



Specifica Tecnica

Gruppo SWEet BIT – Progetto SWEDesigner

Informazioni sul documento

Versione	1.0.0
Redazione	Santimaria Davide Massignan Fabio
Verifica	Massignan Fabio Bodian Malick
Approvazione	Pilò Salvatore
Uso	Esterno
Distribuzione	Prof. Tullio Vardanega Prof. Riccardo Cardin Gruppo SWEet BIT Zucchetti S.p.A.

Descrizione

Questo documento descrive la specifica tecnica e l'architettura del prodotto sviluppato dal gruppo SWEet BIT per la realizzazione del progetto SWEDesigner.

Versioni del documento

Versione	Data	Persone coinvolte	Descrizione
1.y.z	2017/??/??	Pilò Salvatore	Approvazione Documento
1.y.z	2017/??/??	Massignan Fabio	Verifica Documento
1.0.3	2017/05/02	NOME	Stesura sezione Descrizione architettura
1.0.2	2017/05/02	NOME	Stesura sezione Tecnologie utilizzate
1.0.1	2017/05/02	NOME	Stesura sezione Introduzione
1.0.0	2017/05/02	Santimaria Davide	Creazione struttura documento

Indice

1	Introduzione	6
1.1	Scopo del documento	6
1.2	Scopo del prodotto	6
1.3	Glossario	6
1.4	Riferimenti	6
1.4.1	Normativi	6
1.4.2	Informativi	7
2	Tecnologie utilizzate	8
2.1	Server	8
2.1.1	Node.js	8
2.1.1.1	Vantaggi	8
2.1.1.2	Svantaggi	8
2.1.2	Expressjs	9
2.1.2.1	Vantaggi	9
2.1.2.2	Svantaggi	9
2.1.3	MongoDB	9
2.1.3.1	Vantaggi	9
2.1.3.2	Svantaggi	10
2.1.4	Mongoose	10
2.1.4.1	Vantaggi	10
2.1.4.2	Svantaggi	11
2.1.5	Grunt	11
2.2	Librerie	11
2.2.1	Mustache	11
2.2.2	Passport:	11
2.2.3	BodyParser:	11
2.2.4	Forge:	12
2.2.5	PassportJWT:	12
2.2.6	Bcrypt:	12
2.3	Client	12
2.3.1	Angular 4.0	12
2.3.1.1	Vantaggi	12
2.3.1.2	Svantaggi	13
2.3.2	Draw2D	13
2.3.2.1	Vantaggi	13
2.3.2.2	Svantaggi	13
2.3.3	HTML5	13
2.3.3.1	Vantaggi	14
2.3.3.2	Svantaggi	14
2.3.4	CSS3	14

2.3.4.1	Vantaggi	14
3	Descrizione architettura	15
3.1	Metodo e formalismo di specifica	15
3.2	Architettura generale	15
3.3	Interfaccia REST-like	16
3.4	Architettura del Server	16
3.5	Architettura del Client	17
4	Componenti del <i>Back-end</i>	18
4.1	Descrizione packages e classi	18
4.1.1	SWEDesigner::Server	18
4.1.1.1	Informazioni sul Package	18
4.1.1.2	Informazioni sulle Classi	19
4.1.2	SWEDesigner::Server::Controller	19
4.1.2.1	Informazioni sul Package	20
4.1.3	SWEDesigner::Server::Controller::Middleware	20
4.1.3.1	Informazioni sul Package	20
4.1.3.2	Informazioni sulle Classi	20
4.1.4	SWEDesigner::Server::Controller::Services	22
4.1.4.1	Informazioni sul Package	22
4.1.4.2	Informazioni sulle Classi	22
4.1.5	SWEDesigner::Server::Controller::Services::JavaGenService	22
4.1.5.1	Informazioni sul Package	22
4.1.5.2	Informazioni sulle Classi	23
4.1.6	SWEDesigner::Server::Controller::Services::UserServices	24
4.1.6.1	Informazioni sul Package	24
4.1.6.2	Informazioni sulle Classi	24
4.1.7	SWEDesigner::Server::Model	25
4.1.7.1	Informazioni sul Package	25
4.1.7.2	Informazioni sulle Classi	26
5	Front-end	27
5.1	Descrizione packages e classi	27
5.1.1	SWEDesigner::Client	27
5.1.1.1	Informazioni sul Package	27
5.1.1.2	Informazioni sulle Classi	27
5.1.2	SWEDesigner::Client::Components	28
5.1.2.1	Informazioni sul Package	28
5.1.2.2	Informazioni sulle Classi	29
5.1.3	SWEDesigner::Client::Components::EditorComponents	30
5.1.3.1	Informazioni sul Package	30
5.1.3.2	Informazioni sulle Classi	30
5.1.4	SWEDesigner::Client::Components::DashboardComponents	32

5.1.4.1	Informazioni sul Package	32
5.1.4.2	Informazioni sulle Classi	32
5.1.5	SWEDesigner::Client::Service	33
5.1.5.1	Informazioni sul Package	33
5.1.6	SWEDesigner::Client::Services::UserServices	34
5.1.6.1	Informazioni sul Package	34
5.1.6.2	Informazioni sulle Classi	34
5.1.7	SWEDesigner::Client::Services::ProjectServices	35
5.1.7.1	Informazioni sul Package	35
5.1.7.2	Informazioni sulle Classi	35
6	Tracciamento	36
6.1	Tracciamento componenti - requisiti	36
6.2	Tracciamento requisiti - componenti	39
A	Descrizione <i>Design Pattern_G</i>	44
A.1	<i>Design Pattern_G</i> Architetture	44
A.1.1	MVVM	44
A.1.2	Three-Tier	45
A.2	<i>Design Pattern_G</i> Creazionali	46
A.2.1	Factory Method	46
A.3	<i>Design Pattern_G</i> Strutturali	47
A.3.1	Decorator	47
A.3.2	Facade	48
A.4	<i>Design Pattern_G</i> Comportamentali	49
A.4.1	Dependency Injection	49
A.4.2	Command	50

Elenco delle figure

1	Diagramma di $deployment_G$ per l'architettura	16
2	Diagramma dei packages SWEDesigner::Server	18
3	Diagramma dei packages SWEDesigner::Server::Controller	19
4	Diagramma dei packages SWEDesigner::Server::Controller::Services::JavaGenService	23
5	Diagramma dei packages SWEDesigner::Server::Controller::Services::UserServices	24
6	Diagramma dei packages SWEDesigner::Server::Model	26
7	Diagramma dei packages SWEDesigner::Client	27
8	Esempio di funzionamento dell'applicazione lato client	28
9	Diagramma dei packages SWEDesigner::Client::Components	28
10	Diagramma dei packages SWEDesigner::Client::Components::EditorComponents	30
11	Diagramma dei packages SWEDesigner::Client::Components::DashComponents	32
12	Diagramma dei packages SWEDesigner::Client::Service	33
13	Diagramma del $Design Pattern_G$ MVVM	44
14	Diagramma del $Design Pattern_G$ Three-Tier	45
15	Diagramma del $Design Pattern_G$ Factory method	46
16	Diagramma del $Design Pattern_G$ Decorator	47
17	Diagramma del $Design Pattern_G$ Facade	48
18	Diagramma del $Design Pattern_G$ Dependency Injection	49
19	Diagramma del $Design Pattern_G$ Command	50

1 Introduzione

1.1 Scopo del documento

Questo documento ha come scopo quello di definire la *progettazione ad alto livello_G* per il prodotto. Verrà presentata la struttura generale secondo la quale saranno organizzate le varie componenti software e i *Design Pattern_G* utilizzati nella creazione del prodotto SWEDesigner. Verrà dettagliato il tracciamento tra le componenti software individuate ed i requisiti.

1.2 Scopo del prodotto

Lo scopo del progetto è la realizzazione di una *Web App_G* che fornisca all'*Utente_G* un *UML_G Designer_G* con il quale riuscire a disegnare correttamente *Diagrammi_G* delle *Classi_G* e descrivere il comportamento dei *Metodi_G* interni alle stesse attraverso l'utilizzo di *Diagrammi_G* delle attività. La *Web App_G* permetterà all'*Utente_G* di generare *Codice_G Java_G* dall'insieme dei *diagrammi classi_G* e dei rispettivi *metodi_G*.

1.3 Glossario

Con lo scopo di evitare ambiguità di linguaggio e di massimizzare la comprensione dei documenti, il gruppo ha steso un documento interno che è il *Glossario v2.0.0*. In esso saranno definiti, in modo chiaro e conciso i termini che possono causare ambiguità o incomprensione del testo.

1.4 Riferimenti

1.4.1 Normativi

- **Capitolato d'Appalto C6: SWEDesigner**
<http://www.math.unipd.it/~tullio/IS-1/2015/Progetto/C6p.pdf>;
- **Norme di Progetto:** *Norme di Progetto v2.0.0*.
- **Analisi dei Requisiti:** *Analisi dei Requisiti v2.0.0*.

1.4.2 Informativi

- Slide dell'insegnamento Ingegneria del Software modulo A:
<http://www.math.unipd.it/~tullio/IS-1/2016/>.
 - Slides del corso di Ingegneria del Software mod. A: *Diagrammi delle classi_G*: <http://www.math.unipd.it/~tullio/IS-1/2015/Dispense/E03.pdf>;
 - Slides del corso di Ingegneria del Software mod. A: Diagrammi dei package: <http://www.math.unipd.it/~tullio/IS-1/2015/Dispense/E04.pdf>;
 - Slides del corso di Ingegneria del Software mod. A: Diagrammi di sequenza: <http://www.math.unipd.it/~tullio/IS-1/2015/Dispense/E05.pdf>;
 - Slides del corso di Ingegneria del Software mod. A: Diagrammi di attività: <http://www.math.unipd.it/~tullio/IS-1/2015/Dispense/E06.pdf>;
 - Slides del corso di Ingegneria del Software mod. A: *Design pattern_G* strutturali: Decorator, Proxy, Facade, Adapter: <http://www.math.unipd.it/~tullio/IS-1/2015/Dispense/E07.pdf>;
 - Slides del corso di Ingegneria del Software mod. A: *Design pattern_G* creazionali: Singleton, Builder, Abstract Factory: <http://www.math.unipd.it/~tullio/IS-1/2015/Dispense/E08.pdf>;
 - Slides del corso di Ingegneria del Software mod. A: *Design pattern_G* comportamentali: Observer, Template Method, Command, Strategy, Iterator: <http://www.math.unipd.it/~tullio/IS-1/2015/Dispense/E09.pdf>;
- Design Patterns - E. Gamma, R. Helm, R. Johnson, J. Vlissides (Pearson Education, Addison-Wesley, 1995);
- *Node.js_G*: <https://nodejs.org/dist/latest-v5.x/docs/api/>;
- MongoDB: <https://docs.mongodb.org/manual/>;
- HTML5: http://www.w3schools.com/html/html5_intro.asp;
- CSS3: http://www.w3schools.com/css/css3_intro.asp;
- ExpressJS: <http://expressjs.com/en/4x/api.html>.
- Mustache: <http://mustache.github.io/>.

2 Tecnologie utilizzate

L'architettura è stata progettata utilizzando lo stack di **MEAN_G** (<http://mean.io/>), il quale comprende 4 tecnologie, alcune delle quali espressamente richieste nel *capitolato_G* d'appalto. Vengono di seguito elencate e descritte le principali tecnologie impiegate comprese in **MEAN_G** e le motivazioni del loro utilizzo:

- **Node.js**: piattaforma per il *back-end_G*;
- **Expressjs**: *framework_G* per la realizzazione dell'applicazione web in *Node.js_G* ;
- **MongoDB**: *database_G* di tipo *NoSQL_G* per la parte di recupero e salvataggio dei dati;
- **Mongoose**: *libreria_G* per interfacciarsi con il driver di **MongoDB**;
- **Angular 4.0**: *framework_G* *JavaScript_G* per la realizzazione del *front-end_G* .

2.1 Server

2.1.1 Node.js

Node.js è una *piattaforma_G* software costruita sul motore *JavaScript_G* di *Chrome_G* che permette di realizzare facilmente applicazioni di rete scalabili e veloci. *Node.js_G* utilizza *JavaScript_G* come linguaggio di programmazione, e grazie al suo modello *event-driven_G* con chiamate di input/output non bloccanti risulta essere leggero e efficiente.

2.1.1.1 Vantaggi

- **Approccio asincrono**: *Node.js_G* permette di accedere alle risorse del sistema operativo in modalità *event-driven_G* e non sfruttando il classico modello basato su processi concorrenti utilizzato dai classici web *server_G*. Ciò garantisce una maggiore efficienza in termini di prestazioni, poiché durante le attese il runtime può gestire qualcos'altro in maniera asincrona;
- **Architettura modulare**: Lavorando con *Node.js_G* è molto facile organizzare il lavoro in librerie, importare i *moduli_G* e combinarli fra loro. Questo è reso molto comodo attraverso il *node package manager_G* (**npm**) attraverso il quale lo sviluppatore può contribuire e accedere ai *package_G* messi a disposizione dalla community.

2.1.1.2 Svantaggi

- **Supporto incompleto alle feature di ES6_G**: Molte delle feature di ES6 non sono supportate in Node nella versione 4.4 scelta come versione di riferimento per lo sviluppo del progetto.

2.1.2 Expressjs

Expressjs è un *framework_G* minimale per creare *Web App_G* con *Node.js_G*. Richiede *moduli_G* Node di terze parti per applicazioni che prevedono l'interazione con le *database_G*. È stato utilizzato il *framework_G* *Expressjs_G* per supportare lo sviluppo dell'*application server_G* grazie alle utili e robuste caratteristiche da esso offerte, le quali sono pensate per non oscurare le funzionalità fornite da *Node.js_G* aprendo così le porte all'utilizzo di moduli per *Node.js_G* atti a supportare specifiche funzionalità.

2.1.2.1 Vantaggi

- **Minimale:** si basa su *Node.js_G* e permette di estenderlo a seconda dei bisogni dell'applicazione;
- **Documentazione:** esaustiva e completa;
- **Apprendimento:** facile da imparare.

2.1.2.2 Svantaggi

- **Integrazione:** richiede di integrare *moduli_G* diversi per comporre l'applicazione finale. Altri *framework_G* permettono di definire *API_G* (Application Programming Interface) *REST_G* (REpresentational State Transfer) in modo semplice, ma vincolano maggiormente nelle scelte progettuali.

2.1.3 MongoDB

MongoDB_G è un *database_G* *NoSQL_G* *open source_G* scalabile e altamente performante di tipo document-oriented, in cui i dati sono archiviati sotto forma di documenti in stile *JSON_G* con schemi dinamici, secondo una struttura semplice e potente.

2.1.3.1 Vantaggi

- **Alte performance:** non ci sono join che possono rallentare le operazioni di lettura o scrittura. L'indicizzazione include gli indici di chiave anche sui documenti innestati e sugli array, permettendo una rapida interrogazione al *database_G*;
- **Affidabilità:** alto meccanismo di replicazione su server;
- **Schemaless:** non esiste nessuno *schema_G*, è più flessibile e può essere facilmente trasposto in un modello ad oggetti;
- Permette di definire query complesse utilizzando un linguaggio che non è *SQL_G*;

- Permette di processare parallelamente i dati (*Map-Reduce_G*);
- Tipi di dato più flessibili.

2.1.3.2 Svantaggi

- **Flessibilità:** per i tipi di dato. Sebbene questo possa essere visto come vantaggio, è opinione del team che un'eccessiva flessibilità possa portare più problemi che benefici: allo scopo di aggiungere rigidità è stato infatti scelto, come verrà descritto in seguito, Mongoose, che introduce una costruzione a schemi per le collections di *MongoDB_G* e quindi vincola i documenti inseriti ad avere una struttura uniforme;
- **Nessun supporto per le transazioni:** sono supportate alcune operazioni atomiche, ma a livello di documento;
- **Nessun *join_G*:** va simulato via codice attraverso query multiple;
- **Problemi di concorrenza:** per le operazioni di scrittura viene creato un lock sull'intero database. Questo lock blocca anche le operazioni di lettura.

2.1.4 Mongoose

Mongoose è una *libreria_G* per interfacciarsi a *MongoDB_G* che permette di definire degli schemi per modellare i dati del *database_G*, imponendo una certa struttura per la creazione di nuovi Document . Inoltre fornisce molti strumenti utili per la validazione dei dati, per la definizione di query e per il cast dei tipi predefiniti. Per interfacciare l'applicazione *server_G* con *MongoDB_G* sono disponibili diversi progetti *open source_G*. Per questo progetto è stato scelto di utilizzare *Mongoose.js_G* , attualmente il più di uso.

2.1.4.1 Vantaggi

- **Diffusione:** è la libreria più diffusa per interfacciarsi con *MongoDB_G*;
- **Funzionalità aggiuntive:** permette di definire strumenti per la validazione dei dati e per il cast dei tipi;
- **Permette di eseguire dei *join_G* tra collections:** Sebbene non sia previsto da *MongoDB_G*, *mongoose_G* prevede la funzione *populate* per imitare la funzione di *join_G* in modo completamente trasparente per l'utilizzatore;
- **Rapido ed intuitivo:** La strutturazione dei dati con questa libreria è rapida ed intuitiva, ciò dovuto anche dalla sintassi dichiarativa della libreria stessa.

2.1.4.2 Svantaggi

- **Schema-based:** è basato sulla creazione di una forte schematizzazione per i documenti, e questo limita l'estrema flessibilità di *MongoDB_G*.

2.1.5 Grunt

Si tratta di un JavaScript Task Runner che automatizza la compilazione e l'esecuzione dei test di unità. Mediante l'utilizzo di un *gruntfile* è possibile pianificare delle attività che verranno svolte in maniera automatizzata in maniera tale da ottenere un notevole risparmio di tempo sul lavoro.

Tale strumento si è rivelato piuttosto utile per riuscire ad integrare l'utilizzo della libreria grafica Draw2D all'interno di Angular 4 in quanto semplifica e automatizza il building del progetto.

2.2 Librerie

Vengono di seguito descritte le *librerie_G* aggiuntive utilizzate dal *back-end_G*. La scelta è stata effettuata cercando di valutare la diffusione, il livello di stabilità, l'assenza di errori noti.

2.2.1 Mustache

Mustache è un *template_G* engine che permette di espandere tags all'interno di un *template_G*, racchiusi da 2 parentesi graffe, usando valori forniti da oggetti.

2.2.2 Passport:

È un *middleware_G* di autenticazione per *Node.js_G*. Estremamente flessibile e modulare, Passport può essere facilmente inserito in qualsiasi applicazione web basata su *Expressjs_G*.

2.2.3 BodyParser:

Modulo_G di terze parti per la corretta lettura delle informazioni contenute nel body delle richieste *HTTP_G*; viene utilizzato come *middleware_G* per Expressjs e si occupa della corretta lettura delle informazioni contenute nel body di una richiesta *HTTP_G*. Nel nostro caso verrà impiegato per la lettura dei dati del body in formato *JSON_G*.

2.2.4 Forge:

È un *modulo_G* in *Javascript_G* che fornisce funzionalità di crittografia per la sicurezza e un set di strumenti per la realizzazione di *Web app_G*.

2.2.5 PassportJWT:

È un *modulo_G* che fornisce una strategia Passport (§3.2) per l'autenticazione con un *JSON_G* Web Token, che è una tecnica compatta per trasmettere in modo sicuro le informazioni tra 2 oggetti *JSON_G*.

2.2.6 Bcrypt:

È una *libreria_G* in *Javascript_G* che utilizza la funzione di *hash_G*ing per criptare le password.

2.3 Client

2.3.1 Angular 4.0

Angular 4.0 è un *framework_G* web *open source_G* per lo sviluppo di applicazioni Web lato client; utile a semplificare la realizzazione di applicazioni web, come ad esempio le Single Page Application, cioè applicazioni le cui risorse vengono caricate dinamicamente su richiesta, senza necessità di ricaricare l'intera pagina.

2.3.1.1 Vantaggi

- **Velocità:** Riduce in maniera considerevole il codice necessario a realizzare applicazioni *HTML_G*/*JavaScript_G*;
- **Ampia documentazione disponibile;**
- **Data Binding bidirezionale:** approccio automatico per aggiornare la vista ogniqualvolta il model cambia e viceversa. Ciò semplifica lo sviluppo eliminando la necessità di manipolare il *DOM_G*;
- **Sviluppato per facilitare la fase di test;**
- **Direttive:** caratteristica peculiare di Angular e permettono di estendere la sintassi *HTML_G*, creando dei componenti specifici per la propria applicazione e facilmente riutilizzabili;

2.3.1.2 Svantaggi

- **Maggior studio:** curva di apprendimento più ripida rispetto ad altri *framework_G*;
- **Codice articolato:** ciò potrebbe comportare, in caso di variazione dei requisiti, delle difficoltà delle successive modifiche.

2.3.2 Draw2D

Drwa2D è una libreria *JavaScript_G* di *diagrammi_G* che consente di creare rapidamente applicazioni di grafici interattive e grafici che vengono eseguiti in modo nativo su tutti i browser principali.

2.3.2.1 Vantaggi

- **Non sono necessari altri plug-in:** Ciò elimina i plug-in di dipendenza dai fornitori;
- **Open source_G:** Le tecnologie coinvolte sono libere e ci sono molte implementazioni aperte, nessun fornitore può rimuovere un prodotto o una tecnologia che lascia in pratica la tua applicazione inoperabile;
- **Le tecnologie sono standardizzate:** L'applicazione è distribuibile al numero massimo di utenti del browser senza bisogno di ulteriori configurazioni o installazione sul computer *client_G*. Gli ambienti aziendali di grandi dimensioni spesso non amano consentire agli individui di installare plug-in del *browser_G* e non amano cambiare la build standard creata su tutte le macchine.

2.3.2.2 Svantaggi

- **Aumento rapido di celle:** Poiché il numero di celle visibili sullo schermo degli utenti aumenta di centinaia, la valutazione rallenta oltre i limiti accettabili sulla maggior parte dei *browser_G*. Nella teoria della gestione delle informazioni, visualizzare diverse centinaia di celle è generalmente sbagliato, in quanto l'utente non può interpretare i dati.

2.3.3 HTML5

È un linguaggio di markup per la strutturazione delle pagine web, pubblicato come W3C Recommendation da ottobre 2014. L'uso di HTML5 rispetto a XHTML (**eX**tensible **H**yper**T**ext **M**arkup **L**anguage) è stato deciso all'unanimità dal gruppo.

2.3.3.1 Vantaggi

- Raccomandazione W3C;
- reazione di pagine interattive: soprattutto se usato insieme a CSS.

2.3.3.2 Svantaggi

- Supporto: non tutti i browser lo supportano allo stesso modo, e non tutte le caratteristiche definite sono ancora completamente supportate.

2.3.4 CSS3

È un linguaggio utilizzato per definire la formattazione di documenti HTML. Le regole per la composizione di un foglio di stile CSS sono definite dal W3C a partire dal 1996. Inoltre permette di separare i contenuti delle pagine HTML dalla loro formattazione, assicurando una maggiore manutenibilità e riutilizzo.

2.3.4.1 Vantaggi

- Separazione tra contenuto e presentazione;
- Raccomandato da W3C.

3 Descrizione architettura

3.1 Metodo e formalismo di specifica

Le scelte architetturali per lo sviluppo di SWEDesigner sono state fortemente influenzate dallo stack tecnologico utilizzato.

Nell'esposizione dell'architettura dell'applicazione si procederà con un approccio *top-down_G*, descrivendo l'architettura iniziando dal generale ed andando al particolare; si è partiti suddividendo il sistema in *front-end_G* e *back-end_G*, definendo l'interfaccia di comunicazione, scegliendo di seguire in ciascuno l'organizzazione suggeritaci dai *framework_G*.

La descrizione dell'architettura di SWEDesigner è suddivisa in quattro sezioni:

- §3.2: illustra gli aspetti generali dell'architettura del software;
- §3.3: descrive il protocollo che lega le due interfacce tra *Client_G* e *Server_G*; che descrive l'architettura del front end dell'applicazione;
- §3.4: descrive l'architettura del *back-end_G* dell'applicazione;
- §3.5: descrive l'architettura del *front-end_G* dell'applicazione.

Per descrivere in maniera formale l'architettura verranno impiegati lo standard *UML_G* 2.0 per i diagrammi dei *package_G* e delle classi e lo standard *UML_G* 2.5 per i *diagrammi di attività_G* e sequenza.

I diagrammi delle *classi_G* che permettono di mostrare l'architettura generale del sistema vengono affiancati anche dai diagrammi di sequenza e attività, che permettono di definire le interazioni tra le componenti, senza preoccuparsi della loro classificazione. In questo modo è possibile esprimere alcuni meccanismi tipici di un'applicazione *REST_G*-like, come il modo in cui agiscono i *middleware_G*.

3.2 Architettura generale

L'architettura del progetto si divide in una componente *Client_G*, rappresentata da un'applicazione *front-end_G* accessibile da un browser, e in una componente *WebServer_G*, nella quale risiede il *back-end_G* che gestisce le richieste di generazione del *codice_G*.

L'architettura generale di SWEDesigner si divide in 3 macrocomponenti:

- *Client_G*;
- *Server_G REST_G*;
- *Database_G*.

L'architettura proposta segue il *Design Pattern_G Three-tier*. Esso è diviso in 3 livelli:

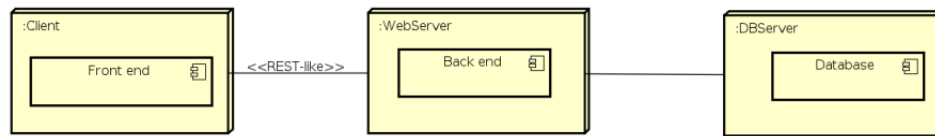


Figura 1: Diagramma di $deployment_G$ per l'architettura

- **Presentation tier:** rappresenta l'interfaccia verso l'utente ovvero il $front-end_G$;
- **Business logic:** è il livello che coordina l'applicazione ed effettua quindi le decisioni logiche e le valutazioni ovvero il $back-end_G$;
- **Data tier:** è il livello dove le informazioni vengono salvate, ovvero il $database_G$.

In particolare i ruoli di Model e Controller verranno implementati a livello di $server_G$, mentre il ruolo di View viene affidato al $front-end_G$. L'interfaccia tra le due componenti verrà gestita grazie ad un set di API_G disposto dal $server_G REST_G$; il $Database_G$ serve per garantire la persistenza del programma generato: ogni $utente_G$ autenticato può salvare i propri progetti e mantenere i diagrammi creati.

Le tre macrocomponenti verranno descritte in dettaglio in seguito su questo documento.

3.3 Interfaccia REST-like

Per l'interfaccia della componente $back-end_G$ si è scelto di utilizzare uno stile basato REST. All'interno di un'unica sessione utente, a partire dall'operazione di login fino a quella di logout, l'interfaccia con cui si accede agli elementi delle collection può considerarsi effettivamente REST.

I motivi che hanno spinto alla scelta di REST sono:

- Semplicità di utilizzo;
- Facile integrazione con i $framework_G$ esistenti;
- Indipendenza dal linguaggio di programmazione utilizzato.

REST utilizza il concetto di risorsa, ovvero un aggregato di dati con un nome (URI) e una rappresentazione, su cui è possibile invocare le operazioni CRUD tramite la seguente corrispondenza:

3.4 Architettura del Server

L'implementazione scelta per il backend dell'applicazione è un server che segue lo stile architetturale $REST_G$; ciò implica che:

- l'applicazione renda disponibili le sue funzioni in veste di risorse web;
- ogni risorsa resa disponibile è indirizzabile univocamente utilizzando un indirizzo URL;
- l'interfaccia delle risorse deve essere uniforme e deve garantire un insieme ben definito di operazioni e una gestione priva di stato delle operazioni.

Tale architettura permette l'indipendenza completa tra *back-end_G* e *front-end_G*, permettendo così espansioni su altre piattaforme senza dover modificare il *back-end_G* dell'applicazione. Il collegamento tra il *front-end_G* e i modelli nel *back-end_G* verrà implementato da uno stack di *middleware_G*.

3.5 Architettura del Client

Il *front-end_G* di SWEDesigner è una *Single Page Application_G* realizzata nel framework Angular4.0. L'architettura è MVVM (*Model-View-View Model*), sono descritti due package principali Components e Services, i Components costituiscono la parte Viewmodel dell'architettura, contiene classi atte a rendere dinamica la pagina web; il package Services, che ripecchia la parte Model, offre metodi per l'interazione con il lato server e la libreria grafica. La parte View non viene descritta in quanto composta da template html statici che sono strettamente legati omonimi components.

4 Componenti del *Back-end*

4.1 Descrizione packages e classi

4.1.1 SWEDesigner::Server

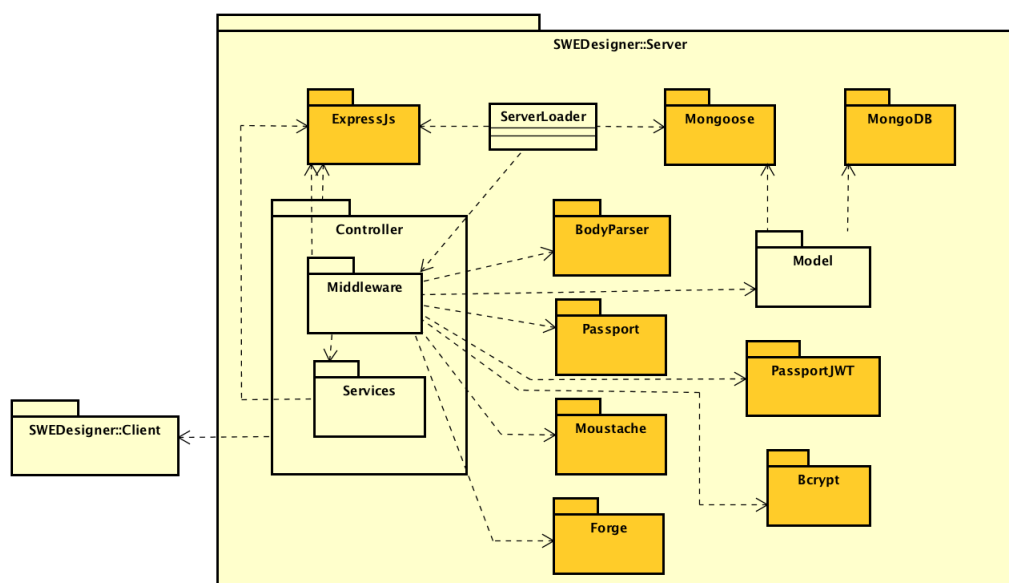


Figura 2: Diagramma dei packages SWEDesigner::Server

4.1.1.1 Informazioni sul Package

- **Descrizione:**
Package che racchiude tutta la componente del server scritta in JavaScript.
- **Padre:**
SWEDesigner
- **Package contenuti:**
 - SWEDesigner::Server::Controller;
Questo package contiene tutte le componenti middleware e i servizi con i quali si interfacciano. Ogni controller si occupa di gestire tutte le richieste del client attraverso i suoi componenti middleware e di rispondere ad esse attraverso l'interfaccia REST definita da express.
 - SWEDesigner::Server::Model;
Questo package contiene le classi e i metodi che si interfacciano con il database

passando dal modulo di moongose. Il model si occupa quindi delle richieste al database e delle operazioni ad esso dedicate rispondendo alle varie richieste del controller.

4.1.1.2 Informazioni sulle Classi

- SWEDesigner::Server::ServerLoader
 - **Descrizione:**
Classe che consente il caricamento del server.
 - **Utilizzo:**
La classe viene utilizzata per caricare tutte le componenti del server nel momento dell'avvio dell'applicazione.

4.1.2 SWEDesigner::Server::Controller

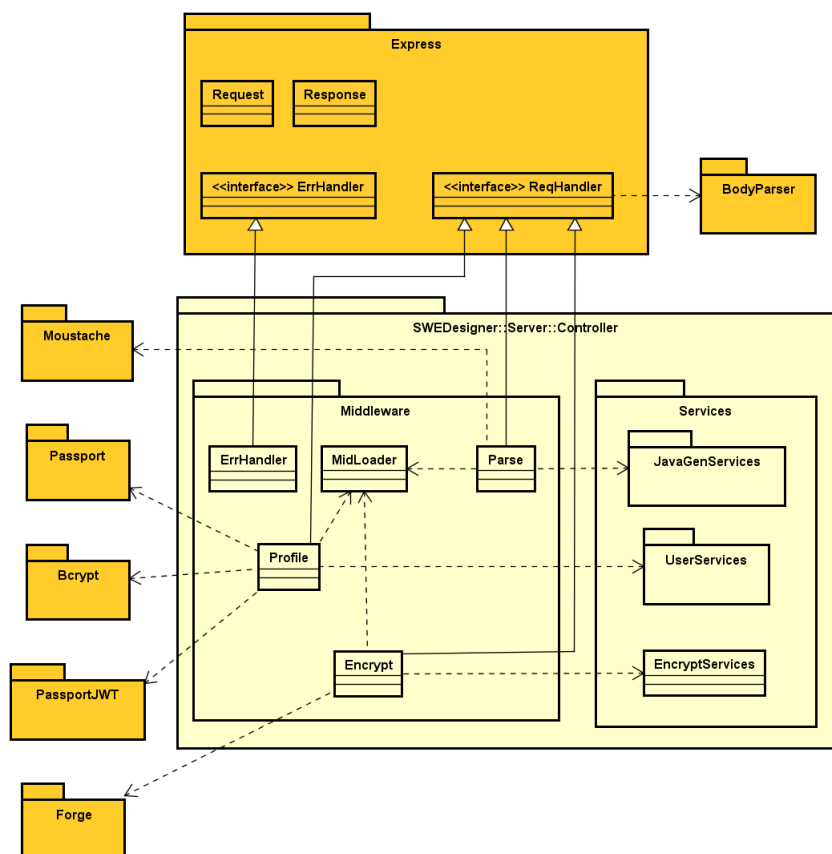


Figura 3: Diagramma dei packages SWEDesigner::Server::Controller

4.1.2.1 Informazioni sul Package

- **Descrizione:**
Il package racchiude al suo interno tutti i servizi e le componenti middleware che regolano la bussiness logic del server.
- **Padre:**
SWEDesigner::Server
- **Package contenuti:**
 - *SWEDesigner::Server::Controller::Middleware;*
Si tratta del package contenente tutte le componenti middleware che rispondono alle varie richieste del client.
 - *SWEDesigner::Server::Controller::Services;*
Si tratta del package contenente tutti i servizi utilizzati dalle componenti middleware per il corretto svolgersi delle loro operazioni.

4.1.3 SWEDesigner::Server::Controller::Middleware

4.1.3.1 Informazioni sul Package

- **Descrizione:**
Si tratta del package contenente tutte le componenti middleware che rispondono alle varie richieste del client.
- **Padre:**
SWEDesigner::Server::Controller

4.1.3.2 Informazioni sulle Classi

- SWEDesigner::Server::Controller::Middleware::ErrorHandler
 - **Descrizione:**
Si tratta della classe che si occupa della gestione degli errori nelle richieste REST che derivano dal client.
 - **Utilizzo:**
La classe, interfacciandosi con un'interfaccia di express, si occupa di gestire, grazie ad una relazione con la realita interfaccia di express, tutti gli errori delle richieste arrivate dal client.
 - **Relazioni con le altre classi:**
 - * *IN* express::ErrorHandler
- SWEDesigner::Server::Controller::Middleware::MidLoader

- **Descrizione:**
Si tratta della classe che si occupa di caricare tutte le componenti middleware del server.
- **Utilizzo:**
La classe carica tutte le componenti middleware del server applicando il design patter *Facade_G*.
- SWEDesigner::Server::Controller::Middleware::Parse
 - **Descrizione:**
Classe per la gestione del parsing dei file JSON.
 - **Utilizzo:**
La classe, utilizzando la componente esterna Moustache, si occupa di fare il parsing dei file JSON in arrivo dal client e creare, tramite il servizio JavaGen, il codice sorgente.
 - **Relazioni con le altre classi:**
 - * *IN* MidLoader
 - * *IN* express::ReqHandler
- SWEDesigner::Server::Controller::Middleware::Profile
 - **Descrizione:**
Classe per la gestione dei servizi middleware riguardanti l'utente, come registrazione e autenticazione.
 - **Utilizzo:**
Con l'ausilio di moduli esterni di Passport e Bcrypt, la classe si occupa di gestire tutti quei servizi middleware che riguardano il profilo dell'utente.
 - **Relazioni con le altre classi:**
 - * *IN* MidLoader
 - * *IN* express::ReqHandler
- SWEDesigner::Server::Controller::Middleware::Encrypt
 - **Descrizione:**
Classe per l'encrypt e il decrypt dei file di progetto.
 - **Utilizzo:**
La classe, utilizzando il modulo esterno Forge, encrypta i file di progetto generati tramite SWEDesigner e ne effettua la decrittazione al momento del caricamento degli stessi.
 - **Relazioni con le altre classi:**

- * *IN* MidLoader
- * *IN* express::ReqHandler

4.1.4 SWEDesigner::Server::Controller::Services

4.1.4.1 Informazioni sul Package

- **Descrizione:**
Package contenente tutti i servizi utili a rispondere alle richieste REST del client.s
- **Padre:**
SWEDesigner::Server::Controller
- **Package contenuti:**
 - *SWEDesigner::Server::Controller::Services::JavaGenServices*;
Tramite questo package si effettuano tutti i servizi dedicati alla generazione di codice sorgente a partire dagli UML creati dall'utente all'interno di SWEDesigner.
 - *SWEDesigner::Server::Controller::Services::UserServices* Il package presenta tutti quei servizi utili alle componenti middleware per la gestione del profilo dell'utente.

4.1.4.2 Informazioni sulle Classi

- SWEDesigner::Server::Controller::Services::EncryptServices
 - **Descrizione:**
La classe offre il servizio di encrypt e decrypt utile alla componente Middleware::Encrypt.
 - **Utilizzo:**
Il servizio in questione offre tutti i metodoti utili al middleware per effettuare la criptazione e decrittazione dei file di progetto caricati e salvati dagli utenti.

4.1.5 SWEDesigner::Server::Controller::Services::JavaGenService

4.1.5.1 Informazioni sul Package

- **Descrizione:**
Il pacchetto contiene tutti i servizi di generazione e download del codice sorgente a partire dagli UML disegnati dall'utente.
- **Padre:**
SWEDesigner::Server::Controller::Services

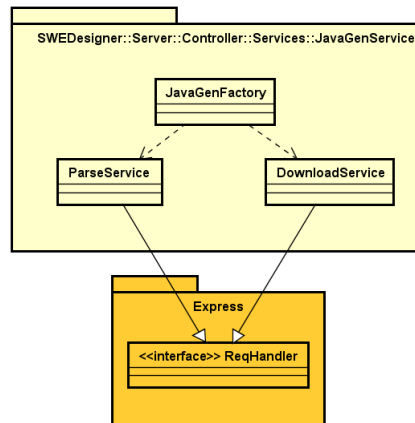


Figura 4: Diagramma dei packages SWEDesigner::Server::Controller::Services::JavaGenService

4.1.5.2 Informazioni sulle Classi

- SWEDesigner::Server::Controller::Services::JavaGenService::JavaGenFactory
 - **Descrizione:**
La classe implementa il pattern *Factory_G* per la creazione di un controller che si occupi di gestire i servizi di generazione codice.
 - **Utilizzo:**
La classe viene utilizzata come creator per la creazione del controller dei servizi di generazione, tramite parsing, e download del codice sorgente.
- SWEDesigner::Server::Controller::Services::JavaGenService::ParseService
 - **Descrizione:**
La classe effettua il parsing del JSON generato dai diagrammi UML e lo trasforma in codice sorgente Java.
 - **Utilizzo:**
Mediante l'utilizzo del modulo esterno Moustache, il servizio si occupa di effettuare il parsing del JSON inviato tramite richiesta REST dal client e di generare il codice sorgente associato.
 - **Relazioni con le altre classi:**
 - * *OUT* Express::ReqHandler
- SWEDesigner::Server::Controller::Services::JavaGenService::DownloadService
 - **Descrizione:**
La classe effettua il download del codice sorgente generato a partire dagli UML dell'utente.

– **Utilizzo:**

La classe invia al client che lo ha richiesto lo stream del file .jar sorgente creato a partire dal JSON inviato tramite richiesta REST.

– **Relazioni con le altre classi:**

* *OUT* Express::ReqHandler

4.1.6 SWEDesigner::Server::Controller::Services::UserServices

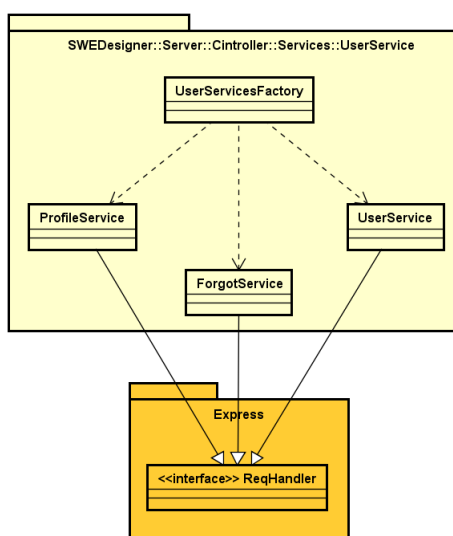


Figura 5: Diagramma dei packages SWEDesigner::Server::Controller::Services::UserServices

4.1.6.1 Informazioni sul Package

• **Descrizione:**

Il package contiene al suo interno tutti i servizi utili all'utente per l'autenticazione e la gestione del profilo.

• **Padre:**

SWEDesigner::Server::Controller::Services

4.1.6.2 Informazioni sulle Classi

• SWEDesigner::Server::Controller::Services::UserServices::UserServicesFactory

– **Descrizione:**

La classe implementa il patter *Factory_G* per la creazione di un controller che gestisca i servizi relativi all'autenticazione e registrazione di un utente.

- **Utilizzo:**
La classe viene utilizzata come creator per la creazione del controller che si occupa dei servizi riguardanti il profilo dell'utente e le sue credenziali.
- SWEDesigner::Server::Controller::Services::UserServices::ProfileService
 - **Descrizione:**
La classe gestisce i servizi di autenticazione e registrazione.
 - **Utilizzo:**
La classe contiene tutti metodi necessari per l'autenticazione e la registrazione di un utente all'interno del database.
 - **Relazioni con le altre classi:**
 - * *OUT* Express::ReqHandler
- SWEDesigner::Server::Controller::Services::UserServices::ForgotService
 - **Descrizione:**
La classe offre il servizio di recupero password.
 - **Utilizzo:**
La classe permette all'utente di recuperare le credenziali del proprio account.
 - **Relazioni con le altre classi:**
 - * *OUT* Express::ReqHandler
- SWEDesigner::Server::Controller::Services::UserServices::UserService
 - **Descrizione:**
La classe gestisce il profilo dell'utente.
 - **Utilizzo:**
La classe offre i servizi di management di un account.
 - **Relazioni con le altre classi:**
 - * *OUT* Express::ReqHandler

4.1.7 SWEDesigner::Server::Model

4.1.7.1 Informazioni sul Package

- **Descrizione:**
Il package si occupa delle comunicazioni con il database passando per il modulo di Mongoose per le richieste allo stesso.
- **Padre:**
SWEDesigner

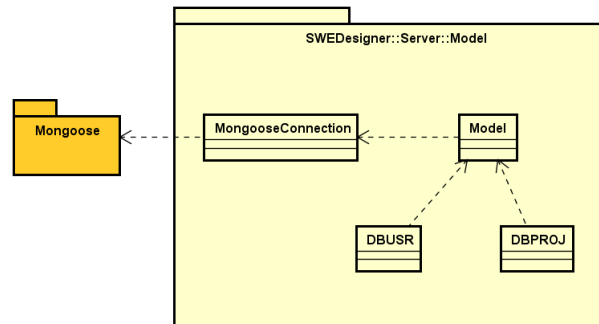


Figura 6: Diagramma dei packages `SWEDesigner::Server::Model`

4.1.7.2 Informazioni sulle Classi

- `SWEDesigner::Server::Model::MongooseConnection`
 - **Descrizione:**
La classe stabilisce la connessione al database.
 - **Utilizzo:**
La classe si connette al database passando per il modulo di Mongoose.
- `SWEDesigner::Server::Model::Model`
 - **Descrizione:**
La classe implementa il pattern *Facade_G* per la realizzazione del data tier.
 - **Utilizzo:**
La classe fornisce un'interfaccia per le classi che gestiscono il database.
- `SWEDesigner::Server::Model::DBUSR`
 - **Descrizione:**
La classe gestisce il database utenti.
 - **Utilizzo:**
La classe fornisce tutti gli strumenti per la lettura e la scrittura di un utente.
- `SWEDesigner::Server::Model::DBPROJ`
 - **Descrizione:**
La classe gestisce il database dei progetti.
 - **Utilizzo:**
La classe fornisce tutti gli strumenti per la lettura e la scrittura di un progetto.

5 Front-end

5.1 Descrizione packages e classi

5.1.1 SWEDesigner::Client

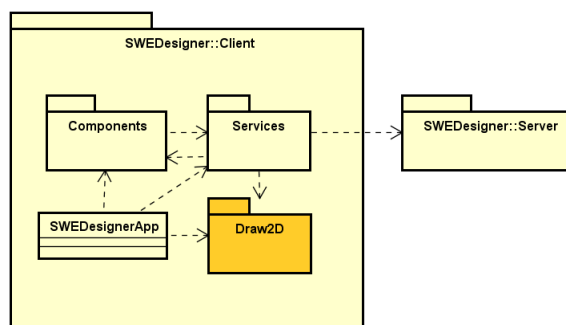


Figura 7: Diagramma dei packages SWEDesigner::Client

5.1.1.1 Informazioni sul Package

- **Descrizione:**
Package che racchiude tutta la componente di Front-end scritta in JavaScript.
- **Padre:**
SWEDesigner
- **Package contenuti:**
 - SWEDesigner::Client::Components;
 - SWEDesigner::Client::Services;
 - SWEDesigner::Server;

5.1.1.2 Informazioni sulle Classi

- SWEDesigner::Client::SWEDesignerApp
 - **Descrizione:**
Questa classe detta anche *root module* si occupa di definire i moduli che compongono l'applicazione e come istanziarli.
 - **Utilizzo:**
È il primo modulo su cui viene fatto il bootstrap per lanciare l'applicazione

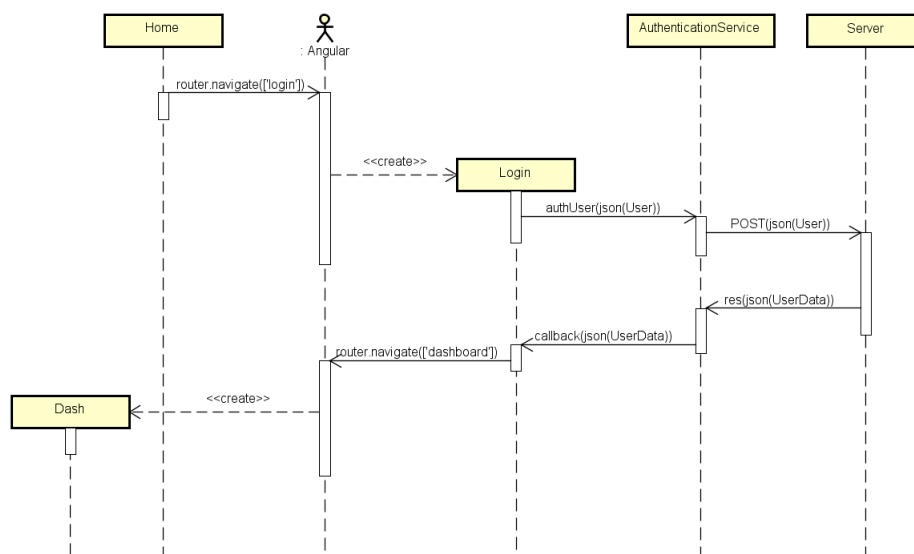


Figura 8: Esempio di funzionamento dell'applicazione lato client

5.1.2 SWEDesigner::Client::Components

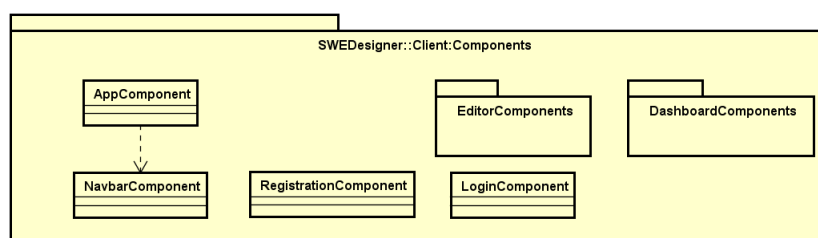


Figura 9: Diagramma dei packages SWEDesigner::Client::Components

5.1.2.1 Informazioni sul Package

- **Descrizione:**
Questo package contiene tutti i components dell'applicazione
- **Padre:**
SWEDesigner::Client
- **Package contenuti:**
 - SWEDesigner::Client::Components::EditorComponents;
 - SWEDesigner::Client::Components::DashComponents;

5.1.2.2 Informazioni sulle Classi

- SWEDesigner::Client::Components::AppComponent
 - **Descrizione:**
Il component descrive un contenitore per la barra di navigazione e le altre componenti dell'applicazione le quali sono istanziate dinamicamente all' interno del template http;
 - **Utilizzo:**
AppComponent è il primo component che viene istanziato tramite bootstrap.
- SWEDesigner::Client::Components::NavbarComponent
 - **Descrizione:**
Questo component permette la navigazione all'interno dell'applicazione tramite links;
 - **Utilizzo:**
NavbarComponent è istanziato per bootstrap subito dopo dell'AppComponent
- SWEDesigner::Client::Components::RegistrationComponent
 - **Descrizione:**
È il componente che descrive la pagina di registrazione dell'applicazione, mette a disposizione dell'utente un form dove inserire le informazioni necessarie alla creazione di un nuovo account utente. Gestisce le operazioni e la logica applicativa per la registrazione servendosi dei metodi forniti dal servizio AuthenticationService;
 - **Utilizzo:**
Questo componente viene istanziato dinamicamente dal servizio Router del framework Angular quando viene richiesta la pagina di registrazione.
- SWEDesigner::Client::Components::LoginComponent
 - **Descrizione:**
È il componente che descrive la pagina di login dell'applicazione, mette a disposizione dell'utente un form dove inserire username e password. Gestisce le operazioni e la logica applicativa per il login servendosi dei metodi forniti dal servizio AuthenticationService;
 - **Utilizzo:**
Questo componente viene istanziato dinamicamente dal servizio Router del framework Angular quando viene richiesta la pagina di login.

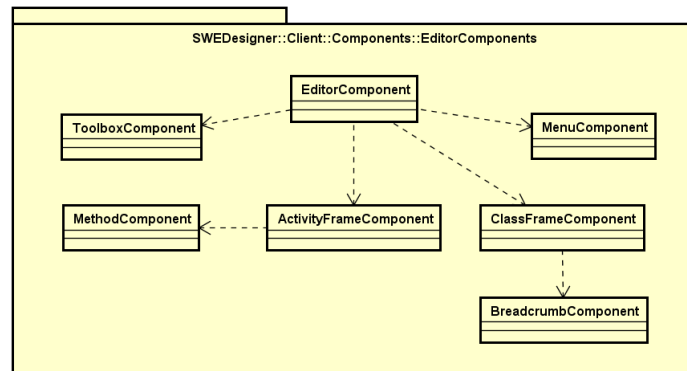


Figura 10: Diagramma dei packages SWEDesigner::Client::Components::EditorComponents

5.1.3 SWEDesigner::Client::Components::EditorComponents

5.1.3.1 Informazioni sul Package

- **Descrizione:**
Il package contiene tutte le components riguardanti l'editor dei diagrammi.
- **Padre:**
SWEDesigner::Client::Components

5.1.3.2 Informazioni sulle Classi

- SWEDesigner::Client::Components::EditorComponents::ToolboxComponent
 - **Descrizione:**
ToolboxComponent descrive il menu dal quale l'utente può selezionare gli strumenti per disegnare i diagrammi all'interno degli appositi frame. Si occupa delle operazioni e della parte logica, riguardante la costruzione dei diagrammi, servendosi dei metodi forniti dal servizio ProjManagService e della API della libreria grafica;
 - **Utilizzo:**
MenuComponent componente viene istanziato per bootstrap dopo che è stato istanziato il component EditorComponent.
- SWEDesigner::Client::Components::EditorComponents::EditorComponent
 - **Descrizione:**
Component che definisce la pagina dell'editor di diagrammi, contiene all'interno del template html gli attributi ToolboxComponent, MenuComponent, ClassFrameComponent, ActivityFrameComponent;

- **Utilizzo:**
Questo componente viene istanziato dinamicamente dal servizio Router del framework Angular quando viene richiesta la pagina dell'editor diagrammi.
- SWEDesigner::Client::Components::EditorComponents::MenuComponent
 - **Descrizione:**
Component che descrive il menu dell'edito dei diagrammi. Il menu dà la possibilità all'utente di salvare, esportare, compilare o uscire dal progetto visualizzato nell'editor dei diagrammi. Si occupa delle operazioni e della logica applicativa per le operazioni, precedentemente elencate, servendosi dei metodi messi a disposizione dai servizi contenuti nel package ProjectServices;
 - **Utilizzo:**
MenuComponent componente viene istanziato per bootstrap dopo che è stato istanziato il component EditorComponent.
- SWEDesigner::Client::Components::EditorComponents::MethodComponent
 - **Descrizione:**
MethodComponent descrive il frame dove l'utente può disegnare gli activity diagrams per definire i metodi di una classe disegnata nel frame del diagramma delle classi. Si occupa delle operazioni e logica applicativa per la generazione dei diagrammi dei metodi, per fare questo si serve dell'API della libreria grafica;
 - **Utilizzo:**
Questo component viene istanziato per bootstrap dopo l'istanziamento del componente ClassFrameComponent.
- SWEDesigner::Client::Components::EditorComponents::ActivityFrameComponent
 - **Descrizione:**
Component che descrive la struttura del frame dove l'utente può visualizzare l'activity frame che rappresenta il flusso logico del programma. Si occupa delle operazioni che portano alla corretta visualizzazione del diagramma servendosi dell'API della libreria grafica;
 - **Utilizzo:**
Questo component viene istanziato per bootstrap dopo l'istanziamento del component EditorComponent.
- SWEDesigner::Client::Components::EditorComponents::ClassFrameComponent
 - **Descrizione:**
ClassFrameComponent descrive la struttura del frame dove l'utente può generare, modificare e cancellare le classi che compongono il suo progetto. Si occupa delle operazioni che portano alla corretta visualizzazione del diagramma servendosi dell'API della libreria grafica;

- **Utilizzo:**
Questo component viene istanziato per bootstrap dopo l'istanziamento del component EditorComponent.
- SWEDesigner::Client::Components::EditorComponents::BreadcrumbComponent
 - **Descrizione:**
Component che facilita la navigazione all'interno del ActivityFrameComponent;
 - **Utilizzo:**
BreadcrumbComponent component viene istanziato per bootstrap dopo l'istanziamento del component ActivityFrameComponent.

5.1.4 SWEDesigner::Client::Components::DashboardComponents

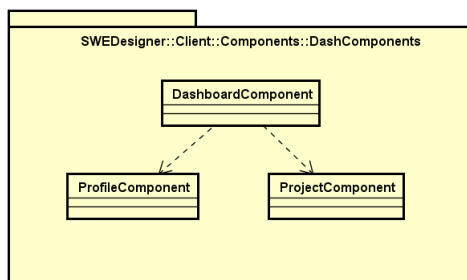


Figura 11: Diagramma dei packages SWEDesigner::Client::Components::DashComponents

5.1.4.1 Informazioni sul Package

- **Descrizione:**
Il package contiene tutti i components riguardanti la gestione del profilo e dei progetti dell'utente.
- **Padre:**
SWEDesigner::Client::Components

5.1.4.2 Informazioni sulle Classi

- SWEDesigner::Client::Components::DashComponents::DashComponent
 - **Descrizione:**
Component che definisce la pagina dove l'utente è reindirizzato dopo aver effettuato il login, contiene all'interno del template html gli elementi per l'istanziamento del ProfileComponent e del ProjectComponent.

- **Utilizzo:**
Questo componente viene istanziato dinamicamente dal servizio Router del framework Angular subito dopo aver effettuato il login.
- SWEDesigner::Client::Components::DashComponents::ProfileComponent
 - **Descrizione:**
ProfileComponent contiene le informazioni personali dell'utente. Si occupa delle operazioni che permettono il recupero dei dati utente servendosi di metodi forniti dal servizio AuthenticationService.
 - **Utilizzo:**
Questo componenet viene istanziato per bootstrap dopo l'istanziamento del component DashComponent.
- SWEDesigner::Client::Components::DashComponents::ProjectComponent
 - **Descrizione:**
ProfileComponent contiene la lista dei progetti salvati dall'utente, e fornisce dei metodi per la modifica e la cancellazione dei progetti. Si occupa delle operazioni che permettono il recupero delle informazioni sui progetti servendosi di metodi forniti dal servizio ProjManagService.
 - **Utilizzo:**
Questo componenet viene istanziato per bootstrap dopo l'istanziamento del component DashComponent.

5.1.5 SWEDesigner::Client::Service

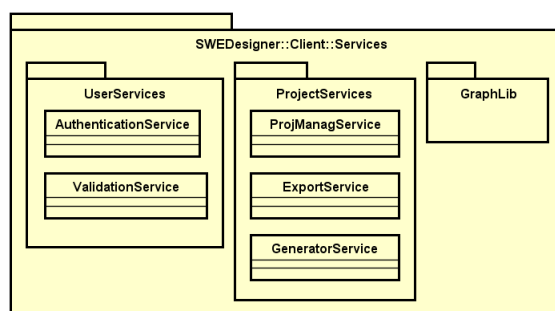


Figura 12: Diagramma dei packages SWEDesigner::Client::Service

5.1.5.1 Informazioni sul Package

- **Descrizione:**
Il package contiene i servizi per le operazioni con la libreria grafica *draw2d* e con il server.
- **Padre:**
SWEDesigner::Client
- **Package contenuti:**
 - SWEDesigner::Client::Services::UserServices;
 - SWEDesigner::Client::Services::ProjectServices;
 - SWEDesigner::Client::Services::GraphLib;

5.1.6 SWEDesigner::Client::Services::UserServices

5.1.6.1 Informazioni sul Package

- **Descrizione:**
Questo package contiene le classi che offrono servizi di registrazione, login e recupero dati utente dal server.
- **Padre:**
SWEDesigner::Client::Services

5.1.6.2 Informazioni sulle Classi

- SWEDesigner::Client::Services::UserServices::AuthenticationService
 - **Descrizione:**
Questa classe definisce i metodi di comunicazione con il server per quanto riguarda i servizi di registrazione, login e recupero dati utente;
 - **Utilizzo:**
La classe è istanziata dal framework Angular e i suoi metodi sono utilizzati dai componenti LoginComponent, RegistrationComponent e ProfileComponent.
- SWEDesigner::Client::Services::UserServices::ValidationService
 - **Descrizione:**
ValidationService descrive dei metodi per verificare la validità degli input operati dall'utente;
 - **Utilizzo:**
Essa è istanziata dal framework Angular e i suoi metodi sono utilizzati dal component RegistrationComponent.

5.1.7 SWEDesigner::Client::Services::ProjectServices

5.1.7.1 Informazioni sul Package

- **Descrizione:**
Questo package contiene i servizi inerenti alla gestione dei progetti e l'operatività dell'editor.
- **Padre:**
SWEDesigner::Client::Services

5.1.7.2 Informazioni sulle Classi

- SWEDesigner::Client::Services::ProjectServices::ProjManagService
 - **Descrizione:**
Classe che definisce le operazioni inerenti alla gestione dei progetti dell'utente;
 - **Utilizzo:**
ProjManagService è istanziata da Angular, i suoi metodi vengono utilizzati dai componenti ProjectComponent e MenuComponent.
- SWEDesigner::Client::Services::ProjectServices::ExportService
 - **Descrizione:**
ExportServices definisce le operazioni necessarie a esportare il progetto dell'utente;
 - **Utilizzo:**
Essa è istanziata dal framework Angular e i suoi metodi sono utilizzati dal componente MenuComponent.
- SWEDesigner::Client::Services::ProjectServices::GeneratorService
 - **Descrizione:**
Classe che definisce le operazioni necessarie alla compilazione del progetto dell'utente;
 - **Utilizzo:**
GeneratorService è istanziata dal framework Angular e utilizzata dai componenti MenuComponent.

6 Tracciamento

In questa sezione vengono tracciate le corrispondenze tra componenti dell'architettura e i requisiti.

- Per ogni componente principale vengono mostrati i requisiti che ne motivano l'esistenza
- Per ogni requisito principale vengono mostrate le componenti principali che servono per soddisfarlo

6.1 Tracciamento componenti - requisiti

Componente	Requisito
SWEDesigner::Client::Components::AppComponent	R0F6
SWEDesigner::Client::Components::DashComponents	R1F3 R0F5 R0F6
SWEDesigner::Client::Components::DashComponents::DashComponent	R0F6
SWEDesigner::Client::Components::DashComponents::ProfileComponent	R1F3 R1F3.1 R1F3.2 R1F3.3 R1F13 R0F5 R0F5.1 R0F5.1.1 R0F5.1.2 R0F5.2 R0F5.3
SWEDesigner::Client::Components::EditorComponents	R0F6 R0F6.1 R0F6.2 R0F6.3 R0F6.4
SWEDesigner::Client::Components::EditorComponents::ActivityFrameComponent	R0F6.4
SWEDesigner::Client::Components::EditorComponents::BreadcrumbComponent	R0F6.4.1
SWEDesigner::Client::Components::EditorComponents::ClassFrameComponent	R0F6.3.1 R0F6.2.1

Componente	Requisito
SWEDesigner::Client::Components::EditorComponents::EditorComponent	R0F6.3
SWEDesigner::Client::Components::EditorComponents::MenuComponent	R0F6.1.1 R0F6.1.1.1 R0F6.1.1.2 R0F6.1.1.3 R0F6.1.1.4 R0F6.2
SWEDesigner::Client::Components::EditorComponents::MethodComponent	R0F6.3.5 R0F6.2.2
SWEDesigner::Client::Components::EditorComponents::ToolboxComponent	R0F6.1
SWEDesigner::Client::Components::LoginComponent	R0F2 R0F2.1 R0F2.2
SWEDesigner::Client::Components::RegistrationComponent	R0F1 R0F1.1 R0F1.2 R0F1.3 R0F1.4
SWEDesigner::Client::Services::ProjectServices::ExportServices	R0F6.1.1.3
SWEDesigner::Client::Services::ProjectServices::GeneratorServices	R0F6.1.1.4
SWEDesigner::Client::Services::ProjectServices::ProjManagServices	R1F14
SWEDesigner::Client::Services::UserServices	R0F1 R0F2
SWEDesigner::Client::Services::UserServices::AuthenticationService	R0F1 R0F1.1 R0F1.2 R0F1.3 R0F1.4
SWEDesigner::Client::Services::UserServices::ValidationServices	R0F2 R0F2.1 R0F2.2
SWEDesigner::Server::Controller::Middleware::Encrypt	R0F6.1.1.3 R0F6.1.1.4

Componente	Requisito
SWEDesigner::Server::Controller::Middleware::ErrorHandler	R0F1.5 R0F1.6 R0F1.7 R0F2.3 R1F3.4 R1F3.5
SWEDesigner::Server::Controller::Middleware::Parse	R0F6.1.1.4
SWEDesigner::Server::Controller::Middleware::Profile	R0F1 R0F1.1 R0F1.2 R0F1.3 R0F2 R0F2.1 R0F2.2
SWEDesigner::Server::Controller::Services::EncryptServices	R0F6.1.1.3 R0F6.1.1.4
SWEDesigner::Server::Controller::Services::JavaGenService	R0F6.1.1.4
SWEDesigner::Server::Controller::Services::JavaGenService::DownloadService	R0F6.1.1.3 R0F6.1.1.4
SWEDesigner::Server::Controller::Services::JavaGenService::JavaGenFactory	R0F6.1.1.4
SWEDesigner::Server::Controller::Services::JavaGenService::ParseService	R0F6.1.1.4
SWEDesigner::Server::Controller::Services::UserServices	R0F1 R0F2
SWEDesigner::Server::Controller::Services::UserServices::ForgotService	R1F14
SWEDesigner::Server::Controller::Services::UserServices::ProfileService	R0F1
SWEDesigner::Server::Controller::Services::UserServices::UserService	R1f3 R1F3.1 R1F3.2 R1F3.3 R1F13

Componente	Requisito
SWEDesigner::Server::Controller::Services::UserServices::UserServicesFactory	R0F1 R0F1.1 R0F1.2 R0F1.3 R0F2 R0F2.1 R0F2.2
SWEDesigner::Server::Model::DBPROJ	R0F5 R0F5.1 R0F5.2 R0F5.3 R0F6.1.1.1 R0F6.1.1.5 R2F6.1.4.1
SWEDesigner::Server::Model::DBUSR	R0F1 R0F2
SWEDesigner::Server::Model::MongooseConnection	R0F5.2 R0F5.3 R0F6.1.1.1 R1F6.1.1.5 R1F6.1.3.2 R2F6.1.4.1 R2F6.1.4.3

Tabella 2: Tracciamento componenti - requisiti

6.2 Tracciamento requisiti - componenti

Requisito	Componente
R0F1	SWEDesigner::Client::Components::RegistrationComponent SWEDesigner::Client::Services::UserServices SWEDesigner::Client::Services::UserServices::AuthenticationService SWEDesigner::Server::Controller::Middleware::Profile SWEDesigner::Server::Controller::Services::UserServices SWEDesigner::Server::Controller::Services::UserServices::ProfileService SWEDesigner::Server::Controller::Services::UserServices::UserServicesFactory SWEDesigner::Server::Model::DBUSR

Requisito	Componente
R0F1.1	SWEDesigner::Client::Components::RegistrationComponent SWEDesigner::Server::Controller::Middleware::Profile SWEDesigner::Client::Services::UserServices::AuthenticationService SWEDesigner::Server::Controller::Services::UserServices::UserServicesFactory
R0F1.2	SWEDesigner::Client::Components::RegistrationComponent SWEDesigner::Client::Services::UserServices::AuthenticationService SWEDesigner::Server::Controller::Middleware::Profile SWEDesigner::Server::Controller::Services::UserServices::UserServicesFactory
R0F1.3	SWEDesigner::Client::Components::RegistrationComponent SWEDesigner::Client::Services::UserServices::AuthenticationService SWEDesigner::Server::Controller::Middleware::Profile SWEDesigner::Server::Controller::Services::UserServices::UserServicesFactory
R0F1.4	SWEDesigner::Client::Components::RegistrationComponent SWEDesigner::Client::Services::UserServices::AuthenticationService
R0F1.5	SWEDesigner::Server::Controller::Middleware::ErrorHandler
R0F1.6	SWEDesigner::Server::Controller::Middleware::ErrorHandler
R0F1.7	SWEDesigner::Server::Controller::Middleware::ErrorHandler
R0F2	SWEDesigner::Client::Components::LoginComponent SWEDesigner::Client::Services::UserServices SWEDesigner::Server::Controller::Middleware::Profile SWEDesigner::Client::Services::UserServices::ValidationServices SWEDesigner::Server::Controller::Services::UserServices SWEDesigner::Server::Controller::Services::UserServices::UserServicesFactory SWEDesigner::Server::Model::DBUSR
R0F2.1	SWEDesigner::Client::Components::LoginComponent SWEDesigner::Client::Services::UserServices::ValidationServices SWEDesigner::Server::Controller::Middleware::Profile SWEDesigner::Server::Controller::Services::UserServices::UserServicesFactory
R0F2.2	SWEDesigner::Client::Components::LoginComponent SWEDesigner::Client::Services::UserServices::ValidationServices SWEDesigner::Server::Controller::Middleware::Profile SWEDesigner::Server::Controller::Services::UserServices::UserServicesFactory
R0F2.3	SWEDesigner::Server::Controller::Middleware::ErrorHandler
R0F5	SWEDesigner::Client::Components::DashComponents SWEDesigner::Client::Components::DashComponents::ProjectComponent SWEDesigner::Server::Model::DBPROJ

Requisito	Componente
R0F5.1	SWEDesigner::Client::Components::DashComponents::ProjectComponent SWEDesigner::Server::Model::DBPROJ
R0F5.1.1	SWEDesigner::Client::Components::DashComponents::ProjectComponent
R0F5.1.2	SWEDesigner::Client::Components::DashComponents::ProjectComponent
R0F5.2	SWEDesigner::Client::Components::DashComponents::ProjectComponent SWEDesigner::Server::Model::DBPROJ SWEDesigner::Server::Model::MongooseConnection
R0F5.3	SWEDesigner::Client::Components::DashComponents::ProjectComponent SWEDesigner::Server::Model::DBPROJ SWEDesigner::Server::Model::MongooseConnection
R0F6	SWEDesigner::Client::Components::AppComponent SWEDesigner::Client::Components::DashComponents SWEDesigner::Client::Components::DashComponents::DashComponent SWEDesigner::Client::Components::EditorComponents
R0F6.1	SWEDesigner::Client::Components::EditorComponents SWEDesigner::Client::Components::EditorComponents::ToolboxComponent
R0F6.1.1	SWEDesigner::Client::Components::EditorComponents::MenuComponent
R0F6.1.1.1	SWEDesigner::Client::Components::EditorComponents::MenuComponent SWEDesigner::Server::Model::DBPROJ SWEDesigner::Server::Model::MongooseConnection
R0F6.1.1.2	SWEDesigner::Client::Components::EditorComponents::MenuComponent
R0F6.1.1.3	SWEDesigner::Client::Components::EditorComponents::MenuComponent SWEDesigner::Client::Services::ProjectServices::ExportServices SWEDesigner::Server::Controller::Middleware::Encrypt SWEDesigner::Server::Controller::Services::EncryptServices SWEDesigner::Server::Controller::Services::JavaGenService::DownloadService
R0F6.1.1.4	SWEDesigner::Client::Components::EditorComponents::MenuComponent SWEDesigner::Client::Services::ProjectServices::GeneratorServices SWEDesigner::Server::Controller::Middleware::Encrypt SWEDesigner::Server::Controller::Middleware::Parse SWEDesigner::Server::Controller::Services::JavaGenService SWEDesigner::Server::Controller::Services::EncryptServices SWEDesigner::Server::Controller::Services::JavaGenService::DownloadService SWEDesigner::Server::Controller::Services::JavaGenService::JavaGenFactory SWEDesigner::Server::Controller::Services::JavaGenService::ParseService

Requisito	Componente
R0F6.1.1.5	SWEDesigner::Server::Model::DBPROJ
R0F6.2	SWEDesigner::Client::Components::EditorComponents SWEDesigner::Client::Components::EditorComponents::MenuComponent
R0F6.2.1	SWEDesigner::Client::Components::EditorComponents::ClassFrameComponent
R0F6.2.2	SWEDesigner::Client::Components::EditorComponents::MethodComponent
R0F6.3	SWEDesigner::Client::Components::EditorComponents SWEDesigner::Client::Components::EditorComponents::EditorComponent
R0F6.3.1	SWEDesigner::Client::Components::EditorComponents::ClassFrameComponent
R0F6.3.5	SWEDesigner::Client::Components::EditorComponents::MethodComponent
R0F6.4	SWEDesigner::Client::Components::EditorComponents SWEDesigner::Client::Components::EditorComponents::ActivityFrameComponent
R0F6.4.1	SWEDesigner::Client::Components::EditorComponents::BreadcrumbComponent
R1F13	SWEDesigner::Client::Components::DashComponents::ProfileComponent SWEDesigner::Server::Controller::Services::UserServices::UserService
R1F14	SWEDesigner::Client::Services::ProjectServices::ProjManagServices SWEDesigner::Server::Controller::Services::UserServices::ForgotService
R1F3	SWEDesigner::Client::Components::DashComponents SWEDesigner::Client::Components::DashComponents::ProfileComponent SWEDesigner::Server::Controller::Services::UserServices::UserService
R1F3.1	SWEDesigner::Client::Components::DashComponents::ProfileComponent SWEDesigner::Server::Controller::Services::UserServices::UserService
R1F3.2	SWEDesigner::Client::Components::DashComponents::ProfileComponent SWEDesigner::Server::Controller::Services::UserServices::UserService
R1F3.3	SWEDesigner::Client::Components::DashComponents::ProfileComponent SWEDesigner::Server::Controller::Services::UserServices::UserService
R1F3.4	SWEDesigner::Server::Controller::Middleware::ErrorHandler
R1F3.5	SWEDesigner::Server::Controller::Middleware::ErrorHandler
R1F6.1.1.5	SWEDesigner::Server::Model::MongooseConnection
R1F6.1.3.2	SWEDesigner::Server::Model::MongooseConnection
R2F6.1.4.1	SWEDesigner::Server::Model::DBPROJ SWEDesigner::Server::Model::MongooseConnection

Requisito	Componente
R2F6.1.4.3	SWEDesigner::Server::Model::MongooseConnection

Tabella 3: Tracciamento requisiti - componenti

A Descrizione *Design Pattern*_G

I *Design Pattern*_G sono un modello logico da applicare per la soluzione di problemi ricorrenti. L'impiego di questi modelli rende l'architettura più manutenibile. Verranno di seguito illustrati i *Design Pattern*_G implementati nella costruzione dell'architettura di alto livello, divisi per categoria di applicazione:

A.1 *Design Pattern*_G Architetture

A.1.1 MVVM

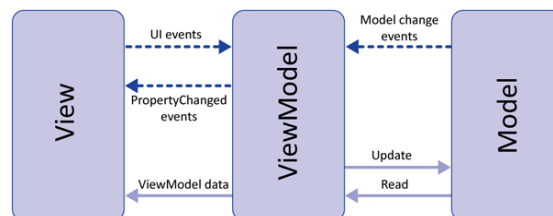


Figura 13: Diagramma del *Design Pattern*_G MVVM

- **Scopo:** Lo scopo del MVVM è quello di separare le seguenti componenti in modo da non mescolare il codice della logica con quella dell'interfaccia utente:
 - Model - rappresenta il punto di accesso ai dati. Trattasi di una o più classi che leggono dati dal *Database*_G, oppure da un servizio web di qualsivoglia natura;
 - View - rappresenta la vista dell'applicazione, l'interfaccia grafica che mostrerà i dati;
 - 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.
- **Motivazione:** MVVM è stato progettato per utilizzare le funzioni di binding dei dati in WPF (Windows Presentation Foundation) per facilitare la separazione della view dal resto del modello, rimuovendo praticamente tutti i codici *GUI*_G dal livello di visualizzazione. Invece di richiedere agli sviluppatori di *user experience* (*UX*)_G di scrivere il codice *GUI*_G, possono utilizzare il linguaggio di marcatura dei *framework*_G e creare connessioni di dati al modello di visualizzazione, che viene scritto e gestito dagli sviluppatori di applicazioni. La separazione dei ruoli consente ai progettisti interattivi di concentrarsi sulle esigenze UX anziché sulla programmazione della logica aziendale. Gli strati di un'applicazione possono quindi essere sviluppati in più flussi di lavoro per una maggiore produttività.

- **Applicabilità:** il modello MVVM è in definitiva la moderna struttura del modello MVC, quindi l'obiettivo principale è sempre lo stesso per fornire una netta separazione tra logica di dominio e livello di presentazione. Esso è applicabile nei seguenti casi:
 - Quando si vuole ottenere la vera separazione tra la view e la model oltre a conseguire la separazione e l'efficienza che si guadagna ad averla;
 - Quando si vuole codice manutenibile e estensibile.

A.1.2 Three-Tier

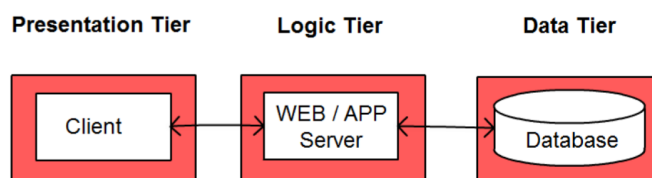


Figura 14: Diagramma del *Design Pattern_G* Three-Tier

- **Scopo:** tale *Design Pattern_G* permette una disgiunzione tra i vari gruppi di entità che cooperano nell'erogazione del servizio. Esisterà un livello che si occuperà di interagire con il cliente offrendo l'interfaccia grafica, un altro livello che gestirà di eseguire la parte algoritmica dell'applicazione e un altro livello che si occuperà di persistere i dati e recuperarli. Ogni livello comunicherà solo con i livelli adiacenti.
- **Motivazione:** si tratta di un pattern molto utilizzato nelle *applicazioni web_G* perché rende l'applicazione flessibile, riutilizzabile e scalabile. Con la separazione di un'applicazione in livelli, gli sviluppatori, per modificare o aggiungere funzionalità, possono infatti modificare solo uno specifico livello piuttosto che dover riscrivere l'intera applicazione. Ciò garantisce dunque una maggiore semplicità di progettazione/implementazione secondo la filosofia del divide et impera ed una maggiore manutenibilità.

A.2 Design Pattern_G Creazionali

A.2.1 Factory Method

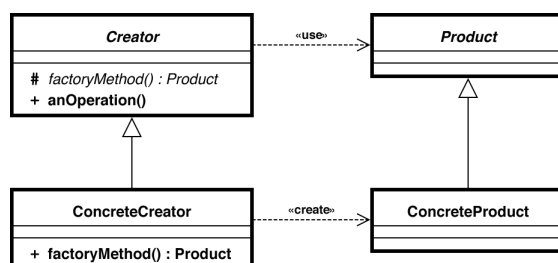


Figura 15: Diagramma del Design Pattern_G Factory method

- **Scopo:** il Design Pattern_G Factory Method indirizza il problema della creazione di oggetti senza specificarne l'esatta classe, fornendo un'interfaccia per creare un oggetto, ma lasciando che le sottoclassi decidano quale oggetto istanziare. Definisce un'interfaccia (**Creator**) per ottenere una nuova istanza di un oggetto (**Product**). Delega ad una classe derivata (**ConcreteCreator**) la scelta di quale classe istanziare (**ConcreteProduct**).
- **Motivazione:** la creazione di un oggetto può, spesso, richiedere processi complessi la cui collocazione all'interno della classe di composizione potrebbe non essere appropriata. Esso può, inoltre, comportare duplicazione di codice, richiedere informazioni non accessibili alla classe di composizione, o non fornire un sufficiente livello di astrazione. Il Factory Method indirizza questi problemi definendo un metodo separato per la creazione degli oggetti. Tale metodo può essere ridefinito dalle sottoclassi per definire il tipo derivato di prodotto che verrà effettivamente creato.
- **Applicabilità:** tale Design Pattern_G verrà utilizzato nei seguenti casi:
 - si desidera che la creazione di un oggetto non precluda il suo riuso senza una significativa duplicazione di codice;
 - si desidera che la creazione di un oggetto non richieda l'accesso ad informazioni o risorse che non dovrebbero essere contenute nella classe di composizione;
 - si desidera che la gestione del ciclo di vita degli oggetti gestiti debba essere centralizzata in modo da assicurare un comportamento consistente all'interno dell'applicazione.

A.3 Design Pattern_G Strutturali

A.3.1 Decorator

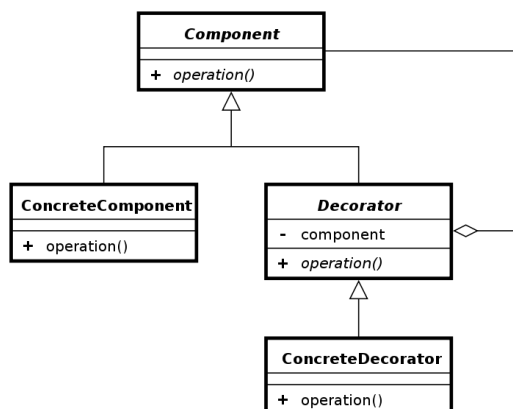


Figura 16: Diagramma del Design Pattern_G Decorator

- **Scopo:** aggiungere dinamicamente responsabilità a un oggetto. I Decorator forniscono un'alternativa flessibile alla definizione di sottoclassi come strumento per l'estensione delle funzionalità;
- **Motivazione:** talvolta si vogliono aggiungere responsabilità a singoli oggetti e non a un'intera classe. Un modo per aggiungere responsabilità consiste nel racchiudere il componente da decorare in un altro. L'oggetto contenitore è chiamato Decorator. Il Decorator ha un'interfaccia conforme a quella dell'elemento decorato, in modo da rendere trasparente la sua presenza ai client. Il decorator trasferisce le richieste al componente decorato e può svolgere azioni aggiuntive prima o dopo il trasferimento della richiesta;
- **Applicabilità:** il pattern Decorator può essere utilizzato nei seguenti casi:
 - Si vuole poter aggiungere responsabilità a singoli oggetti dinamicamente ed in modo trasparente;
 - Si vuole poter togliere responsabilità agli oggetti;
 - Si vuole definire un gran numero di estensioni indipendenti.

A.3.2 Facade

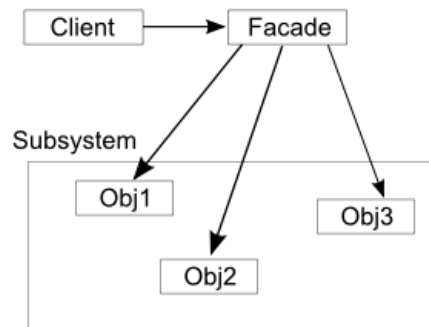


Figura 17: Diagramma del *Design Pattern_G* Facade

- **Scopo:** fornire un'interfaccia unificata per un insieme di interfacce presenti in un sottosistema. Definisce un'interfaccia di livello più alto che rende il sottosistema più semplice da utilizzare;
- **Motivazione:** suddividere un sistema in sottosistemi aiuta a ridurre la complessità. Un obiettivo comune di progettazione è la minimizzazione delle comunicazioni e delle dipendenza fra i diversi sottosistemi. Un modo per raggiungere questo obiettivo è introdurre un oggetto facade, che fornisce un'interfaccia unica e semplificata per accedere alle funzionalità offerte da un sottosistema;
- **Applicabilità:** il pattern Facade può essere utilizzato nei seguenti casi:
 - Quando si vuole fornire un'interfaccia semplice a un sottosistema complesso poiché fornisce una vista semplice di base su un sottosistema che si rivela essere sufficiente per la maggior parte dei client;
 - Nei casi in cui ci sono molte dipendenze fra i client e le *classi_G* che implementano un'astrazione in quanto si disaccoppia il sottosistema dai client e dagli altri sistemi, promuovendo portabilità e indipendenza dei sottosistemi;
 - Quando si vogliono organizzare i sottosistemi in una struttura a livelli.

A.4 Design Pattern_G Comportamentali

A.4.1 Dependency Injection

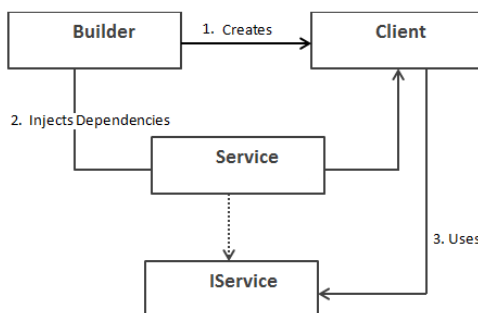


Figura 18: Diagramma del Design Pattern_G Dependency Injection

- **Scopo:** Il Dependency Injection è un Design Pattern_G che permette la separazione del comportamento degli oggetti dalla loro dipendenze. Invece di istanziare le classi_G in modo diretto ogni componente riceve i riferimenti agli altri componenti necessari come parametri nel costruttore. Un utilizzo comune è quello con i plugin che vengono caricati dinamicamente. Gli elementi coinvolti sono:
 - Un dipendente consumatore;
 - Una dichiarazione delle dipendenze tra la componenti, definita come contratto di un interfaccia;
 - Un injector che crea istanze di classi_G che implementano una data dipendenza su richiesta.

Il dependent object dichiara da quali componenti dipende. L'injector decide quali classi_G soddisfano suoi requisiti e in caso affermativo gliele fornisce. Questa operazione può avvenire anche a runtime. Questo è un chiaro vantaggio poiché possono essere create dinamicamente diverse implementazioni di un componente software da passare allo stesso test. In questo modo il test può testare componenti diverse senza sapere che le loro implementazioni sono diverse.

- **Motivazione:** lo scopo principale di questo pattern è quello di permettere una selezione a runtime su più implementazioni di una interfaccia dipendente. È particolarmente utile per fornire delle implementazioni di stub per componenti complesse, ma anche per gestire i plugin e per inizializzare servizi software. I test di unità comportano delle problematiche, poiché spesso richiedono la presenza di una parte di

infrastruttura non ancora implementata. Il Dependency Injection semplifica il processo di testing per un'istanza isolata. Poiché le componenti dichiarano le proprie dipendenze, un test può automaticamente istanziare le componenti necessarie.

- **Applicabilità:** Di seguito vengono elencati tre modi con cui un oggetto può ricevere un riferimento da un modulo esterno:
 - Interface injection: l'oggetto fornisce un'interfaccia che gli utenti possono implementare in modo da ottenere a runtime le dipendenze;
 - Setter injection: il dependent module espone un metodo setter che il frameworkG usa per iniettarvi le dipendenze;
 - Constructor injection: le dipendenze vengono fornite tramite il costruttore della classe.

A.4.2 Command

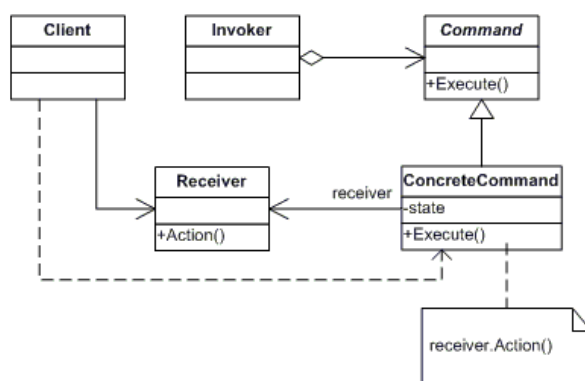


Figura 19: Diagramma del *Design Pattern_G* Command

- **Scopo:** Incapsula una richiesta in un oggetto, consentendo di parametrizzare il client con richieste diverse, accodare o mantenere uno storico delle richieste e gestire richieste cancellabili;
- **Motivazione:** Talvolta è necessario inoltrare richieste a oggetti senza conoscere nulla dell'operazione richiesta o del destinatario della richiesta. Il pattern Command permette agli oggetti dell'ambiente di inoltrare richieste a oggetti sconosciuti dell'applicazione trasformando la richiesta in un oggetto;
- **Applicabilità:** Il pattern Command può essere utilizzato nei seguenti casi:
 - Per parametrizzare gli oggetti rispetto a un'azione da compiere;

- Per specificare, accordare ed eseguire le richieste in tempi diversi;
- Per consentire l'annullamento di operazioni;
- Per organizzare un sistema in operazioni d'alto livello a loro volta basate su operazioni primitive.