UNIVERSITA' DEGLI STUDI DI MESSINA

Progetto Programmazione II

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

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.

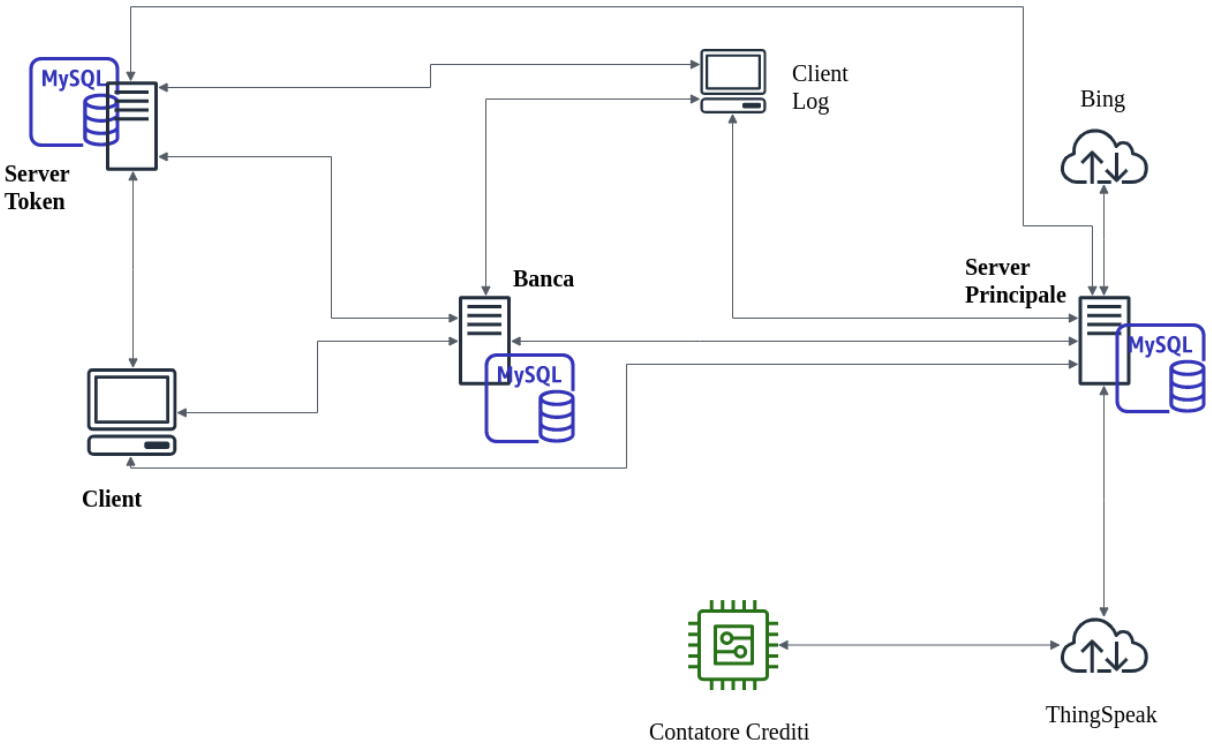


Figure 1: Connessioni di rete del progetto

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::ind
ex);
```

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 toker 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.eyJqdGkiOiI0NSIsInN1YiI6ImNpYW8iLCJpc3MiOiJub1RoaW5nc1Rva2VuIiwiYWRtIjoxLCJpYXQiOjE1Njk0MTE5NjQsImV4cCI6MTU2OTQxMjk2NH0.gUq02U9iyutv3e_nop2EOBONu-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 da 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.

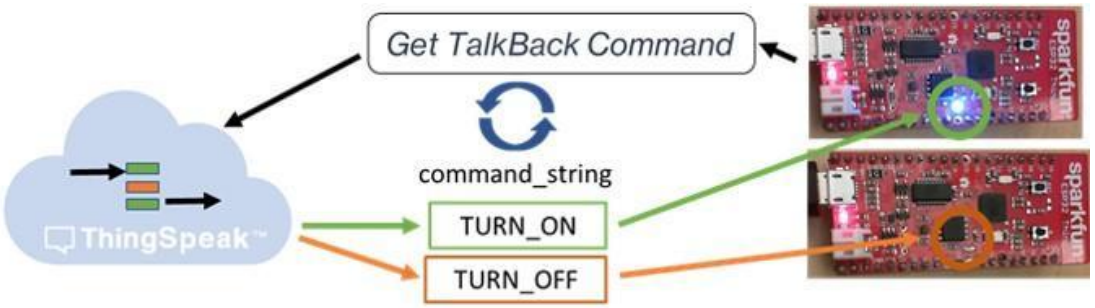


Figure 2: Esempio di comunicazione tra un dispositivo e ThingSpeak

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.

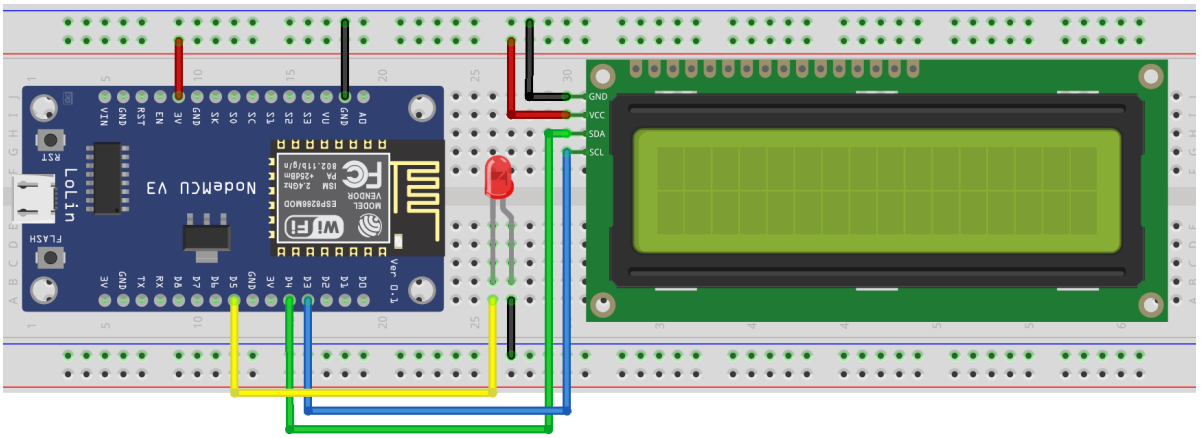


Figure 3: Schema elettrico dispositivo contatore

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

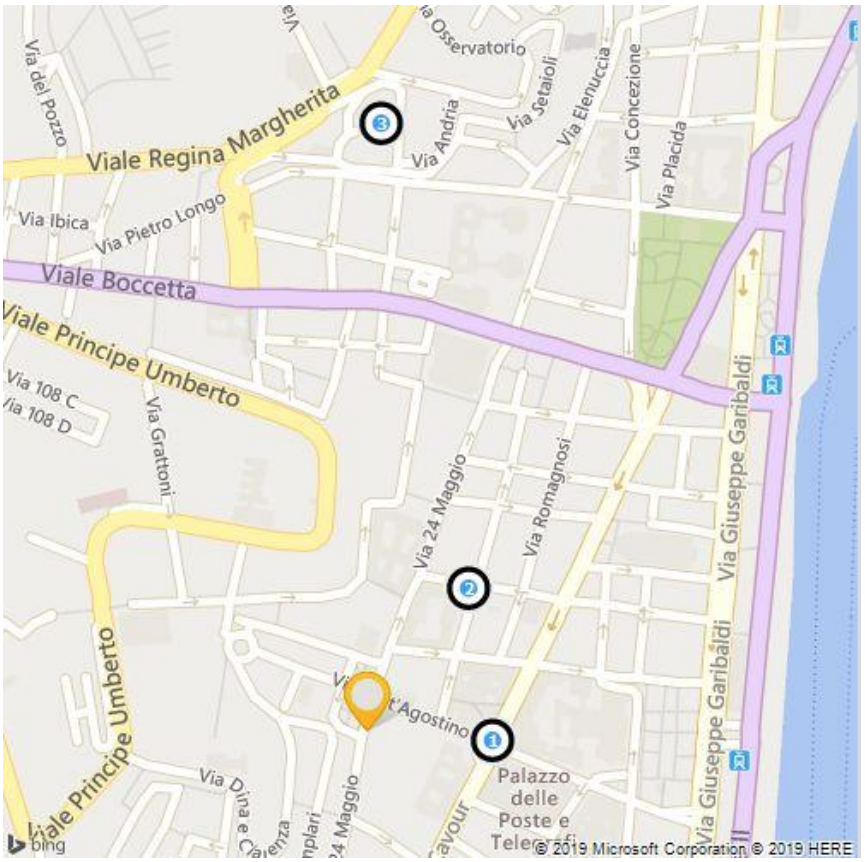


Figure 4: Risultato di una richiesta di mappa alle api di Bing

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'utilizza 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
<?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
<?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 conserve in formato xml le informazioni relative alle chiavi rsa del server principale

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

```
</RSA>
```

config/RSATOKEN.xml

questo file conserve 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 conserve 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 conserve 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 conserve 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 conserve 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>h8SNv28rWd4ajnLXSS4zdiyo2drOvQ3mAAumystV+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 conserve 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 conserve 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 conserve 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 conserve 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 conserve 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 conserve 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	/{token}/registerCard	Registra le informazioni della carta di credito inserite nel payload. Se vi è già una carta registrata per l'utente verrà sostituita.
GET	/{token}/getCard	restituisce gli ultimi 3 numeri della carta registrata dall'utente
GET	/{token}/pay	Effettua il pagamento per effettuare l'upgrade a premium. Utilizzerà le informazioni della carta di credito registrate per l'utente.
POST	/verify	Controlla i dati della transazione inseriti nel payload e ne restituisce la data di esecuzione
GET	/{token}/getLog	restituisce tutti i log delle transazioni
GET	/{token}/getLog/{user_id}	restituisce tutti i log delle transazioni dell'utente con id "user_id"
GET	/{token}/getLog/{start}/{end}	restituisce tutti i log delle transazioni nel periodo di tempo indicato

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

Core Server

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

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

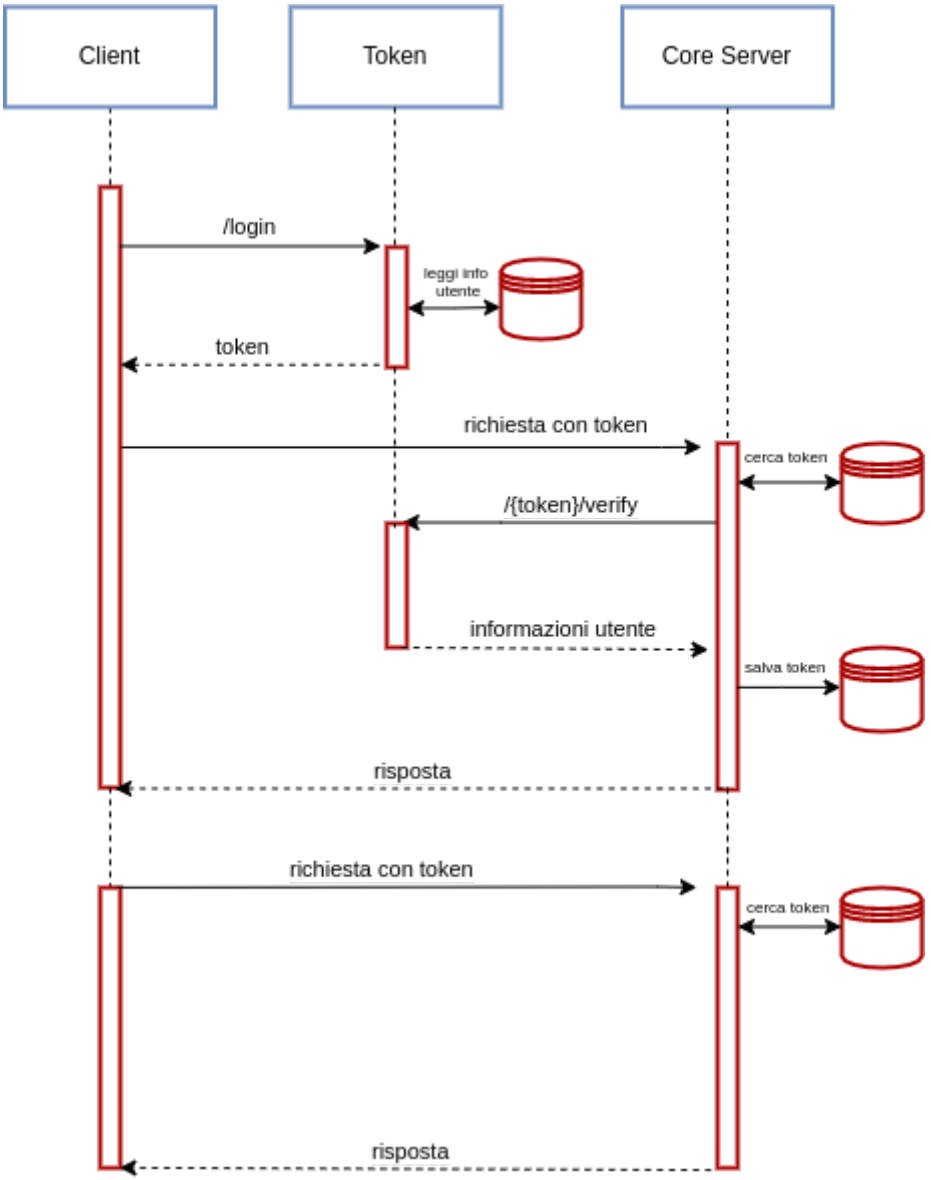


Figure 5: Diagramma di una sequenza di login

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.

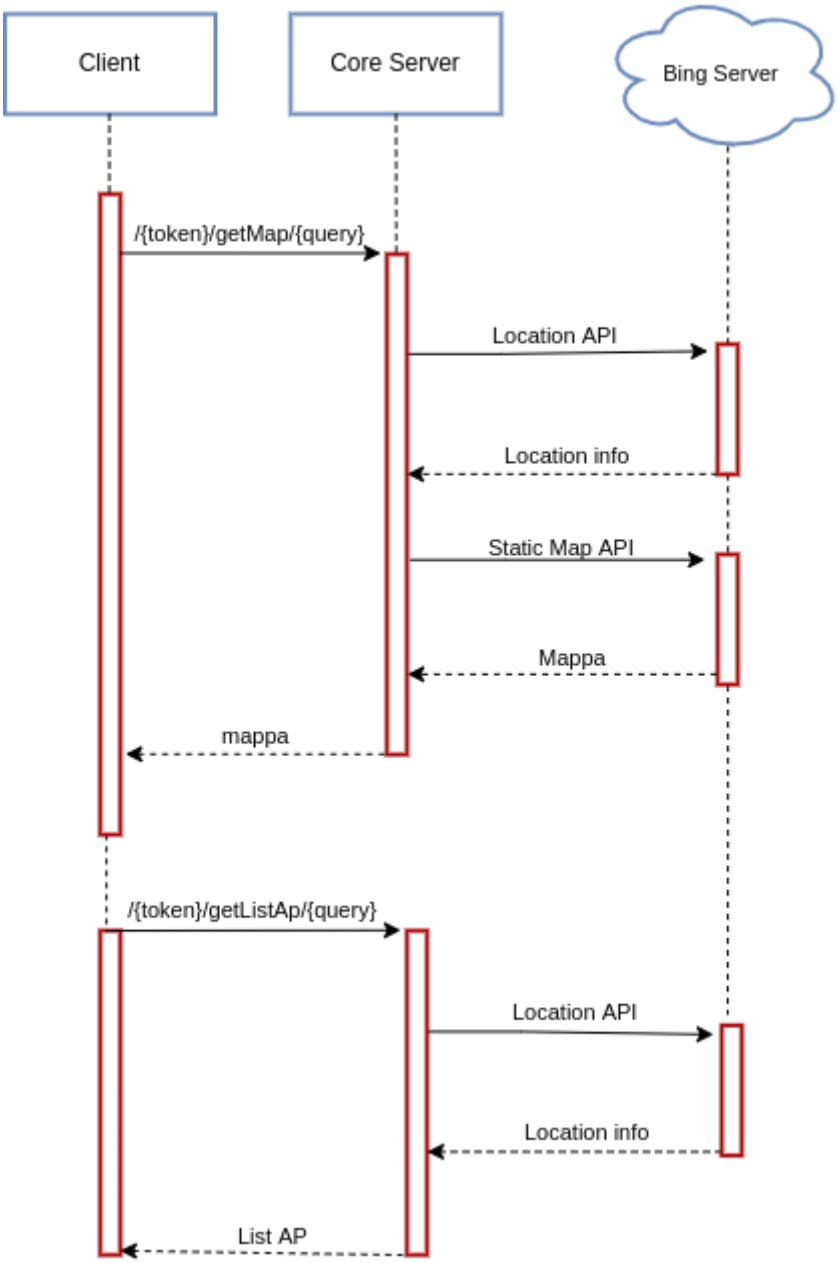


Figure 6: Diagramma di una sequenza di richiesta mappa tramite indirizzo

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.

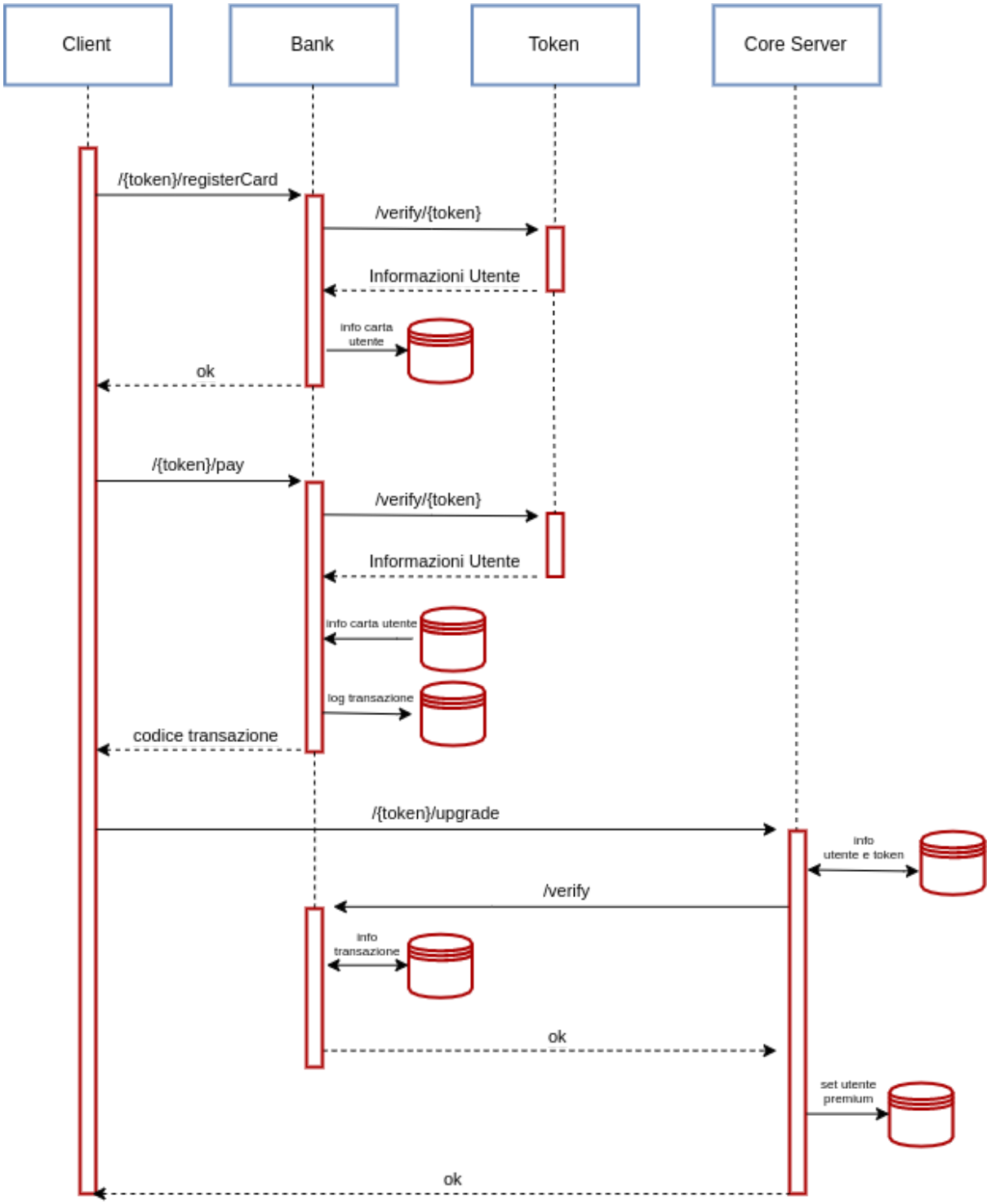


Figure 7: Diagramma della sequenza di una richiesta di upgrade a premium

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.

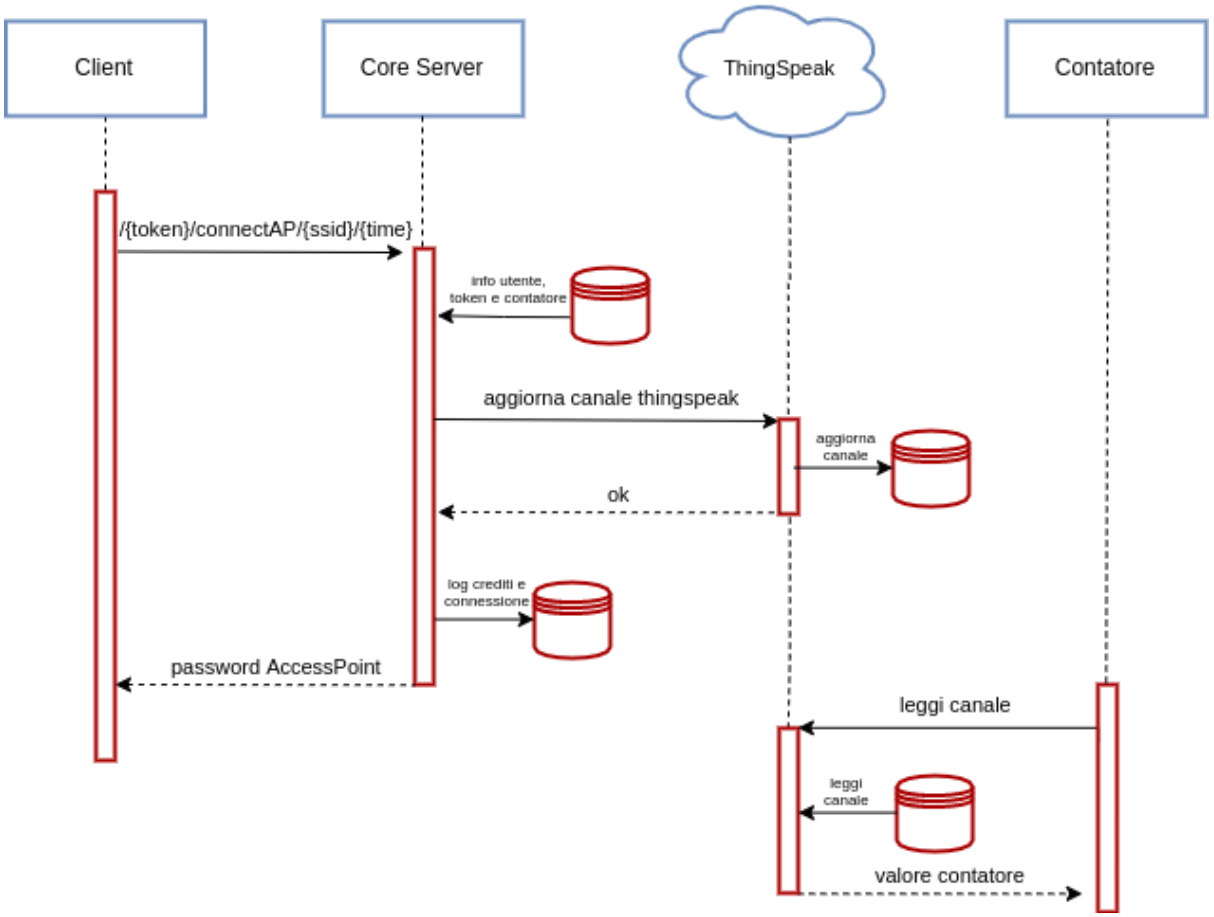


Figure 8: Diagramma di una sequenza di connessione ad un AccessPoint

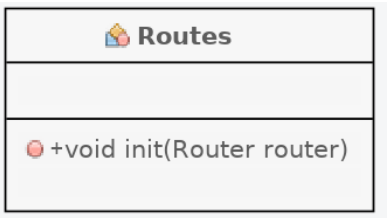
Class Diagram

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

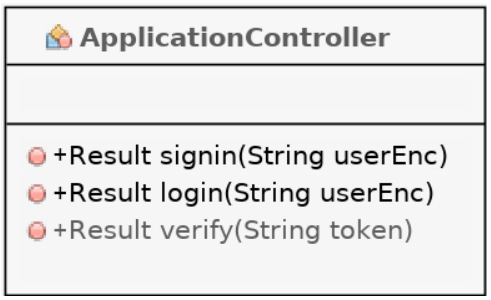
conf

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



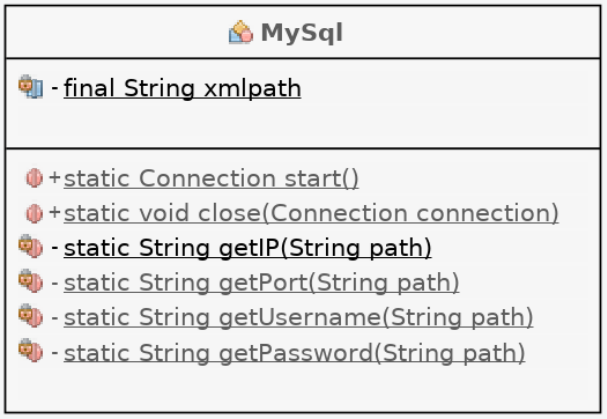
controllers

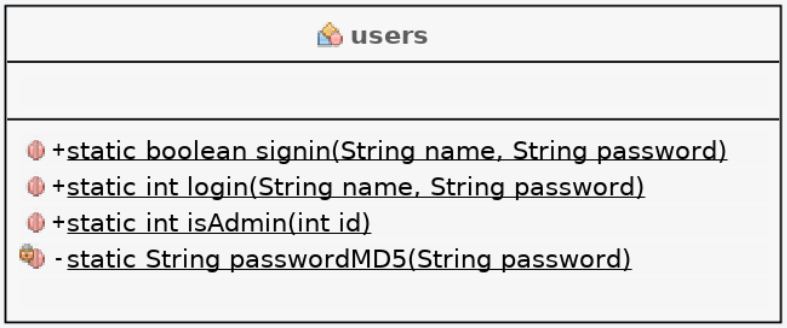
le classi classi controller contengono i metodi che vengono richiamati dalle route http



mysql

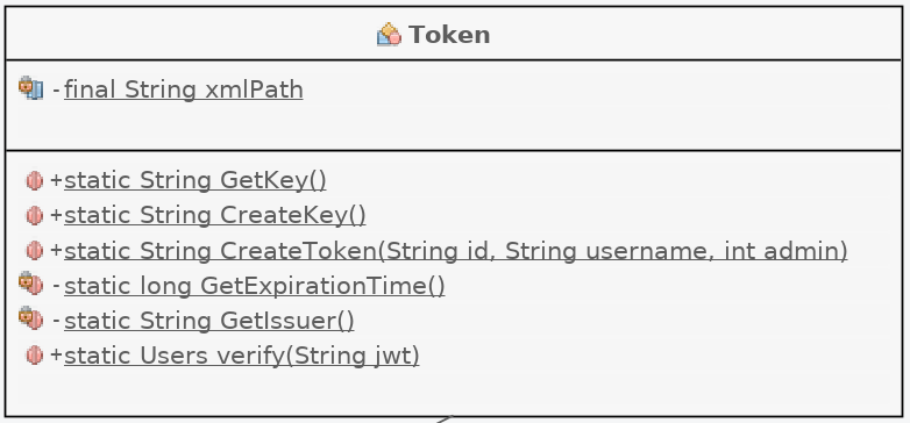
queste classi si occupano di prendere informazioni dal database o inserirle.





utility

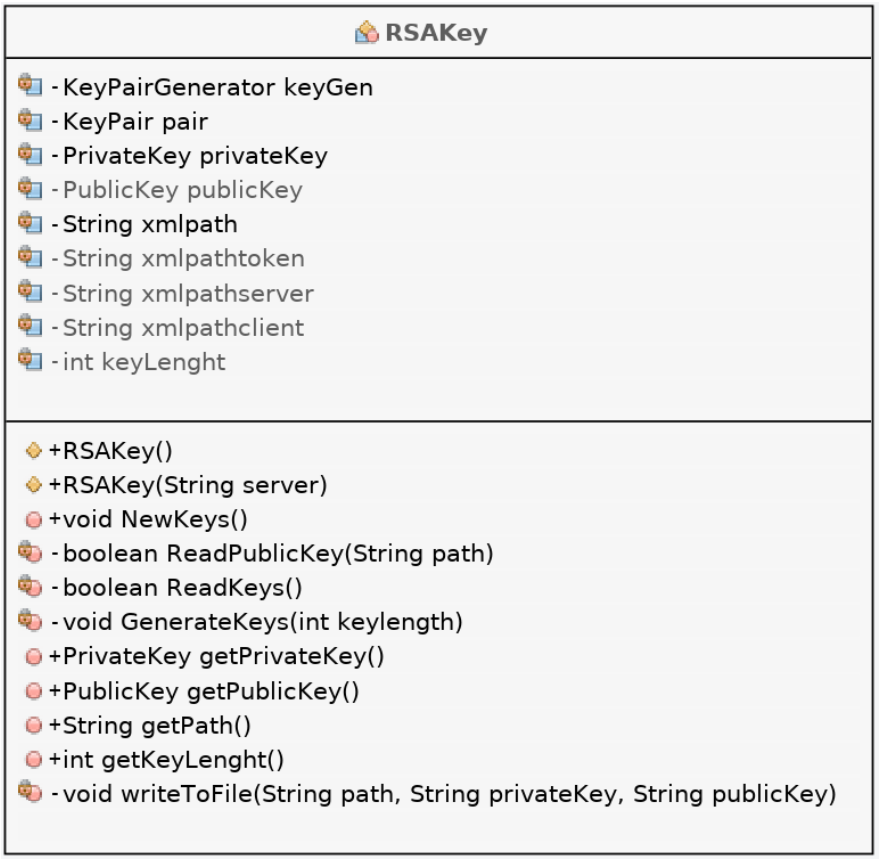
la classe token contiene tutti i metodi per che restituiscono le informazioni all'interno dei file di configurazione relativi ai token.



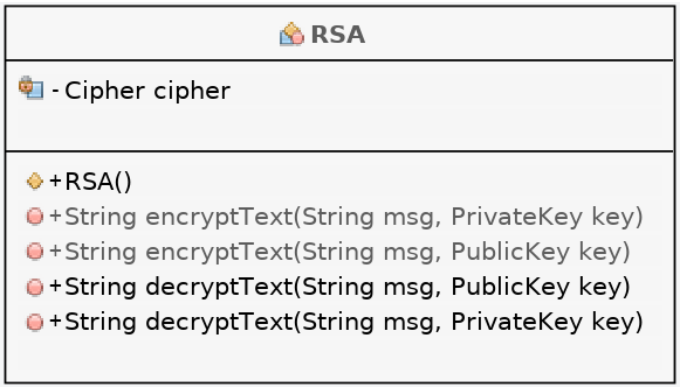
crittography

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).



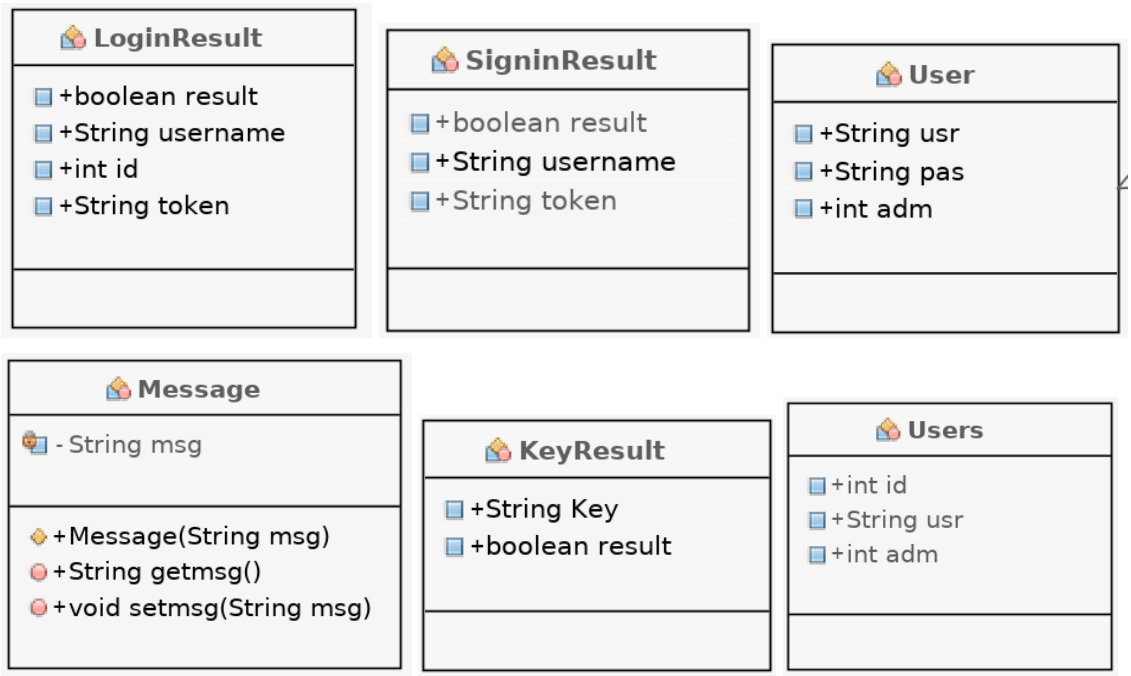
RSA: contiene i metodi utili a cifrare e decifrare un messaggio con una chiave data.



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

POJO

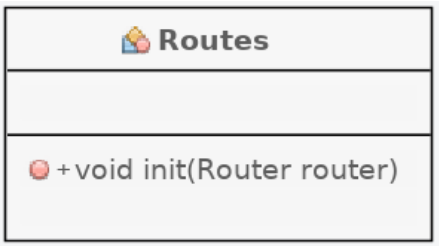
Queste Classi sono utilizzate solamente per rappresentare i dati inviati e/o ricevuti in formato json. Per definizione non sono legati ad alcuna restrizione diversa da quelle costrette dalla specifica del linguaggio Java.



Server Banca

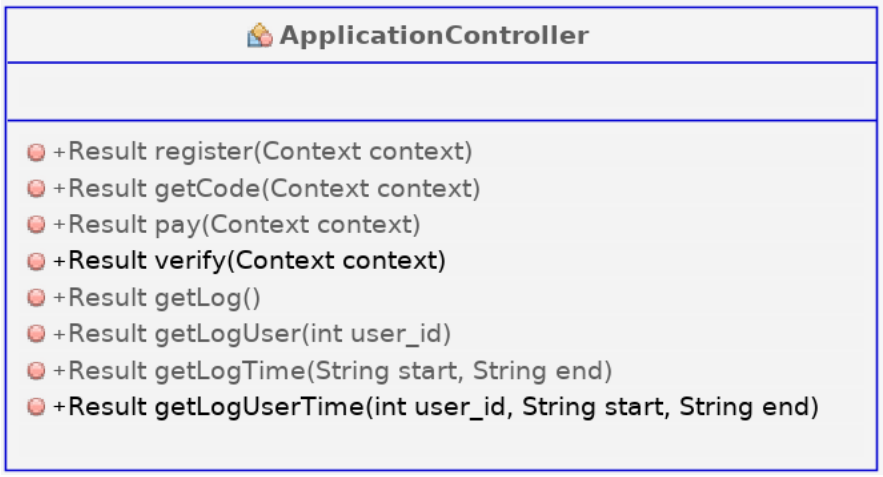
conf

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



controllers

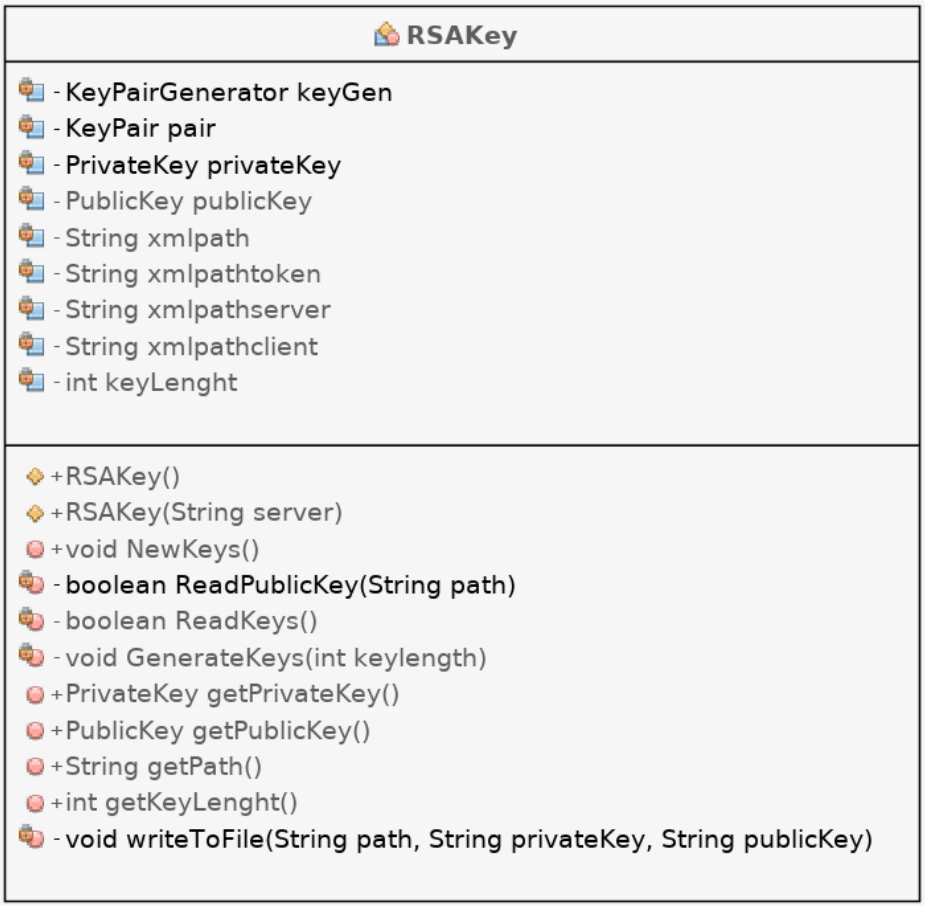
le classi classi controller contengono i metodi che vengono richiamati dalle route http



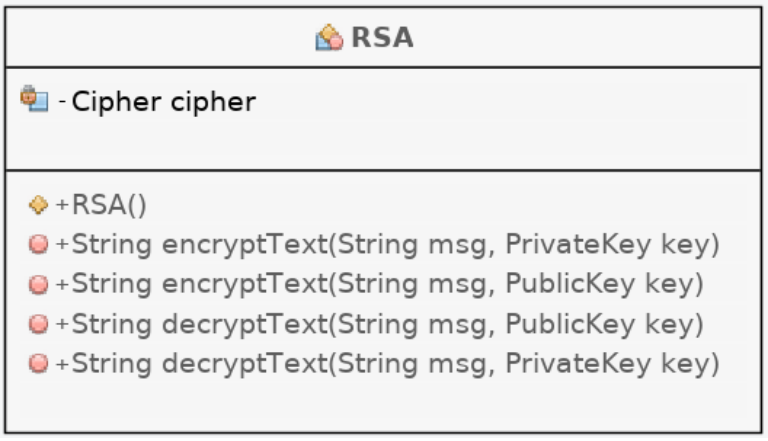
crittography

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.

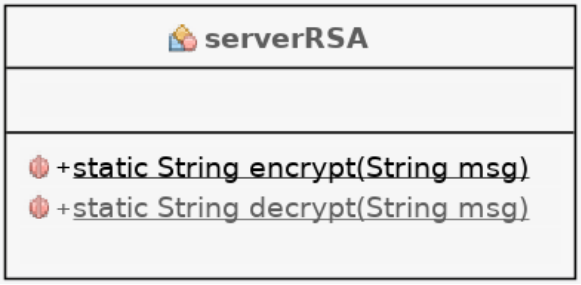
- RSAKey() inizializza le variabili private con le proprie chiavi e se non trova il proprio file di chiavi rsa ne genera di nuove.
- RSAKey(String server) inizializza le variabili private con le chiavi relative al progetto richiesto in argomento (client, bank, server, token).



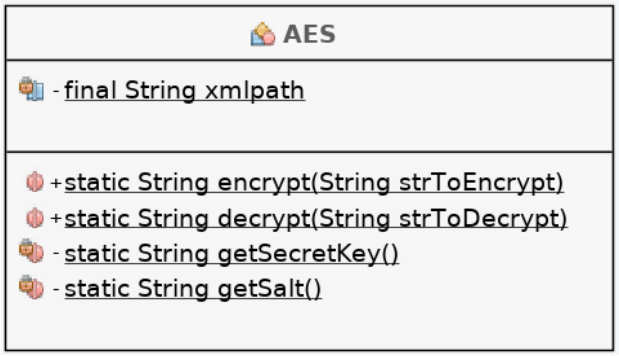
RSA: contiene i metodi utili a cifrare e decifrare un messaggio con una chiave data.



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



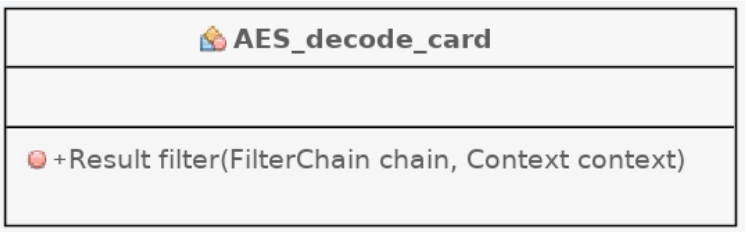
AES: contiene i metodi utili a crittografare e decrittografare una stringa con la relativa chiave presente nel file di configurazione



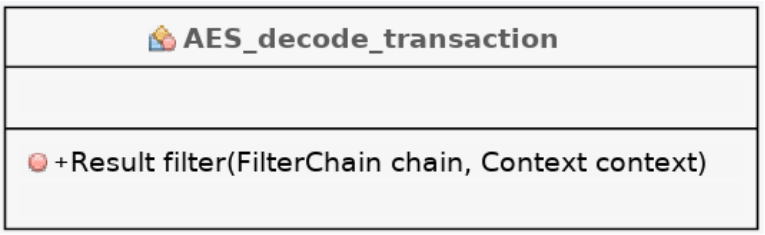
filters

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_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.

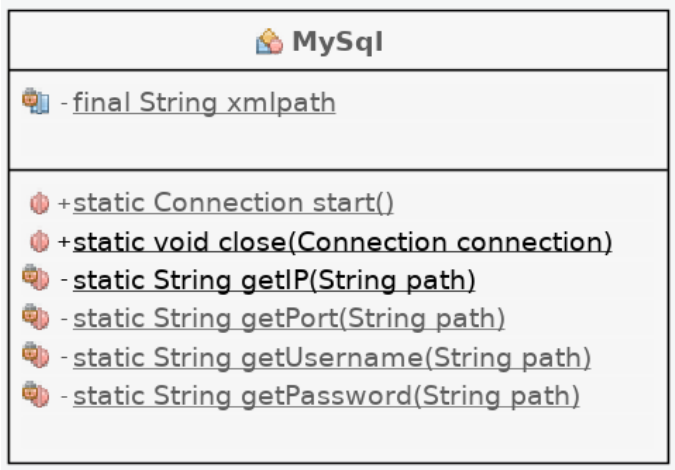


RSA_tokenVerify: decifra con l'algoritmo RSA il token, controlla sul db se è già stato precedentemente utilizzato da un utente e se non è scaduto. Se è così prende le informazioni dell'utente. Se il token non si trova nel db richiede al server token di verificarlo. Questo restituirà le informazioni dell'utente.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

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.

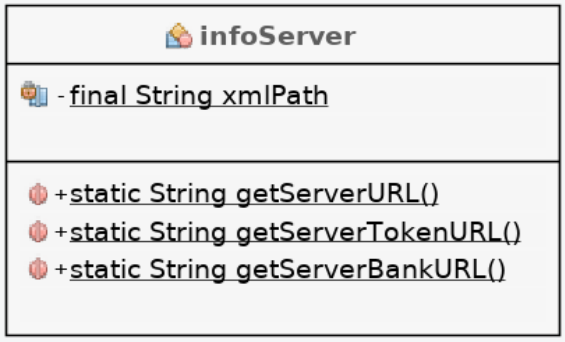
mysql

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.

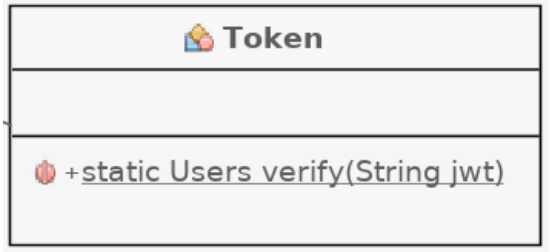


utility

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

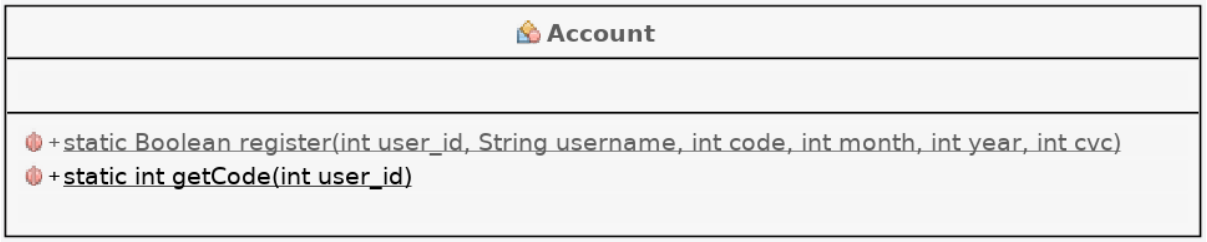


Token: contiene il metodo utile alla verifica del token. effettua una richiesta http al server token chiedendone la verifica e restituendone il risultato.

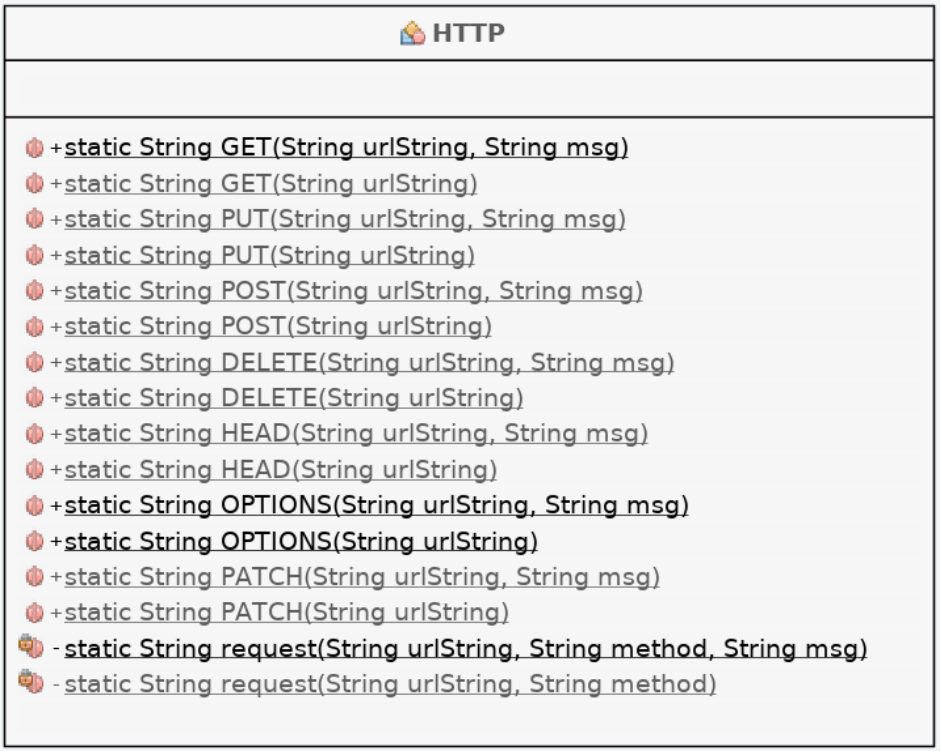


Account: contiene i metodi utili alla gestione delle carte di credito degli utenti

- 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

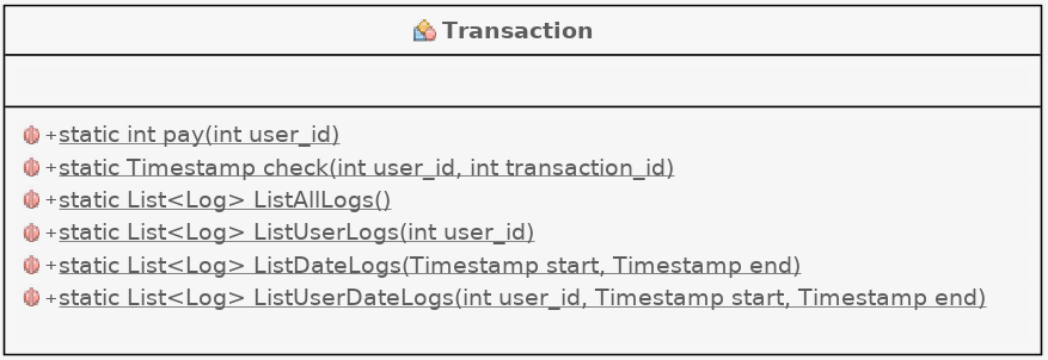


HTTP: contiene i metodi per fare delle richieste http.



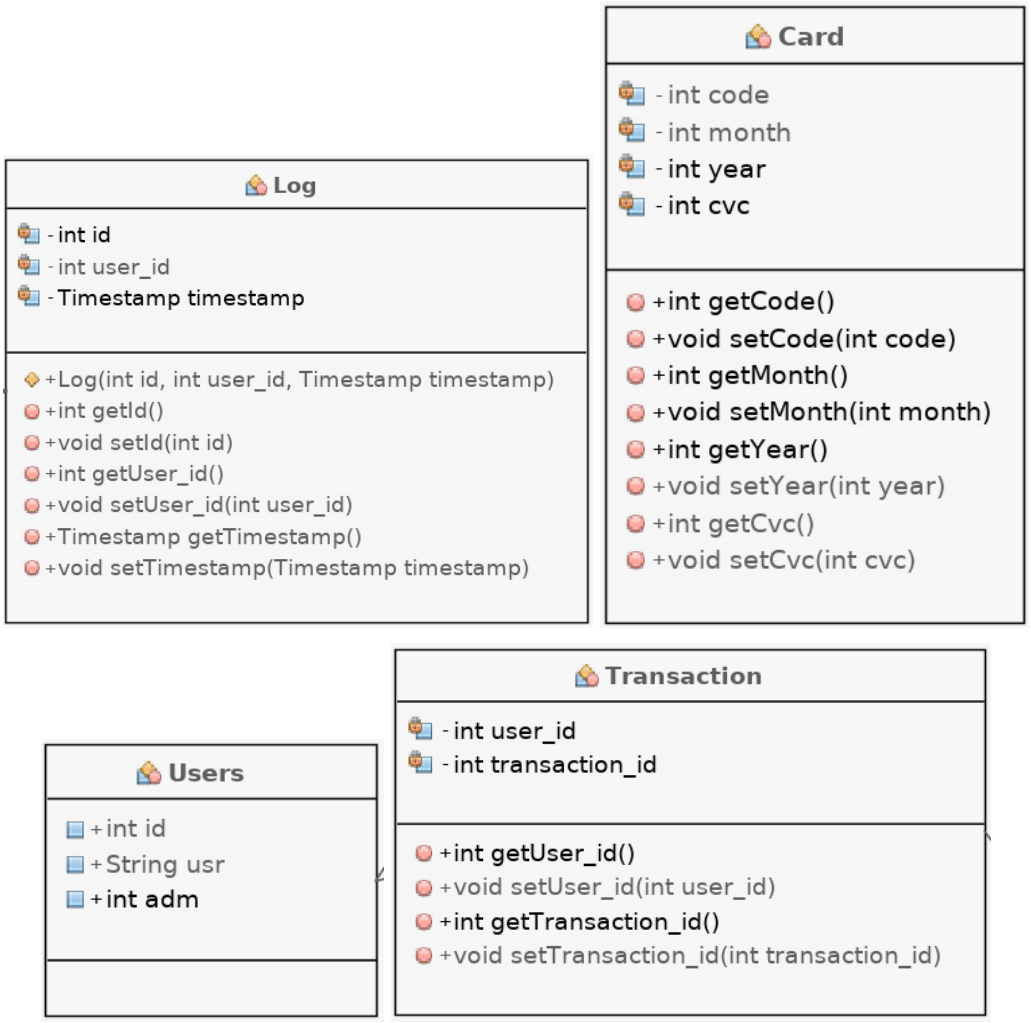
Transaction: contiene i metodi necessari alla gestione delle transazioni.

- 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.



POJO

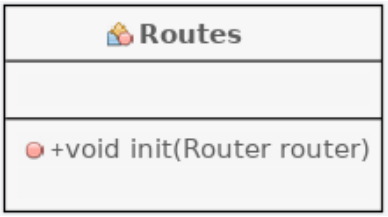
Queste Classi sono utilizzate solamente per rappresentare i dati inviati e/o ricevuti in formato json. Per definizione non sono legati ad alcuna restrizione diversa da quelle costrette dalla specifica del linguaggio Java.



Core Server

conf

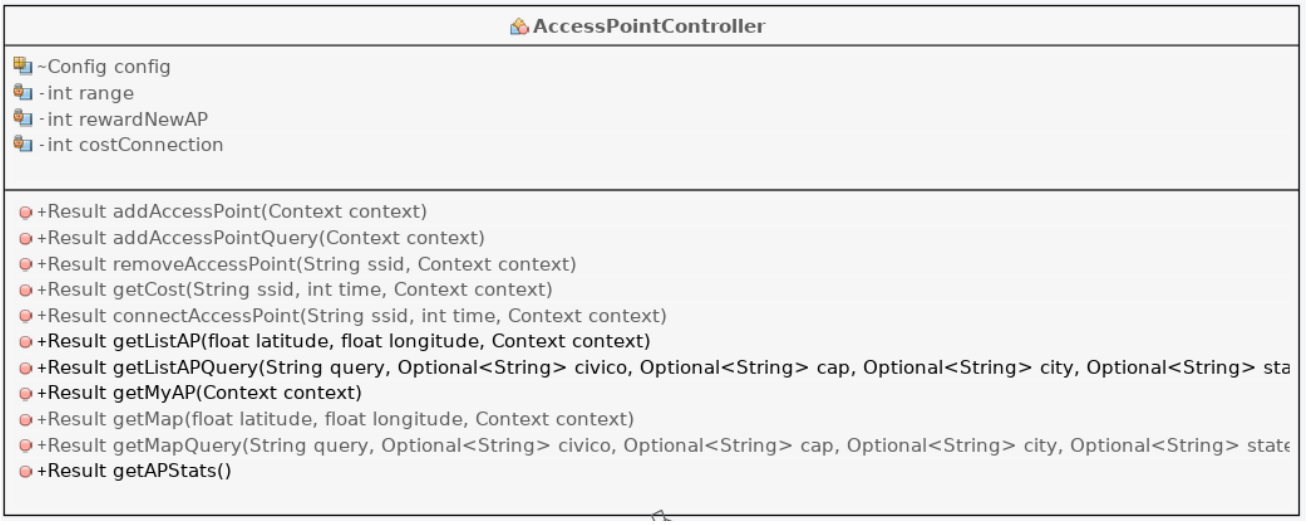
Queste classi sono necessarie al funzionamento di un qualsiasi progetto ninja. La classe route conserva tutte le route http nel metodo init.



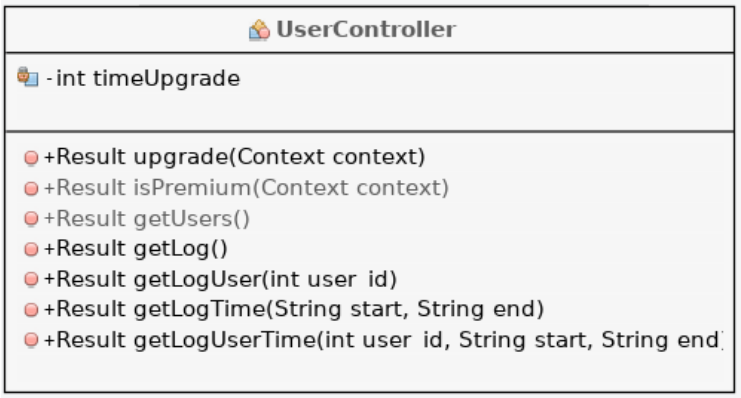
controllers

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

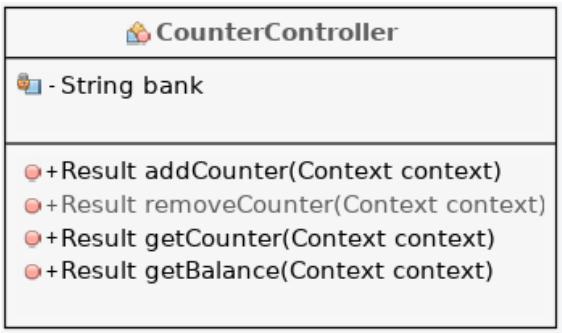
AccessPointController: contiene tutti i metodi per gestire le richieste che riguardano gli AccessPoint



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

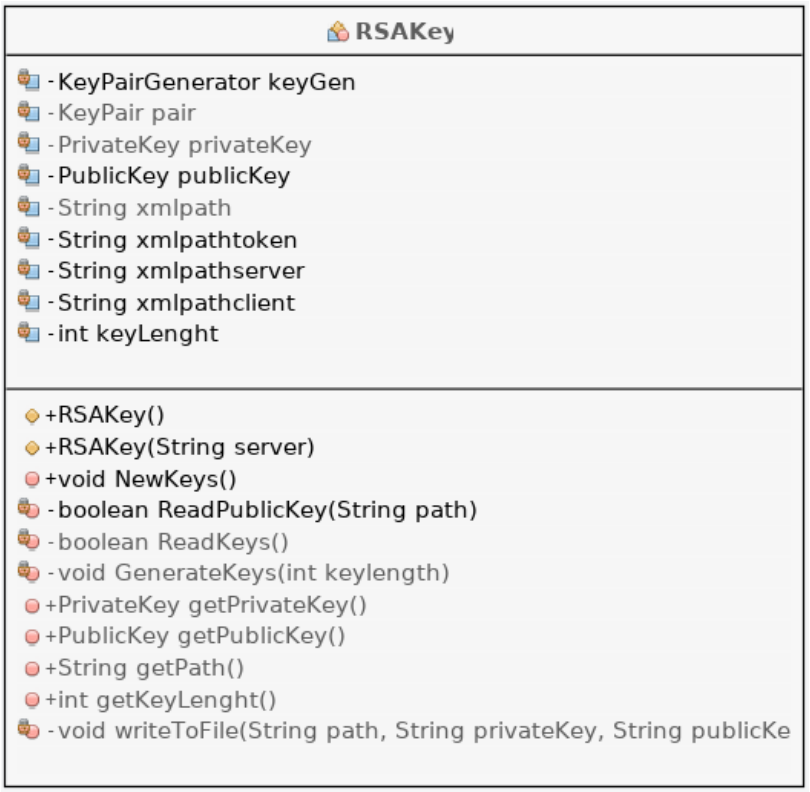


crittography

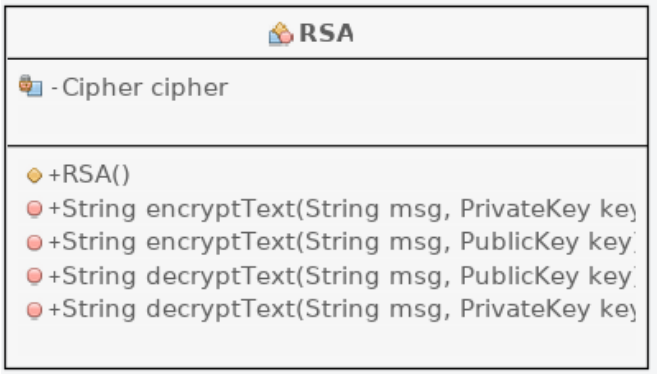
le classi in questo package servono per ciò che riguarda la crittografia.

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)

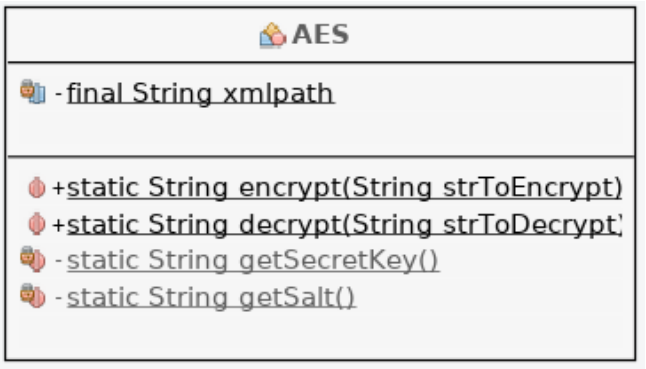


RSA: contiene i metodi utili a cifrare e decifrare un messaggio con una chiave data.



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.

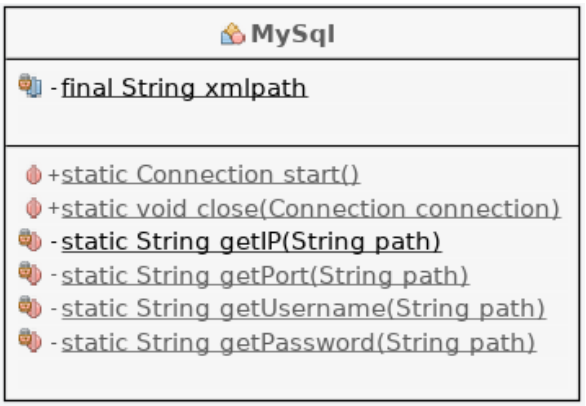
AES_decode_requestAP: controlla il body della richiesta, la decifra con l'algoritmo di AES256, ed estrae le informazioni riguardanti gli AccessPoint. Si ipotizza che siano state formattate in json secondo la classe POJO.AccessPoint
Se dovesse riscontrare qualche problema restituisce errore 401.

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.

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.

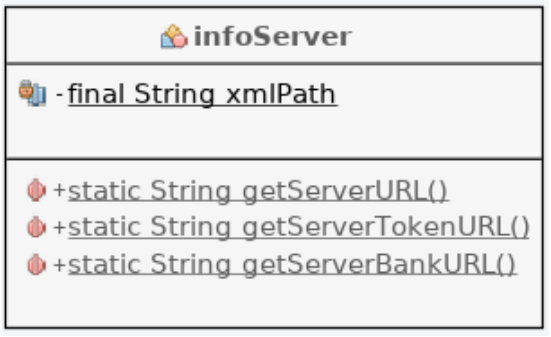
mysql

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.

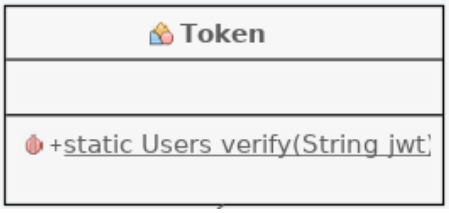


utility

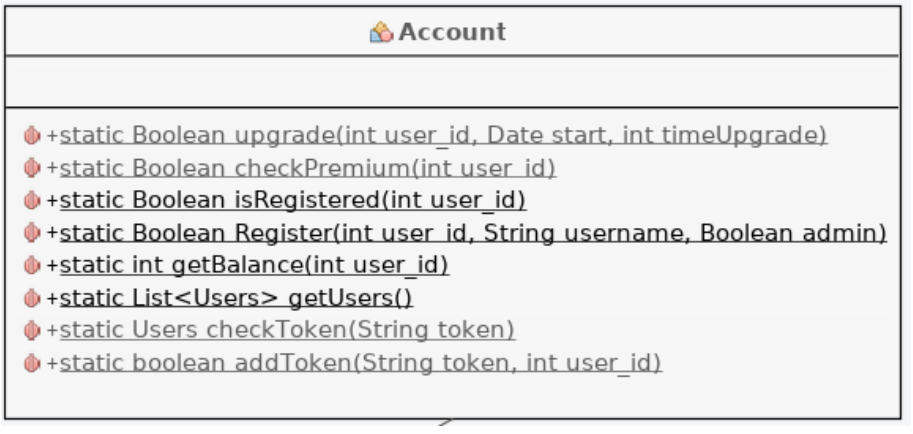
infoServer: restituisce le informazioni sul server contenute nel relativo file di configurazione.



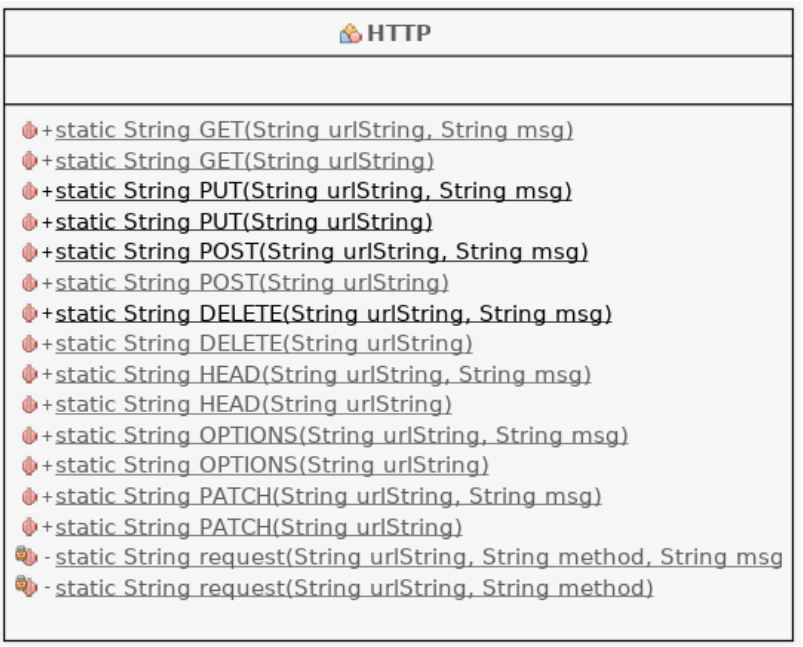
Token: contiene il metodo utile alla verifica del token. effettua una richiesta http al server token chiedendone la verifica e restituendone il risultato.



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

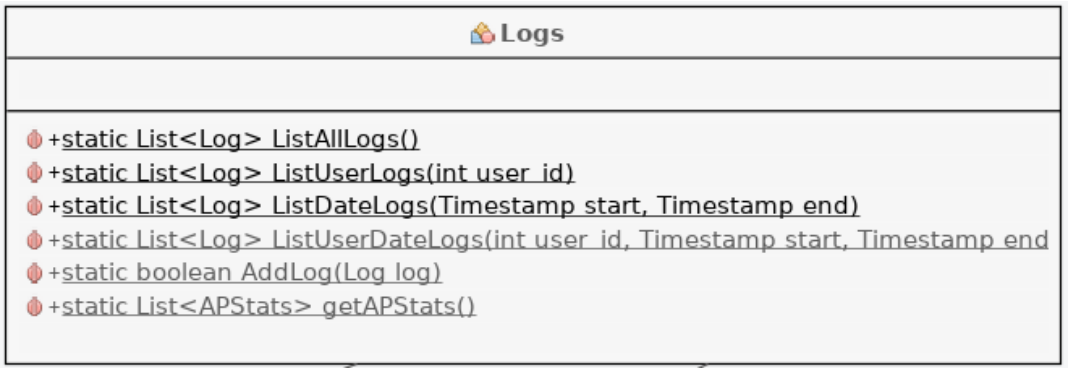


HTTP: contiene i metodi per fare delle richieste http.



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

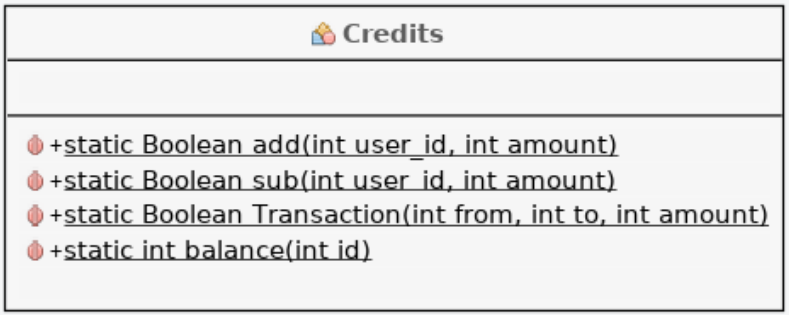
Logs: contiene i metodi utili a leggere i log e a inserirli nel db.



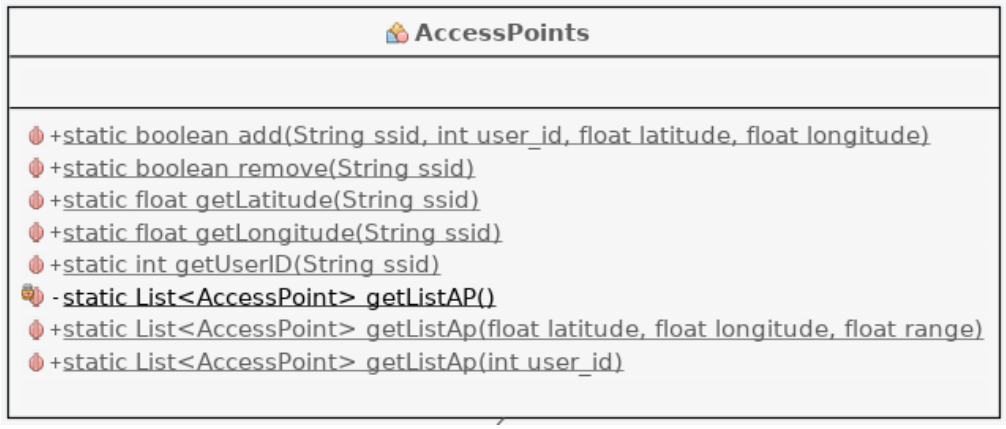
ThingSpeaks: una volta inizializzate con un costruttore le variabili relative alle API di thingspeak è possibile aggiornare il relativo canale. Inoltre contiene metodi per la gestione dei dispositivi contatore, è infatti possibile inserire un contatore e leggere le informazione dei contatori.

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.

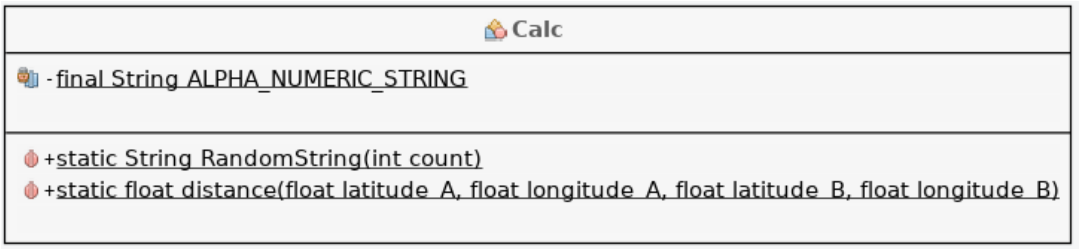


AccessPoints: contiene tutto ciò che può essere utile a gestire gli Access Point. Aggiungerli nel db, rimuoverli, elencarli, ecc…



Calc: raccoglie metodi di operazioni generiche

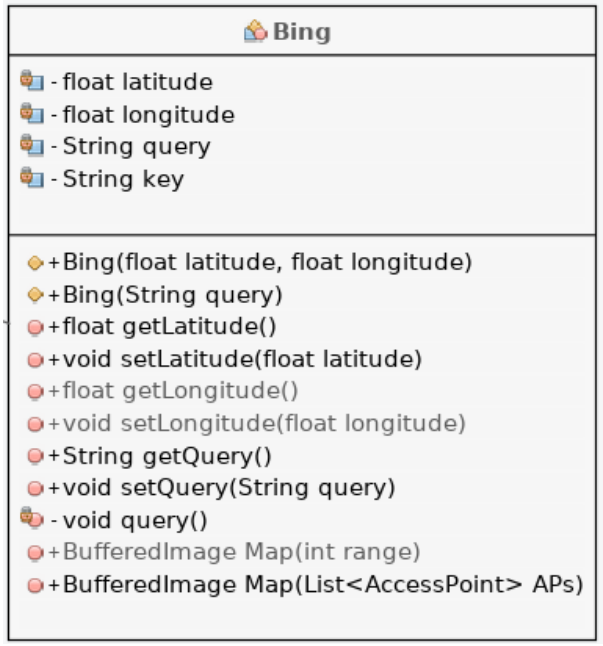
- RandomString(..) restituisce una stringa casuale, il risultato è stato utilizzato come password temporanea per connettersi agli Access Points
- distance(...) restituisce la distanza tra due coordinate. Funzione necessaria per capire se un AP si trova in una determinata area.



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.

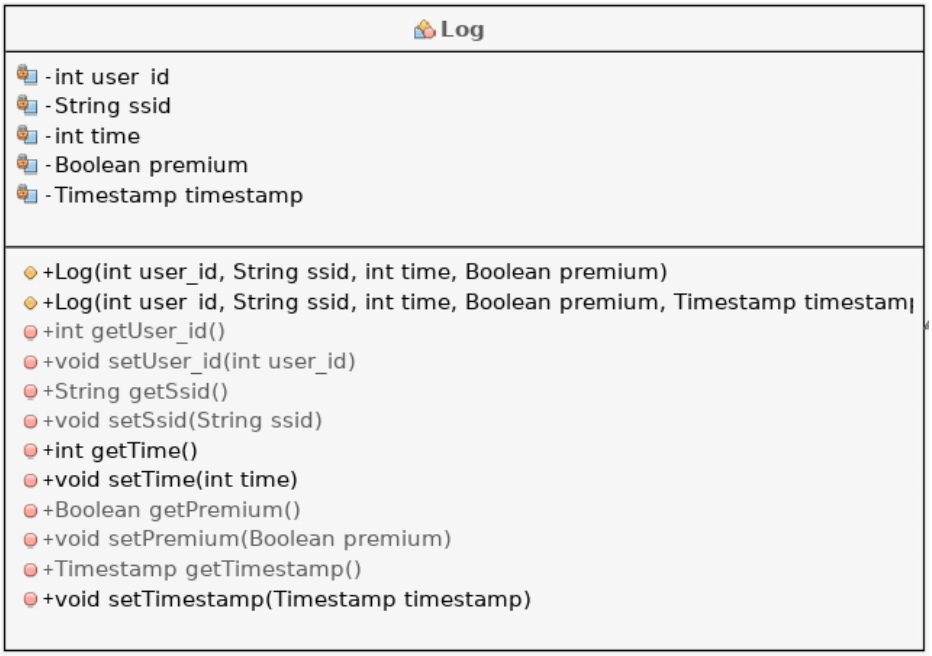
- BIng(float latitude, float longitude): inizializza le coordinate
- una volta inizializzate le variabili latitude e longitude, oppure query, tramite il costruttore designato è possibile richiamare le diverse api
- Map(List<AccessPoint> ap): richiama le api di bing e restituisce una mappa con centro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

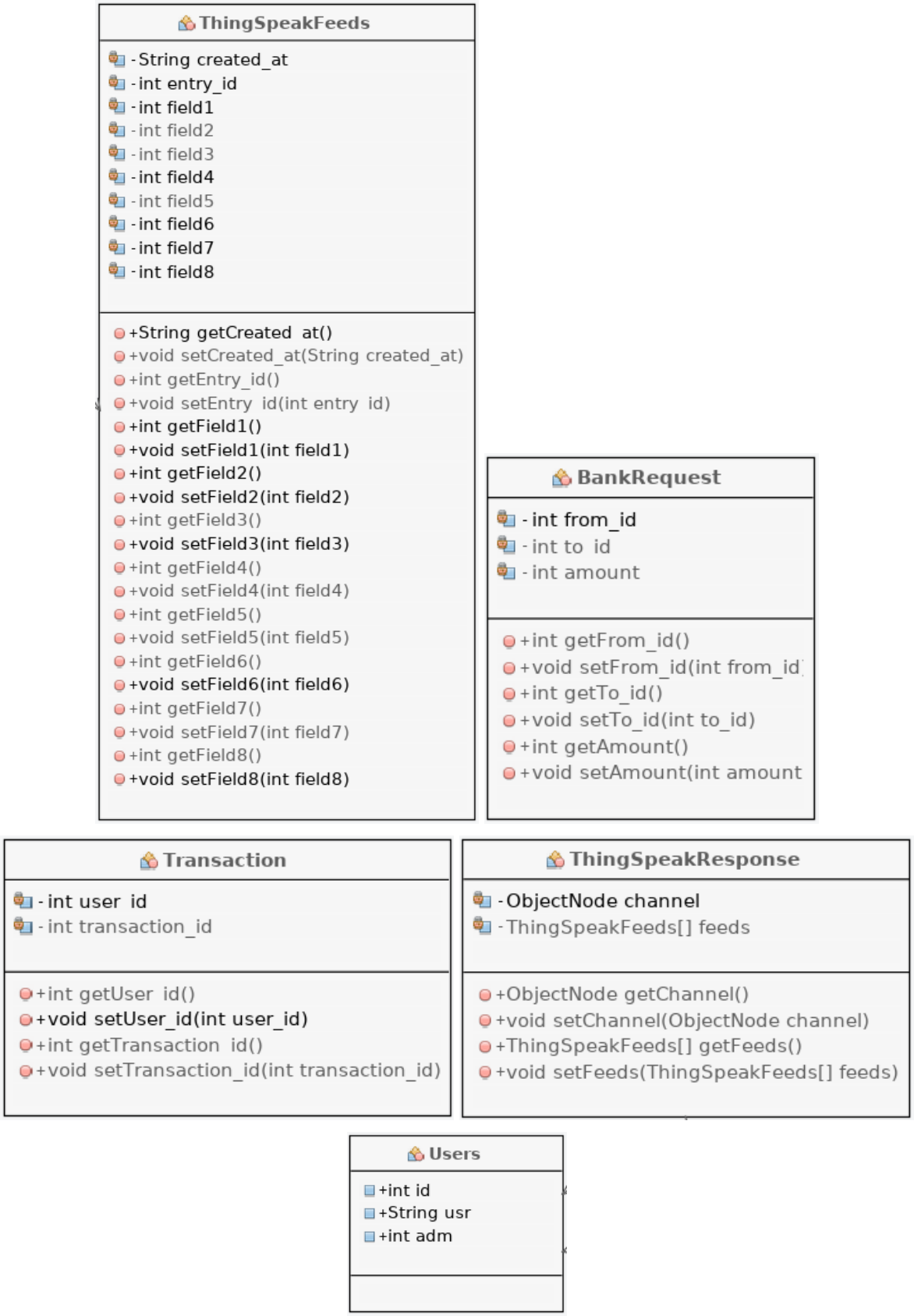


POJO

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.


AccessPoint
-String ssid
-float latitude
-float longitude
-String query
-String cap
-String city
-String state
-String civico
+String getSsid()
+void setSsid(String ssid)
+float getLatitude()
+void setLatitude(float latitude)
+float getLongitude()
+void setLongitude(float longitude)
+String getQuery()
+void setQuery(String query)
+String getCap()
+void setCap(String cap)
+String getCity()
+void setCity(String city)
+String getState()
+void setState(String state)
+String getCivico()
+void setCivico(String civico)
APStats
-String ssid
-int connections
-int users
-double average_time
+String getSsid()
+void setSsid(String ssid)
+int getConnections()
+void setConnections(int connections)
+int getUsers()
+void setUsers(int users)
+double getAverage_time()
+void setAverage_time(double average_time
Config
-int range
-int rewardNewAP
-int costConnection
-int timeUpgrade
-int expirationToken
+int getRange()
+void setRange(int range)
+int getRewardNewAP()
+void setRewardNewAP(int rewardNewAP)
+int getCostConnection()
+void setCostConnection(int costConnection)
+int getTimeUpgrade()
+void setTimeUpgrade(int timeUpgrade)
+int getExpirationToken()
+void setExpirationToken(int expirationToken
Counter
-String api
-int channelId
-int field
+String getApi()
+void setApi(String api)
+int getChannelId()
+void setChannelId(int channelId
+int getField()
+void setField(int field)

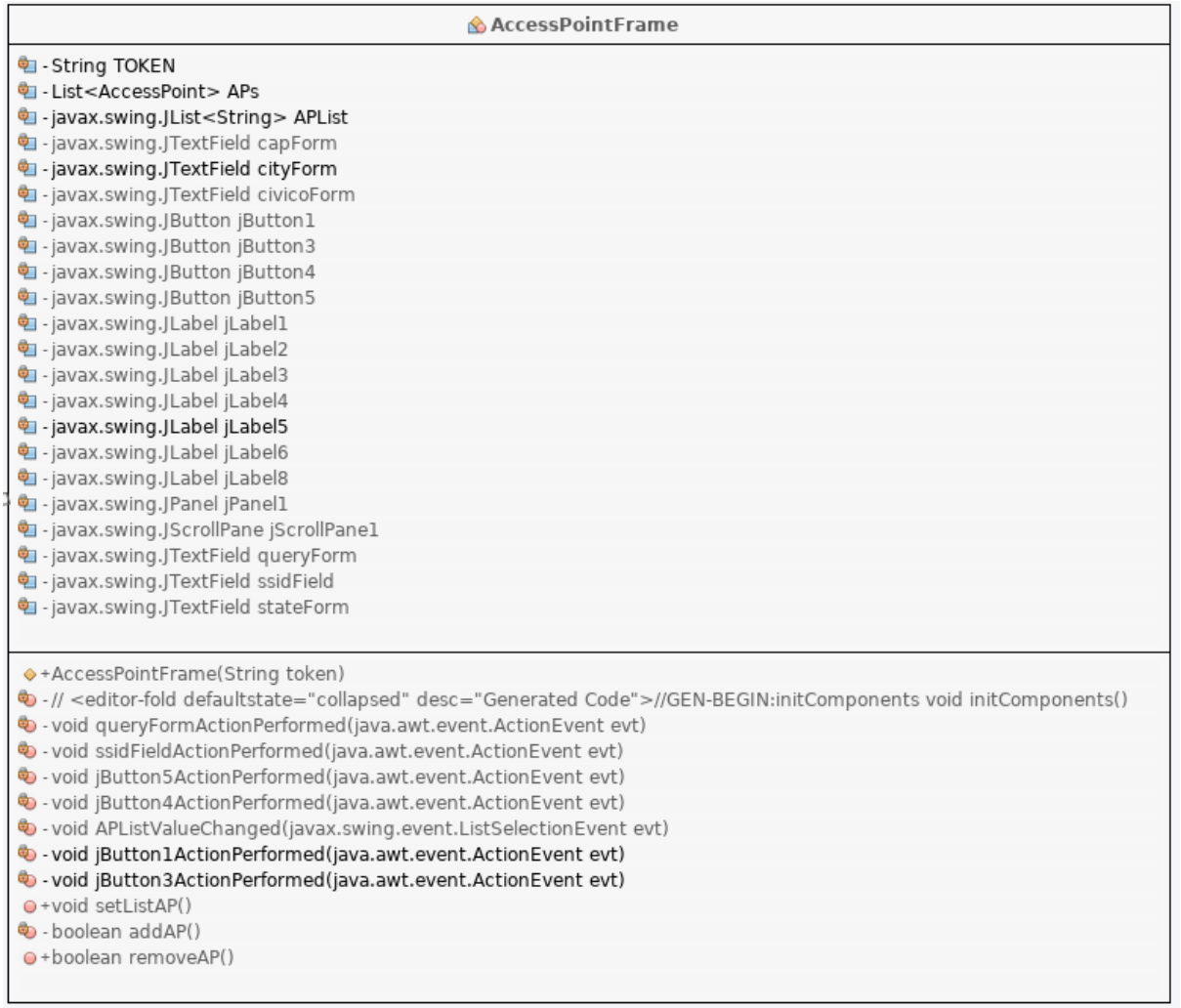

Log
-int user_id
-String ssid
-int time
-Boolean premium
-Timestamp timestamp
+Log(int user_id, String ssid, int time, Boolean premium)
+Log(int user_id, String ssid, int time, Boolean premium, Timestamp timestam
+int getUser_id()
+void setUser_id(int user_id)
+String getSsid()
+void setSsid(String ssid)
+int getTime()
+void setTime(int time)
+Boolean getPremium()
+void setPremium(Boolean premium)
+Timestamp getTimestamp()
+void setTimestamp(Timestamp timestamp)


ThingSpeakFeeds
-String created_at
-int entry_id
-int field1
-int field2
-int field3
-int field4
-int field5
-int field6
-int field7
-int field8
+String getCreated_at()
+void setCreated_at(String created_at)
+int getEntry_id()
+void setEntry_id(int entry_id)
+int getField1()
+void setField1(int field1)
+int getField2()
+void setField2(int field2)
+int getField3()
+void setField3(int field3)
+int getField4()
+void setField4(int field4)
+int getField5()
+void setField5(int field5)
+int getField6()
+void setField6(int field6)
+int getField7()
+void setField7(int field7)
+int getField8()
+void setField8(int field8)
BankRequest
-int from_id
-int to_id
-int amount
+int getFrom_id()
+void setFrom_id(int from_id
+int getTo_id()
+void setTo_id(int to_id)
+int getAmount()
+void setAmount(int amount
Transaction
-int user_id
-int transaction_id
+int getUser_id()
+void setUser_id(int user_id)
+int getTransaction_id()
+void setTransaction_id(int transaction_id)
ThingSpeakResponse
-ObjectNode channel
-ThingSpeakFeeds[] feeds
+ObjectNode getChannel()
+void setChannel(ObjectNode channel)
+ThingSpeakFeeds[] getFeeds()
+void setFeeds(ThingSpeakFeeds[] feeds)
Users
+int id
+String usr
+int adm

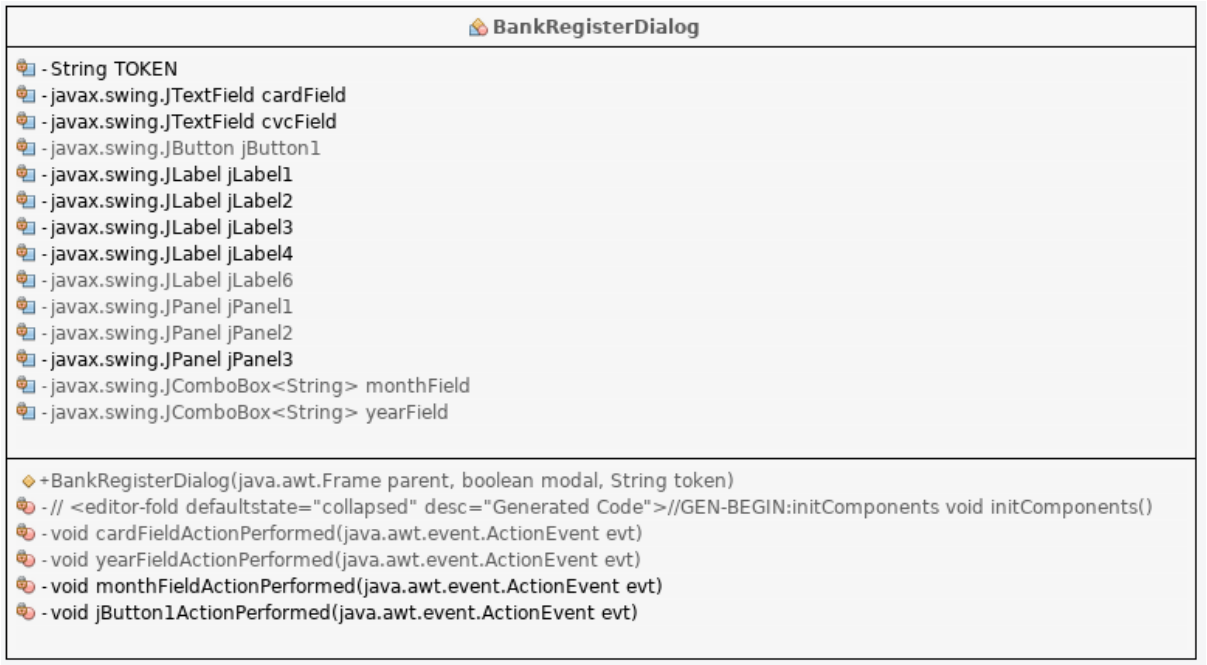
Client

GUI

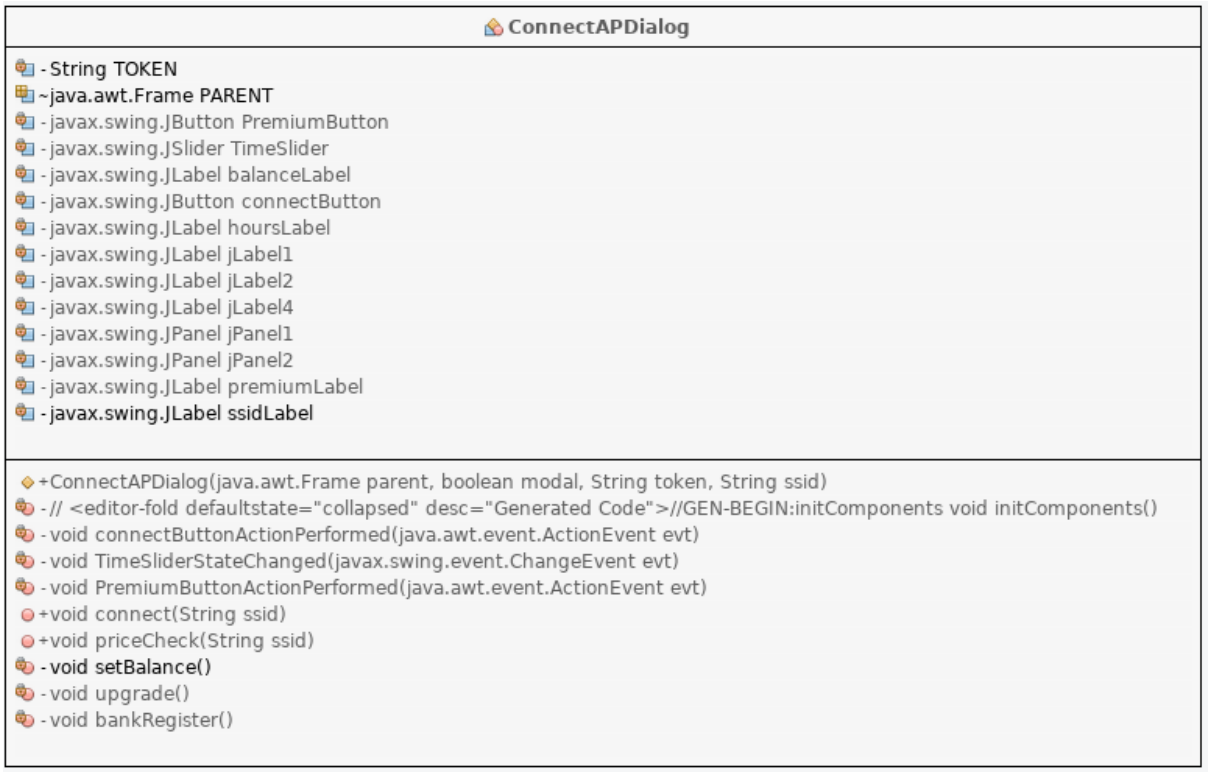
AccessPointFrame: impementa l'interfaccia per gestire i propri AccessPoint: aggiungerli, rimuoverli e visualizzarli.



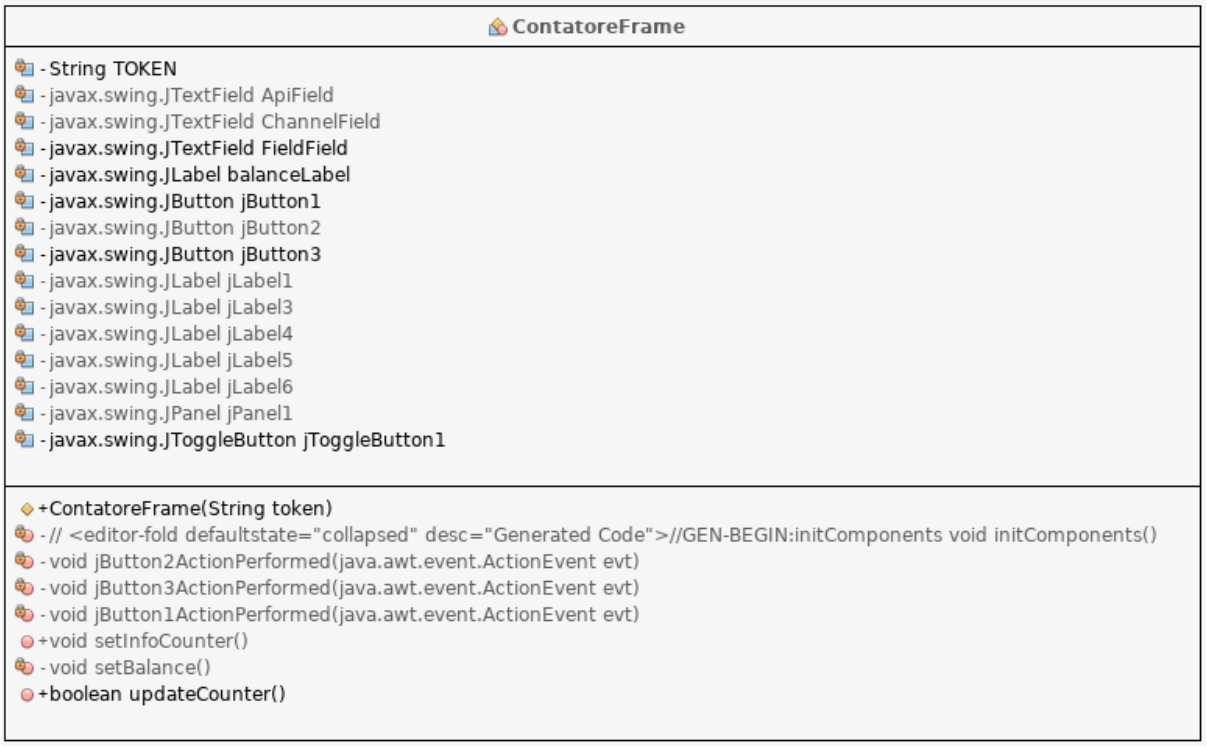
BankRegisterDialog: implementa il dialog bloccante che permette di registrare una nuova carta di credito sul server della banca.



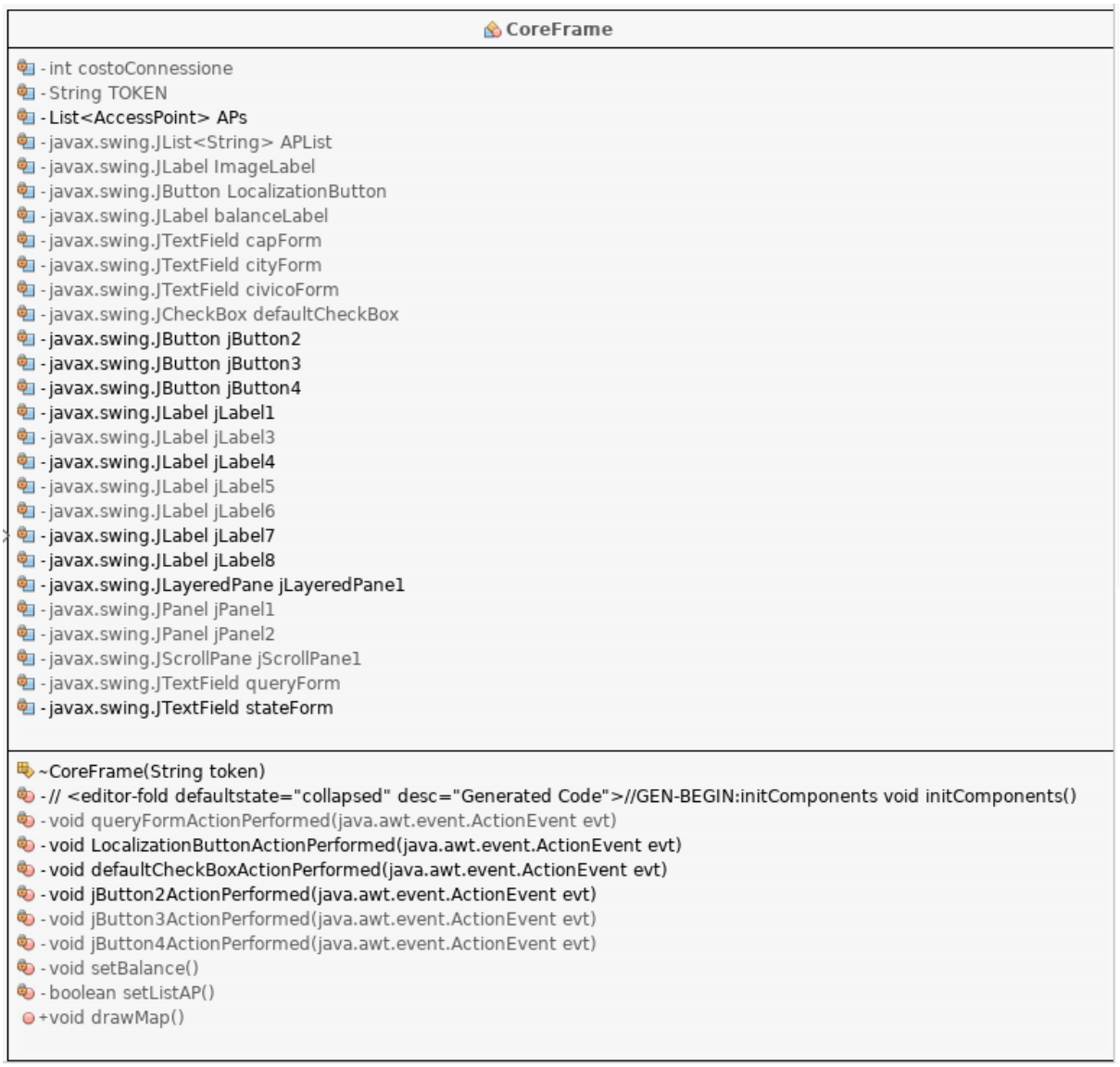
ConnectAPDialog: implementa il dialog bloccante che permette di scegliere il tempo di connessione ad un AccessPoint e richiedere la connessione. Inoltre fornisce la possibilita di effettuare l'upgrade a premium.



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 jToggleButton1
+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.



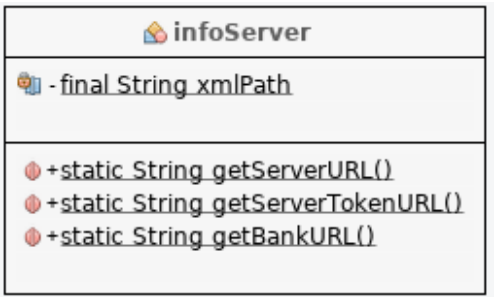
noThingFormLogin: implementa il form per il login

noThingFormSignin: implementa il form per la registrazione

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

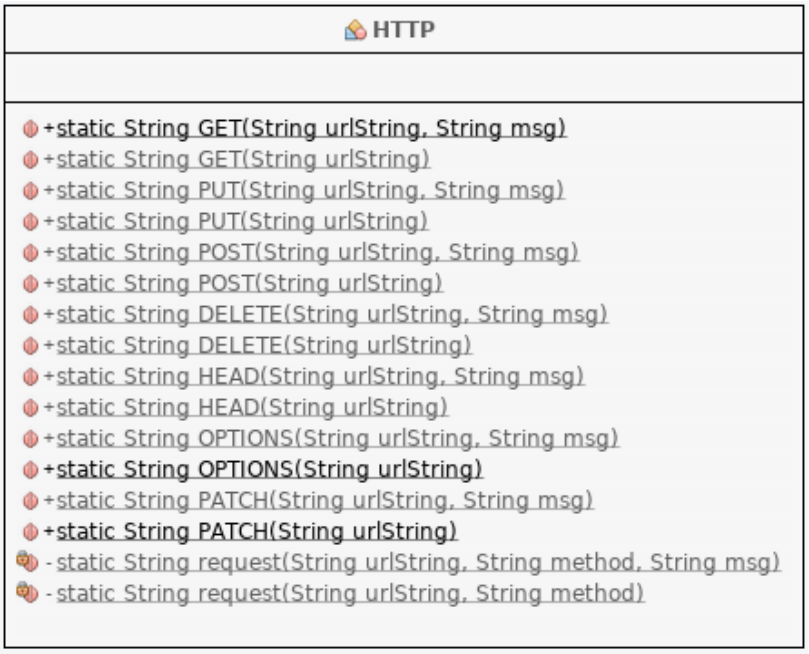


TokenServer

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

Request

HTTP: contiene i metodi per effettuare delle richieste http.

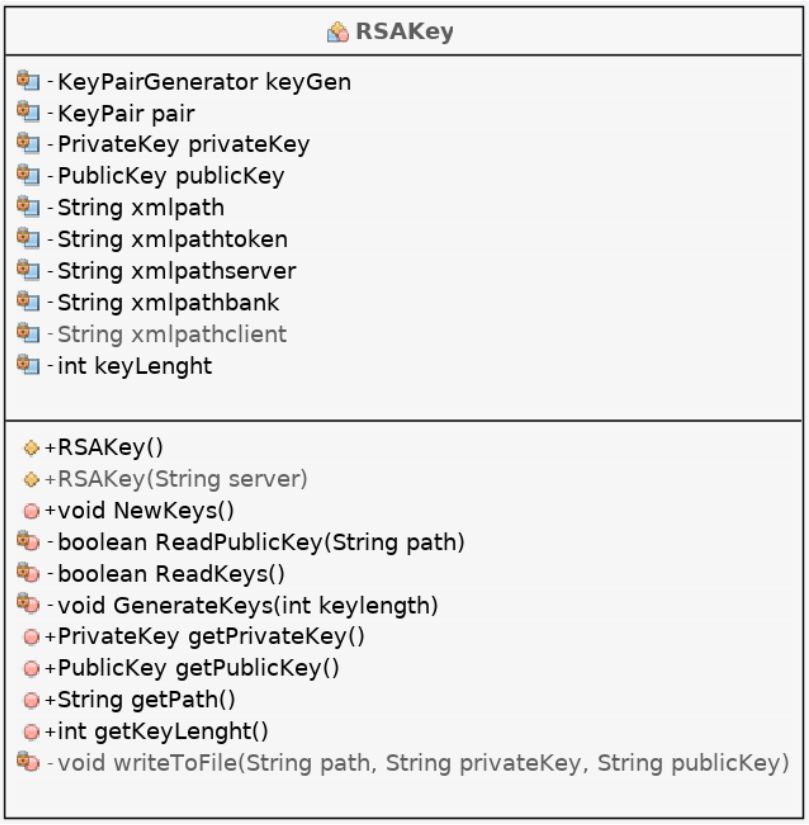


crittography

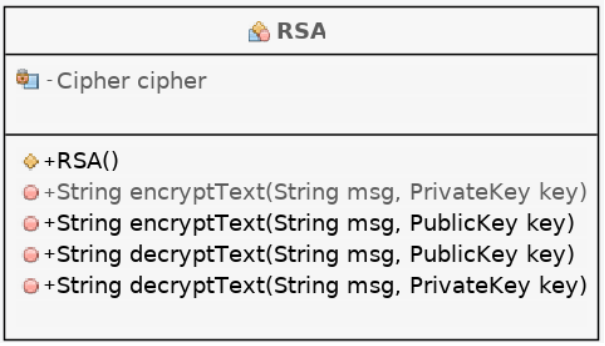
le classi in questo package servono per ciò che riguarda la crittografia.

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

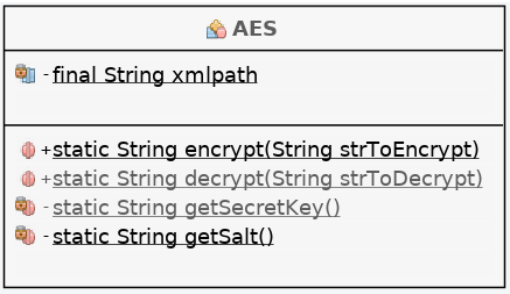
- RSAKey() inizializza le variabili private con le proprie chiavi e se non trova il proprio file di chiavi rsa ne genera di nuove
- RSAKey(String server) inizializza le variabili private con le chiavi relative al progetto richiesto in argomento (client, bank, server, token).



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



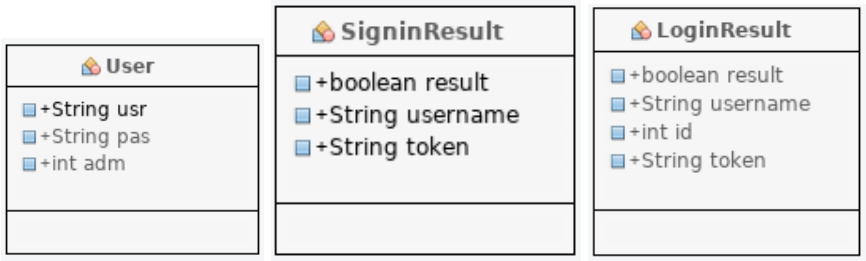
AES: contiene i metodi per crittografare i messaggi con AES



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.



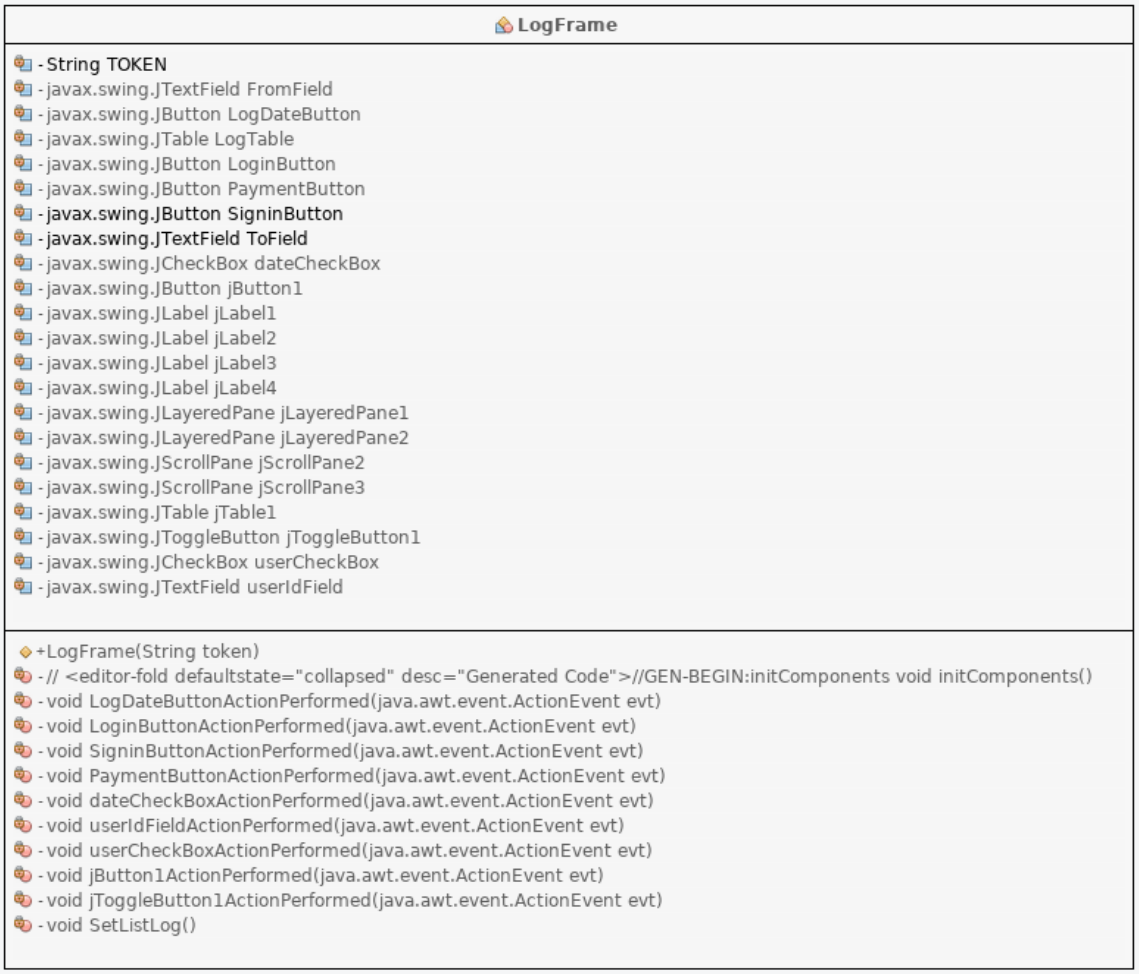

Card
-int code
-int month
-int year
-int cvc
+Card(int code, int month, int year, int cvc)
+int getCode()
+void setCode(int code)
+int getMonth()
+void setMonth(int month)
+int getYear()
+void setYear(int year)
+int getCvc()
+void setCvc(int cvc)
Counter
-String api
-int channelId
-int field
+String getApi()
+void setApi(String api)
+int getChannelId()
+void setChannelId(int channelId)
+int getField()
+void setField(int field)
AccessPoint
-String ssid
-float latitude
-float longitude
-String query
-String cap
-String city
-String state
-String civico
+String getSsid()
+void setSsid(String ssid)
+float getLatitude()
+void setLatitude(float latitude)
+float getLongitude()
+void setLongitude(float longitude)
+String getQuery()
+void setQuery(String query)
+String getCap()
+void setCap(String cap)
+String getCity()
+void setCity(String city)
+String getState()
+void setState(String state)
+String getCivico()
+void setCivico(String civico)
Coords
-float latitude
-float longitude
+float getLatitude()
+void setLatitude(float latitude)
+float getLongitude()
+void setLongitude(float longitude)

Client Log

GUI

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

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



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

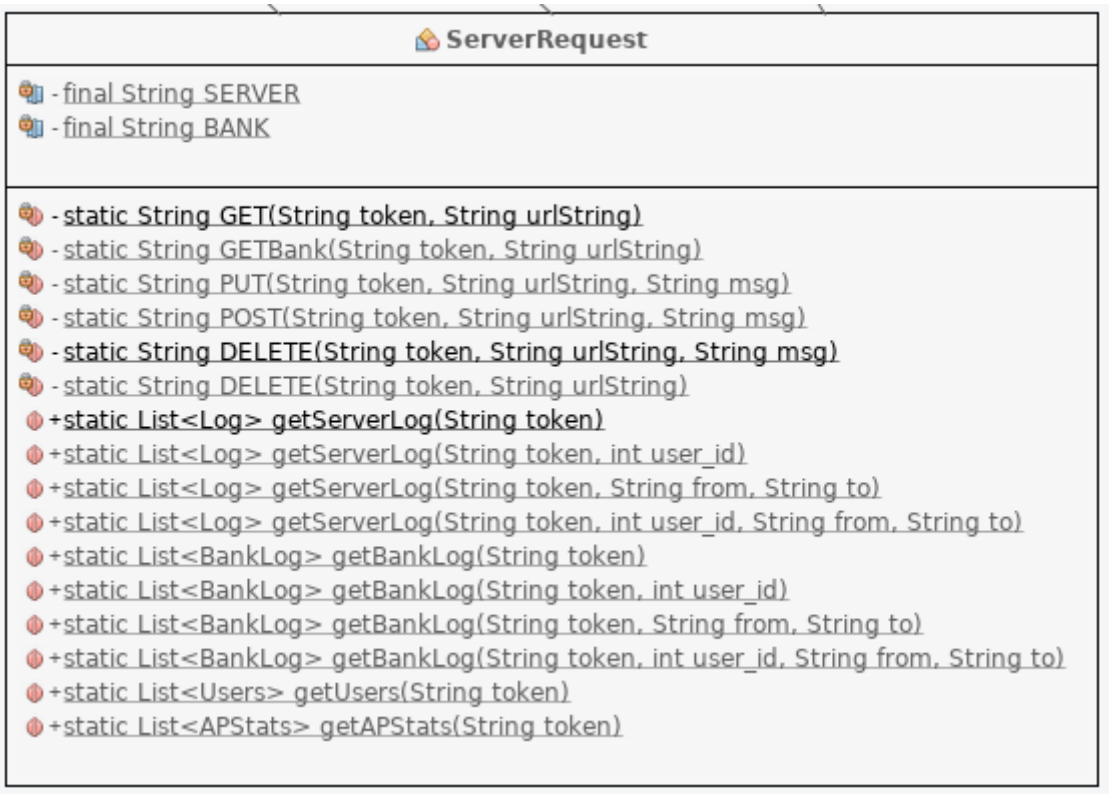
PieFrame: implementa l'interfaccia grafica per la visualizzazione del grafico a torta

NoThingsLoginFrame: mplementa il form per il login.

NoThingsSigninFrame: mplementa il form per la registrazione.

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



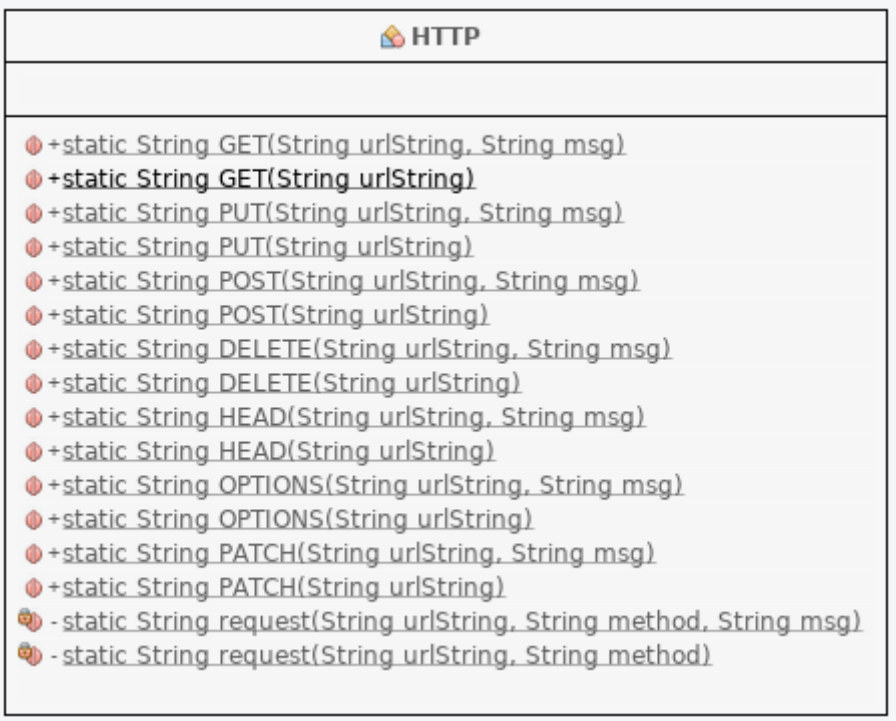
infoServer: implementa i metodi utili a raccogliere informazioni sugli indirizzi del server dai file di configurazione.

TokenServer

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

Request

HTTP: contiene i metodi per effettuare delle richieste http.

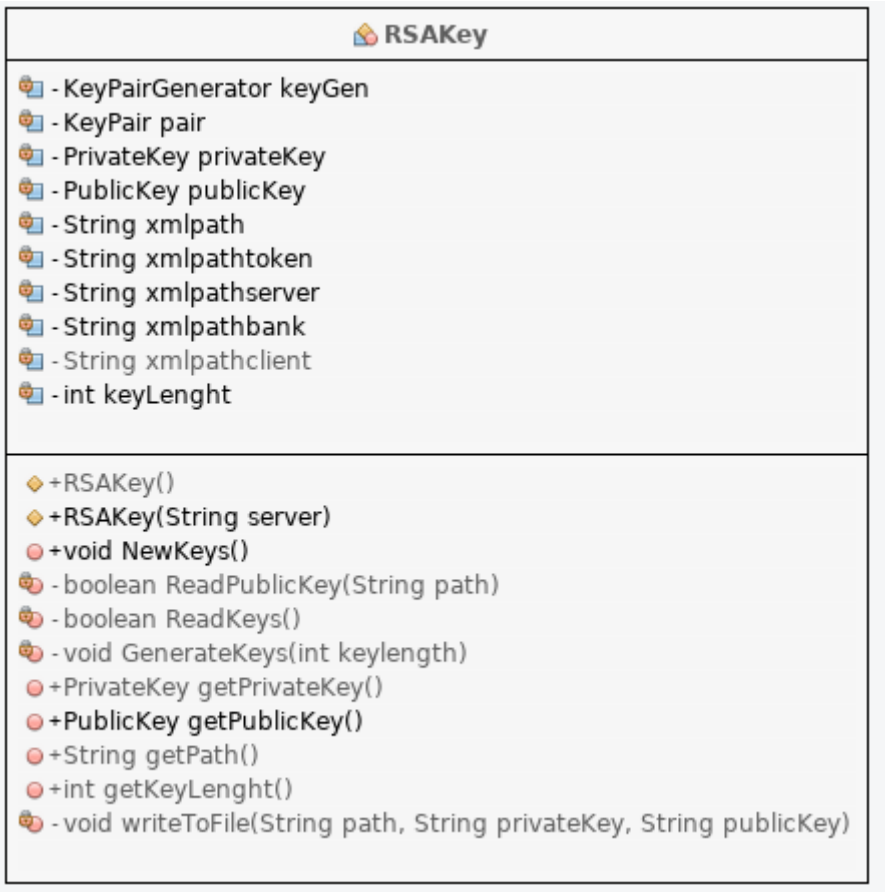


crittography

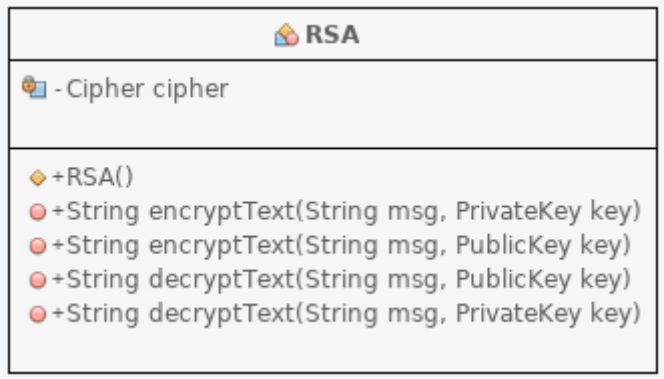
le classi in questo package servono per ciò che riguarda la crittografia.

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

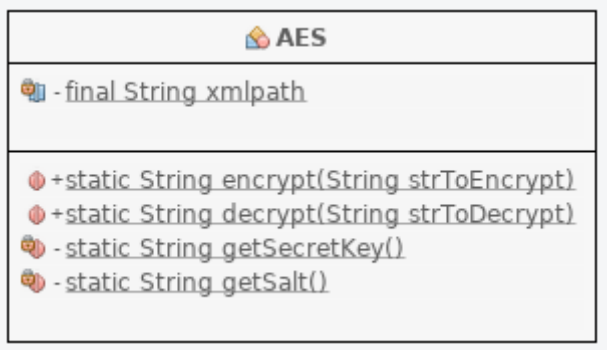
- RSAKey() inizializza le variabili private con le proprie chiavi e se non trova il proprio file di chiavi rsa ne genera di nuove
- RSAKey(String server) inizializza le variabili private con le chiavi relative al progetto richiesto in argomento (client, bank, server, token).



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

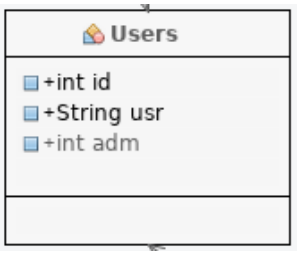


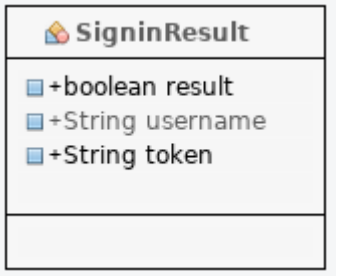
AES: contiene i metodi per crittografare i messaggi con AES

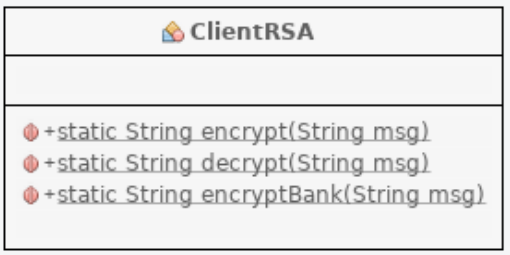


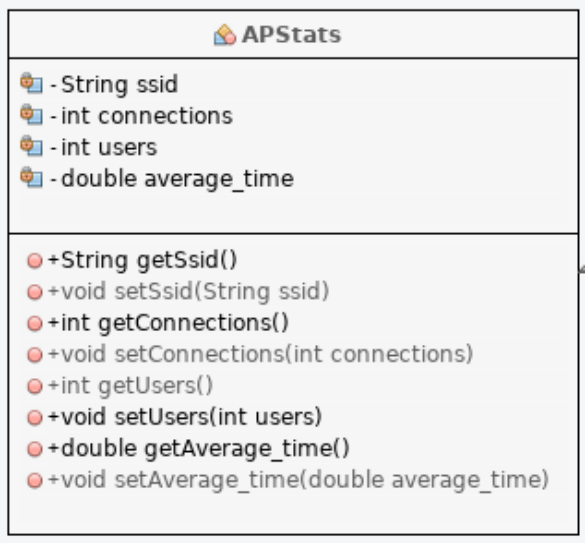
POJO

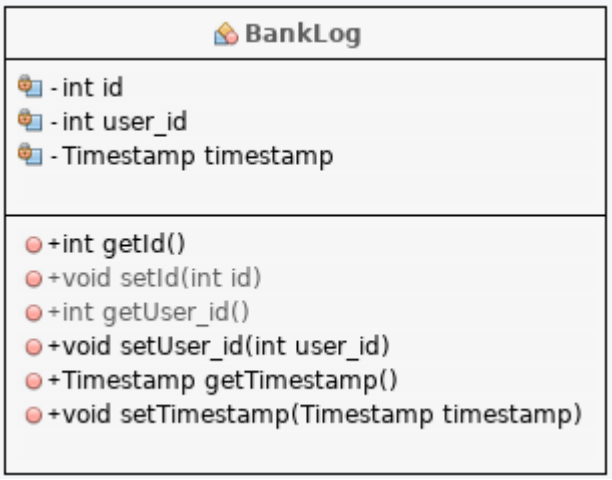
Queste Classi sono utilizzate solamente per rappresentare i dati inviati e/o ricevuti in formato json. Per definizione non sono legati ad alcuna restrizione diversa da quelle costrette dalla specifica del linguaggio Java.

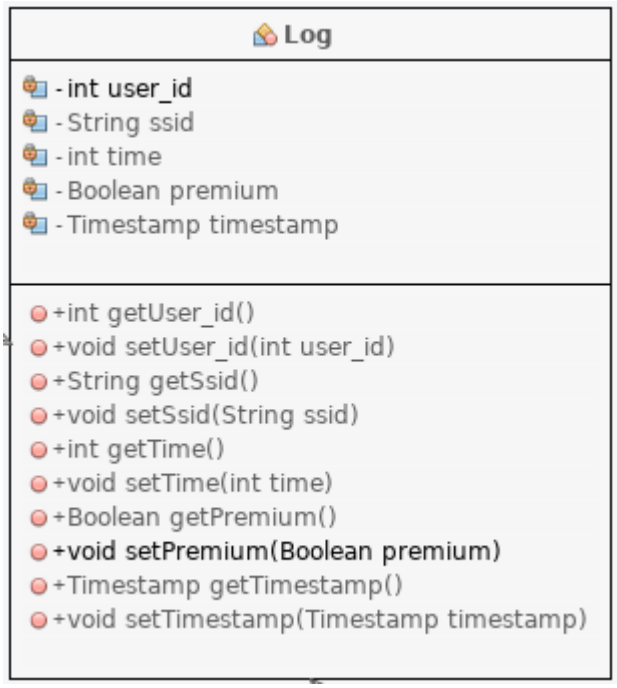


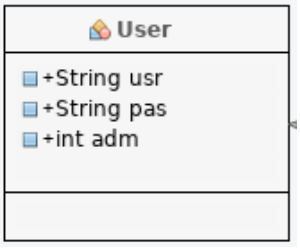












Database

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

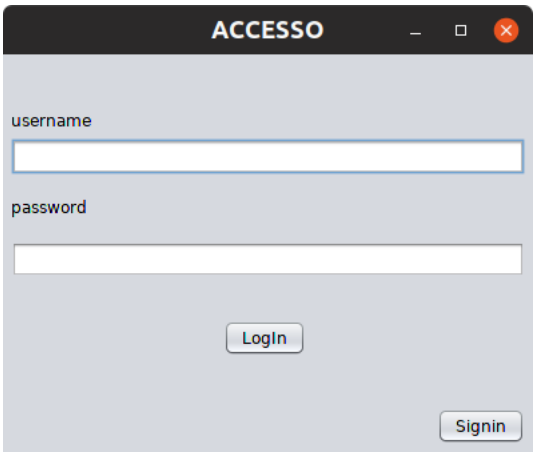


Figure 9: Schermata di login

Figure 10: Schermata di registrazione

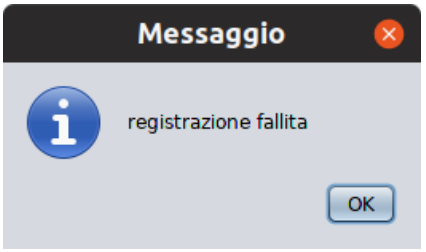


Figure 11: Messaggio di registrazione fallita

Mappa

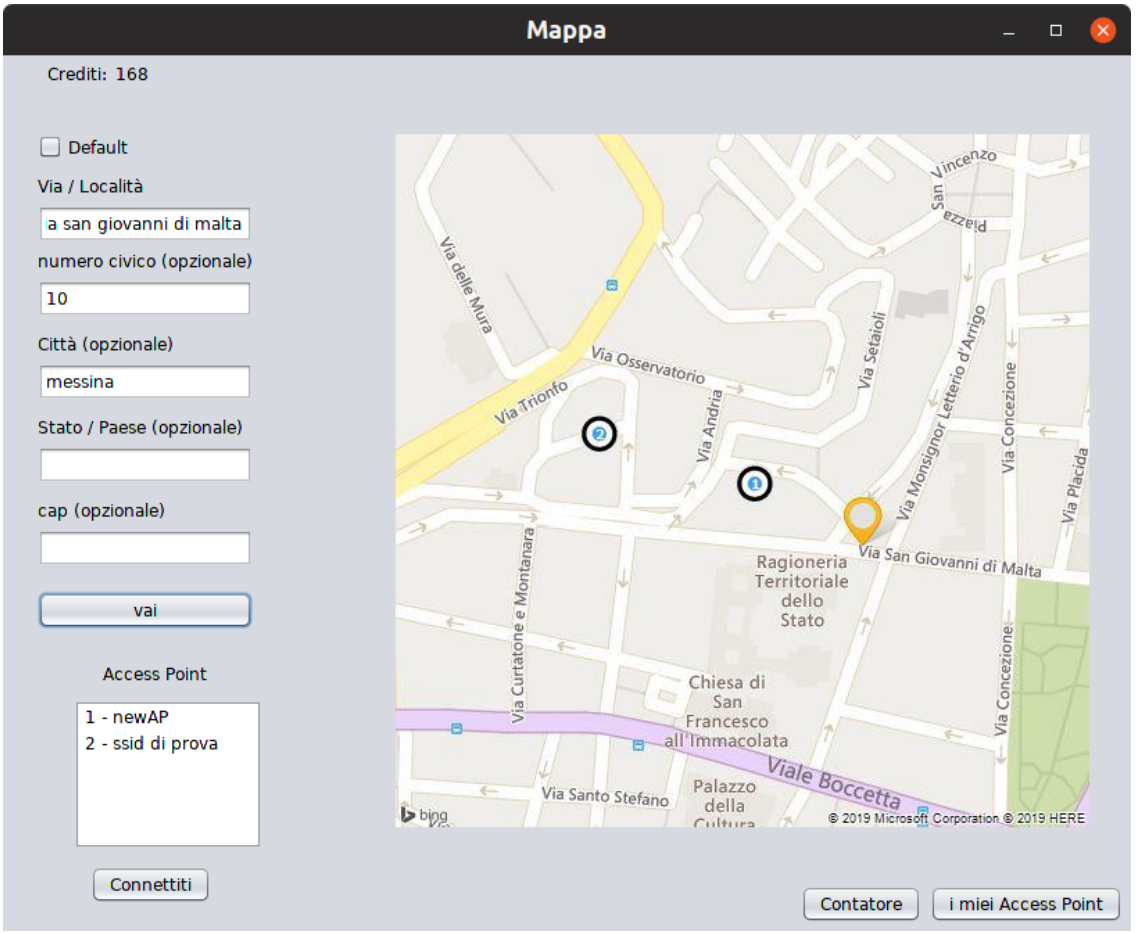


Figure 12: Schermata con la visualizzazione della mappa e degli AP

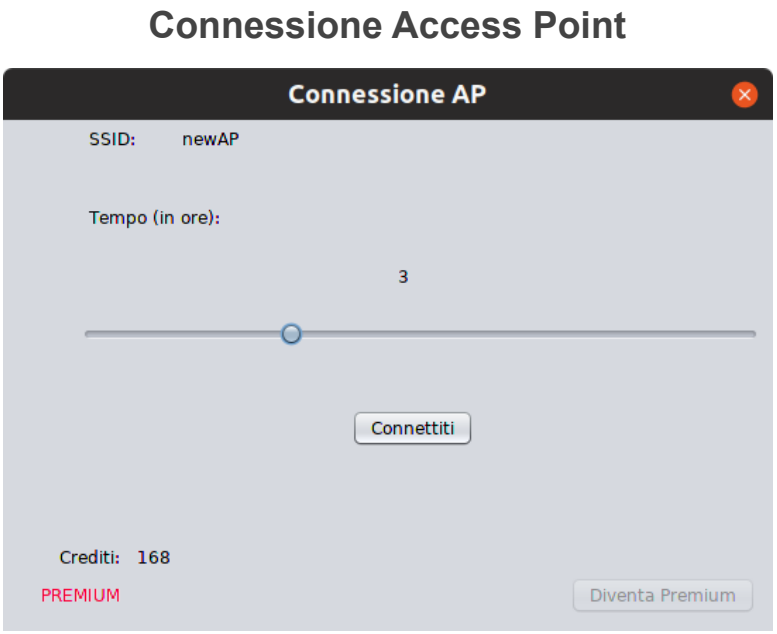


Figure 13: Schermata per la richiesta di connessione ad un AP

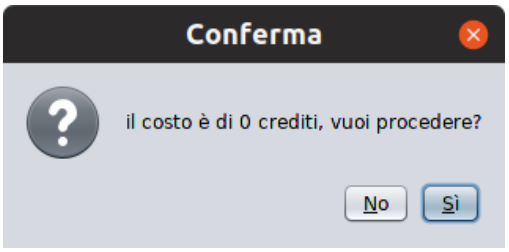


Figure 14: Messaggio di conferma connessione ad un AP

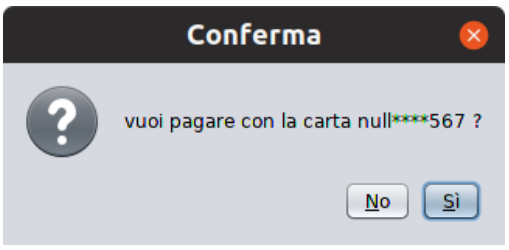


Figure 15: Messaggio di conferma pagamento

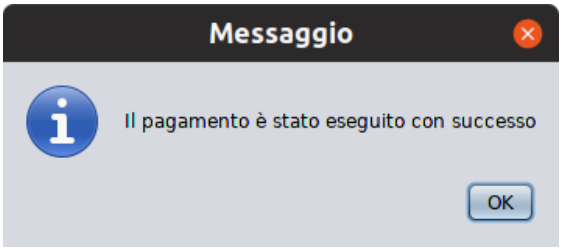


Figure 16: Messaggio di pagamento eseguito con successo

Figure 17: Messaggio di upgrade eseguito con successo

Gestione Contatore

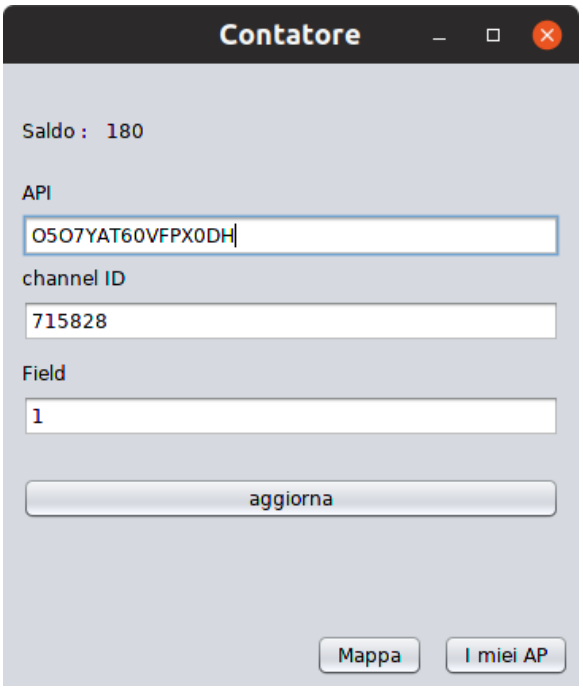


Figure 18: Schermata per la gestione del dispositivo contatore

Gestione Access Point

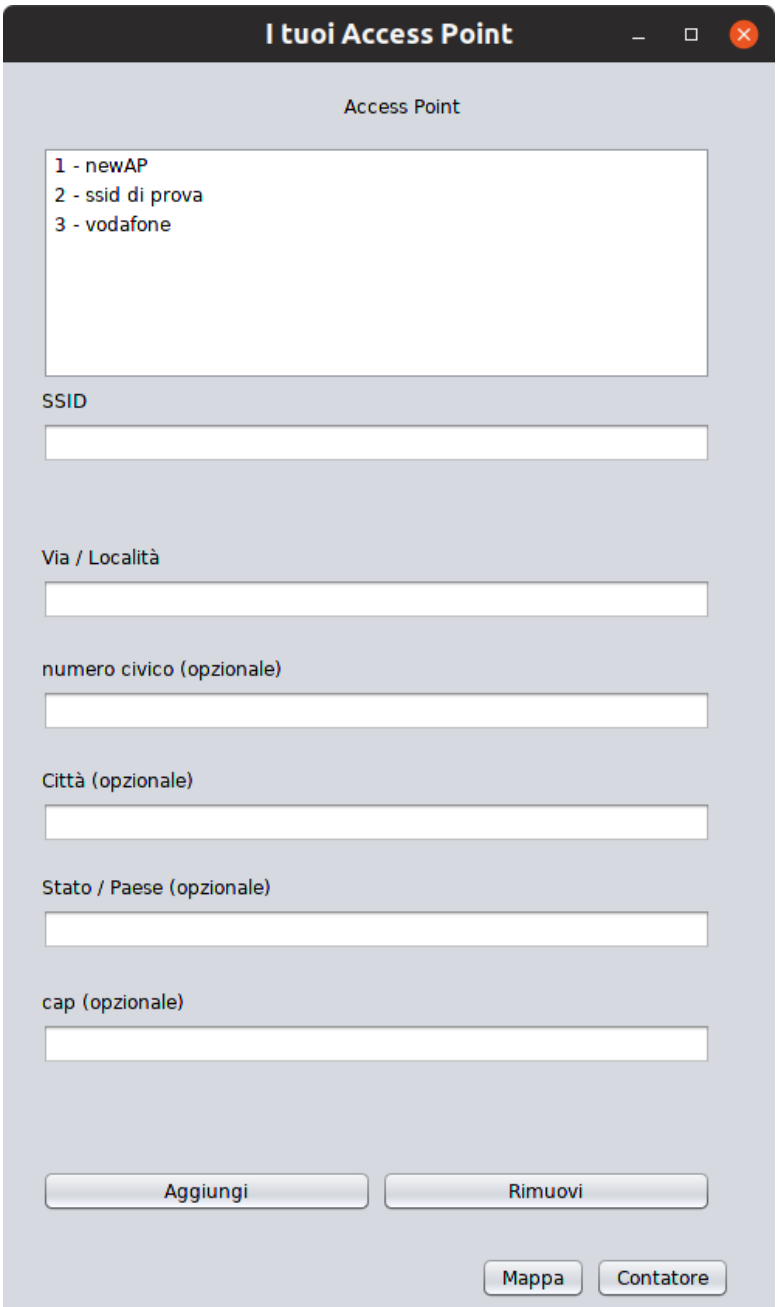


Figure 19: Schermata per la gestione dei propri AP

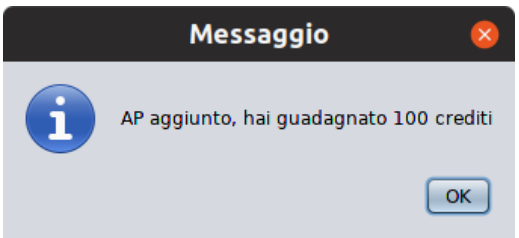


Figure 20: Messaggio successo aggiunta AP

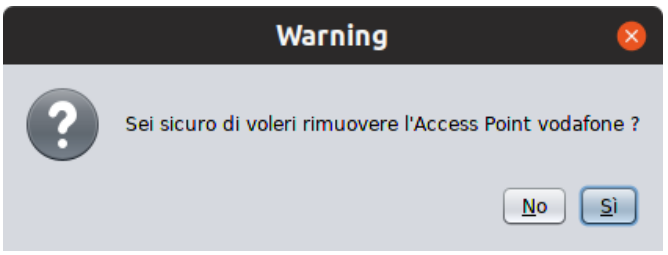


Figure 21: Messaggio di avviso

Registrazione carta di credito

Figure 22: Schermata per la registrazione della carta di credito

Figure 23: Messaggio di successo della registrazione della carta

Client log

Login e Registrazione

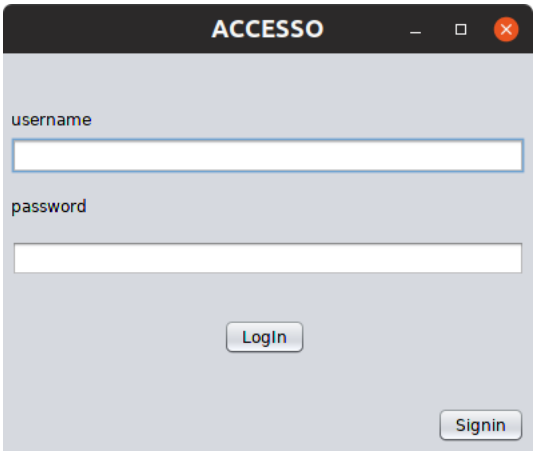


Figure 24: Schermata di login

Figure 25: Schermata di registrazione

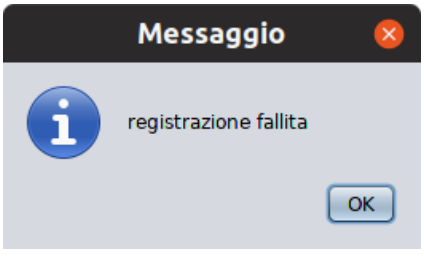


Figure 26: Messaggio di registrazione fallita

Log Connessioni

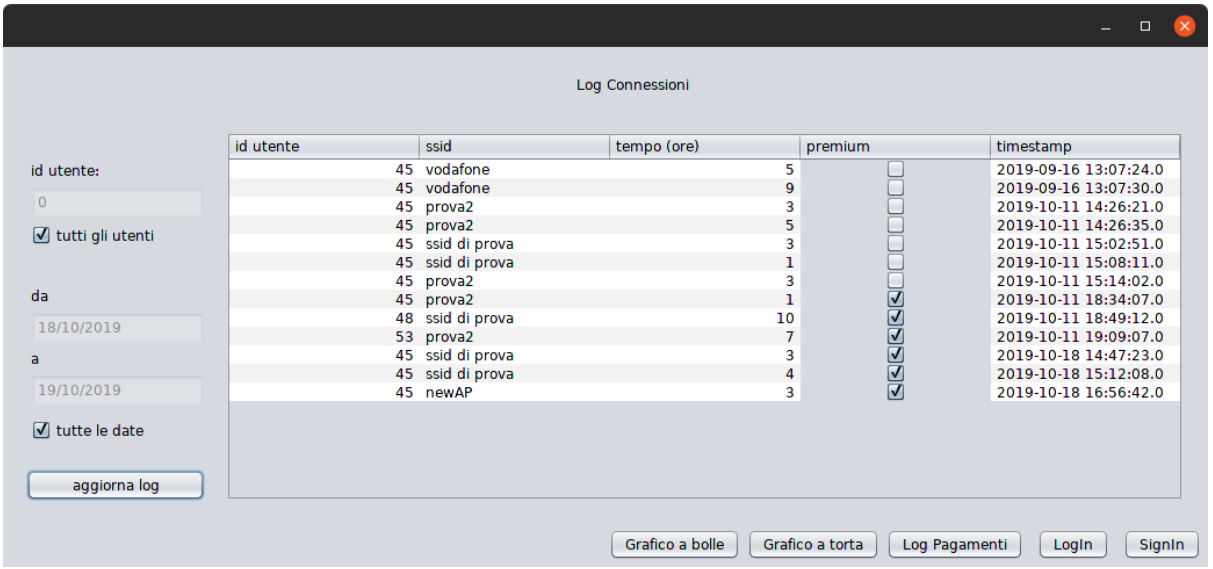


Figure 27: Schermata che mostra i log delle avvenute connessioni agli AP

Log Pagamenti

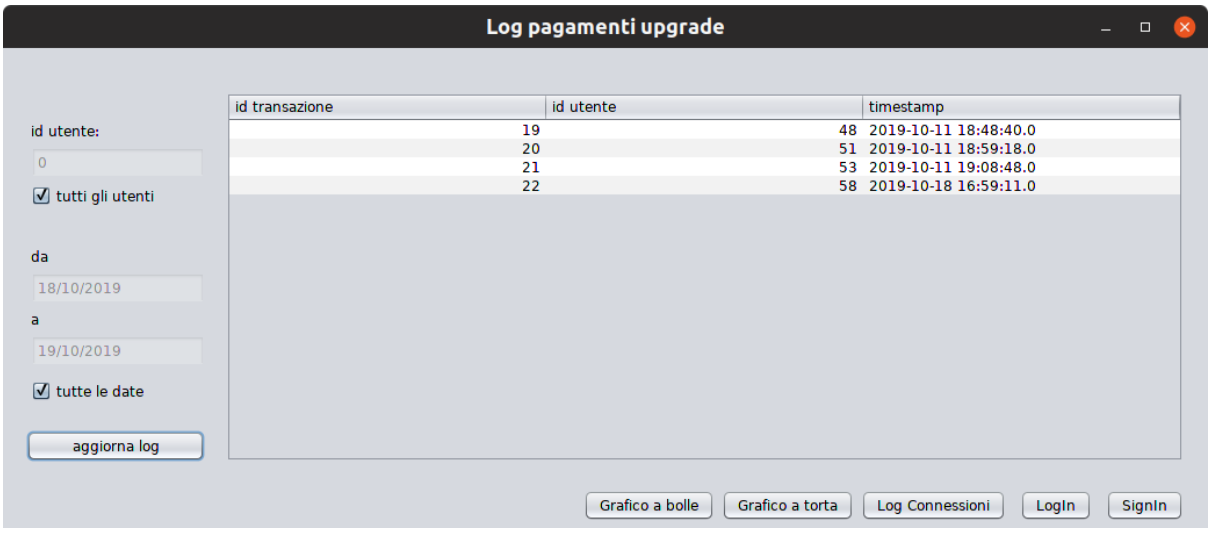


Figure 28: Schermata che mostra i log dei pagamenti degli utenti per effettuare l'upgrade a premium

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.

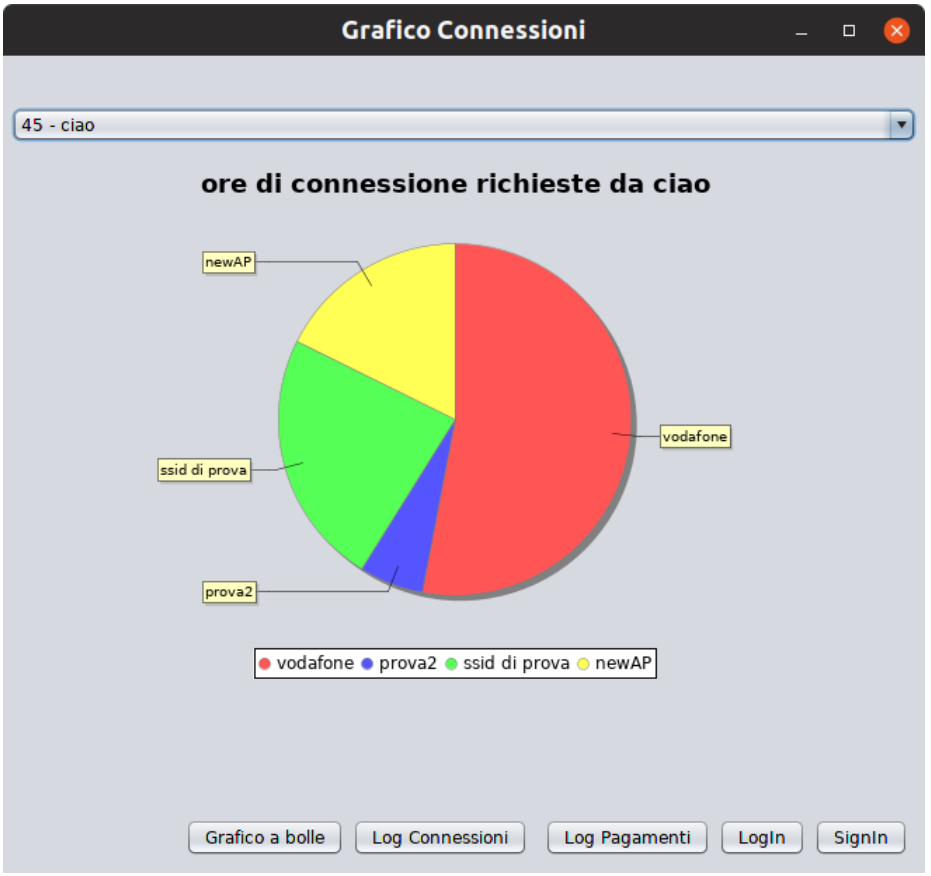


Figure 29: Schermata con grafico a torta con la rappresentazione delle ora di connessione richieste da un utente

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.

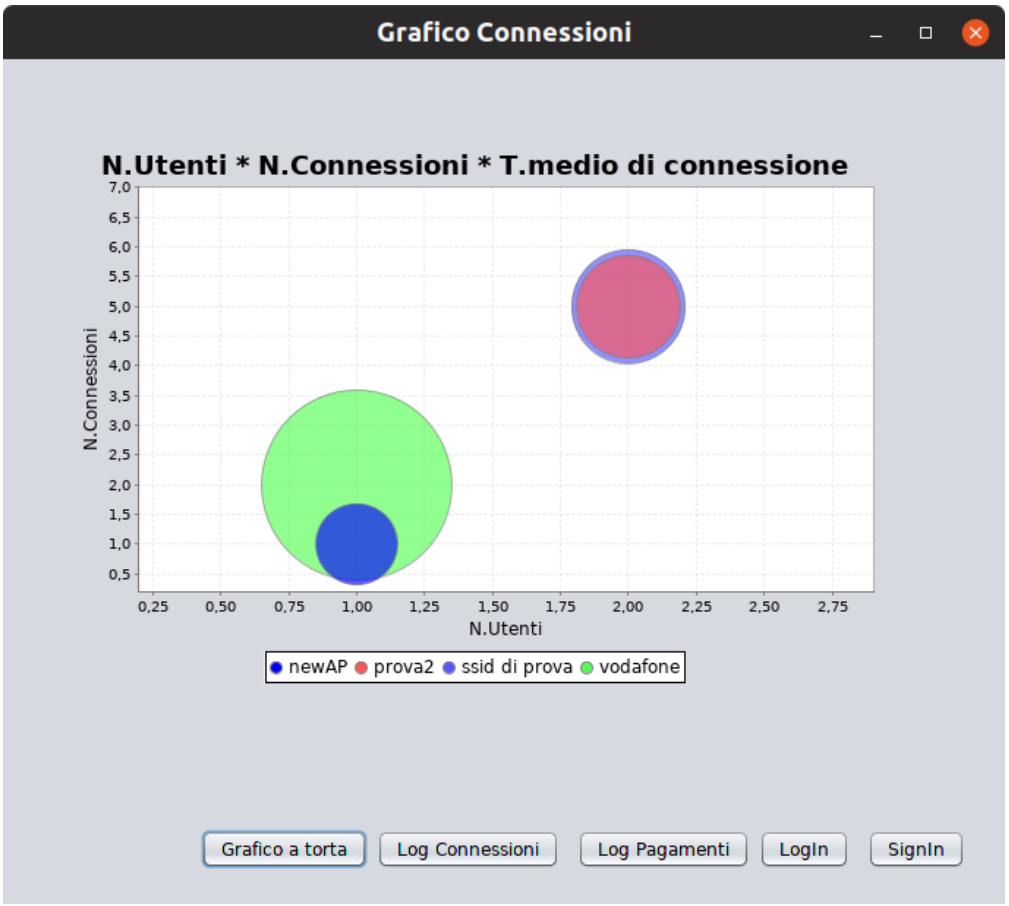


Figure 30: Schermata con il grafico a bolle che rappresenta l'utilizzo degli AccessPoint