SWEG GROUP

Specifica Tecnica

Versione 2.0.0
Data di Rilascio 08/05/2017
Redazione Gianluca Crivellaro
Piergiorgio Danieli
Sebastiano Marchesini
Validazione Pietro Lonardi
Responsabile Alberto Gelmi
Uso Esterno
Destinato ItalianaSoftware S.r.l
Prof. Vardanega Tullio
Prof. Cardin Riccardo

Sommario
Questo documento descrive la specifica tecnica e l'architettura del prodotto sviluppato.

Registro Modifiche

Ver.	Modifica	Nome	Data
2.0.7	Capitolo 4 Aggiunti servizi sottocap. 4.24.2 : buyApi, passRecovery, buyCredits, compare, validation	Sebastiano Marchesini	28/05/2017
2.0.6	Capitolo 4 Aggiunti componenti sottocap. 4.23.2 : buyApi, passRecovery, buyCredits, compare Aggiunte relazioni: uploadApi, userApi, login, registrat.	Sebastiano Marchesini	28/05/2017
2.0.5	Capitolo 4 Aggiunti componenti cap.4.2.2 Aggiunti sotto-cap.18,19,20,21	Sebastiano Marchesini	26/05/2017
2.0.4	Capitolo 4 Modificata la fig.2, 3, 4	Sebastiano Marchesini	26/05/2017
2.0.3	Capitolo 8 e 6 Ridimensionati Diagrammi di Attività e Figura 16	Sebastiano Marchesini	23/05/2017
2.0.2	Capitolo 2.2 Tolto sottocapitolo inerente a Spring Boot 2.2.2	Sebastiano Marchesini	23/05/2017
2.0.1	Capitolo 2.1 Rettificati gli svantaggi in 2.1.4 e 2.1.6	Sebastiano Marchesini	27/03/2017
2.0.0	Validazione	Pietro Lonardi	08/05/2017
1.0.10	Capitolo 9 Rielaborato e corretto	Piergiorgio Danieli	27/03/2017
1.0.9	Capitolo 8.1 Disegnati i diagrammi di attività Aggiunti sottocapitoli 8.1.1 e 8.1.2	Sebastiano Marchesini	25/03/2017
1.0.8	Capitolo 8 Ridisegnati i diagrammi e corretti	Sebastiano Marchesini	24/03/2017
1.0.7	Capitolo 4 Rifatti da cap 4.18 a 4.20	Sebastiano Marchesini	23/03/2017
1.0.6	Capitolo 6 Rifatti da cap 6.3 a 6.9	Piergiorgio Danieli	22/03/2017
1.0.5	Capitolo 4 Rifatti da cap 4.1 a 4.18	Sebastiano Marchesini	22/03/2017
1.0.4	Capitolo 4 Prodotti grafici UML Front End	Sebastiano Marchesini	21/03/2017
1.0.3	Aggiunto sottocapitolo 2.3.4 e 2.3.6 Librerie angular-star-rating e Boostrapt	Sebastiano Marchesini	20/03/2017
1.0.2	Aggiunto sottocapitolo 2.1.4 e 2.1.6 Linguaggi TypeScript e Java	Sebastiano Marchesini	20/03/2017
1.0.1	Capitolo 2 - Tutti Sottocapitoli Riviste desc. tecnologie associandole più fortemente al progetto	Sebastiano Marchesini	20/03/2017
1.0.0	Validazione capitoli	Piergiorgio Danieli	06/03/2017
0.1.5	Appendice A - MockUp	Sebastiano Marchesini	05/03/2017
0.1.4	Tracciamento	Piergiorgio Danieli	05/03/2017
0.1.3	Design Pattern - Utilizzo dei Design Pattern Microservizi - Dependency Injection	Piergiorgio Danieli	04/03/2017

0.1.2	Stesura architettura Back-End	Alberto Gelmi	03/03/2017
0.1.1	Creazione Prototipi MockUp	Sebastiano Marchesini	03/03/2017
0.1.0	Verifica capitoli 2, 7 e Appendice A	Lonardi Pietro	03/03/2017
0.0.20	Stesura testi diagrammi di attività	Gianluca Crivellaro	02/03/2017
0.0.19	Stesura architettura Front-End	Gianluca Crivellaro	02/03/2017
0.0.18	Stesura Introduzione e Architettura generale parte 1	Lonardi Pietro	01/03/2017
0.0.17	Appendice A - Descrizione Design Pattern Comportamentali - Command	Sebastiano Marchesini	01/03/2017
0.0.16	Inizio stesura dei grafici UML per la parte back-end	Sebastiano Marchesini Alberto Gelmi	28/02/2017
0.0.15	Design Pattern - Utilizzo dei Design Pattern Strutturali - Facade	Piergiorgio Danieli	28/02/2017
0.0.14	Inizio stesura dei grafici UML per la parte front-end	Sebastiano Marchesini Gianluca Crivellaro	27/02/2017
0.0.13	Design pattern - Utilizzo dei Design Pattern Creazionali - Abstract Factory Comportamentali - Command	Piergiorgio Danieli	27/02/2017
0.0.12	Design pattern - Utilizzo dei Design Pattern Architetturali - DAO, MVVM	Piergiorgio Danieli	25/02/2017
0.0.11	Appendice A - Descrizione Design Pattern Architetturali - MVVM, Creazionali - Abstract Factory	Sebastiano Marchesini	25/02/2017
0.0.10	Appendice A - Descrizione Design Pattern Architetturali - DAO	Sebastiano Marchesini	24/02/2017
0.0.9	Tecnologie Utilizzate - Librerie Highcharts, Angular Material, Highcharts-ng	Sebastiano Marchesini	23/02/2017
0.0.8	Database - Progettazione logica	Piergiorgio Danieli	22/02/2017
0.0.7	Tecnologie Utilizzate - Framework AngularJS, Spring	Sebastiano Marchesini	22/02/2017
0.0.6	Tecnologie Utilizzate - Librerie Leonardo	Sebastiano Marchesini	21/02/2017
0.0.5	Tecnologie Utilizzate - Linguaggi Jolie, SQL	Sebastiano Marchesini	20/02/2017
0.0.4	Database - Progettazione concettuale	Piergiorgio Danieli	18/02/2017
0.0.3	Tecnologie Utilizzate - Linguaggi Jolie, SQL	Sebastiano Marchesini	17/02/2017
0.0.3	Tecnologie Utilizzate - Linguaggi HTML5, CSS, Javascript	Sebastiano Marchesini	16/02/2017
0.0.2	Appendice A - Descrizione Design Pattern Microservizi	Sebastiano Marchesini	15/02/2017
0.0.1	Impostazione documento	Sebastiano Marchesini	14/02/2017

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del Documento

Lo scopo generale del documento è di misurare l'efficienza e tenerla in considerazione preventivamente. Importantissimo per il committente che tiene d'occhio la stima delle risorse.
È in particolare una dichiarazione di interfaccia di pianificazione e consuntivazione. Sempre redatto dal *Project Manager* schematizzato:

1. Definizione degli obbiettivi;
2. Analisi dei rischi;
3. Descrizione del modello di processo di sviluppo;
4. Suddivisione di sottoinsiemi;
5. Attività di progetto;
6. Stima dei costi;
7. Consuntivo attività.

1.2 Glossario

Al fine di evitare ambiguità e ottimizzare la comprensione dei documenti, viene incluso un Glossario, nel quale saranno inseriti i termini tecnici, acronimi e parole che necessitano di essere chiarite.
Un glossario è una raccolta di termini di un ambito specifico e circoscritto. In questo caso per raccogliere termini desueti e specialistici inerenti al progetto.

1.3 Riferimenti

1.3.1 Normativi

- **Vincoli organigramma e dettagli tecnico-economici**:
  http://www.math.unipd.it/~tullio/IS-1/2016/Progetto/PD01b.html.
- **Norme di Progetto**:
  "Norme di Progetto v4.0.0".

1.3.2 Informativi

- **Metriche di Progetto**:
  https://it.wikipedia.org/wiki/Metriche_di_progetto.
- **Modello incrementale**:
  https://it.wikipedia.org/wiki/Modello_incrementale.
- **Modello incrementale**:
  https://it.wikipedia.org/wiki/Metodologia_agile.
- **Gestione di progetto**:
  http://www.math.unipd.it/~tullio/IS-1/2016/Dispense/L04.pdf.
- **Design Pattern and Refactoring**:
  https://sourcemaking.com/design-patterns-ebook.

2 Tecnologie Utilizzate

In questa sezione illustreremo le tecnologie e i linguaggi utilizzati nel progetto, indicando dove nello specifico sono state implementate (prendendo riferimento anche i capitoli 5, 6 e 7) ed i pro e contro delle scelte utilizzate.

2.1 Linguaggi

2.1.1 HTML 5

HTML$_g$ è la particella elementare di Internet: il linguaggio di mark-up che dà vita ai siti Web statici. HTML5 è la versione 5 di questo linguaggio. HTML5 apre le porte a una nuova serie di funzioni per contenuti interattivi e animati che funzionano universalmente su qualsiasi piattaforma o tipo di dispositivo. In particolare il linguaggio è stato utilizzato per rappresentare la parte di *View*$_g$ nel nostro *Front End*$_g$.

Vantaggi :

- Funziona su la maggior parte dei computer e sui dispositivi mobili;
- Video e animazioni supportati senza plug-in$_g$ esterni;
- Pensiero indirizzato sempre più al web semantico e HTML5 ne è portavoce;
- Maggior flessibilità e potenzialità rispetto alle precedenti versioni e modellabile con la nostra tecnologia primaria di Angular2.

Svantaggi :

- Visualizzazione non uniforme sulle versioni precedenti dei browser o su Internet Explorer$_g$;
- Strumenti non completamente sviluppati. C'è bisogno di un linguaggio di supporto per le pagine dinamiche.

2.1.2 CSS3

Il CSS$_g$ è un linguaggio con il quale formattare le pagine Web. Un file CSS$_g$ viene normalmente chiamato un foglio di stile, e va associato ad una o più pagine Web. I fogli di stile nel progetto saranno rigorosamente esterni. CSS3 aggiorna le funzionalità e le componenti stilistiche.
Come già menzionato la compatibilità massima con HTML5$_g$ e il suo scopo visuale inseriscono il linguaggio all'interno della *View*$_g$ nel modello generale del *Front End*$_g$.

Vantaggi :

- Separata la struttura del sito dalla presentazione;
- Più facile progettazione di accessibilità;
- Template$_g$ unico per varie pagine senza ripetizione;
- Facile la modifica in caso di cambiamento;
- Grafica accattivante per gli utenti;
- Integrabile facilmente con *framework*$_g$ *Bootstrap*$_g$.

Svantaggi :

- I browser più datati hanno una non corretta interpretazione dei CSS$_g$;
- Maggiore attenzione sulla psicologia di marketing grafico posta verso l'utente.

2.1.3 Javascript

La caratteristica principale di *JavaScript*$_g$, è quella di essere un linguaggio di *scripting*$_g$. Ci permetterà di eseguire particolari operazioni grazie alla flessibilità di questo linguaggio orientato agli oggetti ed eventi. Tali funzioni di script possono essere opportunamente inserite in file HTML$_g$, in pagine JSP o in appositi file separati con estensione .js poi richiamati nella logica di business.
Utilizziamo *JavaScript*$_g$ in piccole funzioni sempre all'interno del nostro *Front End*$_g$. Nello specifico per correggere e implementare librerie di utilizzo per la nostra applicazione web che illustreremo in seguito.

Vantaggi :

- Possibilità di rendere dinamiche le pagine web e di estendere funzionalità;
- Il linguaggio di scripting è più sicuro ed affidabile perché in chiaro e da interpretare, quindi può essere filtrato;
- Gli script hanno limitate capacità, per ragioni di sicurezza, per cui non è possibile fare tutto con Javascript$_g$, ma occorre abbinarlo ad altri linguaggi evoluti, (come Jolie$_g$);
- Il codice *JavaScript*$_g$ viene eseguito sul client per cui il server non è sollecitato più del dovuto e la velocità dell'applicazione complessiva è migliore;
- Ci permette di implementare librerie aggiuntive senza dover imparare o creare modelli grafici complessi.

Svantaggi :

- Il codice è visibile e può essere letto da chiunque;
- La mancanza di tipizzazione del linguaggio potrebbe indurre a commettere errori nel codice e rendere più difficile la progettazione dei test.

2.1.4 TypeScript

TypeScript è un linguaggio di programmazione libero ed *Open source*$_g$. Anche questa tecnologia è basata sullo *scripting*$_g$. È il linguaggio su cui si basa Angular2 e nel nostro progetto è ampiamente utilizzato nella parte di *Front End*$_g$. *TypeScript* estende la sintassi di *JavaScript*, in questo modo qualunque programma scritto in *JavaScript* è anche in grado di funzionare con *TypeScript* senza nessuna modifica. *TypeScript* è stato progettato per lo sviluppo di grandi applicazioni ed una volta compilato produce *JavaScript*.
Come già menzionato utilizzeremo *TypeScript* per tutta la parte di *Front End*$_g$, senza contare le *View*$_g$ scritte in linguaggi prettamente visuali, arricchendo i linguaggi web di tipi e classi.

Vantaggi :

- Possibilità di rendere tipizzato *JavaScript* e quindi la parte web;
- Possibilità di generare interfacce;
- Uso di firma digitale (da noi non implementata);
- Tipi di dato aggiuntivi come enum e data;
- Ci permette di utilizzare il *framework*$_g$ principale Angular2.

Svantaggi : A secondo della filosofia di caricamento esso presenta problemi e svantaggi differenti.

- Codice non di immediata comprensione e in-leggibile se precompilato;
- Linguaggio che necessita prima una fase di interpretazione da parte di *NodeJs*$_g$ e in seguito la compilazione runtime consuma molto tempo e risorse.

2.1.5 Jolie

Jolie$_g$ fissa i concetti di programmazione di microservizi come funzionalità del linguaggio native: gli elementi di base del software non sono oggetti o funzioni, ma piuttosto servizi che possono sempre essere trasferiti e replicati in base alle esigenze. Distribuzione e riusabilità si raggiungono con la semplice progettazione e codifica.
È il linguaggio che copre la maggior parte del nostro *Back End*$_g$ indicato per i microservizi e su richiesta della ditta committente. **Vantaggi :**

- Linguaggio orientato agli oggetti, basato su Java, con tutti i vantaggi dei linguaggi ad oggetti;
- Linguaggio nato appositamente per i microservizi che è punto focale del nostro progetto;
- Funziona perfettamente sia in locale sia in remoto, il codice non altera la logica dei programmi;
- I servizi possono scambiare dati utilizzando diversi protocolli, non vi è uno specifico e possono essere diversi d'entrata che in uscita;
- Essendo un codice orientato ai microservizi ogni prodotto creato può essere riutilizzato;
- È dotato nativamente di primitive per *workflow*$_g$, questo rende il codice fluido per le esigenze, evitando le variabili computazionali soggette a errori per verificare ciò che è accaduto finora in un calcolo;
- Jolie$_g$ è dotato di una solida semantica per la gestione di errori della programmazione parallela. Possiamo aggiornare il comportamento dei gestori degli errori in fase di esecuzione;
- Implementa Leonardo$_g$: servizio Jolie$_g$ che agisce come un server web in grado di interagire con le applicazioni web scritte in diverse tecnologie (JSON, XML, AJAX, GWT).

Svantaggi :

- Non compatibile con tutti i linguaggi e ancora in fase di prototipazione l'interazione con database non relazionali;
- Linguaggio nuovo e non usato in ogni suo ramo e sfaccettatura, manca infatti documentazione completa ed esaustiva.

2.1.6 Java

Java$_g$ è il principale linguaggio di alto livello e tipicizzato più in uso al mondo . Ha la possibilità di implementare moltissime librerie esterne ed è anche uno dei linguaggi che il team conosce maggiormente. È il linguaggio implementato per alcune funzioni del *Back End*$_g$ e compatibile con il linguaggio Jolie$_g$. È utilizzato, nel nostro progetto, per facilitare i calcoli e le chiamate di alcune specifiche funzioni. In particolare per il calcolo della dimensione e il *timestamp*$_g$ del flusso di dati nei vari microservizi. In aggiunta ha una funzione di decriptaggio per chiavi e dati sempre rivolta alle Api dei nostri utenti.
Vantaggi :

- Semplice linguaggio, orientato agli oggetti e "famigliare";
- È una tecnologia robusta e sicura, infatti la utilizziamo per le operazioni più critiche e complesse;
- È indipendente da quale piattaforma viene utilizzato;
- Consente strumenti e librerie per il networking, come la nostra implementazione con Jolie;
- Progettato per eseguire codice da sorgenti remote in modo sicuro, probabilmente implementato su server e macchine differenti.

Svantaggi :

- L'interfacciamento con Jolie (linguaggio base del Back End) non è diretto;
- Non sviluppato direttamente per supportare l'architettura a microservizi.

2.1.7 SQL

SQL è il linguaggio che andremo ad usare per quanto riguarda la parte di database della nostra applicazione web. Jolie$_g$ offre dei comandi semplici e si interfacciano comodamente con il linguaggio per basi di dati relazionali. Nel nostro progetto è stato scelto come linguaggio relazionale per il database usato.
È un linguaggio standardizzato per database basati sul modello relazionale (RDBMS) progettato per:

- creare e modificare schemi di database (DDL - Data Definition Language);
- inserire, modificare e gestire dati memorizzati (DML - Data Manipulation Language);
- interrogare i dati memorizzati (DQL - Data Query Language);
- creare e gestire strumenti di controllo ed accesso ai dati (DCL - Data Control Language).

Nonostante il nome, non si tratta dunque solo di un semplice linguaggio di interrogazione, ma alcuni suoi sottoinsiemi si occupano della creazione, della gestione e dell'amministrazione del database.
Vantaggi :

- Già implementato e studiato per il nostro linguaggio cardine Jolie$_g$;
- Già a conoscenza della totalità del team;
- Elastico e integrato da tempo nelle applicazione web;
- Molto veloce e permette di gestire un alto numero di operazioni/secondo;
- Se ben programmato in principio avvantaggia maggiormente la lettura, importante per l'applicazione web;

Svantaggi :

- Gestisce operazioni non troppo complicate;
- Limitato su basi di dati troppo grandi;
- Difficile riadattamento nel caso di modifica della struttura del database.

2.2 Frameworks

2.2.1 Angular2

Angular2$_g$ è un framework JavaScript$_g$ per lo sviluppo di applicazioni Web client side. Pur essendo relativamente giovane (la versione 1.0 è stata rilasciata nel 2012), il progetto ha riscosso un notevole successo dovuto all'approccio di sviluppo proposto e all'infrastruttura fornita che incoraggia l'organizzazione del codice e la separazione dei compiti nei vari componenti.
Per raggiungere questo obiettivo, Angular2$_g$ da un lato esalta e potenzia l'approccio dichiarativo del HTML$_g$ nella definizione dell'interfaccia grafica, dall'altro fornisce strumenti per la costruzione di un'architettura modulare e testabile della logica applicativa di un'applicazione
Angular2$_g$ fornisce tutto quanto occorre per creare applicazioni moderne che sfruttano le più recenti tecnologie, come ad esempio le Single Page Application, cioè applicazioni le cui risorse vengono caricate dinamicamente su richiesta, senza necessità di ricaricare l'intera pagina. Tra le principali funzionalità a supporto dello sviluppo di tali applicazioni segnaliamo:

- il binding bidirezionale (*two-way binding*)
- la dependency injection
- il supporto al pattern MVC

- il supporto ai moduli
- la separazione delle competenze
- la testabilità del codice
- la riusabilità dei componenti

Nel nostro progetto viene utilizzato a pieno implementando il *pattern* architetturale MVVM e comportamentale Dependency Injection; per la completa parte di *Front End*$_g$. Nel capitolo 5 verrà illustrato a pieno la sua implementazione. **Vantaggi :**

- Estremamente espressivo, leggibile e di facile implementazione. Un'evoluzione all'alluma$_g$ per facilitare le applicazioni web;
- È completamente estensibile e funziona bene con altre librerie. Ogni funzione può essere modificato o sostituito in base alle esigenze del flusso di lavoro di sviluppo;
- Integrazione perfetta con uno dei pattern scelti Model View View Model;
- Data-Binding$_g$ Bidirezionale di Angular2$_g$ gestisce la sincronizzazione tra il DON e il modello, e viceversa;
- Angular2$_g$ ha un sottosistema integrato di Dependency Injection$_g$ che aiuta lo sviluppatore a creare un'applicazione facile da sviluppare, capire e provare;
- Utilizza le direttive$_g$, cioè possono essere usate per creare tag HTML$_g$ personalizzati che funzionano come nuovi widget personali. Possono anche essere usati per "decorare" elementi con un comportamento e manipolare attributi DOM in un modo interessante;
- Permette la creazione della nostra applicazione web e dinamica, rimanendo aggiornati in fatti di tecnologie principali e forte nel mercato.

Svantaggi :

- Framework$_g$ che si deve usare completamente e non si può lasciare spazio all'incomprensione;
- Non vi sono IDE specifici o dedicati;
- Non vi sono delle linee guida internazionali, ma solo spunti vari nel web.

2.3 Librerie

2.3.1 Leonardo: il web server di Jolie

Leonardo$_g$ è un server web sviluppato unicamente in Jolie$_g$. È molto flessibile e può scalare da un semplice contenuto statico HTML$_g$ fino a sostenere un complesso servizio web dinamico.
Nel nostro progetto è offerto dal proponente un server preparato con Leonardo. **Vantaggi :**

- Scritto interamente in Jolie$_g$ e facilmente implementabile nel prodotto;
- Si interfaccia con HTML$_g$, JQuery;
- Permette l'uso dei Cookies$_g$.

Svantaggi :

- Non è estendibile con qualsiasi linguaggio anche se vi sono già molti prototipi.

2.3.2 Ng2-Charts

È una libreria JavaScript$_g$ che permette di gestire e inserire grafici all'interno della nostra applicazione web. La utilizziamo in particolare per mostrare le statistiche generiche del sito agli amministratori e i dati di interesse per i microservizi offerti dagli utenti.
Utilizzato nella componente Issue per creare grafici lineari e semplici.

Vantaggi :

- Compatibilità con tutti i browser moderni, sia desktop che mobile;
- Interfacciamento semplice con Angular2$_g$ grazie a delle direttive;
- Aggiornamento dei grafici in tempo reale;
- Possibilità di esportare i grafici in vari formati;
- Open source$_g$, quindi personalizzabile. E gratuito per fini non commerciali.

Svantaggi :

- Non è compatibile con vecchie versioni di Angular2$_g$;
- Non è mai stata utilizzata dai membri del gruppo.

2.3.3 Angular Material

Angular Material è l'implementazione del Material Design$_g$ in AngularJS$_g$. Fornisce un insieme di componenti per l'interfaccia utente riutilizzabili, testati e accessibili, basati sul Material Design$_g$.
Vantaggi :

- Compatibilità con tutti i browser moderni, sia desktop che mobile;
- Facile relazionarsi con AngularJS$_g$ essendo la sua implementazione;
- Documentazione$_g$ e informazioni più volte usate e prodotte, oltre che esempi specifici.

Svantaggi :

- Anche questa tecnologia come AngularJS$_g$ non è stata usata all'interno del gruppo e richiede particolare attenzione;
- Tecnologia non definitiva e in continuo aggiornamento.

Non abbiamo scelto abbiamo infine utilizzato Angular Material perché non perfettamente compatibile con la nuova versione di Angular2.

2.3.4 Bootstrap

Bootstrap$_g$ è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web. Essa contiene modelli di progettazione basati su HTML e CSS, sia per la tipografia, che per le varie componenti dell'interfaccia, come moduli, pulsanti e navigazione, così come alcune estensioni opzionali di JavaScript. Nel nostro progetto va a implementarsi nella parte di *View*$_g$ e arricchisce e facilita le nostre operazioni grafiche andando a strutturare la nostra sintassi. **Vantaggi :**

- Compatibilità con tutti i browser moderni, sia desktop che mobile;
- Facile utilizzo con le tecnologie utilizzate nel progetto e a nostra disposizione;
- Documentazione$_g$ ricca e ad una versione stabile.

Svantaggi :

- Tecnologia sconosciuta e mai utilizzata all'interno del team;
- Non rispetta alcuni principi di accessibilità e del *W3C*$_g$.

Non abbiamo scelto abbiamo infine utilizzato Angular Material perché non perfettamente compatibile con la nuova versione di Angular2.

2.3.5 Angular-star-rating

È una componente per AngularJS e Angular2$_g$ che aiuta e permette facilmente l'implementazione di stelle per il rating dei microservizi.
Nello specifico viene aggiunta e implementata sian nella *View*$_g$ singola delle Api, sia nella parte *Model View*. **Vantaggi :**

- Semplice implementazione ed utilizzo;
- Implementabile nella maggior parte dei browsers;
- La documentazione è scritta in maniera chiara riporta degli esempi.

Svantaggi :

- È una tecnologia in costante migliorato, ha possibili variazioni e aggiornamenti, ma stabile con le nostre tecnologie.

3 Descrizione Architettura

3.1 Metodo di specifica

Il metodo scelto per esporre l'architettura é tramite un approccio top-down, il che vuol dire che verrá prima presentata una architettura molto astratta e poi verrá passo a passo espansa fino ad arrivare ad un livello molto basilare dove potranno essere identificate le singole classi e le loro sottoclassi, queste ultime verranno trattate nello specifico all'interno della fase della Progettazzione in dettaglio. Partiremo quindi ad analizzare il rapporto che é presente tra i vari package per poi espandere i package ed analizzare nello specifico da cosa sono formati, che lavoro svolgono e come comunicano tra di loro, quindi passeremo ad analizzare le classi che ne fanno parte, ovvero i metodi che contengono, il loro funzionamento e l'obiettivo per cui sono state sviluppate. Verranno poi mostrati dei design pattern utilizzati e mostrati alcuni esempi degli stessi nell'appendice A. Il Proponente$_g$ ha chiesto esplicitamente che venisse utillizzato all'interno del progetto il linguaggio Jolie$_g$, un linguaggio da lui sviluppato, e ha messo a disposizione anche Leonardo$_g$, percui nella fase di progettazzione il team ha dovuto integrare questa tecnologia al resto del progetto.

3.2 Architettura generale

Il Proponente$_g$ nel presentare il capitolato ha riportato un'architettura di massima che rappresenterà come dovrà essere il prodotto finale. Tale architettura è la seguente:

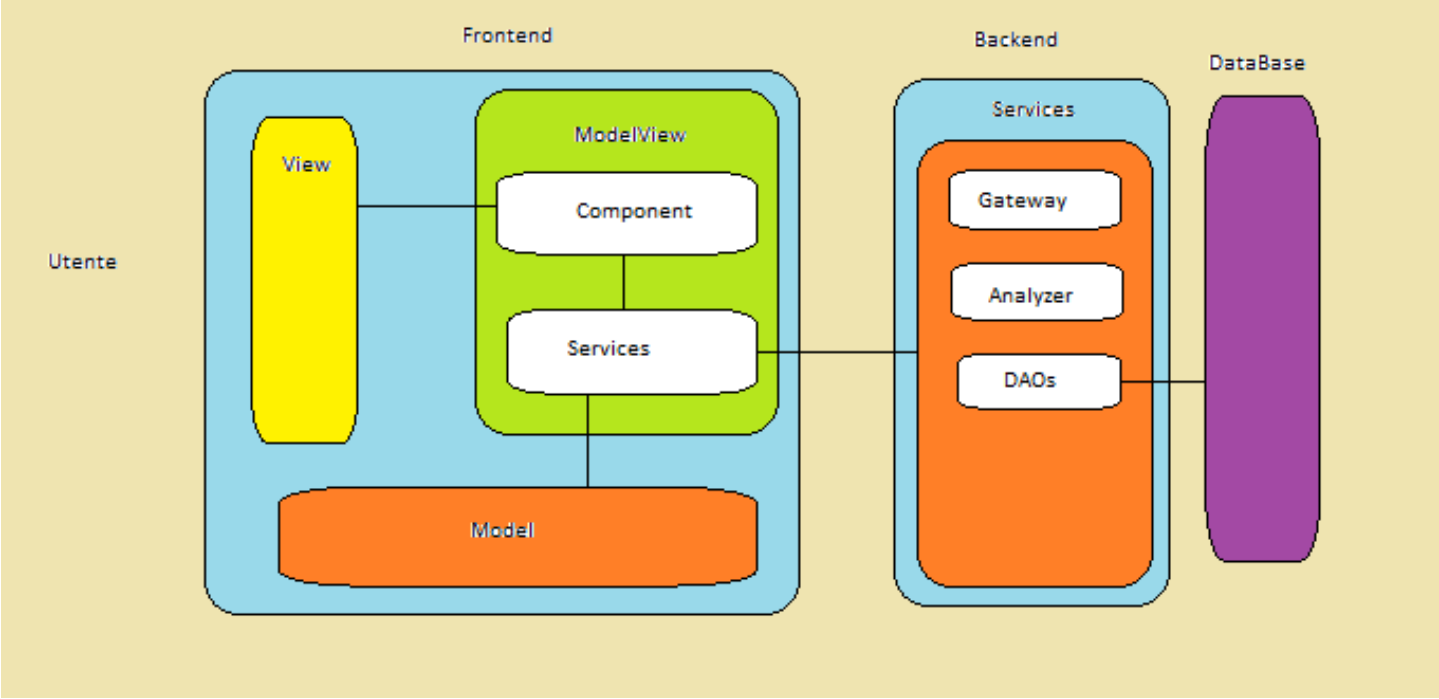


Figura 1: Architettura di massima del progetto

L'APIMarket$_g$ dovrà permettere ad un utente di caricare una API$_g$, deve poter acquistare APIKey$_g$ che gli permettano di utilizzare API$_g$ già presenti nell'APIMarket$_g$ e infine avere la possibilità di monitorare i dati riferiti alle API$_g$. L'architettura sopra riportata rappresenta una possibile soluzione di come deve essere il prodotto finale: attraverso un APIGateway$_g$ è possibile controllare che le API$_g$ caricate siano conformi a standard che verranno concordati con il Proponente$_g$, sempre l'APIGateway$_g$ si occuperà di gestire acquisto e la validazione dell'APIKey$_g$ in possesso dell'utente (o che desidera acquistare), infine sarà sempre lui a monitorare le chiamate alle API$_g$ e a fornire i dati necessari richiesti. Per interfacciare l'utente all'APIGateway$_g$ è necessaria un'interfaccia. Per definire l'interfaccia il team ha optato per un'architettura MVVM (Model-View-ViewModel) che si discosta dall'architettura MVC in quanto in questa architettura

esiste un collegamento bilaterale tra la *View* e la *Model* e questo comporta che ogni modifica che viene effettuata sull'una, va a modificare anche i parametri dell'altra. La scelta dell'utilizzo di questa architettura nei confronti di una architettura MVC viene trattata nel capitolo riguardante il Front-end$_g$. Il Proponente$_g$ ha espressamente chiesto che il team usufruisse di una tecnologia da lui creata, ovvero Jolie$_g$, perciò il team ha deciso di sviluppare il Back-end$_g$ secondo un'architettura a microservizi così da poter sfruttare a pieno le potenzialità di questa tecnologia.

4 Architettura Front end

4.1 com.apim.app

4.1.1 Informazioni sul package

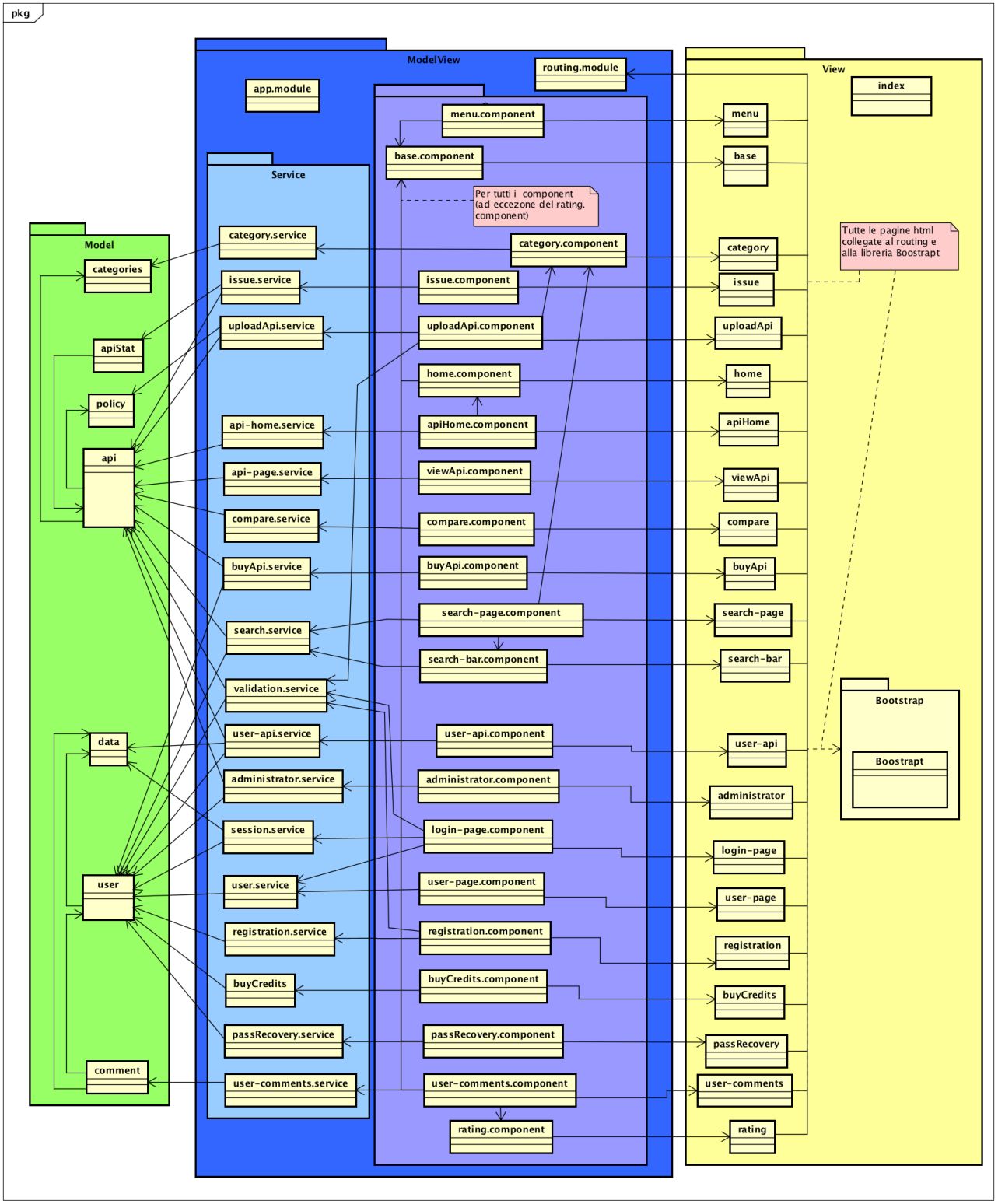


Figura 2: Schema package Front End

Questo package racchiude tutta la parte front-end dell'applicazione.

4.1.2 Package contenuti

- *.model;
- *.view;
- *.modelView.

4.2 com.apim.app.view

4.2.1 Informazioni sul package

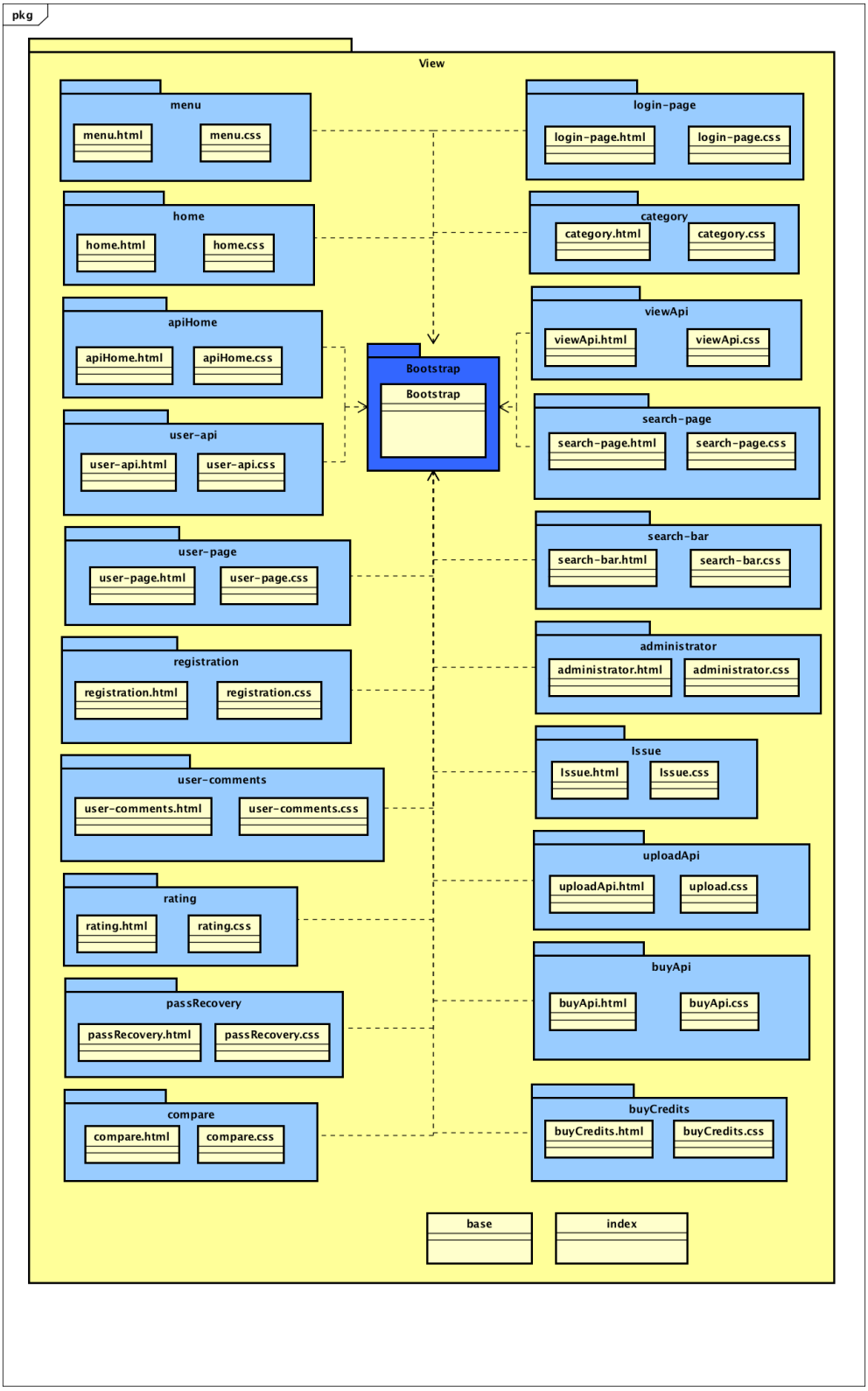


Figura 3: Schema package front end - View

La *View* racchiude i file HTML e CSS interfaccia per l'utente finale. Oltre a contenere la libreria *Bootstrap*

4.2.2 Package contenuti

- com.apim.app.view.base;
- com.apim.app.view.menu;
- com.apim.app.view.home;
- com.apim.app.view.apiHome;
- com.apim.app.view.user-api;
- com.apim.app.view.user-comments;
- com.apim.app.view.registration;
- com.apim.app.view.user-page;
- com.apim.app.view.login-page;
- com.apim.app.view.category;
- com.apim.app.view.viewApi;
- com.apim.app.view.search-page;
- com.apim.app.view.search-bar;
- com.apim.app.view.issue;
- com.apim.app.view.administrator;
- com.apim.app.view.buyApi;
- com.apim.app.view.passRecovery;
- com.apim.app.view.buyCredits;
- com.apim.app.view.compare .

4.2.3 Interazione con altri componenti

- com.apim.app.model-view.component.* .

La parte di componenti dell'applicazione richiama i template grafici per arricchirli. Ogni componente ha associata la sua view.

- com.apim.app.model-view.routing.module .

Tutte le componenti grafiche sono associate al routing.module che si preoccupa di richiamarle durante la navigazione web.

Nel documento non verranno indicati ogni singolo collegamento tra package *View* e componente routing.module perché scontato ed è così per tutte le *View*, tranne la pagina di iniziazione (*index*), le librerie *Bootstrap* e di *Angular-star-rating*.

4.2.4 Classi

- **index.html**
  - **Descrizione:** template $HTML_g$ che si presenta prima del caricamento dell'applicazione. È una pagina vuota che presenta un solo elemento testuale che illustra il caricamento.
  - **Relazioni con altre classi:**
    - Non vi è alcun collegamento con altre classi in quando il *View Model* e in particolare il framework di Angular 2 sostituiscono automaticamente la pagina.

4.3 com.apim.app.view.base

4.3.1 Informazione sul package

Questo package contiene i template $HTML_g$ e CSS_g della pagina in cui fanno base le pagine principale. Racchiude una struttura semplice. La componente nel *View Model* poi che coopera e richiama le altre.

4.3.2 Interazione con altri componenti

- com.apim.app.modelView.Component.base.component .

La componente base richiama il template HTML e inietta grazie alle altre componenti i rispettivi template.

4.3.3 Classi

- **base.html**

  Descrizione: contiene il template HTML relativo alla pagina della API_g di base.

  Relazioni con altre classi:

  - com.apim.app.modelView.Component.base.HTML

  Semplice relazione tra HTML e CSS

- **base.css**

  Descrizione: contiene il template CSS relativo alla pagina del progetto (APIM) di base.

  Relazioni con altre classi:

  - com.apim.app.modelView.Component.base.CSS

  Semplice relazione tra CSS e HTML

4.4 com.apim.app.view.menu

4.4.1 Informazione sul package

Questo package contiene i template $HTML_g$ e CSS_g della barra del menù. Verrà infatti gestita qui la sua parte grafica.

4.4.2 Interazione con altri componenti

- com.apim.app.modelView.Component.menu.component .

La componente menu richiama il template HTML e viene inietta grazie alla componente base nella pagina strutturale.

4.4.3 Classi

- **menu.html**

  Descrizione: contiene il template HTML relativo alla barra del menu .

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.menu.html.

  Semplice relazione tra HTML e CSS

- **menu.css**

  Descrizione: contiene il template CSS relativo alla barra del menu .

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.menu.css.

  Semplice relazione tra CSS e HTML

4.5 com.apim.app.view.home

4.5.1 Informazione sul package

Questo package contiene i template $HTML_g$ e CSS_g della pagina principale. Fare riferimento all'appendice per avere una visione preventiva del modello.

4.5.2 Interazione con altri componenti

- com.apim.app.modelView.Component.home.component .

La componente home richiama il template HTML e viene inietta grazie alla componente base nella pagina strutturale.

4.5.3 Classi

- **home.html**

  Descrizione: contiene il template HTML relativo alla pagina principale (home).

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.home.html.

  Semplice relazione tra HTML e css.

- **home.css**

Descrizione: contiene il template CSS relativo alla pagina principale (home).

Relazioni con altre classi:

- com.apim.app.modelView.Component.home.css.

Semplice relazione tra CSS e HTML

4.6 com.apim.app.view.apiHome

4.6.1 Informazione sul package

Se il precedente package conteneva la struttura della pagina home. Questo contiene il template del contenuto principale. Il template della lista di microservizi consigliati, più scaricati e/o preferiti.

4.6.2 Interazione con altri componenti

- com.apim.app.modelView.Component.apiHome.component .

La componente apiHome richiama il template HTML e viene inietta grazie alla componente base nella pagina strutturale.

4.6.3 Classi

- **apiHome.html**

  Descrizione: contiene il template HTML relativo al contenitore principale nella home.

  Relazioni con altre classi:

  - com.apim.app.modelView.Component.apiHome.html.

  Semplice relazione tra HTML e css.

- **home.css**

  Descrizione: contiene il template CSS relativo al contenitore principale nella home.

  Relazioni con altre classi:

  - com.apim.app.modelView.Component.apiHome.css.

  Semplice relazione tra CSS e HTML

4.7 com.apim.app.view.userApi

4.7.1 Informazione sul package

UserApi è il contenitore delle informazioni principali dell'utente . Presenta un link di modifica delle informazioni , una avatar, il credito, nome, cognome e altre informazioni riconducibili all'utente loggato ed è riutilizzabile nella pagina di soluzione alla ricerca di un utente specifico.

4.7.2 Interazione con altri componenti

- com.apim.app.modelView.Component.userApi.component .

La componente userApi richiama il template HTML e viene inietta grazie alla componente base nella pagina home o pagina utente.

4.7.3 Classi

- **userApi.html**

  Descrizione: contiene il template HTML relativo al contenitore di informazioni profilo.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.userApi.html.

  Semplice relazione tra HTML e css.

- **userApi.css**

  Descrizione: contiene il template CSS relativo al contenitore di informazioni profilo.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.userApi.css.

  Semplice relazione tra CSS e HTML

4.8 com.apim.app.view.user-comments

4.8.1 Informazione sul package

User-comments è specifica per quanto riguarda la parte "social" della nostra applicazione e progetto. È infatti il contenitore dei commenti sia form che non. Al suo interno verrà iniettato anche il template dello *Angular-star-rating*.

4.8.2 Interazione con altri componenti

- com.apim.app.modelView.Component.user-comments.component .

La componente user-comments richiama il template HTML e viene inietta grazie alla componente base dove necessario.

4.8.3 Classi

- **user-comments.html**

  Descrizione: contiene il template HTML relativo al contenitore del commento.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.user-comments.html.

  Semplice relazione tra HTML e css.

- **user-comments.css**

  Descrizione: contiene il template CSS relativo al contenitore del commento.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.user-comments.css.

  Semplice relazione tra CSS e HTML

4.9 com.apim.app.view.registration

4.9.1 Informazione sul package

Registration è la pagina che contiene la form per una registrazione di un nuovo utente. Si può accedere grazie al routing.

4.9.2 Interazione con altri componenti

- com.apim.app.modelView.Component.registration.component .

La componente registration richiama il template HTML e viene indirizzata nella sezione adatta.

4.9.3 Classi

- **registration.html**

  Descrizione: contiene il template HTML relativo alla pagina di registrazione.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.registration.html.

  Semplice relazione tra HTML e css.

- **registration.css**

  Descrizione: contiene il template CSS relativo alla pagina di registrazione.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.registration.css.

  Semplice relazione tra CSS e HTML

4.10 com.apim.app.view.user-page

4.10.1 Informazione sul package

User-page è la struttura di base della pagina utente. All'interno verranno inserite i vari contenitori come: il contenitore per le informazioni utente, le api acquistate e altro. Fare riferimento al mock-up nell'appendice per avere un prototipo.

4.10.2 Interazione con altri componenti

- com.apim.app.modelView.Component.user-page.component .

La componente user-page richiama il template HTML e viene caricate all'interno della nostra single Page Application

4.10.3 Classi

- **user-page.html**

  Descrizione: contiene il template HTML relativo alla pagina utente standard.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.user-page.html.

  Semplice relazione tra HTML e css.

- **user-page.css**

  Descrizione: contiene il template CSS relativo alla pagina utente standard.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.user-page.css.

  Semplice relazione tra CSS e HTML

4.11 com.apim.app.view.login-page

4.11.1 Informazione sul package

Registration è la pagina che contiene la form per il login di un utente. Si può accedere dalla pagina principale e presenta solo due form di inserimento testo e conferma. Da qui inoltre è possibile un re indirizzamento del recupero password.

4.11.2 Interazione con altri componenti

- com.apim.app.modelView.Component.login-page.component .

La componente login-page richiama il template HTML e viene caricate all'interno della nostra single Page Application nella home Page grazie alla componente base.

4.11.3 Classi

- **login-page.html**

  Descrizione: contiene il template HTML relativo al contenitore della form login.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.login-page.html.

  Semplice relazione tra HTML e css.

- **login-page.css**

  Descrizione: contiene il template CSS relativo al contenitore della form login.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.login-page.css.

  Semplice relazione tra CSS e HTML

4.12 com.apim.app.view.category

4.12.1 Informazione sul package

Category è la pagina che contiene la form per la scelta della categoria. In vista alla tipologia di applicazione basate sui microservizi e avendo un modello a parte per le categorie, si è deciso di ideare un microservizio per la scelta della categoria. Presenta una semplice form radio button ma separata dalla form di caricamento di una Api.

4.12.2 Interazione con altri componenti

- com.apim.app.modelView.Component.category.component .

La componente category richiama il template HTML e viene caricate all'interno della form per il caricamento di una nuova api.

4.12.3 Classi

- **category.html**

  Descrizione: contiene il template HTML relativo al contenitore della form di scelta della categoria.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.category.html.

  Semplice relazione tra HTML e css.

- **category.css**

  Descrizione: contiene il template CSS relativo al contenitore della form di scelta della categoria.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.category.css.

  Semplice relazione tra CSS e HTML

4.13 com.apim.app.view.uploadApi

4.13.1 Informazione sul package

UploadApi è la pagina che contiene la form per caricare una nuovo microservizio. Molto simile alla pagina di registrazione in essa verrà iniettata la parte di scelta della categoria.

4.13.2 Interazione con altri componenti

- com.apim.app.modelView.Component.uploadApi.component .

La componente uploadApi richiama i template e vengono caricate all'interno della form per il caricamento di una nuova api.

4.13.3 Classi

- **uploadApi.html**

  Descrizione: contiene il template HTML relativo pagina di caricamento di un nuovo microservizio.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.uploadApi.html.

  Semplice relazione tra HTML e css.

- **uploadApi.css**

  Descrizione: contiene il template CSS relativo pagina di caricamento di un nuovo microservizio.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.uploadApi.css.

  Semplice relazione tra CSS e HTML

4.14 com.apim.app.view.viewApi

4.14.1 Informazione sul package

ViewApi è un contenitore di una pagina di visualizzazione dettagli di un microservizio. Qui si vedranno i commenti e le specifiche di ogni microservizio come la documentazione, lo stato, l'andamento e tutto ciò di cui fa riferimento.

4.14.2 Interazione con altri componenti

- com.apim.app.modelView.Component.viewApi.component .

La componente viewApi richiama i template e vengono caricate dinamicamente iniettate in qualsiasi pagina.

4.14.3 Classi

- **viewApi.html**

  Descrizione: contiene il template HTML relativo al contenitore dei dettagli del microservizio.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.viewApi.html.

  Semplice relazione tra HTML e css.

- **viewApi.css**

  Descrizione: contiene il template CSS relativo al contenitore dei dettagli del microservizio.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.viewApi.css.

  Semplice relazione tra CSS e HTML

4.15 com.apim.app.view.search-page

4.15.1 Informazione sul package

Search-page, come suggerisce il nome, è la pagina su cui verranno visualizzati i risultati della ricerca, al suo interno verrà iniettato form per ricerche specifiche e la selezione della categoria.

4.15.2 Interazione con altri componenti

- com.apim.app.modelView.Component.search-page.component .

La componente search-page richiama i template e vengono caricati all'interno dell'infrastruttura dell'applicazione .

4.15.3 Classi

- **search-page.html**

  Descrizione: contiene il template HTML relativo alla pagina di ricerca.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.search-page.html.

  Semplice relazione tra HTML e css.

- **search-page.css**

  Descrizione: contiene il template CSS relativo alla pagina di ricerca.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.search-page.css.

  Semplice relazione tra CSS e HTML

4.16 com.apim.app.view.issue

4.16.1 Informazione sul package

Issue è la pagina di visualizzazione dell'andamento dell'account. In cui dei grafici mostrano come stanno andando i microservizi con le varie specifiche calcolate dal $Gateway_{g}$. Tutto questo grazie alla libreria *ng2-charts*.

4.16.2 Interazione con altri componenti

- com.apim.app.modelView.Component.issue.component .

La componente issue richiama i template e vengono caricati all'interno del sito.

4.16.3 Classi

- **issue.html**

  Descrizione: contiene il template HTML relativo alla pagina di presentazione delle statistiche.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.issue.html.

  Semplice relazione tra HTML e css.

- **issue.css**

  Descrizione: contiene il template CSS relativo alla pagina di presentazione delle statistiche.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.issue.css.

  Semplice relazione tra CSS e HTML

4.17 com.apim.app.view.administrator

4.17.1 Informazione sul package

Administrator è la pagina di gestione accessibile solo da un utente amministratore. In questa pagina vi saranno i principali tools che un amministratore potrà eseguire per modificare un utente, microservizio; eliminarli ed altro.

4.17.2 Interazione con altri componenti

- com.apim.app.modelView.Component.administrator.component .

La componente administrator richiama i template e vengono caricati all'interno del sito usufruibili solo dall'amministratore.

4.17.3 Classi

- **administrator.html**

  Descrizione: contiene il template HTML relativo alla pagina di amministrazione.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.administrator.css .

  Semplice relazione tra HTML e css.

- **administrator.css**

Descrizione: contiene il template CSS relativo alla pagina di amministrazione.

Relazioni con altre classi:

- com.apim.app.modelView.Component.administrator.html .

Semplice relazione tra CSS e HTML.

4.18 com.apim.app.view.buyApi

4.18.1 Informazione sul package

buyApi è la pagina dove è possibile acquistare una nuova Api. Mostra le caratteristiche principali dell'applicazione, le condizioni generali ed un button dove è possibile effettivamente procedere all'acquisto.

4.18.2 Interazione con altri componenti

- com.apim.app.modelView.Component.buyApi.component .

La componente buyApi richiama i template e vengono caricati all'interno del sito al momento della richiesta di acquisto di un'api.

4.18.3 Classi

- **buyApi.html**

  Descrizione: contiene il template HTML relativo alla pagina di acquisto di un microservizio.

  Relazioni con altre classi:

  - com.apim.app.modelView.Component.buyApi.css.

  Semplice relazione tra HTML e CSS.

- **buyApi.css**

  Descrizione: contiene il template CSS relativo alla pagina di acquisto di un microservizio.

  Relazioni con altre classi:

  - com.apim.app.modelView.Component.buyApi.html.

  Semplice relazione tra CSS e HTML.

4.19 com.apim.app.view.passRecovery

4.19.1 Informazione sul package

Pagina in cui è possibile recuperare le informazioni e la password tramite mail.

4.19.2 Interazione con altri componenti

- com.apim.app.modelView.Component.passRecovery.component .

La componente passRecovery richiama i template e vengono caricati all'interno del client mostrando le form per richiesta di recupero password.

4.19.3 Classi

- **passRecovery.html**

  Descrizione: contiene il template HTML relativo alla richiesta di recupero password.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.passRecovery.css.

  Semplice relazione tra HTML e CSS.

- **passRecovery.css**

  Descrizione: contiene il template CSS relativo alla richiesta di recupero password.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.passRecovery.html.

  Semplice relazione tra CSS e HTML.

4.20 com.apim.app.view.buyCredits

4.20.1 Informazione sul package

buyCredits ti permette di acquistare nuovi crediti e caricare gli ApiCredits del tuo profilo personale.

4.20.2 Interazione con altri componenti

- com.apim.app.modelView.Component.buyCredits.component .

La componente buyCredits richiama i template e vengono caricati all'interno del client mostrando le pagine dove è possibile acquistare nuovi crediti.

4.20.3 Classi

- **buyCredits.html**

  Descrizione: contiene il template HTML relativo alla richiesta di acquisto crediti.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.buyCredits.css.

  Semplice relazione tra HTML e CSS.

- **buyCredits.css**

  Descrizione: contiene il template CSS relativo alla richiesta di acquisto crediti.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.passRecovery.html.

  Semplice relazione tra CSS e HTML.

4.21 com.apim.app.view.compare

4.21.1 Informazione sul package

compare ti permetterà di comparare le specifiche tecniche e social di due Api.

4.21.2 Interazione con altri componenti

- com.apim.app.modelView.Component.compare.component .

La componente buyCredits richiama i template e vengono caricati all'interno del client mostrando le pagine dove è possibile acquistare nuovi crediti.

4.21.3 Classi

- **compare.html**

  Descrizione: contiene il template HTML che mostrerà due applicazioni da comparare.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.compare.css.

  Semplice relazione tra HTML e CSS.

- **compare.css**

  Descrizione: contiene il template CSS che mostrerà due applicazioni da comparare.

  Relazioni con altre classi:

  – com.apim.app.modelView.Component.compare.html.

  Semplice relazione tra CSS e HTML.

4.22 com.apim.app.model View

4.22.1 Informazioni sul package

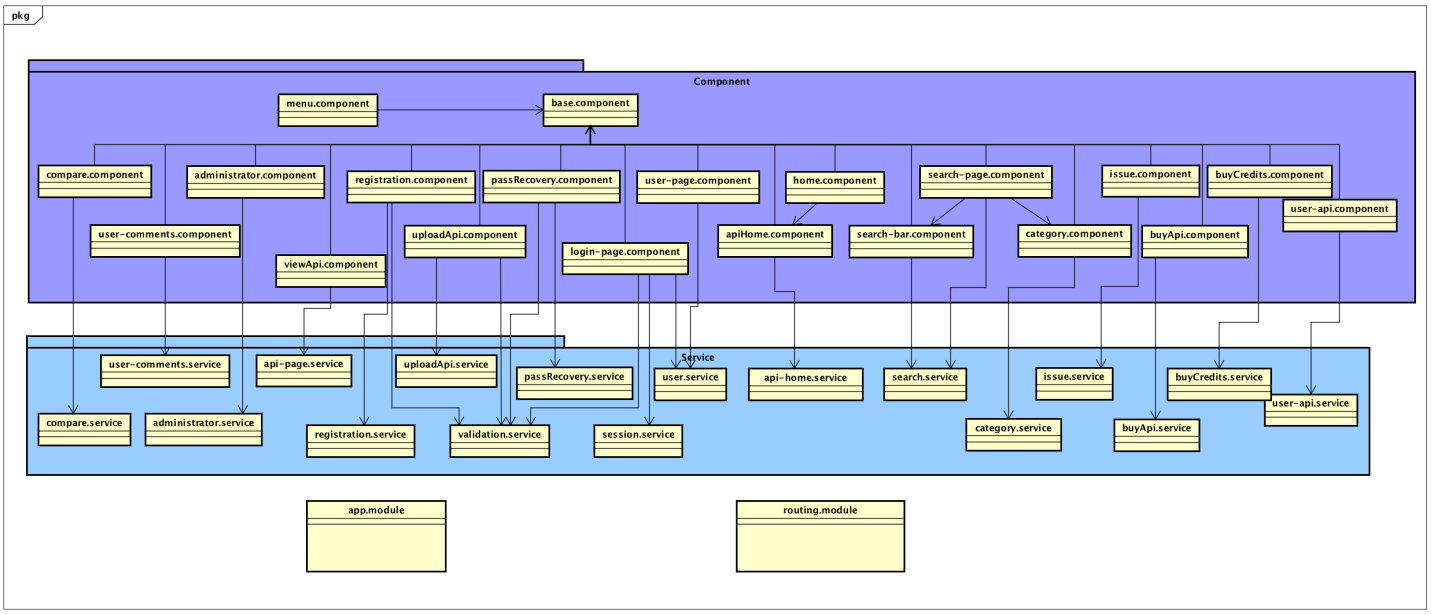


Figura 4: Schema package front end - model View

La model-view racchiude tutti i componenti e i servizi in Angular2 utili a modellare le classi. Richiama inoltre le funzioni del Back End tramite chiamate REST. Oltre a manipolare e invocare i metodi ha la proprietà della gestione logica della vista.

4.22.2 Package contenuti

- com.apim.app.modelView.service;
- com.apim.app.modelView.component.

4.22.3 Interazione con altri componenti

- com.apim.app.view .

Ogni componente dell'applicazione richiama i template grafici per gestirli. Ogni component ha associata la sua view specifica.

- com.apim.app.model .

Interagendo con il modello ed invocando metodi e classi i servizi sono a stretto contatto anche con la parte model dell'applicazione.

4.22.4 Classi

Le uniche due classi contenute nella root del Model View sono:

- **app.module**

Descrizione: Componente del sistema che regola e instrada l'avvio dell'applicazione e tutte le componenti.

Relazioni con altre classi:

- Vi sono collegamenti gestisti unicamente da Angular2 e non rappresentabile tramite schema UML.

- **routing.module**

Descrizione: Componente del sistema che regola e instrada l'avvio dell'applicazione e tutte le componenti.

Relazioni con altre classi:

- Vi sono collegamenti gestisti unicamente da Angular2 e non rappresentabile tramite schema UML.

4.23 com.apim.app.modelView.component

4.23.1 Informazione sul package

Questo package$_g$ è il cuore del Front End e racchiude tutti i componenti. Ogni componente gestisce e decora degli oggetti metadata. Questi oggetti descrivono come i template HTML e le classi a cui fanno riferimento i componenti (model) lavorano insieme.

4.23.2 Classi

- **base.component**

Descrizione: È il componente principale su cui poggiano tutto il resto dei componenti. Lui veicola i vari selettori nella pagina principale.

Relazioni con altre classi:

- com.apim.app.modelView.menu.component;
- com.apim.app.modelView.home.component;
- com.apim.app.modelView.apiHome.component;
- com.apim.app.modelView.user-comments.component;
- com.apim.app.modelView.user-api.component;
- com.apim.app.modelView.registration.component;
- com.apim.app.modelView.user-page.component;
- com.apim.app.modelView.login-page.component;
- com.apim.app.modelView.uploadApi.component;
- com.apim.app.modelView.category.component;
- com.apim.app.modelView.viewApi.component;
- com.apim.app.modelView.search-page.component;
- com.apim.app.modelView.search-bar.component;
- com.apim.app.modelView.issue.component;

- com.apim.app.modelView.administrator.component;
- com.apim.app.modelView.buyApi.component;
- com.apim.app.modelView.passRecovery.component;
- com.apim.app.modelView.buyCredits.component;
- com.apim.app.modelView.compare.component.

Ogni componente è quindi richiamata nella base.component per richiamare i suoi template ed essere iniettati nell'applicazione.

- **menu.component**

  Descrizione: Questa classe si occupa di creare il menu di navigazione.

  Relazioni con altre classi:

  - com.apim.app.modelView.base.component;
  - com.apim.app.view.menu.

  La classe fa riferimento al componente base e al suo template nella view.

- **login-page.component**

  Descrizione: Classe che crea la pagina di login.

  Relazioni con altre classi:

  - com.apim.app.modelView.service.session.service;
    Collegata ai servizi di sessione che vanno ad interrogare il DB ed controllare ogni tot tempo che l'utente sia connesso.
  - com.apim.app.modelView.service.user.service;
    Richiama gli utenti per fare un controllo e verificare che l'utente sia presente nel DB.
  - com.apim.app.modelView.service.validation.service;
    Richiama le funzionalità di controllo, ad esecuzione, dell'inserimento nelle form da parte dell'utente.
  - com.apim.app.modelView.base.component;
  - com.apim.app.view.login-page.

- **user-comments.component**

  Descrizione: Classe che gestisce il commento di un utente.

  Relazioni con altre classi:

  - com.apim.app.modelView.service.user-comments.service;
    Collega i servizi commenti utente, come inserimento commento, visualizza.
  - com.apim.app.modelView.base.component;
  - com.apim.app.view.user-comments.

- **user-api.component**

Descrizione: Classe che gestisce i microservizi di un singolo utente (nella sua pagina).

Relazioni con altre classi:

- com.apim.app.modelView.service.user-api.service;
  Collega i servizi di visualizzazione dei microservizi che interagiscono con un singolo utente.
- com.apim.app.modelView.base.component;
- com.apim.app.view.user-api.

- **viewApi.component**

  Descrizione: Classe che permette di richiamare il servizio di visualizzazione delle api.

  Relazioni con altre classi:

  - com.apim.app.modelView.service.api-page.service;
    Collega i servizi di visualizzazione delle api nella home, nelle categorie, nella pagina utente e nella ricerca.
  - com.apim.app.modelView.base.component;
  - com.apim.app.view.viewApi.

- **administrator.component**

  Descrizione: Componente che inietta e coordina tutte le funzioni di amministrazione.

  Relazioni con altre classi:

  - com.apim.app.modelView.service.administrator.service;
    Collega i servizi di gestione della parte di amministratore.
  - com.apim.app.modelView.base.component;
  - com.apim.app.view.administrator.

- **registration.component**

  Descrizione: Classe che conduce i dati relativi ottenuti dalla form nel template HTML relativo all'utente.

  Relazioni con altre classi:

  - com.apim.app.modelView.service.registration.service;
    Richiama servizi di inserimento nuovo utente dal service e controlli form.
  - com.apim.app.modelView.service.validation.service;
    Richiama le funzionalità di controllo, ad esecuzione, dell'inserimento nelle form da parte dell'utente.
  - com.apim.app.modelView.base.component;
  - com.apim.app.view.registration.

- **user-page.component**

Descrizione: Componente che sovrintende la pagina utente con tutte le informazioni del caso.

Relazioni con altre classi:

- com.apim.app.modelView.service.user-page.service;
  Richiama le funzionalità di visualizzazione del credito e i dati specifici personali.
- com.apim.app.modelView.base.component;
- com.apim.app.view.user-page.

- **issue.component**

Descrizione: Componente che ripartisce statistiche dirette dell'utente e dei microservizi a lui collegati.

Relazioni con altre classi:

- com.apim.app.modelView.service.issue.service;
  Richiama le funzionalità di visualizzazione delle statistiche dei microservizi.
- com.apim.app.modelView.base.component;
- com.apim.app.view.issue.

- **home.component**

Descrizione: È il componente strutturale e generico della pagina principale home. Richiama semplicemente e ordina gli altri componenti presenti come apiHome e cioè la lista dei microservizi.

Relazioni con altre classi:

- com.apim.app.modelView.apiHome.component;
  La componente home non richiama alcun servizio ma ha solo il ruolo di coordinare la home Page È la apiHome.component che si preoccupa delle funzionalità service.
- com.apim.app.modelView.base.component;
- com.apim.app.view.home.

- **apiHome.component**

Descrizione: Componente che inietta le funzioni che ritornano liste di array di microservizi.

Relazioni con altre classi:

- com.apim.app.modelView.service.apiHome.service;
  Richiama le funzionalità di visualizzazione del credito e i dati specifici personali.
- com.apim.app.modelView.home.component;
- com.apim.app.modelView.base.component;
- com.apim.app.view.apiHome.

- **search-page.component**

Descrizione: Super-componente che sta a dirigere la pagina di ricerca.

Relazioni con altre classi:

- com.apim.app.modelView.service.search.service;
  Richiama le funzionalità di visualizzazione del credito e i dati specifici personali.
- com.apim.app.modelView.search-bar.component;
  Search-page sollecita le funzionalità del componente search-bar statico in tutte le pagine della web Application.
- com.apim.app.modelView.category.component;
  La componente di selezione categoria renderà le ricerche più specifiche.
- com.apim.app.modelView.base.component;
- com.apim.app.view.user-page.

- **search-bar.component**

  Descrizione: Componente amministrativa della barra di ricerca progettata.

  Relazioni con altre classi:

  - com.apim.app.modelView.service.search.service;
    Richiama le funzionalità di visualizzazione del credito e i dati specifici personali.
  - com.apim.app.modelView.search-page.component;
    Search-page sollecita le funzionalità del componente search-bar statico in tutte le pagine della web Application.
  - com.apim.app.modelView.base.component;
  - com.apim.app.view.user-page.

- **category.component**

  Descrizione: Componente per la scelta specifica della categoria di un microservizio. Instrada le categorie e le loro funzionalità.

  Relazioni con altre classi:

  - com.apim.app.modelView.service.category.service;
    Richiama le operazioni di scelta delle categorie.
  - com.apim.app.modelView.search-page.component;
    Category viene implementata nella componente di ricerca per migliorare la sua funzionalità.
  - com.apim.app.modelView.base.component;
  - com.apim.app.view.category.

- **uploadApi.component**

  Descrizione: Classe che conduce i dati relativi ottenuti dalla form nel template HTML relativo ad un nuovo microservizio da caricare.

  Relazioni con altre classi:

- com.apim.app.modelView.service.uploadApi.service;
  Richiama le funzionalità come il caricamento di un nuovo microservizio e controllo degli input.
- com.apim.app.modelView.service.validation.service;
  Richiama le funzionalità di controllo, ad esecuzione, dell'inserimento nelle form da parte dell'utente.
- com.apim.app.modelView.base.component;
- com.apim.app.view.upload.

- **buyApi.component**

  Descrizione: Classe che contiene le richieste e incapsula l'acquisto di un microservizio.

  Relazioni con altre classi:

  - com.apim.app.modelView.service.buyApi.service;
    Richiama le funzionalità di acquisto di un nuovo microservizio.
  - com.apim.app.modelView.base.component;
  - com.apim.app.view.buyApi.

- **passRecovery.component**

  Descrizione: Componente di recupero password.

  Relazioni con altre classi:

  - com.apim.app.modelView.service.passRecovery.service;
    Richiama le funzionalità di recupero password di un nuovo microservizio.
  - com.apim.app.modelView.service.validation.service;
    Richiama le funzionalità di controllo, ad esecuzione, dell'inserimento nelle form da parte dell'utente.
  - com.apim.app.modelView.base.component;
  - com.apim.app.view.passRecovery.

- **buyCredits.component**

  Descrizione: Componente di acquisto di nuovi crediti da collegare al proprio profilo.

  Relazioni con altre classi:

  - com.apim.app.modelView.service.buyCredits.service;
    Richiama le funzionalità di acquisto di ApiCredits.
  - com.apim.app.modelView.base.component;
  - com.apim.app.view.buyCredits.

- **compare.component**

  Descrizione: Componente che dispone della possibilità di confronto tecnico e sociale di due microservizi.

Relazioni con altre classi:

- com.apim.app.modelView.service.compare.service;
  Richiama le funzionalità di comparazione tra microservizi.
- com.apim.app.modelView.base.component;
- com.apim.app.view.compare.

4.24 com.apim.app.modelView.service

4.24.1 Informazione sul package

Questo package$_g$ racchiude tutte le funzioni e i metodi utili al component per iniettare le funzionalità dove necessitano. Intendiamo seguire la politica di utilizzo di servizi separati per mantiene i componenti snelli e focalizzati al supporto della view e rende più facile per i test di unità testare i componenti con un servizio mock.
Tali servizi rendono al loro interno la possibilità di essere iniettati, seguendo la teoria del pattern Dependency Injection, attraverso i component.

4.24.2 Classi

- **session.service**

  Descrizione: Servizio che modella le funzioni di sessione.

  Relazioni con altre classi:

  - com.apim.app.modelView.component.login-page.component;
  - com.apim.app.model.user;
    Interroga il modello degli utenti per essere sicuro di poter avviare la sessione.
  - com.apim.app.model.data .
    Utilizza la data attuale e l'ora attuale per una timestamp ad uso della sessione.

- **user-comments.service**

  Descrizione: Classe che gestisce il commento di un utente.

  Relazioni con altre classi:

  - com.apim.app.modelView.component.user-comments.component;
  - com.apim.app.model.comment;
    Interroga il modello dei commenti (associato a quello degli utenti) per avere a disposizioni i capi dati del modello commento.

- **user-api.service**

  Descrizione: Classe che dispone delle funzioni di visualizzazione delle api.

  Relazioni con altre classi:

  - com.apim.app.modelView.component.user-api.component;

- com.apim.app.model.user;
  Interroga il modello degli utenti per un'associazione al singolo utente.
- com.apim.app.model.api .
  Raccoglie un array di Api e quindi necessita del modello specifico.
- com.apim.app.model.data .
  Utilizza la data attuale e l'ora attuale per una timestamp.

- **registration.service**

  Descrizione: Servizio che provvede al compito di registrazione.

  Relazioni con altre classi:

  - com.apim.app.modelView.component.registration.component;
  - com.apim.app.model.user;
    Carica grazie al modello nel DB una nuovo utente dalla componente di registrazione.

- **user.service**

  Descrizione: Macro servizio con la maggior parte delle funzionalità riportate al solo utente, utilizzato infatti da due componenti.

  Relazioni con altre classi:

  - com.apim.app.modelView.component.user-page.component;
  - com.apim.app.modelView.component.login-page.component;
  - com.apim.app.model.user;
    Le maggiori funzioni per un utente e i suoi campi dati devono essere usate dal servizio.

- **session.service**

  Descrizione: Servizio per le funzioni di sessione semplici.

  Relazioni con altre classi:

  - com.apim.app.modelView.component.login-page.component;
  - com.apim.app.model.data .
    Utilizza la data attuale e l'ora attuale per tener traccia della sessione implementandolo alla stringa token.
  - com.apim.app.model.user;
    Richiama il database e in seguito il modello user per tener traccia della chiave con l'utente.

- **uploadApi.service**

  Descrizione: Servizio per le operazioni di caricamento nel DB di un nuovo microservizio.

  Relazioni con altre classi:

  - com.apim.app.modelView.component.uploadApi.component;

- com.apim.app.model.api .

  Il caricamento e il controllo della form di una nuova api necessita del modello della api stessa per interpretare i metadati.

- com.apim.app.model.policy;

  Una funzione adeguata permetterà la scelta della policy di vendita e quindi interrogherà il modello per le specifiche.

- **category.service**

  Descrizione: Servizio per l'indirizzamento alla categoria di scelta e l'esposizione delle possibili tipologie.

  Relazioni con altre classi:

  - com.apim.app.modelView.component.category.component;
  - com.apim.app.model.category .

    Il modello intercetta le categorie ne riporta un array di scelta.

- **api-page.service**

  Descrizione: Servizio per la selezione delle api e la loro gestione .

  Relazioni con altre classi:

  - com.apim.app.modelView.component.api-page.component;
  - com.apim.app.model.api .

    Il modello di riferimento è quello delle api per il trattamento dei dati in maniera corretta.

- **search.service**

  Descrizione: Servizio che richiama dal Back End e collega al Front End le funzioni di ricerca specifica.

  Relazioni con altre classi:

  - com.apim.app.modelView.component.search-bar.component;
  - com.apim.app.modelView.component.search-page.component;
  - com.apim.app.model.api ;

    La ricerca per stringa semplice è prioritariamente usata per le api, e nei loro campi.

  - com.apim.app.model.user;

    Successivamente è possibile, in particolare per l'utente amministratore, fare una ricerca per utente.

- **issue.service**

  Descrizione: Servizio che richiama le statistiche dei microservizi per creare dei grafici grazie al componente.

  Relazioni con altre classi:

- com.apim.app.modelView.component.issue.component;
- com.apim.app.model.apiStat ;
  Le statistiche sono formattate secondo lo schema del modello apiStat .
- com.apim.app.model.user .
  Ogni statistica fa riferimento al singolo utente loggato. Non è possibile consultare grafici statici altrui.

- **administrator.service**

  Descrizione: Servizio con le principali funzionalità ad uso amministrativo.

  Relazioni con altre classi:

  - com.apim.app.modelView.component.administrator.component;
  - com.apim.app.model.api ;
    L'amministratore ha la possibilità grazie al pattern e ai servizi resi disponibili la modifica di tutte le api modellate secondo suddetto modello.
  - com.apim.app.model.user .
    Ugualmente per le modifiche che può applicare a qualsiasi user.

- **buyApi.service**

  Descrizione: Servizio che espone le principali funzionalità e metodi di acquisto di un microservizio.

  Relazioni con altre classi:

  - com.apim.app.modelView.component.buyApi.component;
  - com.apim.app.model.api ;
    Il microservizio deve essere comprato e creato da un modello Api.
  - com.apim.app.model.user .
    Ogni microservizio è associato ad un utente (User) e va trattato, ad esempio il suo conto.

- **passRecovery.service**

  Descrizione: Servizio permette le principali operazioni di recupero password.

  Relazioni con altre classi:

  - com.apim.app.modelView.component.passRecovery.component;
  - com.apim.app.model.user .
    Il servizio deve approcciarsi con un utente per poter trattare i dati, come ad esempio mail e password.

- **buyCredits.service**

  Descrizione: Servizio in cui vi sono funzionalità specifiche per l'acquisto e la gestione di denaro/ApiCredits.

  Relazioni con altre classi:

- com.apim.app.modelView.component.buyCredits.component;
- com.apim.app.model.user .

  Il servizio andrà a interagire con il numero di carta di credito e conto presenti nel modello User.

- **compare.service**

  Descrizione: Servizio che espone metodi di confronto tra i parametri di due microservizi, sia questi tecnici o sociali.

  Relazioni con altre classi:

  - com.apim.app.modelView.component.compare.component;
  - com.apim.app.model.api .

    Il servizio utilizza i dati salvati all'interno di un oggetto avente modello Api.

- **validation.service**

  Descrizione: Servizio che controlla runtime gli errori di inserimento e dei caratteri nelle form. Oltre che lunghezza e altre restrizioni.

  Relazioni con altre classi:

  - com.apim.app.modelView.component.passRecovery.component;

    Il servizio controlla se la mail è corretta e presente nel DB.
  - com.apim.app.modelView.component.uploadApi.component;

    Il servizio controlla tutte le form di caricamento, se il nome è presente e valido, ecc.
  - com.apim.app.modelView.component.login.component;

    Il servizio controlla le due principali form di login e restituisce errori nel caso di negativa autenticazione.
  - com.apim.app.modelView.component.registration.component .

    Il servizio controlla tutte le form di registrazione.

4.25 com.apim.app.model

4.25.1 Informazioni sul package

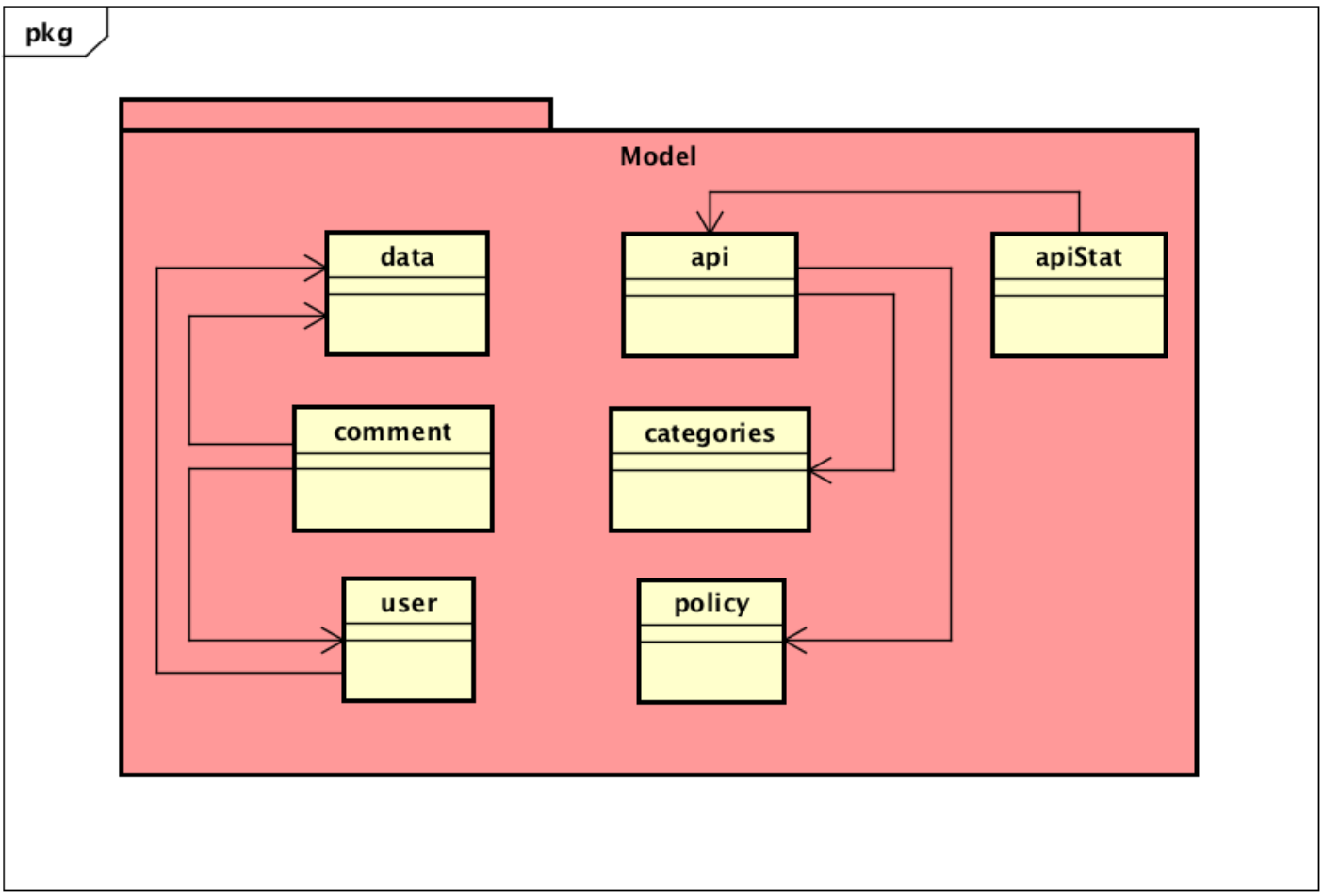


Figura 5: Schema package front end - model

La model è composta da tutti i tipi resi disponibile grazie al linguaggio di TypeScripting. Noi in ogni classe abbiamo dichiarato i tipi da noi utilizzati nell'applicazione e i loro campi dati per dare una forma a i dati passati sotto forma di JSON dal Back End.

Anche in questo caso, nella sezione attuale non verranno riportati in maniera descrittiva le iterazioni tra componenti già indicate in precedenza ma solo quelle tra modelli. Inoltre verranno descritte solo le dipendenze uscenti.

4.25.2 Classi

- **user**

  Descrizione: È il tipo utente, senza distinzione tra amministratore o utente loggato tranne per un booleano interno.

  Relazioni con altre classi:

  – com.apim.app.model.data ;

  Ogni utente presenta una data di nascita e questo porta ad utilizzare il modello data.

–

- **comment**

  Descrizione: È il tipo commento, utilizzabile dagli utenti e applicabile sui microservizi.

  Relazioni con altre classi:

  – com.apim.app.model.data ;
  Ogni commento ha di default un timestamp utilizzante la data del momento della pubblicazione.
  – com.apim.app.model.user .
  Ogni commento ha inoltre la signatura semplice e non completa dell'utente da lui descritto.

- **data**

  Descrizione: È il tipo data, utilizzato per formattare meglio la data per una maggior compatibilità con il DB e il Back End.

  Relazioni con altre classi:

  – com.apim.app.model.comment ;
  – com.apim.app.model.user .

- **api**

  Descrizione: È il tipo api, utilizzato per indicare i microservizi.

  Relazioni con altre classi:

  – com.apim.app.model.categories ;
  Ogni api è associata a una categoria di registrazione.
  – com.apim.app.model.policy .
  Ogni api ha inoltre la policy di vendita di cui è legata.

- **apiStat**

  Descrizione: È un tipo utilizzato per formattare al meglio le statistiche derivanti dal *Gateway*.

  Relazioni con altre classi:

  – com.apim.app.model.api ;
  Ogni apiStat fa riferimento a una singola api e alle suoi massimi o minimi risultati delle variabili.

- **categories**

  Descrizione: Modello che elenca e descrive i tipi di categoria per microservizi.

  Relazioni con altre classi:

  – com.apim.app.model.api .

- **policy**

  Descrizione: È il tipo caratterizzante le politiche di vendita di un api.

  Relazioni con altre classi:

  – com.apim.app.model.api .

api

5 Architettura Back End

5.1 Microservizi Jolie

Di seguito sono illustrate le varie componenti server, realizzato con architettura a microservizi.

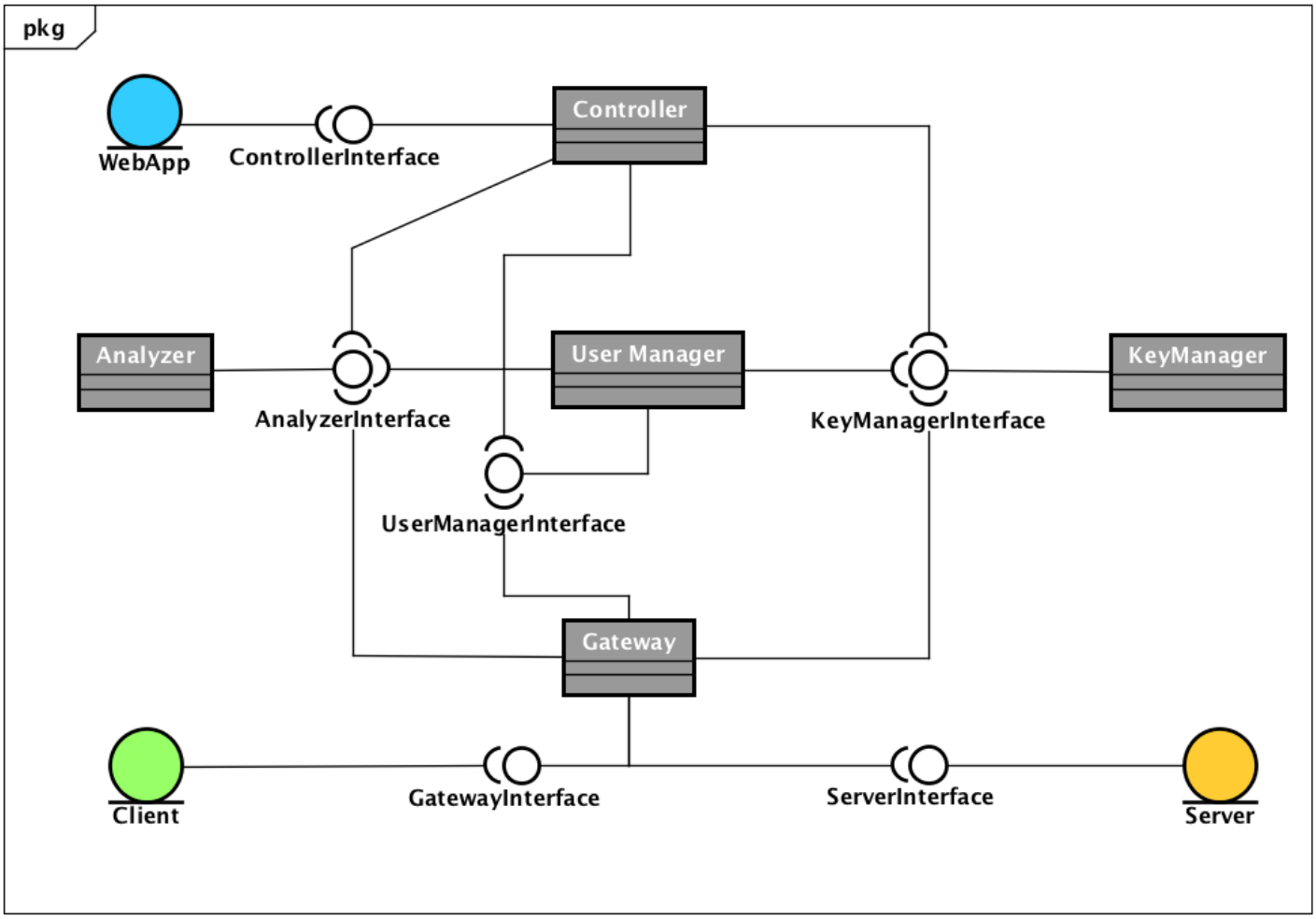


Figura 6: Struttura Back-End

- UserManagerInterface

  Descrizione: Interfaccia che denifisce le operazioni disponibili nel microservizio UserManager;

  Relazione con altri microservizi:

  – UserManager;

- UserManager

  Descrizione: Microservizio responsabile delle operazioni riguardanti gli utenti;

  Relazione con altri microservizi:

  - KeyManager;
  - Analyzer;

- KeyManagerInterface

  Descrizione: Interfaccia che denifisce le operazioni disponibili nel microservizio KeyManager;

  Relazione con altri microservizi:

  - KeyManager;

- KeyManager

  Descrizione: Microservizio responsabile delle operazioni riguardanti le Key;

  Relazione con altri microservizi:

- AnalyzerInterface

  Descrizione: Interfaccia che denifisce le operazioni disponibili nel microservizio Analyzer;

  Relazione con altri microservizi:

  - Analyzer;

- Analyzer

  Descrizione: Microservizio responsabile delle operazioni di raccolta ed elaborazione dei dati statistici sulle API;

  Relazione con altri microservizi:

  - AnalyzerInterface;

- GatewayInterface

  Descrizione: Interfaccia che denifisce le operazioni disponibili nel microservizio Gateway;

  Relazione con altri microservizi:

  - Gateway;

- Gateway

  Descrizione: Microservizio responsabile delle operazioni di verifica, analisi e reindirizzamento delle connessioni da un client verso il server del microservizio associato alla API;

  Relazione con altri microservizi:

  - Analyzer;
  - UserManager;
  - KeyManager;
  - CourierTemplate;
  - Courier;

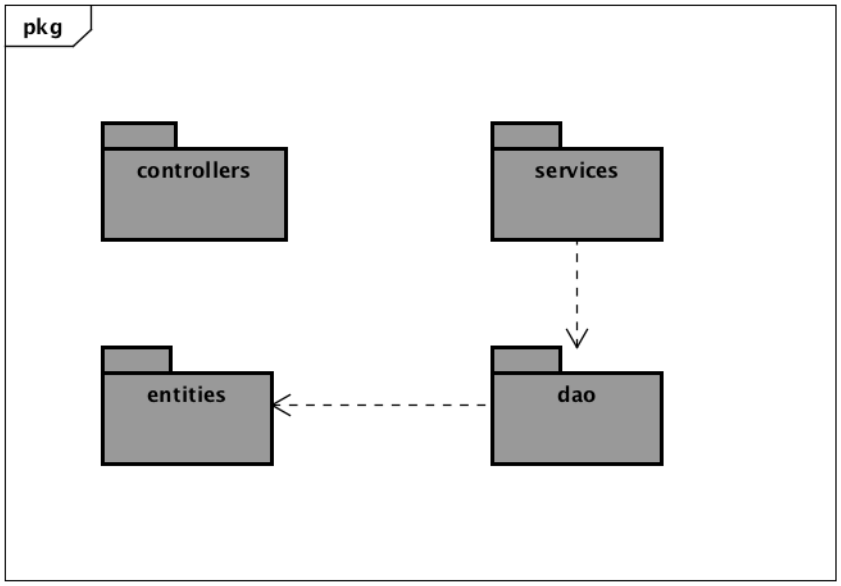


Figura 7: Server

5.2 com.apim.server

5.2.1 Informazioni sul Package

Contiene il back end dell'applicazione.

5.2.2 Package contenuti

- *.APIManager;
- *.Analyzer;
- *.CommentManager;
- *.gateway;
- *.UserManager;
- *.daos;
- *.entities;

5.3 com.apim.server.gateway

5.3.1 Informazioni sul Package

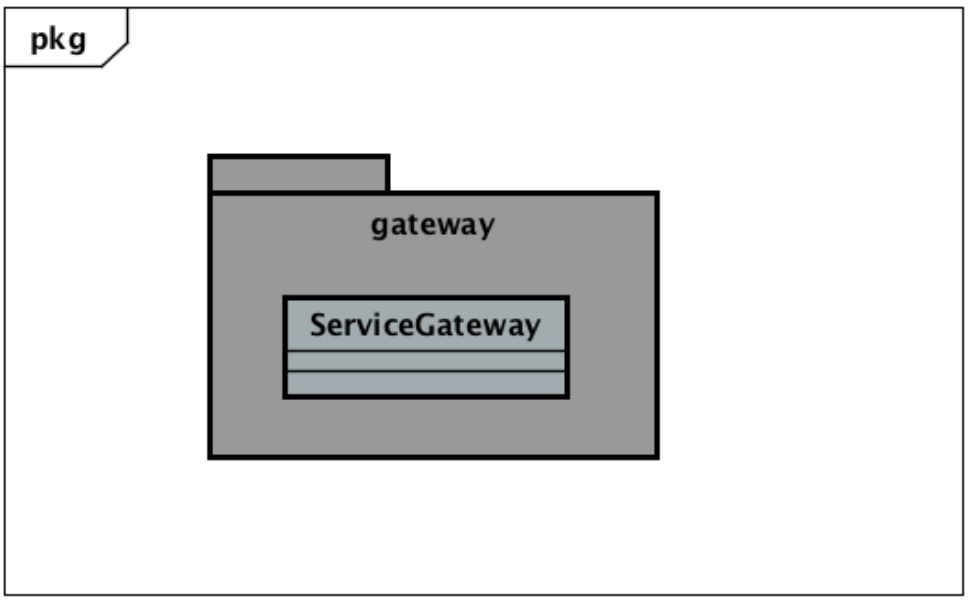


Figura 8: gateway

Package contenente le classi che implementano le funzionalità del gateway;

5.3.2 Classi

- **ServiceGateway**

  Descrizione: Questa classe contiene i metodi necessari a verificare la presenza della API invocata nel database, verificare l'esistenza effettiva del server del microservizio e infine inoltrare la richiesta del client;

  Relazione con altre classi:

  - com.apim.server.keyManager.*;
  - com.apim.server.daos.DaoInterface;
  - com.apim.server.Analyzer.*;
  - com.apim.server.entities.User;
  - com.apim.server.entities.API;
  - com.apim.server.entities.Key;

5.3.3 Interazioni con altri componenti

- com.apim.server.controllers;
- com.apim.server.entities;
- com.apim.server.keyManager;
- com.apim.server.dataAnalysis;

5.4 com.apim.server.Analyzer

5.4.1 Informazioni sul Package

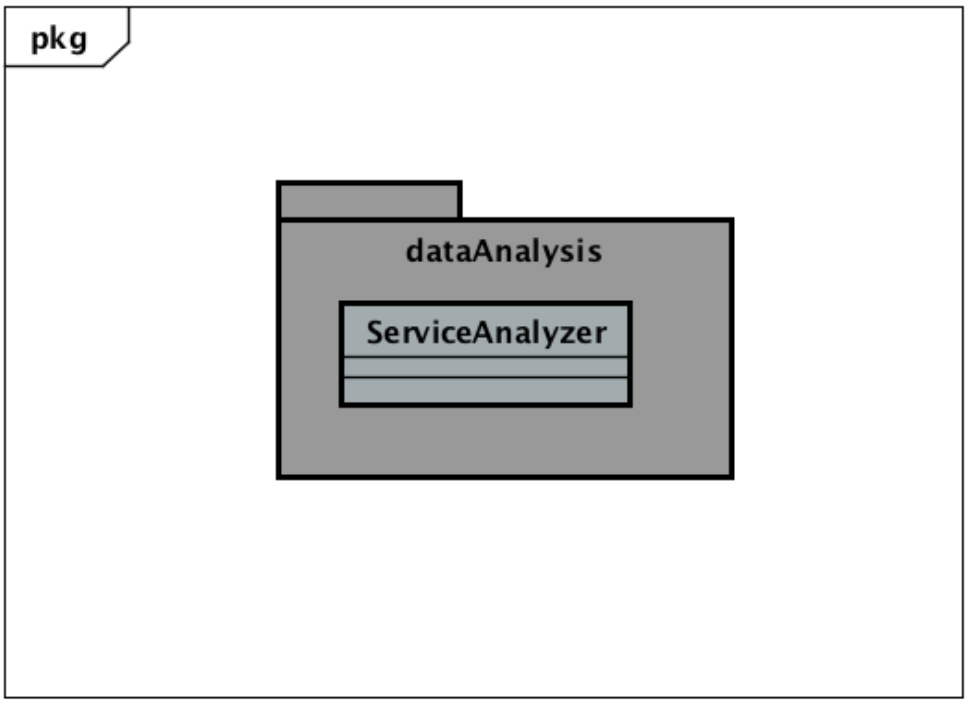


Figura 9: Analyzer

Package contenente le classi che implementano le funzionalità di analisi dei dati;

5.4.2 Classi

- **AnalyzerInterface**

  Descrizione: Questa classe contiene i metodi necessari all'analisi dei pacchetti JSON ritornati dal server del micro servizio, al fine di recuperare dati, quali dimensione, tempo di risposta, eccetera;

  Relazione con altre classi:

  - com.apim.server.daos.DaoInterface;
  - com.apim.server.entities.User;
  - com.apim.server.entities.API;
  - com.apim.server.entities.Key;
  - com.apim.server.utilities.ObjectSizeFetcher;

5.4.3 Interazioni con altri componenti

- com.apim.server.services.controllers;
- com.apim.server.services.entities;
- com.apim.server.services.gateway;

5.5 com.apim.server.keyManager

5.5.1 Informazioni sul Package

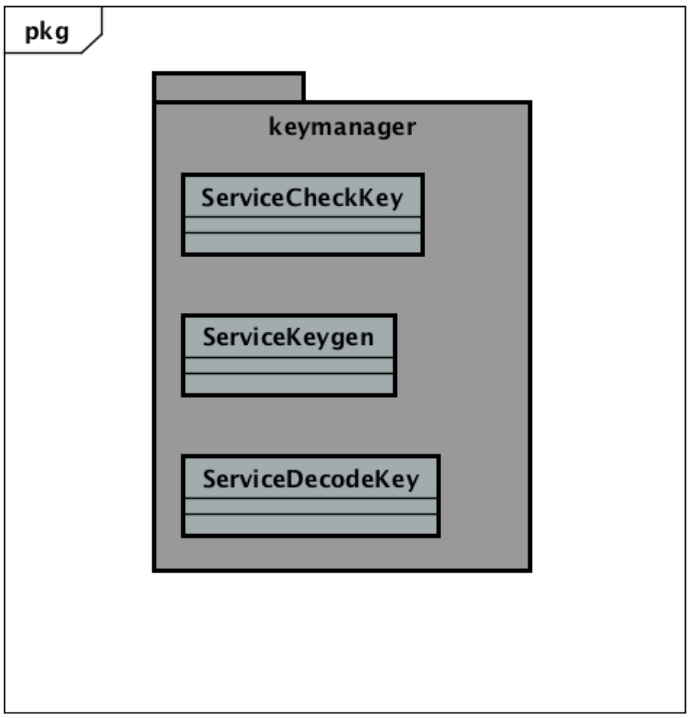


Figura 10: Key Manager

Package contenente le classi che implementano le funzionalità di gestione delle keys;

5.5.2 Classi

- **KeyManagerInterface**

  Descrizione: Questa classe contiene i metodi necessari alla creazione di una nuova key;

  Relazione con altre classi:

  - com.apim.server.daos.DaoInterface;
  - com.apim.server.entities.User;
  - com.apim.server.entities.API;
  - com.apim.server.entities.Key;

- **KeyManager**

  Descrizione: Questa classe contiene i metodi necessari alla verifica della validità della key, ossia correttamente associata ad un microservizio e non scaduta;

  Relazione con altre classi:

  - com.apim.server.daos.DaoInterface;
  - com.apim.server.entities.User;
  - com.apim.server.entities.Key;

5.5.3 Interazioni con altri componenti

- com.apim.server.services.entities;
- com.apim.server.services.gateway;

5.6 com.apim.server.userManager

5.6.1 Informazioni sul Package

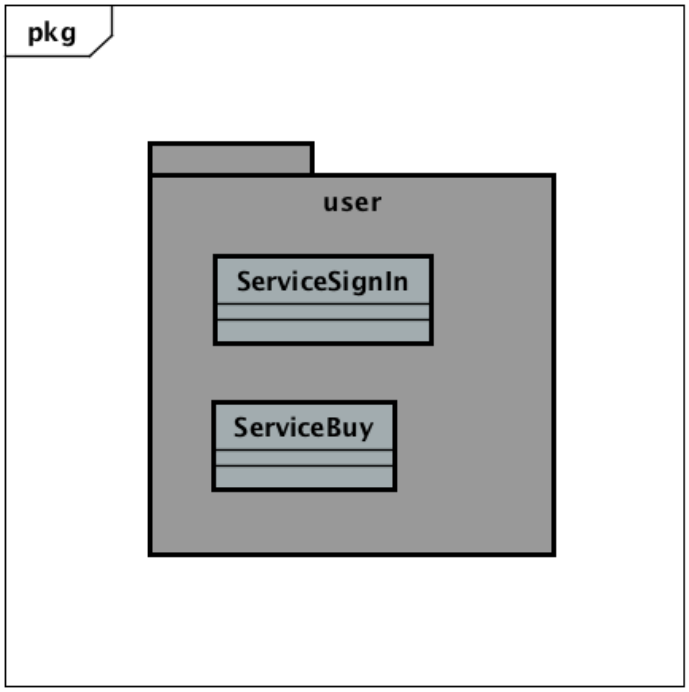


Figura 11: UserManager

Package contenente le classi che implementano le funzionalità disponibili per un utente normale;

5.6.2 Classi

- **UserManageInterface**

  Descrizione: Questa classe contiene i metodi necessaria alla registrazione dell'utente;

  Relazione con altre classi:

  - com.apim.server.daos.DaoInterface;
  - com.apim.server.entities.User;
  - com.apim.server.entities.UserController;

- **UserManager**

  Descrizione: Questa classe contiene i metodi necessari ad acquistare una API dal market;

  Relazione con altre classi:

  - com.apim.server.daos.DaoInterface;

- com.apim.server.entities.User;
- com.apim.server.entities.Acquisto;
- com.apim.server.entities.API;
- com.apim.server.entities.Key;

5.6.3 Interazioni con altri componenti

- com.apim.server.services.entities;
- com.apim.server.services.gateway;

5.7 com.apim.server.daos

5.7.1 Informazioni sul Package

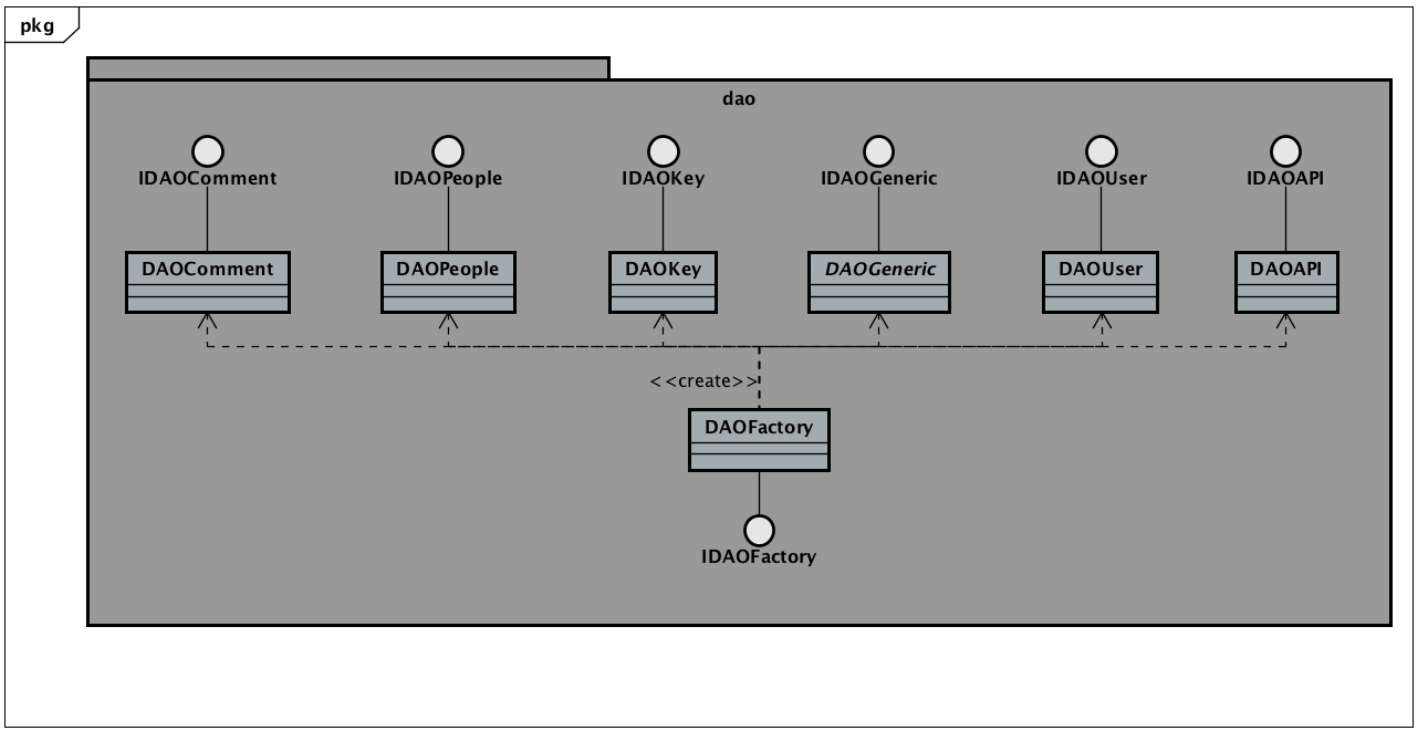


Figura 12: DAO

Package contenente le classi utilizzate per accedere direttamente al database;

5.7.2 Classi

- **DaoInterface**

  Descrizione: Interfaccia per la creazione degli oggetti che andranno ad eseguire l'accesso diretto al database;

  Relazione con altre classi:

  - com.apim.server.daos.DaoApi;
  - com.apim.server.daos.DaoUser;

  - com.apim.server.daos.DaoComment;
  - com.apim.server.daos.DaoKey;

- **DaoApiInterface**

  Descrizione: Questa classe esegue l'accesso alla tabella API del database;

  Relazione con altre classi:

  - com.apim.server.daos.DaoApi;

- **DaoApi**

  Descrizione: Implementazione di DaoApiInterface;

  Relazione con altre classi:

  - com.apim.server.daos.DaoApiInterface;
  - com.apim.server.daos.DaoInterface;

- **DaoUserInterface**

  Descrizione: Questa classe esegue l'accesso alla tabella User del database;

  Relazione con altre classi:

  - com.apim.server.daos.DaoUser;

- **DaoUser**

  Descrizione: Implementazione di DaoUser;

  Relazione con altre classi:

  - com.apim.server.daos.DaoUserInterface;
  - com.apim.server.daos.DaoInterface;

- **DaoCommentInterface**

  Descrizione: Questa classe esegue l'accesso alla tabella Comment del database;

  Relazione con altre classi:

  - com.apim.server.daos.DaoComment;

- **DaoComment**

  Descrizione: Implementazione di DaoCommentInterface;

  Relazione con altre classi:

  - com.apim.server.entities.*;
  - com.apim.server.daos.DaoCommentInterface;
  - com.apim.server.daos.DaoInterface;

- **DaoKeyInterface**

  Descrizione: Questa classe esegue l'accesso alla tabella Key del database;

  Relazione con altre classi:

  – com.apim.server.daos.DaoInterface;

- **DaoKey**

  Descrizione: Implementazione di DaoKeyInterface;

  Relazione con altre classi:

  – com.apim.server.entities.*;
  – com.apim.server.daos.DaoKeyInterface;
  – com.apim.server.daos.DaoInterface;

5.7.3 Interazioni con altri componenti

- com.apim.server.services.controllers;
- com.apim.server.services.entities;

5.8 com.apim.server.APIManager

5.8.1 Informazioni sul package

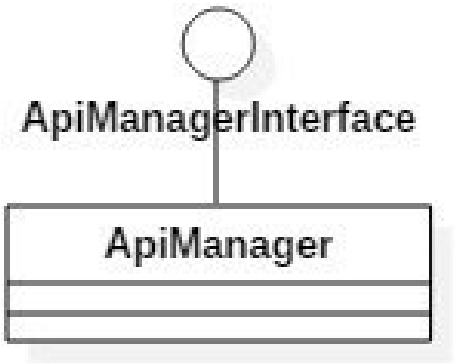


Figura 13: APIManager

Package contenente le classi che permettono di inserire, modificare, eliminare, visualizzare le API da parte di un admin.

5.8.2 Classi

- **APIManagerInterface**

  Descrizione: Interfaccia per mettere ad un admin di gerstire le API

  Relazione con altre classi:

- com.apim.server.daos.APIManager
- **APIManager**

  Descrizione: Implementazione dell'interfaccia APIManager;

  Relazione con altre classi:

  - com.apim.server.daos.APIManagerInterface;
  - com.apim.server.daos.DaoInterface;

5.9 com.apim.server.CommentManager

5.9.1 Informazioni sul package

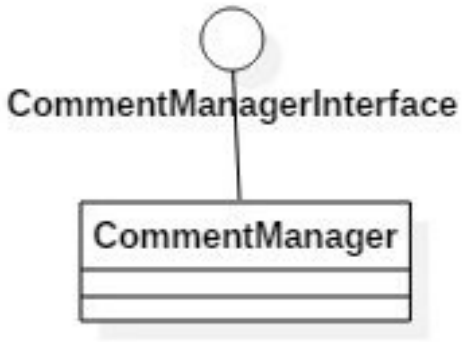


Figura 14: CommentManager

Package che contiene le classi che permettono di gestire i commenti.

5.9.2 Classi

CommentManagerInterface

- **Descrizione :** Interfaccia che gestisce l'inserimento, la modifica o la cancellazione di un commento;
- **Relazione con altre classi:**
  com.apim.server.CommentManager;

CommentManager

- **Descrizione:** Implementazione dell'interfaccia CommentManagerInterface;
- **Relazione con altre classi:**
  com.apim.server.CommentManagerInterface;

6 Architettura Database

In questa sezione viene descritta la progettazione del database che conterra' tutte le informazioni utili al nostro sistema. Abbiamo scelto di utilizzare un database relazionale.

6.1 Progettazione concettuale

Nella progettazione concettuale viene modellata la realta' da rappresentare. Sono state individuate le classi e le relazioni tra di esse, definendo cosi' la struttura che avra' il database.

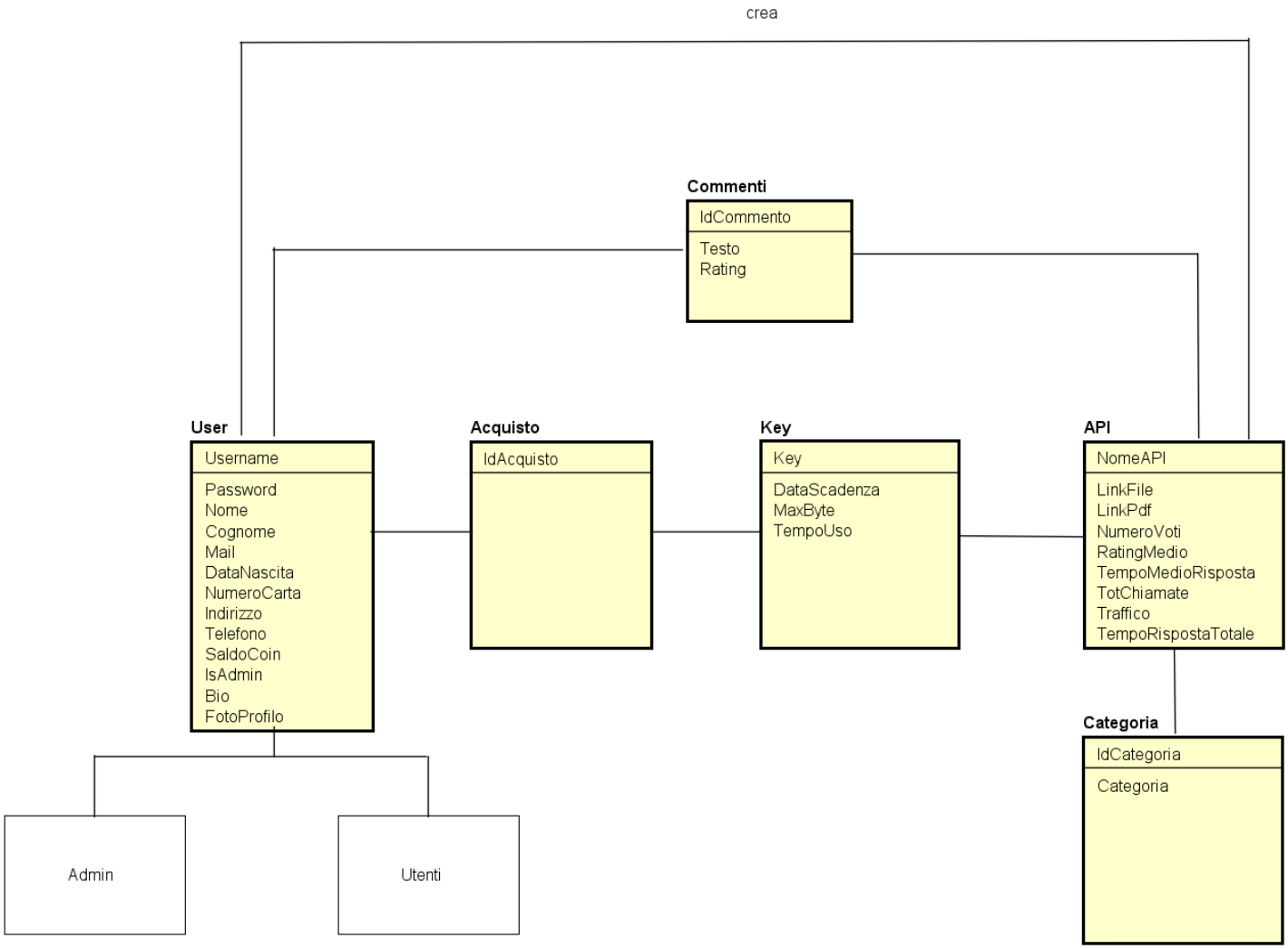


Figura 15: Progettazione Concettuale

6.1.1 Schema concettuale

6.1.2 Classi

User
Questa e' l'entita' che rappresenta tutti gli utenti che utilizzeranno il software, siano essi utenti normali o amministratori. Questa entita' come si vede dallo schema e' superclase di Utenti e di Admin. Gli attributi sono:

Attributo	Tipo	Vincolo
Username	String	PRIMARY KEY
Password	String	null
Nome	String	null
Cognome	String	null
DataNascita	Date	null
Mail	String	null
NumeroCarta	String	null
Indirizzo	String	null
Telefono	String	null
FotoProfilo	Mediumblob	null
Bio	String	null
SaldoCoin	Double	null
IsAdmin	Bool	null

Utenti

Questa entita' rappresenta gli utenti normali che utilizzeranno la piattaforma. E' una sottoclasse di **Users** e quindi eredita i suoi attributi. In particolare il campo **Bio** contiene una piccola biografia con valutazione dell'utente.

Admin

Questa entita' invece rappresenta gli amministratori che gestiranno la piattaforma. Come la precedente e' una sottocalsse di **Users** e ne eredita gli attributi. Per gli admin è stato aggiunto il campo **IsAdmin** per verificare quando un utente che accede è un amministratore.

API

Questa entita' rappresenta le API presenti nel sistema. Gli attributi sono:

	Attributo	Tipo	Vincolo
Nome API	String	PRIMARY KEY	
LinkFile	String	null	
LinkPdf	String	null	
NumeroVoti	Int	null	
RatingMedio	Double	null	
TempoMedioRisposta	double	null	
TotaleChiamate	Int	null	
Traffico	Double	null	
TempoRispostaTotale	Double	null	

In particolare in questa tabella sono contenute le statistiche di ogni API come il **Tempo Medio Risposta**, il **Traffico** di dati che comporta l'tilizzo di quel microservizio, il numero **Totale Chiamate** e il **Tempo Risposta Totale**.

Key

Questa entita' rappresenta le chiavi che verranno assegnate ad un utente quando acquista un microservizio per potervi accedere ed utilizzarlo. Gli attributi sono:

	Attributo	Tipo	Vincolo
Key	String	PRIMARY KEY	
DataScadenza	Date	null	
MaxByte	Double	null	
TempoUso	Double	null	

In particolare gli attributi **DataScadenza, MaxByte e TempoUso** sono i limiti entro i quali la chiave puo' essere utilizzata. Secondo la policy di acquisto di un microservizio, al raggiungimento di tale limite

la chiave scadra' e non permettera' piu' l'utilizzo di tale microservizio. *Commenti*
Questa entita' rappresenta i commenti ed il voto che gli utenti possono lasciare su ogni microservizio che hanno utilizzato. Gli attributi sono:

	Attributo	Tipo	Vincolo
IdCommento	Int	PRIMARY KEY	
Testo	String	null	
Rating	Int	null	

In particolare textbfRating sara' il giudizio che l'utente lascera' del microservizio utilizzato.

Categorie Questa entita' rappresenta la suddivisione in categorie dei vari microservizi. L'amministratore puo' aggiungere o togliere categorie a propria scelta. Gli attributi sono:

	Attributo	Tipo	Vincolo
IdCategoria	Int	PRIMARY KEY	
Categoria	String	null	

Acquisti Questa entita' contiene lo storico degli acquisti effettuati dagli utenti.

	Attributo	Tipo	Vincolo
IdAcquisto	Int	PRIMARY KEY	

6.1.3 Associazioni

- *User-Acquisto*: molteplicita'n - 1. Un utente puo' acquistare uno o piu' microservizi, un microservizio puo' esser acquistata da uno o piu' utenti.
- **User-API**: molteplicita' 1 - n. Un utente puo' mettere a disposizione uno o piu' microservizi, mentre un microservizio e' inserito nel database dal un solo utente.
- **User-Commenti**: molteplicita' 1 - n. Un utente puo' lasciare diversi commenti in diverse API, mentre un commento e' lasciato per forza da un solo utente;
- **API-Commenti**: molteplicita' 1 - n. Una API puo' avere diversi commenti, ma un commento puo' essere lasciato su una sola API.
- **Acquisto-Key**: molteplicita' 1 - 1. Ogni acquisto avra' associata la propria chiave.
- **API - Key**: molteplicita' 1 - n. Una API avra' ad essa associate piu' chiavi, mentre una chiave appartiene ad una sola API.
- **API-Categorire**: molteplicita' 1 - n. Ad una categoria appartengono una o piu' API, mentre una API appartiene ad una sola categoria.
- **Users-Key**: molteplicita' 1 -n. Un utente puo' possedere una o piu' chiavi, mentre una chiave sara' associata ad un solo utente.

6.2 Progettazione logico-relazionale

La progettazione logico-relazionale segue la progettazione concettuale. Da qui le classi o entita' verranno chiamate relazioni. In questa fase vengono inserite in ogni relazione le chiavi esterne per rappresentare le gerarchie e le associazioni.

6.2.1 Schema logico-relazionale

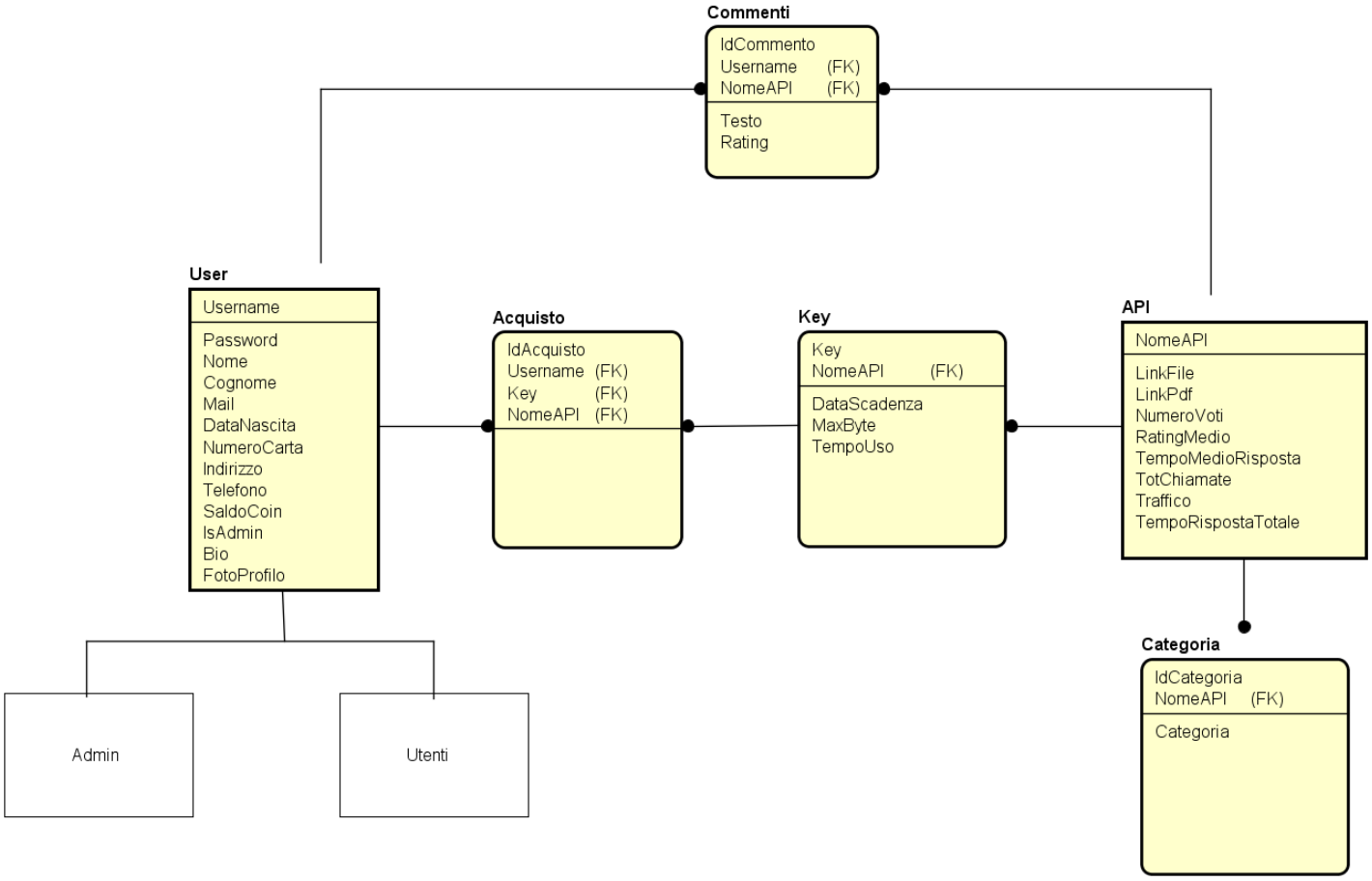


Figura 16: Progettazione Logica

6.2.2 Gerarchie

User - Utenti - Admin: La gerarchia e' stat implementata come tabella unica. La due sottoclassi **Utenti** e **Admin** sono comprese nella superclasse **Users**. Viene solamente aggiunto un attributo discriminante per distinguere un utente normale da un amministratore.

6.2.3 Relazioni

Categorie
Gli attributi sono:

Attributo	Tipo	Vincolo
IdCategoria	Int	PRIMARY KEY
Categoria	Varchar	null

Users
Relazione creata dall'implementazione tramite tabella unica della gerarchia User - Utenti - Admin. Gli attributi sono:

	Attributo	Tipo	Vincolo
Username	Varchar	PRIMARY KEY	
Password	Varchar	null	
Nome	Varchar	null	
Cognome	Varchar	null	
DataNascita	Date	null	
Mail	Varchar	null	
NumeroCarta	Varchar	null	
Indirizzo	Varchar	null	
Telefono	Varchar	null	
Bio	Varchar	null	
FotoProfilo	Mediumblob	null	
SaldoCoin	Double	null	
IsAdmin	Boolean	null	
NomeAPI	Varchar	FOREIGN KEY	

API
Gli attributi sono:

	Attributo	Tipo	Vincolo
NomeAPI	Varchar	PRIMARYKEY	
LinkFile	Varchar	null	
LinkPdf	Varchar	null	
NumeroVoti	Int	null	
RatingMedio	Double	null	
TempoMedioRisposta	Double	null	
TotaleChiamate	Int	null	
Traffico	Doule	null	
TempoRispostaTotale	Double	null	
Username	Varchar	FOREING KEY	
Categoria	Varchar	FOREIGN KEY	

Commenti
Gli attributi sono:

	Attributo	Tipo	Vincolo
IdCommento	Int	PRIMARY KEY	
Testo	Varchar	null	
Rating	Int	null	
Username	Varchar	FOREIGN KEY	
NomeAPI	Varchar	FOREIGN KEY	

Key Questa tabella si crea dalla relazione tra Users e API. Rappresenta l'acquisto da parte di un utente di un microservizio. All'utente viene assegnata una chiave con la quale poter accedere al microservizio. Gli attributi sono:

	Atttributo	Tipo	Vincolo
Key	Varchar	PRIMARY KEY	
DataScadenza	Date	null	
MaxByte	Double	null	
TempoUso	Time	null	
Username	Varchar	FOREIGN KEY	
NomeAPI	Varchar	FOREIGNKEY	

In perticolare, la **DataScadenza** indica la data fino alla quale e' stato acquistato il microservizio; similmente **MaxByte** e **TempoUso** stanno ad indicare rispettivamente il traffico massimo ed il tempo massimo con i quali si puo' usufruire del microservizio in base al contratto d'acquisto stipulato.

Acquisti

Gli attributi sono:

	Attributo	Tipo	Vincolo
IdAcquisto	Int	PRIMARY KEY	
Username	Varchar	FOREIGN KEY	
Key	Varchar	FOREIGN KEY	

7 Design Pattern

I Design Pattern sono soluzioni generali a problemi ricorrenti. Il loro utilizzo semplifica l'attivita' di progettazione, favorisce il riutilizzo del codice e rende l'architettura piu' manutenibile. Esistono diversi tipi di Design pattern e sono suddivisi in base al problema da risolvere:

- **Pattern Architetturali**: definiscono l'architettura dell'applizacione ad un livello piu' elevato rispetto ad altri Design Pattern. Esprimono schemi di base per impostare l'organizzazione di un sistema software. Si descrivono sottoinsiemi predefiniti, i ruoli che assumono e le relaioni reciproche;
- **Pattern Creazionali**: permettono di nascondere i costruttori delle classi e mettono dei metodi al loro posto creando un'interfaccia. In questo modo si possono utilizzare oggetti senza sapere come sono implementati;
- **Pattern Strutturali**: consentono di riutilizzare degli oggetti esistenti fornendo agli utilizzatori un'interfaccia più adatta alle loro esigenze;
- **Pattern Comportamentali**: definiscono soluzioni per le piu' comuni interazioni tra oggetti.

7.1 Design Pattern Architetturali

7.1.1 Microservizi

- **Scopo dell'utilizzo**: Questo pattern è orientato ai microservizi, cioè permette di implementare unità separate distribuite;
- **Contesto d'utilizzo**: In tutta la nostra parte di back-end$_g$ utilizzeremo questo modello a microservizi, cercando quindi di separare le unità permettendo di conoscere solamente come comunicano tra di loro e non invece come sono implementate.

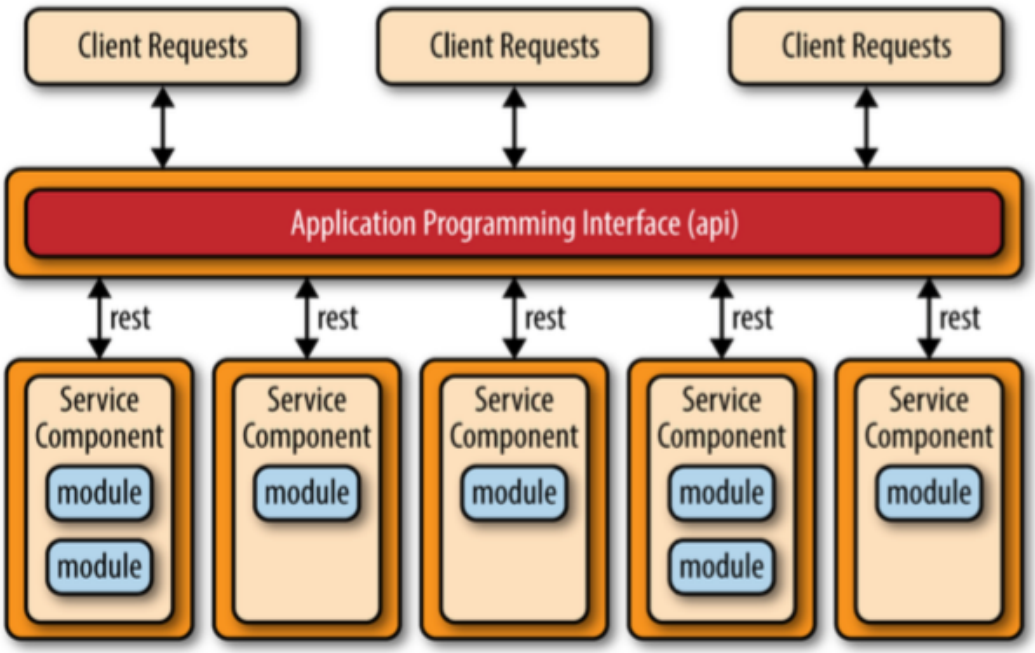
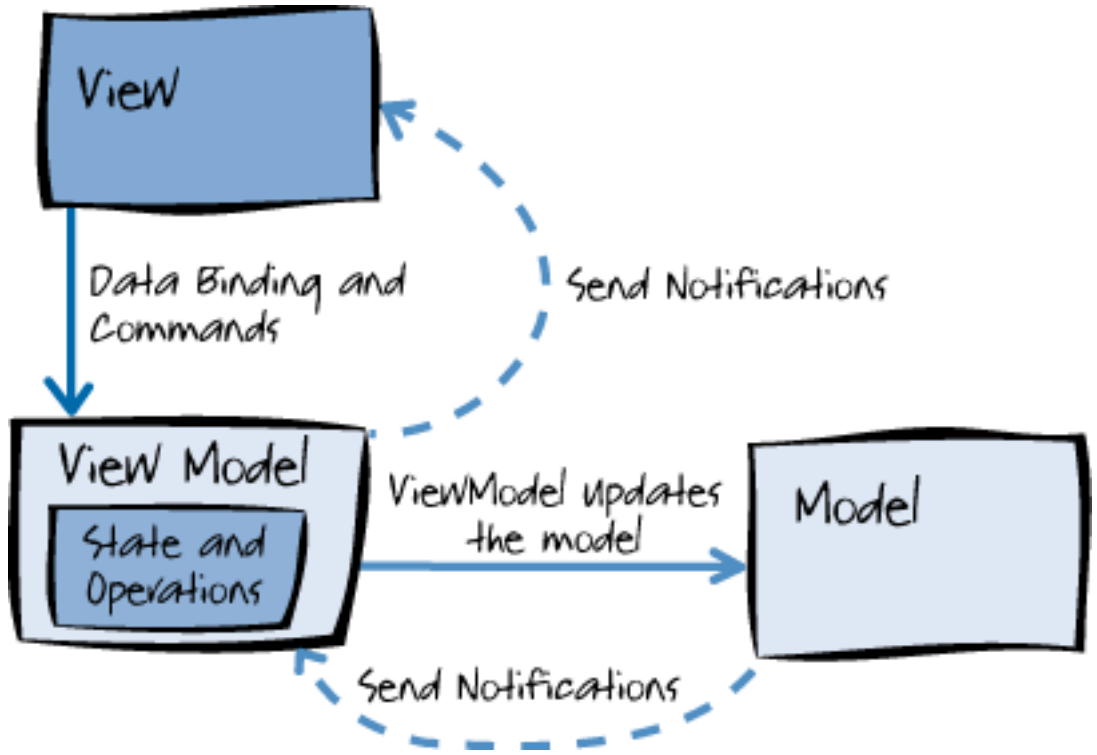


Figura 17: Architettura Microservizi

7.1.2 Model View View Model - MVVM

- **Scopo dell'utilizzo**: Permette di separare lo sviluppo della view dal comportamento;
- **Contesto d'utilizzo**: Verrà usato questo pattern per lo sviluppo del front-end$_g$. Abbiamo scelto il framework$_g$ AngularJS$_g$, il quale si presta bene all'adozione della famiglia di pattern MV. L'utilizzo di questi strumenti ci permette di separare la logica di business del front-end$_g$ dalla sua rappresentazione grafica.



0.5

Figura 18: mvvm

7.1.3 Data Access Object - DAO

- **Scopo dell'utilizzo**: Fornisce un'interfaccia per l'interazione con uno specifico tipo di database o in generale qualche sistema di persistenza. I vantaggi che ne derivano sono un totale disaccoppiamento tra la logica di business ed il database;
- **Contesto dell'utilizzo**: Questo pattern è stato scelto per la sua facilità di cooperare con Java ed i database relazionali. Crea una classe per ogni entità presente nel nostro sistema. Ognuna di questa classi conterrà poi dei metodi che le permetteranno di interagire con il database, dandole la possibilità, ad esempio, di inserire, modificare, rimuovere, aggiornare e ricercare dei dati.

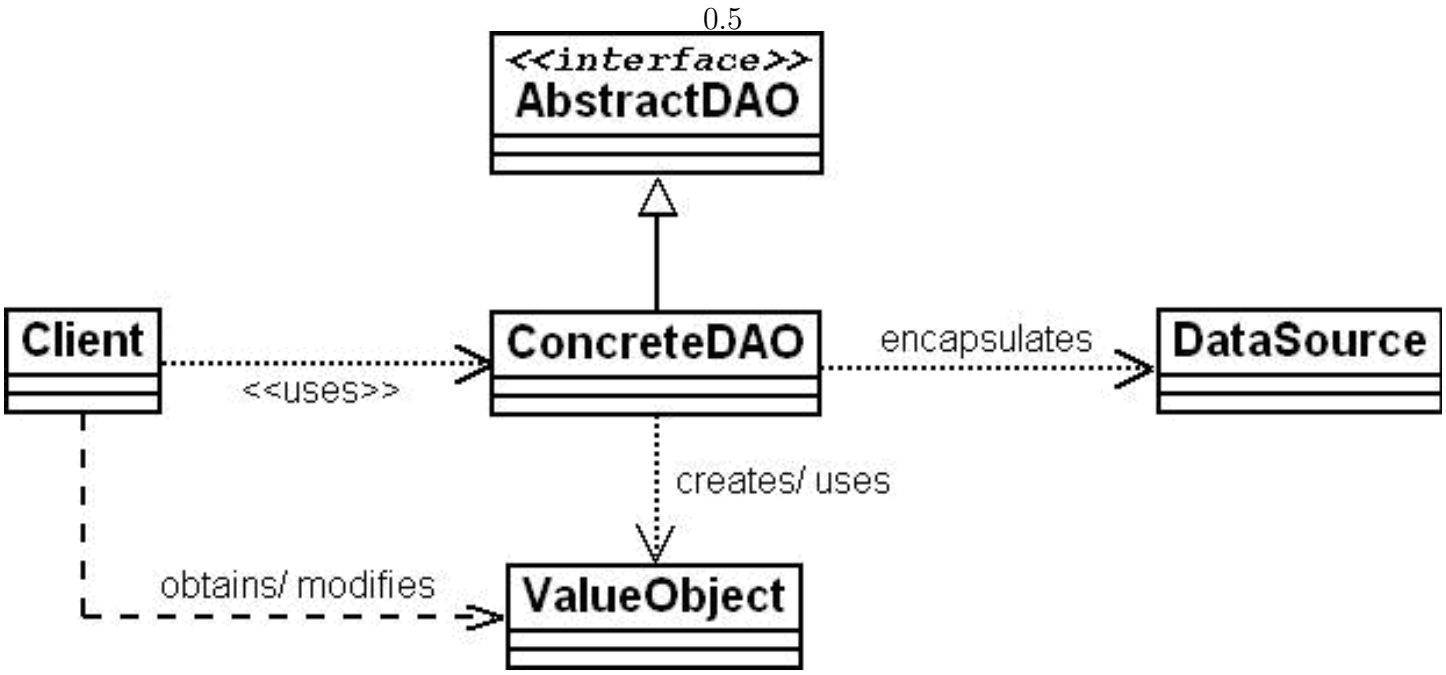


Figura 19: DAO

7.2 Design Pattern Creazionali

7.2.1 Abstract Factory

- **Scopo dell'utilizzo**: Fornisce un'interfaccia per creare famiglie di oggetti connessi o dipendenti fra loro, in modo che non ci sia necessità da parte del client di specificare i nome delle classi concrete all'interno del proprio codice;
- **Contesto dell'utilizzo**: Questo pattern è stato implementato per permettere di istanziare i data access object adeguati rispetto al database voluto, permettendo quindi una facile espansione futura.

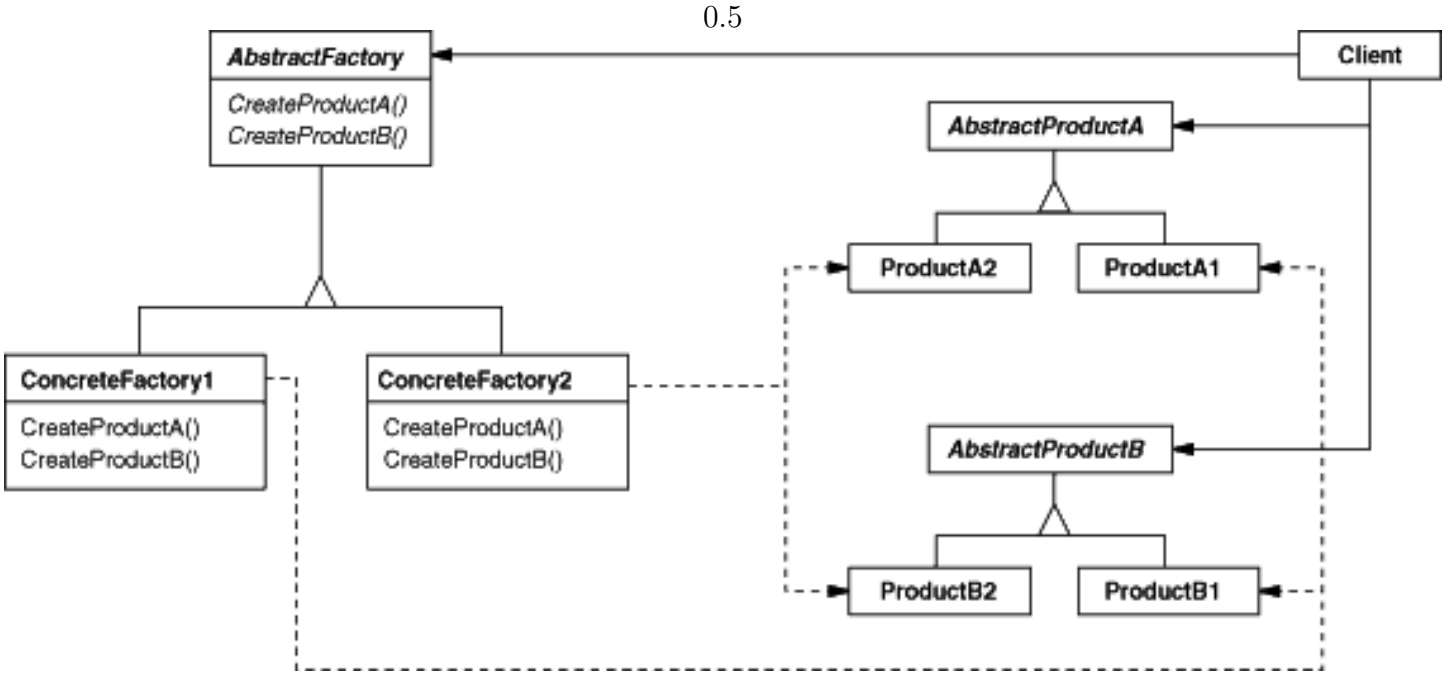


Figura 20: Abstract Factory

7.2.2 Dependency Injection

- **Scopo dell'utilizzo**: Permette di separare il comportamento di una componente dalla risoluzione delle sue dipendenze;
- **Contesto dell'utilizzo**: Ci permette di interagire facilmente con il framework$_g$ AngularJS$_g$. Verrà utilizzato per sviluppare la parte di front-end$_g$. Consente semplicemente ad un componente di AngularJS$_g$, che ha bisogno delle funzionalità messe a disposizione da un servizio, di specificare il suo nome tra i parametri della sua definizione.

0.5

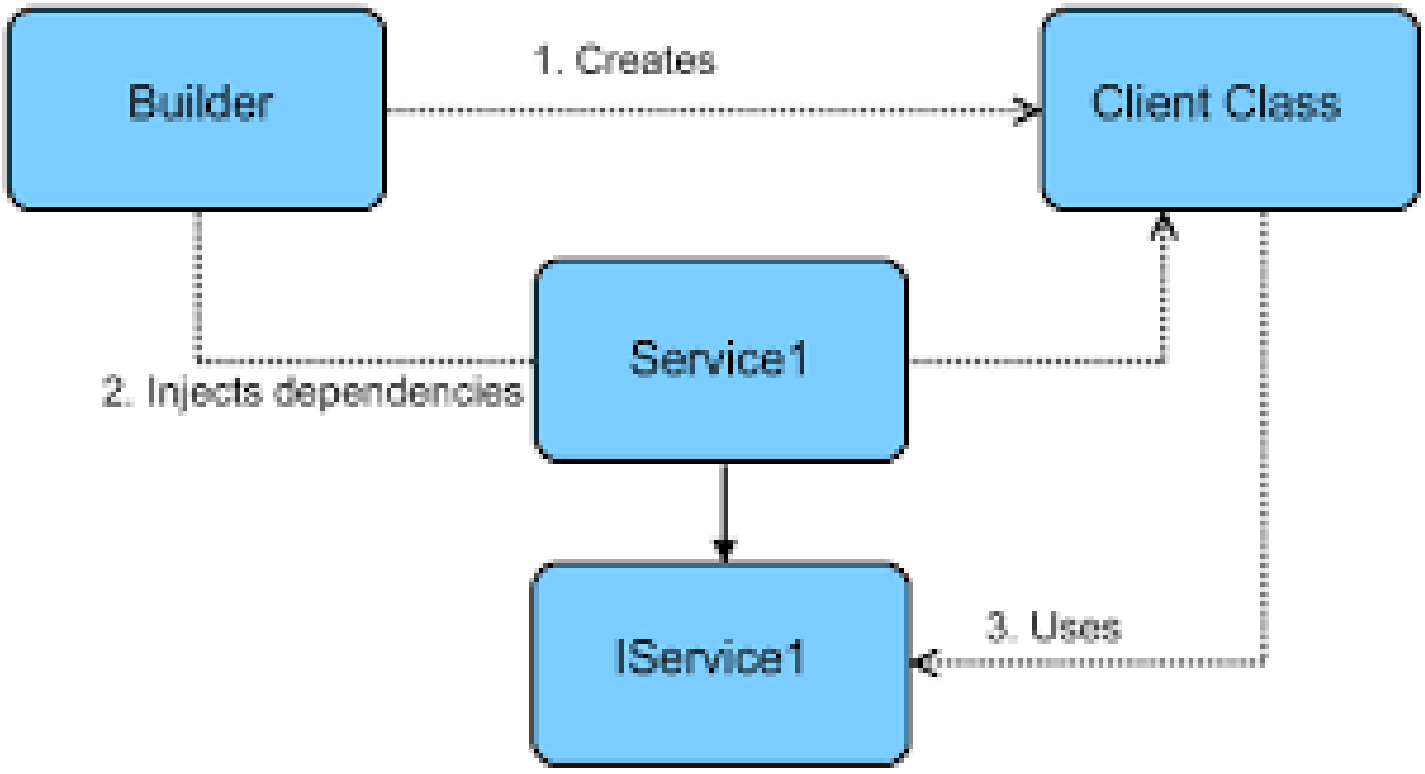
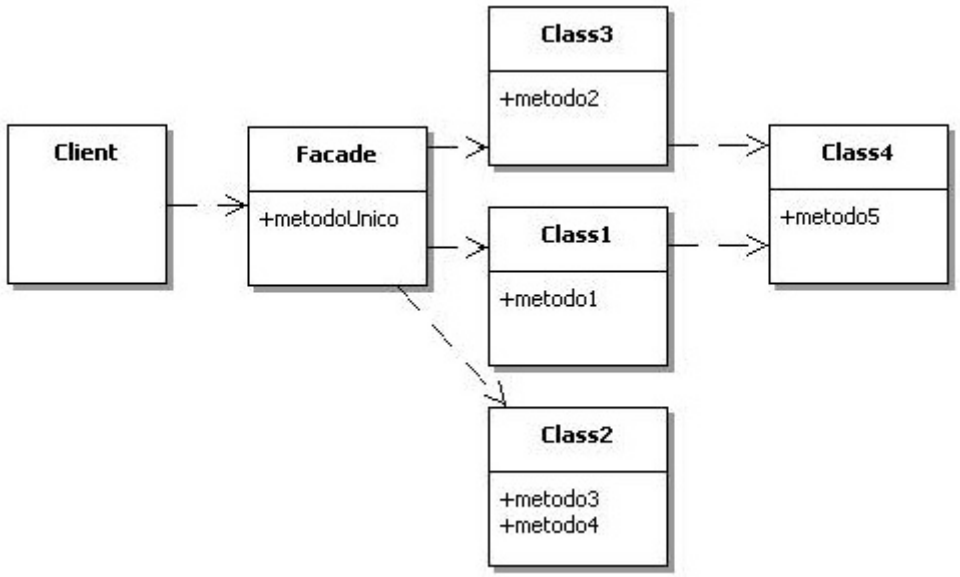


Figura 21: Dependency Injection

7.3 Design Pattern Strutturali

7.3.1 Facade

- **Scopo dell'utilizzo**: Fornisce un'interfaccia unica semplice per un sottosistema complesso, strutturando il sistema in sottoinsiemi;
- **Contesto dell'utilizzo**: Questo pattern è stato scelto perché è in grado di interfacciarsi con un sistema complesso, come può essere il nostro, e facilita le interazioni tra ogni livello del software.



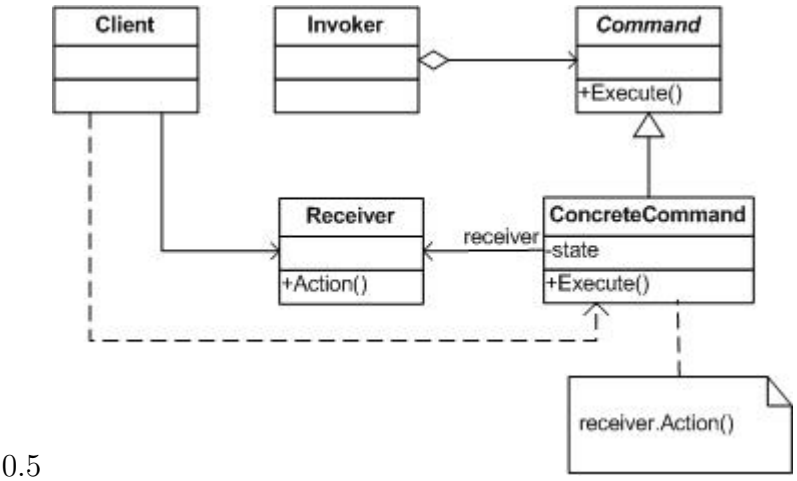
0.5

Figura 22: Facade

7.4 Desgin Pattern Comportamentali

7.4.1 Command

- **Scopo dell'utilizzo**: Permette di incapsulare una richiesta in un oggetto, cosicche' i client sia indipendente dalle richieste;
- **Contesto dell'utilizzo**: Questo pattern ci permette di facilitare il passaggio delle chiavi all'utente.



0.5

Figura 23: Command

8 Diagrammi di Attività Algoritmici

In questa sezione sono presenti i diagrammi di attività di APIMarket. Essi rappresentano il flusso logico dei due unici e principali algoritmi forniti e codificati nella parte di Back End.
Sono entrambi scritti in Jolie_g con parti embeddate_g in Java_g. Risolvono entrambi aggiornamento e finalità di caricamento dei microservizi. Ogni diagramma è accompagnato da una breve descrizione

8.1 Algoritmo di Analisi

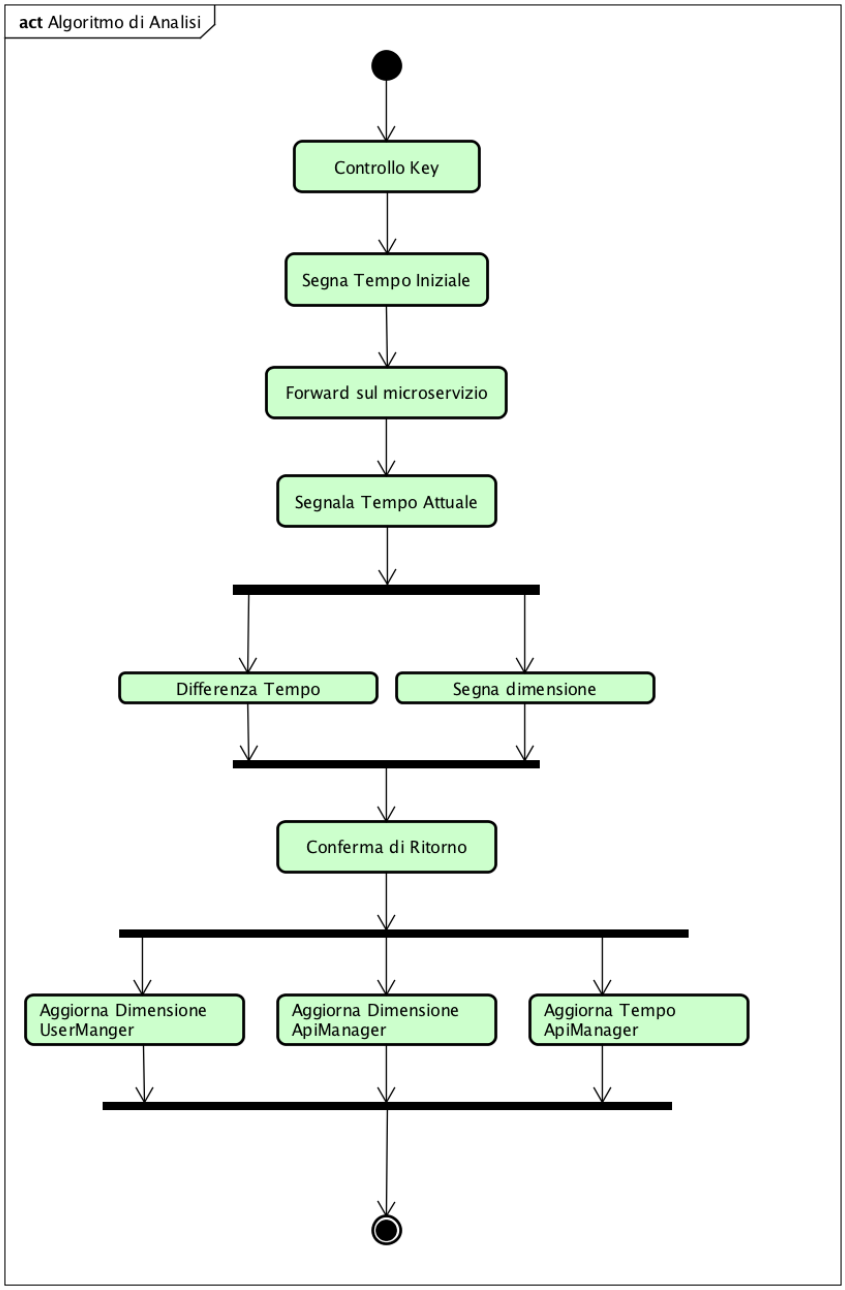


Figura 24: Diagrammi di attività - Attività principali

Algoritmo di Analisi si frappone tra il Gateway e il microservizio preso in esame. Dopo un controllo della key univoca tra utente e microservizio viene effettuato un controllo su base temporale (velocità di connessione) e su base dimensionale (dimensione della somma dei pacchetti inviati/ricevuti). A questo

sussegue l'inserimento dei dati nelle sezione interessanti. L'algoritmo viene usato in qualsiasi microservizio attivo nel database, anche più volte sullo stesso a seconda del numero di utenti che hanno acquistato la chiave.

8.2 Generatore di Courier

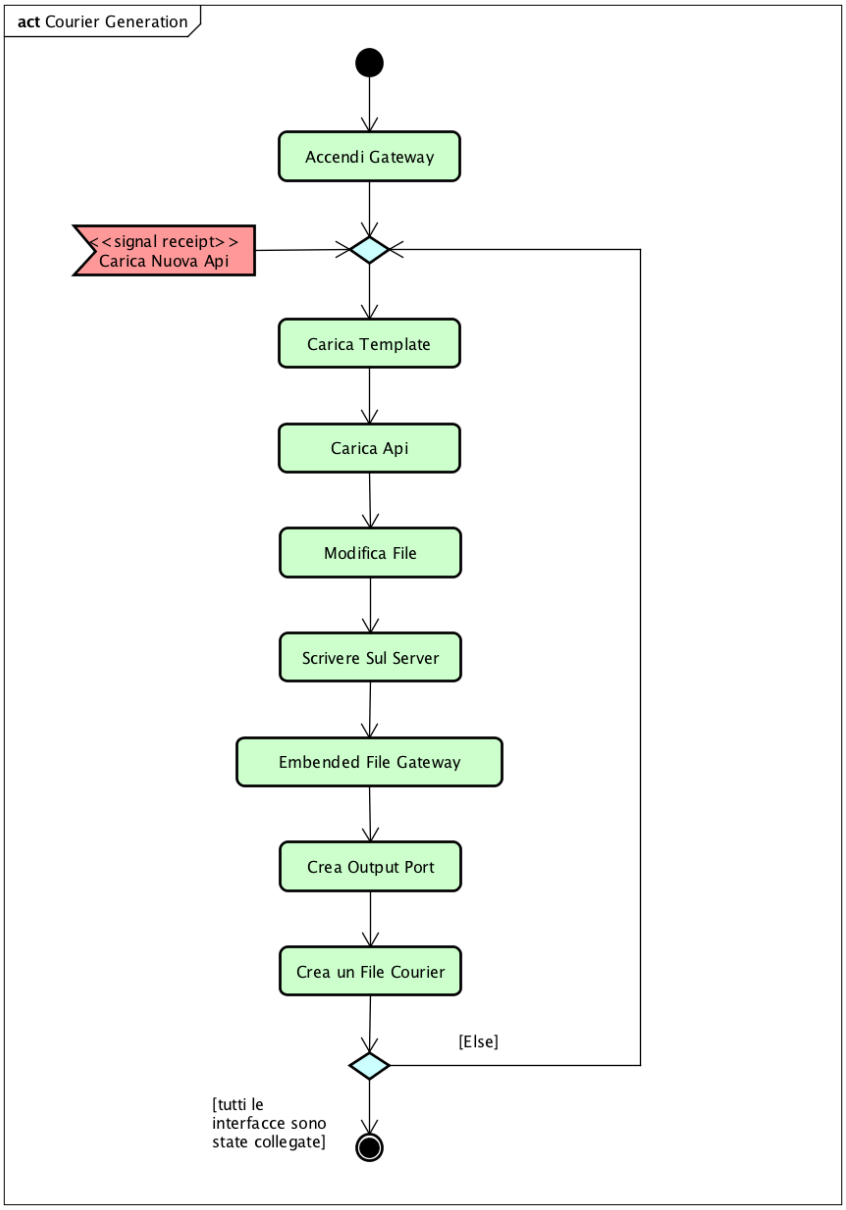


Figura 25: Diagrammi di attività - Autenticazione utente

La Courier è effettivamente un'interfaccia attivabile con un segnale che nel nostro caso verrà osservato in modo da tenere segnati dati sensibili da parte del Gateway. L'algoritmo di generazione di tale interfaccia oltre ad un controllo generale periodico che riprogramma e ricompone le courier di ogni singolo microservizio si attiva al segnale di inserimento di un nuovo microservizio. Quest'ultimo modifica il file avendo come matrice un template già costruito nel nostro server e crea e collega al nostro Gateway con una porta ad hoc.

9 Tracciamento

9.1 Tracciamento requisiti-componenti

Qui seguente il tracciamento dei requisti verso i componenti.

Requisito	Descrizione	Componenti
R0F1	APIMarket deve permettere la creazioni di profili utenti univoci	app::view::base app::view::menu app::view::home app::view::registration app::modelView::service app:modelView::component server::userManager server::daos
R0F2	APIMarket deve permettere l'autenticazione di un utente registrato.	app::view::menu app::view::home app::view::login-page app::modelView::component server.userManager server.daos
R1F2.4	APIMarket deve rendere disponibile una funzionalità per il recupero della password qualora l'utente dimentichi la sua.	app::view::login-page app::modelView::component server::userManager server::daos
R0F2.5	APIMarket deve permettere all'utente autenticato di effettuare il logout.	app::view::user-page server::userManager
R0F3	APIMarket deve permettere di registrare le API di un microservizio attraverso la registrazione e pubblicazione della sua interfaccia.	app::view::user-api app::view::uploadApi app::modelView::component app::modelView::service server::ApiManager server:::daos
R0F4	APIMarket deve fornire una pagina riservata all'utente registrato in cui amministrare il suo account, i suoi messaggi e le interfacce che ha pubblicato.	app::view::user-page app::modelView::service app::model app::modelView::component app::model server::userManager server::daos
R1F4.1	APIMarket deve permettere all'utente di modificare i dati del suo profilo.	app::view::user-page app::modelView::component server::daos server::userManager
R2F4.3.1	APIMarket deve fornire all'utente la possibilità di modificare i dati di un microservizio da lui pubblicato.	app::view::user-api server::daos
R0F4.3.2	APIMarket deve fornire all'utente un'area, per ogni microservizio da lui pubblicato, in cui monitorare le sue statistiche.	app::view::user-api app::modelView::component server::Analyzer server::daos
R1F4.4.1	APIMarket deve dare la possibilità all'utente di modificare la propria carta di credito durante la procedura di acquisto della APICredits, qualora decidesse di utilizzare un'altra carta di credito diversa da quella registrata.	app::view::user-page server::daos

R0F5	APIMarket deve permettere a qualsiasi utente, anche non registrato, di cercare dei microservizi sul sito	app::view::serch-page app::modelView::component app::modelView::service server::controller server::gateway server::daos server::ApiManager
R0F6	APIMarket deve permettere all'utente di visionare la pagina di un microservizio cercato. Se l'utente è autenticato allora deve avere delle informazioni più specifiche	app::view::apiHome app::view::viewApi app::modelView::component app::modelView::service app::model server::daos server::ApiManager
R0F6.6	APIMarket deve dare la possibilità agli utenti autenticati di acquistare una APIKey per il microservizio visitato, utilizzando i propri APICredits.	app::view::user-page app::modelView::component server::controller server::keyManager server::keyManager server::daos
R0F6.8	APIMarket deve permettere all'utente autenticato di confrontare le statistiche tra due microservizi	app::view::apiHome app::modelView::component server::Analyzer
R1F6.9	APIMarket deve permettere all'utente autenticato scrivere un commento e dare un voto al microservizio.	app::view::user-comments app::modelView::component app::modelView::service app::model server::daos server::commentManager
R0F7	APIMarket deve associare una APIKey d'uso per ogni API registrata e per ogni utente che decide di acquistarla	app::view::user-page app::modelView::component server::keyManager server::daos
R0F7.2	Se la APIKey è contraffatta o non valida allora la chiamata all'API viene bloccata dall'APIMaket.	server::keyManager server::daos
R1F8.1	APIMarket nella pagina utente deve rendere visibile alcuni dati personali, commenti scritti e interfacce di microservizi pubblicate.	app::view::user-page app::modelView::component server::daos
R2F8.2	APIMarket nella pagina utente deve mostrare un indice di affidabilità dell'utente calcolato sulle statistiche dei microservizi da lui pubblicati.	app::view::user-page server::daos
R2F9.2	Un utente Admin deve avere la possibilità di modificare i dati anagrafici di qualsiasi utente.	app::view::administration app::modelView::component server::userManager server::daos
R2F9.3	Un utente Admin deve poter eliminare un commento di qualsiasi utente.	app::view::administration app::modelView::component server::userManager server::daos
R1F9.4	Un utente Admin deve poter modificare i dati di una API la cui interfaccia è stata caricata su APIMarket.	app::view::administration app::modelView::component server::userManager server::daos

9.2 Tracciamento componenti-requisiti

Componente	Requisito
app	
app::view	
app::view::base	R0F1 R0F2
app::view::menu	R0F1 R0F2
app::view::home	R0F1 R0F2
app::apiHome	R0F6
app::view::user-api	R0F3
app::view::user-comments	R1F6.9
app::view::registration	R0F1
app::view::user-page	R0F4 R1F4.1
app::view::login-page	R0F2 R1F2.4
app::view::uploadApi	R0F3 R0F3.1
app::view::viewApi	R0F6
app::view::search-page	R0F5
app::view::issue	R1F4.2 R0F4.3
app::view::administration	R1F9 R2F9.3 R1F9.4
app::modelView	
app::modelView::service	R0F1 R0F3 R0F4 R1F4.2 R0F5 R0F6 RF6.9 R1F9
app::modelView::component	R0F1 R0F22 R1F2.4 R0F3 R0F4 R0F4.1 R1F4.2 R0F4.3 R0F5 R0F6 R1F6.9 R1F9 R2F9.3 R1F9.4
app::model	R0F4 R1F6.9 R0F6
server	
server::Gateway	R0F5
server::Analyzer	R0F4.3.2 R6F6.8
server::ApiManager	R0F3 R0F3.1 R0F5 R0F6

server::CommentManager	R1F6.9
server::KeyManager	R0F6.6 R0F7 R0F7.2
server::UserManager	R0F1 R0F2 R1F2.4 R0F2.5 R0F4 R1F4.1 1 R1F9 R2F9.2 R2F9.3 R1F9.4
server::daos	R0F1 R0F2 R1F2.4 R0F3 R0F4 R1F4.1 R2F4.3.1 R0F4.3.2 R1F4.4.1 R0F5 R0F6.6 R1F6.9 R0F7 R0F7.2 R1F8.1 R2F8.2 R2F9.2 R2F9.3 R1F9.4 R1F9.5

A Descrizione Design Pattern

A.1 Design Pattern Architetturali

A.1.1 Pattern Architetturale a Microservizi

Il pattern architetturale a microservizi è un pattern innovativo che va a sostituire la vecchia filosofia dei sistemi monolitici dedicato e alle architettura orientate ai servizi.

Figura 26: Illustrazione Architettura a Microservizi

- **Descrizione Generale** : il primo concetto che caratterizza il pattern è l'idea di unità separate distribuite. Questo aumenta la scalabilità e un alto grado di disaccoppiamento all'interno della nostra applicazione. Forse il fattore più importante è pensare al microservizio non come componente dell'architettura ma come servizio in se, che può variare la propria granularità da un singolo modulo a gran parte dell'applicazione.
  Un altro concetto chiave all'interno del modello microservices architettura è che si tratta di un'architettura distribuita, il che significa che tutti i componenti all'interno dell'architettura sono completamente disaccoppiati da un altro e sono accessibili attraverso una sorta di protocollo di accesso remoto.
  L'architettura stessa si è evoluta da problemi riscontrati in altri modelli e non è in attesa che un problema si verifichi. In particolare si è evoluta da due principali pattern architetturali: le applicazioni monolitiche sviluppate utilizzando il modello di architettura a strati e applicazioni distribuite sviluppate attraverso l'architettura modello orientato ai servizi.
  In generale, nell'architettura a microservizi, ogni singolo servizio è autonomo rispetto agli altri, di conseguenza può raggiungere l'ambiente di produzione in modo indipendente dagli altri, testato e distribuito, senza che tale attività abbia effetti drammatici sul resto del sistema. Disporre di un processo di deployment snello e veloce consente di poter aggiungere o modificare funzionalità di un sistema software in modo efficace ed efficiente, rispondendo alle necessità di mercato e utenti sempre più esigenti.
  L'altro percorso evolutivo che ha portato al modello architetturale a microservizi proviene da problemi rilevati con le applicazioni di attuazione dell'architettura modello orientato ai servizi (SOA_G). Mentre il modello SOA è molto potente e offre livelli senza precedenti di astrazione, connettività eterogenea, orchestrazione dei servizi, e la promessa di allineare gli obiettivi di business con funzionalità IT, è comunque complesso, costoso, onnipresente, difficile da capire e mettere in atto, e di solito è eccessivo per la maggior parte delle applicazioni.

- **Considerazioni** : Robustezza, miglior scalabilità, erogazione continua. Con piccole componenti miglioriamo la distribuzione e questo risolve i problemi delle applicazioni monolitiche e SOA. Abbiamo una disponibilità di servizio continua.

Caratteristica	Valutazione	Descrizione
Agilità	+	Cambiamenti isolati , veloci e di facile sviluppo
Implementazione	+	Di natura singolare e univoca quindi facili da implementare
Testabilità	+	Visto l'isolamento delle funzioni business il test può essere più specifico. Piccola possibilità di regressione
Performance	-	Essendo per la maggior parte nella rete è difficile mantenere delle prestazioni massime e costanti
Scalabilità	+	Ogni componente può essere separato e quindi vi è la massima scalabilità
Sviluppo	+	Piccoli e isolati componenti facilitano lo sviluppo

Tabella 4: Analisi Microservizi

Sorgono i problemi della distribuzione e della disponibilità dei sistemi remoti. Non si può utilizzare nel caso abbiamo bisogno di un orchestratore, a meno che non rimanga all'interno di un microservizio, e nemmeno nel caso abbiamo bisogno di transazionalità.

- **Analisi del Pattern** :

A.1.2 Data Access Object (DAO)

Il DAO (Data Access Object) è un pattern architetturale per la gestione della persistenza: si tratta fondamentalmente di una classe con relativi metodi che rappresenta un'entità tabellare di un database relazionale.

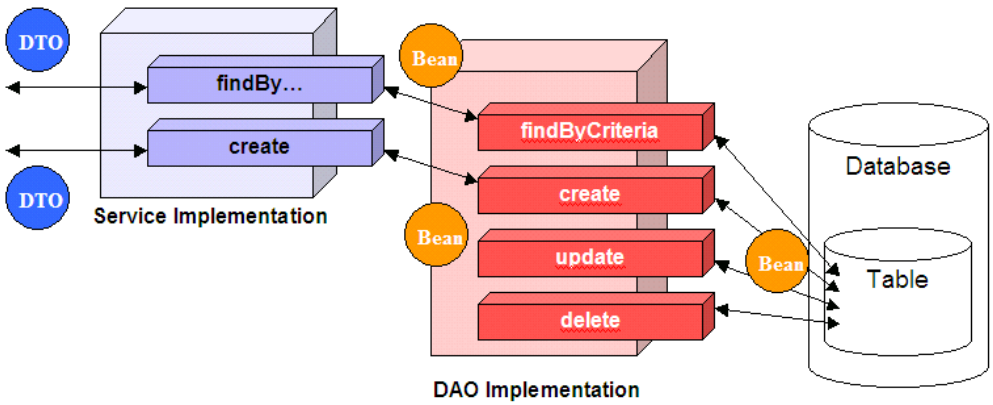


Figura 27: Illustrazione Architettura Data Access Object

- **Descrizione Generale** : Usata principalmente in applicazioni web (come ad esempio Java EE), per stratificare e isolare l'accesso ad una tabella tramite query (poste all'interno dei metodi della classe) ovvero al data layer da parte della business logic creando un maggiore livello di astrazione ed una più facile manutenibilità. I metodi del DAO con le rispettive query dentro verranno così richiamati dalle classi della business logic.
  Anche se questo modello di progettazione è ugualmente applicabile alla maggior parte dei linguaggi di programmazione, dei software tipati che hanno bisogno di persistenza e la maggior parte delle tipologie di basi di dati, ha il vantaggio di cooperare perfettamente con Java e i data base relazionali.

Caratteristica	Valutazione	Descrizione
Agilità	-/+	Semplice da gestire ma l'obbligo di innescare query multiple lo rende meno snello.
Implementazione	+	Facile l'implementazione perché è relativamente semplice e rigoroso separare due parti dell'applicazione.
Testabilità	+	Unit Test il codice è facilitato sostituendo il DAO con un Double Test nel collaudo, rendendo così i test non dipendi dal Data-Base persistente.
Performance	-	Vi è un costo aggiuntivo ad ogni chiamata al server e aumento di complessità.
Scalabilità	+	Eventuali modifiche al DB possono essere implementate da sole , senza che il resto dell'applicazione è interessata.
Sviluppo	-	Obbliga gli sviluppatori a innescare query multiple sul database che potrebbero essere recuperate con una unica.

Tabella 5: Analisi DAO

- **Considerazioni** : Le Data Access Objects si fanno carico di gestire il codice SQL, mentre tutto ciò é trasparente rispetto alle corrispondenti classi di dominio e di controllo.
  A livello di logica dell'applicazione siamo fortemente orientati agli oggetti: ragioniamo solo in termini di Domain Objects, cioè dei concetti pertinenti al dominio dell'applicazione, e non possiamo mai utilizzare i metodi di accesso diretto alla base dati forniti dai DAO.
  Tutti i dettagli dell'archiviazione sono nascosti mantenendo nascoste le informazioni, questo dà pregio al pattern per i nostri scopi.
- **Analisi del Pattern** :

A.1.3 Model View ViewModel

Il Model–view–viewmodel (MVVM) è un pattern software architetturale o schema di progettazione software. È una variante del pattern "Presentation Model design" di Martin Fowler.

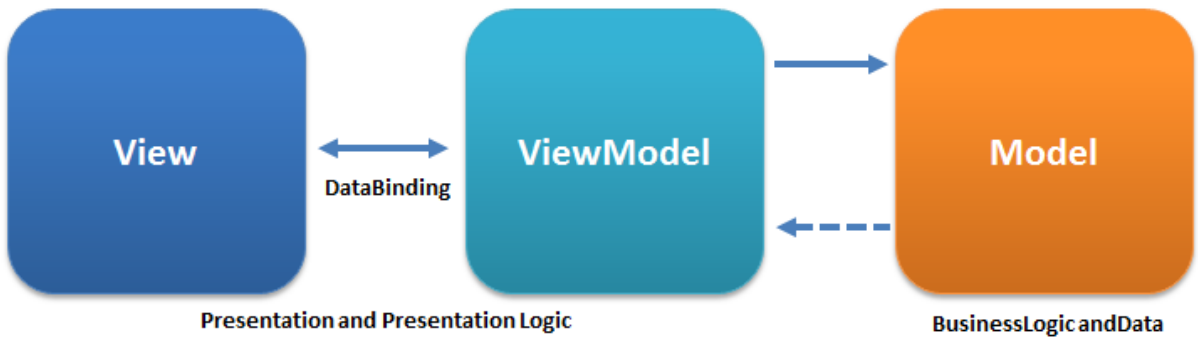


Figura 28: Illustrazione Model View ViewModel

Caratteristica	Valutazione	Descrizione
Agilità	+	È il classico pattern a 3 elementi ed è ideologia di agilità
Implementazione	+	Avendo netta distinzione tra gli elementi è facile l'implementazione e mantenerlo.
Testabilità	+	Lo Unit Testing nel MVVM dove le componenti siano "separate" contribuisce alla progettazione di unità di test efficaci.
Performance	+	Rispetto al classico modello MVC è più snello e quindi a confronto è più veloce.
Scalabilità	+	Possono essere modificati implementati differentemente essendo separati.
Sviluppo	+	Ogni singolo elemento del team può concentrarsi allo sviluppo di ogni diverso elemento.

Tabella 6: Analisi Model View ViewModel

- **Descrizione Generale** : Lo MVVM astrae lo stato di "view" (visualizzazione) e il comportamento. Sebbene, dove il modello di "presentazione" astrae una vista (crea un view model) in una maniera che non dipende da una specifica piattaforma interfaccia utente. In pratica separa lo sviluppo dell'interfaccia utente dalla business logic$_g$.
  Le componenti sono le seguenti:
  - Il **Model** rappresenta il punto di accesso ai dati. Trattasi di una o più classi che leggono dati dal DB, oppure da un servizio Web di qualsivoglia natura.
  - La **View** rappresenta la vista dell'applicazione, l'interfaccia grafica che mostrerà i dati.
  - Il **ViewModel** è il punto di incontro tra la View e il Model: i dati ricevuti da quest'ultimo sono elaborati per essere presentati e passati alla View.

  Il fulcro del funzionamento di questo pattern è la creazione di un componente, il ViewModel appunto, che rappresenta tutte le informazioni e i comportamenti della corrispondente View. La View si limita infatti, a visualizzare graficamente quanto esposto dal ViewModel, a riflettere in esso i suoi cambi di stato oppure ad attivarne dei comportamenti.
- **Considerazioni** : L'utente interagisce solo ed unicamente con li View. E la comunicazione di stato è comunicata al ViewModel. Come risposta al cambio di stato o all'attivazione di un metodo il ViewModel invia un segnale o esegue un operazione sul Model e aggiorna il proprio stato. Il nuovo stato del ViewModel si riflette sulla View.
  È da sottolineare il fatto che il ViewModel mantiene nel proprio stato non solo le informazioni recuperate attraverso il Model, ma anche lo stato attuale della visualizzazione: ciò gli consente di essere del tutto disaccoppiato dalla View. Inoltre il processo step-by-step descritto in precedenza risulta essere un "two-way", funziona cioè in entrambe le direzioni.
  AngularJS implementa un modello basato su questa visione del pattern e utilizza HTML come linguaggio di templating, non richiede operazioni di DOM refresh e controlla attivamente le azioni utente ed eventi nel browser.
- **Analisi del Pattern** :

A.2 Design Pattern Creazionali

A.2.1 Abstract Factory

L'Abstract Factory fornisce un'interfaccia per creare famiglie di oggetti connessi o dipendenti tra loro, in modo che non ci sia necessità da parte dei client di specificare i nomi delle classi concrete all'interno del proprio codice. In questo modo si permette che un sistema sia indipendente dall'implementazione degli oggetti concreti e che il client, attraverso l'interfaccia, utilizzi diverse famiglie di prodotti.

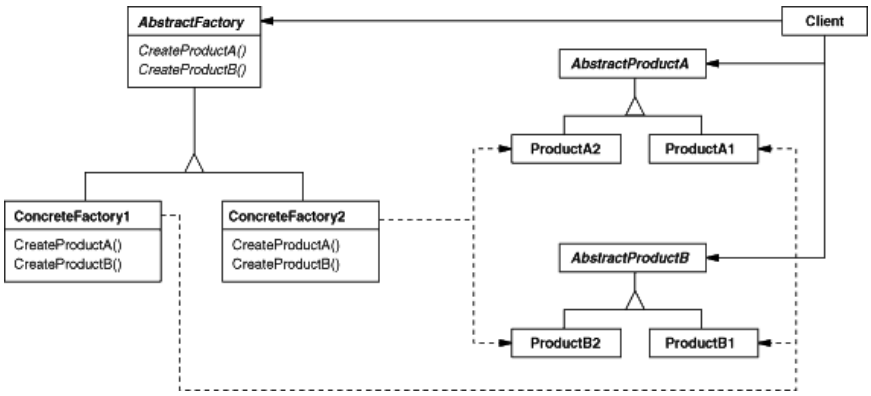


Figura 29: Illustrazione Pattern Abstract Factory

- **Descrizione Generale** : Il pattern Abstract Factory definisce un'interfaccia con una serie di metodi per creare una famiglia di prodotti correlati. La famiglia di oggetti correlati è definita attraverso una serie di tipi di elementi. L'attuazione dei tipi di prodotto è delegata a un insieme di sottoclassi di elementi concreti. La creazione delle classi di prodotto concrete è attuato per serie di classi factory concrete. Il pattern Abstract Factory rinvia la creazione degli elementi concreti alle classi "di fabbrica" concrete che implementa l'Abstract Factory. L'oggetto client è disaccoppiato dalle classi di prodotti e le classi di fabbrica attraverso l'interfaccia Abstract Factory. Il nucleo del pattern Abstract Factory è quello di creare un gruppo di oggetti correlati che potrebbero avere diverse implementazioni. Il sistema è quindi indipendente dell'attuazione dei tipi di prodotto. Il pattern Abstract Factory permette anche il sistema per sostituire un insieme di classi di prodotti correlati con un altro set, cambiando la classe fabbrica.
- **Considerazioni** : Questo pattern è utile quando si vuole un sistema indipendente da come gli oggetti vengono creati, composti e rappresentati. Si vuole permettere la configurazione del sistema come scelta tra diverse famiglie di prodotti. I prodotti che sono organizzati in famiglie siano vincolati ad essere utilizzati con prodotti della stessa famiglia. Si vuole infine fornire una libreria di classi mostrando solo le interfacce e nascondendo le implementazioni.
- **Analisi del Pattern** :

A.2.2 Dependency Injection

Dependency Injection (DI) è un design pattern della programmazione orientata agli oggetti il cui scopo è quello di semplificare lo sviluppo e migliorare la testabilità di software di grandi dimensioni.

Caratteristica	Valutazione	Descrizione
Agilità	+	È nella definizione del pattern creazionale la possibilità di creare oggetti a partire da qualsiasi classe.
Implementazione	+	Singola istanza della factory. Semplicità di aggiungere una nuova famiglia. Difficile però aggiungere un'interfaccia.
Testabilità	+/-	Isolamento di tipo concreto e quindi facilmente verificabile. Nella modifica c'è bisogno di una riverifica generale.
Performance	n.q.	Dipende tutto dall'implementazione della classe su cui si vuole fare la factory.
Scalabilità	-	Aggiungere nuove famiglie di prodotti è difficile perché l'insieme di prodotti gestiti è legato all'interfaccia della factory.
Sviluppo	+	L'interfaccia di per se è di facile implementazione e compare in un unico punto del codice.

Tabella 7: Analisi Abstract Factory

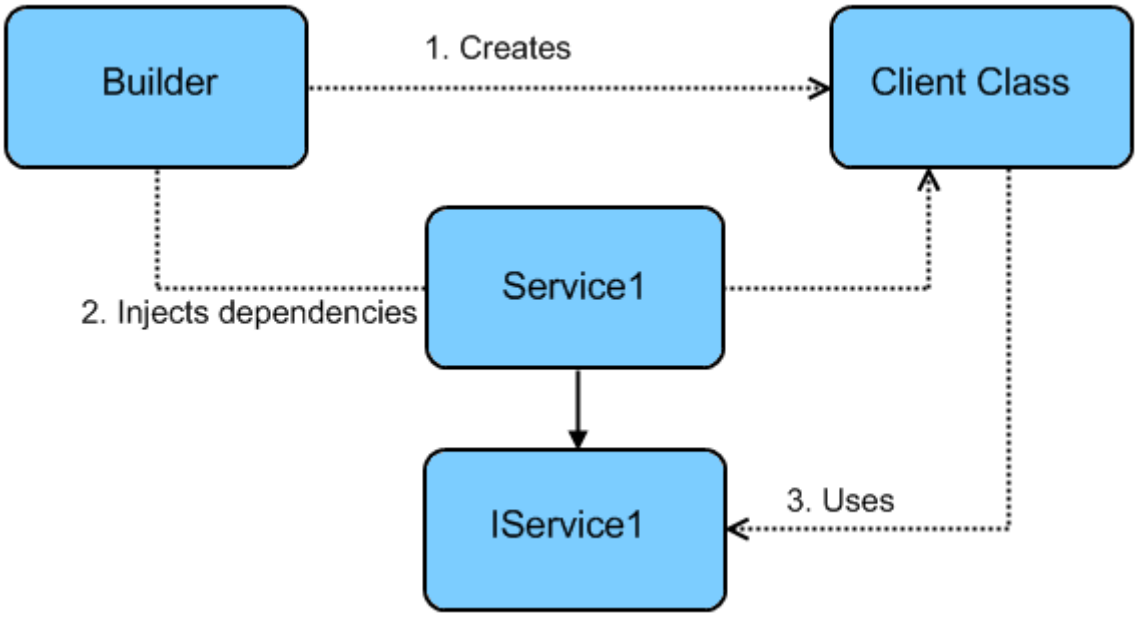


Figura 30: Illustrazione Pattern Dependency Injection

- **Descrizione Generale** : Per utilizzare tale design pattern è sufficiente dichiarare le dipendenze che un componente necessita (dette anche interface contracts). Quando il componente verrà istanziato, un iniettore si prenderà carico di risolvere le dipendenze (attuando dunque l'inversione del controllo). Se è la prima volta che si tenta di risolvere una dipendenza l'injector istanzierà il componente dipendente, lo salverà in un contenitore di istanze e lo ritornerà. Se non è la prima volta, allora ritornerà la copia salvata nel contenitore. Una volta risolte tutte le dipendenze, il controllo può tornare al componente applicativo.

Caratteristica	Valutazione	Descrizione
Agilità	-/+	Non è molto flessibile, ma si può migliorare sostanzialmente con un interfaccia. Inoltre disaccoppia facilmente le classi.
Implementazione	-/+	Attenzione che creare istanze di classe può diventare ingombrante, soprattutto quando le classi crescono di responsabilità e cominciano ad avere troppe dipendenze.
Testabilità	+	La DI associata ad AngularJS promuove la testabilità del framework e del pattern.
Performance	+	Si occupa il pattern insieme al framework AngularJS delle "iniezioni" di dipendenze senza che badiamo noi alle performance, dobbiamo scegliere solo l'implementazione migliore.
Scalabilità	+	Migliorando la modularità del codice è più facile un approccio scalabile al pattern.
Sviluppo	-	Sviluppo delle DI associato ad AngularJS è complesso e per niente scontato.

Tabella 8: Analisi Dependency Injection

- **Considerazioni** : È stata scelta la Dependency Injection perchè ci sono molti riferimenti e esempi con AngularJS. Con il linguaggio è legata anche la pratica della minificazione$_g$ del codice, processo per ridurre la dimensione del codice. Se infatti non si adottasse questo accorgimento, durante il processo che riduce la lunghezza del nome delle variabili si verrebbero a perdere i riferimenti ai nomi dei componenti che rappresentano le dipendenze.
  La dependency injection è un design pattern che delega ad un'entità esterna il compito di individuare e fornire una risorsa di cui un oggetto ha bisogno. Nel nostro caso, se un componente Angular, come ad esempio un controller, ha bisogno delle funzionalità di messe a disposizione da un servizio non deve fare altro che specificare il suo nome tra i parametri della sua definizione.
- **Analisi del Pattern** :

A.3 Design Pattern Strutturali

A.3.1 Facade

Letteralmente facade significa "facciata", ed infatti nella programmazione ad oggetti indica un oggetto che permette, attraverso un'interfaccia più semplice, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro, nonché a blocchi di codice complessi.

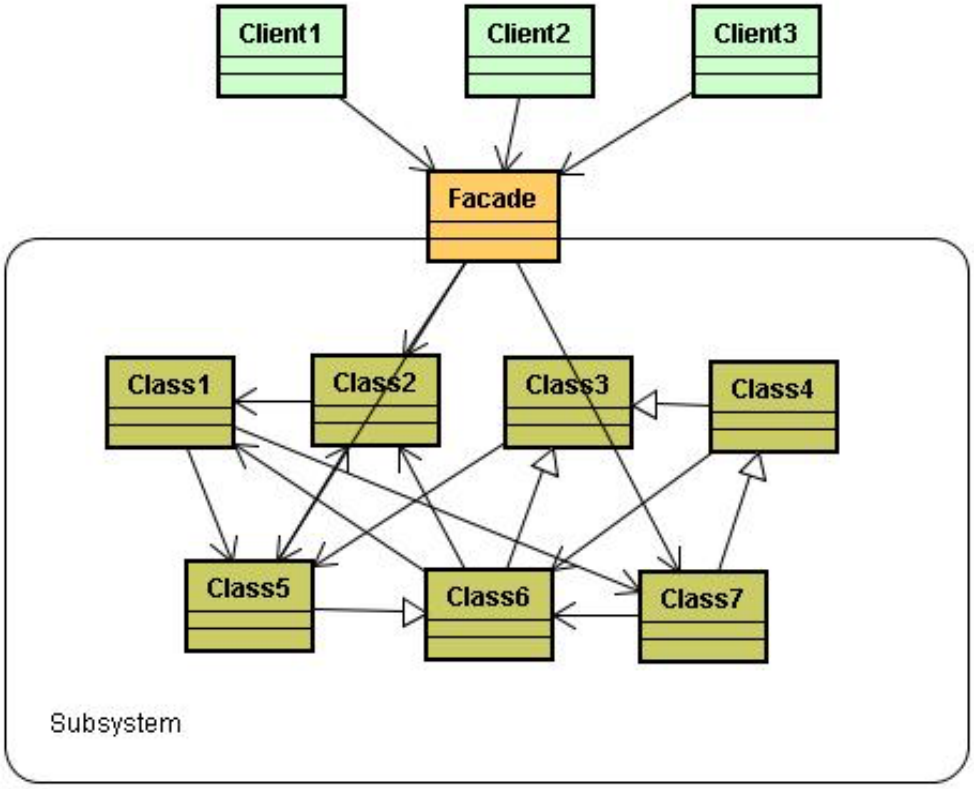


Figura 31: Illustrazione Pattern Facade

- **Descrizione Generale** : Fornisce un interfaccia unica e semplice per un sottosistema complesso. Questo porta a una strutturazione ordinata di un sistema di sottoinsiemi. Questo diminuisce la complessità del sistema avendo una figura intermedia anche se aumenta la dipendenza dei sottoinsiemi; semplificando le dipendenze però non mantiene il principio di information-hiding mostrando funzionalità che sarebbero nascoste dai sottogruppi.
- **Considerazioni** : Facade si usa quando si vuole semplicemente comunicare con un sottosistema di elementi. Si usa oltre per interfacciarsi a un sistema difficile, quando il sistema sottostante è veramente complesso e difficile da comprendere. Facade fa anche da punto di comunicazione per ogni livello dello strato software oppure perché le astrazioni e le implementazioni di un sottosistema sono strettamente accoppiati.
  Vi è quindi un disaccoppiamento tra sottosistemi e client creando così una divisione a noi tanto cara come l'idea a microservizi, nascondendo così i livelli tra l'astrazione e l'implementazione. Così si crea una stratificazione di sistema.
- **Analisi del Pattern** :

A.4 Design Pattern Comportamentali

A.4.1 Command

Nella programmazione ad oggetti, il Command pattern è uno dei pattern fondamentali, definiti originariamente dalla "gang of four"$_g$. Fa parte della tipologia dei pattern comportamentali cioè rispondono alle domande:

- In che modo un oggetto svolge la sua funzione?

Caratteristica	Valutazione	Descrizione
Agilità	-	Non riduce le dipendenze ma le concentra con un unico elemento Facade.
Implementazione	+	Semplifica i sottosistemi con un interfaccia unica. Quindi aiuta a ridurre la complessità.
Testabilità	+/-	La filosofia facade ispira un test automatizzato specifico denominato facade-test che è più di un test di unità, ma non abbastanza come un test di integrazione.
Performance	-	Sistema centrato. Se manca il facade il sistema crolla rende instabile e non performante il pattern.
Scalabilità	+	Aggiungere nuove famiglie di prodotti è semplice, basta collegarlo all'interfaccia facade senza sforzo.
Sviluppo	n.q.	Interfaccia di facile implementazione; dipende dalla complessità del sottosistema lo sviluppo.

Tabella 9: Analisi Facade

- In che modo diversi oggetti collaborano tra loro?

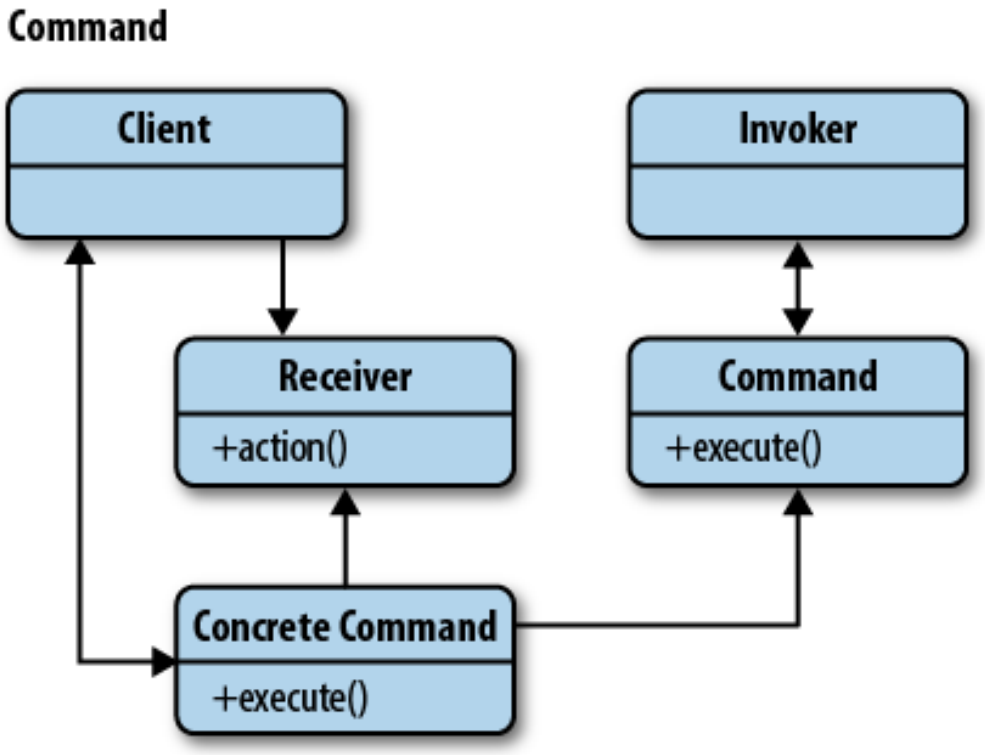


Figura 32: Illustrazione Pattern Command

- **Descrizione Generale** : Incapsula una richiesta (callback$_g$) in un oggetto, così il client sia indipendente dalle richieste. Questo per gestire richieste (toolkit) di cui non si conoscono i particolari ed

Caratteristica	Valutazione	Descrizione
Agilità	+	Con chiamate asincrone è possibile eseguire più richieste agilmente anche se con un passaggio in più. E il client manda solamente la richiesta.
Implementazione	+	Buona l'implementazione che unisce varie sottoclassi di creazione in una unica.
Testabilità	-	Pecca del pattern è che deve essere implementato prima del test e difficile sostituzione.
Performance	+	Senza la necessità per il cliente di essere a conoscenza dell'esistenza di funzioni particolari e in maniera asincrona il command lavora ad alte prestazioni.
Scalabilità	+	Non è per niente complesso e scalare aggiungere command.
Sviluppo	-	Lo sviluppo del command richiede una complessa riflessione oltre che alla creazione di più classi per implementarlo al meglio.

Tabella 10: Analisi Command

è qui che interviene l'interfaccia (come classe astratta Command) che ne definisce le proprietà per eseguire la richiesta.
Parametrizzazione di oggetti sull'azione da eseguire. Specifica, accordare ed eseguire richieste molteplici volte. Se necessario supporto anche delle operazioni di annullamento e ripristino. Oltre a supportare la transizione con un comando equivalente ad un operazione atomica.

- **Considerazioni** : L'accoppiamento tra oggetto invocante e quelle che portano a termine l'operazione è più lasco. È il Command che svolge operazioni differenti e può essere tranquillamente esteso con tutti i pregi del caso.
  Così si forma il binding fra il ricevente e l'azione da eseguire. L'oggetto dell'operazione non è deciso al momento della creazione delle azioni ma a tempo di esecuzione. È possibile incapsulare un'azione in modo che questa sia atomica. È così possibile implementare un paradigma basato su transazioni in cui un insieme di operazioni è svolto in toto o per nulla. Il Command può memorizzare lo stato precedente alla sua esecuzione, ripristinandolo qualora l'operazione debba essere annullata. E infine non meno importante è possibile rendere asincrona la scelta dei comandi rispetto alla loro esecuzione. Un certo numero di command possono essere consumati da un altro oggetto che li riceve in un tempo diverso dalla loro selezione.
- **Analisi del Pattern** :

B MockUp

Il mockup è l'attività di riprodurre un oggetto o modello in scala ridotta o maggiorata. In questa sezione infatti riportiamo e descriviamo tre delle principali visioni della nostra applicazione web. L'idea è di presentare, oltre alla prima pagina, due delle importanti funzioni del nostro prodotto: la pagina del profilo utente e la ricerca di un nuovo microservizio. Evitando le pagine di inserimento di un nuovo microservizio e statistica del nostro prodotto che presenteranno rispettivamente una lista di form per l'inserimento del microservizio e una serie di grafici non ancora opportunamente studiati.
Per la nostra visione grafica sono stati scelti espressamente due modelli molto importanti, già compresi e assimilati dai consumatori finali:

- Store di Apple (Formato Desktop);
- GitHub website.

Le motivazioni delle scelte sono che nel primo caso il prodotto ha alle spalle innumerevoli designer ed esperti del settore. Ha una visione commerciale che invita il consumatore all'acquisto. I servizi sono offerti in modo chiaro per categorie e ci fornisce uno stimolo creativo su cui prendere spunto pulito ed elegante.
Nel secondo caso la scelta è stata riposta per la maggior parte delle nostra visione. È intanto un servizio offerto per programmatori da programmatori, quindi pensiamo che i requisiti coincidano: cioè una distribuzione di servizi per programmatori che offrono e cercano nuovi servizi per i loro obbiettivi. Ha una buonissima sezione sociale, niente di invasivo come grandi piattaforme moderne, ma abbastanza per rimanere in contatto con le sezioni di interesse. Anche qui ci sono più categorie e più livelli di ricerca e di immagazzinamento. I prodotti che vengono caricati sono simili alla nostra visione di microservizi (avranno una documentazione, un interfaccia, un numero di dati di dettaglio). Infine la pagina principale è molto simile all'idea che voglia far avere al supporter dei microservizi.
Ora in dettaglio mostriamo le scelte fatte per le abbiamo fatto di design dell'interfaccia.

B.1 Home

La home è la prima pagina che ci si trova dopo la corretta fase di login.

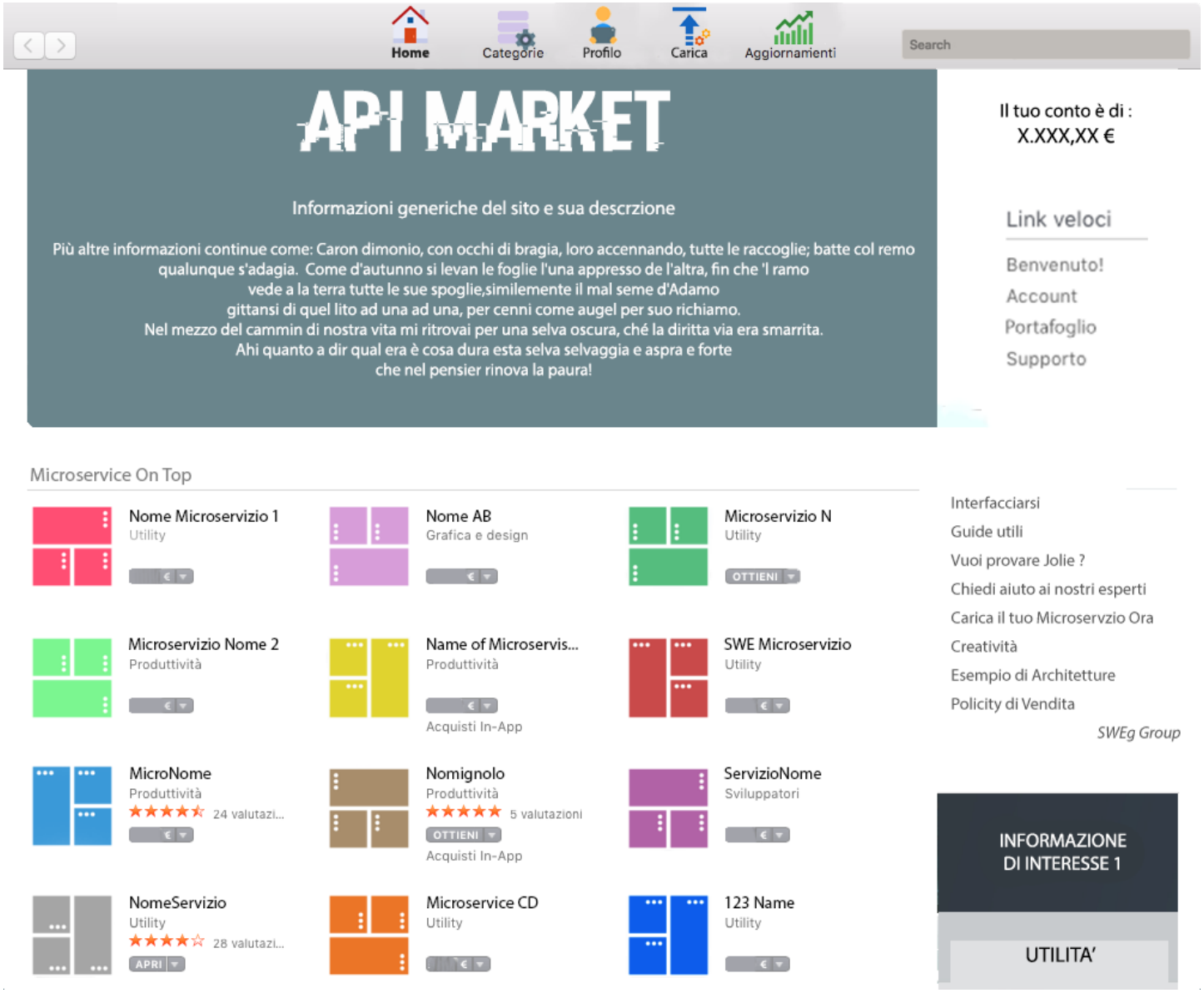


Figura 33: Idea di visione della Home

La prima pagina offre un menù con le principali operazioni che l'utente finale può eseguire. Notiamo:

- Una ordine dei microservizi per *Categoria*. In un futuro rigoglioso per la produzione di microservizi, colui che usufruirà del servizio deve avere visione specifica e caratteristica di quello che vuole per comporre la sua API_g. Una differenziazione intelligente faciliterà la progettazione del API_g finale e quindi il cliente tornerà soddisfatto della prestazione d'opera.
- *Profilo* è una pagina che andremo a presentare dopo. Messa apposta in seguito alle categorie perché è secondario dall'idea che voglia imprimere al consumatore.
- La sezione *Carica*, che abbiamo deciso di non rappresentare con un prototipo è l'idea del caricamento di un nuovo microserivio. Perché vogliamo far si che oltre all'acquisto vi sia un'importante valore di popolazione del prodotto finale. La pagina conterrà, come già spiegato, una serie di form che permette al nostro attore di caricare il suo personale microservizio.
- Infine la parte di *Aggiornamenti* non è ancora stata ideata alla perfezione , ma l'idea è di dare una visione dell'andamento dei microservizi collegati all'account oltre che un piccolo reportage del portafoglio modificato nel tempo.

Il menù viene inteso come header del sito e si ritroverà in ogni pagina con la variante che nella sezione attuale il link sarà in grossetto. Non mancheranno la possibilità di cercare grazie alla form e un comodo set di pulsanti "avanti" e "indietro".
Subito sotto l'header troviamo nella parte sinistra un titolo semplice con descrizione. Accanto dei link veloci ed utili per l'utente oltre che l'importante importo disponibile nell'account.
Terza e ultima parte della nostra Home è una serie di importanti microservizi che gestiamo noi personalmente nella prima fase del progetto. L'idea è che un algoritmo auto genera la home con i microservizi che più sono attinenti con le ricerche del account, o più fiorenti equilibrando durata da quando sono stati prodotti e rating generico. Ma sono ancora solamente idee.
Sulla destra invece link utili sui microservizi di ordine generale, che possono variare da domande frequenti a vere e proprio guide alla creazione del microservizio.

B.2 Profilo

La seguente pagina che vogliamo presentare è la sezione profilo. Come da standard il link presenta il nome in grassetto.

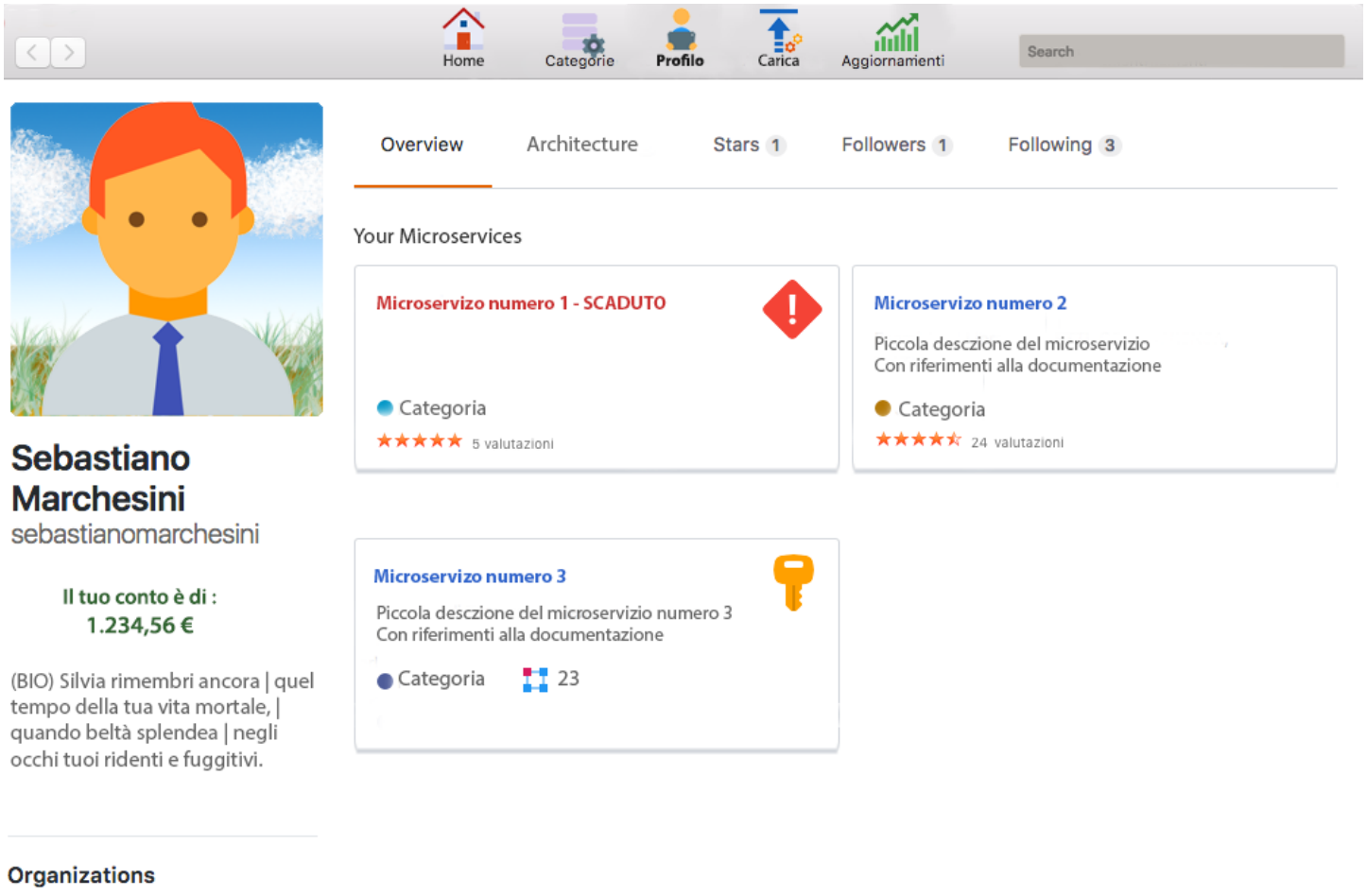


Figura 34: Idea di visione della Pagina profilo

La prima pagina offre un menù con le principali operazioni che l'utente finale può eseguire. Notiamo oltre al principale "header" che non è cambiato; due sezioni della pagina:

- La parte a sinistra con che comprende l'anagrafe del nostro profilo. Con i principali dati che distinguano la parte sociale della nostra applicazione.
  - Un'immagine profilo caricata precedentemente;
  - Nome;
  - Cognome;
  - UserId;
  - ContoCorrente;
  - BIO.
- Nella parte più estesa invece una serie di microservizi a noi collegati. Prima troviamo dei sotto-menù del nostro profilo, ancora non studiati dettagliatamente.
  Subito sottostante in questo esempio abbiamo una visione panoramica dei microservizi. Uno in particolare a cui è scaduta la chiave e quindi inutilizzabile; gli altri due che potrebbero essere intesi come acquistato e caricato. Vediamo che anche qui ritornano le categorie a cui appartengono i microservizi, con altre informazioni come il rating (con il numero di commenti) e il simbolo che indica il numero delle connessioni. Quest'ultimo è molto generico, l'idea finale del prodotto sarà più preciso e completo, può presentare anche il numero di byte trasmessi o numero di keys vendute.

B.3 Ricerca

Ricerca è una pagina utilizzata dagli utenti autenticati per selezionare, studiare e informarsi su una stringa preimpostata.

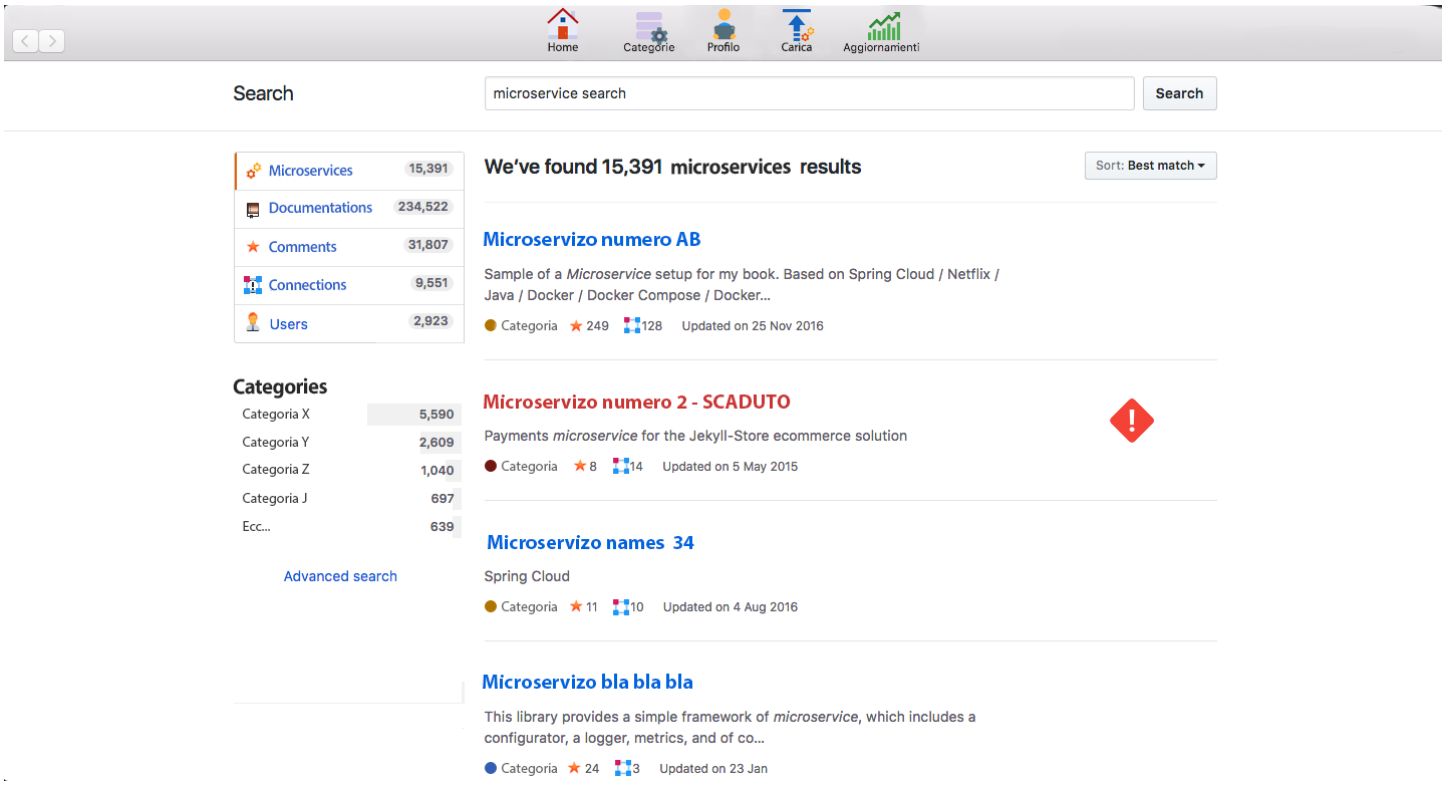


Figura 35: Idea di visione della Pagina Ricerca

Intendiamo come stringa qualsiasi cosa, non per forza una parola che si associ a un microservizio ma anche un utente o una riferimento alla documentazione.

A differenza delle pagine precedenti la form della ricerca è situata appena sotto l'header, al centro e con un formato ingrandito. Capiamo automaticamente che è il punto focale della sezione.
Anche qui la struttura del nostro prodotto è principalmente divisa in due colonne per non distrarre il consumatore finale. Ne descriviamo quindi in due porzioni:

- La parte sinistra è una parte di raffinamento della ricerca. Possiamo quindi perfezionarla con due diversi sotto-elenchi:
  - Il primo ti riporta a dove è presente la stringa tra le varie entità del database. Non sono per niente definitive, anzi. Ma si può immaginare che sicuramente una stringa riporterà al titolo di un microservizio , alla documentazione e a un utente. Perché veri e propri contenitori di attributi String.
  - Il secondo, come di consueto nell'applicazione, riporta alle categorie/tipologie che i nostri microservizi possono riportare. Al momento non ancora ideati lasciando una vaga intestazione.
  - Abbiamo pensato che la ricerca può essere ulteriormente affinata tramite link a una sezione di ricerca avanzata. Ma non è un requisito che ci siamo posti obbligatorio.
- Nel contenitore principale invece troviamo una lista ordinata verticalmente (a differenza della pagina Profilo) che presenta anche qui una visione generica dei microservizi. Verranno messi al primo posto i servizi più attinenti alla ricerca, probabilmente con all'interno del nome la stringa cercata.
  Un resoconto generico darà al consumatore finale la possibilità di farsi un idea del microservizio. In rosso e con un simbolo esclamativo un microservizio sconsigliato per inattività.