Object Design Document

KeyMaster

Riferimento	
Versione	2.0
Data	21/01/2022
Destinatario	Prof.ssa F. Ferrucci
Presentato da	Andrea Santaniello, Mattia d'Argenio, Daniele Dello Russo, Michele Corcione
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
17/12/2021	0.1	Prima stesura	Tutti
30/01/2021	1.0	Stesura finale	Tutti
20/01/2022	2.0	Aggiunta di UML per Design Patterns	Tutti

Sommario

1. Introduzione

1.1. Object design trade-offs

Durante la fase di Object Design sono sorti diversi compromessi di progettazione. Di seguito sono riportati i trade-off:

- **Tempo di risposta vs Affidabilità**: Il sistema sarà implementato in modo tale da preferire il tempo di risposta per garantire un servizio sempre disponibile anche con grandi carichi di lavoro.
- **Usabilità vs Funzionalità**: Il sistema sarà facilmente usufruibile dall'utente perché l'interfaccia web realizzata sarà intuitiva.
- **Leggibilità vs Tempo**: L'obiettivo sarà scrivere codice che rispetti lo standard per la programmazione con il linguaggio java, con l'aggiunta di commenti per eventuali chiarimenti. Questo favorirà la leggibilità ed agevolare il processo di mantenimento e modifica del codice. Tuttavia, questo vantaggio aumenterà il tempo necessario per lo sviluppo e la realizzazione dell'intero sistema.
- **Sicurezza vs Costi**: La sicurezza rappresenta uno degli aspetti principali del sistema. Implementeremo un sistema di autenticazione tramite username e password ed SSL (Secure Sockets Layer) creando un link sicuro tra il sito e il browser client.
- **Sviluppo rapido vs Features**: Le funzionalità specifiche dell'applicazione verranno realizzate seguendo un sistema basato su delle priorità. Privilegiando uno sviluppo rapido, verrà data la precedenza agli elementi che dispongono di una priorità alta per poi integrare le restanti funzionalità in un secondo momento.

1.1.1. Componenti off-the-shelf

KeyMaster farà affidamento su varie componenti off-the-shelf.
Ciò nasce dalla necessità di fornire un prodotto funzionale in tempi stringenti, tenendo comunque in considerazione che il budget economico e le risorse umane sono limitate.

Per quanto riguarda il front-end verrà utilizzata la libreria di Bootstrap.
Bootstrap è un framework che contiene modelli di progettazione basati su CSS e JavaScript che agevolerà la realizzazione di una parte grafica dell'applicazione che sia responsive e ben strutturata, garantendo la stessa qualità nell'esperienza di utilizzo indipendentemente dall'hardware (ad esempio le dimensioni dello schermo del device o la modalità di interazione con l'applicazione) e dal software (browser e sistema operativo).

Lato back-end saranno invece utilizzate le seguenti componenti off-the-shelf: MySQL, Tomcat, Maven, NginX.

MySQL è un celebre RDBMS di Oracle. Il suo utilizzo sarà fondamentale per la realizzazione, la gestione e l'implementazione della base di dati di KeyMaster. Tutti i dati che per necessità di business devono essere salvati persistentemente saranno affidati ad un database basato su MySQL. Sarà inoltre utilizzata la componente MySQL Connector, il driver JDBC ufficiale per MySQL necessario per la comunicazione con la base di dati da Java, e il driver C3P0 per la Connection Pool allo stesso database.

Tomcat è un server web open-source sviluppato dalla Apache Software Foundation. Il suo impiego è necessario, poiché tutto il back-end sarà eseguito in ambiente Tomcat.

Maven è uno strumento di gestione dei progetti software basati su Java, anch'esso sviluppato dalla Apache Software Foundation. Agevolerà la gestione delle varie librerie utilizzate, delle loro versioni e delle dipendenze tra esse, questo permette agli sviluppatori di concentrarsi esclusivamente sulla scrittura del codice.

NginX è un web server open source impiegato nel sistema come reverse proxy, per gestire il bilanciamento del carico di lavoro, ed ottenere minor tempo di risposta grazie alla memorizzazione delle cache e alla distribuzione dei file statici.

1.2. Linee guida per la documentazione dell'interfaccia

È richiesto agli sviluppatori di seguire le seguenti linee guida al fine di essere consistenti nell'intero progetto e facilitare la comprensione delle funzionalità di ogni componente.

Tipi	Regole per la denominazione	Esempi
Package	Il prefisso di un nome di pacchetto univoco sempre scritto in minuscolo e tutte le lettere ASCII minuscole.	Package hf.keymaster.api; Package hf.keymaster.database;
Classi	I nomi delle classi dovrebbero essere sostantivi, scritti in CamelCase. Abbiamo mantenuto i nomi delle	class Application; class ApiResponse;

	nostre classi semplici e descrittivi. Tuttavia, abbiamo utilizzato parole intere ed evitato per quanto possibile acronimi e abbreviazioni.	
Metodi	I nostri metodi sono verbi, scritti nella forma camelCase.	createApplication(); updateLicense();
Variabili	I nomi delle variabili sono brevi ma significativi. La scelta del nome di una variabile è mnemonica, cioè finalizzata a indicare all'osservatore casuale l'intento del suo utilizzo. I nomi delle variabili composte da una sola lettera vengono evitati tranne che per le variabili "usa e getta" temporanee.	int userID; User u; Int I;

1.3. Definizioni, acronimi e abbreviazione

Termine	Definizione
Epoch	Unix Timestrap

Acronimo	Definizione
ID	Identification

Abbreviazione	Definizione
App	Application
ow	Owned
lic	License

1.4. Riferimenti

KM_RAD_V_2

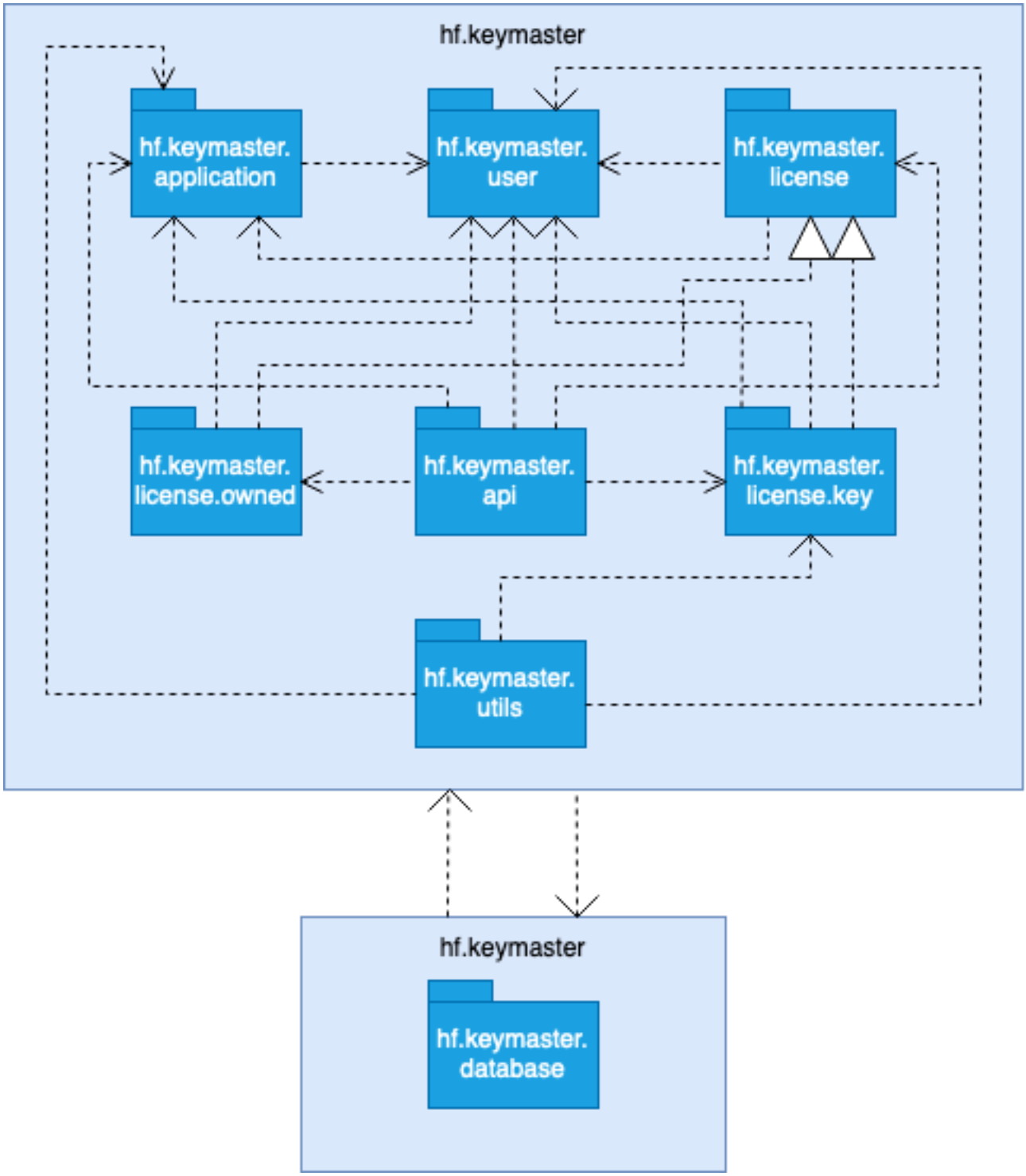
KM_SDD_V_2

2. Packages

In questa sezione viene presentato in modo più approfondito quella che è la divisione in pacchetti e l'organizzazione del codice in file.

2.1. Divisione in pacchetti

Il sistema è stato diviso in un discreto numero di pacchetti, per ognuno dei quali sono stati realizzate le opportune classi ed inserite nell' apposito pacchetto. Questa divisione nasce dall'esigenza di accorpare nello stesso pacchetto le classi che sono legate da un'alta coesione e ridurre al minimo quelle che sono le dipendenze tra i pacchetti che tuttavia sono ugualmente presenti. Di seguito è riportato uno schema generale della vista dei pacchetti del sistema, e le loro dipendenze:


hf.keymaster
hf.keymaster.
application
hf.keymaster.
user
hf.keymaster.
license
hf.keymaster.
license.owned
hf.keymaster.
api
hf.keymaster.
license.key
hf.keymaster.
utils
hf.keymaster
hf.keymaster.
database

Si evidenzia che i pacchetti "hf.keymaster.license.owned" e "hf.keymaster.license.key", sono generalizzazioni del pacchetto "hf.keymaster.license" e che il database ha dipendenze con tutto il dominio del sistema.

Nome Pacchetto	hf.keymaster.user
Descrizione	Definisce le proprietà dell'utente generico e di quello sviluppatore della piattaforma ed espone i servizi di autenticazione, logout, e ottenimento dell'autenticazione
Dipendenze	hf.keymaster.database

Nome Pacchetto	hf.keymaster.application
Descrizione	Definisce le proprietà delle applicazioni, ed espone i servizi di visualizzazione, creazione, modifica, e revoca delle informazioni sulle stesse
Dipendenze	hf.keymaster.database hf.keymaster.user

Nome Pacchetto	hf.keymaster.license
Descrizione	Definisce le proprietà delle licenze, ed espone i servizi di visualizzazione, creazione e modifica delle informazioni sulle stesse
Dipendenze	hf.keymaster.database hf.keymaster.user hf.keymaster.application

Nome Pacchetto	hf.keymaster.license.owned
Descrizione	Definisce le proprietà delle licenze possedute, ed espone i servizi di visualizzazione delle informazioni sulle stesse

Dipendenze	hf.keymaster.database hf.keymaster.user hf.keymaster.license.key hf.keymaster.license

Nome Pacchetto	hf.keymaster.license.key
Descrizione	Definisce le proprietà delle chiavi associate alle licenze, ed espone i servizi di visualizzazione e modifica delle stesse
Dipendenze	hf.keymaster.database hf.keymaster.user hf.keymaster.application hf.keymaster.license

Nome Pacchetto	hf.keymaster.api
Descrizione	Definisce l'implementazione delle classi per la comunicazione con il gli applicativi e la piattaforma
Dipendenze	hf.keymaster.database hf.keymaster.license.owned hf.keymaster.license hf.keymaster.application hf.keymaster.user hf.keymaster.license.key

Nome Pacchetto	hf.keymaster.database
Descrizione	Definisce l'implementazione delle classi per la comunicazione con la base dati

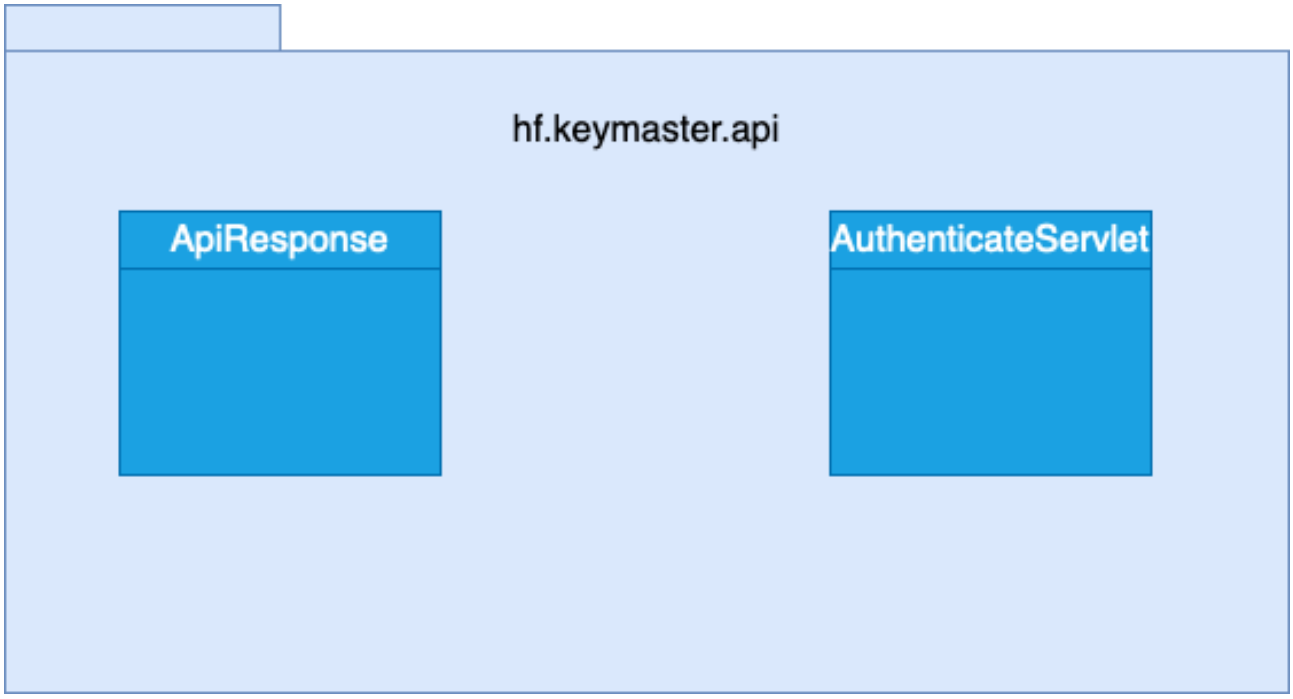
Nome Pacchetto	hf.keymaster.utils
Descrizione	Definisce l'implementazione delle classi utili per il pieno funzionamento del sistema

Dipendenze	hf.keymaster.key hf.keymaster.application hf.keymaster.user hf.keymaster.database

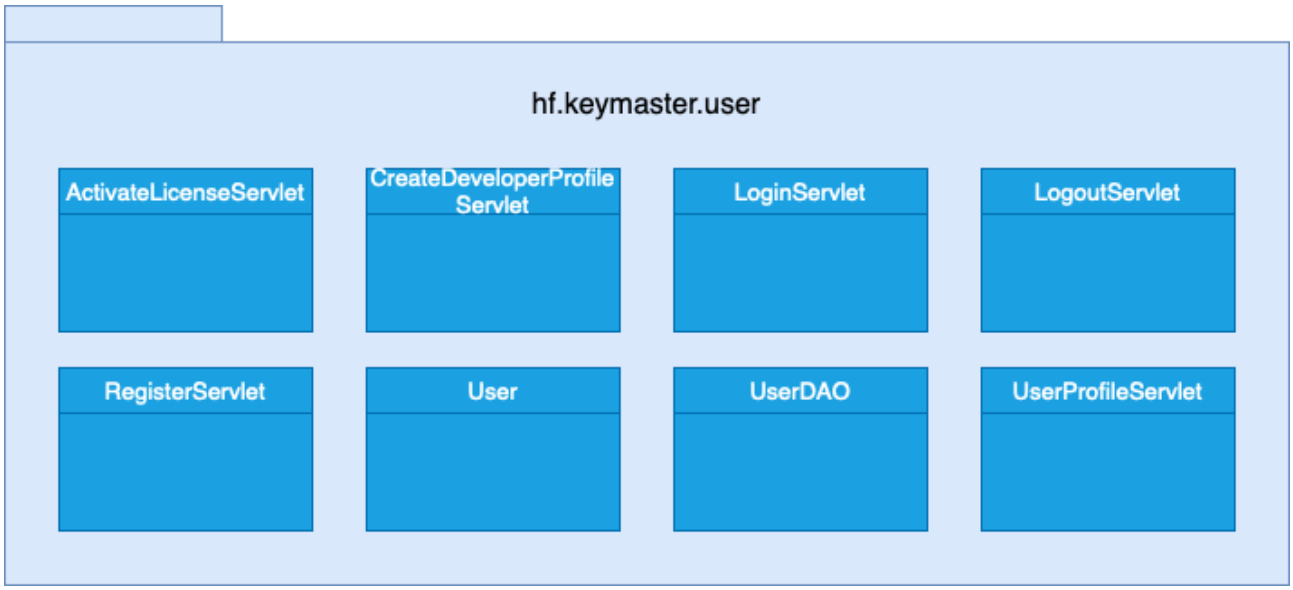
Organizzazione del codice in file

Come imposto da Java, ogni classe del sistema sarà collocata nel relativo file. Ognuno di essi sarà quindi collocato nel pacchetto dedicato. I nomi dei pacchetti sono autodescrittivi, forniscono immediatamente l'idea della loro funzione all'interno del sistema, questa scelta è stata attuata allo scopo di rendere il sistema mantenibile nel caso si vogliano apportare modifiche oppure aggiungere nuove funzionalità: basterà modificare/aggiungere le nuove componenti nell'apposito pacchetto. I pacchetti avranno tutti come prefisso "hf.keymaster" (e saranno mappati nel rispettivo percorso src/main/java/…) mentre l'insieme delle pagine Web dinamiche che compongono il sito sarà collocato nella directory "src/main/webapp/skeletons/pages". Di seguito è riportato nel dettaglio la struttura in file dei pacchetti:

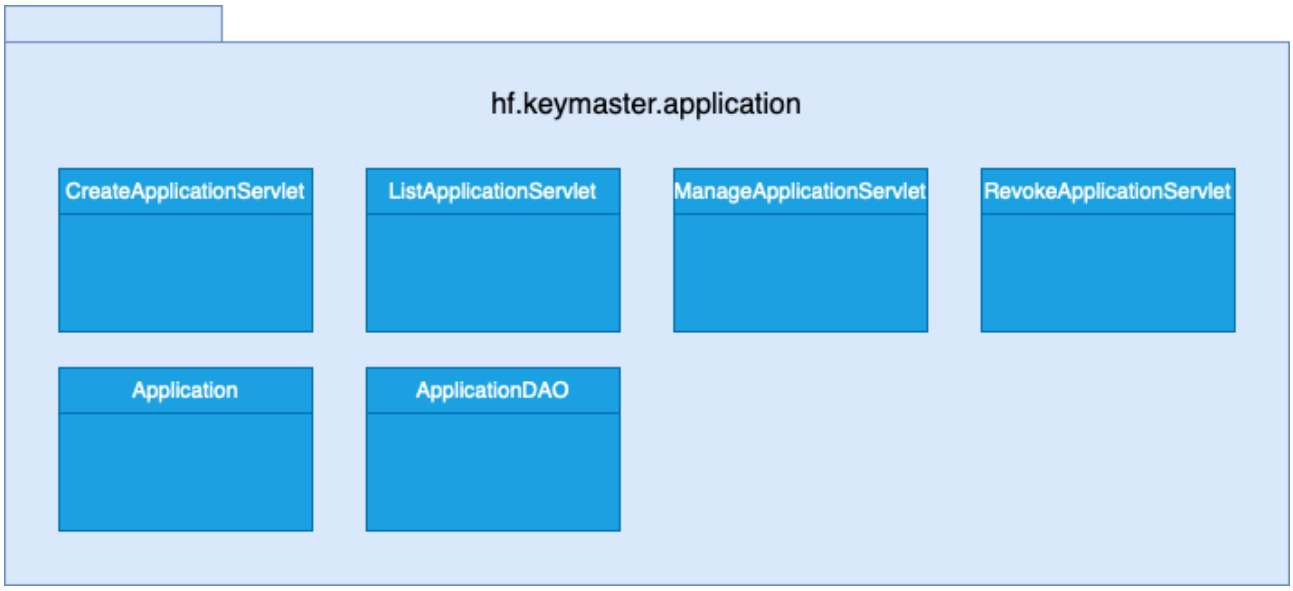
Pacchetto "hf.keymaster.api"



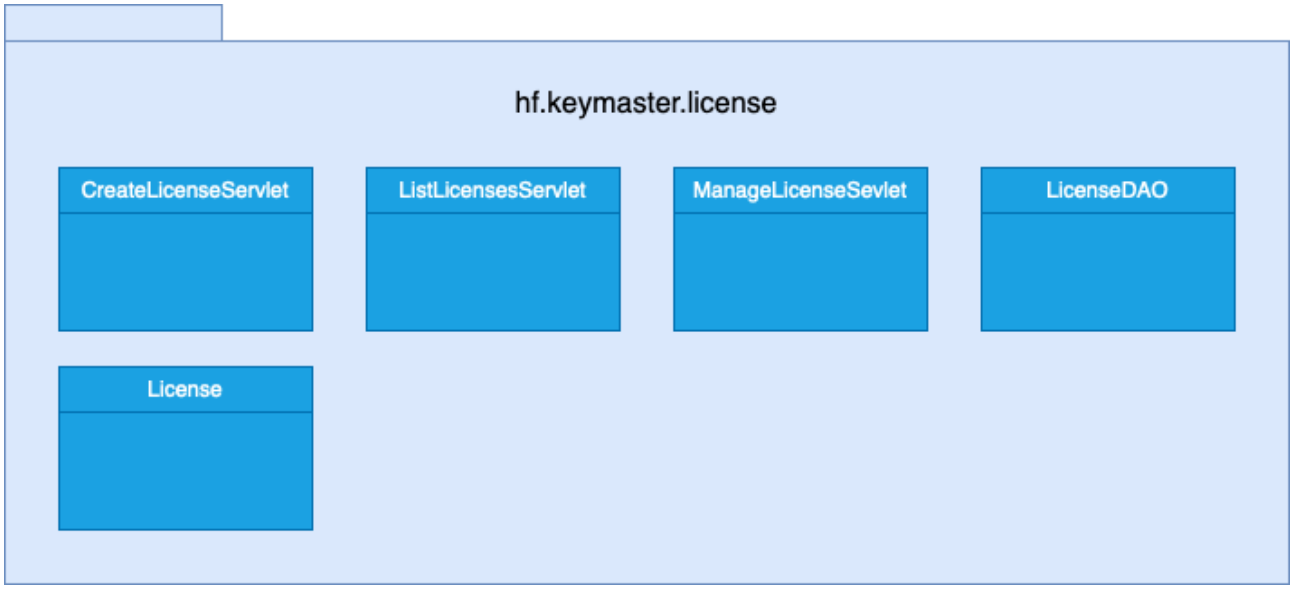
Pacchetto "hf.keymaster.user"



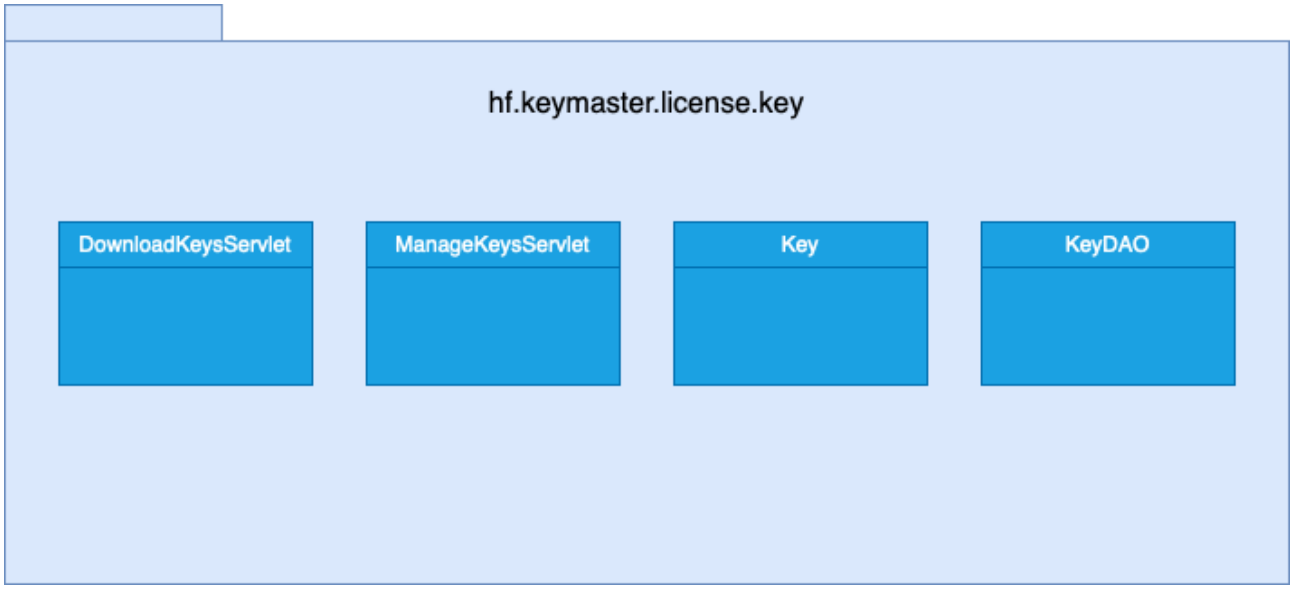
Pacchetto "hf.keymaster.application"



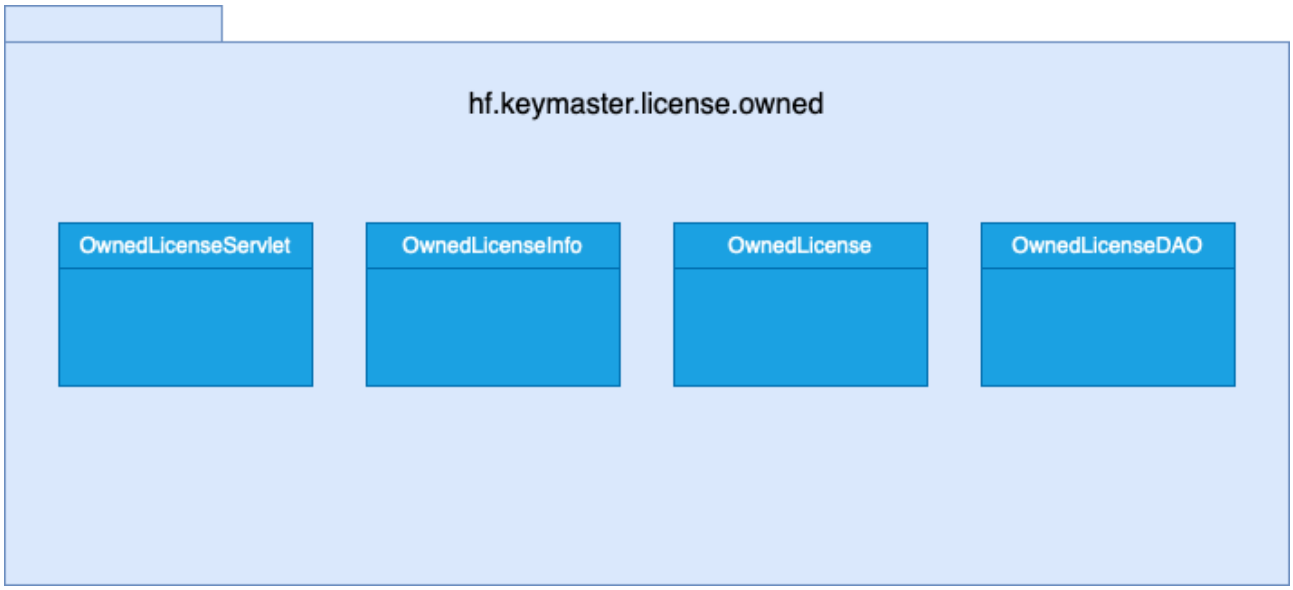
Pacchetto "hf.keymaster.license"



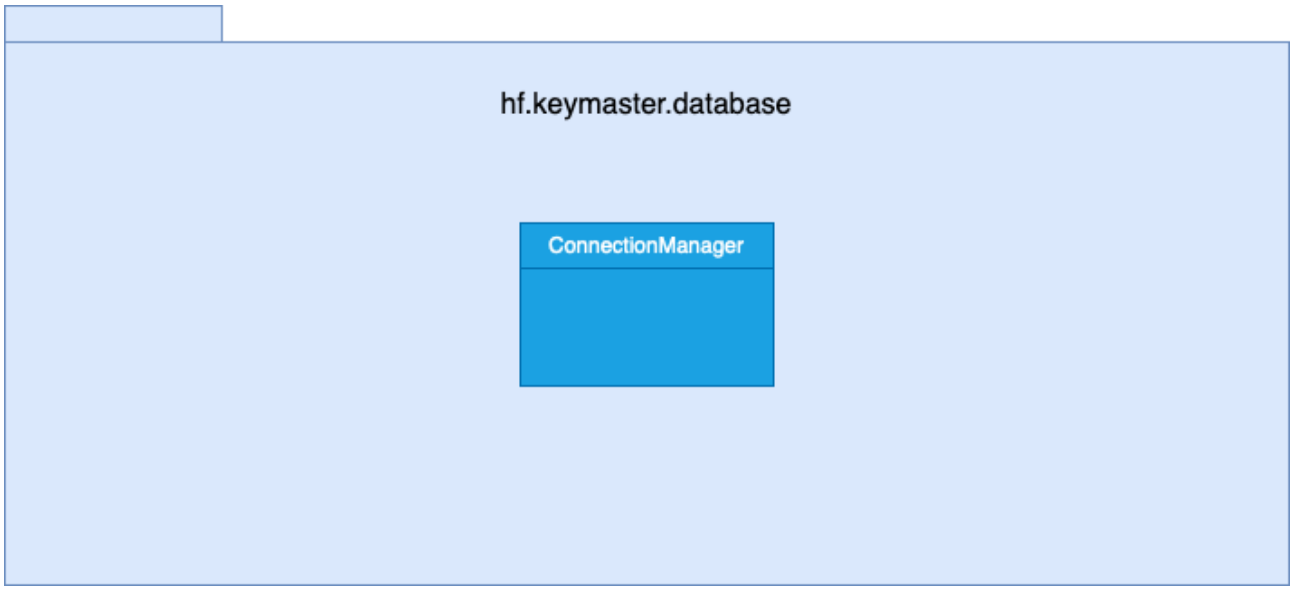
Pacchetto "hf.keymaster.license.key"



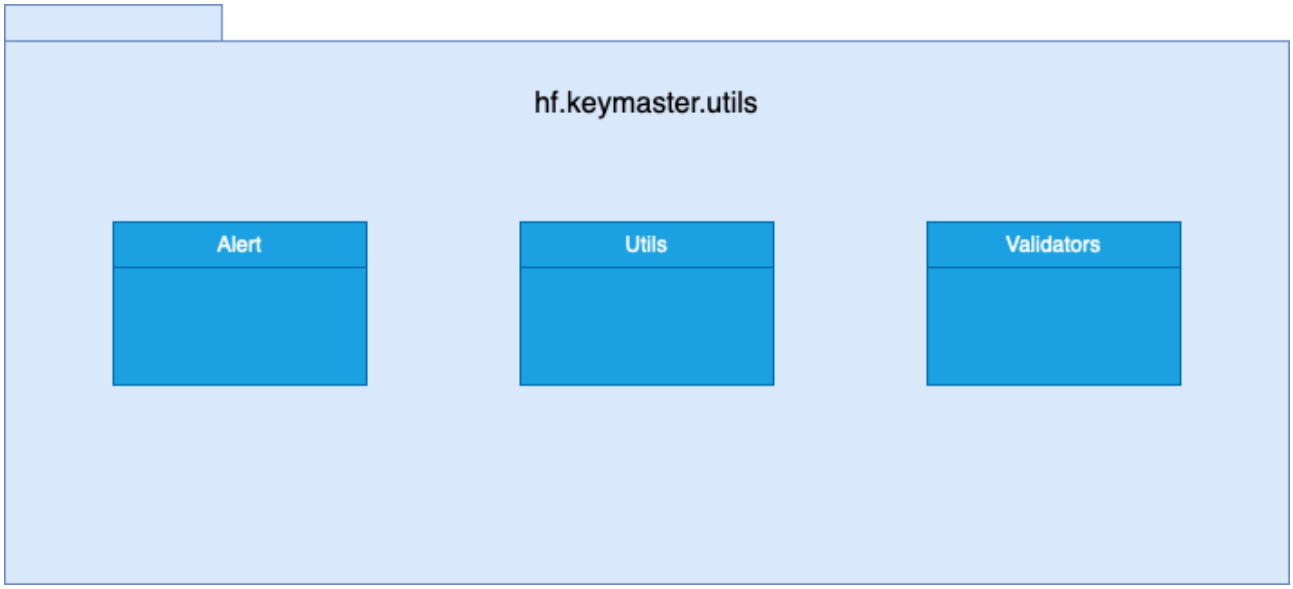
Pacchetto "hf.keymaster.license.owned"



Pacchetto "hf.keymaster.database"



Pacchetto "hf.keymaster.utils"



3. Interfacce delle classi

La documentazione relativa all'interfaccia pubblica delle classi è reperibile nei vari file Javadoc presenti in javadoc/.

Tali documenti presentano definizione degli eventuali parametri in input dei metodi e descrizione di eventuali parametri output oltre ad una breve spiegazione sul funzionamento dei metodi.

4. Design Pattern con class diagram

Design patterns

Nello sviluppo del sistema abbiamo deciso di utilizzare dei design patterns per evitare di rielaborare soluzioni già precedentemente fornite da altri programmatori.

I design patterns utilizzati sono:

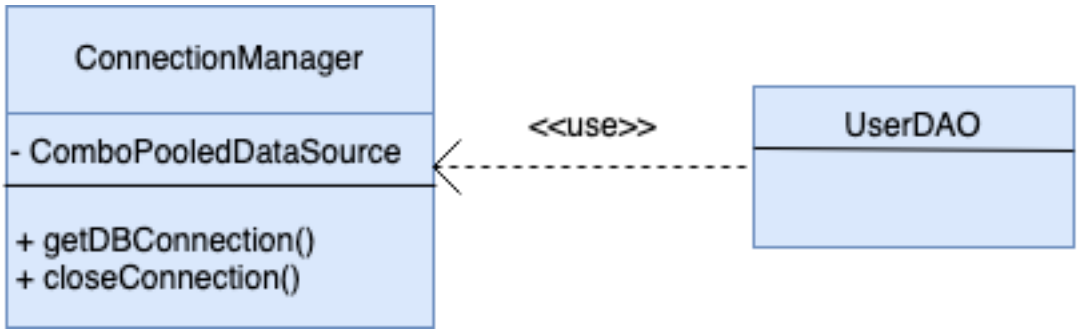
- ObjectPool per garantire velocità nella generazione di connessioni al database
- MVC (Model -View - Control) per la realizzazione del sistema
- (Static) Factory per garantire bassa coesione tra le classi
    - **Object Pool**

Nome del design pattern: Object Pool

Descrizione del problema: siccome ci aspettiamo più richieste da diversi utenti non possiamo aspettarci che il nostro sistema sia in grado di aprire nuove connessioni per ogni utente e rispondere alle richieste in un tempo accettabile.

Soluzione del problema: ogni volta che un utente richiede una connessione al database il sistema controlla che ci siano connessioni già aperte in un pool di connessioni. Se il sistema ritorna vero, viene utilizzata una connessione già aperta, altrimenti ne apre una nuova. Quando la connessione viene rilasciata essa torna nel pool delle connessioni aperte, in attesa di essere utilizzata.

Conseguenze: il sistema è in grado di rispondere alle richieste dell'utente più velocemente poiché non ha sempre bisogno di aprire nuove connessioni.



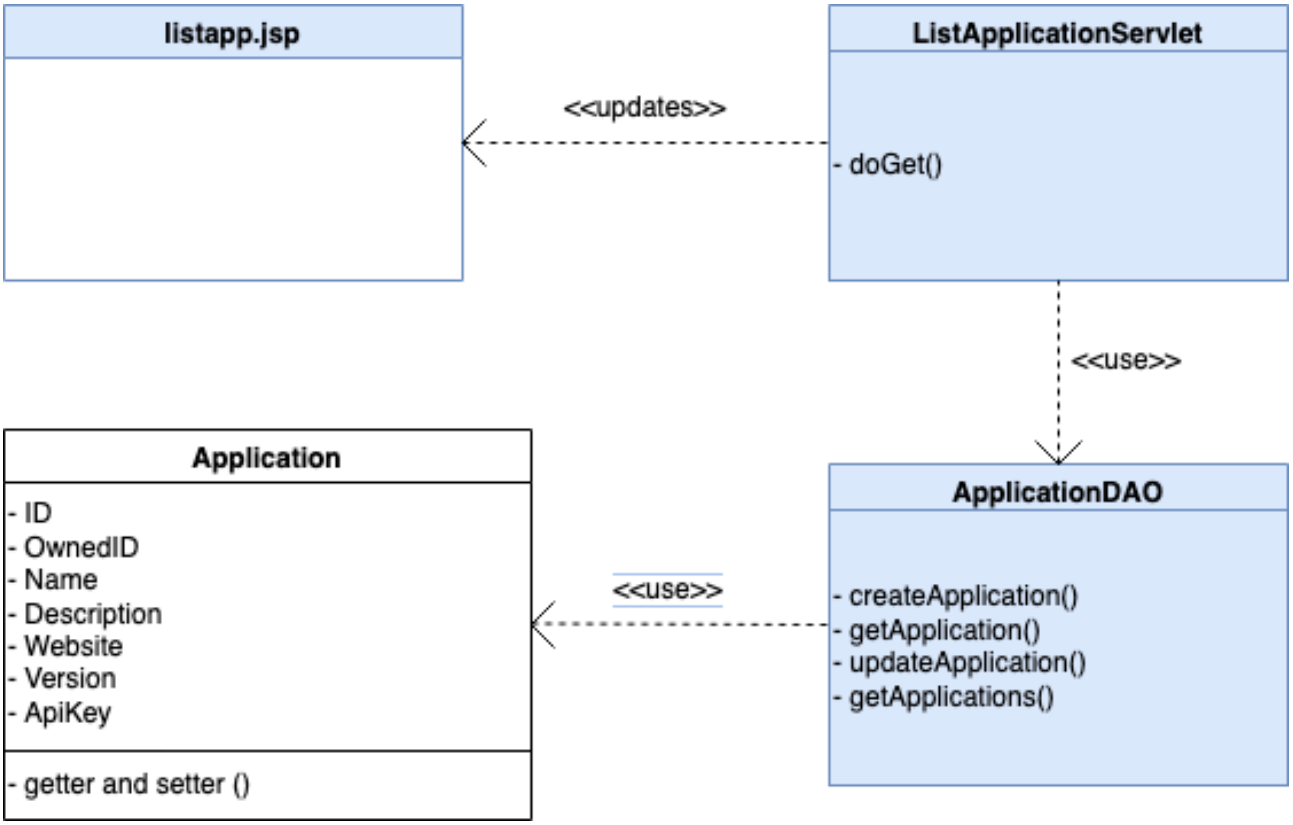
- **MVC**

Nome del Design Pattern: MVC (Model - View - Control).

Descrizione del problema: vogliamo rendere la componente view, model e control facilmente gestibile ed organizzata, mantenendo una netta separazione tra le componenti.

Soluzione del problema: la visualizzazione dell'interfaccia utente è resa possibile grazie alle servlet che modellano i dati bean, che a loro volta vengono visualizzati secondo le necessità dell'utente nelle pagine web dinamiche.

Conseguenze: all'utente sono nascosti completamente il livelli model e control in modo da rendere l'interfaccia grafica pulita ed intuitiva.



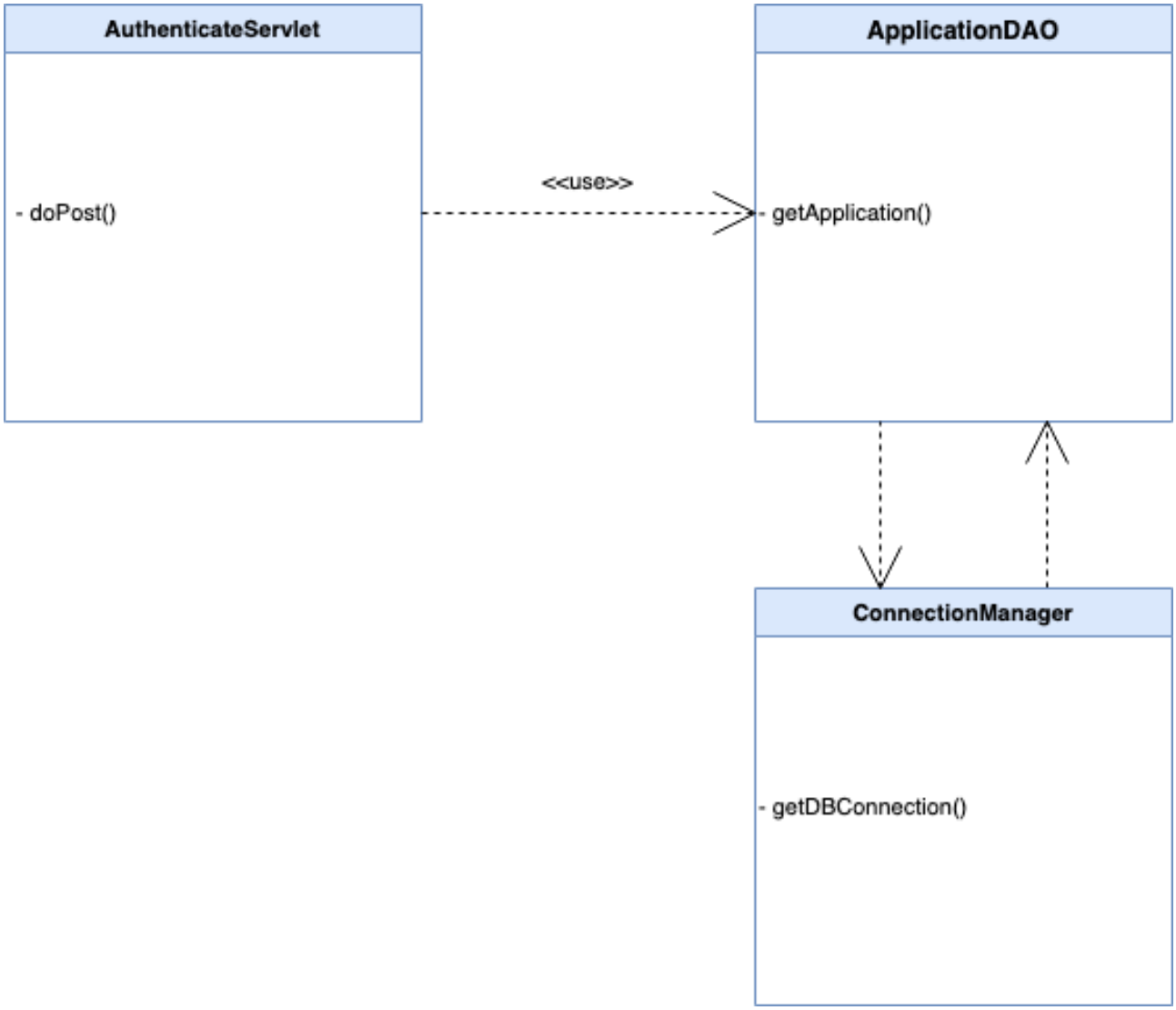
- **(Static) Factory**

Nome del Design Pattern: (Static) Factory Method

Descrizione del problema: vogliamo mantenere una bassa coesione ed alto accoppiamento, senza l'utilizzo di interfacce e costruttori.

Soluzione del problema: l'implementazione di metodi statici possono ritornare un oggetto di ogni sottotipo e riducono la complessità nella creazione di istanze attraverso la memorizzazione nella cache.

Conseguenze: sono più semplici da utilizzare e meno costose della creazione di un nuovo oggetto.



5. Glossario

Termine	Definizione
Api	Application Program Interface
DAO	Data Access Object