

# SWEG GROUP

## Specifica Tecnica

**Versione** 1.0.0

**Data di Rilascio** XX/XX/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.

# 1 Registro Modifiche

Ver.	Modifica	Nome	Data
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

<b>1</b>	<b>Registro Modifiche</b>	<b>2</b>
<b>2</b>	<b>Introduzione</b>	<b>5</b>
2.1	Scopo del Documento . . . . .	5
2.2	Glossario . . . . .	5
2.3	Riferimenti . . . . .	5
2.3.1	Normativi . . . . .	5
2.3.2	Informativi . . . . .	5
<b>3</b>	<b>Tecnologie Utilizzate</b>	<b>6</b>
3.1	Linguaggi . . . . .	6
3.1.1	HTML5 . . . . .	6
3.1.2	CSS3 . . . . .	6
3.1.3	Javascript . . . . .	6
3.1.4	Jolie . . . . .	7
3.1.5	SQL . . . . .	7
3.2	Frameworks . . . . .	8
3.2.1	AngularJS . . . . .	8
3.2.2	Spring Framework - Spring Boot . . . . .	9
3.3	Librerie . . . . .	10
3.3.1	Leonardo: il web server di Jolie . . . . .	10
3.3.2	Highcharts . . . . .	10
3.3.3	Angular Material . . . . .	10
3.3.4	Highcharts-ng . . . . .	11
<b>4</b>	<b>Descrizione Architettura</b>	<b>12</b>
4.1	Metodo di specifica . . . . .	12
4.2	Architettura generale . . . . .	12
<b>5</b>	<b>Titolo</b>	<b>13</b>
5.1	SottoTitolo . . . . .	13
5.2	SottoTitolo . . . . .	13
5.3	Sottotitolo . . . . .	13
5.3.1	SottoSottoTitolo . . . . .	13
<b>6</b>	<b>Architettura Back End</b>	<b>14</b>
6.1	com.apim.server . . . . .	14
6.1.1	Informazioni sul Package . . . . .	14
6.1.2	Package contenuti . . . . .	14
6.1.3	Relazione tra le componenti interne . . . . .	14
6.2	com.apim.server.controllers . . . . .	14
6.2.1	Informazioni sul Package . . . . .	14
6.2.2	Classi . . . . .	14
6.3	com.apim.server.services . . . . .	14
6.3.1	Informazioni sul Package . . . . .	14
6.3.2	Packages contenuti . . . . .	15
6.3.3	Classi . . . . .	15
6.3.4	Interazioni con altri componenti . . . . .	15

6.4	com.apim.server.gateway . . . . .	15
6.4.1	Informazioni sul Package . . . . .	15
6.4.2	Classi . . . . .	16
6.4.3	Interazioni con altri componenti . . . . .	16
6.5	com.apim.server.dataAnalysis . . . . .	16
6.5.1	Informazioni sul Package . . . . .	16
6.5.2	Classi . . . . .	16
6.5.3	Interazioni con altri componenti . . . . .	16
6.6	com.apim.server.keyManager . . . . .	17
6.6.1	Informazioni sul Package . . . . .	17
6.6.2	Classi . . . . .	17
6.6.3	Interazioni con altri componenti . . . . .	17
6.7	com.apim.server.user . . . . .	17
6.7.1	Informazioni sul Package . . . . .	17
6.7.2	Classi . . . . .	18
6.7.3	Interazioni con altri componenti . . . . .	18
6.8	com.apim.server.admin . . . . .	18
6.8.1	Informazioni sul Package . . . . .	18
6.8.2	Classi . . . . .	18
6.8.3	Interazioni con altri componenti . . . . .	20
6.9	com.apim.server.shared . . . . .	20
6.9.1	Informazioni sul Package . . . . .	20
6.9.2	Classi . . . . .	20
6.9.3	Interazioni con altri componenti . . . . .	23
6.10	com.apim.server.dao . . . . .	23
6.10.1	Informazioni sul Package . . . . .	23
6.10.2	Classi . . . . .	23
6.10.3	Interazioni con altri componenti . . . . .	25
6.11	com.apim.server.entities . . . . .	25
6.11.1	Informazioni sul Package . . . . .	25
6.11.2	Classi . . . . .	25
6.11.3	Interazioni con altri componenti . . . . .	27

## Elenco delle tabelle

## Elenco delle figure

1	Architettura di massima del progetto . . . . .	14
3	Testo figura . . . . .	51
4	Testo figura . . . . .	52
5	Testo figura . . . . .	52
6	Testo figura . . . . .	52
7	Testo figura . . . . .	52
8	Testo figura . . . . .	52
9	Testo figura . . . . .	52
10	Testo figura . . . . .	53
11	Testo figura . . . . .	53
12	Testo figura . . . . .	53

13	Testo figura . . . . .	53
14	Progettazione Concettuale . . . . .	54
15	Progettazione Logica . . . . .	55
16	Architettura Microservizi . . . . .	56
17	mvvm . . . . .	56
18	DAO . . . . .	56
19	Abstract Factory . . . . .	56
20	Dependency Injection . . . . .	56
21	Facade . . . . .	56
22	Command . . . . .	56
24	Testo figura . . . . .	57
26	Testo figura . . . . .	58
27	Illustrazione Architettura a Microservizi . . . . .	59
28	Illustrazione Model View ViewModel . . . . .	59
29	Illustrazione Pattern Abstract Factory . . . . .	59
30	Illustrazione Pattern Dependency Injection . . . . .	60
31	Illustrazione Pattern Facade . . . . .	60
32	Illustrazione Pattern Command . . . . .	61
33	Idea di visione della Home . . . . .	62
34	Idea di visione della Pagina profilo . . . . .	63

## 2 Introduzione

### 2.1 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 obiettivi;
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.

### 2.2 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.

### 2.3 Riferimenti

#### 2.3.1 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 v1.0.0".

#### 2.3.2 Informativi

- **Metriche di Progetto:**  
[https://it.wikipedia.org/wiki/Metriche\\_di\\_progetto](https://it.wikipedia.org/wiki/Metriche_di_progetto).
- **Modello incrementale:**  
[https://it.wikipedia.org/wiki/Modello\\_incrementale](https://it.wikipedia.org/wiki/Modello_incrementale).
- **Modello incrementale:**  
[https://it.wikipedia.org/wiki/Metodologia\\_agile](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>.

## 3 Tecnologie Utilizzate

In questa sezione illustreremo le tecnologie e i linguaggi utilizzati nel progetto, indicando pro e contro delle scelte utilizzate.

### 3.1 Linguaggi

#### 3.1.1 HTML5

HTML è la particella elementare di Internet: il linguaggio di markup 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.

**Vantaggi :**

- Funziona su la maggior parte dei computer e sui dispositivi mobili;
- Video e animazioni supportati senza plug-in<sub>g</sub> esterni;
- Pensiero indirizzato sempre più al web semantico e HTML5 ne è portavoce;
- Maggior flessibilità e potenzialità rispetto alle precedenti versioni.

**Svantaggi :**

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

#### 3.1.2 CSS3

Il CSS è un linguaggio con il quale formattare le pagine Web. Un file CSS 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. **Vantaggi :**

- Separata la struttura del sito dalla presentazione;
- Più facile progettazione di accessibilità;
- Template unico per varie pagine senza ripetizione;
- Facile la modifica in caso di cambiamento;
- Grafica accattivante per gli utenti.

**Svantaggi :**

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

#### 3.1.3 Javascript

La caratteristica principale di Javascript, è quella di essere un linguaggio di scripting. 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, in pagine JSP o in appositi file separati con estensione .js poi richiamati nella logica di business. **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, ma occorre abbinarlo ad altri linguaggi evoluti, ( come Jolie );
- Il codice Javascript viene eseguito sul client per cui il server non è sollecitato più del dovuto e la velocità dell'applicazione complessiva è migliore;

**Svantaggi :**

- Il 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.

**3.1.4 Jolie**

Jolie 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.

**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, 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 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<sub>g</sub>: servizio Jolie 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'iterazione con database non relazionali;
- Linguaggio nuovo e non usato in ogni suo ramo e sfaccettatura, manca infatti documentazione completa ed esaustiva.

**3.1.5 SQL**

SQL è il linguaggio che andremo ad usare per quanto riguarda la parte di database della nostra applicazione web. Jolie offre dei comandi semplici e si interfacciano comodamente con il linguaggio per basi di dati relazionali.

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;
- 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.

## 3.2 Frameworks

### 3.2.1 AngularJS

AngularJS<sub>g</sub> è un framework JavaScript 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, AngularJS da un lato esalta e potenzia l'approccio dichiarativo del HTML nella definizione dell'interfaccia grafica, dall'altro fornisce strumenti per la costruzione di un'architettura modulare e testabile della logica applicativa di un'applicazione.

Angular 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

**Vantaggi :**

- Estremamente espressivo, leggibile e di facile implementazione. Un'evoluzione all'HTML 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 (che nei prossimi capitoli illustriamo) Model View ViewModel;
- Data-Binding Bidirezionale di AngularJS gestisce la sincronizzazione tra il DOM e il modello, e viceversa;
- AngularJS ha un sottosistema integrato di Independence Injection<sub>g</sub> che aiuta lo sviluppatore a creare un'applicazione facile da sviluppare, capire e provare;
- Utilizza le direttive<sub>g</sub>, cioè possono essere usate per creare tag HTML 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.

**Svantaggi :**

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

**3.2.2 Spring Framework - Spring Boot**

Il progetto Spring Boot permette di semplificare, e di molto, lo sviluppo di applicazioni basate sul framework Spring.

La Spring Framework è un framework applicativo e inverte i controlli del contenitore per la piattaforma Java. Le funzionalità di base del framework possono essere utilizzate da qualsiasi applicazione Java, ma ci sono delle estensioni per la creazione di applicazioni web. **Vantaggi :**

- Pur essendo molto ampio, grazie alla sua modularità si può scegliere di integrare solo alcune parti all'interno del progetto;
- Open Source<sub>g</sub>;
- Basato su piattaforma Java, già conosciuta dei membri del gruppo;
- Adatto per le RESTful web service framework;
- Semplifica le sviluppo delle applicazioni basate su Spring;
- Risolve il problema di quali librerie Spring utilizzare e quale versione;
- Risolve il problema dell'individuazione e configurazione di tutti i bean che saranno gestiti dal framework e necessari alla nostra applicazione.

**Svantaggi :**

- Poca documentazione ufficiale (il sito contiene una sola pagina funzionante!);
- Poca o nulla conoscenza da parte del gruppo;

### 3.3 Librerie

#### 3.3.1 Leonardo: il web server di Jolie

Leonardo è un server web sviluppata in puro Jolie. È molto flessibile e può scalare da un semplice contenuto statico HTML fino a sostenere un complesso servizio web dinamico.

**Vantaggi :**

- Scritto interamente in Jolie e facilmente implementabile nel prodotto;
- Si interfaccia con HTML, JQuery;
- Permette l'uso dei Cookies<sub>g</sub>.

**Svantaggi :**

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

#### 3.3.2 Highcharts

È una libreria JavaScript 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.

**Vantaggi :**

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

**Svantaggi :**

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

#### 3.3.3 Angular Material

Angular Material è l'implementazione del Material Design in AngularJS. Fornisce un insieme di componenti per l'interfaccia utente riutilizzabili, testati e accessibili, basati sul Material Design<sub>g</sub>.

**Vantaggi :**

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

**Svantaggi :**

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

### 3.3.4 Highcharts-ng

È una direttiva AngularJS per Highcharts. Servirà nella pratica ad usare le due librerie nel framework scelto.

**Vantaggi :**

- Ausilio integrale della libreria Highcharts;
- La documentazione presenta eventi ed è possibile comunicare direttamente con l'ideatore.

**Svantaggi :**

- È una tecnologia in continuo sviluppo migliorata dagli utenti, ha possibili variazioni e aggiornamenti.

## 4 Descrizione Architettura

### 4.1 Metodo di specifica

Il metodo scelto per esporre l'architettura è tramite un approccio top-down, il che vuol dire che verrà prima presentata una architettura molto astratta e poi verrà 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 Progettazione in dettaglio. Partiremo quindi ad analizzare il rapporto che è 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<sub>g</sub> ha chiesto esplicitamente che venisse utilizzato all'interno del progetto il linguaggio Jolie<sub>g</sub>, un linguaggio da lui sviluppato, e ha messo a disposizione anche Leonardo<sub>g</sub>, per cui nella fase di progettazione il team ha dovuto integrare questa tecnologia al resto del progetto.

### 4.2 Architettura generale

Il Proponente<sub>g</sub> nel presentare il capitolato ha riportato un'architettura di massima che rappresenterà come dovrà essere il prodotto finale. Tale architettura è la seguente:

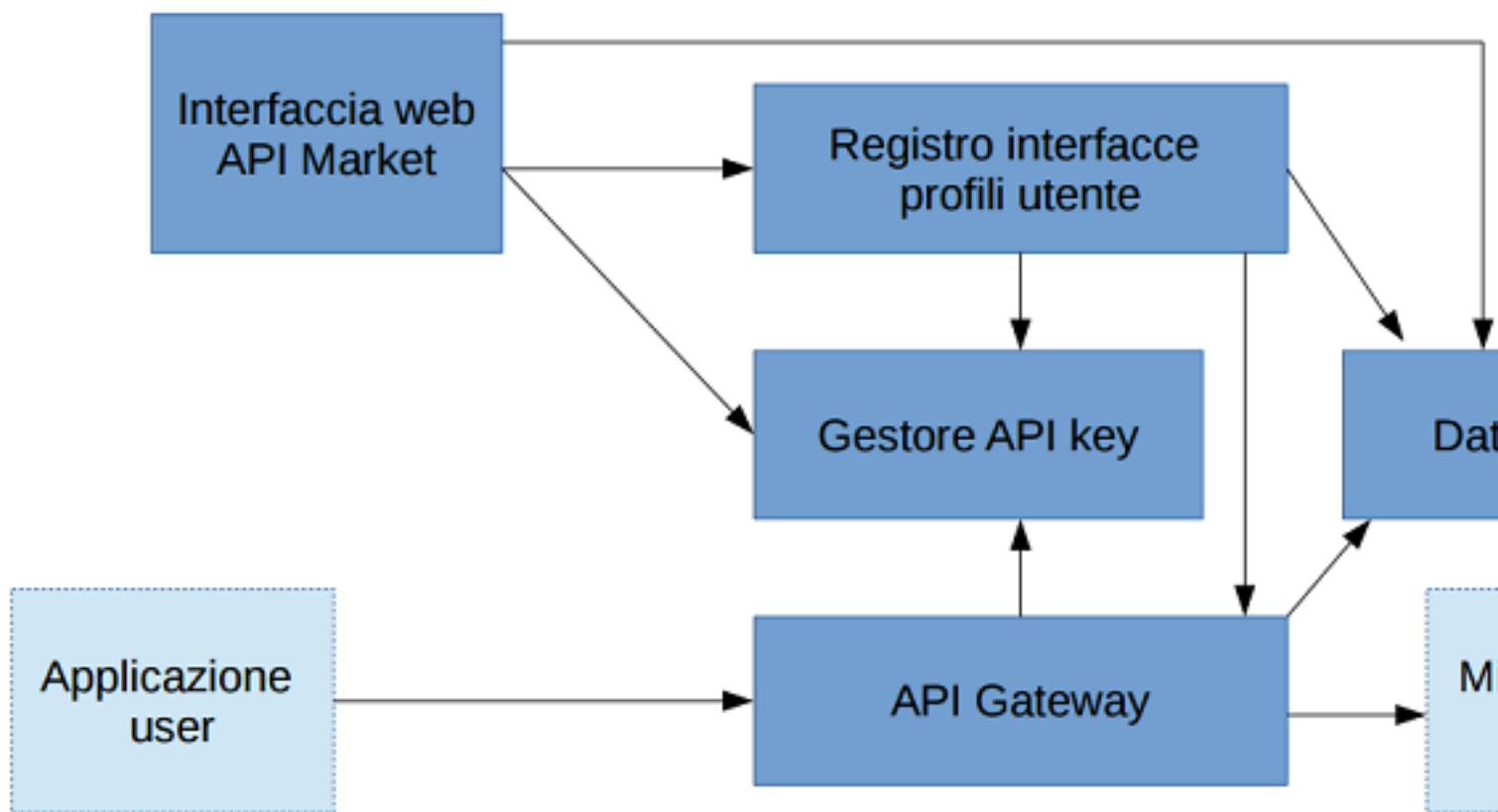


Figura 1: Architettura di massima del progetto

L'APIMarket<sub>g</sub> dovrà permettere ad un utente di caricare una API<sub>g</sub>, deve poter acquistare APIKey<sub>g</sub> che gli permettano di utilizzare API<sub>g</sub> già presenti nell'APIMarket<sub>g</sub> e infine avere la possibilità di monitorare i dati riferiti alle API<sub>g</sub>. L'architettura sopra riportata rappresenta una possibile soluzione di come deve essere il prodotto finale: attraverso un APIGateway<sub>g</sub> è possibile controllare che le API<sub>g</sub> caricate siano

conformi a standard che verranno concordati con il Proponente<sub>g</sub>, sempre l'APIGateway<sub>g</sub> si occuperà di gestire acquisto e la validazione dell'APIKey<sub>g</sub> in possesso dell'utente (o che desidera acquistare), infine sarà sempre lui a monitorare le chiamate alle API<sub>g</sub> e a fornire i dati necessari richiesti. Per interfacciare l'utente all'APIGateway<sub>g</sub> 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<sub>g</sub>. Il Proponente<sub>g</sub> ha espressamente chiesto che il team usufruisse di una tecnologia da lui creata, ovvero Jolie<sub>g</sub>, perciò il team ha deciso di sviluppare il Back-end<sub>g</sub> secondo un'architettura a microservizi così da poter sfruttare a pieno le potenzialità di questa tecnologia.

## 5 Titolo

Testo:

- lista;
- lista.

### 5.1 SottoTitolo

Testo.

### 5.2 SottoTitolo

Testo.

Testo.

Testo.

Testo:

- lista; con:
  - sottolista
  - **testogrossetto**: da da da;
  - *Testo corsivo*.
  - Ancora testo.
  - testo.
- Continua la lista di prima.

### 5.3 Sottotitolo

Testo.

#### 5.3.1 SottoSottoTitolo

Testo `http://www.url.com/`.

## 6 Architettura Back End

### 6.1 Microservizi Jolie

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

- ControllerInterface
  - **Descrizione:** Interfaccia che denifisce le operazioni disponibili nel microservizio Controller;
  - **Relazione con altri microservizi:**
    - \* Controller;
- Controller
  - **Descrizione:** Microservizio che riceve le chiamate dalla web-app;
  - **Relazione con altri microservizi:**
    - \* UserManager;
    - \* KeyManager;
    - \* Analyzer;
- 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:**
- GatewayInterface
  - **Descrizione:** Interfaccia che definisce 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;

## 6.2 com.apim.server

### 6.2.1 Informazioni sul Package

Contiene il back end dell'applicazione.

### 6.2.2 Package contenuti

- \*.controllers;
- \*.services;
- \*.dao;
- \*.entities;

### 6.2.3 Relazione tra le componenti interne

Di seguito sono illustrate le relazioni tra le componenti dell'applicazione.

## 6.3 com.apim.server.controllers

### 6.3.1 Informazioni sul Package

Package contenente le classi che espongono le funzioni chiamabili dall'utente;

### 6.3.2 Classi

- **UserController**
  - **Descrizione:** Questa classe contiene i metodi invocabili da un utente normale;
  - **Relazione con altre classi:**
    - \* com.apim.server.services.user.\*;
- **AdminController**
  - **Descrizione:** Questa classe contiene i metodi invocabili da un utente amministratore;
  - **Relazione con altre classi:**
    - \* com.apim.server.services.admin.\*;
- **SharedController**
  - **Descrizione:** Questa classe contiene i metodi invocabili sia da un utente normale che da un amministratore;
  - **Relazione con altre classi:**
    - \* com.apim.server.services.shared.\*;

## 6.4 com.apim.server.services

### 6.4.1 Informazioni sul Package

Package contenente le classi che si occupano di implementare in Java le funzionalità dei microservizi Jolie responsabili di ricevere le richieste dai controllers e inoltrarle al livello sottostante. Inoltre ogni Service implementerà l'interfaccia IService contenuta in questo package.

### 6.4.2 Packages contenuti

- com.apim.server.services.gateway;
- com.apim.server.services.keyManager;
- com.apim.server.services.dataAnalysis;
- com.apim.server.services.user;
- com.apim.server.services.admin;
- com.apim.server.services.shared;

### 6.4.3 Classi

- **IService**
  - **Descrizione:** Interfaccia che verrà implementata da ogni Service;
  - **Relazione con altre classi:**
    - \* com.apim.server.gateway.\*;
    - \* com.apim.server.keyManager.\*;
    - \* com.apim.server.dataAnalysis.\*;
    - \* com.apim.server.user.\*;
    - \* com.apim.server.admin.\*;
    - \* com.apim.server.shared.\*;

#### 6.4.4 Interazioni con altri componenti

- com.apim.server.services.controllers;
- com.apim.server.services.gateway;
- com.apim.server.services.keyManager;
- com.apim.server.services.dataAnalysis;
- com.apim.server.services.user;
- com.apim.server.services.admin;
- com.apim.server.services.shared;

### 6.5 com.apim.server.gateway

#### 6.5.1 Informazioni sul Package

Package contenente le classi che implementano le funzionalità del gateway;

#### 6.5.2 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.dao.DAOFactory;
    - \* com.apim.server.dataAnalysis.\*;
    - \* com.apim.server.entities.User;
    - \* com.apim.server.entities.API;
    - \* com.apim.server.entities.Key;

#### 6.5.3 Interazioni con altri componenti

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

### 6.6 com.apim.server.dataAnalysis

#### 6.6.1 Informazioni sul Package

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

### 6.6.2 Classi

- **ServiceAnalyzer**

- **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.dao.DAOFactory;
  - \* com.apim.server.entities.User;
  - \* com.apim.server.entities.API;
  - \* com.apim.server.entities.Key;

### 6.6.3 Interazioni con altri componenti

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

## 6.7 com.apim.server.keyManager

### 6.7.1 Informazioni sul Package

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

### 6.7.2 Classi

- **ServiceKeygen**

- **Descrizione:** Questa classe contiene i metodi necessari alla creazione di una nuova key;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;
  - \* com.apim.server.entities.User;
  - \* com.apim.server.entities.API;
  - \* com.apim.server.entities.Key;

- **ServiceDecodeKey**

- **Descrizione:** Questa classe contiene i metodi necessari alla decodifica di una Key inviata dall'utente;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;
  - \* com.apim.server.entities.User;
  - \* com.apim.server.entities.API;
  - \* com.apim.server.entities.Key;

- **ServiceCheckKey**

- **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.dao.DAOFactory;
  - \* com.apim.server.entities.User;
  - \* com.apim.server.entities.Key;

### 6.7.3 Interazioni con altri componenti

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

## 6.8 com.apim.server.user

### 6.8.1 Informazioni sul Package

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

### 6.8.2 Classi

- **ServiceSignIn**
  - **Descrizione:** Questa classe contiene i metodi necessaria alla registrazione dell'utente;
  - **Relazione con altre classi:**
    - \* com.apim.server.dao.DAOFactory;
    - \* com.apim.server.entities.User;
    - \* com.apim.server.entities.UserController;
- **ServiceBuy**
  - **Descrizione:** Questa classe contiene i metodi necessari ad acquistare una API dal market;
  - **Relazione con altre classi:**
    - \* com.apim.server.dao.DAOFactory;
    - \* com.apim.server.entities.User;
    - \* com.apim.server.entities.Acquisto;
    - \* com.apim.server.entities.API;
    - \* com.apim.server.entities.Key;
    - \* com.apim.server.controllers.UserController;

### 6.8.3 Interazioni con altri componenti

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

## 6.9 com.apim.server.admin

### 6.9.1 Informazioni sul Package

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

### 6.9.2 Classi

- **ServiceSearchUser**
  - **Descrizione:** Questa classe contiene i metodi necessaria alla ricerca di un profilo utente da parte di un amministratore;
  - **Relazione con altre classi:**
    - \* com.apim.server.dao.DAOFactory;
    - \* com.apim.server.entities.User;
    - \* com.apim.server.entities.AdminController;
- **ServiceViewUserAdmin**
  - **Descrizione:** Questa classe contiene i metodi necessari a visualizzare i dettagli di un utente da parte di un amministratore;
  - **Relazione con altre classi:**
    - \* com.apim.server.dao.DAOFactory;
    - \* com.apim.server.entities.User;
    - \* com.apim.server.entities.AdminController;
- **ServiceEditProfileAdmin**
  - **Descrizione:** Questa classe contiene i metodi necessari a modificare i dettagli di un utente da parte di un amministratore;
  - **Relazione con altre classi:**
    - \* com.apim.server.dao.DAOFactory;
    - \* com.apim.server.entities.User;
    - \* com.apim.server.entities.AdminController;
- **ServiceEditAPIAdmin**
  - **Descrizione:** Questa classe contiene i metodi necessari a modificare i dettagli di una API da parte di un amministratore;
  - **Relazione con altre classi:**
    - \* com.apim.server.dao.DAOFactory;
    - \* com.apim.server.entities.API;
    - \* com.apim.server.entities.Categoria;
    - \* com.apim.server.entities.AdminController;
- **ServiceDeleteAPIAdmin**
  - **Descrizione:** Questa classe contiene i metodi necessari a rimuovere una API dal market da parte di un amministratore;
  - **Relazione con altre classi:**

- \* com.apim.server.dao.DAOFactory;
- \* com.apim.server.entities.API;
- \* com.apim.server.entities.AdminController;

- **ServiceEditComment**

- **Descrizione:** Questa classe contiene i metodi necessari a modificare un Commento;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;
  - \* com.apim.server.entities.API;
  - \* com.apim.server.entities.Comment;
  - \* com.apim.server.entities.AdminController;

- **ServiceDeleteComment**

- **Descrizione:** Questa classe contiene i metodi necessari a rimuovere un Commento;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;
  - \* com.apim.server.entities.Comment;
  - \* com.apim.server.entities.AdminController;

- **ServiceInsertCategoria**

- **Descrizione:** Questa classe contiene i metodi necessari alla creazione di una nuova categoria;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;
  - \* com.apim.server.entities.Categoria;
  - \* com.apim.server.entities.AdminController;

- **ServiceEditCategoria**

- **Descrizione:** Questa classe contiene i metodi necessari a modificare una Categoria;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;
  - \* com.apim.server.entities.Categoria;
  - \* com.apim.server.entities.AdminController;

- **ServiceDeleteCategoria**

- **Descrizione:** Questa classe contiene i metodi necessari a rimuovere una Categoria;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;
  - \* com.apim.server.entities.Categoria;
  - \* com.apim.server.entities.AdminController;

### 6.9.3 Interazioni con altri componenti

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

## 6.10 com.apim.server.shared

### 6.10.1 Informazioni sul Package

Package contenente le classi che implementano le funzionalità disponibili sia ad un utente normale che ad un utente amministratore;

### 6.10.2 Classi

- **ServiceLogin**

- **Descrizione:** Questa classe contiene i metodi necessari ad eseguire il login;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;
  - \* com.apim.server.entities.User;
  - \* com.apim.server.entities.SharedController;

- **ServiceSearch**

- **Descrizione:** Questa classe contiene i metodi necessari alla ricerca di una API da parte di un utente;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;
  - \* com.apim.server.entities.User;
  - \* com.apim.server.entities.SharedController;

- **ServiceViewSearch**

- **Descrizione:** Questa classe contiene i metodi necessari a visualizzare i risultati di una ricerca da parte di un utente;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;
  - \* com.apim.server.entities.API;
  - \* com.apim.server.entities.SharedController;

- **ServiceViewUser**

- **Descrizione:** Questa classe contiene i metodi necessari a visualizzare le informazioni generali di un utente, quali ad esempio il suo rating;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;
  - \* com.apim.server.entities.User;
  - \* com.apim.server.entities.SharedController;

- **ServiceViewAPI**

- **Descrizione:** Questa classe contiene i metodi necessari alla visualizzazione dei dettagli di una API da parte di un utente;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;



- \* com.apim.server.entities.API;
- \* com.apim.server.entities.SharedController;

#### • ServiceLogout

- **Descrizione:** Questa classe contiene i metodi necessari ad un utente per eseguire il logout;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;
  - \* com.apim.server.entities.SharedController;

#### • ServiceEditProfile

- **Descrizione:** Questa classe contiene i metodi necessari a modificare il proprio profilo utente;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;
  - \* com.apim.server.entities.User;
  - \* com.apim.server.entities.SharedController;

#### • ServiceViewStats

- **Descrizione:** Questa classe contiene i metodi necessari a Visualizzare le statistiche di una particolare API;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;
  - \* com.apim.server.entities.API;
  - \* com.apim.server.entities.SharedController;

#### • ServiceInsertAPI

- **Descrizione:** Questa classe contiene i metodi necessari a inserire una nuova API nel market ed associarla al proprio profilo;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;
  - \* com.apim.server.entities.User;
  - \* com.apim.server.entities.API;
  - \* com.apim.server.entities.Categoria;
  - \* com.apim.server.entities.SharedController;

#### • ServiceEditAPI

- **Descrizione:** Questa classe contiene i metodi necessari a rimuovere una Categoria;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;
  - \* com.apim.server.entities.API;
  - \* com.apim.server.entities.Categoria;
  - \* com.apim.server.entities.SharedController;

#### • ServiceDeleteAPI

- **Descrizione:** Questa classe contiene i metodi necessari a rimuovere una API associata al proprio profilo dal market;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;
  - \* com.apim.server.entities.API;
  - \* com.apim.server.entities.SharedController;

#### • ServiceInsertComment

- **Descrizione:** Questa classe contiene i metodi necessari a inserire un nuovo Commento nel market ed associarlo ad una API;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;
  - \* com.apim.server.entities.API;
  - \* com.apim.server.entities.Comment;
  - \* com.apim.server.entities.User;
  - \* com.apim.server.entities.SharedController;

### 6.10.3 Interazioni con altri componenti

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

## 6.11 com.apim.server.dao

### 6.11.1 Informazioni sul Package

Package contenente le classi utilizzate per accedere direttamente al database;

### 6.11.2 Classi

#### • IDAOFactory

- **Descrizione:** Interfaccia per la creazione degli oggetti che andranno ad eseguire l'accesso diretto al database;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;

#### • DAOFactory

- **Descrizione:** Implementazione di IDAOFactory;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.IDAOFactory;
  - \* com.apim.server.user.\*;
  - \* com.apim.server.admin.\*;
  - \* com.apim.server.shared.\*;
  - \* com.apim.server.gateway.\*;
  - \* com.apim.server.keymanager.\*;

- \* com.apim.server.dataanalysys.\*;
- \* com.apim.server.dao.DAOAPI;
- \* com.apim.server.dao.DAOUser;
- \* com.apim.server.dao.DAOComment;
- \* com.apim.server.dao.DAOPeople;
- \* com.apim.server.dao.DAOKey;

#### ● IDAOGeneric

- **Descrizione:** Interfaccia che rappresenta un Data Access object generico;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.IDAOGeneric;

#### ● DAOGeneric

- **Descrizione:** Implementazione di IDAOGeneric;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.IDAOGeneric;
  - \* com.apim.server.dao.DAOAPI;
  - \* com.apim.server.dao.DAOUser;
  - \* com.apim.server.dao.DAOComment;
  - \* com.apim.server.dao.DAOPeople;
  - \* com.apim.server.dao.DADAOKeYUser;

#### ● IDAOAPI

- **Descrizione:** Questa classe esegue l'accesso alla tabella API del database;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;

#### ● DAOAPI

- **Descrizione:** Implementazione di IDAOAPI;
- **Relazione con altre classi:**
  - \* com.apim.server.entities.\*;
  - \* com.apim.server.dao.IDAOAPI;
  - \* com.apim.server.dao.DAOGeneric;

#### ● IDAOUser

- **Descrizione:** Questa classe esegue l'accesso alla tabella User del database;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;

#### ● DAOUser

- **Descrizione:** Implementazione di IDAOUser;
- **Relazione con altre classi:**

- \* com.apim.server.entities.\*;
- \* com.apim.server.dao.IDAOUser;
- \* com.apim.server.dao.DAOGeneric;

- **IDAOCComment**

- **Descrizione:** Questa classe esegue l'accesso alla tabella Comment del database;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;

- **DAOComment**

- **Descrizione:** Implementazione di IDAOCComment;
- **Relazione con altre classi:**
  - \* com.apim.server.entities.\*;
  - \* com.apim.server.dao.IDAOCComment;
  - \* com.apim.server.dao.DAOGeneric;

- **IDAOPeople**

- **Descrizione:** Questa classe esegue l'accesso alla tabella People del database;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;

- **DAOPeople**

- **Descrizione:** Implementazione di IDAOPeople;
- **Relazione con altre classi:**
  - \* com.apim.server.entities.\*;
  - \* com.apim.server.dao.IDAOPeople;
  - \* com.apim.server.dao.DAOGeneric;

- **IDAOKey**

- **Descrizione:** Questa classe esegue l'accesso alla tabella Key del database;
- **Relazione con altre classi:**
  - \* com.apim.server.dao.DAOFactory;

- **DAOKey**

- **Descrizione:** Implementazione di IDAOKey;
- **Relazione con altre classi:**
  - \* com.apim.server.entities.\*;
  - \* com.apim.server.dao.IDAOKey;
  - \* com.apim.server.dao.DAOGeneric;

### 6.11.3 Interazioni con altri componenti

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

## 6.12 com.apim.server.entities

### 6.12.1 Informazioni sul Package

Package contenente le classi che rappresentano le entità del database;

### 6.12.2 Classi

- **IEntity**

- **Descrizione:** Interfaccia che rappresenta un'entità generica;
- **Relazione con altre classi:**
  - \* com.apim.server.entities.User;
  - \* com.apim.server.entities.API;
  - \* com.apim.server.entities.Key;
  - \* com.apim.server.entities.Comment;
  - \* com.apim.server.entities.Acquisto;
  - \* com.apim.server.entities.Categoria;

- **IEntityFactory**

- **Descrizione:** Interfaccia responsabile per la creazione di entità;
- **Relazione con altre classi:**
  - \* com.apim.server.entities.EntityFactory;

- **EntityFactory**

- **Descrizione:** Implementazione di IEntityFactory;
- **Relazione con altre classi:**
  - \* com.apim.server.entities.User;
  - \* com.apim.server.entities.API;
  - \* com.apim.server.entities.Key;
  - \* com.apim.server.entities.Comment;
  - \* com.apim.server.entities.Acquisto;
  - \* com.apim.server.entities.Categoria;

- **User**

- **Descrizione:** Questa classe rappresenta l'entità User del database;
- **Relazione con altre classi:**
  - \* com.apim.server.entities.IEntity;

- **API**

- **Descrizione:** Questa classe rappresenta l'entità API del database;
- **Relazione con altre classi:**
  - \* com.apim.server.entities.IEntity;

- **Key**

- **Descrizione:** Questa classe rappresenta l'entità Key del database;
- **Relazione con altre classi:**
  - \* com.apim.server.entities IEntity;

- **Comment**

- **Descrizione:** Questa classe rappresenta l'entità Comment del database;
- **Relazione con altre classi:**
  - \* com.apim.server.entities IEntity;

- **Acquisto**

- **Descrizione:** Questa classe rappresenta l'entità Acquisto del database;
- **Relazione con altre classi:**
  - \* com.apim.server.entities IEntity;

- **Categoria**

- **Descrizione:** Questa classe rappresenta l'entità Categoria del database;
- **Relazione con altre classi:**
  - \* com.apim.server.entities IEntity;

### 6.12.3 Interazioni con altri componenti

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

## 7 Architettura Database

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

### 7.1 Progettazione concettuale

Nella progettazione concettuale viene modellata la realtà da rappresentare. Sono state individuate le classi e le relazioni tra di esse, definendo così la struttura che avrà il database.

#### 7.1.1 Schema concettuale

##### 7.1.2 Classi

###### *User*

Questa è l'entità che rappresenta tutti gli utenti che utilizzeranno il software, siano essi utenti normali o amministratori. Questa entità come si vede dallo schema è superclasse 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
Bio	String	null
SaldoCoin	Double	null
IsAdmin	Bool	null

###### *Utenti*

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

###### *API*

Questa entità 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'utilizzo di quel microservizio, il numero **Totale Chiamate** e il **Tempo Risposta Totale**.

#### *Key*

Questa entità 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 può essere utilizzata. Secondo la policy di acquisto di un microservizio, al raggiungimento di tale limite la chiave scadrà e non permetterà più l'utilizzo di tale microservizio. *Commenti*

Questa entità 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 sarà il giudizio che l'utente lascerà del microservizio utilizzato.

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

Attributo	Tipo	Vincolo
IdCategoria	Int	PRIMARY KEY
Categoria	String	null

*Acquisti* Questa entità contiene lo storico degli acquisti effettuati dagli utenti.

Attributo	Tipo	Vincolo
IdAcquisto	Int	PRIMARY KEY



### 7.1.3 Associazioni

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

## 7.2 Progettazione logico-relazionale

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

### 7.2.1 Schema logico-relazionale

#### 7.2.2 Gerarchie

**User - Utenti - Admin:** La gerarchia é 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.

### 7.2.3 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
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:

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

In particolare, la **DataScadenza** indica la data fino alla quale é stato acquistato il microservizio; similmente **MaxByte** e **TempoUso** stanno ad indicare rispettivamente il traffico massimo ed il tempo massimo con i quali si può 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

**Crea** Questa tabella viene a crearsi dalla relazione tra Users e API. Rappresenta il caricamento nel sistema di un microservizio da parte di un utente. Gli attributi sono:

Attributo	Tipo	Vincolo
IdMS	Int	PRIMARY KEY
NomeAPI	Varchar	FOREIGN KEY
Username	Varchar	FOREIGN KEY

## 8 Design Pattern

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

- **Pattern Architettureali:** definiscono l'architettura dell'applicazione ad un livello più 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 relazioni 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 più comuni interazioni tra oggetti.

### 8.1 Design Pattern Architettureali

#### 8.1.1 Microservizi

- **Scopo dell'utilizzo:** Questo pattern è orientato ai microservizi, cioè permette di implementare unità separate distribuite;
- **Contesto d'utilizzo:**

#### 8.1.2 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. Abbiamo scelto il framework AngularJS, 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 dalla sua rappresentazione grafica.

#### 8.1.3 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 queste 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.

## 8.2 Design Pattern Creazionali

### 8.2.1 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.

### 8.2.2 Dependency Injection

- **Scopo dell'utilizzo:** Permette di separare il comportamento di una componente dalla risoluzione delle sue dipendenze;
- **Contesto dell'utilizzo:**

## 8.3 Design Pattern Strutturali

### 8.3.1 Facade

- **Scopo dell'utilizzo:** Fornisce un'interfaccia unica semplice per un sottosistema complesso, strutturando il sistema in sottoinsiemi;
- **Contesto dell'utilizzo:**

## 8.4 Design Pattern Comportamentali

### 8.4.1 Command

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

## 9 Titolo

Testo:

- lista;
- lista.

### 9.1 SottoTitolo

Testo.

### 9.2 SottoTitolo

Testo.

Testo.

Testo.

Testo:

- lista; con:
  - sottolista
  - **testogrossetto**: da da da;
  - *Testo corsivo*.  
Ancora testo.
  - testo.
- Continua la lista di prima.

### 9.3 Sottotitolo

Testo.

#### 9.3.1 SottoSottoTitolo

Testo `http://www.url.com/`.

# 10 Titolo

Testo:

- lista;
- lista.

## 10.1 SottoTitolo

Testo.

## 10.2 SottoTitolo

Testo.

Testo.

Testo.

Testo:

- lista; con:
  - sottolista
  - **testogrossetto**: da da da;
  - *Testo corsivo*.  
Ancora testo.
  - testo.
- Continua la lista di prima.

## 10.3 Sottotitolo

Testo.

### 10.3.1 SottoSottoTitolo

Testo <http://www.url.com/>.

## 11 Descrizione Design Pattern

### 11.1 Design Pattern Architetture

#### 11.1.1 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.

- **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<sub>G</sub>*). 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.

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** :



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

### 11.1.2 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.

- **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.

- **Considerazioni** : Le Data Access Objects si fanno carico di gestire il codice SQL, mentre tutto ciò è 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** :

Caratteristica	Valutazione	Descrizione
<b>Agilità</b>	-/+	Semplice da gestire ma l'obbligo di innescare query multiple lo rende meno snello.
<b>Implementazione</b>	+	Facile l'implementazione perché è relativamente semplice e rigoroso separare due parti dell'applicazione.
<b>Testabilità</b>	+	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.
<b>Performance</b>	-	Vi è un costo aggiuntivo ad ogni chiamata al server e aumento di complessità.
<b>Scalabilità</b>	+	Eventuali modifiche al DB possono essere implementate da sole , senza che il resto dell'applicazione è interessata.
<b>Sviluppo</b>	-	Obbliga gli sviluppatori a innescare query multiple sul database che potrebbero essere recuperate con una unica.

### 11.1.3 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.

- **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.

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 :**

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

## 11.2 Design Pattern Creazionali

### 11.2.1 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.

- **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 :**

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

### 11.2.2 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.

- 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.
- 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<sub>g</sub> 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 :**

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

## 11.3 Design Pattern Strutturali

### 11.4 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.

- **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** :

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.

## 11.5 Design Pattern Comportamentali

### 11.5.1 Command

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

- In che modo un oggetto svolge la sua funzione?
- In che modo diversi oggetti collaborano tra loro?

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

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.

## 12 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 ms 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 della 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 obiettivi. 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.

### 12.1 Home

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

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. Una differenziazione intelligente faciliterà la progettazione del API 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 microservizio. Perché vogliamo far sì 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 **grassetto**. E 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.

## 12.2 Profilo

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

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 distinguono 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.

## 12.3 Ricerca

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

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 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.



(a) Titolo Figura 1



(b) Titolo Figura 2



(c) Titolo Figura 3



Figura 3: Testo figura

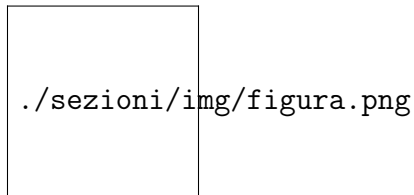


Figura 4: Testto figura

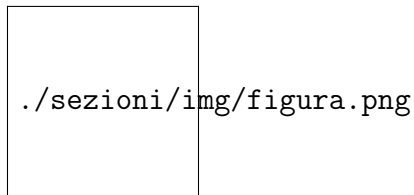


Figura 5: Testto figura

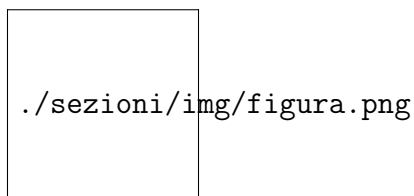


Figura 6: Testto figura

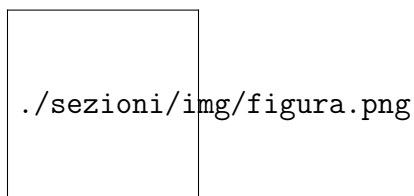


Figura 7: Testto figura

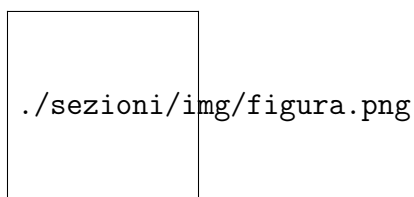


Figura 8: Testto figura

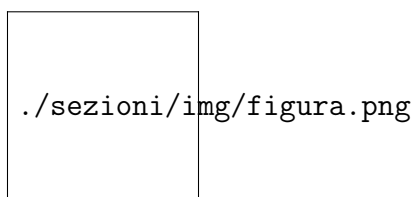


Figura 9: Testto figura

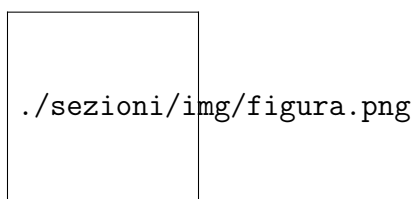


Figura 10: Testito figura

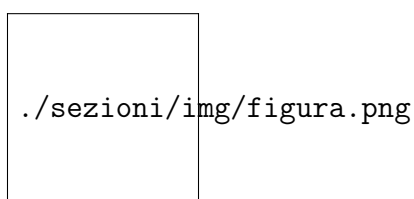


Figura 11: Testito figura

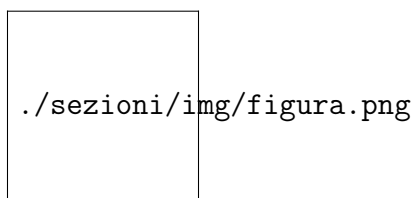


Figura 12: Testito figura

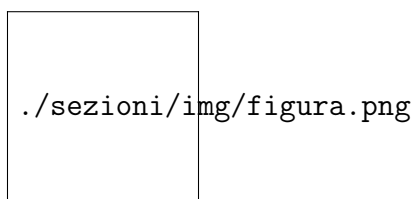


Figura 13: Testito figura

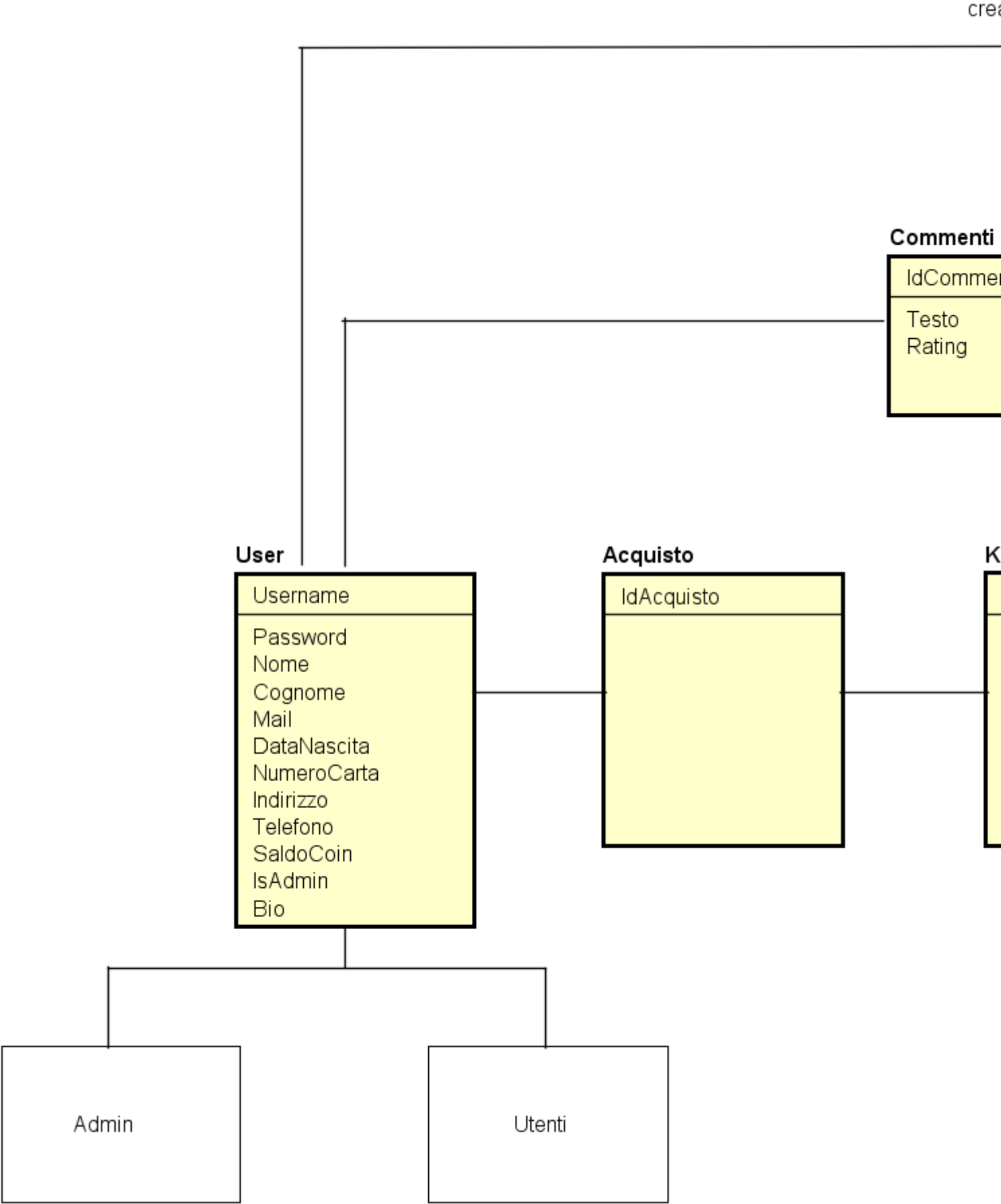
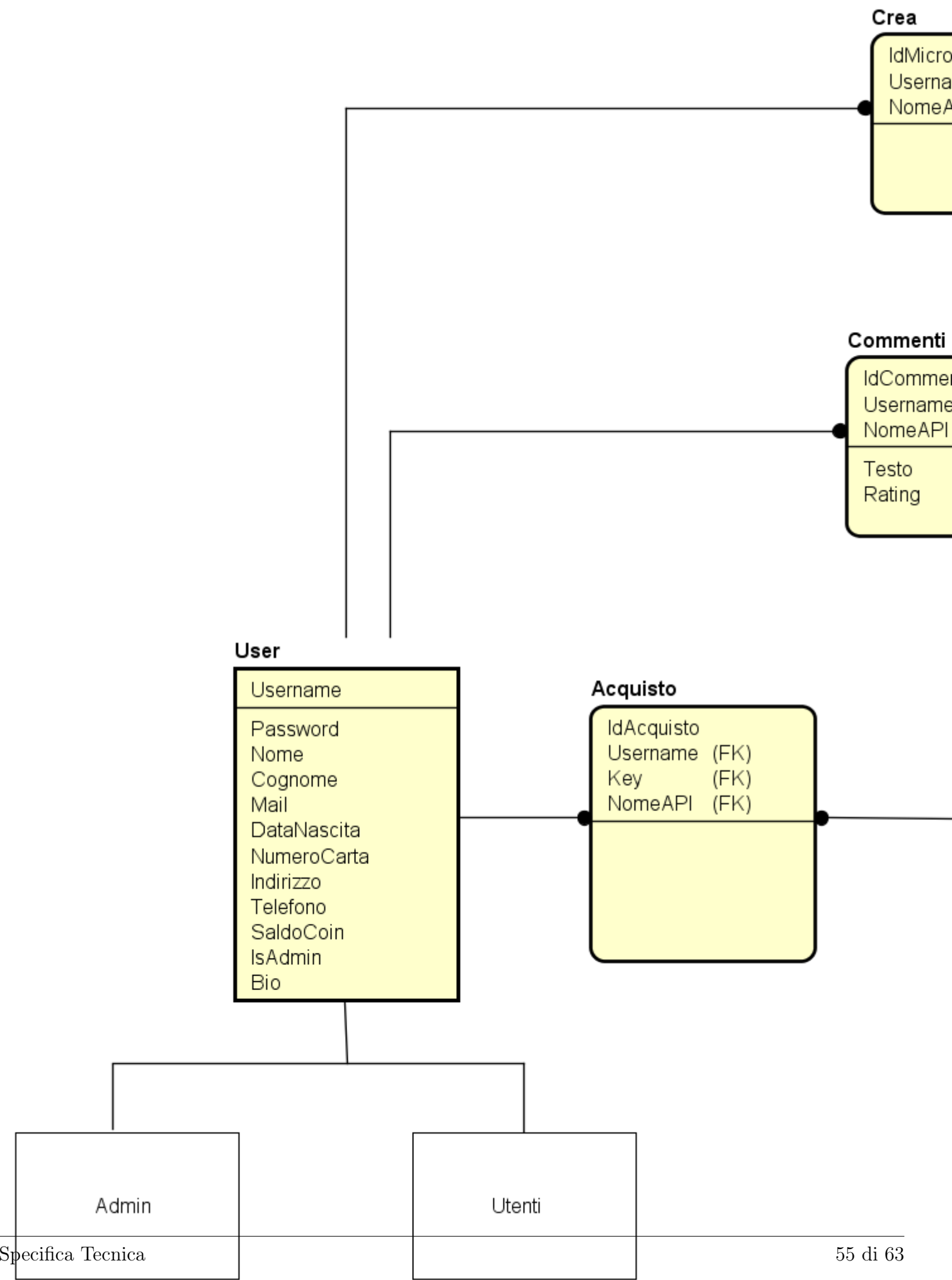


Figura 14: Progettazione Concettuale



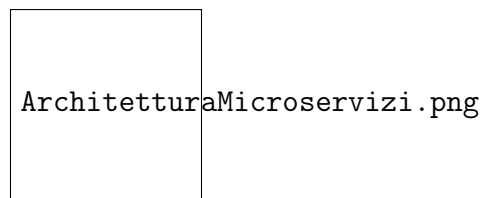


Figura 16: Architettura Microservizi

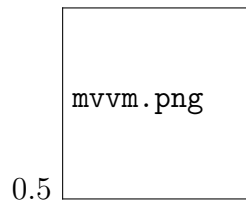


Figura 17: mvvm

0.5

Figura 18: DAO

0.5 abstractfactory.png

Figura 19: Abstract Factory

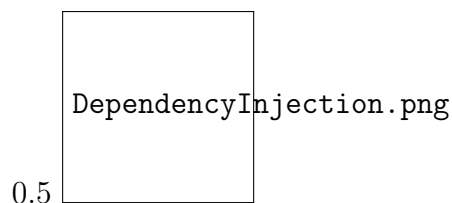


Figura 20: Dependency Injection

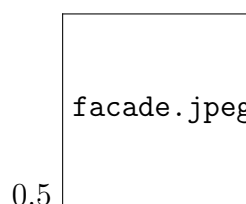


Figura 21: Facade

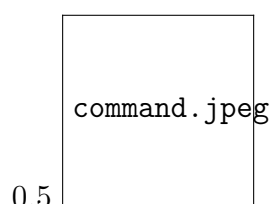
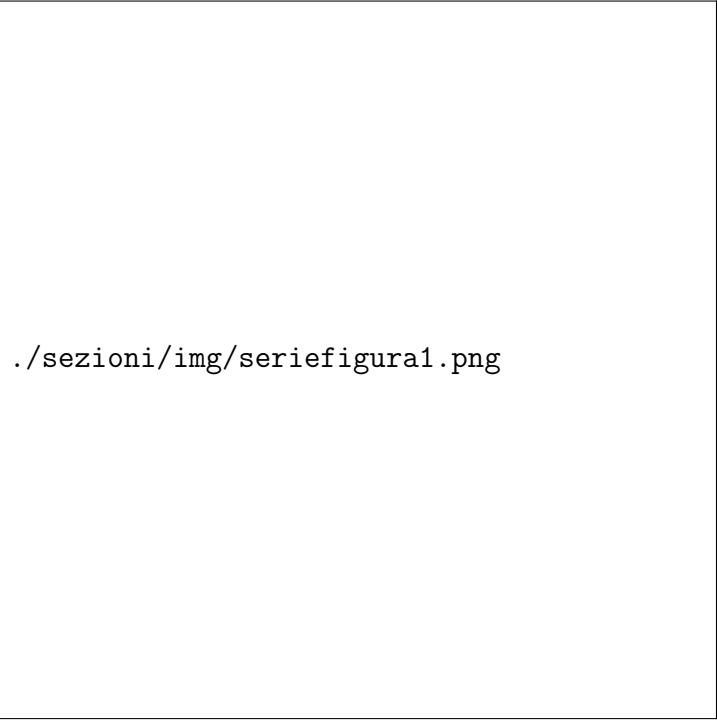


Figura 22: Command





(a) Titolo Figura 1



(b) Titolo Figura 2



(c) Titolo Figura 3



Figura 24: Testo figura



(a) Titolo Figura 1

(b) Titolo Figura 2



(c) Titolo Figura 3



Figura 26: Testo figura

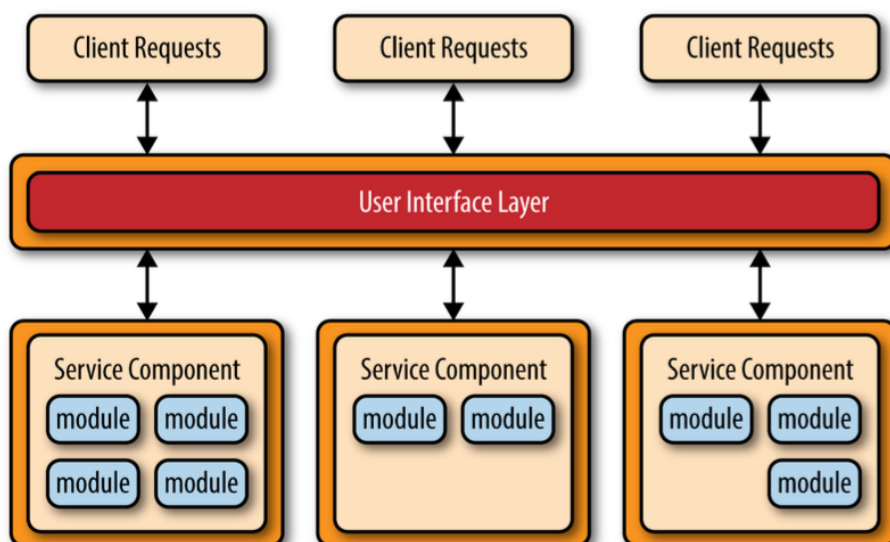


Figura 27: Illustrazione Architettura a Microservizi

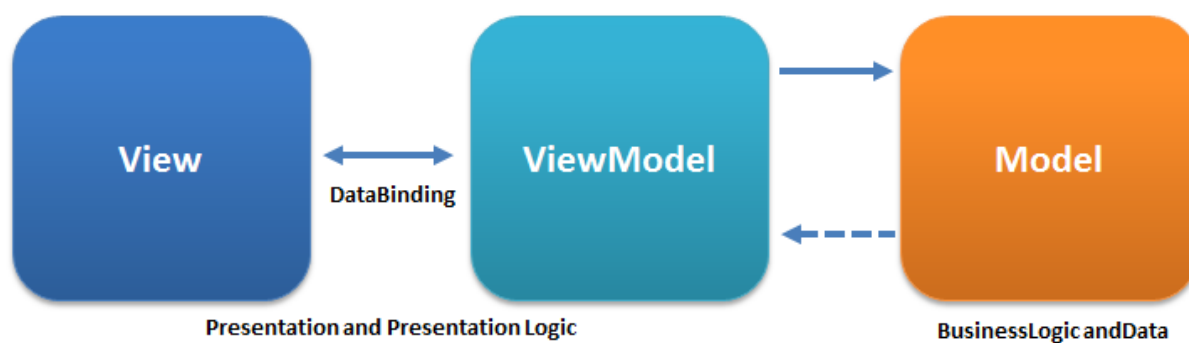


Figura 28: Illustrazione Model View ViewModel

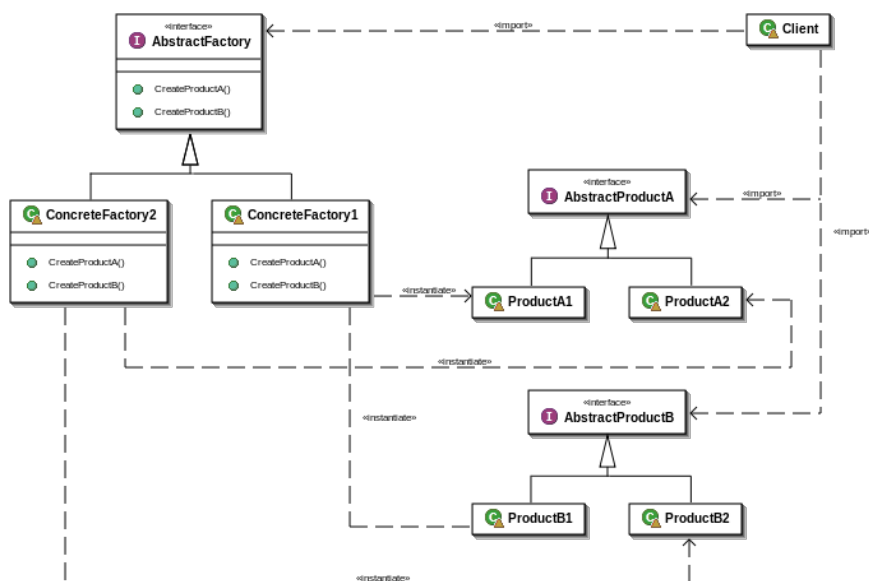


Figura 29: Illustrazione Pattern Abstract Factory

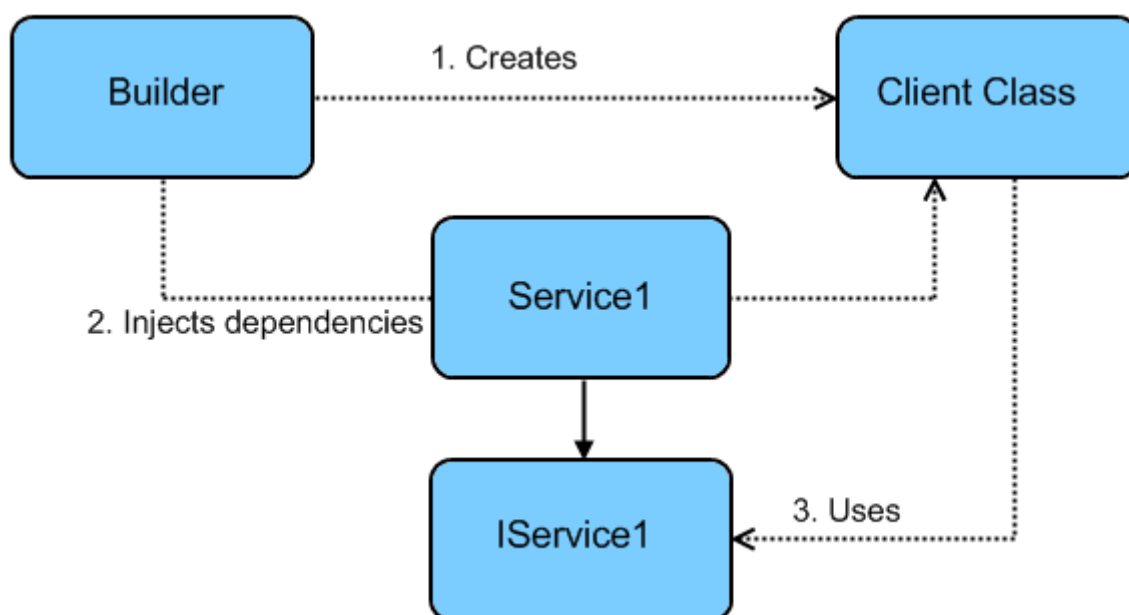


Figura 30: Illustrazione Pattern Dependency Injection

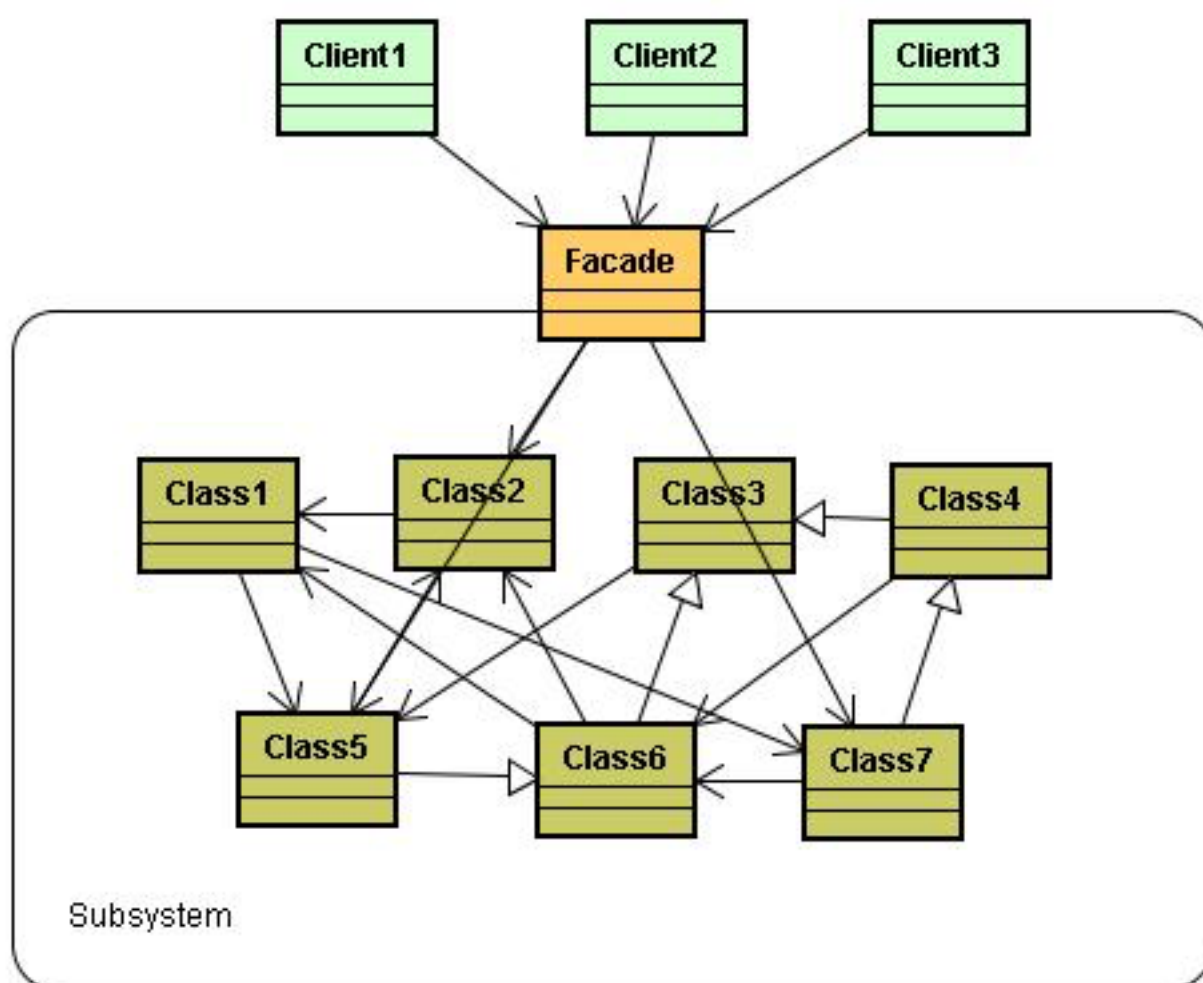


Figura 31: Illustrazione Pattern Facade

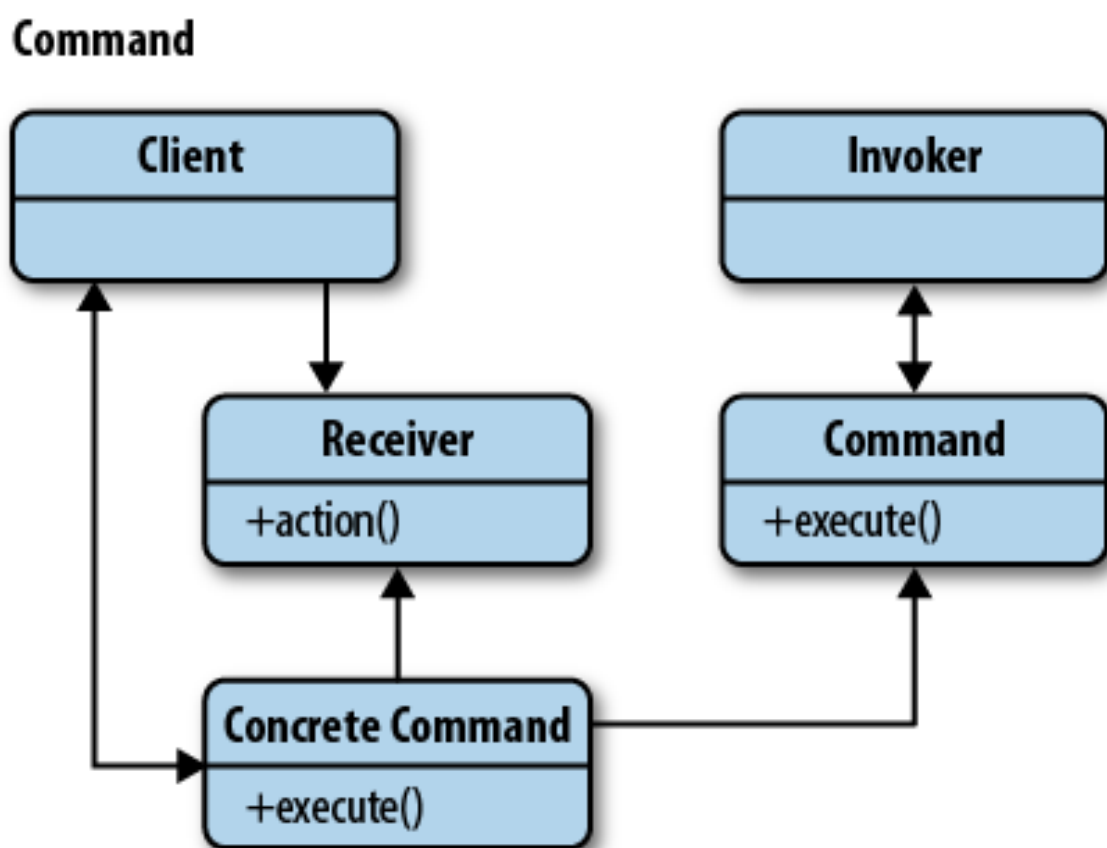


Figura 32: Illustrazione Pattern Command

<

>



Home



Categorie



Profilo




Ca

# API MARKET

Informazioni generiche del sito e sua descrzione

Più altre informazioni continue come: Caron dimonio, con occhi di bragia, loro accennando qualunque s'adagia. Come d'autunno si levan le foglie l'una appresso de l'a vede a la terra tutte le sue spoglie,similmente il mal seme d'Ac gittansi di quel lito ad una ad una, per cenni come augel per suo r Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura, ché la di Ahi quanto a dir qual era è cosa dura esta selva selvaggia e aspra che nel pensier rinnova la paura!


Microservice On Top



Nome Microservizio 1

Utility


€



Nome AB

Grafica e design

€





Microservizio Nome 2

Produttività

€




Name of Microservis...

Produttività

€

Acquisti In-App






MicroNome

Produttività

★★★★★

24 valutazi...

€



Nomignolo

Produttività

★★★★★

5 valutazioni

OTTIENI

Acquisti In-App



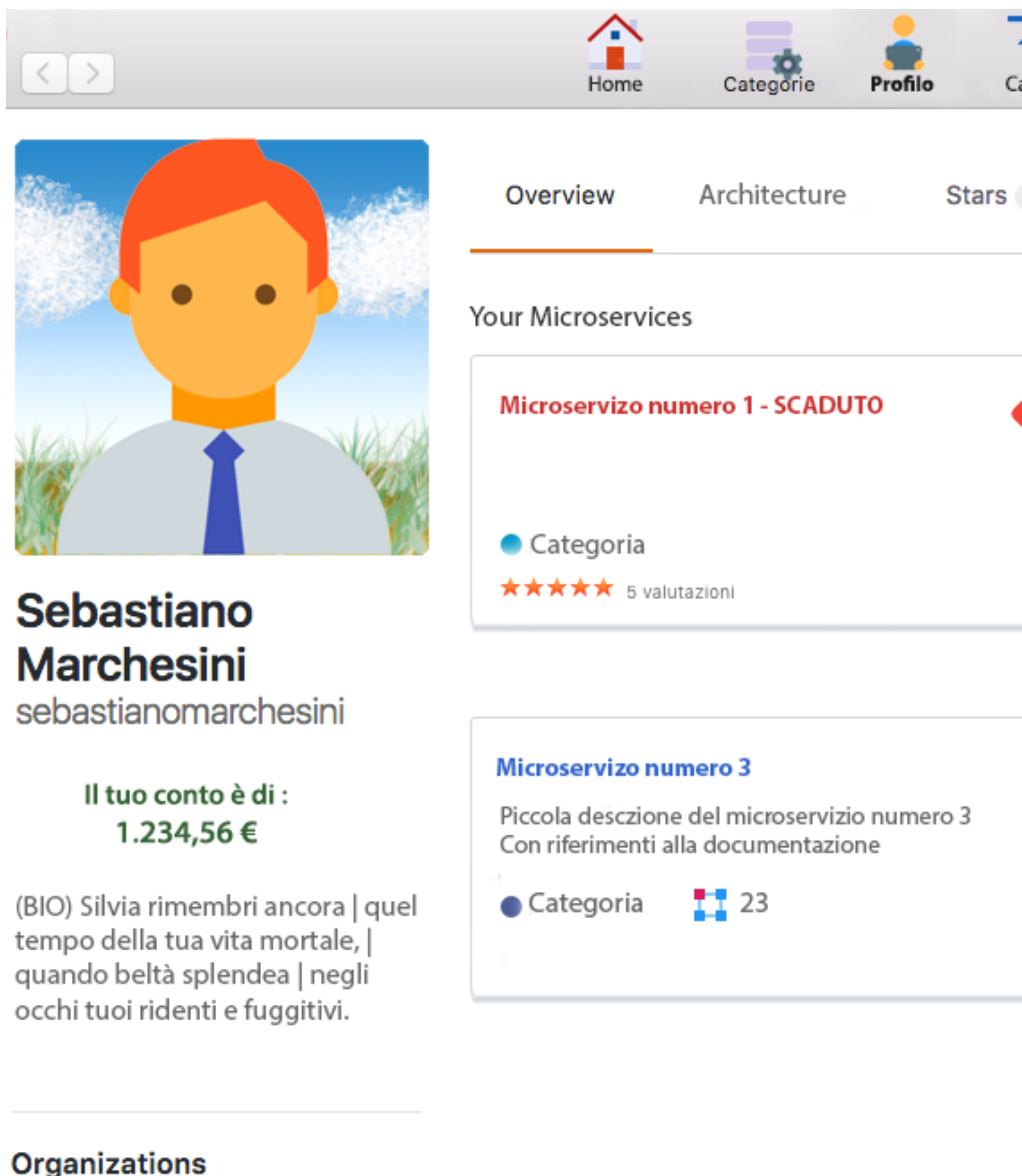







Figura 34: Idea di visione della Pagina profilo

<>

  
Home

Search

microservice

	Microservices	15,391
	Documentations	234,522
	Comments	31,807
	Connections	9,551
	Users	2,923

We've found

Microservices

Sample of a Microservices  
Java / Docker

● Categoria

Categories

Categoria X	5,590
Categoria Y	2,609
Categoria Z	1,040
Categoria J	697
Ecc...	639

Microservices

Payments microservices

● Categoria

Advanced search

Microservices

Spring Cloud

● Categoria

Microservices

This library provides a  
configurator,