Ideazione progetto e Scelte Implementative

Ingegneria del Software

Stefano Ravetta, Simone Renzo, Matteo Scarpone, Mattia Tollari

29 Febbraio, 2016

1 Scelte Implementative

1.1 Introduzione

1.1.1 Informazioni su questa sezione del ducumento

In questa parte di documento verranno trattate e discusse le scelte implementative che sono state prese per la realizzazione del progetto "Team Diary Management" e le motivazioni delle stesse .

1.1.2 Contenuti

Si porrà l'attenzione in modo particolare su

- Linguaggio di programmazione
- Librerie per la creazione dell'interfaccia utente
- Formato e struttura della base di dati
- Librerie usate per la gestione del flusso di dati
- Struttura dei sistemi di sicurezza adottati
- Software per la stesura della documentazione

1.2 Linguaggio di Programmazione

1.2.1 Introduzione

É stato scelto il linguaggio di programmazione Python[1] per alcune caratteristiche piuttosto interessanti che lo possono far risaltare nell'ecosistema dei linguaggi di programmazione odierno.

1.2.2 Alto livello

Python consente di programmare ad un livello molto alto: con poche righe riesce a gestire in modo semplice e diretto operazioni complesse senza che il codice risulti troppo complesso da leggere e comprendere.

Questo ha permesso al team di sviluppo di concentrarsi maggiormente sulle parti più astratte e progettuali (come i design pattern e la rimozione di vari code smell) rispetto a questioni più vicine alla comprensione del funzionamento del linguaggio.

1.2.3 Filosofia

Una parte di un famoso "easter egg"[2] recita:

> If the implementation is hard to explain, it's a bad idea.
>
> If the implementation is easy to explain, it may be a good idea.

Queste righe sembrano denotare con toni sarcastici la filosofia del linguaggio. Uno degli obiettivi del linguaggio è appunto far sì che il codice scritto risulti il miglior compromesso tra stringatezza e comprensibilità. Si intende, quindi, che codice estremamente compatto non è sinonimo di efficienza a lungo termine in quanto provocherà quasi sicuramente un dilatamento delle tempistiche sia in fasi di sviluppo successive, sia in fase di testing ed ottimizzazione. Un esempio che può far notare il modo in cui questo tipo di filosofia inserito nella struttura del linguaggio possa effettivamente influire sul codice è quello dell'indentazione: per forzare il programmatore ad indentare sempre il codice (e a farlo in modo corretto) questo linguaggio non usa caratteri di begin ed end nella definzione di classi, espressioni condizionali, espressioni iterative, funzioni e procedure (come fa ad esempio Java con le parentesi graffe) ma usa solo l'indentazione. In tal modo, da una parte i programmatori che di norma non indentano il loro codice risulteranno "costretti" a farlo, e dall'altra i programmatori che gia' di norma indentano il proprio codice non dovranno inserire altri simboli per esprimere ciò che concettualmente può essere già espresso dall'indentazione.

Secondo il nostro team di sviluppo, questa caratteristica risulta molto interessante per

[1] https://www.python.org/

[2] Questo easter egg, "The Zen of Python" è stato inserito nell'interprete Python e che può apparire appena si prova ad importare la libreria "this".

la gestione di un progetto in cui è richiesta particolare attenzione alla qualità del codice prodotto. Inoltre, a livello pratico, tutto ciò è risultato addirittura comodo perche‘ si è mostrata la necessità di lavorare con uno stile di stesura del codice unico (ogni membro del gruppo ne presentava uno diverso).

1.2.4 Portabilità

Python è un linguaggio precompilato e il suo interprete è disponibile per numerose[3] piattaforme. La presenza di questa caratteristica non è stata sicuramente ignorata dato che quasi tutti i membri del team di sviluppo usano piattaforme ed ambienti diversi. Anche la distribuzione dell'applicazione e di pacchetti contenenti le varie dipendenze risulta estremamente semplificata grazie alla portabilità di Python.

1.2.5 Dizionari

Una delle funzionalità che sono state più apprezzate per la stesura del codice relativa al progetto è senza dubbio la gestione dei dizionari che Python offre e la loro semplice e diretta interazione con classi e anche dati Json.

1.3 Librerie per la creazione dell'interfaccia utente

1.3.1 Introduzione

Sono state scelte le librerie Qt[4] che fanno parte di uno storico framework open source.

1.3.2 Potenzialità

Le librerie Qt sono in grado di gestire interfacce grafiche, basi di dati, attività multimediali, flussi di rete, gestione di file e notifiche di sistema. Una caratteristica piuttosto interessante di queste librerie è che possono essere usate attraverso molti[5] linguaggi di programmazione e non sono legate a nessuna piattaforma specifica. Permettono anche lo sviluppo di applicazioni mobile[6]

1.3.3 Ambienti di sviluppo

Per lo sviluppo dell'interfaccia grafica è stato comodo usare lo strumento QtDesigner[7] (che segue il paradigma WYSIWYG) che permette di disegnare GUI, modificare le proprietà dei vari oggetti e generare codice XML dell'interfaccia realizzata.

---

[3] L'elenco delle piattaforme supportate è reperibile a https://www.python.org/download/other/

[4] http://www.qt.io/

[5] È possibile visionare tutti i linguaggi supportati a questa pagina: http://wiki.qt.io/Category:LanguageBindings

[6] il progetto SailfishOs può dare un'idea delle potenzialità delle librerie Qt: http://sailfishos.org/develop/sdk-overview/

[7] http://doc.qt.io/qt-4.8/designer-manual.html

1.3.4 interazione con Python

Per far interagire il file XML generato da QtDesigner con Python è stato usato lo strumento Pyuic4[8] che svolge il lavoro meccanico di conversione da file che descrive un'interfaccia ad uno script Python che mette a disposizione classi che rappresentano gli oggetti usati ed i metodi per accedervi.

Le librerie Qt sono inoltre state usate sulla parte server per implementare una gestione dei thread.

1.4 Base di dati

1.4.1 Introduzione

Dato che la base di dati usata ha una struttura molto semplice è stato scelto di non usare uno strumento complesso come un database relazionale. È stata creata una base di dati in formato Json che è risultata semplice da strutturare e da gestire. È risultato comodo interagire assieme sulla base di dati tramite Git. L'unico problema riscontrato è stato dover gestire i vari file dal codice dell'applicazione. Questo problema è stato praticamente risolto con l'introduzione di una struttura di metodi e costanti. Con un database relazionale sarebbe stato necessario usare i file ottenuti dal dump del database ed importarli in locale ad ogni modifica. L'ideazione e la gestione del database relazionale avrebbero occupato piú tempo della soluzione adottata.

1.4.2 Struttura

La struttura della base di dati è così definita:

```
database
├── user.json
├── location.json
├── activity.json
├── group.json
└── project.json
```

User.json

Contiene i dati relativi all'utente: id, username, password, nome, cognome, gruppi (una lista che contiene l'id del gruppo e il ruolo ("partecipante" o "team leader"), vacanze (una listas che contiene un id, un nome, data di inizio e data di fine).

Location.json

Contiene i dati relativi ai luoghi: un id ed un nome.

Activity.json

Contiene i dati relativi alle attività: id, nome, id del progetto di riferimento, id dell'utente

[8] http://pyqt.sourceforge.net/Docs/PyQt4/designer.html

che ha creato l'attività, tipo di attività (se di gruppo, di progetto o singola), l'id del luogo, id del grupppo, lista di id dei partecipanti.

Group.json
Contiene i dati relativi ai grupi: id, nome, father (False se il gruppo non è un sottogruppo, id del sottogruppo altrimenti), lista degli id dei sottogruppi.

1.4.3 Interazione con Python

Per importare e gestire la base di dati dal programma sono stati usati i dizionari costruiti, come Json, con [chiave: valore]. Questo tipo di dato ha reso molto agile l'accesso e la generazione di codice Json.

1.4.4 Librerie usate per la gestione del flusso di dati

È stato scelto di usare la libreria Flask[9] che permette una semplice implementazione delle REST Api. È stato scelto questo strumento in quanto risulta avere una curva d'apprendimento leggera. Ha meno funzionalità di altre soluzioni, ma per le il progetto implementato era più che sufficiente. Questo ha comunque reso il pacchetto software finale più leggero e più reattivo.

1.5 Struttura dei sistemi di sicurezza adottati

1.5.1 Codifica delle password

Per evitare di salvare le password degli utenti in chiaro è stato fatto ricorso ad un algoritmo di hashing one-way: esiste una funzione semplice da calcolare che associa ad ogni stringa un hash ma non esiste una funzione semplice da calcolare che associa ad ogni hash una stringa tale che, se applicata la funzione iniziale, l'hash ottenuto sia quello di partenza. È stato scelto SHA512 in quanto oggi risulta quello più robusto a livello di problemi di collisione e preimmagine. Per l'implementazione in Python è stata usata la libreria Hashlib.

1.5.2 Token, autenticazione e gestione delle sessioni

Per gestire autenticazione e sessioni è stato implementato un sistema di token. Un token è un hash SHA512 di utente, password e timestamp. Con un controllo su questi tre campi si impedisce ad un utente malintenzionato (che non dispone dei una password valida) di poter accedere ad una sessione già attiva. Il timestamp serve per imporre scadenze al token (limite attualmente posto uguale ad un giorno). Così facendo risulterà difficile anche provare un attacco che mira a scoprire il token dato che in poco tempo i tentativi precedenti risulteranno inutili.

[9] http://flask.pocoo.org/

1.5.3 Password dimenticate

Nel caso venisse smarrita la password di un utente verrà visualizzato un messaggio che indicherà di contattare gli amministratori di sistema. È stata presa questa scelta perchè non erano presenti le risorse necessarie per implementare e testare a fondo un sistema di recupero password che usi email o SMS.

1.6 Software per la stesura della documentazione

1.6.1 Testi

È stato scelto di usare LaTeXperchè lavorare su file di testo è molto comodo (anche per quanto riguarda l'uso di Git). Inoltre offre numerose funzionalità utili e semplici da gestire per impinazione, gestione paragrafi e adattabilità dello stile.

1.6.2 Diagrammi

Per il disegno dei diagrammi è stato usato il software "Visual Paradigm"; è in grado di strutturare diagrammi di classi, sequenza, attività e stato in modo piuttosto elegante.

2 Ideazione del progetto

2.1 Obiettivi del progetto

Il cliente richiede un sistema di gestione e controllo del flusso di lavoro con relative agende per i progetti di sviluppo software.

Il sistema permette di mantenere aggiornate le attività per ogni singolo progetto, ogni gruppo di lavoro gestito da un Team Leader e ogni sviluppatore che intende pianificare attività individuali.

La gerarchia delle utenze che possono utilizzare il sistema parte dal Project Manager, figura con la responsabilità completa di uno o più progetti, passando per il Team Leader a capo di uno o più gruppi di lavoro all'interno di un progetto fino al singolo sviluppatore, denominato partecipante, con il set più ristretto di operazioni.

2.2 Introduzione a Manageit

La soluzione sviluppata, denominata ManageIT, permette di integrare al meglio le richieste del cliente mantenendo una scalabilità costante ed un alto livello di estensibilità.

Il software si presenta in versione multipiattaforma, utilizzante tecnologie open source e a basso impatto sul sistema.

Le funzionalità implementate comprendono:

- Sistema di login sicuro con crittografia SHA-512
- Gestione integrata dell'agenda di progetto

- Sistema di gerarchia utenza con permessi dedicati
- Notifiche alla modifica di dati in modo distribuito
- Controllo dei vincoli e delle collisioni dei dati

L'usabilità è garantita da un'interfaccia grafica semplice, che presenta tutte informazioni di interesse, dotata di sistemi che evitano operazioni non consentite. Il tutto usa particolari librerie grafiche multipiattaforma dal design piacevole ed uniforme.

Le visualizzazioni principali includono il login iniziale, la presentazione d'agenda generale con varie operazioni effettuabili e le sotto-sezioni dove aggiungere, modificare o cancellare dati. Vegono gestite sia attività individuali, di gruppo o progettuali. Il sistema non integra un automatismo per recuperare le credenziali in modo da prevenire attacchi alla base di dati che dovrà essere direttamente gestita dal Database Manager dell'azienda in questione.

2.3 Analisi dell'Architettura Software

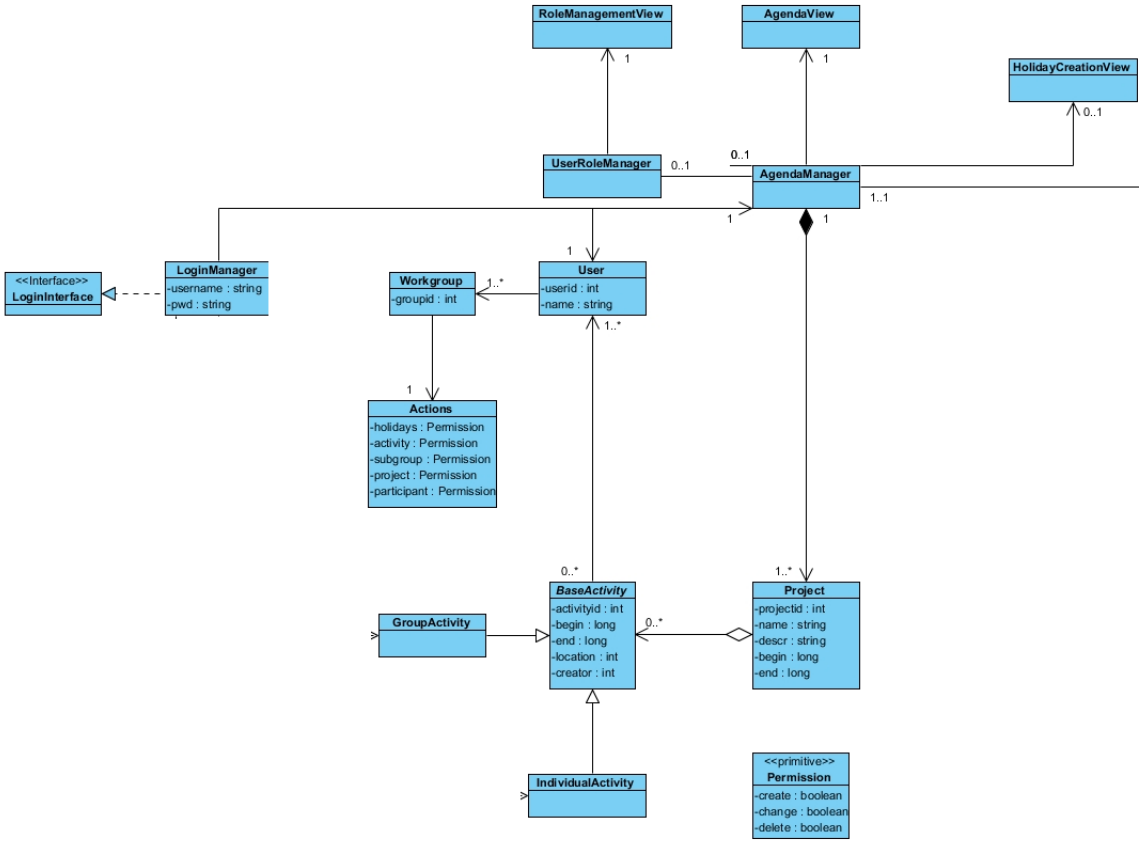
ManageIT è un sistema basato su interazione Client - Server, strutturato in modo da poter lavorare anche in modalità offline.

La parte server è ampliamente configurabile ed estendibile e non è dipendente da base di dati e sistema di login. L'interfaccia è resa inoltre indipendente dal sottosistema di controllo, permettendo così un mantenimento più veloce e una migliore adattabilità alle possibili esigenze future.

2.4 Design Patterns

2.4.1 Observer

Nella progettazione dell'interfaccia grafica sono stati sfruttati componenti ad eventi: il sottosistema inizializza la GUI e si registra come ascoltatore di alcuni eventi della stessa attivando quindi determinati metodi solo all'utilizzo di tali componenti.

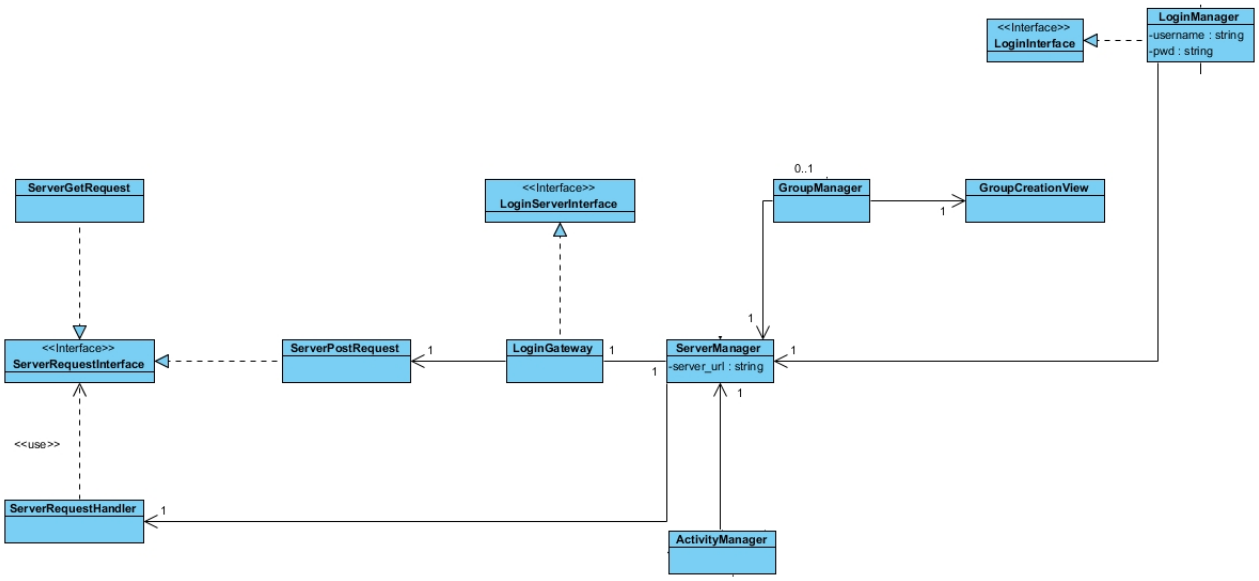


Il vantaggio di questo Design Pattern risiede nell'evitare eventi periodici di controllo della interfaccia grafica mediante polling, risparmiando quindi grandi quantità di risorse di calcolo e mantenendo più leggero e reattivo il software sviluppato.

A livello di accoppiamento inoltre il gestore sottostante conosce la parte di GUI a cui è collegato ma non avviene il contrario, permettendo così la sostituzione veloce di parti di interfaccia in base alle esigenze. Inoltre se necessario più classi possono ascoltare eventi dalla stessa sezione di GUI senza dover modificare la stessa che diffonde i suoi eventi a tutti gli ascoltatori.

2.4.2 Facade

Il Design Pattern denominato Facade è stato utile per dividere in modo efficace i singoli sottosistemi dei package individuati permettendo una facile comunicazione tra di loro senza aumentare il numero di accoppiamenti tra classi diverse.



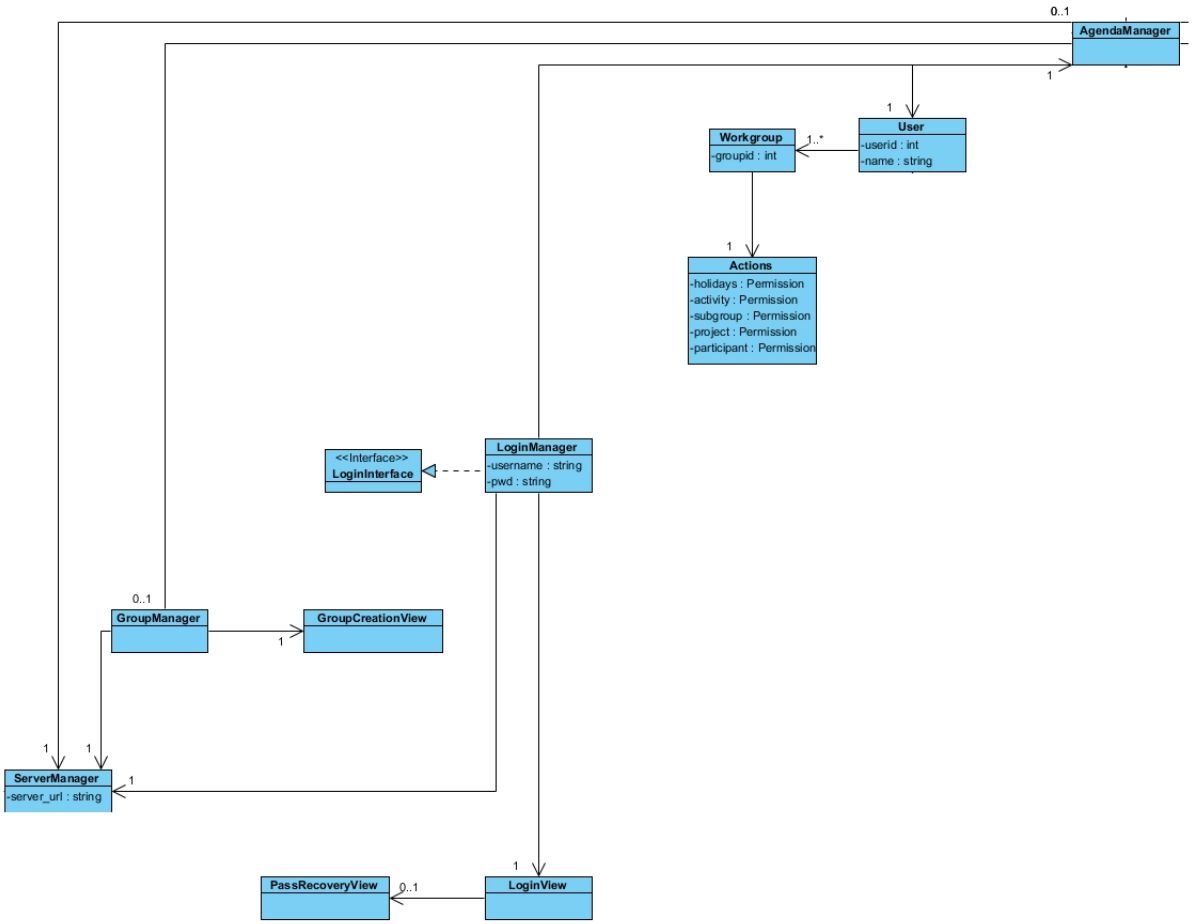
Nel progetto in questione il pattern è stato sfruttato in modo semplificato in tutti i package ad esclusione della GUI dove non è previsto. In particolare le chiamate ai package destinati al networking o allo storage dei dati vengono interamente filtrate e smistate da classi dedicate che sono appunto le Facade individuate.

Tra i vantaggi di questo pattern troviamo sicuramente una riduzione dell'accoppiamento tra classi, un maggiore riutilizzo di un sottosistema o package del sistema che può quindi essere interfacciato facilmente con altri sistemi senza reimplementare intere classi ed inoltre non esclude la possibilità all'utente di accedere direttamente al sistema "nascosto" che rimane a tutti gli effetti visibile ed usabile nel modo che si ritiene opportuno.

2.5 Architectural Patterns

2.5.1 Remote Facade

Il pattern in questione rientra nella categoria architetturale e può essere visto come una estensione del precedente Facade applicato però in ambito di networking. In questo caso la classe che funge da Remote Facade permette di aggregare più operazioni che necessitano di connessione di rete per essere eseguite riducendo così l'overhead, il traffico, la congestione e la latenza della richiesta stessa.



Il funzionamento del Remote Facade comprende il ricevere dati da diverse classi o da diversi attributi/metodi di una classe, raggrupparli in un unico metodo che creerà la richiesta web unificata senza inviarli singolarmente. Nel progetto è stato utilizzato in diverse casistiche tra cui il sistema di login oppure le operazioni che comprendono creazione di contenuti. Il pattern inoltre viene comunemente utilizzato, seppur in modo diverso, nella progettazione di API RESTful che permettono di far risaltare al meglio questo tipo di pattern.

2.5.2 Gateway

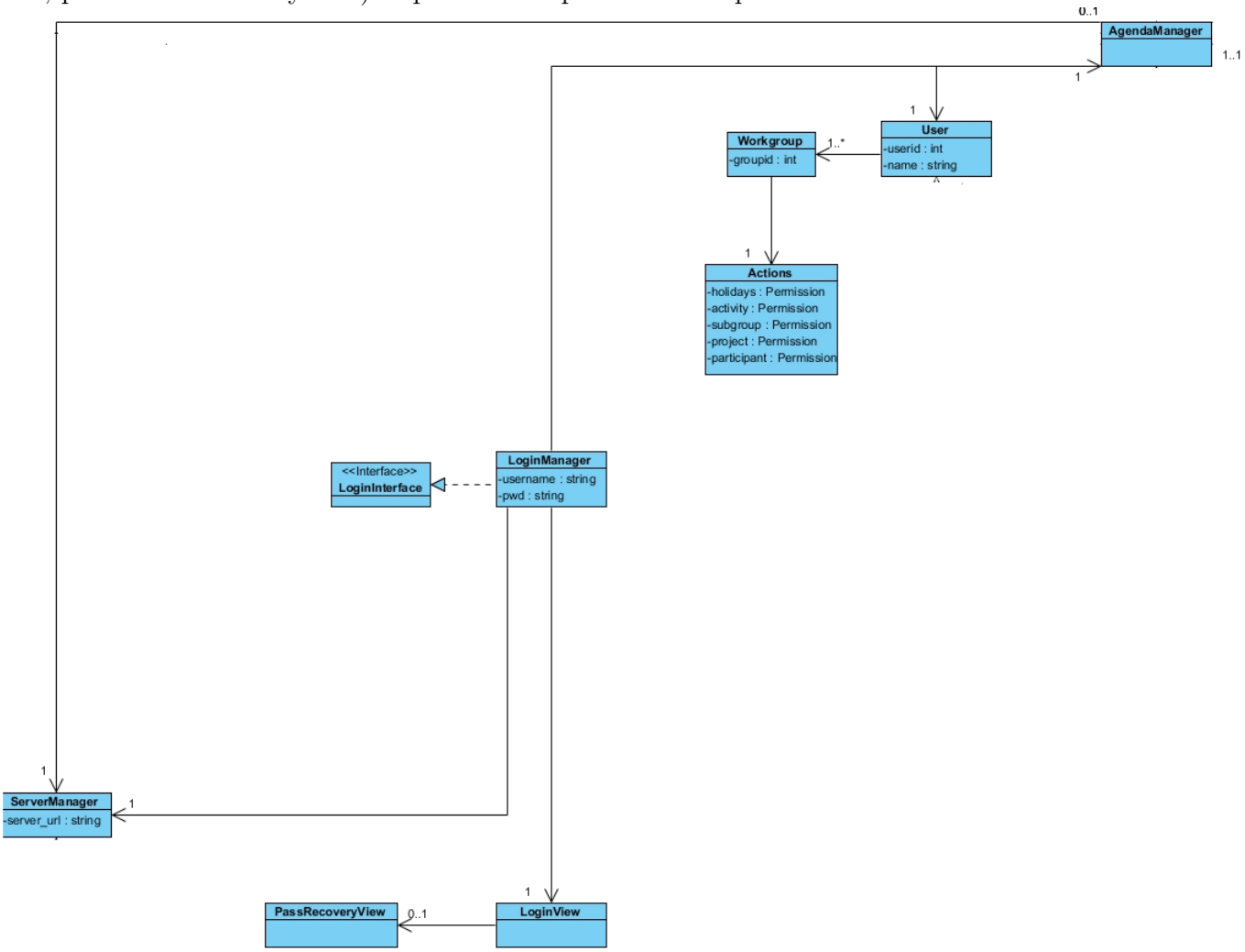
Durante la creazione di classi Facade ci si può confondere con un pattern architetturale denominato Gateway che sostazialmente fornisce un sistema simile ma risolve un problema affine. Infatti il pattern Gateway permette di semplificare l'accesso ad una determinata interfaccia impacchettando le operazioni necessarie ad effettuare quella di interesse.

Un altro vantaggio da non sottovalutare risiede nella semplicità di testing di una classe Gateway rispetto ad utilizzare una interfaccia complessa o più classi correlate, rendendo

quindi il controllo del software più veloce ed efficace. Il sistema sfrutta questo pattern nell'interfacciamento con il sistema di Database sottostante, essendo sviluppato in modo da semplificare le query ed aggregarle in semplici operazioni richiamabili dalle classi che ne necessitano l'accesso.

2.5.3 Separated Interface

Durante la progettazione del sistema software ci si è concentrato sullo sviluppare una architettura che potesse supportare al meglio qualsiasi tecnologia utilizzata riguardo i sistemi di Login o di storage di dati in Database. Per separare l'implementazione dalla interfaccia per utilizzarla viene in aiuto il pattern Separated Interface che prevede la definizione di una interfaccia comune e separata in un altro package (non nel nostro caso, per la natura di Python) implementata poi nel modo più conveniente.



In particolare il sistema di Login dell'applicativo può essere personalizzato a piacimento in base alle richieste di sicurezza o per semplificare l'accesso. Lo stesso vale per i sistemi di Database che possono variare molto in base al tipo di dati da salvare, alla mole di dati o alle preferenze dell'utente, generalizzando una interfaccia d'accesso alle

operazioni possiamo quindi suddividere il sistema di storage dal suo utilizzo che rimane invariato dall'esterno.