



Corso di laurea triennale in Informatica

Progetto di Programmazione II

Applicazione Java Client-Server REST per la “perturbazione” di immagini con l’utilizzo del framework Spring Boot

Studente: Puliafito Marco

Docente: prof. Distefano Salvatore

ANNO ACCADEMICO 2020 -2021

Indice

1. Introduzione	<i>pag.6</i>
2. Client Utente.....	<i>pag.9</i>
2.1 Introduzione al client.....	<i>pag.9</i>
2.2 Login Utente.....	<i>pag.9</i>
2.3 Controllo utenza Premium.....	<i>pag.10</i>
2.4 GUI Principale.....	<i>pag.11</i>
3. Gestione richieste Client.....	<i>pag.12</i>
3.1 Introduzione al client.....	<i>pag.12</i>
3.2 Invio e ricezione immagine.....	<i>pag.13</i>
3.3 Altro.....	<i>pag.13</i>
4. Server HTTPS.....	<i>pag.14</i>
4.1 Cenni HTTPS.....	<i>pag.14</i>
4.2 Spring Boot.....	<i>pag.15</i>
4.3 Server HTTPS in questo progetto (introduzione).....	<i>pag.16</i>

4.4	<i>Gestione del login Facebook.....</i>	<i>pag.17</i>
4.5	<i>Controllo iniziale utenza Premium.....</i>	<i>pag.18</i>
4.6	<i>Gestione “diventa” Premium.....</i>	<i>pag.18</i>
4.7	<i>Elaborazione immagini.....</i>	<i>pag.20</i>
4.8	<i>Reperimento immagine filtrata.....</i>	<i>pag.21</i>
5.	<i>Applicazione Filtri.....</i>	<i>pag.22</i>
5.1	<i>Introduzione.....</i>	<i>pag.22</i>
5.2	<i>BufferedImage.....</i>	<i>pag.22</i>
5.3	<i>BlackandWhite.....</i>	<i>pag.23</i>
5.4	<i>Blue.....</i>	<i>pag.24</i>
5.5	<i>Sepia.....</i>	<i>pag.25</i>
5.6	<i>Red.....</i>	<i>pag.26</i>
5.7	<i>Gray.....</i>	<i>pag.27</i>
5.8	<i>Aggiungi Testo (filtro Premium).....</i>	<i>pag.28</i>
5.9	<i>Trova un volto (filtro Premium).....</i>	<i>pag.29</i>
	5.9.1 <i>Introduzione libreria OpenCV.....</i>	<i>pag.29</i>
	5.9.2 <i>Haar Cascade Classifier.....</i>	<i>pag.30</i>
	5.9.3 <i>Funzionamento OpenCV nel progetto.....</i>	<i>pag.30</i>

6. JWT (JSON WEB TOKEN).....	pag.32
6.1 Introduzione.....	pag.32
6.2 JWT nel progetto.....	pag.33
7. AES Crittografia simmetrica.....	pag.35
7.1 Cenni.....	pag.35
7.2 AES (Advanced Encrypt Standard) nel progetto.....	pag.35
8. Servizi di Terze Parti.....	pag.37
8.1 Login con API Facebook.....	pag.37
8.1.1 Introduzione.....	pag.37
8.1.1 Descrizione.....	pag.37
9. Servizi di Terza Parti (Cloud).....	pag.39
9.1 Cloud Storage.....	pag.39
9.1.1 Introduzione.....	pag.39
9.1.2 Google Drive.....	pag.40
9.1.2 Google Drive nel progetto.....	pag.41

9.2	Twilio API.....	pag.42
9.2.1	Introduzione.....	pag.42
9.1.2	API di verifica attraverso sms.....	pag.43
9.1.2	API di verifica nel progetto.....	pag.43
10.	Caratteristiche OOP.....	pag.46
10.1	Incapsulamento.....	pag.46
10.2	Information hidin.....	pag. 47
10.3	Polimorfismo.....	pag. 48
10.4	Astrazione.....	pag. 50
10.5	Ereditarietà.....	pag. 52
11.	Descrizione interfacce grafiche.....	pag.54
11.1	Client utenti.....	pag.54
11.2	Server.....	pag 68
12.	Diagrammi UML.....	pag 69
12.1	Class Diagram Client.....	pag 70
12.2	Class Diagram Server.....	pag 71

1. Introduzione

L'idea progettuale proposta è basata sulla costruzione di un'applicazione Java client-server REST, che comunica su HTTPS, dove vengono effettuate delle perturbazione di immagini inviate dal client ed elaborata sul server che risponderà con le immagine modificate come da richiesta. L'interfaccia dell'intera applicazione è una GUI sviluppata con la libreria Swing.

La prima GUI che verrà mostrata sarà quella dell'accesso utente, che attraverso un API del servizio di autenticazione di Facebook rilascerà un token d'accesso e preleverà le informazioni dell'utente tramite il suo account Facebook.

La seconda GUI sarà quella principale del software dove sarà possibile scegliere un immagine dal proprio OS, acquisita essa verrà mostrata al centro della GUI del software, ed a questo punto, l'utente avrà la possibilità di sfruttare l'interfaccia utente per manipolare l'immagine e di sfruttare altre funzionalità del software. Il client invia la richiesta utente "impacchettando" l'immagine sottoforma di byte e aggiungendo nel body di richiesta ulteriori informazioni utili come il filtro che si vuole applicare all'immagine originale. Il server si occuperà di prelevare i byte di immagine e di leggere la richiesta di filtro, successivamente applicherà il filtro all'immagine e risponderà al client con una risposta positiva se l'elaborazione è andata a buon fine, o con un messaggio di errore se l'elaborazione ha riscontrato degli errori, a questo punto il client consentirà all'utente di ottenere l'immagine appena filtrata e di visualizzarla al centro della GUI. Il software permetterà all'utente di avere un set completo di funzionalità che consentiranno ad esso il salvataggio dell'immagine sul proprio OS o in alternativa sarà possibile sfruttare la memorizzazione su GoogleDrive, servizio di Cloud Storage.

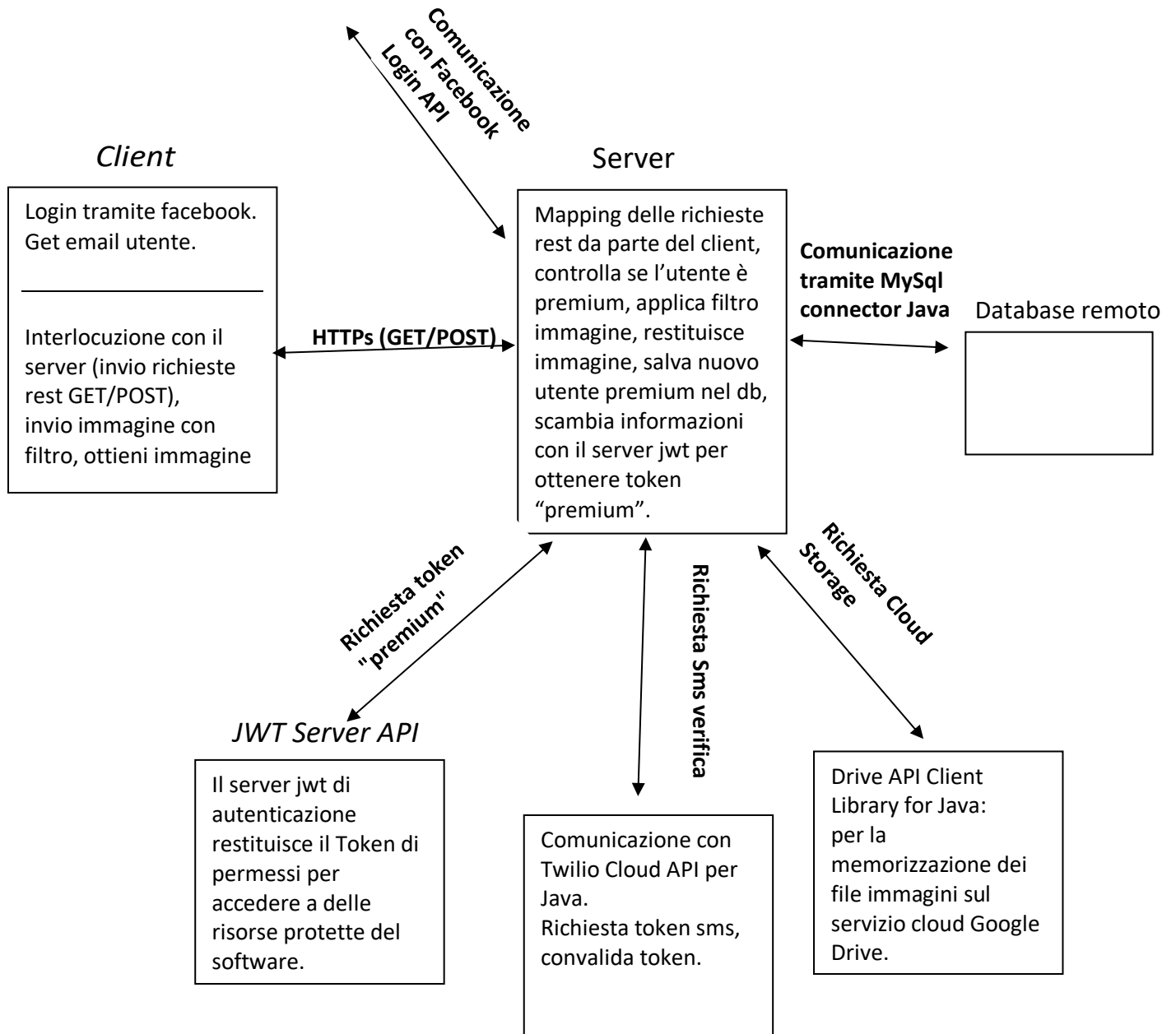
Il software suddividerà l'utente in due livelli di servizi distinte:

- *Utente comune: che potrà sfruttare le funzionalità base del software, in particolare l'utilizzo dei principali filtri e la funzionalità di salvataggio dell'immagine.*
- *Utente premium: che tramite codice di verifica inviato via sms, dà il consenso al programma di essere inserito nella lista di utenza premium e quindi di ricevere un token (jwt) che consentirà all'utente di avere il lascia passare per altre funzionalità del software.*

Ogni token (accesso, premium) inviato al Client verrà crittografato dal server tramite codifica AES256, il Client conserverà i token in dei file json, che se richiesti, verranno inviati al server e decodificati solo e soltanto dal server stesso.

Il progetto prevede inoltre la presenza di un database MySQL, accessibile e visibile soltanto dal server, nel quale verranno memorizzati gli utenti che hanno sottoscritto il servizio premium del software. Per l'interlocuzione con il database, è stato utilizzato il software XAMPP, considerando il framework JDBC Driver Java.

Nel servizio GoogleDrive verrà utilizzato il framework "Drive API Client Library for Java". Per la gestione dei token d'accesso per gli utenti premium verrà utilizzata la libreria JWT (json web token).



2. Client Utente

2.1 Introduzione al Client

Il Client si presenterà all'utente con un interfaccia grafica e comunicherà con il server tramite richieste HTTPs. Il client si connetterà al Server prelevando da file json le informazioni necessarie alla connessione (indirizzo ,porta), il file in questione sarà conservato nella directory del Client e si presenterà con il nome di "settings.json". Il client invierà le richiesta al server tramite libreria Java HttpsURLConnection.

2.2 Login Utente

All'avvio del programma l'utente di ritroverà davanti una GUI per accedere al software, il meccanismo di accesso avviene mediante API Facebook, al click del bottone d'accesso l'utente verrà reindirizzato al browser per dare il consenso dell'utilizzo dei propri dati utente al software, dopodichè potrà confermare il tutto sempre da bottone presente nella GUI. Se tutto è andato a buon fine il client riceverà dal Server un token facebook e l'email dell'utente loggato. Il token verrà conservato nella directory del client codificato con crittografia AES256, il tutto verrà salvato nel file json "data_access_user.json".

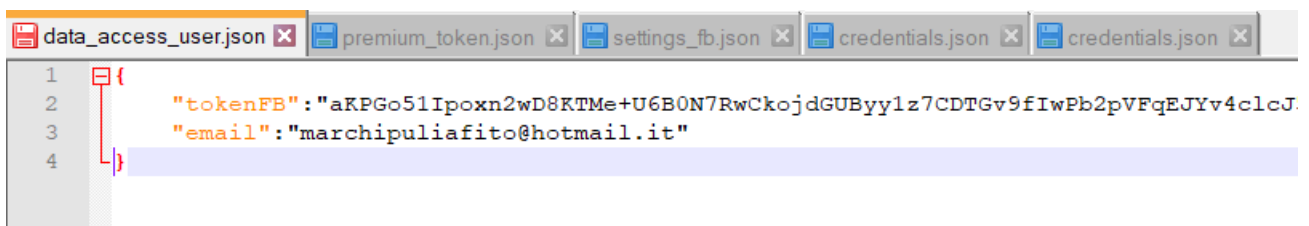
Di seguito uno screenshot della GUI di Login:



2.3 Controllo iniziale utenza premium

Appena l'utente viene loggato il Client invierà al Server una richiesta POST in cui verrà inviata l'email dell'utente appena loggato, il server HTTPs interrogherà il database MySql e cercherà la corrispondenza per email nella tabella "utenti(premium)", se trova una corrispondenza verrà inviata come risposta al Client un token "premium" che sarà il lascia passare per le funzionalità premium del software, questo token sarà incapsulato in un file json e codificato tramite crittografia AES256, il file in questione sarà conservato nella directory del Client e si presenterà con il nome di "premium_token.json".

Di seguito un esempio di file json contenente il token d'accesso Facebook e l'email utente loggato (token codificato):



```
1 {  
2   "tokenFB": "aKPGo51Ipoxn2wD8KTMe+U6B0N7RwCkojdGUByy1z7CDTGv9fIwPb2pVFqEJYv4clcJ  
3   "email": "marchipuliafито@hotmail.it"  
4 }
```

2.4 GUI principale

Dopo che l'utente ha effettuato l'accesso tramite facebook e sono stati fatti i controlli per quanto riguarda i privilegi dell'utente, si vedrà a schermo il pannello principale del software. Per prima cosa l'utente dovrà scegliere un'immagine da aggiungere al pannello, non sarà vincolato, ma se vorrà usare qualche funzionalità del software, gli verrà mostrato un avviso che ricorderà all'utente di dover caricare un'immagine. Inoltre nel pannello sarà visibile un'informazione che indicherà l'email utente con la presenza della parola "PREMIUM" se rientra in tale categoria.

Di seguito uno screenshot della GUI principale:



Nella GUI principale sarà possibile utilizzare il software in tutti i suoi aspetti.

Qui troviamo i pulsanti per salvare l'immagine e troviamo anche il cuore del software, cioè i pulsanti che invieranno la richiesta POST al server contenente nel body i byte d'immagine e nell'header il filtro che si vuole applicare.

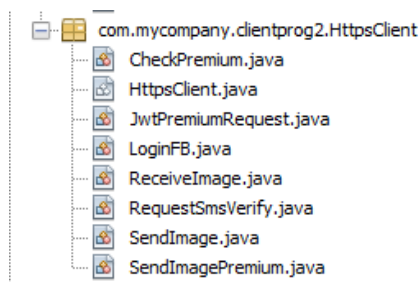
Nel capitolo sui “filtri immagine” verrà mostrato nel dettaglio l'elaborazione dei filtri.

3. Gestione richieste Client

3.1 Introduzione

La gestione delle richieste Client dirette al server HTTPS sono raccolte nel package “package com.mycompany.clientprog2.HttpsClient”, dove la classe HttpsClient.java è una classe astratta che serve da superclasse per le restanti classi che implementeranno i metodi “post_request()”, “get_request()”, “get_image()”. Ogni classe avrà come stato il path della risorsa che verrà richiesta al Server.

Di seguito uno screen del package HttpsClient sull'IDE Apache NetBeans:



3.2 Invio e ricezione immagine

L'invio dell'immagine da parte del Client avviene trasformando l'immagine, inizialmente sottoforma di `BufferedImage` in byte di immagine, e aggiungendo tutte le informazioni necessarie (filtro, token) per l'elaborazione da parte del Server. La gestione delle richieste Client per l'elaborazione delle immagini viene suddivisa in due classi, che principalmente si differenziano per la presenza del Token "Premium". Importante al tal fine è il metodo `elaboraInvioImmagine(HttpClient)` che avrà come argomento l'oggetto `HttpClient`, e chiamerà polimorficamente il metodo di invio immagine al Server in caso si tratti del sottotipo `"SendImage.java"` o del sottotipo `"SendImagePremium.java"`. Alla fine il Client riceverà come risposta una stringa, che sarà il nome del file temporaneo dell'immagine appena elaborata dal Server, cliccando "ottieni immagine filtrata", questa stringa, sarà inviata mediante richiesta POST, in modo tale che il Server saprà quale immagine mostrare al Client.

3.3 Altro

Le restanti classi dedicate alle richieste POST/GET grazie ad una corretta modellazione della programmazione ad oggetti hanno gli stessi metodi gestiti mediante overriding. Ognuna di esse avrà il proprio path per l'accesso alla risorsa RestFul e il proprio stato per l'invio dei dati al Server.

4. Server HTTPS

4.1 Cenni HTTPS

L'HyperText Transfer Protocol over Secure Socket Layer (HTTPS), (anche noto come HTTP over TLS, HTTP over SSL e HTTP Secure) è un protocollo per la comunicazione sicura attraverso una rete di computer utilizzato su Internet. La porta utilizzata generalmente (ma non necessariamente) è la 443. HTTPS consiste nella comunicazione tramite il protocollo HTTP (HyperText Transfer Protocol) all'interno di una connessione criptata dal Transport Layer Security (TLS) o dal suo predecessore, Secure Sockets Layer (SSL). Il principio che sta alla base di HTTPS è quello di avere:

- *Un'autenticazione del sito web visitato*
- *Protezione della privacy*
- *Integrità dei dati scambiati tra le parti comunicanti.*

In telecomunicazioni e informatica è il risultato dell'applicazione di un protocollo di crittografia asimmetrica al protocollo di trasferimento di ipertesti HTTP. Viene utilizzato per garantire trasferimenti riservati di dati nel web, in modo da impedire intercettazioni dei contenuti. Nel suo popolare funzionamento su internet, HTTPS fornisce l'autenticazione del sito web e del server web associato con cui una delle parti sta comunicando, proteggendo la comunicazione dagli attacchi noti tramite la tecnica del man in the middle. Inoltre, HTTPS fornisce una cifratura bidirezionale delle comunicazioni tra un client e un server, che protegge la stessa contro le possibili operazioni di eavesdropping, (azione mediante il quale viene ascoltata segretamente la conversazione privata tra le parti senza il loro consenso) e tampering (letteralmente manomissione o alterazione della comunicazione) falsificandone i contenuti. In pratica, tale meccanismo fornisce una garanzia soddisfacente del fatto che si sta comunicando esattamente con il sito web voluto (al contrario di un sito falso), oltre a garantire che i contenuti delle comunicazioni tra l'utente e il sito web non possano essere intercettate o alterate da terzi.

Storicamente, le connessioni HTTPS erano usate soprattutto per i pagamenti nelle transazioni sul World Wide Web, e-mail e per le transazioni sensibili all'interno dei sistemi informativi aziendali. Nel tardo 2000 e nei primi anni del 2010, HTTPS ha iniziato ad avere una larga diffusione e ampio utilizzo per proteggere l'autenticità delle pagine web, la sicurezza degli account utente e per mantenere private le comunicazioni, l'identità e la navigazione web dell'utente. [3]

Prima di avviare il Server HTTPS, bisogna caricare il certificato SSL in formato “.jks” in modo da abilitare il suddetto protocollo. Per stabilire una connessione sicura tramite SSL, è necessario che l'applicazione abbia una chiave di protezione, chiave che deve essere assegnata da un'Authority preposta che la rilascia sotto forma di certificato. Il file nel nostro è stato generato tramite prompt dei comandi e successivamente salvato nella directory /resources del Server.

4.2 Spring Boot

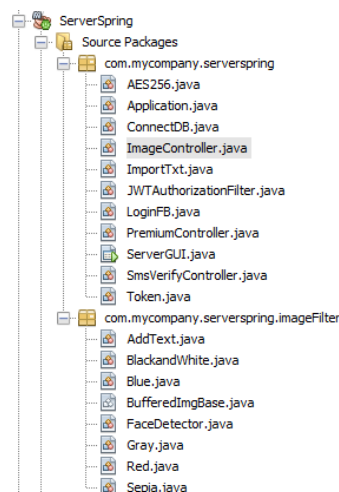
Per la gestione del servizio RestFul su HTTPS è stato utilizzato il framework Spring Boot. Spring Boot è un *framework* per lo sviluppo di applicazioni web basate su codice Java (precisamente su stack Java) che offre un ulteriore livello di astrazione rispetto all'utilizzo di Spring Framework. *Soprattutto in termini di setup e configurazione del progetto e di gestione delle dipendenze. In questo modo, si ottiene una riduzione ulteriore dei tempi necessari per l'implementazione e la distribuzione delle nostre applicazioni. Spring Boot permette che le app utilizzino la configurazione automatica, la scansione dei componenti e siano in grado di definire una configurazione aggiuntiva nella loro “classe di applicazione”. È possibile utilizzare un'unica annotazione **@SpringBootApplication** per abilitare queste tre funzionalità, ovvero:*

- **@EnableAutoConfiguration**: *abilita il meccanismo di configurazione automatica di Spring Boot.*
- **@ComponentScan**: *abilita la scansione @Component sul pacchetto in cui si trova l'applicazione.*
- **@Configurazione**: *consente di registrare bean extra nel contesto o importare classi di configurazione aggiuntive.*

4.3 Server HTTPS in questo progetto (introduzione)

Per quanto riguarda il Server in questo progetto, inizialmente verrà mostrata una GUI dove sarà possibile decidere se attivare il Server o disattivarlo, inoltre vi sarà una *TextArea* dove sarà possibile vedere l'immagine elaborata in attesa di essere restituita al Client. Le immagini in attesa verranno memorizzate in timestamp e conservate nella directory *"/resource/tmpImg"* del Server, appena verranno restituite al Client attraverso una richiesta GET il file *tmp* dell'immagine verrà eliminato dal Server. Il Server per connettersi reperirà dal file *"application.properties"* le informazioni utili alla sua connessione come l'indirizzo e la porta su cui restare in ascolto.

Di seguito lo screen del package completo Server HTTPS:



4.4 Gestione del Login Facebook

Per analizzare il funzionamento del login utente attraverso API Facebook, dobbiamo fare un passo indietro tornando alla GUI Login del Client, in quanto l'utente attiva il processo di autenticazione tramite facebook al click del bottone "Accedi con Facebook". Avviato il processo di autenticazione la ci ritroveremo sul web browser che ci chiederà l'accesso a Facebook, soltanto la prima volta dovremo autorizzare il software a poter interloquire con Facebook API, successivamente all'indirizzo e la porta del Server in ascolto sulla risorsa /loginFB otterremo il code dell'utente Facebook, che verrà impacchettato sottoforma di file json e salvato sul Server con il nome "codeFB.json". D'ora in avanti sulla GUI del Login troveremo un nuovo bottone di conferma d'accesso, esso non farà altro che effettuare una chiamata HTTPS al Server, che utilizzando il code impacchettato nel file json richiederà a Facebook API il Token e l'email utente, infine il nostro Server risponderà alla richiesta GET del client con un json che avrà come chiavi il Token e l'email utente. Adesso possiamo dire che l'utente ha effettuato l'accesso al software.

Di seguito riporto la tabella relativa alle richieste inviate al Server per quanto riguarda il processo di Login:

Richiesta	Risorsa	Successo	Errore
GET	/loginFB	Success message	Error message
GET	/getTokenFB	String Token, email	Error message

4.5 Controllo iniziale utenza Premium

In questa fase il software (in modo trasparente) invierà un richiesta, con l'email dell'utente appena loggato al Server, che comunicherà tramite MySQL API con il Database e verificherà se l'email utente si trova in un record della tabella "utenti(premium)", in caso di esito positivo (Utente Premium) il Server passerà alla creazione di un JWT Token che verrà consegnato al client sottoforma di json, esso verrà inviato codificato con crittografia simmetrica AES256, in caso di esito negativo (utente Base) il Server invierà al Client una risposta "null".

Di seguito riporto la tabella relativa alla richiesta inviata al Server per quanto riguarda i controlli per i diversi tipi di utenza:

Richiesta	Risorsa	Successo	Errore
POST	/ checkPremium	String JWT Premium	null

4.6 Gestione "diventa" Premium

L'attivazione di utente che vuole usufruire dei vantaggi Premium avviene attraverso un bottone "Diventa Premium" presente sulla GUI principale del Client, al click dovremo inserire un numero di cellulare che servirà da validatore per diventare Premium. Il Client invierà una richiesta POST al Server, contenente il numero di cellulare inserito, alla risorsa "/smsVerify", a questo punto il Server comunicherà attraverso API con il servizio Twilio che genererà un codice inviandolo al numero di cellulare prima inserito, se non

vi saranno errori il Server risponderà al Client con un valore booleano “true”, altrimenti risponderà con “false”.

Richiesta	Risorsa	Successo	Errore
POST	/ smsVerify	Boolean True	Boolean False

Lo step successivo del processo sarà quello di inserire il codice ricevuto dal servizio Twilio in un input text del Client, che si occuperà di inviarlo con una richiesta POST al Server, dove nel body verrà inserito il Codice sms inserito, il Server darà accesso alla risorse “/checkNumberToken”, successivamente il Server comunicherà sempre con il servizio Twilio che verificherà se il codice inserito corrispondente al numero di cellulare prima specificato è valido, se valido il nostro Server risponderà al Client con esito positivo, altrimenti risponderà in modo negativo.

Richiesta	Risorsa	Successo	Errore
POST	/ checkNumberToken	Boolean True	Boolean False

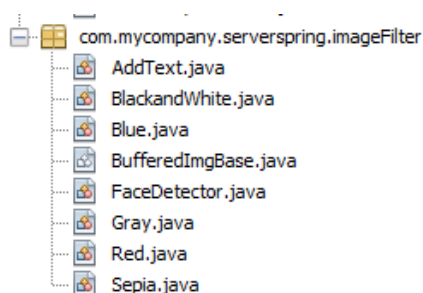
Infine se il codice è stato convalidato l’utente diventerà ufficialmente “premium”, ultimo step sarà quello di inviare nuovamente una richiesta POST al server contattando la risorsa “/getTokenPremium”, ed inserendo nel body l’email utente, in modo tale che il nostro Server scriva tramite API JDBC MySql sul nostro Database nella tabella “utenti(premium)” un nuovo record che indicherà il nuovo utente Premium del software. Se non ci sono stati problemi il Server creerà un JWT Token “Premium” che verrà codificato e inviato come risposta al Client, esso sarà il lascia passare per le successive richieste alle

risorse protette del Server.

Richiesta	Risorsa	Successo	Errore
POST	/ getTokenPremium	String JWT Premium	null

4.7 Elaborazione Immagine

Arriviamo alla parte principale del software, essa è organizzata in un package dedicato che prenderà il nome della classe astratta di riferimento “.imageFilter” , le classi figlie che implementeranno il metodo per applicare il filtro prenderanno il nome dei filtri stessi. Inizialmente il Server riceverà una richiesta POST contenente nel body i byte dell’immagine originale, inoltre nell’header verrà aggiunto il filtro scelto, ed altre informazioni come il Token “Premium” se richiesto. Il Server gestirà l’elaborazione delle immagini sulla risorsa “/filterImage” nel caso di filtri base e nella risorsa “/filterImagePremium” nel caso di filtri dedicati agli utenti Premium. Ricevute le informazioni la classe dedicata si occuperà alla gestione polimorfica del metodo “applicaFiltro()” , in base alla Stringa di filtro la superclasse “BufferedImgBase” assumerà la forma delle sue sottoClassi. Di seguito il package con la classe astratta e le classi che implementeranno essa:



Le classi che si occuperanno dell'applicazione dei filtri restituiranno l'immagine sempre sottoforma di byte, a fine processo se tutto è andato a buon fine il Server conserverà l'immagine in un file temporaneo nella propria directory “/resources/tmplmg” e risponderà al Client con il nome temporaneo (timestamp) dell'immagine filtrata. Successivamente il Client mostrerà una “notifica” che permetterà la visualizzazione nella GUI della nuova immagine.

Di seguito riporto la tabella relativa alla richiesta inviata al Server per quanto riguarda l'elaborazione dell'immagine:

Richiesta	Risorsa	Successo	Errore
POST	/ filterImage	String tmpfilename	null
POST	/ filterImagePremium	String tmpfilename	null

4.8 Reperimento immagine filtrata

Quando il Server finisce la sua elaborazione applicando il filtro sull'immagine originale, salverà in una directory dedicata il file d'immagine con un valore di timestamp, esso resterà in attesa di poter essere “chiamato” dal Client, il nome del file può essere letto anche nella GUI del Server. Per ottenere l'immagine l'utente al click del bottone “Ottieni immagine filtrata” invierà una richiesta POST al Server che trasporterà il timestamp del file creato cosichè all'arrivo il Server leggerà il parametrò e restituirà l'immagine presente nella directory “/resources/tmplmg” corrispondente, infine il Server in caso di esito positivo risponderà al Client con i byte della “nuova” immagine.

Successivamente il Server “pulirà” la directory cancellando l’ultima Immagine richiesta.

Di seguito riporto la tabella relativa alla richiesta inviata al Server per quanto riguarda il reperimento della “nuova” immagine:

Richiesta	Risorsa	Successo	Errore
POST	/ getImage	Byte[] img	null

5. Applicazione Filtri

5.1 Introduzione

Come già spiegato al paragrafo 4.7 il sistema di applicazione filtri funziona in maniera scalabile in quanto l’oggetto di tipo “BufferedImage” prende la forma dei suoi sottotipi e quindi utilizza il metodo “applyFilter()” polimorficamente. Nei prossimi paragrafi elencheremo le classi dei filtri disponibili.

5.2 BufferedImage

“BufferedImage.java” sarà la classe astratta che dichiara le caratteristiche comuni delle classi “Filtri”, per prima cosa essa implementa nel proprio costruttore la ricezione dei byte immagine e la trasformazione di essi in BufferedImage, inoltre avrà un metodo public “saveImg” che si occuperà del salvataggio della nuova immagine nella directory del Server, infine dichiara un metodo astratto “applyFilter()”

che sarà implementato nelle sottoclassi in base al filtro da dover applicare. Esso restituirà l'immagine sottoforma di byte.

Di seguito l'immagine in forma "base":



5.3 BlackandWhite

"BlackandWhite.java" è sottoclasse di BufferedImage in quanto estende il suo stato ed il suo comportamento, essa implementa in overriding il metodo "applicaFiltro()" ereditando l'immagine sottoforma di BufferedImage dalla superclasse "BufferedImage". In generale essa prenderà l'immagine originale e la trasformerà in un'immagine in bianco e nero, dopodiché la riconsegnerà in byte.

Di seguito l'immagine in forma "BlackandWhite":



5.4 Blue

“Blue.java” è sottoclasse di `BufferedImageBase` in quanto estende il suo stato ed il suo comportamento, essa implementa in overriding il metodo “`applicaFiltro()`” ereditando l’immagine sottoforma di `BufferedImage` dalla superclasse “`BufferedImageBase`”. In questo processo il metodo “`applicaFiltro()`” utilizzerà i metodi di `BufferedImage` `getRGB()` e `setRGB` per trasformare l’immagine in blu, dopodichè essa verrà riconsegnata in byte.

Di seguito l’immagine in forma “Blue”:



5.5 Sepia

“Sepia.java” è sottoclasse di BufferedImageBase in quanto estende il suo stato ed il suo comportamento, essa implementa in overriding il metodo “applicaFiltro()” ereditando l’immagine sottoforma di BufferedImage dalla classe astratta “BufferedImageBase”. Per l’applicazione di questo filtro si trasforma il BufferedImage in un intero di pixel, ogni pixel viene convertito in RGB e scorrendo ogni pixel si farà la conversione dell’immagine originale in immagine con il filtro “sepia” applicato, infine il BufferedImage dell’immagine convertita sarà trattato e trasformato in byte e restituito alla sorgente.

Di seguito l’immagine in forma “Sepia”:



5.6 Red

“Red.java” è sottoclasse di BufferedImageBase in quanto estende il suo stato ed il suo comportamento, essa implementa in overriding il metodo “applicaFiltro()” ereditando l’immagine sottoforma di BufferedImage dalla classe astratta “BufferedImageBase”. In questo processo il metodo “applicaFiltro()” utilizzerà i metodi di BufferedImage getRGB() e setRGB per trasformare l’immagine in rosso, infine il BufferedImage dell’immagine convertita sarà trattato e trasformato in byte e restituito alla sorgente.

Di seguito l’immagine in forma “Red”:



5.7 Gray

“Gray.java” è sottoclasse di BufferedImage in quanto estende il suo stato ed il suo comportamento, essa implementa in overriding il metodo “applicaFiltro()” ereditando l’immagine sottoforma di BufferedImage dalla classe astratta “BufferedImageBase”. In questo processo il metodo “applicaFiltro()” utilizzerà i metodi del package image, in particolare utilizzerà il metodo ColorConvertOp() che creerà un oggetto ed utilizzerà il metodo filter passando il BufferedImage dell’immagine originale, infine il BufferedImage dell’immagine convertita sarà trattato e trasformato in byte e restituito alla sorgente.

Di seguito l’immagine in forma “Gray”:



5.8 Aggiungi Testo (Filtro Premium)

“AddText.java” è sottoclasse di BufferedImageBase in quanto estende il suo stato ed il suo comportamento, essa implementa in overriding il metodo “applicaFiltro()” ereditando l’immagine sottoforma di BufferedImage dalla classe astratta “BufferedImageBase”. Questo filtro rientra nei filtri speciali che possono essere utilizzati soltanto dagli utenti con permessi “Premium”, la classe che contiene la gestione di questo filtro avrà nel suo costruttore una proprietà chiamata “addText” che sarà la stringa di testo inviata dal Client da inserire a sua volta nell’immagine da modificare. In questo processo viene importato “java.awt.Graphics” che servirà per la modellazione del BufferedImage originale; settato il font e settata la posizione della stringa di testo sull’immagine essa verrà convertita in byte e restituita alla sorgente.

Di seguito l’immagine con un esempio di testo aggiunto:



5.9 Trova un volto (Filtro Premium)

“FaceDetector.java” è sottoclasse di BufferedImgBase in quanto estende il suo stato ed il suo comportamento, essa implementa in overriding il metodo “applicaFiltro()” ereditando l’immagine sottoforma di BufferedImage dalla classe astratta “BufferedImgBase”. Questo filtro rientra nei filtri speciali che possono essere utilizzati soltanto dagli utenti con permessi “Premium”. In generale inserita un immagine esso controlla se è presente un volto, se non è presente non segnala nulla, invece se trova un volto nell’immagine segnalerà esso con delle linee in rosso. Questa funzionalità utilizza la libreria OpenCV.

5.9.1 Introduzione Libreria OpenCV

OpenCV (Open Source Computer Vision Library) è una libreria di software di visione artificiale e apprendimento automatico open source. È stato costruito per vari scopi come l'apprendimento automatico, la visione artificiale, l'algoritmo, le operazioni matematiche, l'acquisizione di video, l'elaborazione di immagini ecc. Nel corso degli anni è diventato molto popolare tra i ricercatori e gli sviluppatori per il suo supporto in diverse piattaforme (Windows, Linux , android, ios). Inoltre ha wrapper in vari rinomati linguaggi di programmazione. Passiamo, nel prossimo paragrafo, alla descrizione di come avviene il riconoscimento facciale.

5.9.2 Haar Cascade Classifier

Un classificatore è un programma che cerca di inserire una nuova osservazione in un gruppo dipendente dall'esperienza passata. I classificatori a cascata cercano di farlo utilizzando una concatenazione di diversi classificatori. Ogni classificatore successivo utilizza l'output del precedente come informazioni aggiuntive, migliorando notevolmente la classificazione. In generale, il classificatore a cascata deve essere pre-addestrato per essere in grado di rilevare qualsiasi cosa. Poiché il processo di formazione può essere lungo e richiederebbe un grande set di dati, utilizzeremo uno dei modelli pre-addestrati offerti da OpenCV.

Troveremo questo file XML nella nostra directory sul Server “/resources”, esso sarà contrassegnato con il nome “haarcascade_frontalface_alt.xml”.

5.9.3 Funzionamento OpenCV nel progetto

Per iniziare ad utilizzare OpenCV dobbiamo inizializzare la libreria con il comando “OpenCV.loadShared()”, OpenCV è una classe che contiene metodi relativi al caricamento di pacchetti nativi richiesti dalla libreria OpenCV per varie piattaforme e architetture. OpenCV legge l'immagine sottoforma di oggetto Mat, in particolare avendo inizialmente l'immagine originale in BufferedImage, dobbiamo trasformarla come già detto in oggetto Mat per essere processata con OpenCV, quindi dichiareremo un oggetto MatofRect per memorizzare le facce che troviamo, successivamente dobbiamo inizializzare CascadeClassifier per eseguire il riconoscimento, fatto ciò con un loop passeremo il risultato ed infine trasformeremo l'oggetto iniziale Mat in un BufferedImage in modo tale che possa essere riconvertito in byte e restituito alla sorgente di richiesta.

Di seguito l'immagine con un esempio di riconoscimento facciale:

Immagine originale:



Immagine dopo aver trovato un volto:



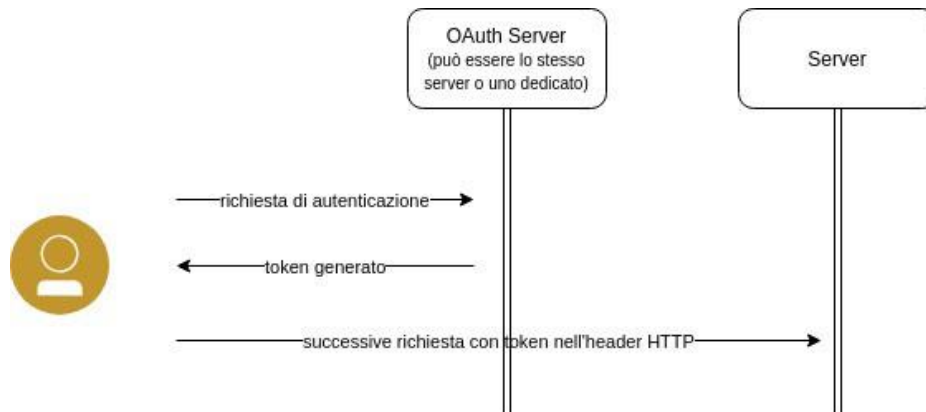
6. JWT (Json Web Token)

6.1 Introduzione

JWT è uno standard che viene usato per “regolare” le richieste tra due parti. JSON Web Token è un security token (una stringa) che agisce come un container per le claims degli user. I claims sono informazioni di un utente: chi è l’utente loggato (e altre info di contorno: ad es mail, nome, cognome), scadenza di un token (quindi scadenza dell’autenticazione dell’utente), privilegi dell’utente (è un admin o un normale utente) ed altre info customizzabili a seconda del contesto applicativo. I JWT sono molto utilizzati per autenticare le richieste nei Web Services e nei meccanismi di autenticazione OAuth dove il client invia una richiesta di autenticazione al server, il server genera un token firmato e lo restituisce al client che, da quel momento in poi, lo utilizzerà per autenticare le successive richieste. I token sono composti da tre parti: header, payload e chiave segreta:

- *Header* : Se decodifichiamo la prima parte del token scopriamo che si riferisce ad un json che contiene le due voci “typ” e “alg”. Il primo ha come valore sempre “JWT”, mentre il nodo “alg” contiene il nome dell’algoritmo usato per il token.
- *Payload* : La seconda parte del token è il corpo vero e proprio che contiene dei dati “variabili” in base al contesto in formato json e codificati in base64.
- *Signature (chiave segreta)*: La terza parte di un token è la firma che non è altro che il risultato di una funzione hash 256 che prende in input la codifica base64 dell’header concatenandola con un punto alla codifica base64 del payload, il tutto codificato con la nostra “chiave segreta” che solo il server conosce.

Immagine di esempio generico di comunicazione HTTP con Token incapsulato nell'header di richiesta:



6.2 JWT nel progetto

La gestione di JWT in questa applicazione è avvenuta le potenzialità del framework Spring, sono state aggiunte al progetto le dipendenze di Spring security e di jjwt API per Java.

La classe che conterrà al suo interno lo stato e i metodi del token prende il nome di "JWTPremium.java", essa inoltre conterrà al suo interno il metodo "secret_key_jwt()" che preleverà nel file json (salvato sul Server: "settings.json") la chiave segreta, che verrà utilizzata per firmare il jwt e renderlo unico, la chiave segreta verrà conservata soltanto lato server in quanto elemento fondamentale in termini di sicurezza. Come già detto il comportamento del token viene racchiuso nella classe "JWTPremium.java", qui utilizzeremo il metodo setToken(...) per costruire il token, delegando il Jwts nella classe utility che include le informazioni sulla sua scadenza e un oggetto Spring GrantedAuthority che, come vedremo più avanti, verrà utilizzato per autorizzare le richieste alle risorse protette.

Di seguito uno screen del metodo “setToken()” per la generazione del Token nel progetto, come si può notare è stato utilizzato come algoritmo di codifica “HS512” ed è stata aggiunta nel payload del Token l’email utente personale:

```
public void setToken(String email){
    //chiamo metodo per reperire chiave segreta da file json
    String secretKey = secret_key_jwt();
    List<GrantedAuthority> grantedAuthorities = AuthorityUtils
        .commaSeparatedStringToAuthorityList("ROLE_USER");
    this.token = Jwts
        .builder()
        .setSubject(sub:email)
        .claim(name:"authorities",
            value:grantedAuthorities.stream()
                .map(GrantedAuthority::getAuthority)
                .collect(Collectors.toList()))
        .setIssuedAt(new Date(1: System.currentTimeMillis()))
        .setExpiration(new Date(System.currentTimeMillis() + 6000000))
        .signWith(alg: SignatureAlgorithm.HS512,
            secretKey: secretKey.getBytes()).compact();
}
```

Grazie a Spring possiamo utilizzare delle annotazioni all’avvio del nostro Server che ci consentono di specificare la configurazione di accesso alle risorse. Nel nostro caso abbiamo specificato i permessi “liberi” a tutte le risorse REST tranne che alla risorsa “/filterImagePremium” che appunto richiede la validazione del token nell’header di richiesta. (Il token inviato sarà conservato nell’header con chiave “Authentication”).

Di seguito lo screen del codice dove all’arrivo delle richieste controlla se la risorsa in cui si vuole accedere deve passare prima sotto la validazione di un jwt:

```
@EnableWebSecurity
@Configuration
class WebSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .addFilterAfter(new JWTAuthorizationFilter(), UsernamePasswordAuthenticationFilter.class)
            .authorizeRequests()
                //chiedi il token per le richieste premium;
                .antMatchers(method: HttpMethod.POST, antPatterns: "/filterImagePremium").authenticated()
                .antMatchers(method: HttpMethod.POST, antPatterns: "/*").permitAll()
                .antMatchers(method: HttpMethod.GET, antPatterns: "/*").permitAll();
    }
}
```

La classe “JWTAuthorizationFilter” serve ad intercettare le chiamate alle risorse protette ed inoltre serve per recuperare il token determinando se il Client dispone delle autorizzazioni o meno. Questa classe intercetta tutte le chiamate al server (estende OncePerRequestFilter) e:

- *Verifica l'esistenza del token (checkJWTToken (...));*
- *Se esiste, decrittografa e convalida (validateToken (...));*
- *Se tutto è ok, aggiunge la configurazione necessaria al contesto Spring per autorizzare la richiesta (setUpStringAuthentication (...));*

7. AES Crittografia simmetrica

7.1 Cenni

AES è un algoritmo di crittografia simmetrica. AES è un cifrario a blocchi in grado di gestire blocchi a 128 bit , utilizzando chiavi di dimensioni a 128, 192 e 256 bit. Ogni cifra crittografa e decrittografa i dati in blocchi di 128 bit utilizzando chiavi crittografiche di 128, 192 e 256 bit, rispettivamente. Utilizza la stessa chiave per crittografare e decrittografare , quindi il mittente e il destinatario devono conoscere e utilizzare la stessa chiave segreta.

7.2 AES (Advanced Encryption Standard) nel progetto

Nel progetto è previsto l'utilizzo dell'algoritmo di crittografia AES-256 per codificare i dati sensibili che dal server saranno inviati al client, in particolare il Token del login Facebook e il JWT per l'utenza Premium prima di essere inviati al Client come risposta, vengono impacchettati e codificati con una chiave segreta che è memorizzata in un file json “settings.json”; per nessun motivo il Client avrà possibilità di possedere tale chiave e/o di leggere i dati codificati in “chiaro.”

Ogni volta che il Client invia tali dati codificati in AES-256 il Server prima di leggerli li decodifica utilizzando sempre la chiave segreta. Inoltre tale chiave sarà resa ulteriormente più sicura concatenando ad essa un'altra stringa alfanumerica (salt).

Di seguito il codice nel progetto del metodo che codifica con l'algoritmo di crittografia AES-256 una stringa passata come parametro:

```
public static String encrypt(String strToEncrypt) {
    try{
        byte[] iv = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
        IvParameterSpec ivspec = new IvParameterSpec(bytes: iv);

        SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
        KeySpec spec = new PBEKeySpec(chars: AES256.secretKey.toCharArray(), bytes: AES256.salt.getBytes(), i: 65536, ii: 256);
        SecretKey tmp = factory.generateSecret(spec);
        SecretKeySpec key = new SecretKeySpec(bytes: tmp.getEncoded(), string: "AES");

        Cipher cipher = Cipher.getInstance(string: "AES/CBC/PKCS5Padding");
        cipher.init(i: Cipher.ENCRYPT_MODE, key, aps: ivspec);
        return Base64.getEncoder().encodeToString(src: cipher.doFinal(bytes: strToEncrypt.getBytes(charsetName: "UTF-8")));
    }
    catch (UnsupportedEncodingException | InvalidAlgorithmParameterException | InvalidKeyException | NoSuchAlgorithmException) {
        System.out.println("Error while encrypting: " + e.toString());
    }
    return null;
}
```

8. Servizi di terze parti

8.1 Login con API Facebook

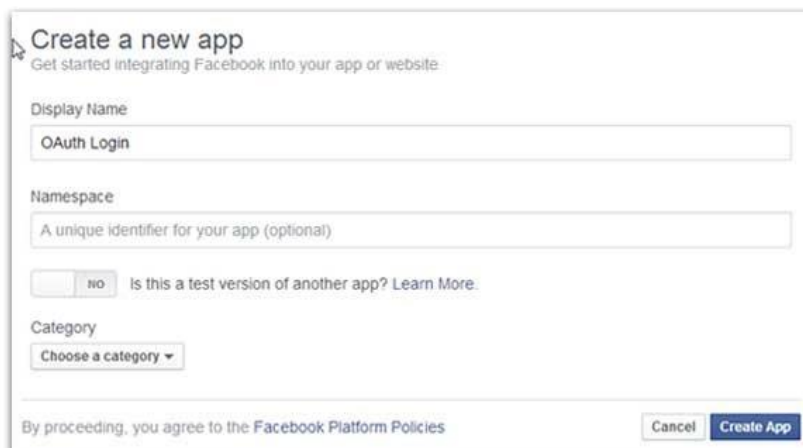
8.1.1 Introduzione

Facebook Login è un modo utile e veloce per creare account e accedere alla tua app su diverse piattaforme. È disponibile per iOS, Android, web, app per computer e dispositivi, come smart TV e oggetti per l'Internet delle cose. Facebook Login prevede due scenari: autenticazione e richiesta di autorizzazioni per accedere ai dati degli utenti. Puoi usare Facebook Login solo per l'autenticazione o sia per l'autenticazione sia per l'accesso ai dati.

8.1.2 Descrizione

Per prima cosa abbiamo creato un'applicazione web su Facebook.

Crea applicazione Facebook:



The screenshot shows the 'Create a new app' form on Facebook. The form has the following fields and options:

- Display Name:** A text input field containing 'OAuth Login'.
- Namespace:** A text input field with the placeholder text 'A unique identifier for your app (optional)'.
- Is this a test version of another app?:** A toggle switch set to 'no', with a 'Learn More' link.
- Category:** A dropdown menu with the text 'Choose a category'.
- Footer:** A line of text stating 'By proceeding, you agree to the Facebook Platform Policies', followed by 'Cancel' and 'Create App' buttons.

Abbiamo specificato l'URL di callback dell'applicazione nelle impostazioni FB. Questo verrà utilizzato dal server FB durante l'autenticazione per restituire il controllo alla nostra applicazione. Il nostro Server HTTPS aspetta la comunicazione all'indirizzo: "https://localhost:8443/loginFB".

Di seguito verrà descritto il funzionamento generico della comunicazione tra l'app e l'API di Facebook per accedere e per richiedere dati utente:

- *All'accesso di un URL o nella pagina di benvenuto viene visualizzato il pulsante di accesso a Facebook. L'utente farà clic sul pulsante FB, facendo clic su quel pulsante verrà richiamato un URL di Facebook.*
- *Facebook convaliderà l'ID dell'applicazione e quindi reindirizzerà alla sua pagina di accesso.*
- *L'utente inserirà le credenziali di accesso FB e invierà il modulo.*
- *Facebook convaliderà le credenziali e quindi reindirizzerà al browser con una richiesta da inoltrare a `redirect_url`. `Redirect_url` è l'URL nella nostra applicazione che si occuperà dell'ulteriore elaborazione.*
- *Il browser chiamerà l'URL di reindirizzamento.*
- *Il Client chiamerà nuovamente Facebook per richiedere `access_token`.*
- *Facebook in caso di successo della convalida risponderà con `access_token`.*
- *La pagina URL di reindirizzamento chiamerà nuovamente Facebook per richiedere i dati dell'utente inviando `access_token`.*
- *Facebook alla convalida di `access_token` risponderà con i dati dell'utente richiesti.*

9. Servizi di terze parti (Cloud)

9.1 Cloud Storage

9.1.1 Introduzione

In informatica con il termine inglese cloud computing (in italiano nuvola informatica) si indica un paradigma di erogazione di risorse informatiche, come l'archiviazione, l'elaborazione o la trasmissione di dati, caratterizzato dalla disponibilità on demand attraverso Internet a partire da un insieme di risorse preesistenti e configurabili.



Diagramma logico di una rete cloud computing

9.1.2 Google Drive

Google Drive è un servizio web, in ambiente cloud computing, di memorizzazione e sincronizzazione online introdotto da Google il 24 aprile 2012. Basato su software open source, comprende il file hosting, il file sharing e la modifica collaborativa di documenti, inizialmente fino a 5 GB, da ottobre 2013, invece fino a 15 GB gratuiti (inclusivi dello spazio di memorizzazione di Gmail e delle foto di Google+) estendibili fino a 30 TB in totale.

Può essere usato via Web, caricando e visualizzando i file tramite il web browser, oppure tramite l'applicazione installata su computer, che sincronizza automaticamente una cartella locale del file system con quella condivisa. Su Google Drive sono presenti anche i documenti creati con Google Document. La peculiarità dell'applicazione è di risiedere sul server Google e di essere lanciata da remoto, non richiedendo l'installazione di alcun software sul computer locale.



9.1.3 Google Drive nel progetto

In questo software l'utente potrà utilizzare liberamente l'API di Google Drive cliccando il bottone "Carica immagine su Google Drive", al click si instaura una connessione HTTPS con il server Google, specificando il nome dell'applicazione, creata dall'account Google Drive Developers, dopo di che, mediante la procedura `getCredentials()` si richiede l'autorizzazione al server Google, di poter uploadare il file fornendo come credenziali quelle contenute nel file "credentials.json" che conterrà tutti i dati relativi all'account dove verranno caricati i dati. Infine se tutto è andato a buon fine verrà mostrato un messaggio di successo e l'immagine visualizzata sulla GUI del Client sarà caricato sull'account personale di Google Drive.

Di seguito metodo "uploadFile(...)" del progetto, chiamato dopo l'autorizzazione d'accesso dell'utente all'account Google Drive:

```
public void uploadFile(String filename) throws GeneralSecurityException, IOException {
    try{
        // Build a new authorized API client service.
        final NetHttpTransport HTTP_TRANSPORT = GoogleNetHttpTransport.newTrustedTransport();
        Drive service = new Drive.Builder(transport: HTTP_TRANSPORT, jsonFactory: JSON_FACTORY, httpRequestInitializer: getCredentials(HTTP_
            .setApplicationName( applicationName: APPLICATION_NAME)
            .build();
        //Upload
        File fileMetadata = new File();
        //scelgo il file img
        //String filename = MyImage.uploadFileImg();
        fileMetadata.setName( name: filename);
        java.io.File filePath = new java.io.File( string: filename);
        FileContent mediaContent = new FileContent( type: "image/jpeg", file: filePath);
        File file = service.files().create( content: fileMetadata, mediaContent)
            .setFields( fields: "id")
            .execute();
        System.out.println("File ID: " + file.getId());
        showMessageDialog( parentComponent: null, message: "L'immagine visualizzata su IMAGico è stata caricata sul tuo Google Drive!",
    }
    catch(IOException | GeneralSecurityException e){
        showMessageDialog( parentComponent: null, message: "Qualcosa nel caricamento su Google Drive è andato storto", title: "Attenzione"
    }
}
```

9.2 Twilio API

9.2.1 Introduzione

Twilio è una delle tante proposte cloud oggi esistenti sul mercato, ma a differenza di tutti gli altri usi, suggerisce un utilizzo del cloud davvero innovativo. Twilio è un sistema cloud per la personalizzazione dei servizi telefonici e SMS. In pratica, con un semplice linguaggio di programmazione ogni sviluppatore ha l'opportunità di creare una propria applicazione telefonica o SMS personalizzata secondo le proprie esigenze. L'API Twilio è un'API RESTful che fornisce funzionalità voce ed SMS per le applicazioni.



9.2.2 API di verifica attraverso sms

Nel progetto abbiamo utilizzato l'API di Twilio dedicata alla verifica di un numero di telefono, in questo modo l'utente riceverà un Token univoco da inserire per essere validato e riconosciuto.

Verify utilizza due endpoint API per verificare senza problemi che un utente è il proprietario del numero di telefono fornito.

- *L'endpoint API iniziale:*

Quando l'utente convalida il proprio numero di telefono, il servizio invia all'utente un codice di verifica di 4-10 cifre tramite SMS e attende la sua risposta.

- *L'endpoint dell'API di controllo*

Il servizio controlla che l'input dell'utente corrisponda al codice. Se corrispondono, il servizio registrerà il numero di telefono come approvato.

9.2.3 API di verifica nel progetto

Per poter utilizzare il servizio di autenticazione via sms per prima cosa bisogna creare un account Twilio, tale account ci fornirà un id account e un token segreto, entrambi ci serviranno per comunicare con l'API di Twilio. In questo caso abbiamo salvato tali dati in un file json conservato nella directory del Server "settings_verify_sms.json". Dopodichè suddividiamo il funzionamento in 3 passaggi:

1. *Creiamo un servizio di verifica. In questo caso tramite account Twilio abbiamo creato un servizio di prova gratuito, esso ci fornirà un id che aggiungeremo nel file json allocato nella directory del Server "settings_verify_sms.json", questo sarà necessario per comunicare con API Twilio.*
2. *Invia un Token di verifica. Nel nostro software il Client si occuperà di inviare al nostro Server il numero di cellulare inserito dall'utente, dopodichè il nostro Server HTTPS comunicherà con l'API di Twilio inviando la richiesta con il numero di cellulare inserito e gli altri dati conservati nel file json "settings_verify_sms.json".*

Di seguito il codice di invio Token in questa applicazione:

```
@PostMapping("smsVerify")
@ResponseBody
private boolean sendVerificaToken(@RequestParam("number") String number) {
    try{
        Twilio.init( username: SmsVerifyController.account_Sid, password: SmsVerifyController.auth_token);
        Verification verification = Verification.creator(
            pathServiceSid: SmsVerifyController.serviceSid,
            "+39"+ number,
            channel: "sms"
        ).create();
        System.out.println("verification.getStatus() sendVerificaToken ---> " + verification.getStatus());
        return true; //ok
    }
    catch(Exception e){
        System.out.println("e: " + e);
        return false;
    }
}
```

3. *Controlla il token di verifica. Nel nostro software il Client si occuperà di inviare al nostro Server il token di verifica ricevuto via sms dal numero di cellulare inserito dall'utente, dopodichè il nostro Server HTTPS comunicherà con l'API di Twilio inviando la richiesta con il token di verifica inserito e gli altri dati conservati nel file json "settings_verify_sms.json", il servizio con cui comunichiamo controllerà se il token fornito dall'utente è corretto.*

Di seguito il codice di validazione Token di verifica (sms) in questa applicazione:

```
@PostMapping("checkNumberToken")
@ResponseBody
private boolean sendVerificaToken(@RequestParam("tokenNumber") String tokenNumber, @RequestParam("number") String number){
    try{
        Twilio.init( username: SmsVerifyController.account_Sid, password: SmsVerifyController.auth_token);
        VerificationCheck verificationCheck = VerificationCheck.creator(
            pathServiceSid: SmsVerifyController.serviceSid,
            code: tokenNumber
        ).setTo("+39" + number).create();
        System.out.println("verificationCheck.getStatus() checkNumberToken -->" + verificationCheck.getStatus());
        if("approved".equals( anObject: verificationCheck.getStatus())){ //ok codice approvato
            return true; //approved
        }
        else if("pending".equals( anObject: verificationCheck.getStatus())) //codice errato
            return false;
        else
            return false;
    }
    catch(Exception e){
        System.out.println( e);
        return false;
    }
}
```

10. Caratteristiche OOP

10.1 Incapsulamento

Con il termine *Incapsulamento* si intende la possibilità di un linguaggio di programmazione ad oggetti di accorpare metodi e proprietà all'interno di un'unica area, ovvero all'interno dell'Oggetto. In questo modo il nostro programma verrà ridotto a tante piccole parti, ognuna che incapsula una qualche funzionalità.

Nella classe *MyImage.java*, ad esempio, sono stati associati degli attributi (*bufferedImg*, *width*, *height*) e dei metodi (*setBufferedImg*, *uploadFileImg*, ecc..) che sono caratteristici della classe stessa:

```
public class MyImage {

    private BufferedImage bufferedImg;
    private int width;
    private int height;

    public MyImage(int width, int height, String fileImg) throws IOException{
        //appena creo l'oggetto mi converto subito il path in un bufferedImg
        this.bufferedImg = ImageIO.read(new File( string: fileImg));
        this.width = width;
        this.height = height;
    }
    //Costruttore che a differenza del primo ha come terzo argomento un BufferedImage (util:
    public MyImage(int width, int height, BufferedImage bufferedImg) throws IOException{
        this.bufferedImg = bufferedImg;
        this.width = width;
        this.height = height;
    }

    public void setBufferedImg(BufferedImage bufferedImg){
        this.bufferedImg = bufferedImg;
    }

    public BufferedImage getBufferedImg(){
        return this.bufferedImg;
    }
    public void setWidth(int width){
        this.width = width;
    }
    public void setHeight(int height){
        this.height = height;
    }
    public int getWidth(){
        return this.width;
    }
    public int getHeight(){
        return this.height;
    }

    public static String uploadFileImg(){
```

10.2 Information hiding

Direttamente legato all'incapsulamento, vi è l'information hiding. Questo modo di intendere gli oggetti induce a considerarli come delle scatole nere (black box). I dettagli sulle caratteristiche e la struttura dell'oggetto sono nascosti all'interno, garantendo l'information hiding. Partendo dai principi dell'information hiding si è sviluppato un nuovo modo di concepire il software come formato da scatole, che mettono a disposizione un insieme di funzionalità. Il contenuto di questi componenti resta nascosto, mentre se ne conosce il modo con cui operano.

Esempio nella classe "JWTPremium.java":

```
public class JWTPremium {
    private String token;

    protected void setToken(String email) {
        //chiamo metodo per reperire chiave segreta da file json
        String secretKey = secret_key_jwt();
        List<GrantedAuthority> grantedAuthorities = AuthorityUtils
            .commaSeparatedStringToAuthorityList( authorityString: "ROLE_USER");
        this.token = Jwts
            .builder()
            .setSubject( sub:email)
            .claim( name: "authorities",
                value: grantedAuthorities.stream()
                    .map(GrantedAuthority::getAuthority)
                    .collect( collector: Collectors.toList()))
            .setIssuedAt(new Date( 1: System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() + 6000000))
            .signWith( alg: SignatureAlgorithm.HS512,
                secretKey: secretKey.getBytes() ).compact();
    }

    protected String getToken() {
        return "Bearer " + this.token;
    }
}
```

10.3 Polimorfismo

Il termine polimorfismo si riferisce in generale alla possibilità data ad una determinata espressione di assumere valori diversi in relazione ai tipi di dato a cui viene applicata. Nel contesto della programmazione OO e del linguaggio Java si possono distinguere tre tipi di polimorfismo: ad hoc polymorphism; inclusion polymorphism; parametric polymorphism.

- *Il polimorfismo ad hoc (chiamato anche “method overloading”) consiste nella possibilità di “ridefinire” un medesimo metodo per set di parametri diversi.*
- *Il polimorfismo ad inclusione è strettamente legato al concetto di ereditarietà e di sub-typing e consiste nella possibilità che una sottoclasse A di una data classe B ridefinisca uno dei metodi della super-classe e che quindi quando verrà utilizzata un'istanza della classe A le invocazioni al metodo ridefinito (spesso detto overridden) eseguiranno il codice definito nella sotto-classe.*
- *Il polimorfismo parametrico è legato ad una feature di Java detta generics2. I Generics sono stati introdotti a partire dalla versione J2SE 5, il termine ha un significato di tipo parametrizzato. I tipi parametrizzati rivestono particolare importanza perché consentono di creare classi, interfacce e metodi per i quali il tipo di dato sul quale si opera può essere specificato come parametro. Parliamo quindi di classi, interfacce e metodi generici. Attraverso un codice generico possiamo realizzare un algoritmo che astrae dal tipo di dato specifico sul quale opera. Ad esempio un algoritmo per l'ordinamento di entità numeriche è lo stesso qualunque sia il tipo numerico che si utilizza; il tipo numerico utilizzato rappresenta quindi un parametro.*

Di seguito un esempio di polimorfismo ad inclusione presente nel codice del progetto:

```
public class ImageController {
    //img di tipo BufferedImgBase
    private BufferedImgBase img;
    private byte[] newByte;
    private String fileName;
    private Timestamp timestamp;
    private Class classeFiltro;

    //POST request
    @PostMapping("/filterImage")
    @ResponseBody
    public String postRequest(@RequestBody byte[] imgbyte, @RequestHeader(value = "Filter") String filter, @
        //ottengo dinamicamente nome del sottotipo di classe che ci interessa in base a filter
        classeFiltro = Class.forName("com.mycompany.serverspring.imageFilter." + filter);
        Constructor con = classeFiltro.getConstructor( parameterTypes: byte[].class);
        //creo polimorficamente l'oggetto di tipo filter che ci interessa
        this.img = (BufferedImgBase) con.newInstance( initargs: imgbyte);

        //call applicaFiltro()
        this.newByte = this.img.applicaFiltro();

        if(this.newByte != null){
            //Salvo l'img con il valore di timestamp attuale
            timestamp = new Timestamp(1: System.currentTimeMillis());
            this.fileName = timestamp.getTime() + ".jpeg";
            this.img.saveImg( byteImg: this.newByte, fileName: this.fileName);
            return fileName; //ritorno il nome del file salvato (timestamp)
        }
        else
            return null;
    }
}
```

In questo caso la potenza del polimorfismo sta nel fatto che la classe astratta di tipo "BufferedImgBase" prende le forme dei sottotipi che sono individuati nei filtri da applicare, quindi automaticamente "img" che è una proprietà di tipo "BufferedImgBase" prenderà la forma di uno di questi "filtri" e quindi, solo una volta, verrà chiamato il metodo "applicaFiltro()" che agirà a secondo della forma in cui si trova.

10.4 Astrazione

L'astrazione è un paradigma della programmazione ad oggetti che potremmo definire come l'arte di sapersi concentrare solo sui dettagli veramente essenziali per far funzionare correttamente un programma.

Le classi astratte in Java sono utilizzate per poter dichiarare caratteristiche comuni fra classi di una determinata gerarchia. Pur definendo il nome di un tipo, la classe astratta non può essere istanziata, analogamente a quanto accade per le interfacce; ma a differenza di una interfaccia può avere field non statici, metodi non pubblici, un costruttore, insomma una classe a tutti gli effetti ma non istanziabile.

Una classe astratta può contenere o meno metodi astratti (un metodo astratto è un metodo che può essere dichiarato ma per il quale non viene fornita una implementazione), ma una classe che contiene metodi astratti deve necessariamente essere dichiarata come astratta, facendo uso della keyword `abstract`.

Nel progetto, un esempio di classe astratta è la classe "BufferedImage.java". Essa definisce in generale le forme che potrà assumere l'immagine originale inviata dall'utente dopo che verranno applicati i filtri desiderati, infatti come possiamo vedere nel codice è presente un metodo astratto "applicaFiltro", che troverà la sua implementazione in tutti gli oggetti che saranno sottotipi di BufferedImage.

Di seguito lo screen di un esempio di classe astratta scritta nel progetto:

```
public abstract class BufferedImgBase {

    protected BufferedImage img;

    //converti i byte di img in un buffered img ogni qualvolta viene istanziata una
    public BufferedImgBase(byte[] byteimg) throws IOException{
        // convert byte[] back to a BufferedImage
        InputStream is = new ByteArrayInputStream(bytes:byteimg);
        BufferedImage newBi = ImageIO.read(input:is);
        //this.img --> BufferedImage
        this.img = newBi;
    }

    public BufferedImage getImg(){
        return this.img;
    }

    public void setImg(BufferedImage img){
        this.img = img;
    }

    //metodo che converte array di byte in bufferedimage
    public void saveImg(byte[] byteImg, String fileName) throws IOException{
        File outputFile = new File("src/main/resources/tmpImg/" + fileName);
        // convert byte[] back to a BufferedImage
        BufferedImage newBi = ImageIO.read(new ByteArrayInputStream(bytes:byteImg));
        // save it
        ImageIO.write(im:newBi, formatName:"jpeg", output:outputFile);
    }

    public abstract byte[] applicaFiltro();
}
```

10.5 Ereditarietà

L' ereditarietà è uno dei concetti fondamentali del paradigma della programmazione ad oggetti.

Essa consiste in un legame che il linguaggio di programmazione, o il programmatore stesso, stabilisce tra due classi.

Se la classe «B» eredita dalla classe «A», si dice che «B» è una classe derivata (o sottoclasse) di «A» e che «A» è una classe principale (o superclasse) di «B».

Una classe B dichiarata sottoclasse di un'altra classe A:

- *Eredita (ha implicitamente) tutte le variabili di istanza e tutti i metodi di A;*
- *Può avere variabili o metodi aggiuntivi;*
- *Può ridefinire i metodi ereditati da A attraverso l'overriding, in modo che essi eseguano la stessa operazione concettuale in un modo specializzato (polimorfismo).*

Essa è legata al concetto di astrazione.

Di seguito lo screen di un esempio nel progetto di classe (SendImage.java) che estende (extends) un'altra classe (HttpClient.java), e quindi eredita il suo stato e comportamento diventando sottoclasse di essa:

```
public class SendImage extends HttpClient{
    private String postmapping;
    private String charset;
    private byte[] img;
    private String filter;

    public SendImage(byte[] img, String filter){
        super();
        this.postmapping = "filterImage";
        this.charset = "UTF-8";
        this.img = img;
        this.filter = filter;
    }
}
```

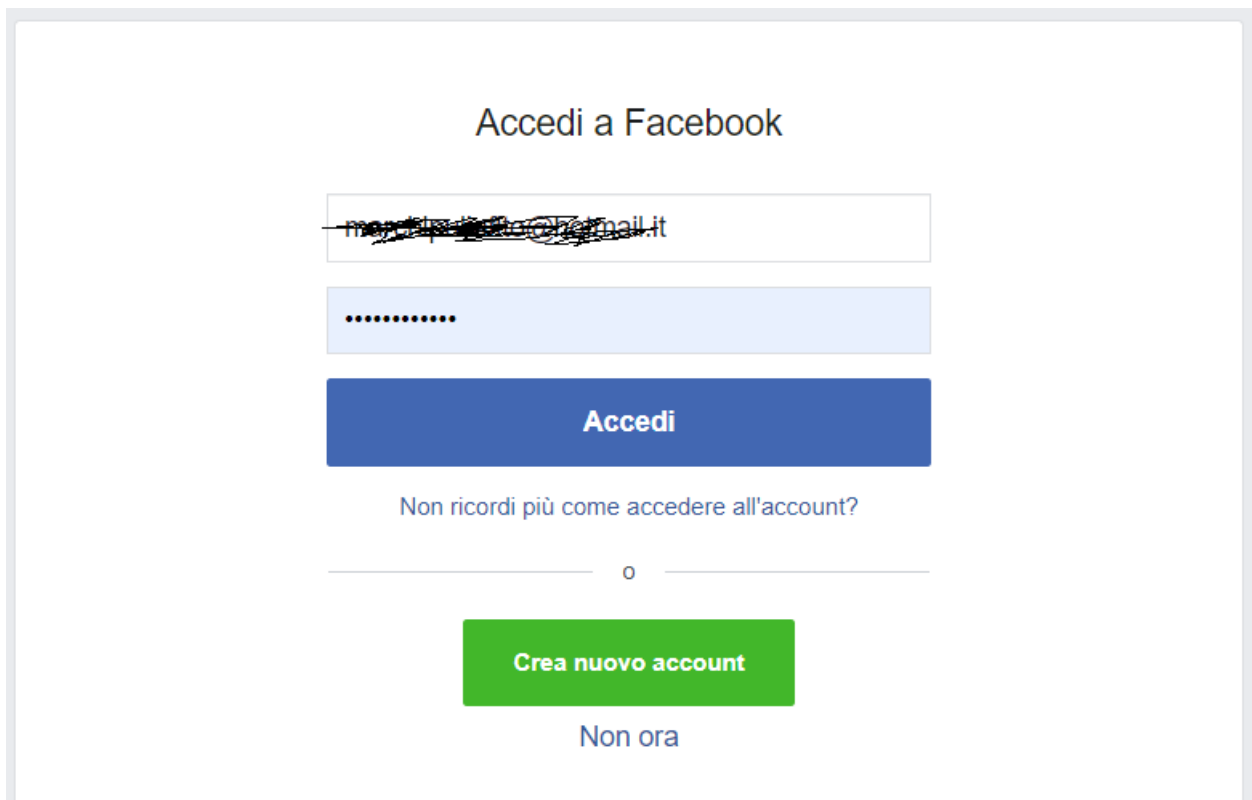
11. *Descrizione interfacce grafiche*

11.1 *Client Utente*



GUI del Client Utente al primo accesso

Dopo aver cliccato il bottone “Accedi con Facebook” si verrà proiettati sul browser per accesso con account Facebook e permessi di autorizzazione applicazione:

A screenshot of the Facebook login page. At the top, the text "Accedi a Facebook" is centered. Below it is a text input field containing the email address "mario.pasquale@unimessina.it", which is partially obscured by a black scribble. Underneath the email field is a password field represented by a light blue rectangle with ten black dots. A large blue button with the text "Accedi" is positioned below the password field. Under the button, the text "Non ricordi più come accedere all'account?" is displayed. Below this text is a horizontal line with a small "o" in the center. At the bottom of the login area is a green button with the text "Crea nuovo account". Below the green button is the text "Non ora" in a smaller font.

Accedi con il tuo account Facebook



Java project riceverà le seguenti informazioni:
il tuo nome e la tua immagine del profilo e indirizzo e-mail.

 [Modifica](#)

Ciò non consente all'app di pubblicare contenuti su Facebook

Continua come Marco

Annulla

Se continui, Java project otterrà accesso alle informazioni che condividi e Facebook registrerà quando Java project vi accede. [Scopri di più](#) su questa condivisione e sulle impostazioni a tua disposizione.

[Normativa sulla privacy](#) di Java project

Autorizza questa applicazione di accedere ai tuoi dati utenti attraverso Facebook

Confermando il tutto verrà mostrato un messaggio sul browser, a questo punto bisognerà tornare alla GUI utente e cliccare “Conferma accesso”:



GUI del Client Utente dopo conferma Facebook

A questo punto saremo dentro al software:



GUI principale del Client Utente

In alto a sinistra potremo vedere l'Email con il quale ci siamo loggati, e da qui saremo in grado di capire se siamo loggati con un utente Premium o meno:

Email utente: marchipuliafito@hotmail.it

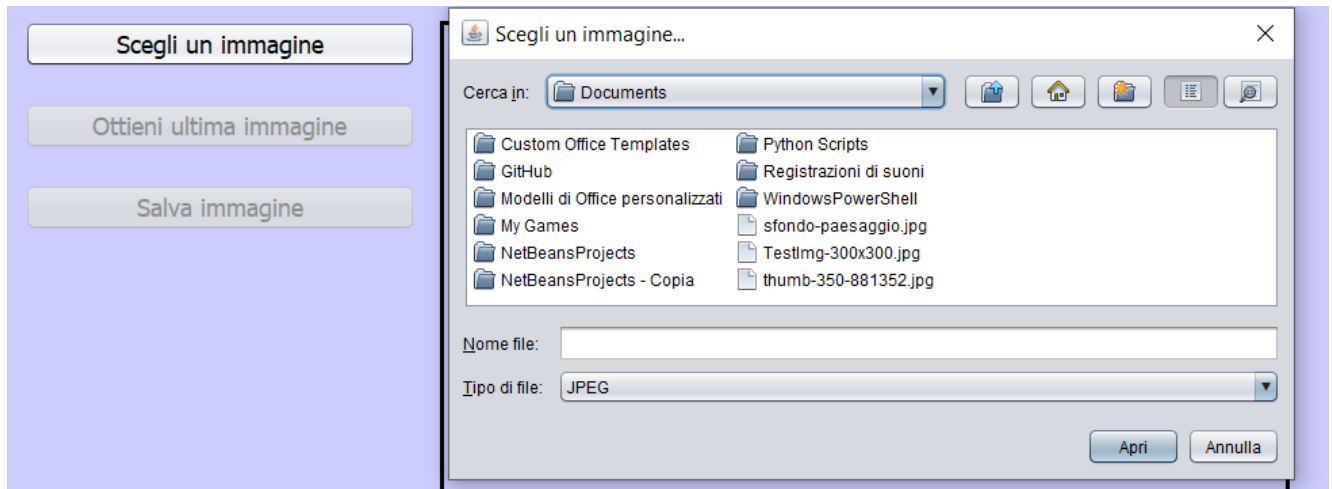
Descrizione email utente "Base" loggato

Email utente PREMIUM: marchipuliafito@hotmail.it

Descrizione email utente "Premium" loggato

UNIVERSITA' DEGLI STUDI DI MESSINA

*Per caricare una nuova immagine basta cliccare il bottone sulla GUI
"Scegli un Immagine":*



Finestra per la selezione di una nuova immagine



GUI utente dopo aver caricato una nuova immagine

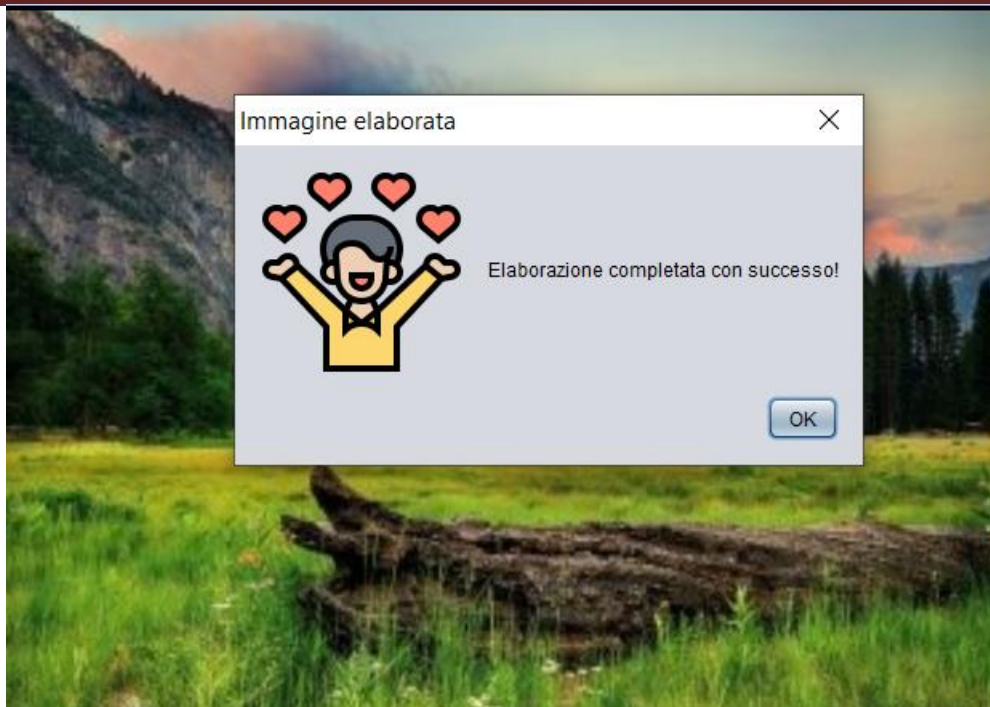
Per applicare un filtro all'immagine sarà possibile scegliere nel menù laterale:



Bottoni per applicare i filtri

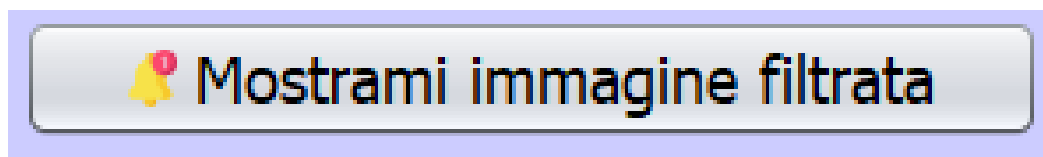


Bottoni filtri per utenti Premium

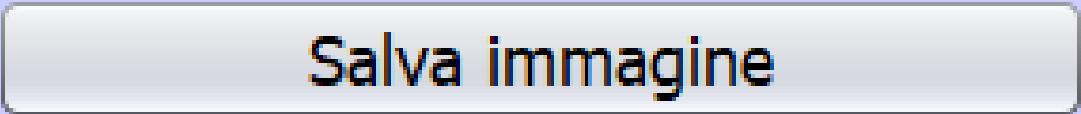


Finestra di successo al click di un filtro


Dopo la visualizzazione di “immagine elaborata” nel menù a sinistra verrà sbloccato il bottone “Mostrami immagine filtrata”, esso ci permetterà di ricevere la nuova immagine:



Bottone di avviso di un immagine elaborata dal Server e pronta ad essere mostrata sulla GUI

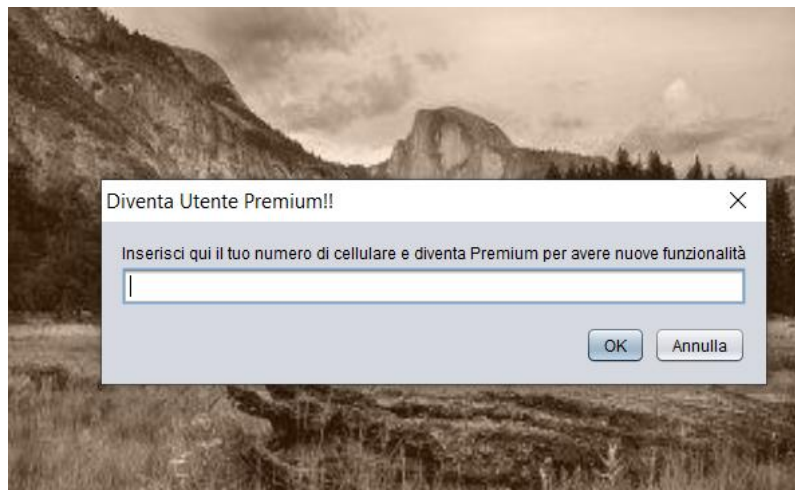
A rectangular button with a light gray gradient and a thin black border. The text "Salva immagine" is centered in a bold, black, sans-serif font.

Bottone per salvare immagine mostrata sulla GUI utente

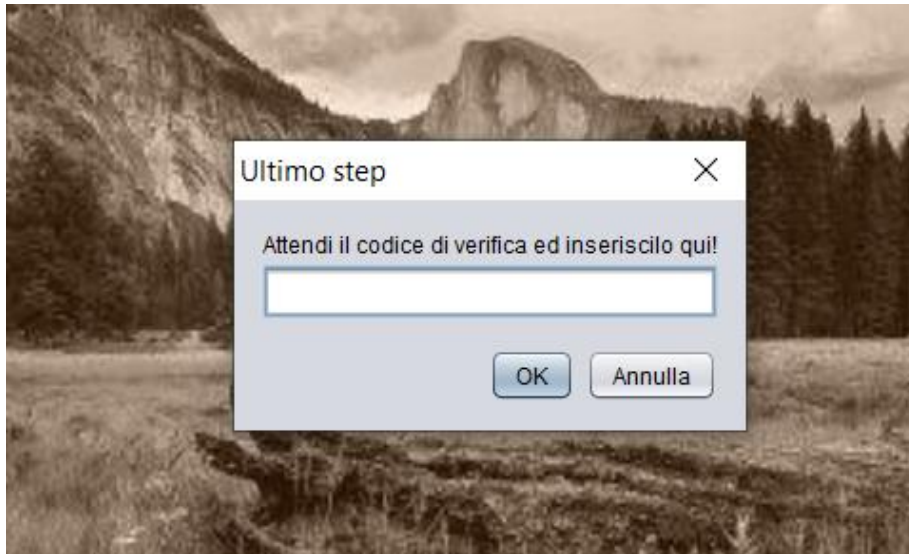
A blue rectangular button with rounded corners and a slight 3D effect. The text "Diventa Premium" is centered in a bold, red, sans-serif font.

Bottone per diventare utenti “premium” (visibile solo ad utenti “base”)

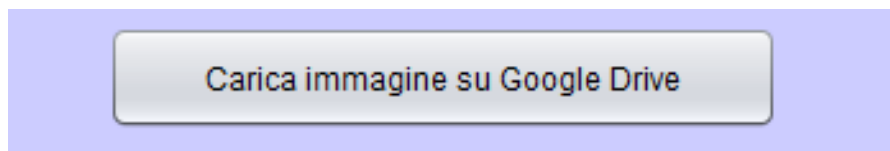
Al click di “Diventa Premium”:



Finestra con input di testo dove inserire numero telefonico per ricevere sms con codice “Premium”



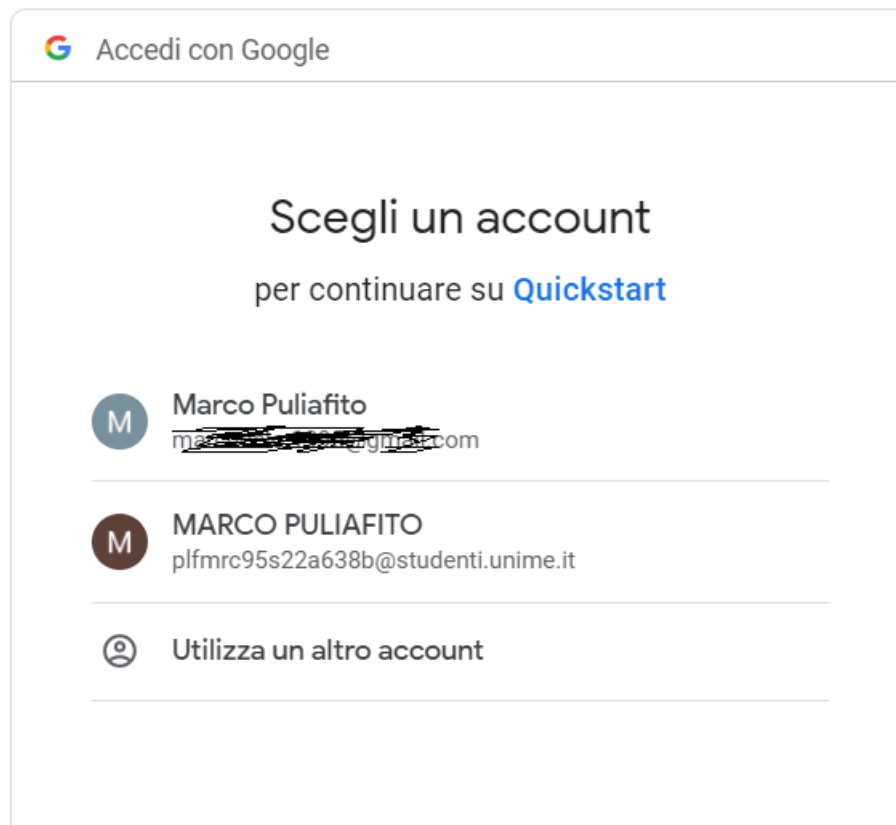
Finestra con input di testo dove inserire codice "Premium"



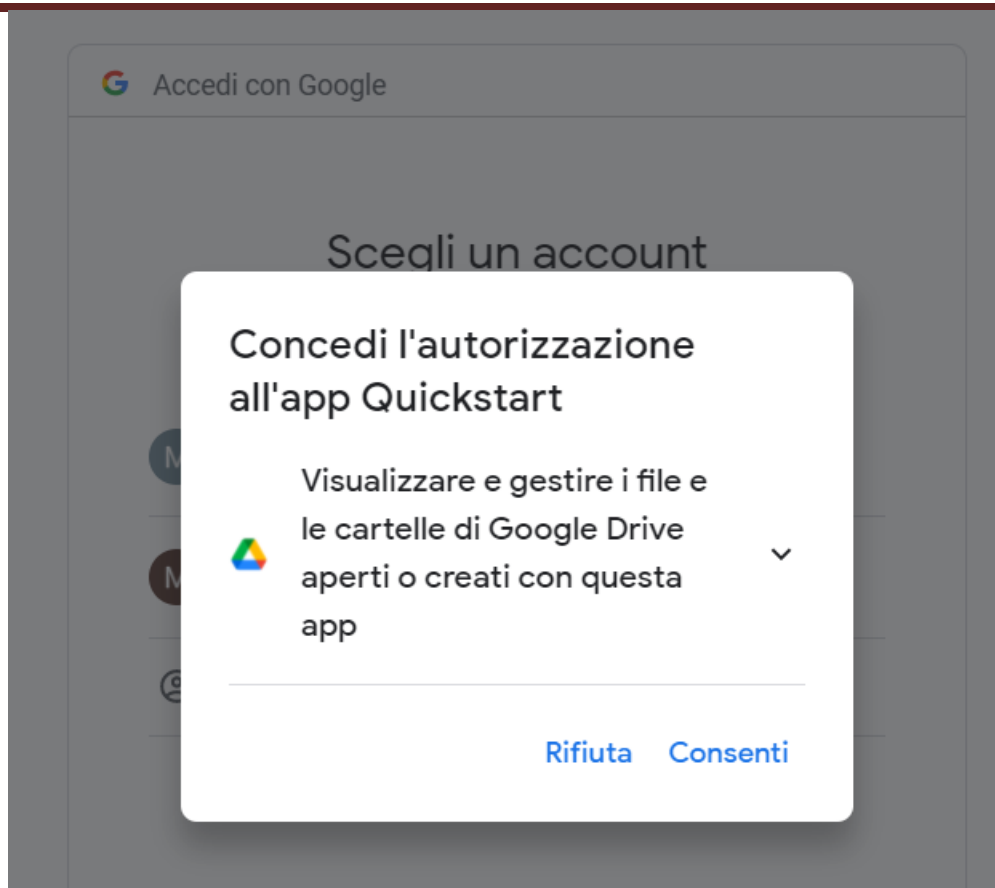
Bottone per salvare immagine corrente su Googl Drive

UNIVERSITA' DEGLI STUDI DI MESSINA

Dopo il click per caricare immagine su Google Drive:

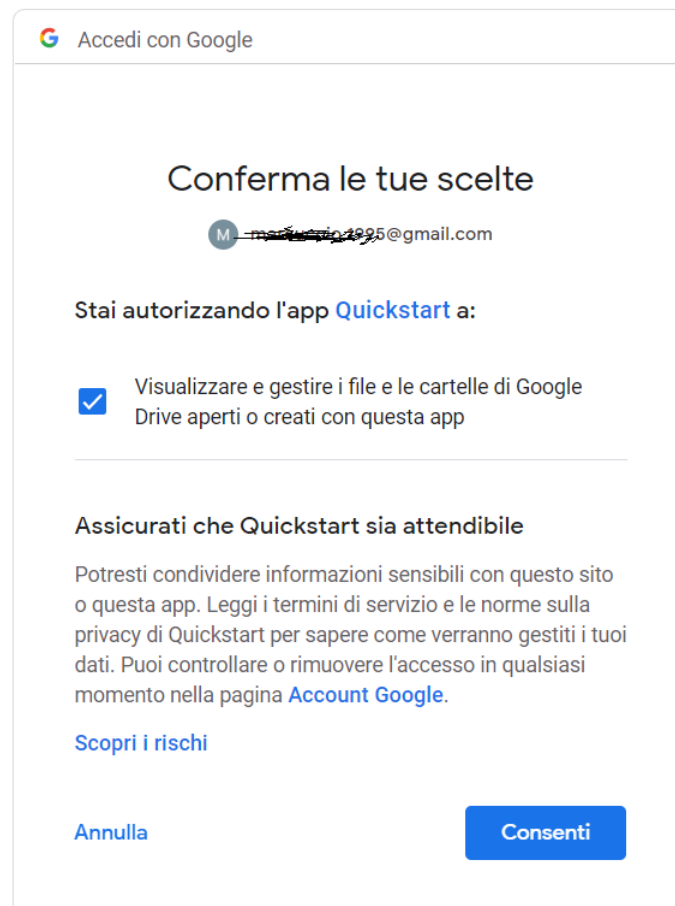


Finestra sul browser per accedere con Google Drive

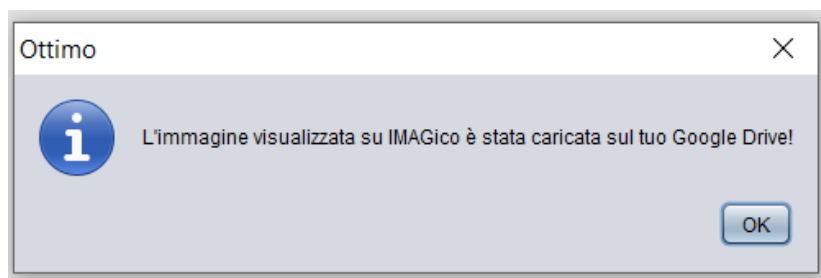


Finestra per consentire alla nostra app l'autorizzazione da Google Drive

UNIVERSITA' DEGLI STUDI DI MESSINA

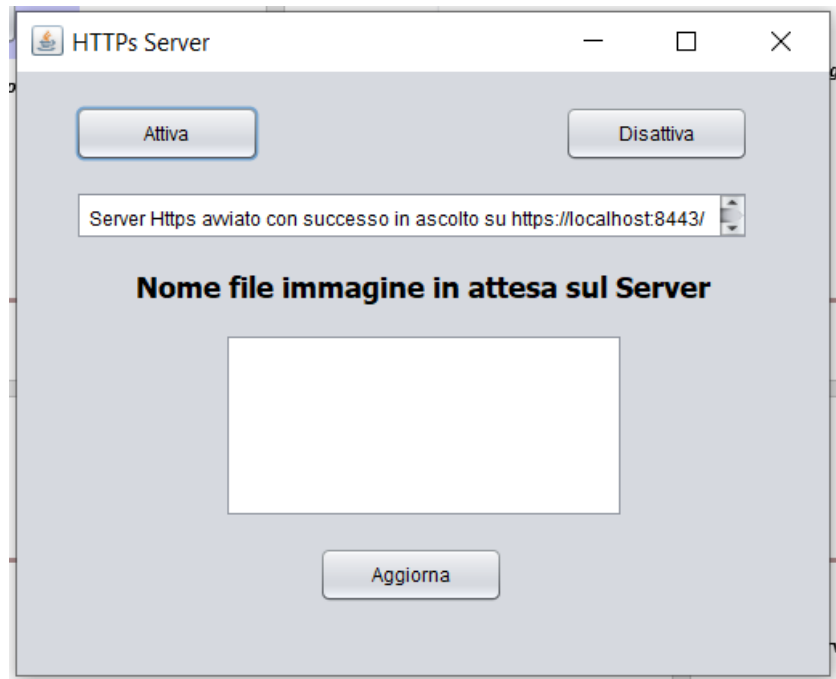


Finestra per consentire alla nostra app l'autorizzazione da Google Drive



Messaggio di conferma caricamento su Google Drive

11.2 Server HTTPS



GUI del Server

12 Diagrammi UML

12.1 Introduzione

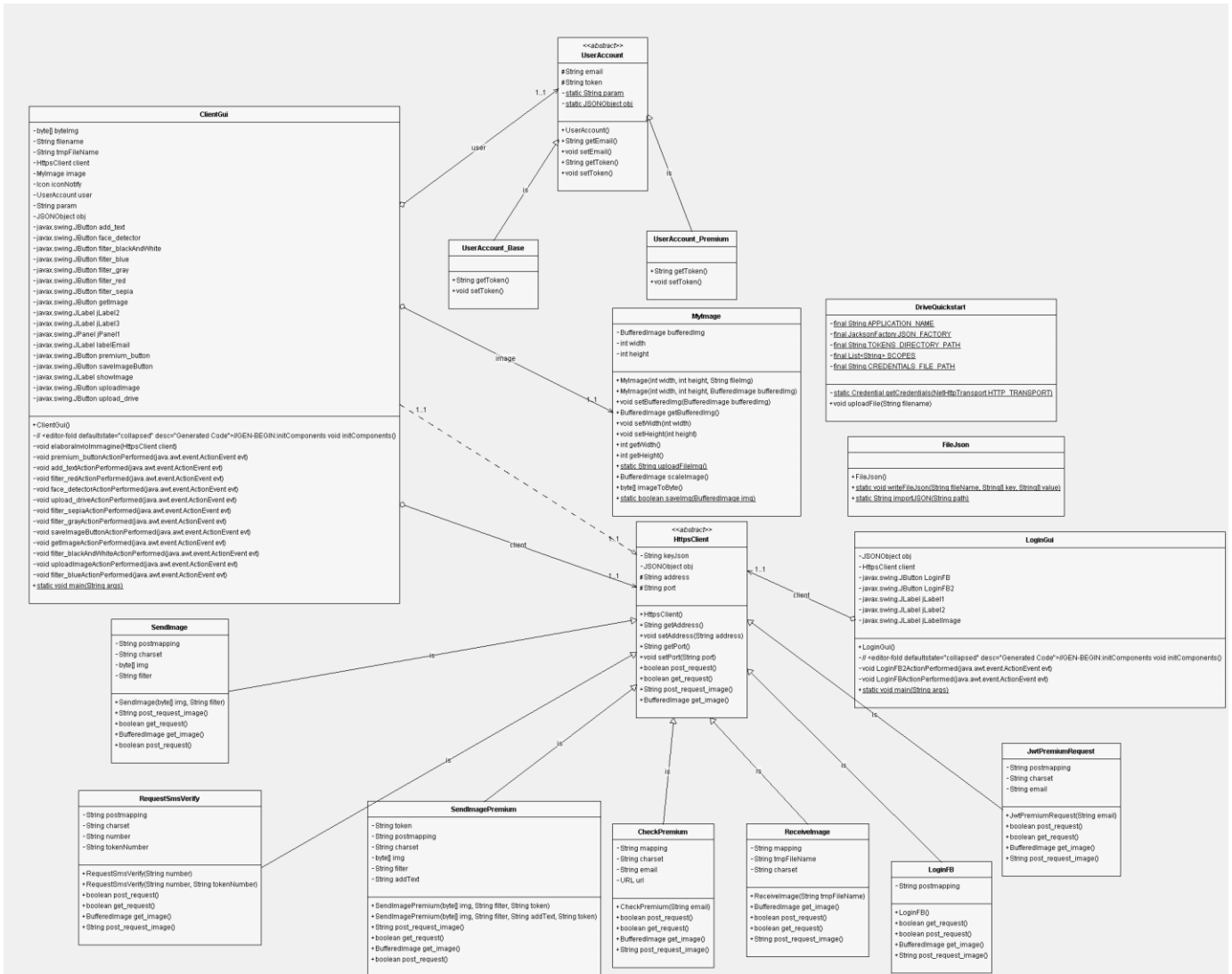
UML (unified modeling language, "linguaggio di modellizzazione unificato") è un linguaggio di modellazione e specifica basato sul paradigma orientato agli oggetti. Il nucleo del linguaggio fu definito nel 1996 da Grady Booch, Jim Rumbaugh e Ivar Jacobson.

Un modello UML è costituito da una collezione organizzata di diagrammi correlati, costruiti componendo elementi grafici (con significato formalmente definito), elementi testuali formali, ed elementi di testo libero. Ha una semantica molto precisa e un grande potere descrittivo.

Il linguaggio è stato progettato con l'obiettivo esplicito di facilitare il supporto software alla costruzione di modelli e l'integrazione di questo supporto con gli ambienti integrati di sviluppo.

Il linguaggio nacque con l'intento di unificare approcci precedenti, raccogliendo le migliori prassi nel settore e definendo così uno standard industriale unificato.

12.2 Class Diagram Client



12.3 Class Diagram Server

