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

Elenco delle tabelle

Elenco delle figure

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 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.

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.
- **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.

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$_g$ 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$_g$;
- 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 normalemente 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$_g$: 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$_g$ 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$_g$ 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 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$_g$;
- 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 solo 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$_g$.

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$_g$.
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 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 ha chiesto esplicitamente che venisse utillizzato all'intrno 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.

4.2 Architettura generale

Testo.
Testo.
Testo.
Testo:

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

4.3 Sottotitolo

Testo.

4.3.1 SottoSottoTitolo

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

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 Titolo

Testo:

- lista;
- lista.

6.1 SottoTitolo

Testo.

6.2 SottoTitolo

Testo.
Testo.
Testo.
Testo:

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

6.3 Sottotitolo

Testo.

6.3.1 SottoSottoTitolo

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

7 Titolo

Testo:

- lista;
- lista.

7.1 SottoTitolo

Testo.

7.2 SottoTitolo

Testo.
Testo.
Testo.
Testo:

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

7.3 Sottotitolo

Testo.

7.3.1 SottoSottoTitolo

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

8 Titolo

Testo:

- lista;
- lista.

8.1 SottoTitolo

Testo.

8.2 SottoTitolo

Testo.
Testo.
Testo.
Testo:

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

8.3 Sottotitolo

Testo.

8.3.1 SottoSottoTitolo

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

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 Architetturali

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_{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.
  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
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

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 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** :

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.

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$_g$.
  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** :

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.

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
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.

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$_g$ 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** :

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.

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"$_g$. 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$_g$) 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 Titolo

Testo:

- lista;
- lista.

12.1 SottoTitolo

Testo.

12.2 SottoTitolo

Testo.
Testo.
Testo.
Testo:

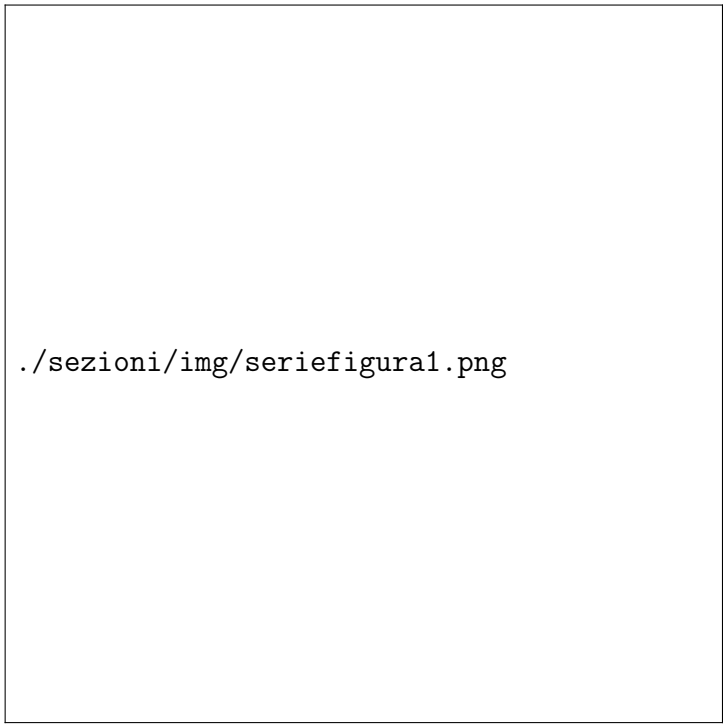
- lista; con:
  - sottolista
  - **testogrossetto**: da da da;
  - *Testo corsivo.*
    Ancora testo.
  - testo.
- Continua la lista di prima.

12.3 Sottotitolo

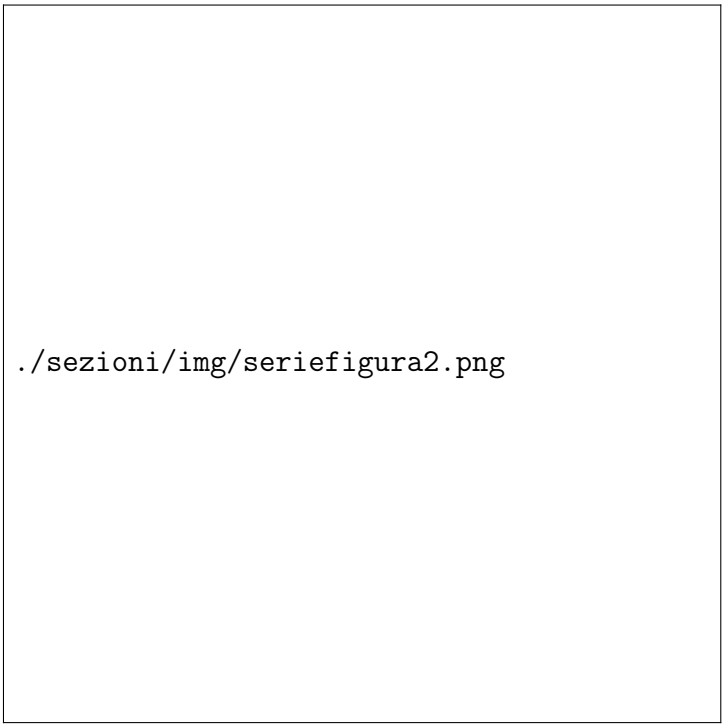
Testo.

12.3.1 SottoSottoTitolo

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



(a) Titolo Figura 1



(b) Titolo Figura 2

(c) Titolo Figura 3

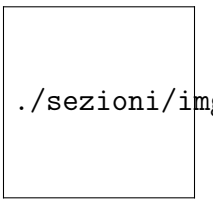
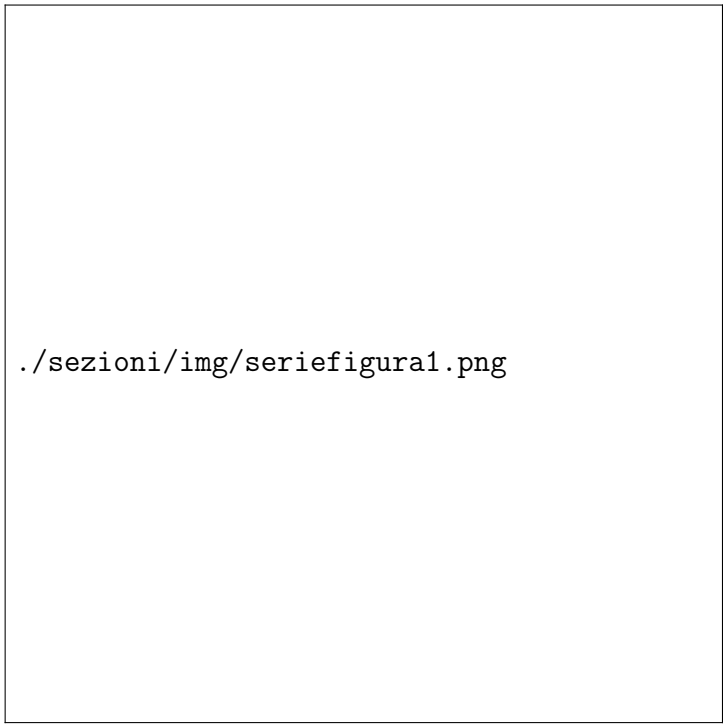
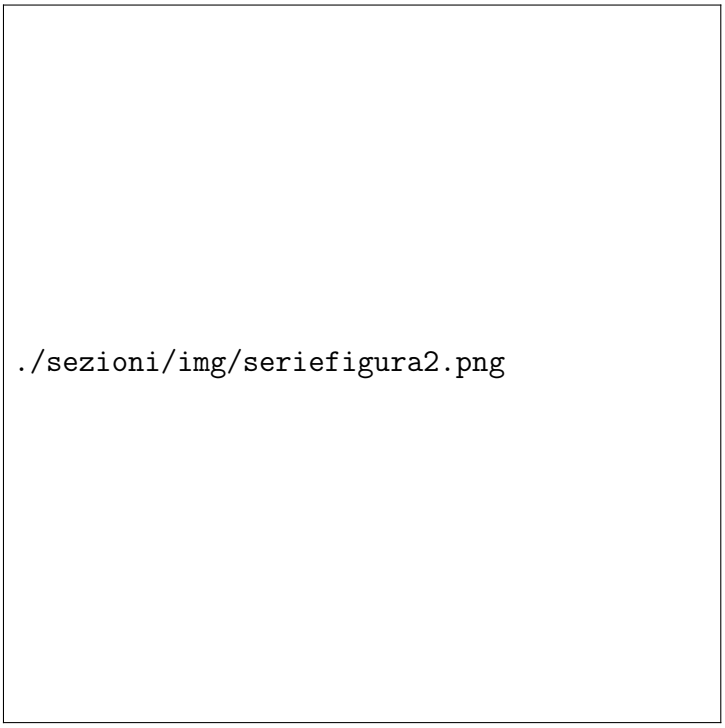


Figura 2: Testo figura



(a) Titolo Figura 1



(b) Titolo Figura 2

(c) Titolo Figura 3

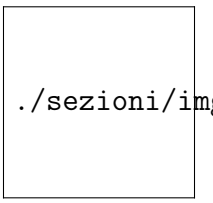
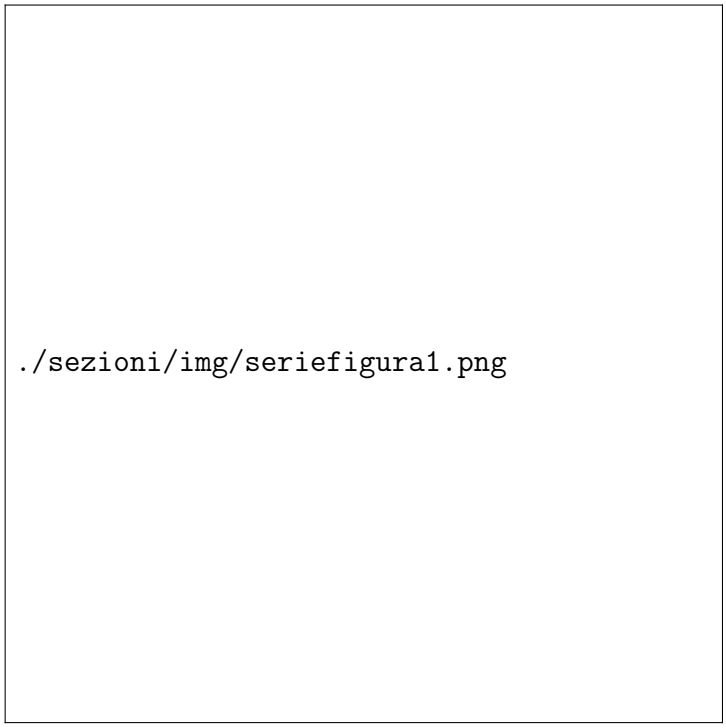
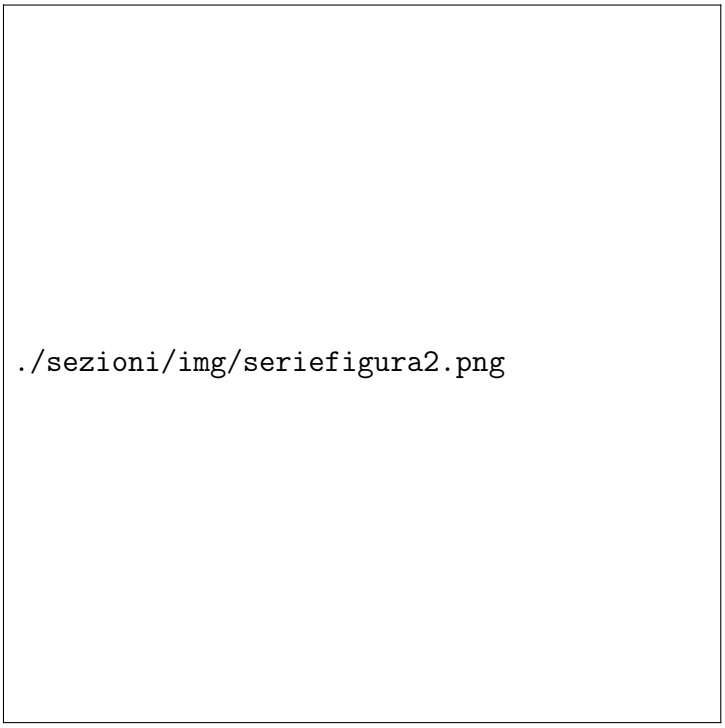


Figura 4: Testo figura



(a) Titolo Figura 1



(b) Titolo Figura 2

(c) Titolo Figura 3

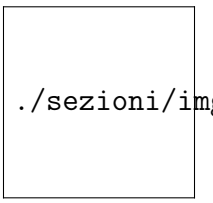
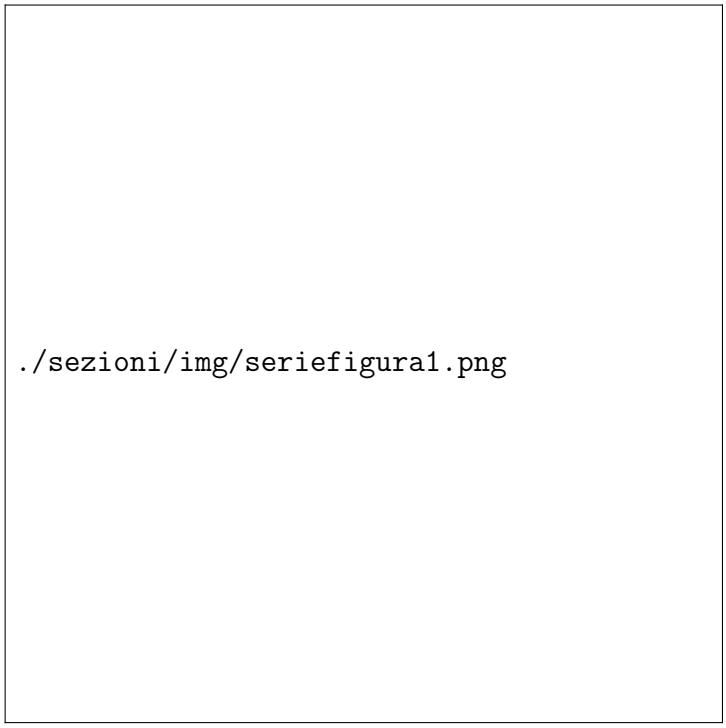
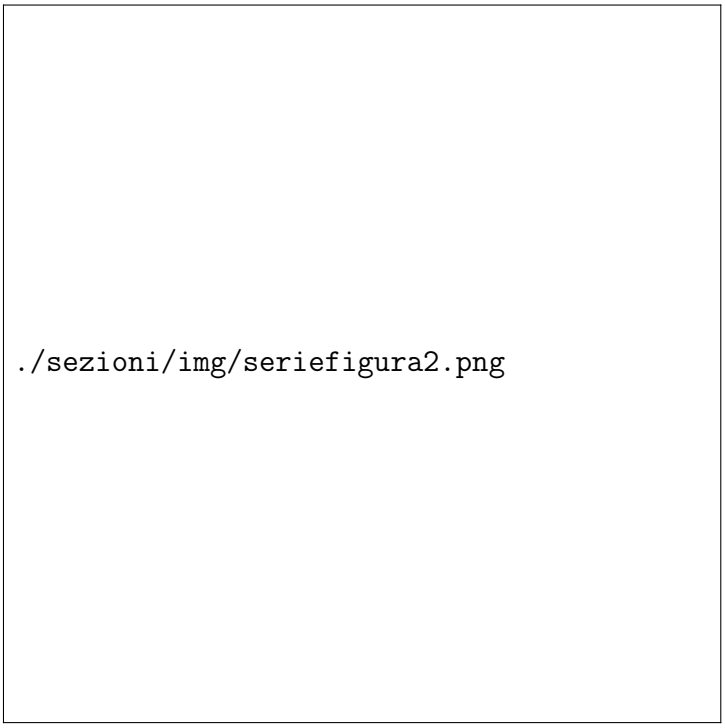


Figura 6: Testo figura



(a) Titolo Figura 1



(b) Titolo Figura 2

(c) Titolo Figura 3

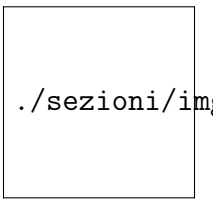
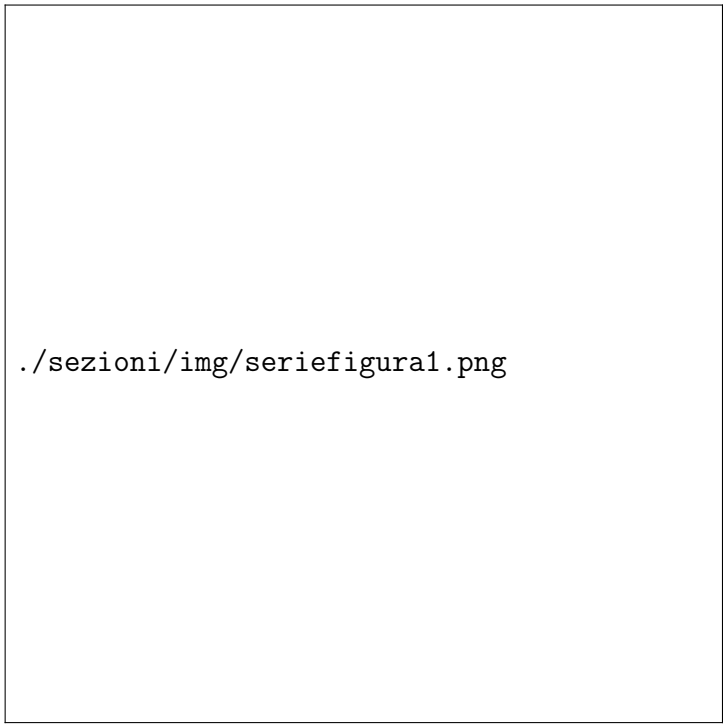
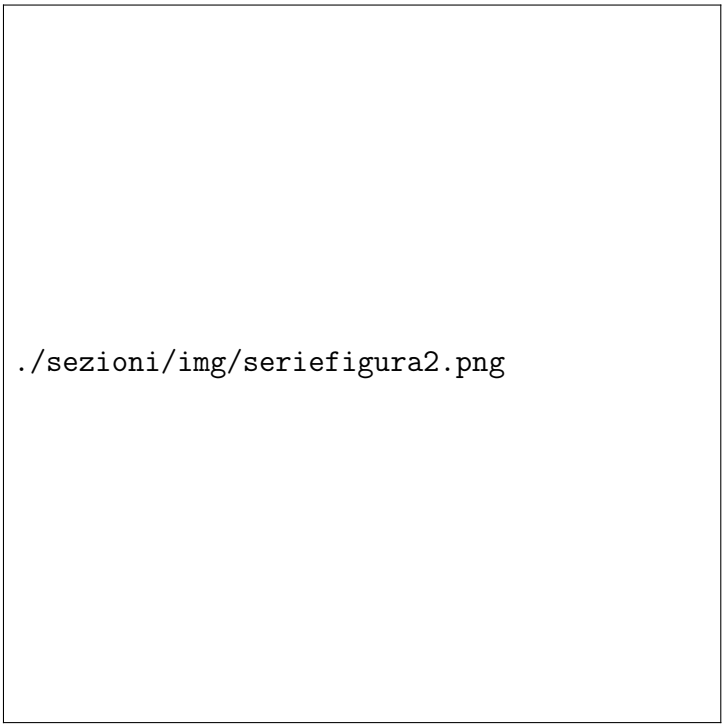


Figura 8: Testo figura



(a) Titolo Figura 1



(b) Titolo Figura 2

(c) Titolo Figura 3

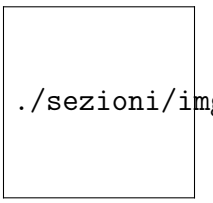
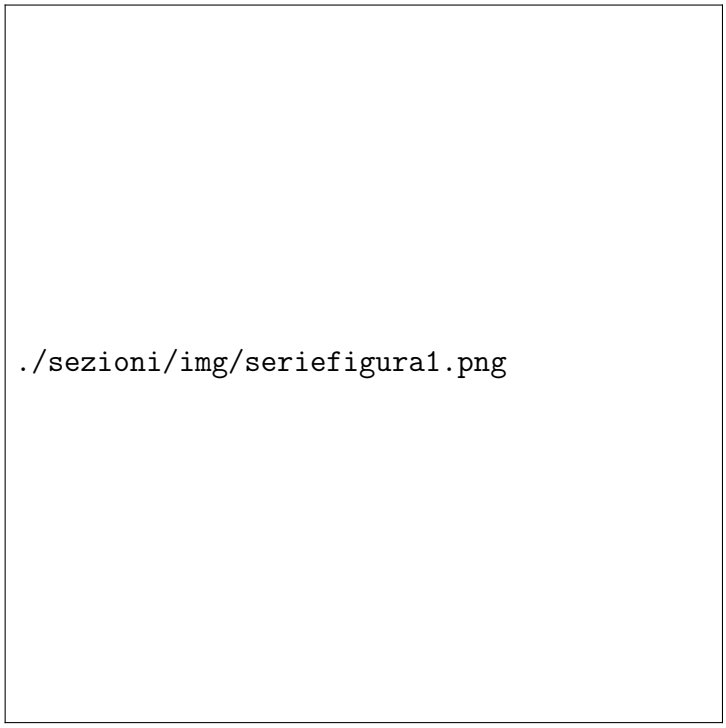
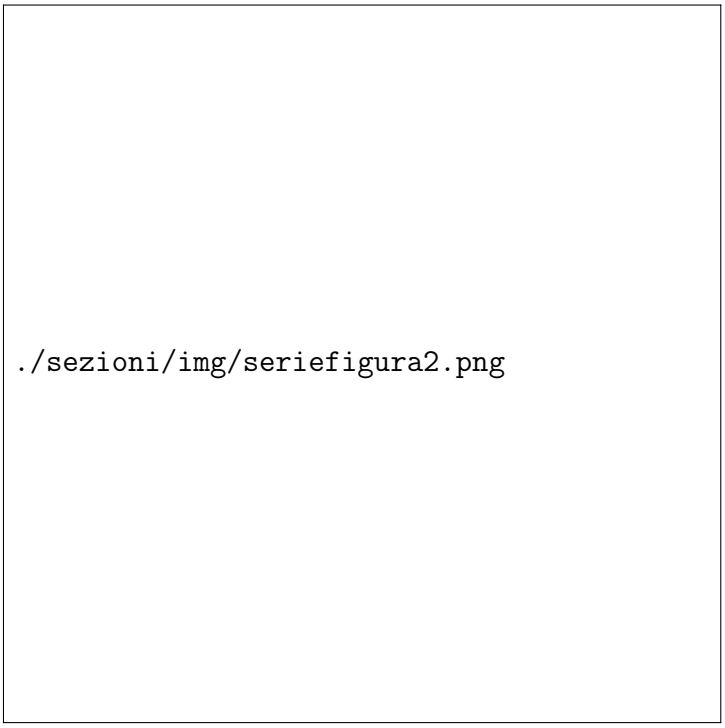


Figura 10: Testo figura



(a) Titolo Figura 1



(b) Titolo Figura 2

(c) Titolo Figura 3

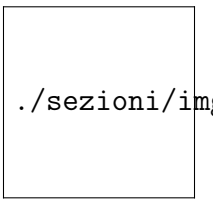
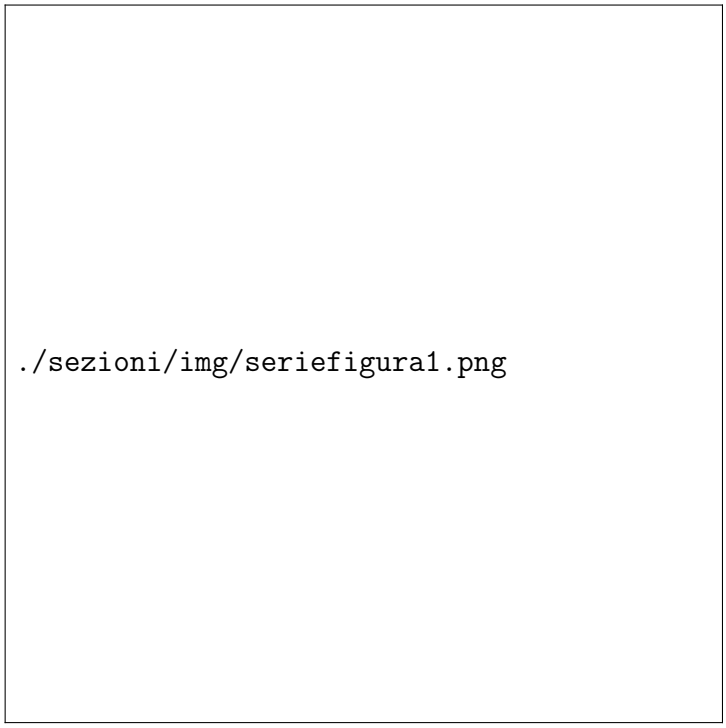
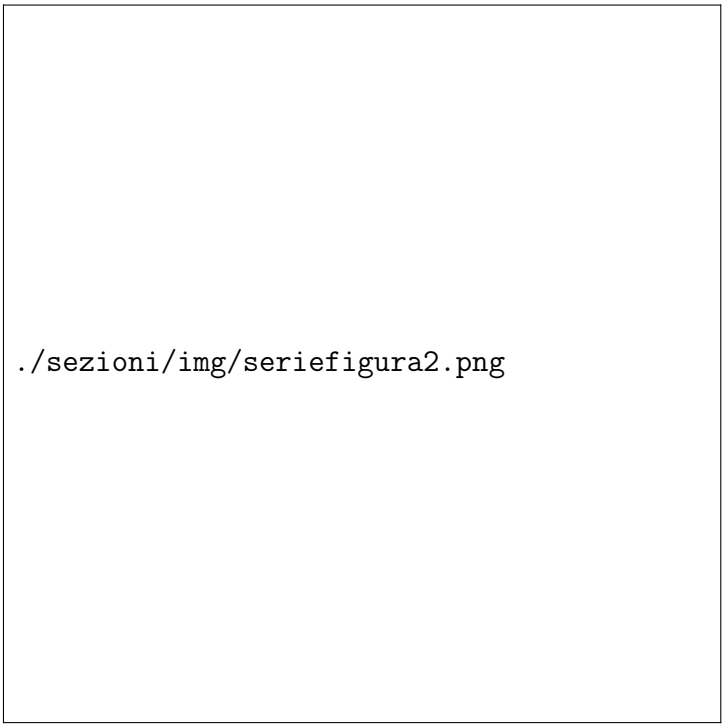


Figura 12: Testo figura



(a) Titolo Figura 1



(b) Titolo Figura 2

(c) Titolo Figura 3

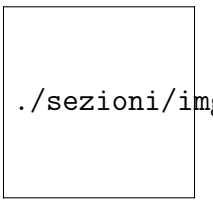


Figura 14: Testo figura

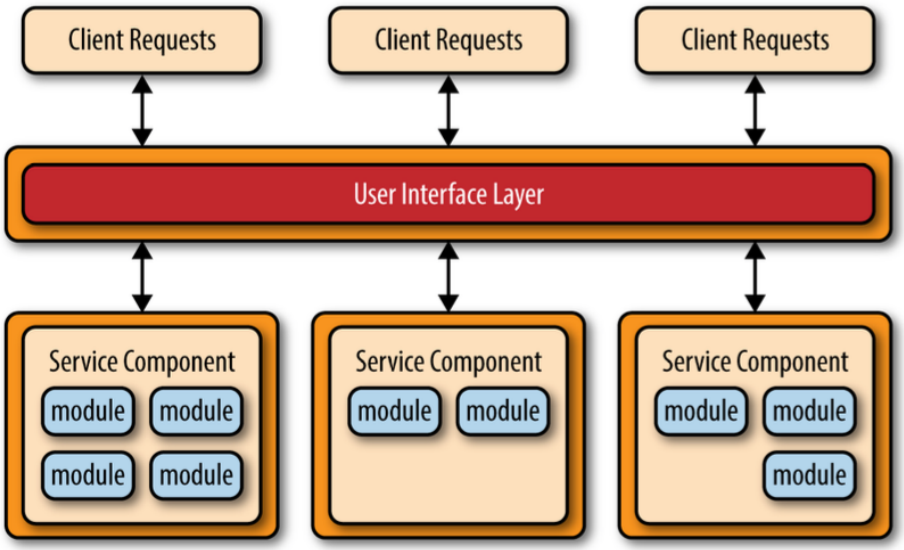


Figura 15: Illustrazione Architettura a Microservizi

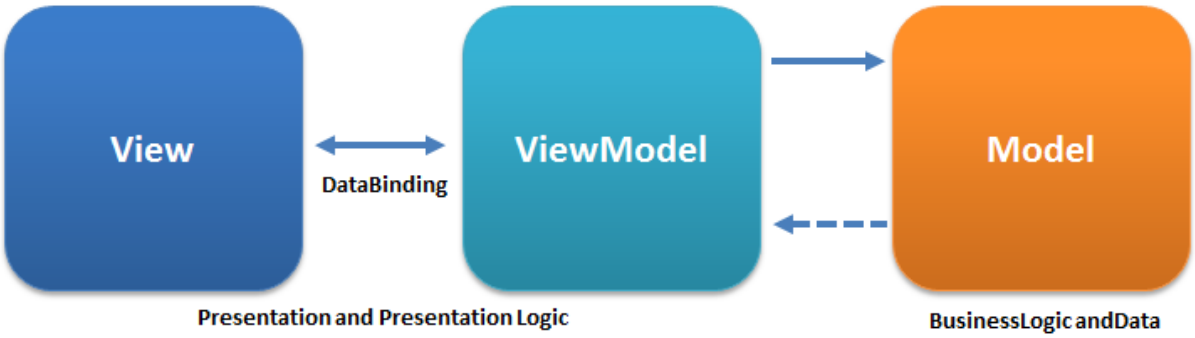


Figura 16: Illustrazione Model View ViewModel

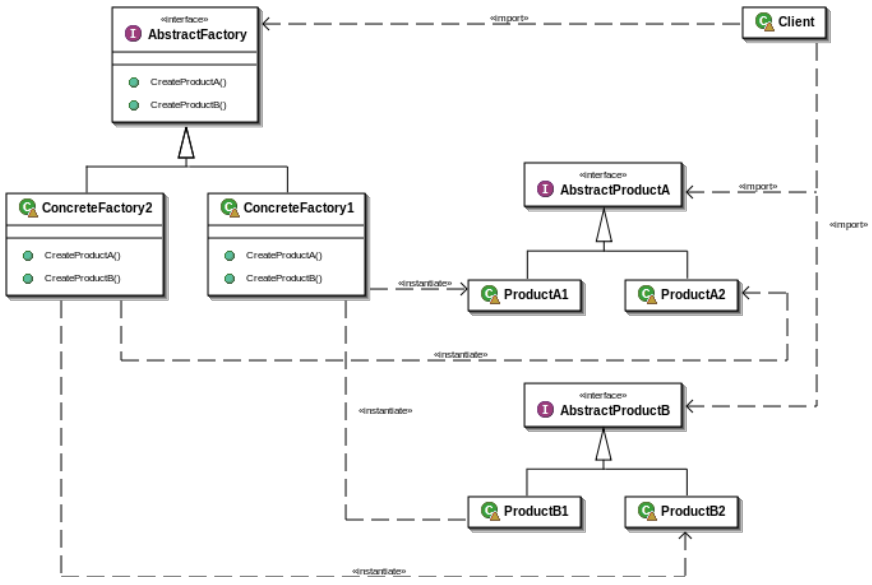


Figura 17: Illustrazione Pattern Abstract Factory

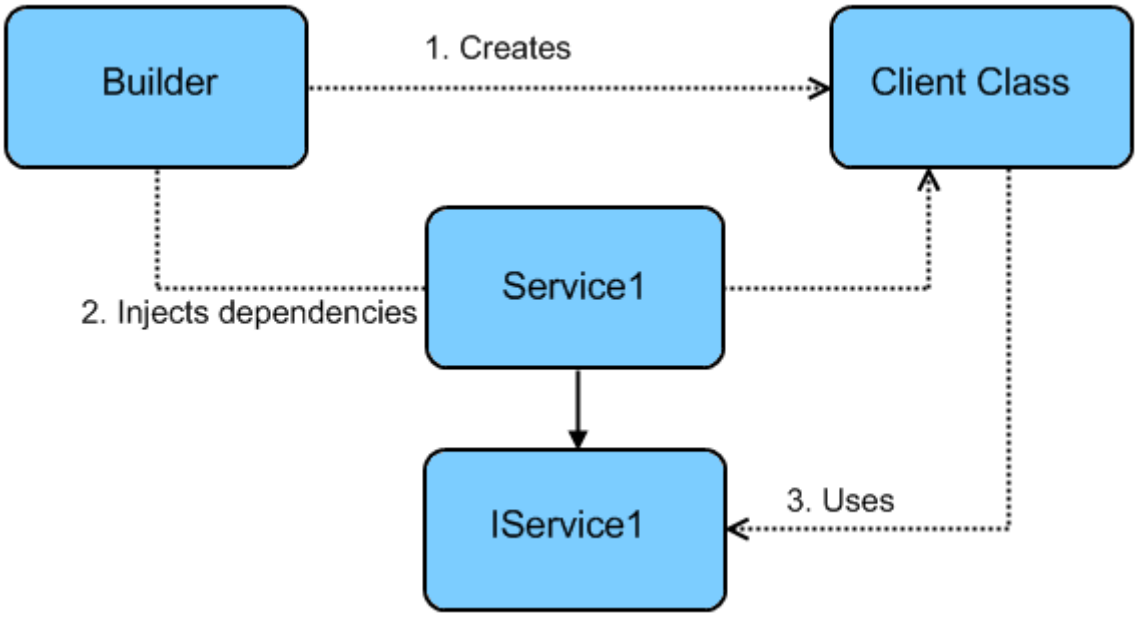


Figura 18: Illustrazione Pattern Dependency Injection

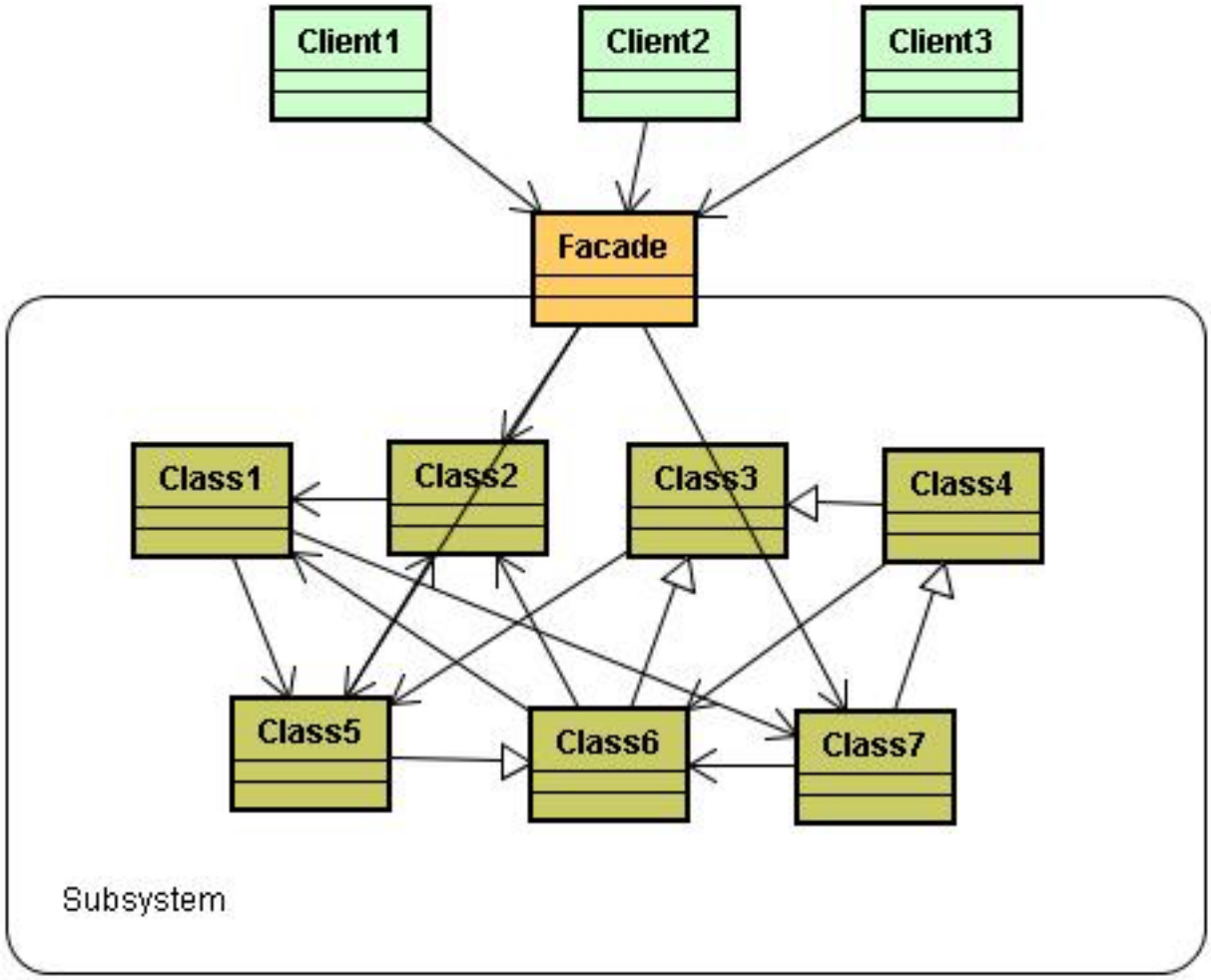


Figura 19: Illustrazione Pattern Facade

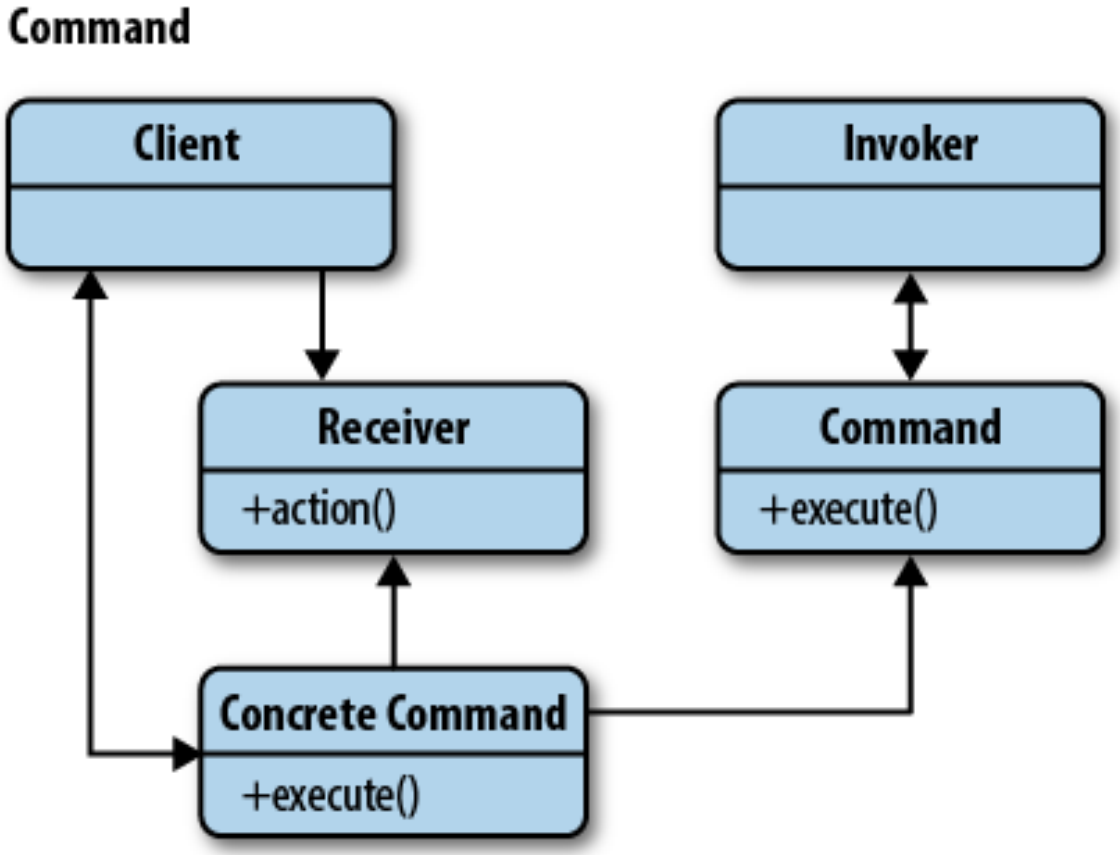


Figura 20: Illustrazione Pattern Command