
**isAdmin:** controlla che l'utente che ha effettuato la richiesta sia un admin. Il filtro deve essere necessariamente chiamato successivamente al filtro RSA\_tokenVerify, da cui ricava le informazioni dell'utente.

Se dovesse riscontrare qualche problema restituisce errore 401.

 <b>isAdmin</b>
 +Result filter(FilterChain chain, Context context)

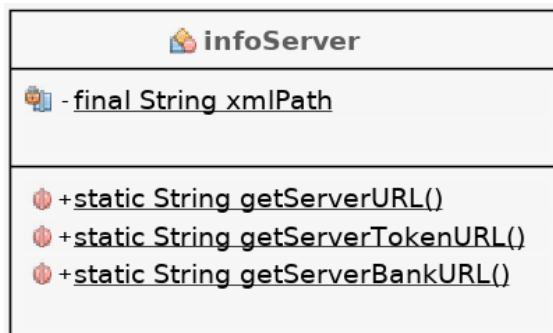
## mysql

La classe **MySql** si occupa di leggere le informazioni del db dai file di configurazione e fornire metodi per aprire e chiudere la connessione al db.

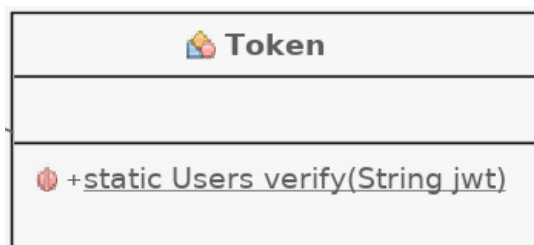


## utility

**infoServer:** restituisce le informazioni sul server contenute nel relativo file di configurazione.



**Token:** contiene il metodo utile alla verifica del token. effettua una richiesta http al server token chiedendone la verifica e restituendone il risultato.



**Account:** contiene i metodi utili alla gestione delle carte di credito degli utenti

- getCode(int user\_id) restituisce il numero della carta di credito di un utente
- register(...) salva le informazioni relative alla carta di credito sul db

Account
<ul style="list-style-type: none"> <li>+static Boolean register(int user_id, String username, int code, int month, int year, int cvc)</li> <li>+static int getCode(int user_id)</li> </ul>

**HTTP:** contiene i metodi per fare delle richieste http.

HTTP
<ul style="list-style-type: none"> <li>+static String GET(String urlString, String msg)</li> <li>+static String GET(String urlString)</li> <li>+static String PUT(String urlString, String msg)</li> <li>+static String PUT(String urlString)</li> <li>+static String POST(String urlString, String msg)</li> <li>+static String POST(String urlString)</li> <li>+static String DELETE(String urlString, String msg)</li> <li>+static String DELETE(String urlString)</li> <li>+static String HEAD(String urlString, String msg)</li> <li>+static String HEAD(String urlString)</li> <li>+static String OPTIONS(String urlString, String msg)</li> <li>+static String OPTIONS(String urlString)</li> <li>+static String PATCH(String urlString, String msg)</li> <li>+static String PATCH(String urlString)</li> <li>-static String request(String urlString, String method, String msg)</li> <li>-static String request(String urlString, String method)</li> </ul>

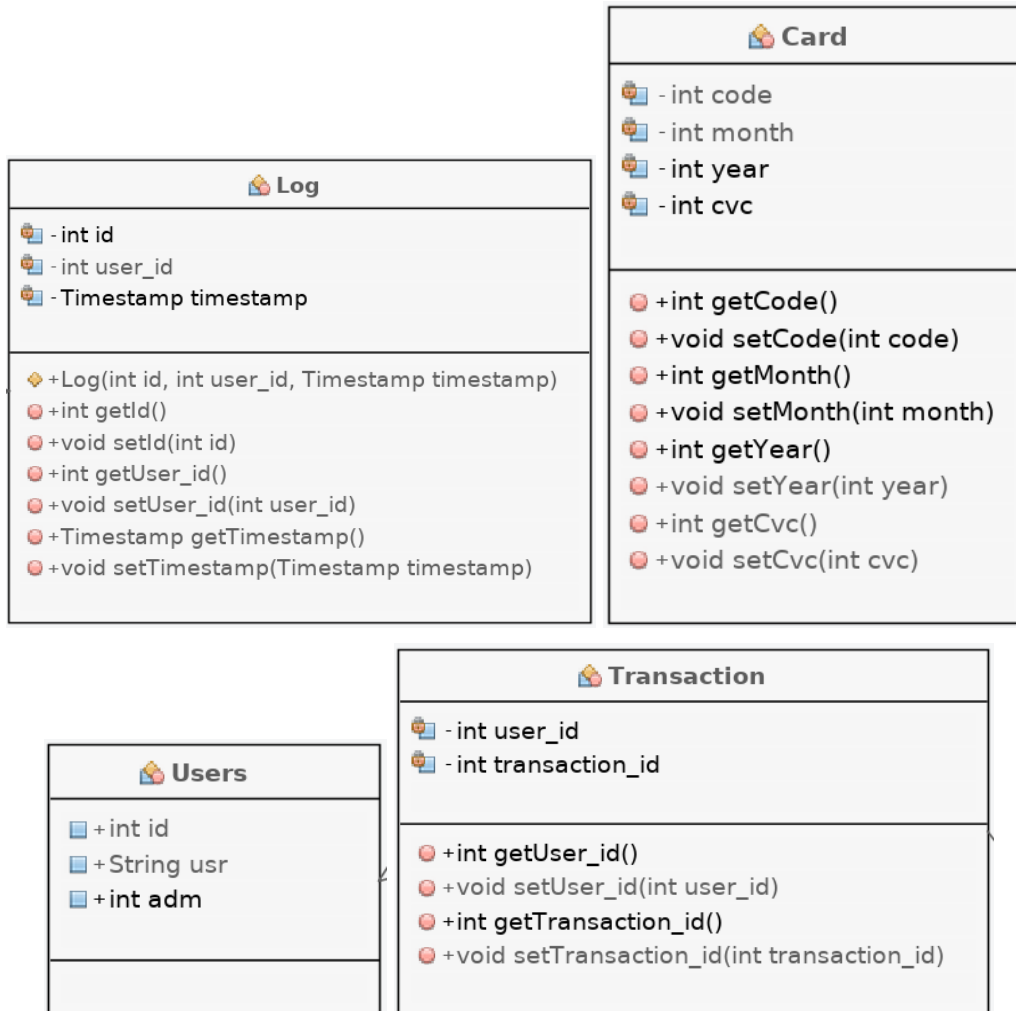
**Transaction:** contiene i metodi necessari alla gestione delle transazioni.

- pay(int user\_id) effettua la transazione relativa all'utente indicato, restituisce il codice di transazione.
- check(...) controlla che il codice di transazione esista e corrisponda all'utente indicato. Ne restituisce la data dell'esecuzione della transazione.

Transaction
<ul style="list-style-type: none"> <li>+static int pay(int user_id)</li> <li>+static Timestamp check(int user_id, int transaction_id)</li> <li>+static List&lt;Log&gt; ListAllLogs()</li> <li>+static List&lt;Log&gt; ListUserLogs(int user_id)</li> <li>+static List&lt;Log&gt; ListDateLogs(Timestamp start, Timestamp end)</li> <li>+static List&lt;Log&gt; ListUserDataLogs(int user_id, Timestamp start, Timestamp end)</li> </ul>

## POJO

Queste Classi sono utilizzate solamente per rappresentare i dati inviati e/o ricevuti in formato json. Per definizione non sono legati ad alcuna restrizione diversa da quelle costrette dalla specifica del linguaggio Java.

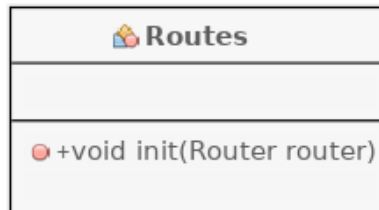




# Core Server

## conf

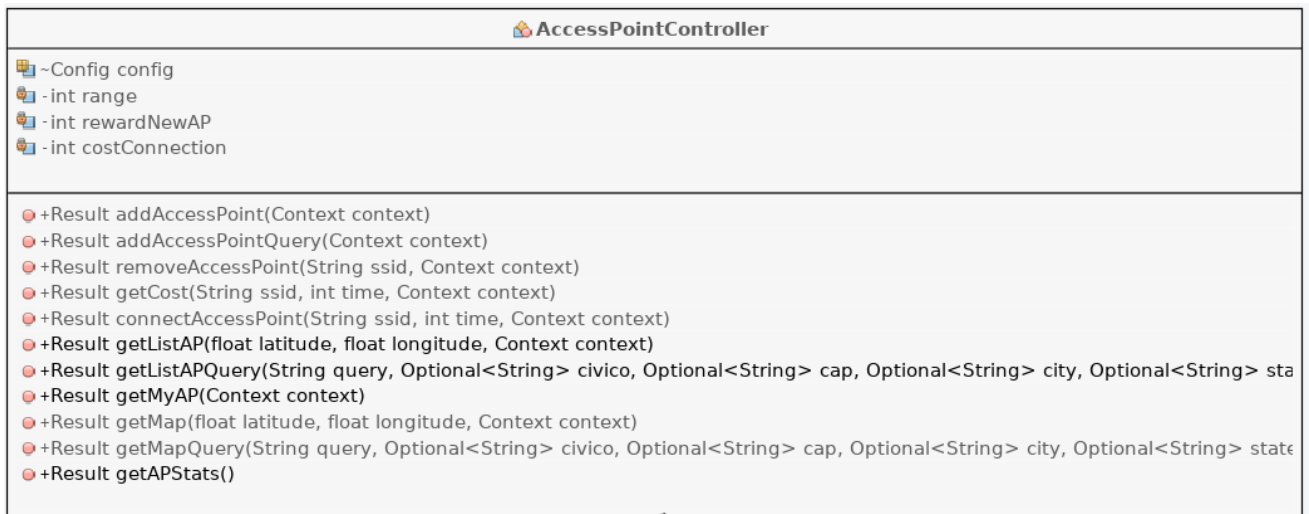
Queste classi sono necessarie al funzionamento di un qualsiasi progetto ninja. La classe route conserva tutte le route http nel metodo init.



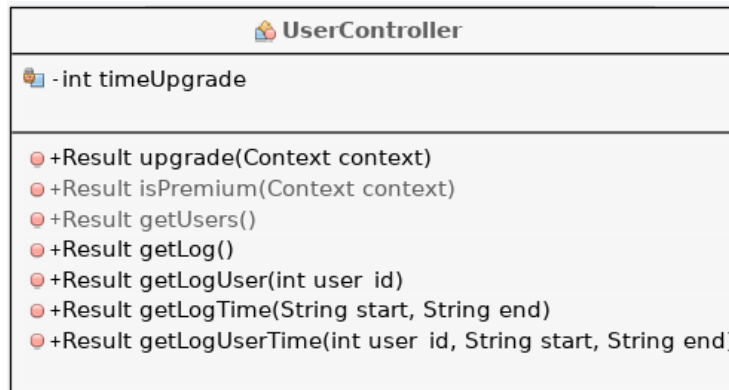
## controllers

Le classi controller contengono i metodi che vengono richiamati dalle route http

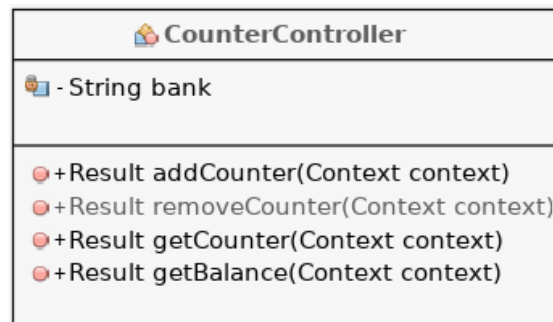
**AccessPointController:** contiene tutti i metodi per gestire le richieste che riguardano gli AccessPoint



**UserController:** contiene tutti i metodi per gestire le richieste che riguardano gli utenti



**CounterController:** contiene tutti i metodi per gestire le richieste che riguardano i contatori

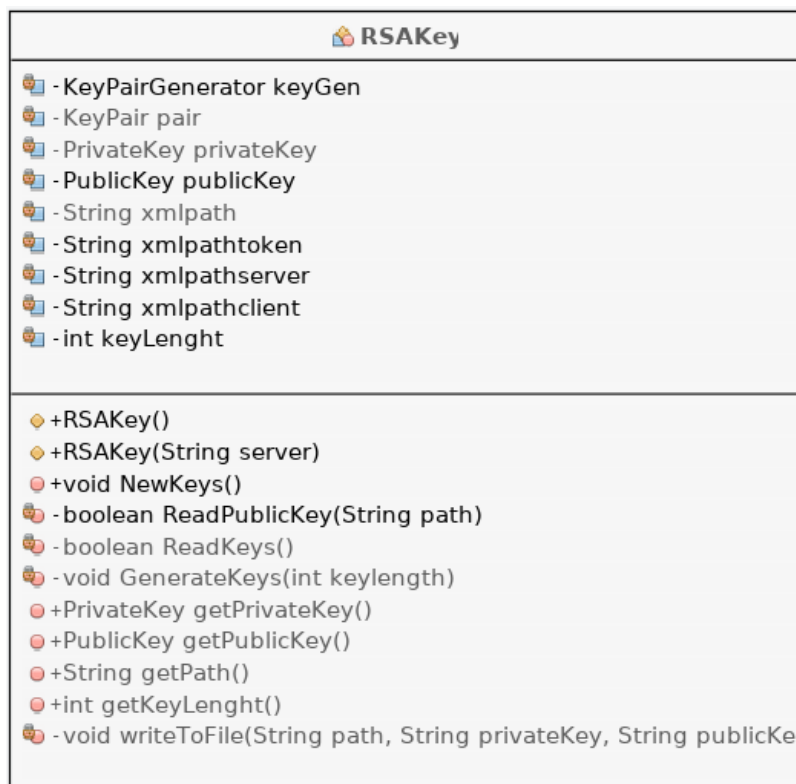


## crittography

le classi in questo package servono per ciò che riguarda la crittografia.

**RSAKey:** contiene tutti i metodi per leggere dai file di configurazione le chiavi.

- RSAKey() inizializza le variabili private con le proprie chiavi e se non trova il proprio file di chiavi rsa ne genera di nuove.
- RSAKey(String server) inizializza le variabili private con le chiavi relative al progetto richiesto in argomento (client, bank, server, token)



**RSA:** contiene i metodi utili a cifrare e decifrare un messaggio con una chiave data.



**serverRSA:** :contiene i metodi utili a cifrare con la chiave pubblica del client e a decifrare con la propria chiave privata.



**AES:** contiene i metodi utili a crittografare e decrittografare una stringa con la relativa chiave presente nel file di configurazione



## filters

in questo package vi sono i filtri, classi con solamente un metodo che possono essere applicati alle route prima che venga chiamato il controller.

**AES\_decode\_transaction:** controlla il body della richiesta, la decifra con l'algoritmo di AES256, ed estrae le informazioni riguardanti le transazioni. Si ipotizza che siano state formattate in json secondo la classe POJO.Transaction.

Se dovesse riscontrare qualche problema restituisce errore 401.



**RSA\_tokenVerify:** decifra con l'algoritmo RSA il token, controlla sul db se è già stato precedentemente utilizzato da un utente e se non è scaduto. Se è così prende le informazioni dell'utente. Se il token non si trova nel db richiede al server token di verificarlo.

Questo restituirà le informazioni dell'utente. Quindi se non è presente nel db verrà aggiunto. In ogni caso verrà inserito il token nel db e calcolata la data di scadenza secondo i parametri inseriti nei file di configurazione.

Se dovesse riscontrare qualche problema restituisce errore 401.

 <b>RSA tokenVerify</b>
 +Result filter(FilterChain chain, Context context)

**AES\_decode\_requestAP:** controlla il body della richiesta, la decifra con l'algoritmo di AES256, ed estrae le informazioni riguardanti gli AccessPoint. Si ipotizza che siano state formattate in json secondo la classe POJO.AccessPoint

Se dovesse riscontrare qualche problema restituisce errore 401.

 <b>AES_decode_requestAP</b>
 +Result filter(FilterChain chain, Context context)

**isAdmin:** controlla che l'utente che ha effettuato la richiesta sia un admin. Il filtro deve essere necessariamente chiamato successivamente al filtro RSA\_tokenVerify, da cui ricava le informazioni dell'utente.

Se dovesse riscontrare qualche problema restituisce errore 401.

 <b>isAdmin</b>
 +Result filter(FilterChain chain, Context context)








**AES\_decode\_requestCounter:** controlla il body della richiesta, la decifra con l'algoritmo di AES256, ed estrae le informazioni riguardanti i contatori. Si ipotizza che siano state formattate in json secondo la classe POJO.Counter.

Se dovesse riscontrare qualche problema restituisce errore 401.

 <b>AES_decode_requestCounter</b>
 +Result filter(FilterChain chain, Context context)

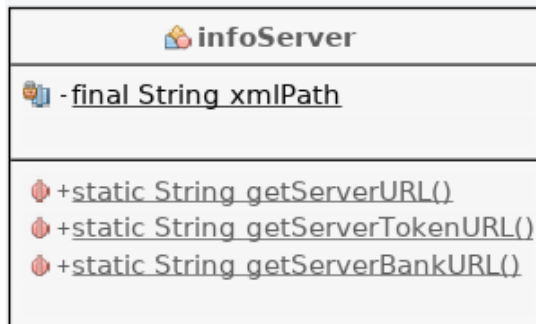
## mysql

La classe **MySql** si occupa di leggere le informazioni del db dai file di configurazione e fornire metodi per aprire e chiudere la connessione al db.

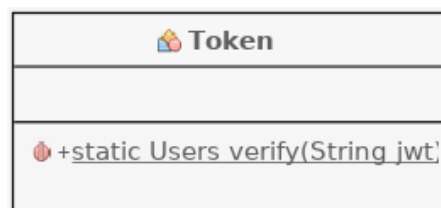
MySql
 - final String xmlpath
 +static Connection start()  +static void close(Connection connection)  - static String getIP(String path)  - static String getPort(String path)  - static String getUsername(String path)  - static String getPassword(String path)

## utility

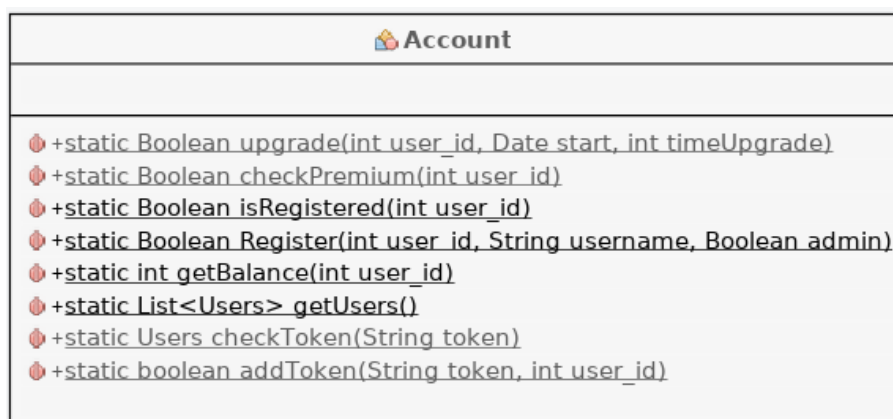
**infoServer:** restituisce le informazioni sul server contenute nel relativo file di configurazione.



**Token:** contiene il metodo utile alla verifica del token. effettua una richiesta http al server token chiedendone la verifica e restituendone il risultato.



**Account:** contiene i metodi utili alla gestione degli utenti, come la registrazione dell'utente nel db, l'esecuzione dell'upgrade a utente premium ecc..



**HTTP:** contiene i metodi per fare delle richieste http.

📁 HTTP
<ul style="list-style-type: none"> <li>🔑 +static String GET(String urlString, String msg)</li> <li>🔑 +static String GET(String urlString)</li> <li>🔑 +static String PUT(String urlString, String msg)</li> <li>🔑 +static String PUT(String urlString)</li> <li>🔑 +static String POST(String urlString, String msg)</li> <li>🔑 +static String POST(String urlString)</li> <li>🔑 +static String DELETE(String urlString, String msg)</li> <li>🔑 +static String DELETE(String urlString)</li> <li>🔑 +static String HEAD(String urlString, String msg)</li> <li>🔑 +static String HEAD(String urlString)</li> <li>🔑 +static String OPTIONS(String urlString, String msg)</li> <li>🔑 +static String OPTIONS(String urlString)</li> <li>🔑 +static String PATCH(String urlString, String msg)</li> <li>🔑 +static String PATCH(String urlString)</li> <li>🔑 - static String request(String urlString, String method, String msg)</li> <li>🔑 - static String request(String urlString, String method)</li> </ul>

**Config:** il costruttore legge il file config.json e inizializza le variabili.

📁 Config
<ul style="list-style-type: none"> <li>📄 - String jsonPath</li> <li>📄 - int range</li> <li>📄 - int rewardNewAP</li> <li>📄 - int costConnection</li> <li>📄 - int timeUpgrade</li> <li>📄 - int expirationToken</li> </ul>
<ul style="list-style-type: none"> <li>🔑 +Config()</li> <li>🔑 +int getRange()</li> <li>🔑 +int getRewardNewAP()</li> <li>🔑 +int getCostConnection()</li> <li>🔑 +int getTimeUpgrade()</li> <li>🔑 +int getExpirationToken()</li> </ul>

**Logs:** contiene i metodi utili a leggere i log e a inserirli nel db.



📦 Logs
<ul style="list-style-type: none"> <li>🔑 +static List&lt;Log&gt; ListAllLogs()</li> <li>🔑 +static List&lt;Log&gt; ListUserLogs(int user_id)</li> <li>🔑 +static List&lt;Log&gt; ListDateLogs(Timestamp start, Timestamp end)</li> <li>🔑 +static List&lt;Log&gt; ListUserDateLogs(int user_id, Timestamp start, Timestamp end)</li> <li>🔑 +static boolean AddLog(Log log)</li> <li>🔑 +static List&lt;APStats&gt; getAPStats()</li> </ul>

**ThingSpeaks:** una volta inizializzate con un costruttore le variabili relative alle API di thingspeak è possibile aggiornare il relativo canale. Inoltre contiene metodi per la gestione dei dispositivi contatore, è infatti possibile inserire un contatore e leggere le informazioni dei contatori.

📦 ThingSpeak
<ul style="list-style-type: none"> <li>📦 - String thingspeak</li> <li>📦 - String api</li> <li>📦 - int field</li> <li>📦 - int channelId</li> <li>📦 - int user id</li> </ul>
<ul style="list-style-type: none"> <li>🔑 +ThingSpeak(int user_id, String api, int field, int channelId)</li> <li>🔑 +ThingSpeak(int user_id)</li> <li>🔑 +ThingSpeak()</li> <li>🔑 +int getUserId()</li> <li>🔑 +String getAPI()</li> <li>🔑 +int getField()</li> <li>🔑 +int getChannelId()</li> <li>🔑 +void setUserId(int id)</li> <li>🔑 +void setAPI(String api)</li> <li>🔑 +void setField(int field)</li> <li>🔑 +void setChannelId(int id)</li> <li>🔑 +Boolean checkAPI()</li> <li>🔑 +Boolean insert()</li> <li>🔑 +Boolean update()</li> <li>🔑 +Boolean select()</li> <li>🔑 +Boolean exist()</li> <li>🔑 +boolean isSet()</li> <li>🔑 +Boolean setValue(int value)</li> <li>🔑 +int getValue()</li> <li>🔑 +static List&lt;ThingSpeak&gt; ListItems()</li> <li>🔑 +static Boolean addLog(int id_user, String username, int id_item, int value)</li> <li>🔑 +Boolean removeItem()</li> </ul>

**Credits:** raccoglie i metodi utili ad aggiungere, sottrarre e trasferire crediti tra utenti. Ogni funzione scrive anche le operazioni eseguite su log, tutto atomicamente, se non dovesse

andare a buon fine una scrittura su db allora viene effettuato il rollback delle operazioni richieste.

Credits
<ul style="list-style-type: none"><li>+static Boolean add(int user_id, int amount)</li><li>+static Boolean sub(int user_id, int amount)</li><li>+static Boolean Transaction(int from, int to, int amount)</li><li>+static int balance(int id)</li></ul>

**AccessPoints:** contiene tutto ciò che può essere utile a gestire gli Access Point. Aggiungerli nel db, rimuoverli, elencarli, ecc...

AccessPoints
<ul style="list-style-type: none"><li>+static boolean add(String ssid, int user_id, float latitude, float longitude)</li><li>+static boolean remove(String ssid)</li><li>+static float getLatitude(String ssid)</li><li>+static float getLongitude(String ssid)</li><li>+static int getUserID(String ssid)</li><li>-static List&lt;AccessPoint&gt; getListAP()</li><li>+static List&lt;AccessPoint&gt; getListAp(float latitude, float longitude, float range)</li><li>+static List&lt;AccessPoint&gt; getListAp(int user_id)</li></ul>

**Calc:** raccoglie metodi di operazioni generiche

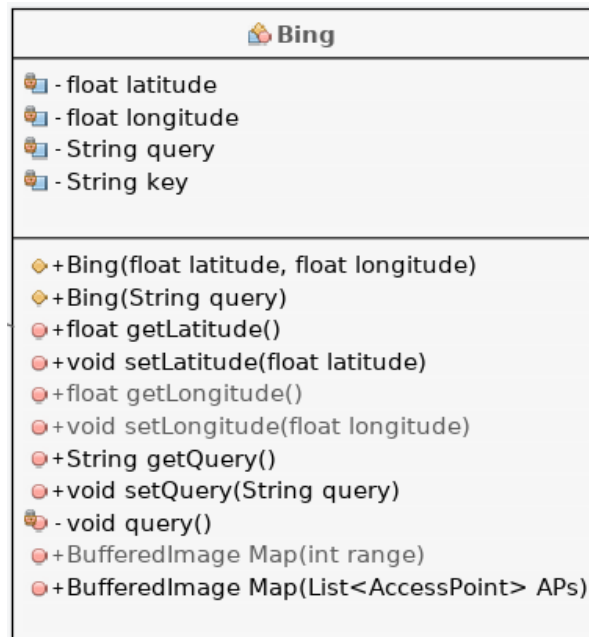
- RandomString(..) restituisce una stringa casuale, il risultato è stato utilizzato come password temporanea per connettersi agli Access Points
- distance(..) restituisce la distanza tra due coordinate. Funzione necessaria per capire se un AP si trova in una determinata area.

Calc
<ul style="list-style-type: none"><li>-final String ALPHA_NUMERIC_STRING</li></ul>
<ul style="list-style-type: none"><li>+static String RandomString(int count)</li><li>+static float distance(float latitude A, float longitude A, float latitude B, float longitude B)</li></ul>

**Bing:** una volta istanziato un oggetto permette di richiedere una mappa utilizzando le api di bing.

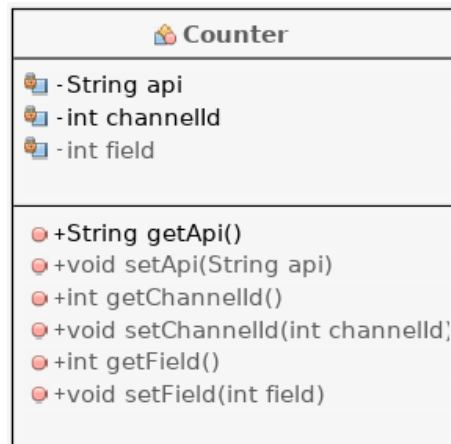
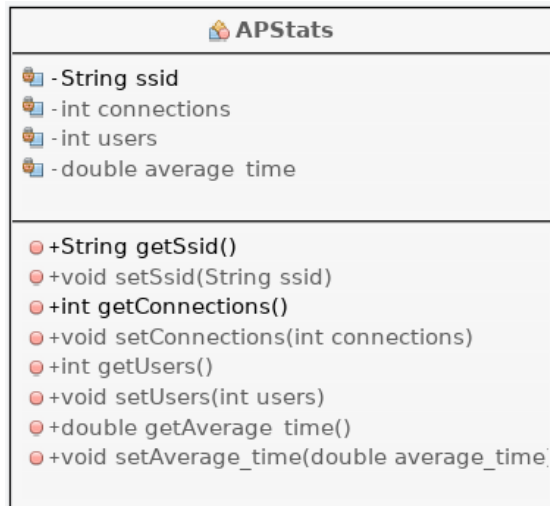
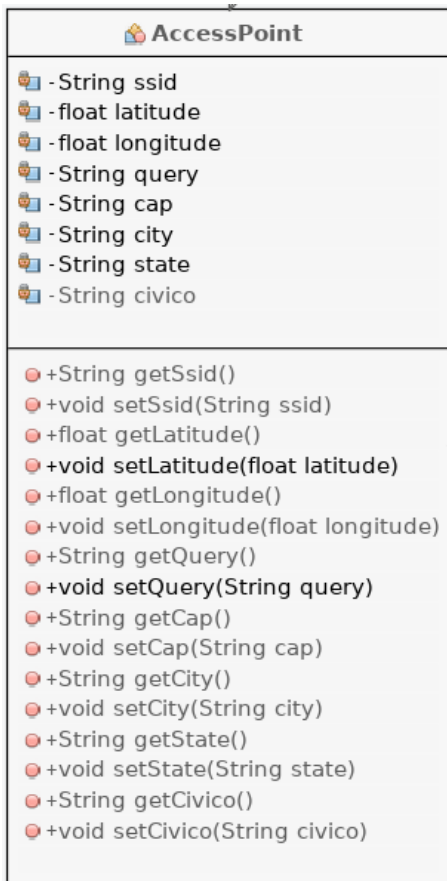
- Bing(String query): è un costruttore che inizializza la variabile query e successivamente utilizza le api di localizzazione di bing per inizializzare le variabili latitude e longitude.

- Bing(float latitude, float longitude): inizializza le coordinate
- una volta inizializzate le variabili latitude e longitude, oppure query, tramite il costruttore designato è possibile richiamare le diverse api
- Map(List<AccessPoint> ap): richiama le api di bing e restituisce una mappa con centro le coordinate latitude e longitude e con indicati tutti gli Access Point in argomento.




















## POJO

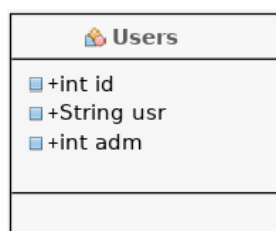
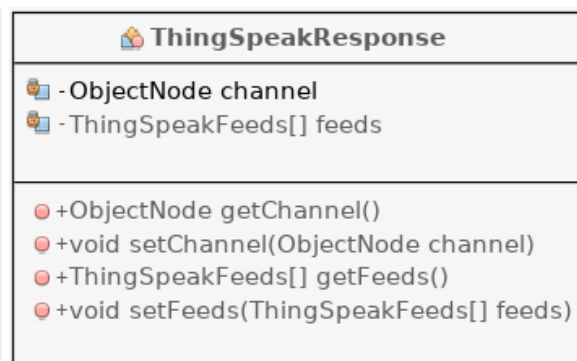
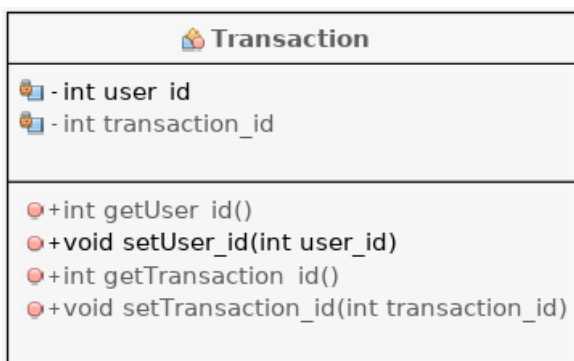
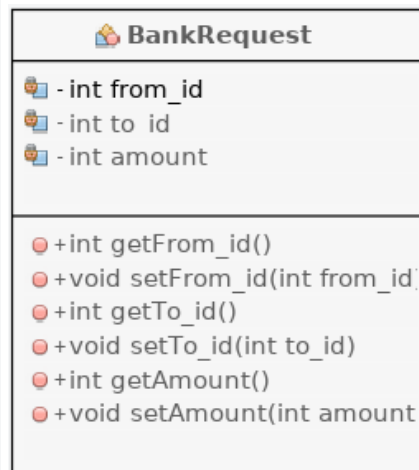
Queste Classi sono utilizzate solamente per rappresentare i dati inviati e/o ricevuti in formato json. Per definizione non sono legati ad alcuna restrizione diversa da quelle costrette dalla specifica del linguaggio Java.



## Log

 -int user\_id  
 -String ssid  
 -int time  
 -Boolean premium  
 -Timestamp timestamp

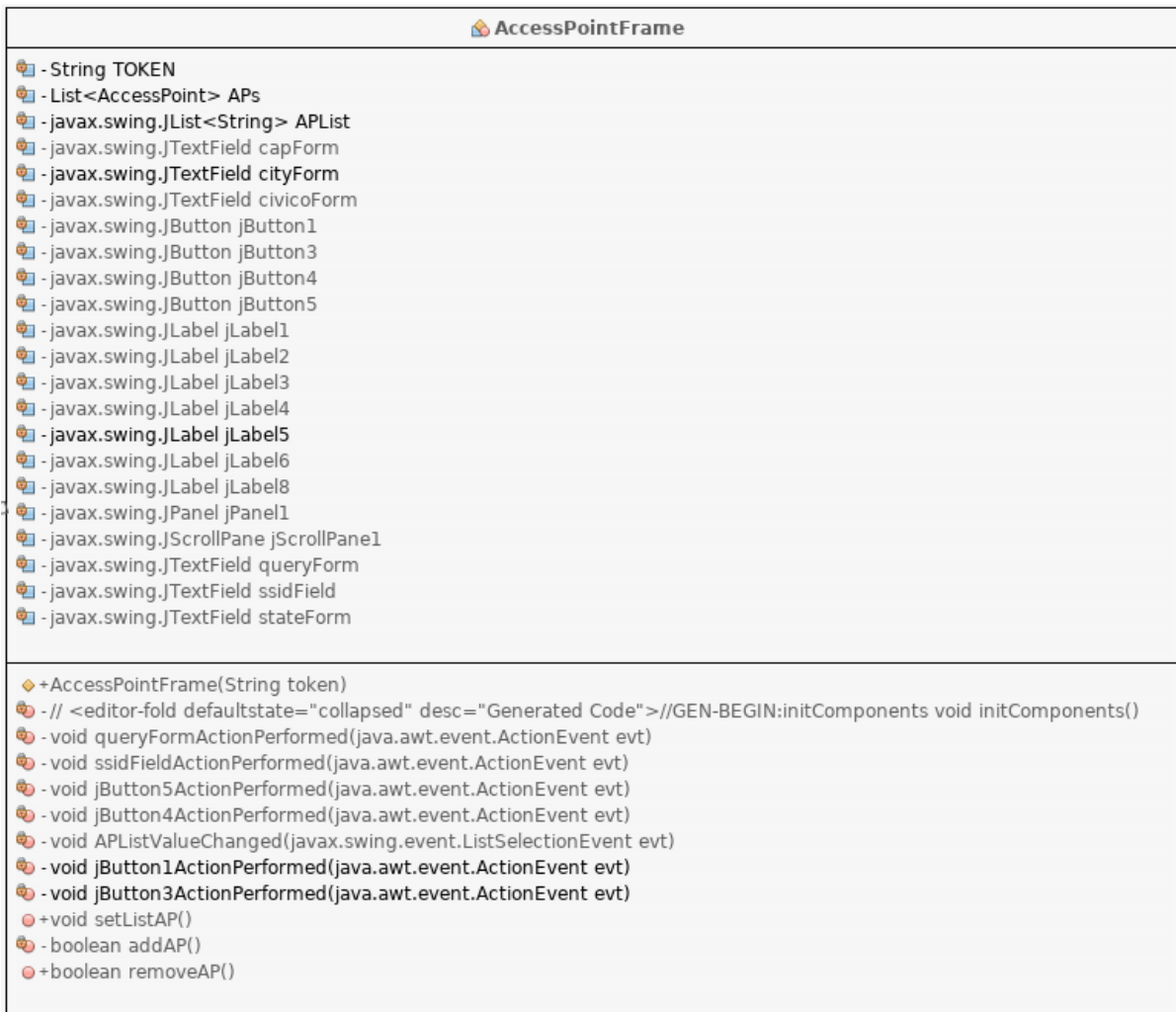
 +Log(int user\_id, String ssid, int time, Boolean premium)  
 +Log(int user\_id, String ssid, int time, Boolean premium, Timestamp timestamp)  
 +int getUser\_id()  
 +void setUser\_id(int user\_id)  
 +String getSSID()  
 +void setSSID(String ssid)  
 +int getTime()  
 +void setTime(int time)  
 +Boolean getPremium()  
 +void setPremium(Boolean premium)  
 +Timestamp getTimestamp()  
 +void setTimestamp(Timestamp timestamp)



# Client

## GUI

**AccessPointFrame:** implementa l'interfaccia per gestire i propri AccessPoint: aggiungerli, rimuoverli e visualizzarli.



The screenshot displays the class structure and methods for `AccessPointFrame`. The class contains several attributes and methods:

- Attributes:
  - String TOKEN
  - List<AccessPoint> APs
  - javax.swing.JList<String> APList
  - javax.swing.JTextField capForm
  - javax.swing.JTextField cityForm
  - javax.swing.JTextField civicoForm
  - javax.swing.JButton jButton1
  - javax.swing.JButton jButton3
  - javax.swing.JButton jButton4
  - javax.swing.JButton jButton5
  - javax.swing.JLabel jLabel1
  - javax.swing.JLabel jLabel2
  - javax.swing.JLabel jLabel3
  - javax.swing.JLabel jLabel4
  - javax.swing.JLabel jLabel5
  - javax.swing.JLabel jLabel6
  - javax.swing.JLabel jLabel8
  - javax.swing.JPanel jPanel1
  - javax.swing.JScrollPane jScrollPane1
  - javax.swing.JTextField queryForm
  - javax.swing.JTextField ssidField
  - javax.swing.JTextField stateForm
- Methods:
  - + AccessPointFrame(String token)
  - // <editor-fold defaultstate="collapsed" desc="Generated Code"> // GEN-BEGIN: initComponents void initComponents()
  - void queryFormActionPerformed(java.awt.event.ActionEvent evt)
  - void ssidFieldActionPerformed(java.awt.event.ActionEvent evt)
  - void jButton5ActionPerformed(java.awt.event.ActionEvent evt)
  - void jButton4ActionPerformed(java.awt.event.ActionEvent evt)
  - void APListValueChanged(javax.swing.event.ListSelectionEvent evt)
  - void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
  - void jButton3ActionPerformed(java.awt.event.ActionEvent evt)
  - + void setListAP()
  - boolean addAP()
  - + boolean removeAP()

**BankRegisterDialog:** implementa il dialog bloccante che permette di registrare una nuova carta di credito sul server della banca.

BankRegisterDialog
<ul style="list-style-type: none"> <li>- String TOKEN</li> <li>- javax.swing.JTextField cardField</li> <li>- javax.swing.JTextField cvcField</li> <li>- javax.swing.JButton jButton1</li> <li>- javax.swing.JLabel jLabel1</li> <li>- javax.swing.JLabel jLabel2</li> <li>- javax.swing.JLabel jLabel3</li> <li>- javax.swing.JLabel jLabel4</li> <li>- javax.swing.JLabel jLabel6</li> <li>- javax.swing.JPanel jPanel1</li> <li>- javax.swing.JPanel jPanel2</li> <li>- javax.swing.JPanel jPanel3</li> <li>- javax.swing.JComboBox&lt;String&gt; monthField</li> <li>- javax.swing.JComboBox&lt;String&gt; yearField</li> </ul>
<ul style="list-style-type: none"> <li>+BankRegisterDialog(java.awt.Frame parent, boolean modal, String token)</li> <li>// &lt;editor-fold defaultstate="collapsed" desc="Generated Code"&gt;//GEN-BEGIN: initComponents void initComponents()</li> <li>- void cardFieldActionPerformed(java.awt.event.ActionEvent evt)</li> <li>- void yearFieldActionPerformed(java.awt.event.ActionEvent evt)</li> <li>- void monthFieldActionPerformed(java.awt.event.ActionEvent evt)</li> <li>- void jButton1ActionPerformed(java.awt.event.ActionEvent evt)</li> </ul>

**ConnectAPDialog:** implementa il dialog bloccante che permette di scegliere il tempo di connessione ad un AccessPoint e richiedere la connessione. Inoltre fornisce la possibilità di effettuare l'upgrade a premium.

ConnectAPDialog
<ul style="list-style-type: none"> <li>- String TOKEN</li> <li>~java.awt.Frame PARENT</li> <li>- javax.swing.JButton PremiumButton</li> <li>- javax.swing.JSlider TimeSlider</li> <li>- javax.swing.JLabel balanceLabel</li> <li>- javax.swing.JButton connectButton</li> <li>- javax.swing.JLabel hoursLabel</li> <li>- javax.swing.JLabel jLabel1</li> <li>- javax.swing.JLabel jLabel2</li> <li>- javax.swing.JLabel jLabel4</li> <li>- javax.swing.JPanel jPanel1</li> <li>- javax.swing.JPanel jPanel2</li> <li>- javax.swing.JLabel premiumLabel</li> <li>- javax.swing.JLabel ssidLabel</li> </ul>
<ul style="list-style-type: none"> <li>+ConnectAPDialog(java.awt.Frame parent, boolean modal, String token, String ssid)</li> <li>// &lt;editor-fold defaultstate="collapsed" desc="Generated Code"&gt;//GEN-BEGIN: initComponents void initComponents()</li> <li>- void connectButtonActionPerformed(java.awt.event.ActionEvent evt)</li> <li>- void TimeSliderStateChanged(javax.swing.event.ChangeEvent evt)</li> <li>- void PremiumButtonActionPerformed(java.awt.event.ActionEvent evt)</li> <li>+void connect(String ssid)</li> <li>+void priceCheck(String ssid)</li> <li>- void setBalance()</li> <li>- void upgrade()</li> <li>- void bankRegister()</li> </ul>

**ContatoreFrame:** implementa l'interfaccia necessaria a visualizzare, aggiungere o modificare le informazioni riguardo al proprio contatore di crediti.



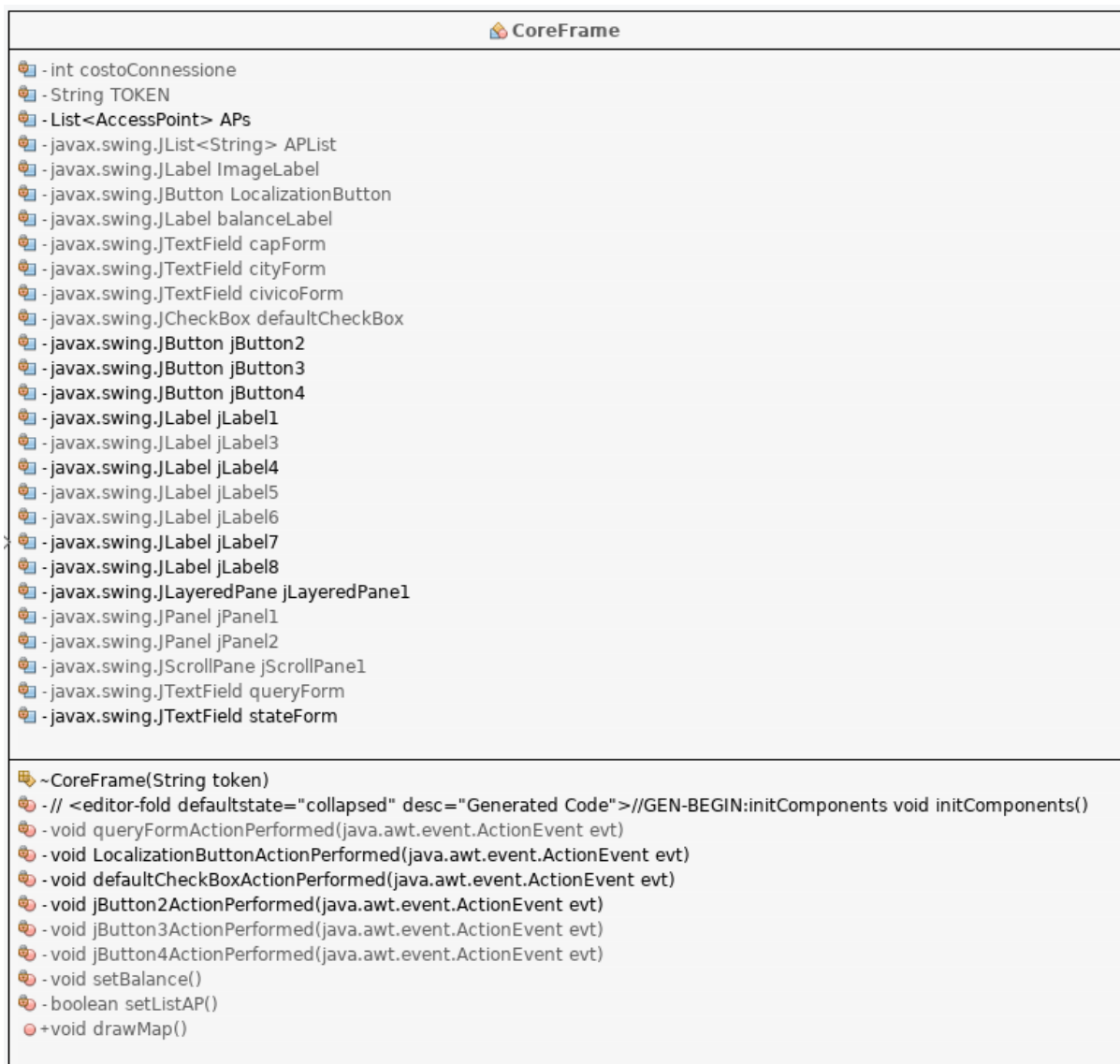
## ContatoreFrame

- String TOKEN
- javax.swing.JTextField ApiField
- javax.swing.JTextField ChannelField
- javax.swing.JTextField FieldField
- javax.swing.JLabel balanceLabel
- javax.swing.JButton jButton1
- javax.swing.JButton jButton2
- javax.swing.JButton jButton3
- javax.swing.JLabel jLabel1
- javax.swing.JLabel jLabel3
- javax.swing.JLabel jLabel4
- javax.swing.JLabel jLabel5
- javax.swing.JLabel jLabel6
- javax.swing.JPanel jPanel1
- javax.swing.JToggleButton jButton1

### +ContatoreFrame(String token)

- // <editor-fold defaultstate="collapsed" desc="Generated Code"> //GEN-BEGIN: initComponents void initComponents()
- void jButton2ActionPerformed(java.awt.event.ActionEvent evt)
- void jButton3ActionPerformed(java.awt.event.ActionEvent evt)
- void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
- +void setInfoCounter()
- void setBalance()
- +boolean updateCounter()

**CoreFrame:** è la prima interfaccia visualizzata dopo aver fatto il login o la registrazione. Visualizza la mappa con centro nelle coordinate richieste.




The screenshot displays the 'CoreFrame' class in an IDE. The top section shows the class's attributes and components, including a list of AccessPoints, a list of APs, and various Swing components like labels, buttons, text fields, and a checkbox. The bottom section shows the class's methods, including a constructor and several event listener methods.

```
CoreFrame
- int costoConnessione
- String TOKEN
- List<AccessPoint> APs
- javax.swing.JList<String> APList
- javax.swing.JLabel ImageLabel
- javax.swing.JButton LocalizationButton
- javax.swing.JLabel balanceLabel
- javax.swing.JTextField capForm
- javax.swing.JTextField cityForm
- javax.swing.JTextField civicoForm
- javax.swing.JCheckBox defaultCheckBox
- javax.swing.JButton jButton2
- javax.swing.JButton jButton3
- javax.swing.JButton jButton4
- javax.swing.JLabel jLabel1
- javax.swing.JLabel jLabel3
- javax.swing.JLabel jLabel4
- javax.swing.JLabel jLabel5
- javax.swing.JLabel jLabel6
- javax.swing.JLabel jLabel7
- javax.swing.JLabel jLabel8
- javax.swing.JLayeredPane jLayeredPanel1
- javax.swing.JPanel jPanel1
- javax.swing.JPanel jPanel2
- javax.swing.JScrollPane jScrollPane1
- javax.swing.JTextField queryForm
- javax.swing.JTextField stateForm

~CoreFrame(String token)
- // <editor-fold defaultstate="collapsed" desc="Generated Code"> //GEN-BEGIN: initComponents void initComponents()
- void queryFormActionPerformed(java.awt.event.ActionEvent evt)
- void LocalizationButtonActionPerformed(java.awt.event.ActionEvent evt)
- void defaultCheckBoxActionPerformed(java.awt.event.ActionEvent evt)
- void jButton2ActionPerformed(java.awt.event.ActionEvent evt)
- void jButton3ActionPerformed(java.awt.event.ActionEvent evt)
- void jButton4ActionPerformed(java.awt.event.ActionEvent evt)
- void setBalance()
- boolean setListAP()
+ void drawMap()
```

**noThingFormLogin:** implementa il form per il login

 noThingFormLogin
<ul style="list-style-type: none"><li>- javax.swing.JButton LoginButton</li><li>- javax.swing.JLabel TokenLabel</li><li>- javax.swing.JLabel jLabel1</li><li>- javax.swing.JLabel jLabel2</li><li>- javax.swing.JPasswordField passwordField</li><li>- javax.swing.JButton toSignInButton</li><li>- javax.swing.JTextField usernameField</li></ul>
<ul style="list-style-type: none"><li>+noThingFormLogin()</li><li>- // &lt;editor-fold defaultstate="collapsed" desc="Generated Code"&gt;//GEN-BEGIN: initComponents void initComponents()</li><li>- void LoginButtonActionPerformed(java.awt.event.ActionEvent evt)</li><li>- void usernameFieldActionPerformed(java.awt.event.ActionEvent evt)</li><li>- void toSignInButtonActionPerformed(java.awt.event.ActionEvent evt)</li><li>+static void main(String args)</li></ul>

**noThingFormSignin:** implementa il form per la registrazione

 noThingFormSignin
<ul style="list-style-type: none"><li>- javax.swing.JButton SigninButton</li><li>- javax.swing.JLabel TokenLabel</li><li>- javax.swing.JLabel jLabel1</li><li>- javax.swing.JLabel jLabel2</li><li>- javax.swing.JPasswordField passwordField</li><li>- javax.swing.JButton toLoginButton</li><li>- javax.swing.JTextField usernameField</li></ul>
<ul style="list-style-type: none"><li>+noThingFormSignin()</li><li>- // &lt;editor-fold defaultstate="collapsed" desc="Generated Code"&gt;//GEN-BEGIN: initComponents void initComponents()</li><li>- void SigninButtonActionPerformed(java.awt.event.ActionEvent evt)</li><li>- void usernameFieldActionPerformed(java.awt.event.ActionEvent evt)</li><li>- void toLoginButtonActionPerformed(java.awt.event.ActionEvent evt)</li></ul>

## Server

**ServerRequest** contiene tutti i metodi utili ad effettuare delle richieste al server principale e alla banca. Per ogni path dei server utile al client esiste un metodo.

ServerRequest
- final String SERVER
<ul style="list-style-type: none"><li> - static String GET(String token, String urlString)</li><li> - static String PUT(String token, String urlString, String msg)</li><li> - static String POST(String token, String urlString, String msg)</li><li> - static String DELETE(String token, String urlString, String msg)</li><li> - static String DELETE(String token, String urlString)</li><li> +static int getBalance(String token)</li><li> +static List&lt;AccessPoint&gt; getListAp(String token, float latitude, float longitude)</li><li> +static List&lt;AccessPoint&gt; getListAp(String token, String query, String civico, String cap, String city, String state)</li><li> +static List&lt;AccessPoint&gt; getListAP(String token)</li><li> +static BufferedImage getMap(String token, float latitude, float longitude)</li><li> +static BufferedImage getMapQuery(String token, String query, String civico, String cap, String city, String state)</li><li> +static String Connect(String token, String ssid, int time)</li><li> +static boolean addAP(String token, String ssid, float latitude, float longitude)</li><li> +static boolean addAP(String token, String ssid, String query, String civico, String cap, String city, String state)</li><li> +static boolean removeAP(String token, String ssid)</li><li> +static Counter getInfoCounter(String token)</li><li> +static boolean updateCounter(String token, String API, int channel, int field)</li><li> +static int getPrice(String token, String ssid, int value)</li><li> +static String getCode(String token)</li><li> +static int payBank(String token)</li><li> +static Boolean upgrade(String token, int bank_id)</li><li> +static Boolean isPremium(String token)</li><li> +static Boolean RegisterCard(String token, int code, int cvc, int month, int year)</li></ul>

**infoServer:** implementa i metodi utili a raccogliere informazioni sugli indirizzi del server dai file di configurazione.

infoServer
- final String xmlPath
<ul style="list-style-type: none"><li> +static String getServerURL()</li><li> +static String getServerTokenURL()</li><li> +static String getBankURL()</li></ul>

## TokenServer

**Token:** raccoglie i metodi per effettuare il login e registrazione sul server token

Token
<ul style="list-style-type: none"><li>-final String SERVER</li><li>-final String SERVETOKEN</li></ul>
<ul style="list-style-type: none"><li>+static String login(String username, String password)</li><li>+static String signin(String username, String password)</li><li>-static String decrypt(String msg)</li><li>-static String encryptToken(String msg)</li></ul>

## Request

**HTTP:** contiene i metodi per effettuare delle richieste http.

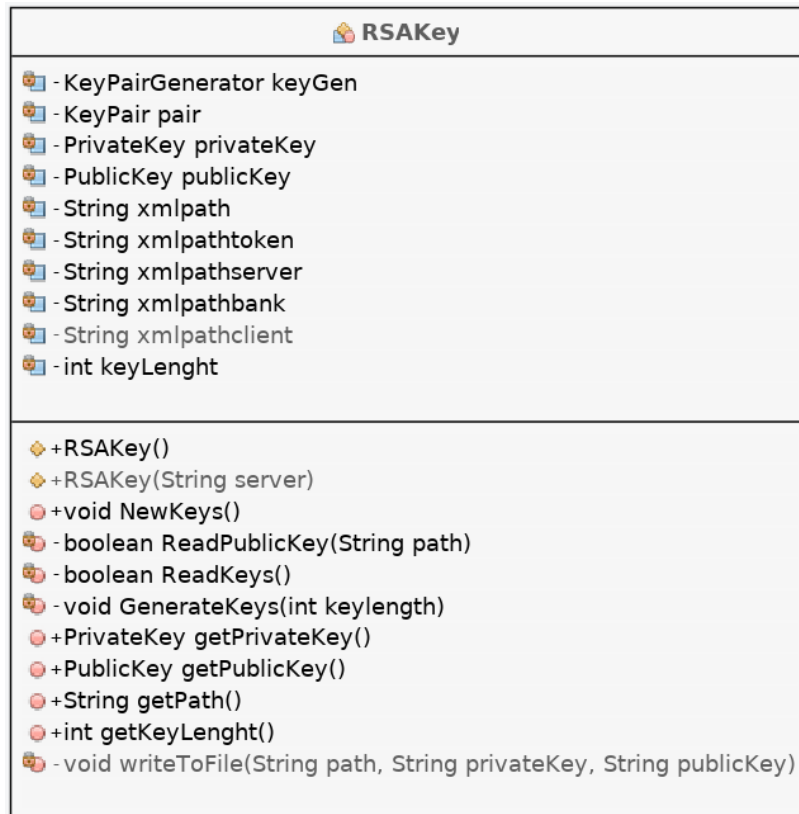
HTTP
<ul style="list-style-type: none"><li>+static String GET(String urlString, String msg)</li><li>+static String GET(String urlString)</li><li>+static String PUT(String urlString, String msg)</li><li>+static String PUT(String urlString)</li><li>+static String POST(String urlString, String msg)</li><li>+static String POST(String urlString)</li><li>+static String DELETE(String urlString, String msg)</li><li>+static String DELETE(String urlString)</li><li>+static String HEAD(String urlString, String msg)</li><li>+static String HEAD(String urlString)</li><li>+static String OPTIONS(String urlString, String msg)</li><li>+static String OPTIONS(String urlString)</li><li>+static String PATCH(String urlString, String msg)</li><li>+static String PATCH(String urlString)</li><li>-static String request(String urlString, String method, String msg)</li><li>-static String request(String urlString, String method)</li></ul>

## crittography

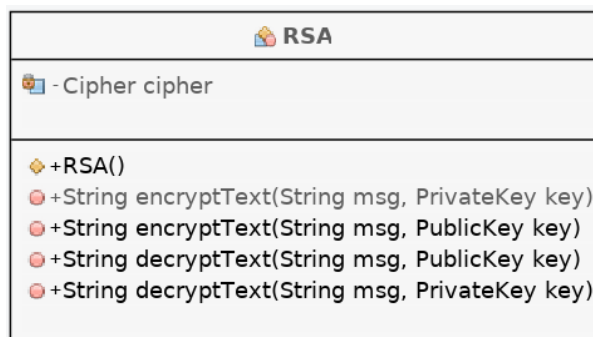
le classi in questo package servono per ciò che riguarda la crittografia.

**RSAKey:** contiene tutti i metodi per leggere dai file di configurazione le chiavi.

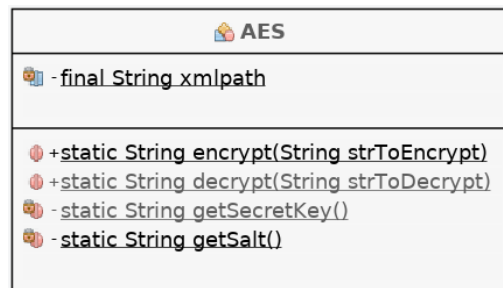
- RSAKey() inizializza le variabili private con le proprie chiavi e se non trova il proprio file di chiavi rsa ne genera di nuove
- RSAKey(String server) inizializza le variabili private con le chiavi relative al progetto richiesto in argomento (client, bank, server, token).



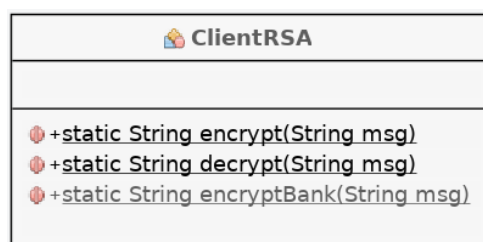
**RSA:** contiene i metodi utili a cifrare e decifrare un messaggio con una chiave data.



**AES:** contiene i metodi per crittografare i messaggi con AES

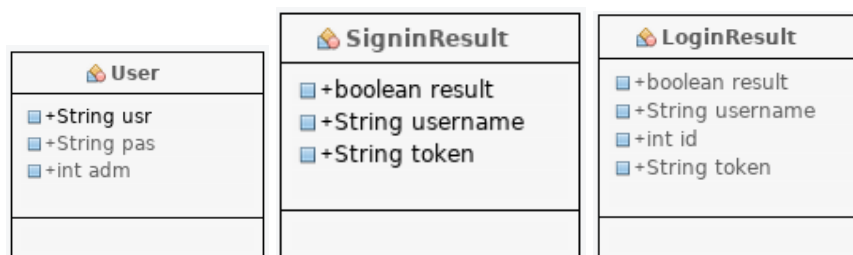


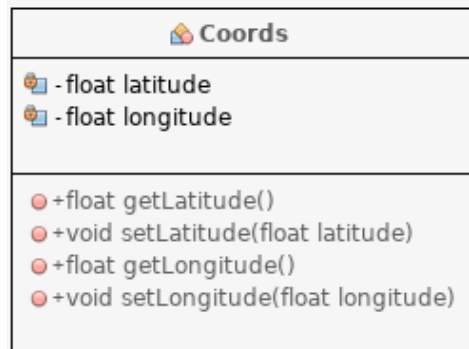
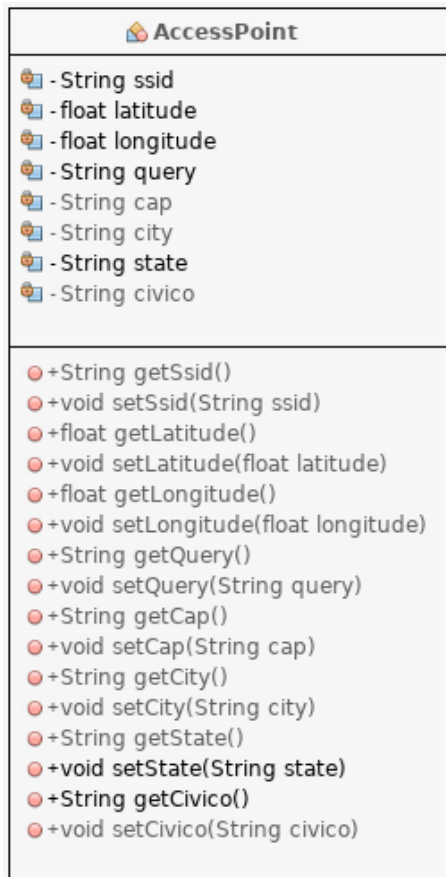
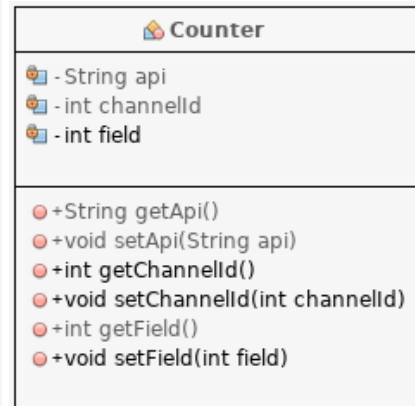
**ClientRSA:** contiene i metodi per crittografare con RSA utilizzando la chiave pubblica della banca, del server principale e quello per decifrare con la propria chiave privata.



## POJO

Queste Classi sono utilizzate solamente per rappresentare i dati inviati e/o ricevuti in formato json. Per definizione non sono legati ad alcuna restrizione diversa da quelle costrette dalla specifica del linguaggio Java.







# Client Log

## GUI

**BubbleFrame:** implementa l'interfaccia per la visualizzazione del grafico a bolle.


```
BubbleFrame
- List<Users> users
- String TOKEN
- javax.swing.JToggleButton LogButton
- javax.swing.JButton LoginButton
- javax.swing.JButton PaymentButton
- javax.swing.JButton SigninButton
- javax.swing.JPanel chartPanel
- javax.swing.JButton jButton1
- javax.swing.JLayeredPane jLayeredPane2

+BubbleFrame(String token)
// <editor-fold defaultstate="collapsed" desc="Generated Code"> //GEN-BEGIN: initComponents void initComponents()
- void PaymentButtonActionPerformed(java.awt.event.ActionEvent evt)
- void LoginButtonActionPerformed(java.awt.event.ActionEvent evt)
- void SigninButtonActionPerformed(java.awt.event.ActionEvent evt)
- void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
- void LogButtonActionPerformed(java.awt.event.ActionEvent evt)
- XYZDataset createDataset()
- JFreeChart createChart(XYZDataset xyzdataset)
+JPanel createChartPanel()
+void drawChart()
```

**LogFrame:** implementa l'interfaccia per la visualizzazione dei log delle connessioni degli utenti agli Access Point.


 **LogFrame**

---

-  - String TOKEN
-  - javax.swing.JTextField FromField
-  - javax.swing.JButton LogDateButton
-  - javax.swing.JTable LogTable
-  - javax.swing.JButton LoginButton
-  - javax.swing.JButton PaymentButton
-  - javax.swing.JButton SigninButton
-  - javax.swing.JTextField ToField
-  - javax.swing.JCheckBox dateCheckBox
-  - javax.swing.JButton jButton1
-  - javax.swing.JLabel jLabel1
-  - javax.swing.JLabel jLabel2
-  - javax.swing.JLabel jLabel3
-  - javax.swing.JLabel jLabel4
-  - javax.swing.JLayeredPane jLayeredPane1
-  - javax.swing.JLayeredPane jLayeredPane2
-  - javax.swing.JScrollPane jScrollPane2
-  - javax.swing.JScrollPane jScrollPane3
-  - javax.swing.JTable jTable1
-  - javax.swing.JToggleButton jButton1
-  - javax.swing.JCheckBox userCheckBox
-  - javax.swing.JTextField userIdField

---

 +LogFrame(String token)

-  - // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN: initComponents void initComponents()
-  - void LogDateButtonActionPerformed(java.awt.event.ActionEvent evt)
-  - void LoginButtonActionPerformed(java.awt.event.ActionEvent evt)
-  - void SigninButtonActionPerformed(java.awt.event.ActionEvent evt)
-  - void PaymentButtonActionPerformed(java.awt.event.ActionEvent evt)
-  - void dateCheckBoxActionPerformed(java.awt.event.ActionEvent evt)
-  - void userIdFieldActionPerformed(java.awt.event.ActionEvent evt)
-  - void userCheckBoxActionPerformed(java.awt.event.ActionEvent evt)
-  - void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
-  - void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
-  - void SetListLog()

**LogPayFrame:** implementa l'interfaccia grafica per la visualizzazione dei log dei pagamenti degli utenti per eseguire l'upgrade a premium.

 **LogPayFrame**

-  - String TOKEN
-  - javax.swing.JButton ConnectionButton
-  - javax.swing.JTextField FromField
-  - javax.swing.JButton LogDateButton
-  - javax.swing.JTable LogTable
-  - javax.swing.JButton LoginButton
-  - javax.swing.JButton SigninButton
-  - javax.swing.JTextField ToField
-  - javax.swing.JCheckBox dateCheckBox
-  - javax.swing.JButton jButton1
-  - javax.swing.JLabel jLabel1
-  - javax.swing.JLabel jLabel2
-  - javax.swing.JLabel jLabel4
-  - javax.swing.JLayeredPane jLayeredPane1
-  - javax.swing.JLayeredPane jLayeredPane2
-  - javax.swing.JScrollPane jScrollPane2
-  - javax.swing.JScrollPane jScrollPane3
-  - javax.swing.JTable jTable1
-  - javax.swing.JToggleButton jToggleButton1
-  - javax.swing.JCheckBox userCheckBox
-  - javax.swing.JTextField userIdField

---

 +LogPayFrame(String token)

-  // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN: initComponents void initComponents()
-  - void LogDateButtonActionPerformed(java.awt.event.ActionEvent evt)
-  - void LoginButtonActionPerformed(java.awt.event.ActionEvent evt)
-  - void SigninButtonActionPerformed(java.awt.event.ActionEvent evt)
-  - void ConnectionButtonActionPerformed(java.awt.event.ActionEvent evt)
-  - void dateCheckBoxActionPerformed(java.awt.event.ActionEvent evt)
-  - void userIdFieldActionPerformed(java.awt.event.ActionEvent evt)
-  - void userCheckBoxActionPerformed(java.awt.event.ActionEvent evt)
-  - void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
-  - void jToggleButton1ActionPerformed(java.awt.event.ActionEvent evt)
-  - void SetListLog()

**PieFrame:** implementa l'interfaccia grafica per la visualizzazione del grafico a torta

PieFrame	
<ul style="list-style-type: none"> <li>- List&lt;Users&gt; users</li> <li>- String TOKEN</li> <li>- javax.swing.JToggleButton LogButton</li> <li>- javax.swing.JButton LoginButton</li> <li>- javax.swing.JButton PaymentButton</li> <li>- javax.swing.JButton SigninButton</li> <li>- javax.swing.JPanel chartPanel</li> <li>- javax.swing.JLayeredPane jLayeredPane2</li> <li>- javax.swing.JToggleButton jToggleButton1</li> <li>- javax.swing.JComboBox&lt;String&gt; usersComboBox</li> </ul>	
<ul style="list-style-type: none"> <li>+PieFrame(String token)</li> <li>- // &lt;editor-fold defaultstate="collapsed" desc="Generated Code"&gt; //GEN-BEGIN: initComponents void initComponents()</li> <li>- void usersComboBoxActionPerformed(java.awt.event.ActionEvent evt)</li> <li>- void PaymentButtonActionPerformed(java.awt.event.ActionEvent evt)</li> <li>- void LoginButtonActionPerformed(java.awt.event.ActionEvent evt)</li> <li>- void SigninButtonActionPerformed(java.awt.event.ActionEvent evt)</li> <li>- void LogButtonActionPerformed(java.awt.event.ActionEvent evt)</li> <li>- PieDataset createDataset(List&lt;Log&gt; logs)</li> <li>- int contain(List&lt;Log&gt; log, String ssid)</li> <li>- JFreeChart createChart(PieDataset dataset, String username)</li> <li>+ JPanel createChartPanel(List&lt;Log&gt; log, String username)</li> <li>+ void setUsers()</li> <li>+ void drawChart()</li> </ul>	

**NoThingsLoginForm:** mplementa il form per il login.

noThingFormLogin	
<ul style="list-style-type: none"> <li>- javax.swing.JButton LoginButton</li> <li>- javax.swing.JLabel TokenLabel</li> <li>- javax.swing.JLabel jLabel1</li> <li>- javax.swing.JLabel jLabel2</li> <li>- javax.swing.JLabel jLabel3</li> <li>- javax.swing.JPasswordField passwordField</li> <li>- javax.swing.JButton toSigninButton</li> <li>- javax.swing.JTextField usernameField</li> </ul>	
<ul style="list-style-type: none"> <li>+noThingFormLogin()</li> <li>- // &lt;editor-fold defaultstate="collapsed" desc="Generated Code"&gt; //GEN-BEGIN: initComponents void initComponents()</li> <li>- void LoginButtonActionPerformed(java.awt.event.ActionEvent evt)</li> <li>- void usernameFieldActionPerformed(java.awt.event.ActionEvent evt)</li> <li>- void toSigninButtonActionPerformed(java.awt.event.ActionEvent evt)</li> <li>+static void main(String args)</li> </ul>	

**NoThingsSigninFrame:** implementa il form per la registrazione.

noThingFormSignin
<ul style="list-style-type: none"> <li>- javax.swing.JButton SigninButton</li> <li>- javax.swing.JLabel TokenLabel</li> <li>- javax.swing.JLabel jLabel1</li> <li>- javax.swing.JLabel jLabel2</li> <li>- javax.swing.JLabel jLabel3</li> <li>- javax.swing.JPasswordField passwordField</li> <li>- javax.swing.JButton toLoginButton</li> <li>- javax.swing.JTextField usernameField</li> </ul>
<ul style="list-style-type: none"> <li>◆ +noThingFormSignin()</li> <li>🔒 // &lt;editor-fold defaultstate="collapsed" desc="Generated Code"&gt; //GEN-BEGIN: initComponents void initComponents()</li> <li>🔒 - void SigninButtonActionPerformed(java.awt.event.ActionEvent evt)</li> <li>🔒 - void usernameFieldActionPerformed(java.awt.event.ActionEvent evt)</li> <li>🔒 - void toLoginButtonActionPerformed(java.awt.event.ActionEvent evt)</li> </ul>

## Server

**ServerRequest:** contiene tutti i metodi utili ad effettuare delle richieste al server principale e alla banca. Per ogni path dei server utile al client esiste un metodo.

InfoServer

ServerRequest
<ul style="list-style-type: none"> <li>- final String SERVER</li> <li>- final String BANK</li> </ul>
<ul style="list-style-type: none"> <li>🔒 - static String GET(String token, String urlString)</li> <li>🔒 - static String GETBank(String token, String urlString)</li> <li>🔒 - static String PUT(String token, String urlString, String msg)</li> <li>🔒 - static String POST(String token, String urlString, String msg)</li> <li>🔒 - static String DELETE(String token, String urlString, String msg)</li> <li>🔒 - static String DELETE(String token, String urlString)</li> <li>🔒 +static List&lt;Log&gt; getServerLog(String token)</li> <li>🔒 +static List&lt;Log&gt; getServerLog(String token, int user_id)</li> <li>🔒 +static List&lt;Log&gt; getServerLog(String token, String from, String to)</li> <li>🔒 +static List&lt;Log&gt; getServerLog(String token, int user_id, String from, String to)</li> <li>🔒 +static List&lt;BankLog&gt; getBankLog(String token)</li> <li>🔒 +static List&lt;BankLog&gt; getBankLog(String token, int user_id)</li> <li>🔒 +static List&lt;BankLog&gt; getBankLog(String token, String from, String to)</li> <li>🔒 +static List&lt;BankLog&gt; getBankLog(String token, int user_id, String from, String to)</li> <li>🔒 +static List&lt;Users&gt; getUsers(String token)</li> <li>🔒 +static List&lt;APStats&gt; getAPStats(String token)</li> </ul>

**infoServer:** implementa i metodi utili a raccogliere informazioni sugli indirizzi del server dai file di configurazione.

infoServer
- final String xmlPath
+static String getServerURL() +static String getServerTokenURL() +static String getBankURL()

## TokenServer

**Token:** raccoglie i metodi per effettuare il login e registrazione sul server token

Token
- final String SERVER - final String SERVETOKEN
+static String login(String username, String password) +static String signin(String username, String password) - static String decrypt(String msg) - static String encrypt(String msg)

## Request

**HTTP:** contiene i metodi per effettuare delle richieste http.

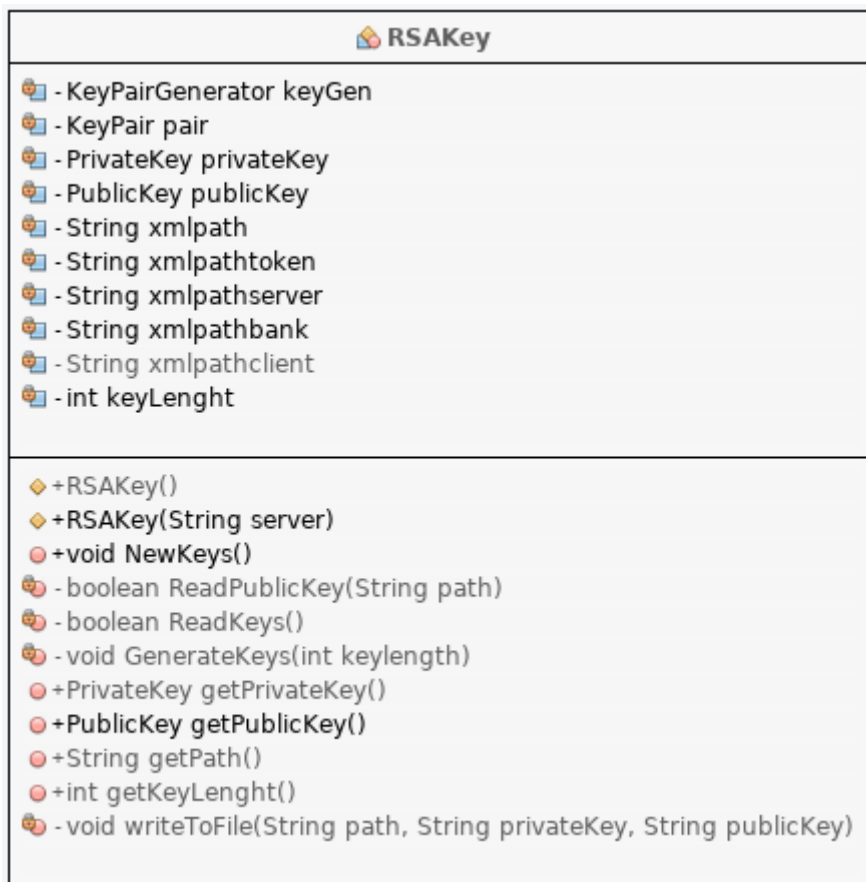
HTTP
<ul style="list-style-type: none"><li>④ +static String GET(String urlString, String msg)</li><li>④ <b>+static String GET(String urlString)</b></li><li>④ +static String PUT(String urlString, String msg)</li><li>④ +static String PUT(String urlString)</li><li>④ +static String POST(String urlString, String msg)</li><li>④ +static String POST(String urlString)</li><li>④ +static String DELETE(String urlString, String msg)</li><li>④ +static String DELETE(String urlString)</li><li>④ +static String HEAD(String urlString, String msg)</li><li>④ +static String HEAD(String urlString)</li><li>④ +static String OPTIONS(String urlString, String msg)</li><li>④ +static String OPTIONS(String urlString)</li><li>④ +static String PATCH(String urlString, String msg)</li><li>④ +static String PATCH(String urlString)</li><li>④ - static String request(String urlString, String method, String msg)</li><li>④ - static String request(String urlString, String method)</li></ul>

## crittography

le classi in questo package servono per ciò che riguarda la crittografia.

**RSAKey:** contiene tutti i metodi per leggere dai file di configurazione le chiavi.

- RSAKey() inizializza le variabili private con le proprie chiavi e se non trova il proprio file di chiavi rsa ne genera di nuove
- RSAKey(String server) inizializza le variabili private con le chiavi relative al progetto richiesto in argomento (client, bank, server, token).





**RSA:** contiene i metodi utili a cifrare e decifrare un messaggio con una chiave data.

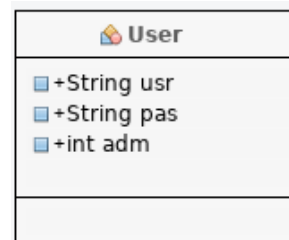
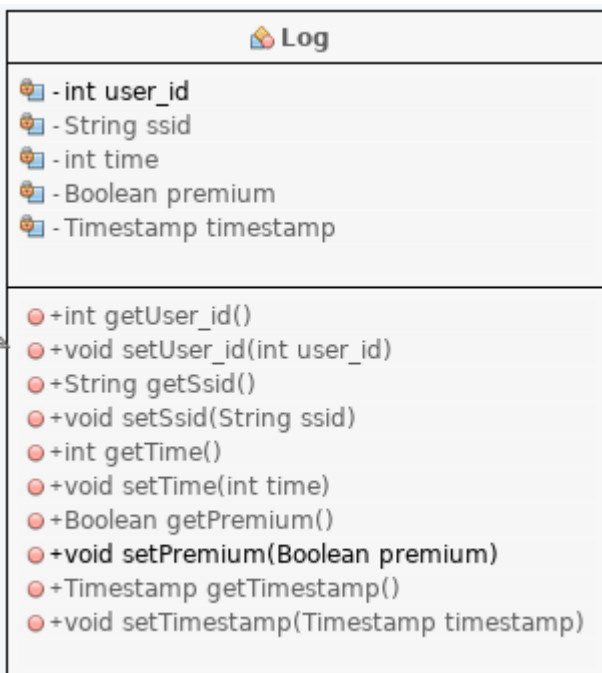
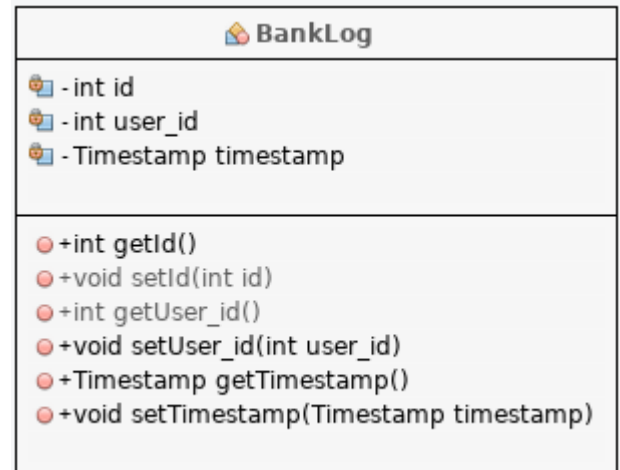
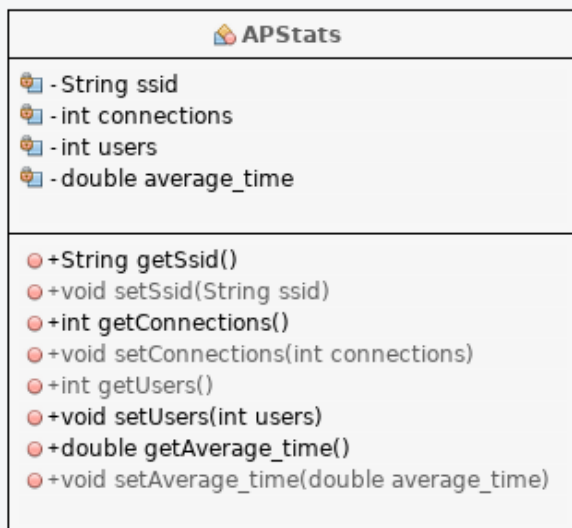
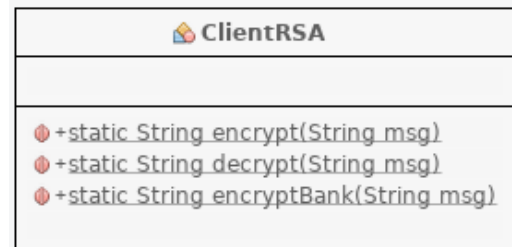
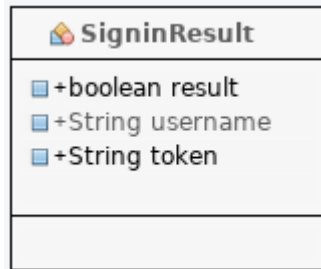
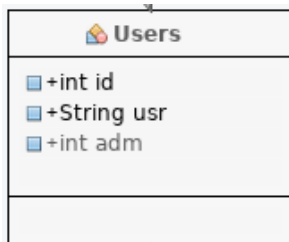
RSA
- Cipher cipher
<ul style="list-style-type: none"><li>◆ +RSA()</li><li>● +String encryptText(String msg, PrivateKey key)</li><li>● +String encryptText(String msg, PublicKey key)</li><li>● +String decryptText(String msg, PublicKey key)</li><li>● +String decryptText(String msg, PrivateKey key)</li></ul>

**AES:** contiene i metodi per crittografare i messaggi con AES

AES
- final String xmlpath
<ul style="list-style-type: none"><li>● +static String encrypt(String strToEncrypt)</li><li>● +static String decrypt(String strToDecrypt)</li><li>● - static String getSecretKey()</li><li>● - static String getSalt()</li></ul>

## POJO

Queste Classi sono utilizzate solamente per rappresentare i dati inviati e/o ricevuti in formato json. Per definizione non sono legati ad alcuna restrizione diversa da quelle costrette dalla specifica del linguaggio Java.



# Database

Il progetto prevede l'utilizzo di un database relazionale mysql per ogni server.

## bank

bank è il database utilizzato dal server banca.

transaction	
id	int
user_id	int
timestamp	timestamp

account	
id	int
username	varchar
code	int
month	int
year	int
cvc	int

## core

core è il database utilizzato dal server principale

accessPoint	
ssid	varchar
user_id	int
latitude	float
longitude	float

log	
id	int
user_id	int
premium	tinyint
ssid	varchar
time	int
timestamp	timestamp

log_credits	
id	int
user_id	int
amount	int
timestamp	timestamp

account	
id	int
username	varchar
admin	tinyint
startPremium	date
endPremium	date
credits	int
last_token	varchar
expiration_token	datetime

device	
user_id	int
API	varchar
ChannelID	int
field	int

# nothings

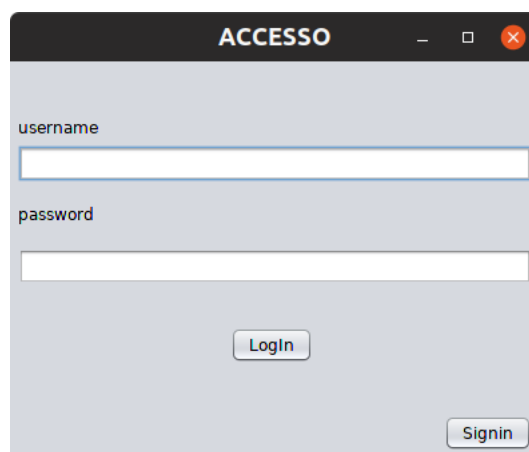
nothings è il database utilizzato dal server token

users	
id	int
username	varchar
password	varchar
admin	date
admin	tinyint

## GUI

### Client principale

#### Login e Registrazione



The screenshot shows a window titled "ACCESSO" with a dark header bar. Below the header, there are two input fields: "username" and "password". The "username" field is currently selected. Below the "password" field, there are two buttons: "Login" and "Signin".

Figure 9: Schermata di login



Figure 10: Schermata di registrazione

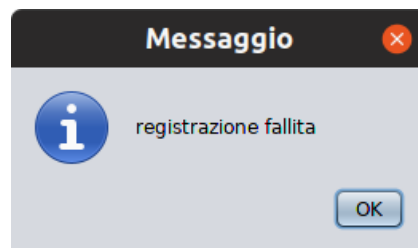


Figure 11: Messaggio di registrazione fallita

# Mappa

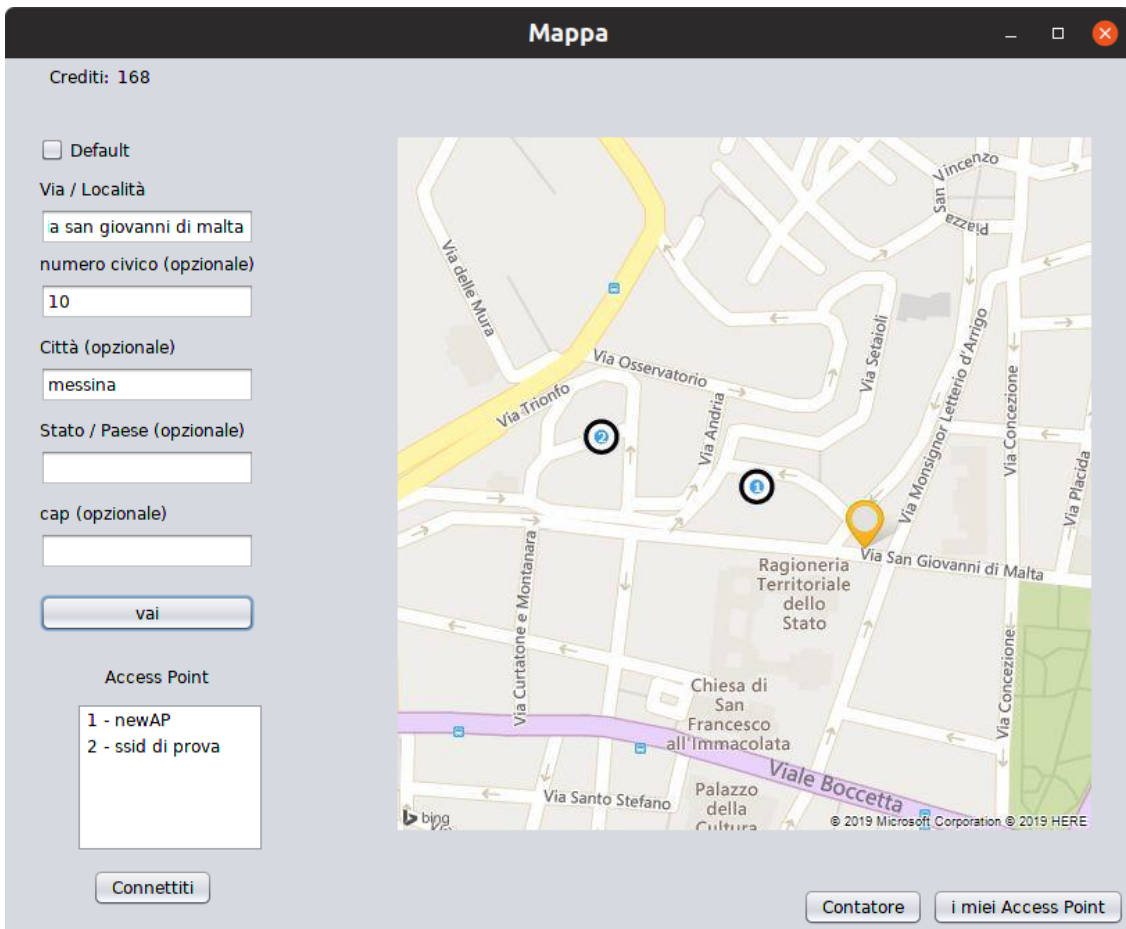


Figure 12: Schermata con la visualizzazione della mappa e degli AP



## Connessione Access Point

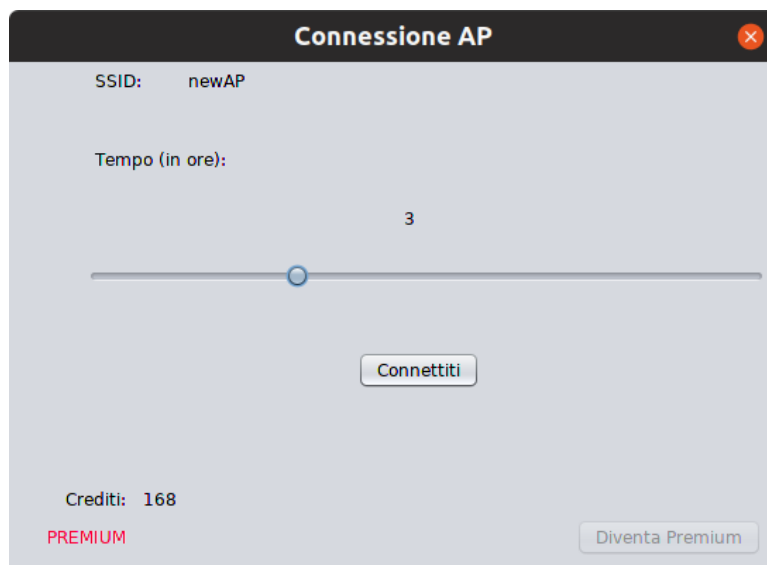


Figure 13: Schermata per la richiesta di connessione ad un AP

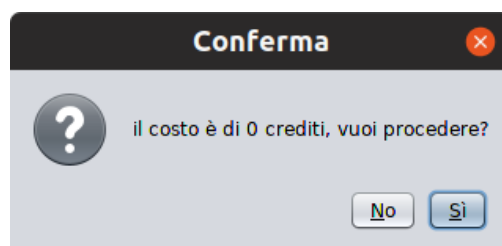


Figure 14: Messaggio di conferma connessione ad un AP

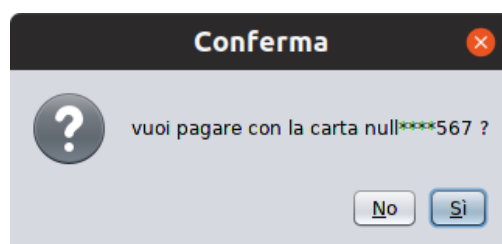


Figure 15: Messaggio di conferma pagamento

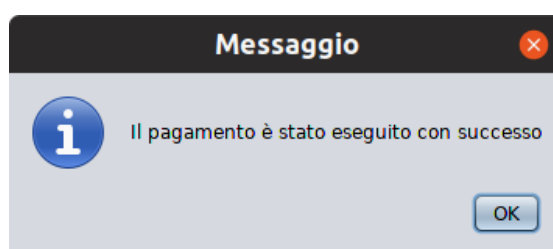


Figure 16: Messaggio di pagamento eseguito con successo

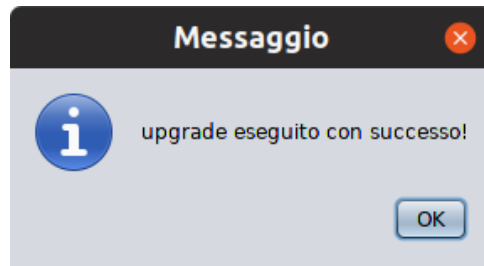


Figure 17: Messaggio di upgrade eseguito con successo

## Gestione Contatore

A screenshot of a mobile application screen titled "Contatore". The screen displays the following information and controls:

- Saldo : 180
- API:
- channel ID:
- Field:
- aggiorna button
- Mappa button
- I miei AP button

The screen has a dark header bar with the title "Contatore" and standard mobile window controls (minimize, maximize, close).

Figure 18: Schermata per la gestione del dispositivo contatore

## Gestione Access Point



The screenshot shows a web application window titled "I tuoi Access Point". Inside, there is a section titled "Access Point" containing a list of three items: "1 - newAP", "2 - ssid di prova", and "3 - vodafone". Below this list are several input fields for adding a new AP: "SSID", "Via / Località", "numero civico (opzionale)", "Città (opzionale)", "Stato / Paese (opzionale)", and "cap (opzionale)". At the bottom of the form are two buttons: "Aggiungi" and "Rimuovi". In the bottom right corner, there are two more buttons: "Mappa" and "Contatore".

Figure 19: Schermata per la gestione dei propri AP

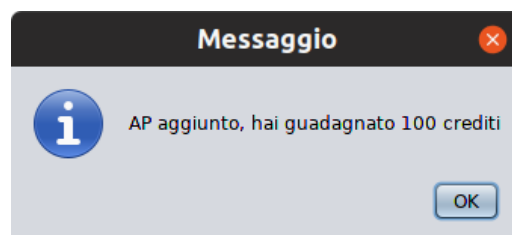


Figure 20: Messaggio successo aggiunta AP

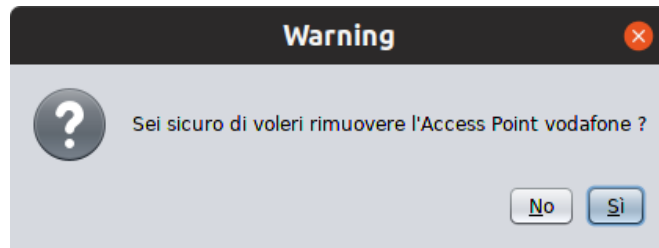


Figure 21: Messaggio di avviso

## Registrazione carta di credito

A registration form titled "Registrazione carta" with a red close button. The form has a light gray background. It contains a text input field for "Codice carta". Below this, there are two sections: "CVC" with a text input field, and "Scadenza" with two dropdown menus for "mese" (set to 1) and "anno" (set to 2019). At the bottom center, there is a button labeled "Registra Carta".

Figure 22: Schermata per la registrazione della carta di credito

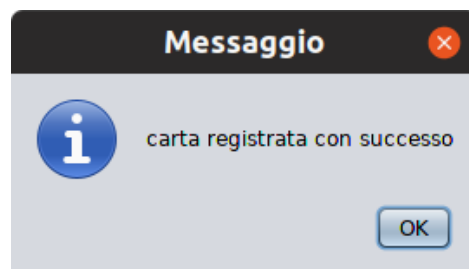
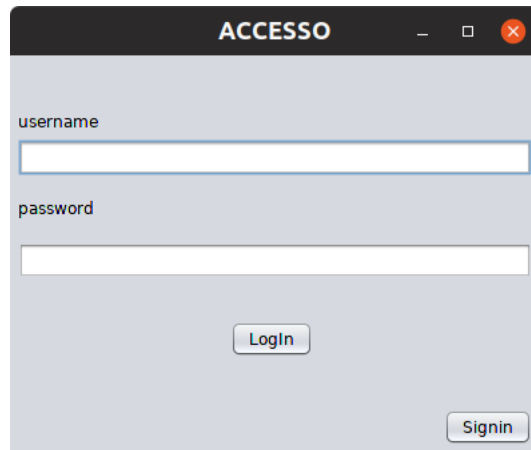


Figure 23: Messaggio di successo della registrazione della carta

# Client log

## Login e Registrazione



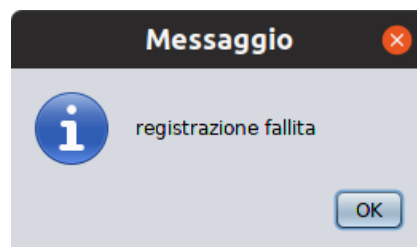
The screenshot shows a window titled "ACCESSO" with a dark header bar. Below the header, there are two input fields: "username" and "password". The "username" field is currently empty. Below the "password" field, there are two buttons: "LogIn" and "Signin". The "LogIn" button is positioned to the left of the "Signin" button.

Figure 24: Schermata di login



The screenshot shows a window titled "REGISTRAZIONE" with a dark header bar. Below the header, there are two input fields: "username" and "password". The "username" field is currently empty. Below the "password" field, there are two buttons: "Signin" and "LogIn". The "Signin" button is positioned to the left of the "LogIn" button.

Figure 25: Schermata di registrazione



The screenshot shows a dialog box titled "Messaggio" with a dark header bar. On the left side, there is a blue circular icon with a white lowercase letter 'i'. To the right of the icon, the text "registrazione fallita" is displayed. At the bottom right of the dialog, there is a button labeled "OK".

Figure 26: Messaggio di registrazione fallita

## Log Connessioni

The screenshot shows a web application window titled "Log Connessioni". On the left, there are filters: "id utente:" with a dropdown set to "0", a checked checkbox "tutti gli utenti", "da:" with a date field "18/10/2019", "a:" with a date field "19/10/2019", and a checked checkbox "tutte le date". Below these is an "aggiorna log" button. The main area contains a table with the following data:

id utente	ssid	tempo (ore)	premium	timestamp
45	vodafone	5	<input type="checkbox"/>	2019-09-16 13:07:24.0
45	vodafone	9	<input type="checkbox"/>	2019-09-16 13:07:30.0
45	prova2	3	<input type="checkbox"/>	2019-10-11 14:26:21.0
45	prova2	5	<input type="checkbox"/>	2019-10-11 14:26:35.0
45	ssid di prova	3	<input type="checkbox"/>	2019-10-11 15:02:51.0
45	ssid di prova	1	<input type="checkbox"/>	2019-10-11 15:08:11.0
45	prova2	3	<input type="checkbox"/>	2019-10-11 15:14:02.0
45	prova2	1	<input checked="" type="checkbox"/>	2019-10-11 18:34:07.0
48	ssid di prova	10	<input checked="" type="checkbox"/>	2019-10-11 18:49:12.0
53	prova2	7	<input checked="" type="checkbox"/>	2019-10-11 19:09:07.0
45	ssid di prova	3	<input checked="" type="checkbox"/>	2019-10-18 14:47:23.0
45	ssid di prova	4	<input checked="" type="checkbox"/>	2019-10-18 15:12:08.0
45	newAP	3	<input checked="" type="checkbox"/>	2019-10-18 16:56:42.0

At the bottom of the window are buttons for "Grafico a bolle", "Grafico a torta", "Log Pagamenti", "LogIn", and "SignIn".

Figure 27: Schermata che mostra i log delle avvenute connessioni agli AP

## Log Pagamenti

The screenshot shows a web application window titled "Log pagamenti upgrade". On the left, there are filters: "id utente:" with a dropdown set to "0", a checked checkbox "tutti gli utenti", "da:" with a date field "18/10/2019", "a:" with a date field "19/10/2019", and a checked checkbox "tutte le date". Below these is an "aggiorna log" button. The main area contains a table with the following data:

id transazione	id utente	timestamp
19	48	2019-10-11 18:48:40.0
20	51	2019-10-11 18:59:18.0
21	53	2019-10-11 19:08:48.0
22	58	2019-10-18 16:59:11.0

At the bottom of the window are buttons for "Grafico a bolle", "Grafico a torta", "Log Connessioni", "LogIn", and "SignIn".

Figure 28: Schermata che mostra i log dei pagamenti degli utenti per effettuare l'upgrade a premium

## Grafico a torta

il grafico a torta rappresenta per ogni utente (selezionabile dal menù a tendina in alto) il tempo totale di connessione richiesto per ogni access point.

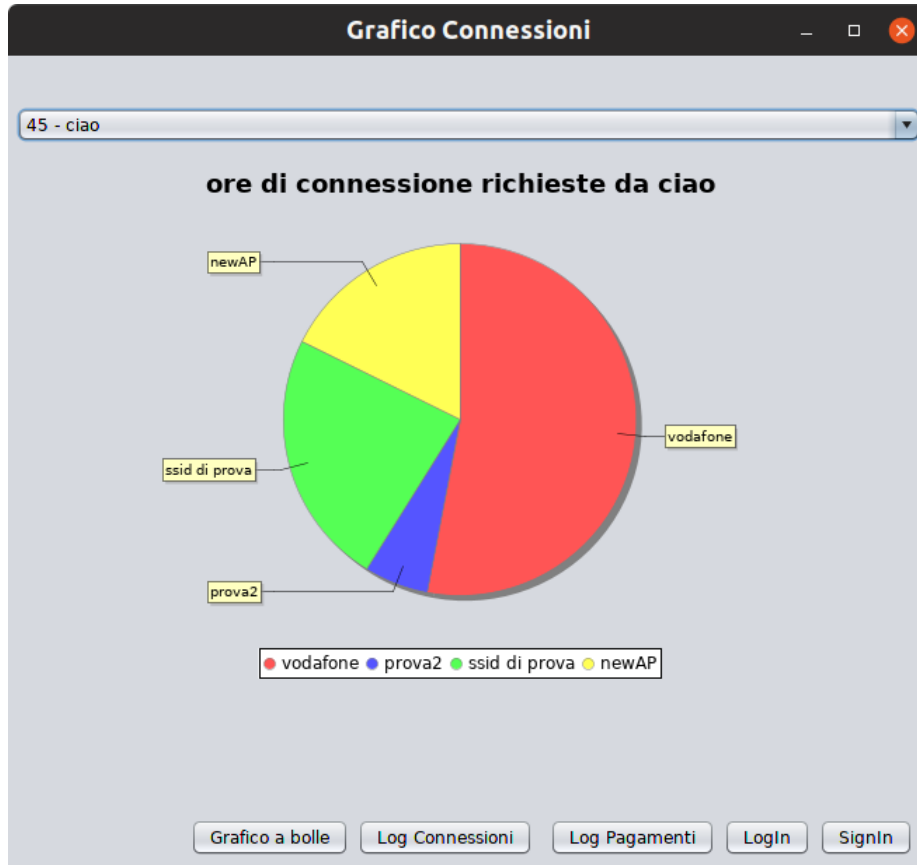


Figure 29: Schermata con grafico a torta con la rappresentazione delle ore di connessione richieste da un utente

## Grafico a bolle

il grafico a bolle rappresenta le connessioni ali Access Point.

Ogni bolla rappresente un Access Point. Sul' asse X è indicato il numero di utenti tra loro differenti che hanno effettuato almeno una volta il collegamento all'access point.

Sull'asse Y è indicato il numero di connessioni totali effettuate all'access point.

La grandezza delle bolle indica il tempo medio di connessione richiesto.

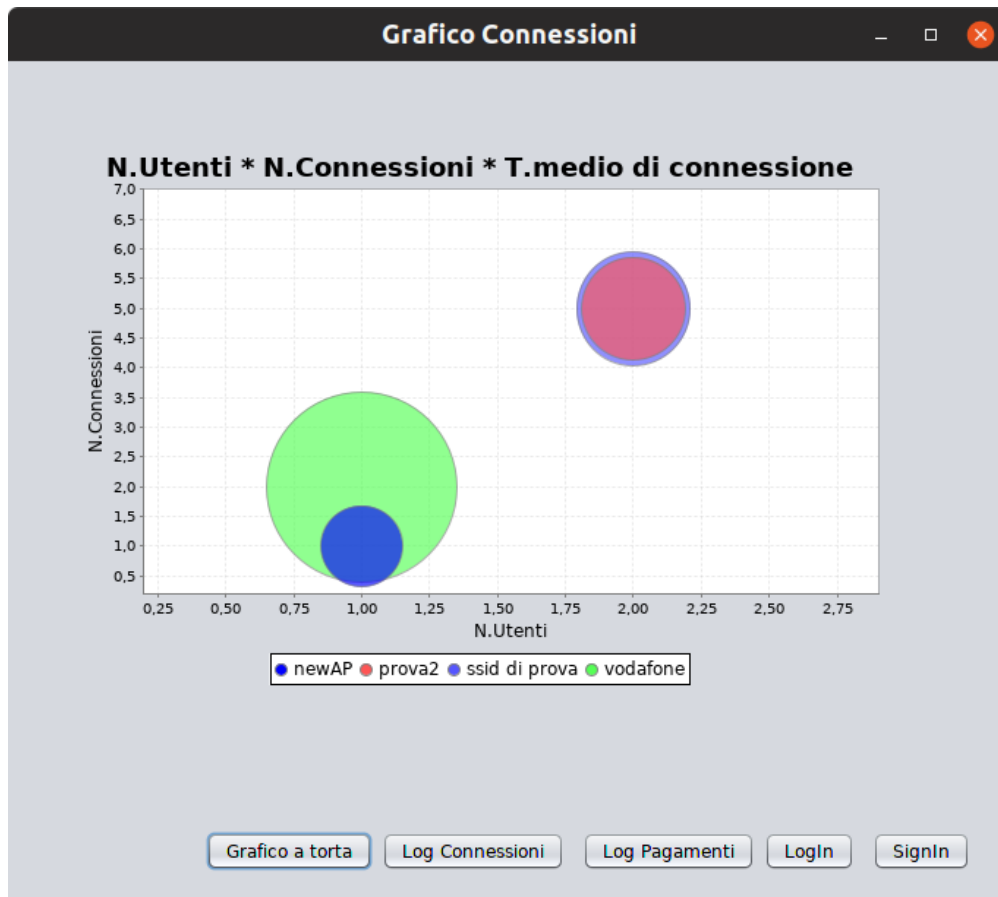


Figure 30: Schermata con il grafico a bolle che rappresenta l'utilizzo degli AccessPoint