Sommario

1. Scopo del documento

Lo scopo del presente documento è quello di descrivere le features di configurabilità degli aspetti di security in guigen, che permettono di adattare le applicazioni generate da guigen a differenti contesti di piattaforme di sicurezza.

2. Riferimenti

3. Introduzione

Guigen gestisce due aspetti di security:

1. L'**autenticazione**, intesa come protezione dell'accesso applicativo e contestualizzazione della logica applicativa a fronte dell'identità dell'utente autenticato.
2. L'**autorizzazione**, intesa come abilitazione di componenti di user interface a fronte dell'utente autenticato (es. un pulsante viene attivato solo se l'utente corrente ricopre un particolare ruolo).

L'implementazione standard prevista in *guigen* permette di realizzare queste due features secondo le modalità standard utilizzate in CSI Piemonte, ovvero:

- Autenticazione:
  - Interazione con l'implementazione custom CSI di *shibboleth*, che utilizza il sistema proprietario IRIDE2 come *policy enforcement point (P.E.P)*
  - Interazione con la customizzazione CSI del sistema di single-sign-on *CAS* (denominato SSOBART), che utilizza il sistema proiprietatio IRIDE2 come *policy enforcement point*
- Autorizzazione:
  - Verifica dell'abilitazione tramite meccanismi basati su *UseCase*, *Actor*, *Role*, delegata ad una serie di appositi servizi forniti dal sistema proprietario IRIDE2

Questi scenari sono evidentemente strettamente legati al contesto CSI. Per questo motivo, per l'utilizzo di MDDTools in contesti differenti, è necessario introdurre nello strumento *guigen* una soluzione più generica che permetta di implementare l'integrazione con le differenti piattaforme di security presenti nell'ambiente in cui dovrà poi essere impiantato il sistema generato con *guigen*.

Occorre pertanto predisporre un sistema generico che permetta di ricoprire un insieme ben determinato di scenari architetturali permettendo in questo modo, mediante opportuni plugin, alcuni dei quali possono eventualmente essere forniti come "libreria", l'adattamento ai vari contesti[1].

4. Scenari supportati

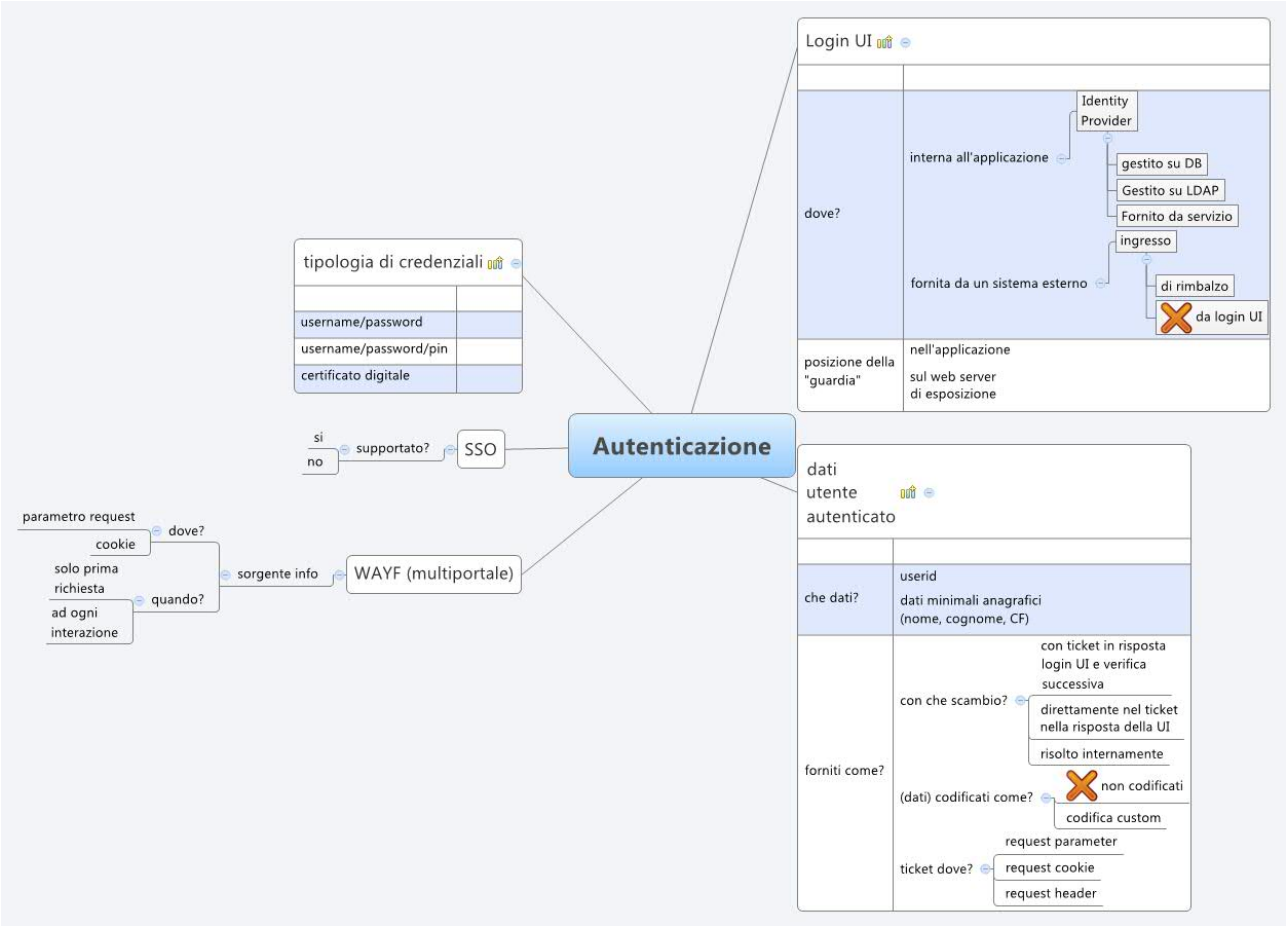
Al fine di progettare una struttura "pluggabile" di security è necessario passare in rassegna alcuni meccanismi di security con l'obiettivo di estrapolare un insieme di possibili scenari architetturali che risultino *rappresentativi della maggior parte di casistiche*. E' infatti proprio al livello architetturale che si esplicitano maggiormente le difficoltà di integrazione con i differenti sistemi di security. A livello di progettazione interna, invece i gradi di libertà sono maggiori, rendendo meno cruciale un'analisi approfondita preventiva.

[1] In questo scenario di implementazione generica "pluggabile" le contestualizzazioni CSI potrebbero risultare semplicemente un insieme di implementazioni già pronte all'uso; tuttavia si ritiene utile continuare a mantenere la modlaità standard/semplificata.

Di seguito sono elencati e descritti gli scenari architetturali supportati dalla soluzione per quanto riguarda le problematiche di autenticazione e di autorizzazione.

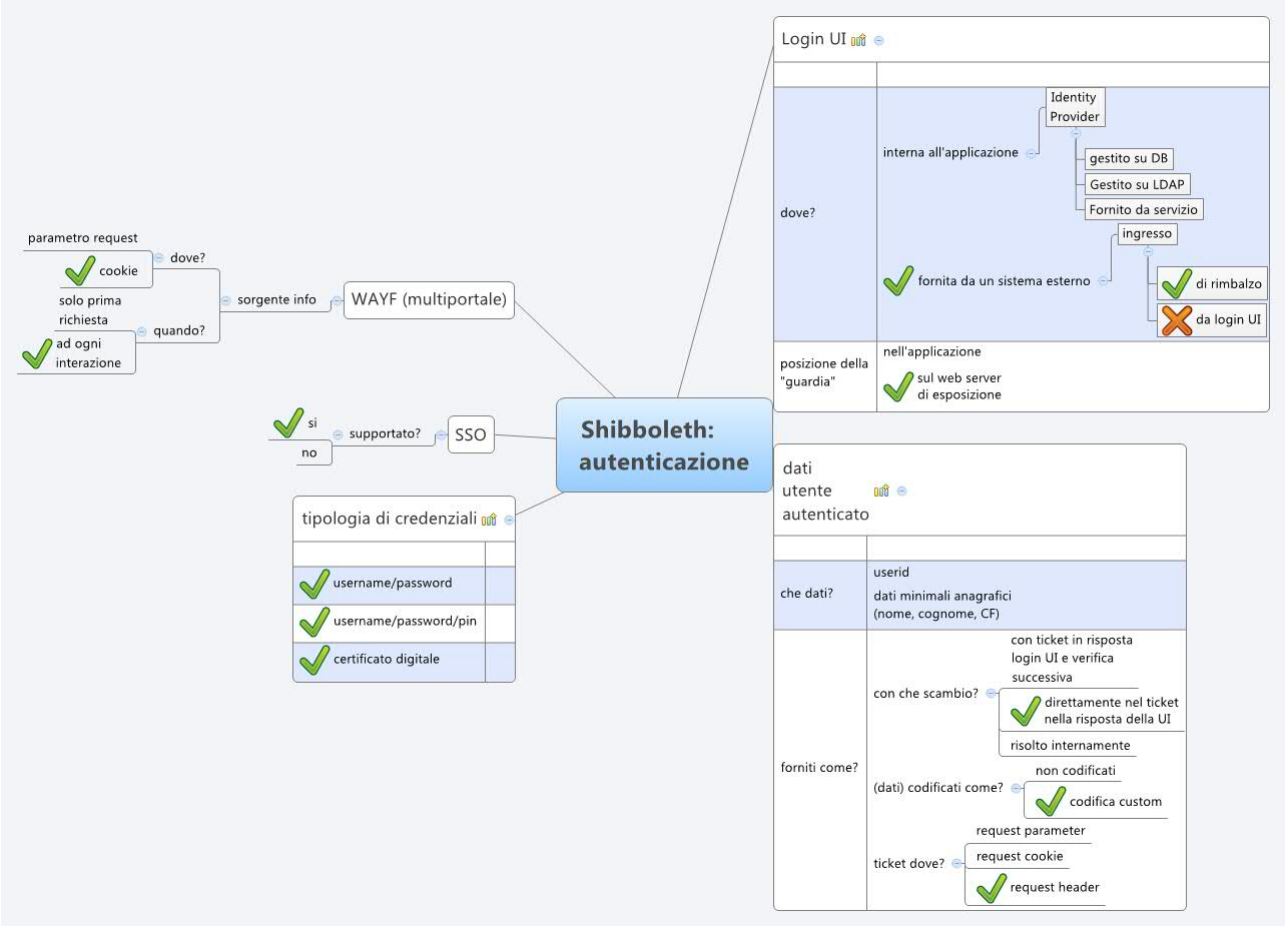
4.1 Scenari di implementazione dell'autenticazione

Le funzioni di autenticazione sono caratterizzate dalle caratteristiche riportate nell'immagine seguente:

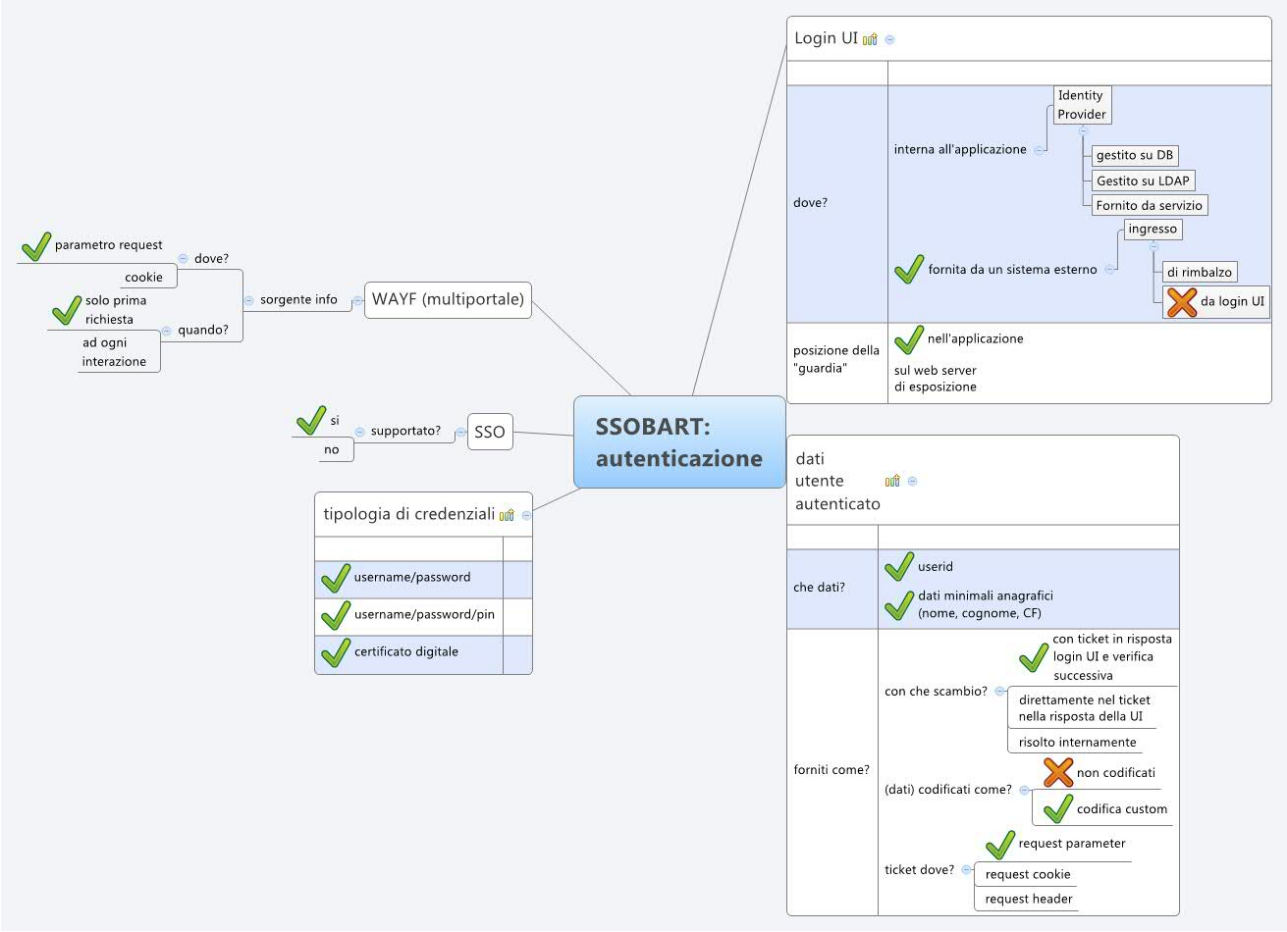


Le foglie del grafo riportato in figura rappresentano in pratica l'insieme delle possibili caratteristiche che un sistema di autenticazione può possedere.
Ovviamente oltre alle caratteristiche discriminative (es. ticket nell'header piuttosto che nei parametri), vi sono poi alcune parametrizzazioni specifiche (es. nome del parametro o dell'attributo dell'header).
A titolo esemplificativo, e come verifica parziale della bontà del modello, si riporta la valorizzazione di tali caratteristiche relativamente ai due principali sistemi di autenticazione CSI (shibboleth e SSOBART) e ad un ipotetica implementazione minimale, locale all'applicativo e basata su uno store delle utenze su DB locale.

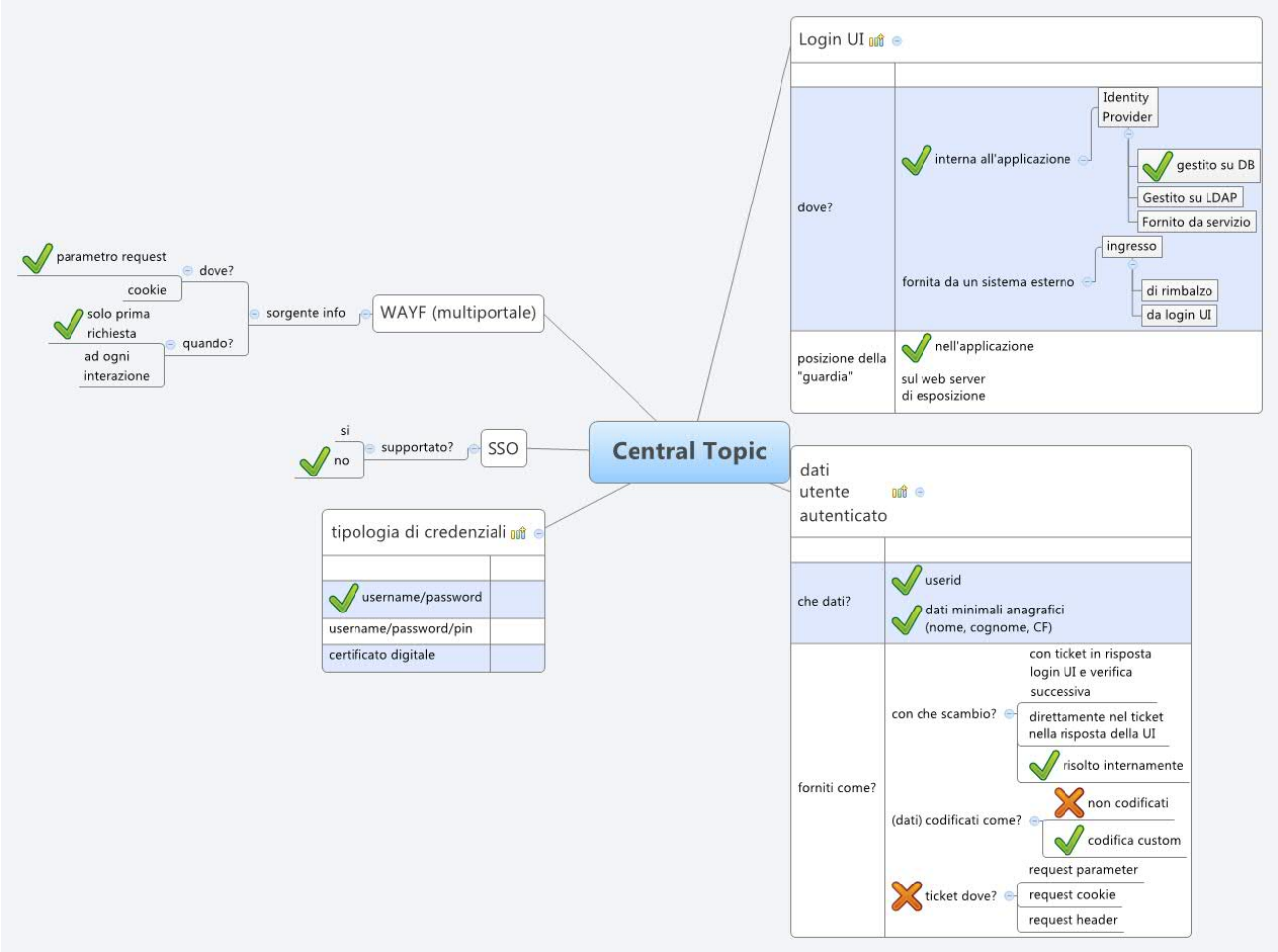
4.1.1 Esempio: Shibboleth



4.1.2 Esempio: SSOBART



4.1.3 Esempio: autenticazione locale basata su DB



4.2 Scenari di implementazione dell'autorizzazione

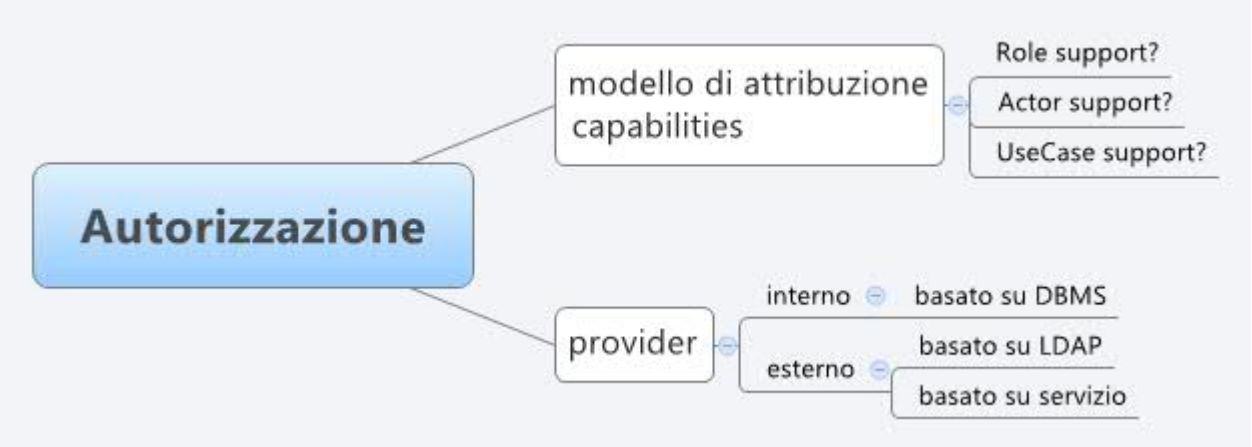
Per quanto riguarda gli scenari di autorizzazione la situazione è meno articolata, in quanto più o meno tutte le implementazioni di autorizzazione sono basati su concetti assimilabili a:

- *Role* (ruolo)
- *Actor* (tipologia di utenza)
- *UseCase* (funzionalità specifica)

Le varianti sono legate alle varie tipologie di provider che risponde alle domande:

- L'utente corrente impersonifica un dato *Actor*?
- L'utente corrente appartiene ad un dato *Role*?
- L'utente corrente è abilitato ad una determinata funzionalità (*Use Case*)?

Le tre entità tipicamente non sono scorrelate (es. un *Role* potrebbe essere collegato ad un insieme di *UseCase*) ma questo aspetto potrebbe essere risolto in modo trasparente dal provider di autorizzazione.



5. Dettagli di design

A fronte dei punti di variabilità individuati nei capitoli 4.1 e 4.2 per l'implementazione della struttura "pluggabile" è necessario esplicitare la struttura del sottosistema di security presente nelle applicazioni generate da guigen. Tale struttura prevede alcuni punti di estensione/configurazione necessari per la realizzazione dei vari scenari.

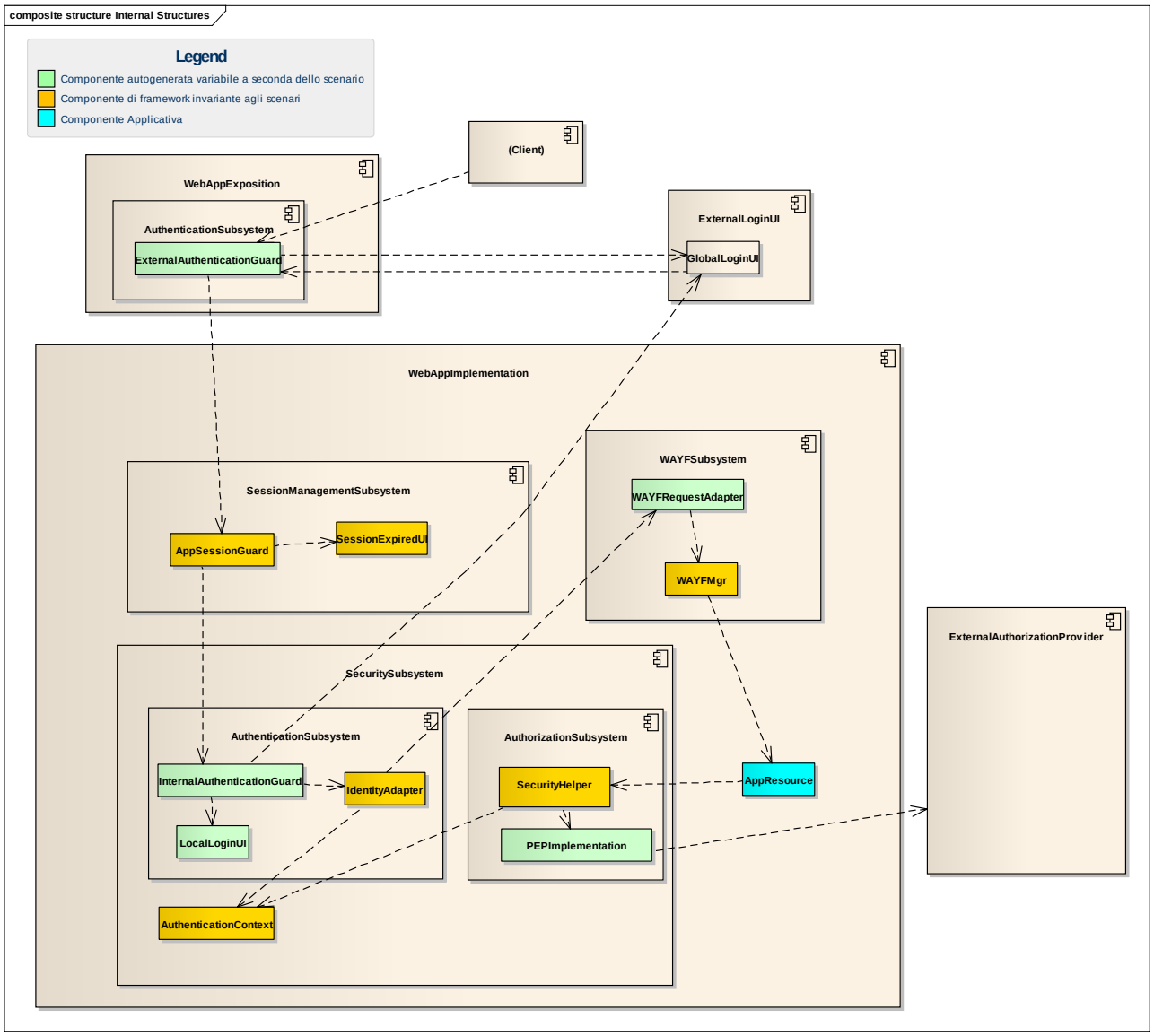


Figura 1 - Struttura a componenti del sistema pluggabile di security di guigen.

L'immagine rappresenta le componenti che entrano in gioco nelle features di autenticazione e autorizzazione di guigen.

Si nota innanzitutto che:

- Alcune componenti sono da considerarsi componenti di framework, con comportamento immutabile a fronte delle variazioni di scenario
- Alcune componenti sono da considerarsi variabili a seconda dello scenario. In taluni casi una componente può anche essere addirittura assente (o ad effetto nullo).
- Le altre componenti sono o applicative o appartenenti ad infrastruttura esterna.

Di seguito sono dettagliati i singoli elementi. E' importante prendere dimestichezza con essi in quanto parte della terminologia utilizzata viene ripresa nelle estensioni al metamodello realizzate per permettere la modellazione della custom security.

5.1 ExternalAuthenticationGuard

- Tipo: framework/variabile
- Responsabilità:
  - verifica che la sessione di autenticazione (e/o di SSO) pe rl'utnet ecorrente sia presente e valida, rimandando alla pagina di login nel caso in cui ce ne sia bisogno. La condizione di "sessione di autenticazione valida" è realizzata tramite un *marker*: la presenza di tale *marker* nella sessione dell'utente[2] significa che la sessione è valida.
- Interazioni:
  - Se la sessione di autenticazione non è valida il flusso applicativo è rimandato alla interfaccia di richiesta credenziali (**GlobalLoginUI**)
  - Se la sessione di autenticazione è valida il flusso applicativo prosegue verso l'**AppSessionGuard**
- Varianti:
  - È possibile la variante che supporta il SSO e quella che non la supporta
- Scenari:
  - Il componente (quando ad azione non nulla) è presente solo nello scenario con **GlobalLoginUI** e solo in presenza di alcune tipologie di sistemi di autenticazione (quelli, come ad es. Shibboleth, che prevedono che sia il web-server a fare la "guardia"). Tipicamente la sua presenza è in alternativa all'**InternalAuthenticationGuard**

5.2 AppSessionGuard

- Tipo: framework/invariabile
- Responsabilità: ha la responsabilità di evitare navigazioni nell'applicazione quando la sessione applicativa non è valida. La sessione applicativa è da ritenersi valida quando il flusso applicativo è passato almeno una volta dal punto di ingresso dopo l'autenticazione.
- Interazioni:
  - nei casi in cui la sessione non sia valida e la navigazione avvenga in un punto intermedio l'effetto è quello di presentare la schermata di cortesia di sessione non valida.
  - Negli altri casi il flusso prosegue verso l'InternalAuthenticationGuard
- Varianti: --
- Scenari: il componente è presente in tutti gli scenari

5.3 SessionExpiredUI

- Tipo: framework/invariabile
- Responsabilità:
  - ha la responsabilità di presentare all'utente un messaggio che lo informi circa la situazione di navigazione incoerente, permettendogli di ritornare alla schermata iniziale.
- Interazioni:
  - È innescato dall'**AppSessionGuard** in caso di sessione non valida
  - Su interazione dell'utente permette di spostare il flusso applicativo verso la schermata iniziale (che rappresenta l'unico punto in cu è possibile spostarsi legalmente se la sessioen applicativa non è valida)
- Varianti: --
- Scenari:
  - il componente è presente in tutti gli scenari, anzi è un elemento presente indipendentemente dallo strato di security.

[2] Non si specifica quale sia il punto in cui viene mantenuto il *marker* di sessione valida; dipende dal tipo di guardia: potrebbe essere mantenuto nella sessione server side, oppure nella sessione del browser, oppure nello strato di webserver, a seconda del tipo di guardia.

5.4 InternalAuthenticationGuard

- Tipo: framework/variabile
- Responsabilità:
  - verifica che la sessione di autenticazione (e/o di SSO) sia valida, rimandando alla pagina di login nel caso in cui ce ne sia bisogno
- Interazioni:
  - Se la sessione di autenticazione non è valida il flusso applicativo è rimandato alla interfaccia di richiesta credenziali (**GlobalLoginUI** o **LocalLoginUI**)
  - Se la sessione di autenticazione è valida il flusso applicativo prosegue verso l'**AppSessionGuard**
- Varianti:
  - È possibile la variante che supporta il SSO e quella che non la supporta
- Scenari:
  - Il componente (quando ad azione non nulla) è presente solo in presenza di alcune tipologie di sistemi di autenticazione (quelli, come ad es. CAS, che prevedono che sia l'application - server a fare la "guardia"). Tipicamente la sua presenza è in alternativa all'**ExternalAuthenticationGuard**

5.5 LocalLoginUI

- Tipo: framework/invariabile
- Responsabilità:
  - ha la responsabilità di richiedere interattivamente all'utente le credenziali di autenticazione
- Interazioni:
  - E' richiamato dall'**InternalAuthenticationGuard**, se la sessione di autenticazione non è valida
  - Al termine delle operazioni di autenticazione ripassa il controllo alla pagina iniziale dell'applicazione
- Varianti: --
- Scenari: il componente è presente solo in congiunzione conl'**InternalAuthenticationGuard** e solo nel caso in cui il meccanismo non preveda una interfaccia di autenticazione esterna all'applicazione.

5.6 IdentityAdapter

- Tipo: framework/variabile
- Responsabilità:
  - ha la responsabilità di trasformare le informazioni relative all'utente autenticato (provenienti dal sistema di autenticazione) nell'oggetto CurrentUser che viene utilizzato per mantenere le informazioni relative all'utente collegato.
- Interazioni:
  - Recepisce le informazioni relative all'utente dagli strati precedenti (**ExternalAuthenticationGuard** o **InternalAuthenticationGuard**) e valorizza la variabile *CurrentUser*
- Varianti:
  - La fonte dell'informazione può essere:
    - Request header
    - Request cookie
    - Request attribute
    - Session attribute
  - La logica di remapping è totalmente custom
- Scenari:

- il componente è fondamentale nell'implementazione della security e pertanto è sempre presente

5.7 AuthenticationContext

- Tipo: framework/immutabile
- Responsabilità:
  - ha la responsabilità di mantenere in memoria le informazioni relative all'utente autenticato provenienti dal sistema di autenticazione nell'oggetto CurrentUser.
- Interazioni:
  - Viene alimentato dall'**IdentityAdapter**
  - Il *CurrentUser* viene interrogato dallo strato di View per il rendering dello UserInfoPanel
  - Il *CurrentUser* viene utilizzato dal **SecurityHelper** per la realizzazione delle verifiche di abilitazione per l'utente collegato.
- Varianti: --
- Scenari:
  - il componente è fondamentale nell'implementazione della security e pertanto è sempre presente

5.8 WhereAreYouFromRequestAdapter

Componente che non fa parte del subsystem di security. E' riportato solo perché:

- Talvolta le informazioni di WAYF sono fornite dal sistema di SSO

5.9 WhereAreYouFromManager

Componente che non fa parte del subsystem di security. E' riportato solo perché:

- Talvolta le informazioni di WAYF sono fornite dal sistema di SSO

5.10 GlobalLoginUI

- Tipo: servizio esterno
- Responsabilità:
  - ha la responsabilità di realizzare la fase di interazione (richiesta credenziali) necessaria per stabilire una sessione di autenticazione valida (sso o non sso).
- Interazioni:
  - Riceve il controllo dall' *AuthenticationGuard (Internal o External) nel caso in cui quest'ultimo non rilevi una sessione di autenticazione valida
  - Al termine della fase di autenticazione, se questa termina con successo, restituisce il controllo all'applicazione, fornendo le informazioni necessarie per individuare l'utente autenticato.
- Varianti: --
- Scenari:
  - il componente è caratterizzante di uno scenario (quello con UI di login globale) pertanto è presente solo in quello scenario

5.11 AppResource

- Tipo: applicativo
- Responsabilità: implementa parte della logica dell'applicativo.
- Interazioni:
  - E' a valle di tutti i moduli del subsystem di security: viene pertanto richiamato quando tutti i passi precedenti hanno successo.

- Utilizza il SecurityHelper per la realizzazione delle funzionalità di mostra/nascondi e abilita/disabilita a fronte di security constraint sui Widget.

- Varianti: --
- Scenari: il componente è sempre presente

5.12 SecurityHelper

- Tipo: framework/immutabile
- Responsabilità:
  - ha la responsabilità di fare da intermediario tra gli strati applicativi e il sottosistema di autorizzazione.
- Interazioni:
  - Riceve le richieste di verifica autorizzazione da parte delle componenti applicative
  - Inoltra le richieste di verifica autorizzazione all'AuthorizationProviderAdapter.
- Varianti: --
- Scenari: il componente è presente in tutti gli scenari

5.13 PEPImplementation

- Tipo: framework/variabile
- Responsabilità: ha la responsabilità di fare da intermediario di adattamento tra il SecurityHelper e un eventuale sistema esterno di autenticazione/autorizzazione. Si espone verso il **SecurityHelper** con l'interfaccia del PEP di IRIDE2, che prevede tutti i metodi necessari per assolvere alle funzioni di verifica credenziali (autenticazione) e verifica abilitazioni (autorizzazione).
- Interazioni:
  - Riceve le richieste di verifica autorizzazione da parte del **SecurityHelper**
- Varianti: --

Scenari: il componente è presente in tutti gli scenari

6. Guida all'utilizzo della custom security in guigen

In questo capitolo si descrivono le attività necessarie per realizzare un meccanismo custom di security con guigen.

6.1 Security profile

La modellazione di un meccanismo custom di security in guigen si basa sulla definizione del modello di un **Security Profile**, nel quale è necessario specificare tutte le caratteristiche del sistema di security, secondo gli elementi introdotti nel capitolo 5 (ovviamente è solo necessario specificare gli elementi variabili: gli elementi fissi – es. AppSessionGuard - non sono oggetto di modellazione).

Le (meta)classi coinvolte nella modellazione del Security Profile sono:

- SecurityProfile
- ExternalAuthenticationGuard
- InternalAuthenticationGuard
- LoginModule
- LoginUI
- IdentityAdapter
- PEPImplementation

Per i dettagli relativi alle varie classi si faccia riferimento alla guida di riferimento del metamodello *guigen*, inclusa nella distribuzione degli MDDTools.

Di seguito si riporta la descrizione di alcuni esempi di modellazione, a titolo esemplificativo.

6.2 Esempi di modellazione di SecurityProfile

6.2.1 Esempio 1: sistema custom di securizzazione totalmente interno all'applicazione

Questo scenario di esempio prevede che la securizzazione dell'applicazione sia realizzata in modo totalmente interno all'applicazione:

- È l'applicazione che richiede le credenziali di autenticazione
- Le credenziali sono verificate dall'applicazione stessa (nell'esempio con una logica fittizia "cablata")
- L'autorizzazione dell'utente collegato è verificata dall'applicazione stessa

6.2.1.1 Modello del *SecurityProfile* custom

Per implementare questo tipo di custom security è necessario definire un SecurityProfile custom, strutturato come segue:

- SecurityProfile
  - *name*: "fullInternal"
  - InternalAuthenticationGuard
    - LoginModule
      - *local*=true
      - LoginUI
        - UIType: USERNAME_PASSWORD
  - IdentityAdapter
    - *sourceName*: "myTicketSource"
    - *infoSourceType*: SESSION_ATTRIBUTE
    - *ticketVerifyMethod*: LOCAL
  - PEPImplementation
    - *custom*: true

In pratica si tratta di un security profile di nome "fullInternal" (è un codice mnemonico a piacere) che possiede:

1. un meccanismo di guardia di autenticazione di tipo interno che, in caso di sessione di autenticazione non valida, rimanda ad una interfaccia di richiesta credenziali anch'essa interna all'applicazione. Nel caso particolare l'interfaccia richiede l'immissione di username e password
2. Un meccanismo custom di preparazione delle informazioni dell'utente collegato (IdentityAdapter); il ticket derivante dalla fase di autenticazione viene memorizzato in un attributo di sessione di nome "myTicketSource".
3. Una implementazione custom del Policy Enforcement Point

6.2.1.2 Generazione e completamento delle logiche custom

A fronte di questo modello di security il generatore genera tutti gli artefatti necessari per l'implementazione delle logiche di protezione dell'applicativo. Per completare il tutto è necessario inserire il codice che implementa le logiche custom nelle apposite regioni protette.

6.2.1.2.1 PEPProvider

Classe: `it.csi.<prod>.<comp>.presentation.<comp>.security.pepprovider.PEPPRoviderFacade`;

Nel PEPProvider è necessario completare la logica dei vari metodi che si occupano di verificare le credenziali dell'utente e di verificare l'abilitazione di un utente secondo criteri basati su Role/UseCase/Actor.
In generale è opportuno completare tutti i metodi di tale classe nei quali sia presente una regione protetta; a titolo esemplificativo si riporta il codice necessario per l'implementazione del metodo di identificazione dell'utente tramite username e password (che è effettivamente necessario nell'esempio corrente, dato il tipo di LoginUI modellato).

```
…
public Identita identificaUserPassword(String user, String password)
                    throws UnrecoverableException, SystemException, InternalException,
                    AuthException, IdProviderNotFoundException,
                    MalformedUsernameException {

              Identita i = new Identita();

              /*PROTECTED REGION ID(R-624812120) ENABLED START*/
              if (user != null && user.equals("pippo")) {
                    i.setCodFiscale("PPIPPI70H17I138F");
                    i.setNome("Pippo");
                    i.setCognome("DEPIPPIS");
                    i.setIdProvider("IPA");
              } else
                    return null;
              /*PROTECTED REGION END*/

              return i;
        }
…
```

Come si può notare la logica implementata è *fake*: l'unico utente riconosciuto è quello che ha lo username = "pippo". L'esempio è comunque utile a mostrare quella che è la *post-condizione* che il metodo deve soddisfare, ovvero la restituzione dell'oggetto *Identita* contenente le informazioni minimali identificative dell'utente collegato (nel caso in cui la verifica delle credenziali abbia successo).
Quest'oggetto è a tutti gli effetti il *ticket* di cui si parla nella modellazione dell'elemento *IDAdapter*: nel prossimo paragrafo si spiega come sfruttare tale *ticket* per materializzare in sessione l'oggetto "*currentUser*", che rappresenta l'utente correntemente collegato.
Giova ancora notare che tipicamente la verifica delle credenziali è effettuata accedendo allo store delle utenze: non vi è limite alla tipologia di tale store (DB? Servizio? LDAP?) in quanto il codice di accesso a tale elemento è custom.

6.2.1.2.2 IDAdapter

Classe: `it.csi.<prod>.<comp>.presentation.<comp>.filter.IDAdapterCustomFilter`

L'IDAdapter ha la responsabilità di trasformare il *ticket* risultante dalla fase di autenticazione in una stringa normalizzata nel formato previsto dalla piattaforma IRIDE2:

`codice_fiscale/nome/cognome/provider/timestamp/livello_auth/mac`[3]

`PPIDPP70H17I138F/Pippo/DEPIPPIS/IPA/1354870914563/1/298`

[3] Queste informazioni non sono tutte necessarie nel caso in cui il sistema di autenticazione non sia effettivamente IRIDE2. E' responsabilità di chi implementa l'integrazione con il meccanismo di sicurezza specifico utilizzare adeguatamente l'oggetto *Identita*.

Nell'esempio il ticket risultante dalla fase di autenticazione in realtà è già l'oggetto *Identita*: pertanto il codice necessario per effettuare questa normalizzazione è particolarmente semplice.

```
private String extractEncodedUserInfo(Object ticket) {
            String res = null;

            /*PROTECTED REGION ID(R-482995998) ENABLED START*/
            res = ticket.toString();
            /*PROTECTED REGION END*/

            return res;
}
```

In altri casi (specialmente nei casi i cui il sistema di autenticazione è esterno) il ticket è un oggetto molto differente e questa fase di normalizzazione assume una complessità maggiore.