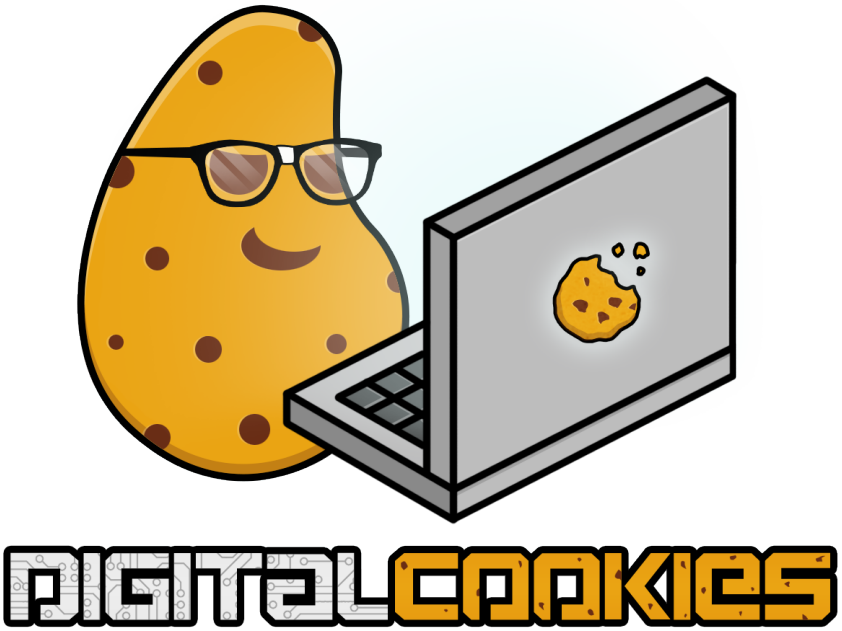

Specifica Tecnica

Gruppo DigitalCookies — Progetto SWEDesigner

digitalcookies.group@gmail.com

Informazioni sul documento

Versione	1.0.0
Redazione	Carlo Sindico, Chrisian Cabrera, Alessia Bragagnolo, Alberto Rossetti, Saverio Follador
Verifica	Davide Albertini, Carlo Sindico
Approvazione	Alberto Giudice
Uso	Esterno
Distribuzione	Prof. Tullio Vardanega Prof. Riccardo Cardin Gruppo DigitalCookies

Descrizione

Questo documento descrive la progettazione architetturale delle componenti definita dal gruppo DigitalCookies relativa al progetto SWEDesigner.

Registro delle modifiche

Versione	Data	Collaboratori	Ruolo	Descrizione
1.0.0	08-05-2017	Alberto Giudice	Responsabile	Approvazione
0.3.0	08-05-2017	Davide Albertini	Verificatore	Verifica del documento
0.2.1	05-05-2017	Christian Cabrera	Progettista	Modificati diagrammi di sequenza in sezione 8
0.2.0	03-05-2017	Carlo Sindico	Verificatore	Verifica sezioni 5, 6, 7, 9, 10.1, 10.2 e appendice A
0.1.0	02-05-2017	Davide Albertini	Verificatore	Verifica sezioni 1, 2, 3 e 4
0.0.13	01-05-2017	Christian Cabrera	Progettista	Stesura sezioni relative ai tracciamenti
0.0.12	28-04-2017	Saverio Follador	Progettista	Stesura sezioni relative ai diagrammi delle attività e al database
0.0.11	28-04-2017	Saverio Follador	Progettista	Stesura sezione relativa ai design pattern utilizzati e appendice con descrizione di tali design pattern
0.0.10	27-04-2017	Alberto Rossetti	Progettista	Stesura sezione relativa al Data Tier del Back-End
0.0.9	27-04-2017	Alberto Rossetti	Progettista	Stesura sezione relativa all'Application Tier del Back-End

0.0.8	26-04-2017	Alberto Rossetti	Progettista	Stesura sezione relativa al Presentation Tier del Back-End
0.0.7	26-04-2017	Alessia Bragagnolo	Progettista	Stesura sezione relativa al ViewModel del Front-End
0.0.6	25-04-2017	Alessia Bragagnolo	Progettista	Stesura sezione relativa al Model del Front-End
0.0.5	25-04-2017	Alessia Bragagnolo	Progettista	Stesura sezione relativa alla View del Front-End
0.0.4	24-04-2017	Christian Cabrera	Progettista	Stesura sezione Descrizione architetturale
0.0.3	24-04-2017	Carlo Sindico	Progettista	Stesura sezione Tecnologie utilizzate
0.0.2	24-04-2017	Carlo Sindico	Progettista	Stesura sezione introduzione
0.0.1	24-04-2017	Alberto Rossetti	Amministratore	Creazione del template

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

1.1 Scopo del documento

Il presente documento ha lo scopo di definire la progettazione ad alto livello del prodotto software SWEDesigner.
In particolare vengono descritti in dettaglio i package, le classi e le interfacce, concludendo con il tracciamento tra le componenti software ed i requisiti analizzati nell'*Analisi dei Requisiti v2.0.0*; vengono inclusi inoltre i *design pattern*$_G$ utilizzati.

1.2 Scopo del prodotto

Lo scopo del *prodotto*$_G$ è creare un software di costruzione di diagrammi *UML*$_G$ con relativa generazione di codice *Java*$_G$. Il codice potrà essere generato dall'utente a partire dai diagrammi UML delle *classi*$_G$ e da una versione modificata del diagramma delle *attività*$_G$.

L'utente, interagendo con il sistema, sarà in grado di:

- delineare la struttura delle classi utilizzando lo standard UML;
- definire il corpo dei metodi delle classi sfruttando una versione modificata del diagramma delle attività;
- generare un applicativo scritto in codice Java a partire dai diagrammi sopracitati.

L'utente potrà inoltre sfruttare la *libreria*$_G$ fornita con il prodotto per generare con facilità diagrammi relativi al dominio dei giochi di carte.

L'*editor*$_G$ sarà fruibile dall'utente attraverso un *browser*$_G$ desktop idoneo all'utilizzo delle tecnologie *HTML5*$_G$, *CSS3*$_G$ e *JavaScript*$_G$.

1.3 Ambiguità

Al fine di evitare ogni ambiguità relativa al linguaggio impiegato nei documenti viene fornito il *Glossario v2.0.0*, contenente la definizione dei termini in corsivo marcati con una G pedice.

1.4 Riferimenti

1.4.1 Normativi

- ***Norme di Progetto v2.0.0***;

- ***Analisi dei Requisiti v2.0.0***.
- **Capitolato d'appalto C6: SWEDesigner editor di diagrammi UML con generazione di codice:** http://www.math.unipd.it/~tullio/IS-1/2016/Progetto/C6p.pdf (ultima consultazione effettuata in data 24-04-2017).
- **Verbale di incontro interno** con i componenti del gruppo del 21-04-2017;
- **Verbale di incontro interno** con i componenti del gruppo del 27-04-2017;
- **Verbale di incontro esterno** con i componenti del gruppo e il proponente Zucchetti S.p.A. del 04-05-2017.

1.4.2 Informativi

- **Design Patterns** - Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides - 1a edizione italiana (2006) (sezioni §3, §4);
- **Martin Fowler - UML Distilled** - 2nd edition (sezioni §4, §5);
- **Interfaccia REST**: https://it.wikipedia.org/wiki/Representational_State_Transfer (ultima consultazione effettuata in data 24-04-2017);
- **GoJS**: Interactive JavaScript Diagrams in HTML, OpenSource https://gojs.net/latest/index.html (ultima consultazione effettuata in data 24-04-2017);
- **Express**: http://expressjs.com/it/guide/using-middleware.html (ultima consultazione avvenuta in data 26-04-2017)
- **RequireJS** http://requirejs.org/docs/api.html (ultima consultazione effettuata in data 24-04-2017)
- **Angular2**: JavaScript framework http://www.angular2.com/ (ultima consultazione effettuata in data 27-04-2017);
- **Node.js**: https://nodejs.org/en/docs/guides/ (ultima consultazione effettuata in data 30-04-2017);
- **Mongodb**: https://docs.mongodb.com/manual/introduction/ (ultima consultazione effettuata in data 27-04-2017);
- **Mongoose**: http://mongoosejs.com/docs/api.html (ultima consultazione effettuata in data 27-04-2017);

2 Tecnologie utilizzate

2.1 Client

Il *Client*$_G$ di SWEDesigner verrà implementato con le tecnologie web richieste dal capitolato d'appalto: HTML5, CSS3 e JavaScript. Il Client ha bisogno di alcune librerie JavaScript open-source, come richiesto dal capitolato.

2.1.1 GoJS

GoJS è una libreria JavaScript ricca di funzionalità, in quanto permette di realizzare diagrammi interattivi in maniera altamente personalizzabile. Tra le funzionalità più note troviamo:

- creazione di elementi per diagrammi base (e.g. rettangoli, cerchi, commenti);
- creazione di elementi custom basati su *SVG*$_G$;
- creazione collegamenti personalizzabili (punta, etichette);
- inserimento, modifica di immagini;
- altamente *event-driven*$_G$;
- stampa dei diagrammi prodotti;
- zoom in/out;
- serializzazione/deserializzazione con file JSON.

. La libreria sfrutta le seguenti tecnologie per il suo funzionamento:

- *HTML5*: *GoJS* richiede che una pagina HTML5 sia popolata con un tag , che conterrà un diagramma.

GoJS, per il suo normale utilizzo, non richiede alcuna libreria JavaScript o framework.

2.1.2 Angular2

Angular2 è un framework JavaScript, patrocinato da *Google*$_G$, utile a semplificare la realizzazione di applicazioni Web single page (*SPA*$_G$): favorisce un approccio dichiarativo allo sviluppo client-side, migliore per la creazione di interfacce utente, laddove l'approccio imperativo è ideale per realizzare la logica applicativa.

2.1.2.1 Vantaggi

- ***Data-Binding***$_G$ **Bidirezionale:** il data-binding bidirezionale di Angular2 gestisce la sincronizzazione tra il *DOM*$_G$ e il modello, e viceversa;
- ***MVVM***$_G$**:** Angular2 permette una facile implementazione lato client del pattern MVVM;
- ***Dependency injection***$_G$**:** Angular2 permette di descrivere in maniera dichiarativa quali sono le dipendenze che l'applicazione possiede, isolando i comportamenti e le responsabilità dei componenti e garantendo un facile rimpiazzo di quest'ultimi. Questo meccanismo favorisce inoltre la testabilità del codice dell'applicazione.

2.1.2.2 Svantaggi

- Il ciclo di vita di un'applicazione Angular2 è complesso. La compilazione e il link non sono intuitivi e in specifici casi possono essere confusi (ricorsione in compilazione, collisioni tra direttive).

2.1.3 RequireJS

La libreria *RequireJS* è un loader di moduli e file *JavaScript*. L'obiettivo principale di *RequireJS* è quello di risolvere le dipendenze delle librerie JavaScript utilizzate.

2.1.3.1 Vantaggi

- Esplicita le dipendenze tra i diversi moduli;
- Caricamento dei moduli asincrono.

2.1.3.2 Svantaggi

- Lo svantaggio principale risiede nella necessità di uno studio completo e approfondito della libreria prima di un utilizzo efficace.

2.1.4 HTML5

L'*HTML5* è un linguaggio di markup per la strutturazione delle pagine web, pubblicato come *W3C*$_G$ *Recommendation* da ottobre 2014. Le novità introdotte dall'*HTML5* rispetto all'HTML 4 sono finalizzate soprattutto a migliorare il disaccoppiamento fra struttura, definita dal markup, caratteristiche di resa (tipo di carattere, colori, eccetera), definite dalle direttive di stile, e contenuti di una pagina web, definiti dal testo vero e proprio. Inoltre l'*HTML5* prevede il supporto per la memorizzazione locale di grandi quantità di dati scaricati dal web browser, per consentire l'utilizzo di applicazioni basate

su web (come per esempio le caselle di posta di *Google* o altri servizi analoghi) anche in assenza di collegamento a Internet.

2.1.4.1 Vantaggi

- **Standard:** *HTML5* è uno standard *W3C*.

2.1.4.2 Svantaggi

- **Scarso supporto ai vecchi browser:** *HTML5* non è supportato dai browser più datati, per rimediare tali mancanze, sarà necessario utilizzare altre tecnologie di supporto.

2.1.5 CSS

CSS (Cascading Style Sheets) è un linguaggio utile a descrivere la presentazione di un documento scritto in un linguaggio di markup; uno degli scopi principali che hanno spinto alla creazione di questo linguaggio è la necessità di separare la struttura del documento dalla sua presentazione rendendo cosi maggiormente accessibili le pagine dei siti web.

2.1.5.1 Vantaggi

- **Standard:** CSS è uno standard *W3C*;
- **Accessibilità:** la separazione tra struttura e presentazione consente di avere pagine molto più accessibili rispetto ad avere tutto in un unico file.

2.1.5.2 Svantaggi

- **Comportamento imprevedibile**: il comportamento delle regole CSS potrebbe variare a seconda dei browser utilizzati.

2.2 Server

2.2.1 Amazon Web Service

Il *back-end* della nostra web-application è ospitato su un $server_G$ AWS_G.

2.2.2 Node.js

Node.js è un framework event-driven basato sul motore *JavaScript* V8 di Google *Chrome*$_G$, per piattaforme *UNIX*$_G$. Node.js usa un modello I/O non bloccante e ad eventi, risulta quindi framework leggero ed efficiente per l'utilizzo server-side.

2.2.2.1 Vantaggi

- **Approccio asincrono**: grazie alla modalità event-driven node.js evita il modello basato su processi o *thread*$_G$ concorrenti utilizzato dai classici web server. Questa caratteristica permette una migliore efficienza in termini di prestazioni, sfruttando la gestione di altri processi durante le attese in maniera asincrona.
- **Architettura modulare**: node.js si appoggia allo strumento node package manager (*npm*$_G$) che permette un'organizzazione semplice del lavoro grazie alla combinazione di librerie con moduli importati. Infatti npm permette allo sviluppatore di accedere ai package messi a disposizione dalla community.

2.2.2.2 Svantaggi

- **Modello di programmazione asincrono**: il programmatore che utilizza per la prima volta node.js deve adottare uno stile di programmazione asincrono, il quale risulta essere più complicato della programmazione di input e output lineare a blocchi; il codice prodotto risulta essere più disordinato e meno comprensibile e costringe lo sviluppatore a fare affidamento su *callback*$_G$ nidificate;

Le applicazioni Node.js vengono eseguite su un singolo thread, sebbene sia utilizzato un modello multi-thread per la gestione degli eventi legati a file e connessioni di rete.

2.2.3 Express

Express è un framework minimale per creare applicazioni web con Node.js. Express offre funzionalità che semplificano e aumentano le potenzialità di Node.js, fornendo una migliore implementazione del sistema di *routing*$_G$. Ciò è ottenuto incrementando le funzioni di richiesta e risposta, estendendole per una maggior flessibilità, integrando nuovi *middleware*$_G$ e agevolando la realizzazione delle viste.

Express non limita l'utente nella scelta del linguaggio di templating, lo aiuta a gestire le *route*$_G$, le *request*$_G$ e le *view*$_G$.

2.2.4 MongoDB

MongoDB$_G$ è un database *NoSQL*$_G$ open-source scalabile e altamente performante di tipo *document-oriented*$_G$, in cui i dati sono archiviati sotto forma di documenti in stile *JSON*$_G$ con schemi dinamici che MongoDB chiama *BSON*$_G$, secondo una struttura semplice e potente.

2.2.4.1 Vantaggi

- **Alte performance**: non ci sono *join*$_G$ 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;
- **Affidabilità**: alto meccanismo di replicazione su server;
- **Schemaless**: non esiste nessuno schema, è più flessibile e può essere facilmente trasposto in un modello ad oggetti;
- **Map-Reduce**: Permette di processare parallelamente i dati.

2.2.4.2 Svantaggi

- **Assenza di Join**: non è possibile creare delle join come nei database relazionali, il che significa che quando si necessita di tale funzionalità bisogna creare più query e fare il join dei dati manualmente con il codice. Quest'aspetto porta a creare codice non flessibile, in quanto le strutture dei dati potrebbero cambiare;
- **Utilizzo della memoria**: MongoDB tende ad utilizzare più memoria di quella necessaria, in quanto deve salvare il nome della chiave per ogni documento.

Altre funzionalità comprendono la possibilità di creare delle query ad hoc, l'Auto-Sharding, ovvero la capacità di scalare orizzontalmente e di aggiungere nuove macchine al database operativo. Inoltre, MongoDB supporta la definizione di *collection*$_G$ con una dimensione fissata. Questo particolare tipo di collection mantiene l'ordine di inserimento e, una volta raggiunta la dimensione massima prefissata, si comporta come una lista circolare.

2.2.5 Mongoose

Mongoose è una libreria per interfacciarsi a MongoDB che permette di definire degli schemi per modellare i dati del database, imponendo una certa struttura per la creazione di nuovi *Document*$_G$. Inoltre, fornisce molti strumenti utili per la validazione dei dati, per la definizione di query e per il cast dei tipi predefiniti.

2.2.5.1 Vantaggi

- La documentazione di *Mongoose* è ben fornita e descrive le API_G in maniera completa, fornendo degli $snippet_G$ del codice sorgente.

3 Descrizione architetturale

3.1 Metodo e formalismo di specifica

Per una migliore rappresentazione dell'architettura di SWEDesigner abbiamo deciso di organizzarne la descrizione in quattro sezioni:

- 3.2, illustra l'architettura del software ad alto livello;
- 3.3, illustra l'architettura dell'applicazione front-end;
- 3.4, illustra l'architettura dell'applicazione back-end;
- 3.5, illustra il protocollo di comunicazione tra le due applicazioni precedenti.

L'approccio alla progettazione scelto è stato di tipo *meet-in-the-middle*$_G$: da un lato l'approccio top-down è stato utilizzato per i diagrammi dei packages, mentre l'approccio bottom-up è stato utilizzato per il diagramma delle attività e di sequenza. Per tutti i diagrammi sono stati seguiti i formalismi di UML 2.0, evidenziando in essi con colore arancione le parti dell'architettura facenti parte di librerie esterne.

3.2 Architettura generale

L'architettura di SWEDesigner è suddivisa in due applicazioni distinte, ma che si relazionano tra di loro:

- **front-end**: editor UML fruibile da browser con cui si interfaccia l'utente;
- **back-end**: applicazione su server che fornisce al front-end la pagina iniziale di SWEDesigner, permette di generare codice, partendo da file JSON, e di restituire tale codice e i template contenuti nella libreria su database.

3.3 Front-end

Dal lato front-end SWEDesigner è pensato come una Single Page Application scritta nei linguaggi HTML5, CSS3 e JavaScript. L'architettura utilizzata per tale scopo prevede l'uso di un pattern MVVM, per uniformarsi all'uso di Angular2, composto da:

- **Model**: si occupa della parte logica della creazione dei diagrammi e dell'iterazione con l'applicazione back-end per le richieste di template o di generazione di codice;
- **View**: si occupa dell'implementazione dell'interfaccia grafica;
- **ViewModel**: si occupa di far interagire model e view per la creazione dei diagrammi.

In particolare per quanto riguarda la view l'intestazione *head* della pagina HTML conterrà riferimenti a:

- uno script JavaScript per risolvere le dipendenze tra le varie librerie JavaScript utilizzate sfruttando RequireJS;
- fogli di stile CSS di librerie esterne;
- fogli di stile CSS della pagina.

Il corpo *body* si compone invece di quattro blocchi che vanno a identificare una barra di navigazione ancorata in alto, un riquadro centrale contenente i diagrammi attualmente visualizzati e due pannelli laterali contenenti strumenti e utilità per interagire coi diagrammi.

In particolare tramite il front-end l'utente potrà andare a:

- creare diagrammi delle classi che seguono lo standard UML 2.0, con in aggiunta la possibilità di caricare classi predefinite da libreria, di assegnare priorità, colori per applicare dei filtri e ottenere una maggior chiarezza visiva;
- creare diagrammi delle attività, revisionati rispetto allo standard e strutturati a blocchi componibili. Tali blocchi prevedono costrutti tipici dei linguaggi di programmazione, con la possibilità in alcuni di essi di includere delle sotto strutture a blocchi annidate;
- generare codice a partire dai diagrammi di classi e attività e visualizzarlo in un editor online con possibilità di scaricarlo come file.

3.4 Back-end

Dal lato back-end abbiamo deciso di gestire un server in Node.JS tramite il framework Express, con anche l'interazione con un database MongoDB sfruttando la libreria Mongoose. L'architettura prevede un pattern three-tier, che va a scomporre l'applicazione in tre livelli comunicanti.

Tra i vantaggi derivanti da questo stile architetturale troviamo:

- **flessibilità**: ogni componente è progettata per essere estensibile al variare di requisiti e tecnologie;
- **distribuzione**: i tre livelli possono risiedere su macchine distinte;
- **manutenibilità**: la divisione in tre livelli di comunicazione garantisce che ogni modifica alla logica di un singolo livello non abbia ripercussioni sull'intero sistema.

In particolare la suddivisione prevede:

- **PresentationTier**: si occupa della gestione della ricezione e dell'invio dei file JSON tramite middleware, implementato grazie ad Express;
- **ApplicationTier**: si occupa della manipolazione dei file JSON per la generazione di codice e della gestione degli errori e delle richieste effettuate al database;
- **DataTier**: si occupa del recupero dei dati di libreria interfacciandosi con un database MongoDB, grazie alla libreria Mongoose.

3.5 Protocollo di comunicazione client-server

La comunicazione tra front-end e back-end aderisce il più possibile allo stile architetturale REST. Tale scelta è motivata dai seguenti vantaggi:

- semplicità di utilizzo;
- facile interazione con i framework da noi utilizzati;
- indipendenza dalle piattaforme software.

Le scelte effettuate per aderire ai principi REST prevedono:

- utilizzo del metodo HTTP GET per gestire la richiesta della pagina HTML, per rendere la pagina cacheable;
- utilizzo del metodo HTTP GET per gestire la richiesta dei dati di libreria;
- utilizzo del metodo HTTP POST per lo scambio dei file JSON, per permetterne la persistenza nel server e per aumentare l'estensibilità del prodotto;
- disaccoppiamento dei servizi di risposta alla richiesta della pagina HTML, recupero dei template forniti e della generazione di codice, con lo scopo di aumentare la manutenibilità;

L'uso dell'interfaccia REST garantisce inoltre l'idempotenza dei metodi GET e POST utilizzati.

Risorsa	Metodo	Utilizzo
/index.html	GET	Utilizzata dal client per richiedere al server la single page app
/template/id	GET	Utilizzata dal client per richiedere al server il file JSON relativo al template di codice identificativo `id`
/code	POST	Utilizzata dal client per richiedere al server il codice generato a partire dal file JSON fornito con la richiesta

Tabella 2: Risorse e metodi REST utilizzati

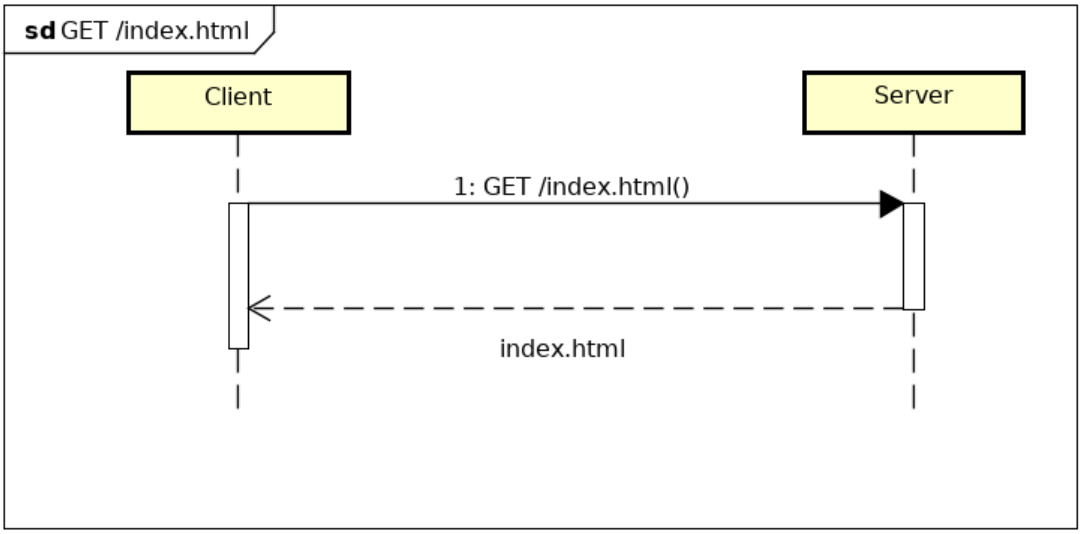

Figura 1: Diagramma di sequenza GET /index.html

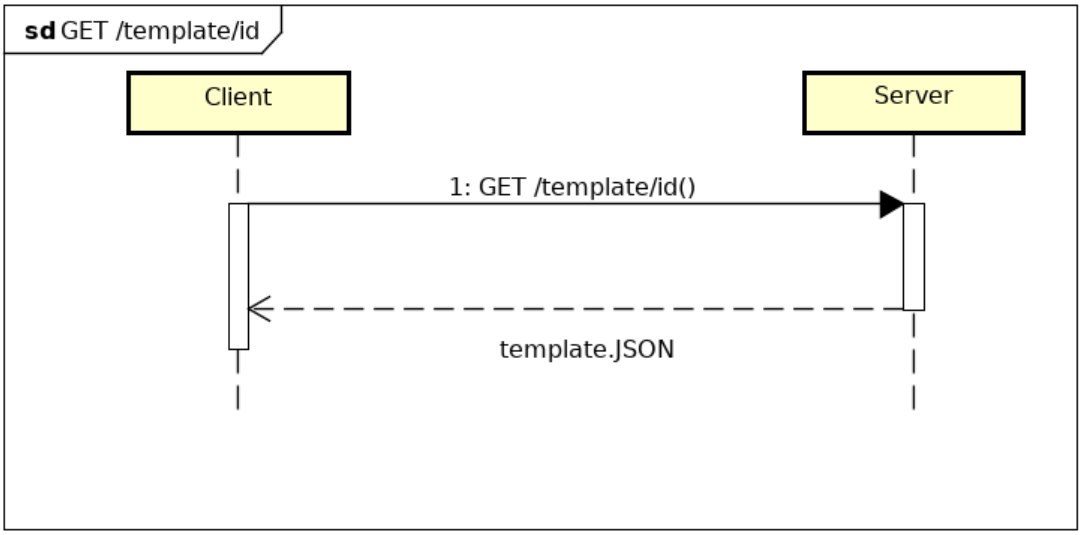

Figura 2: Diagramma di sequenza GET /template/id

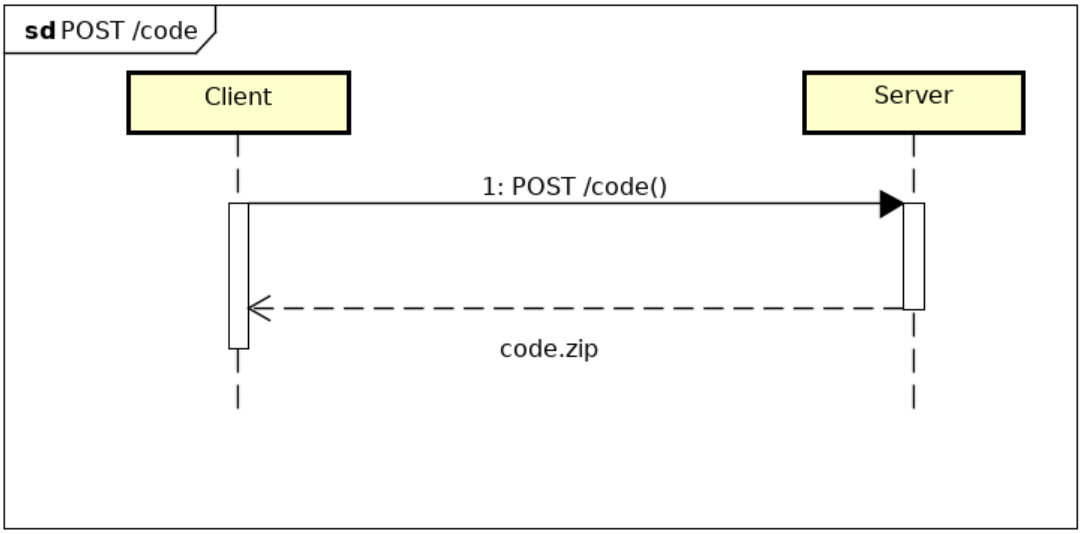

Figura 3: Diagramma di sequenza POST /code

4 Front-end

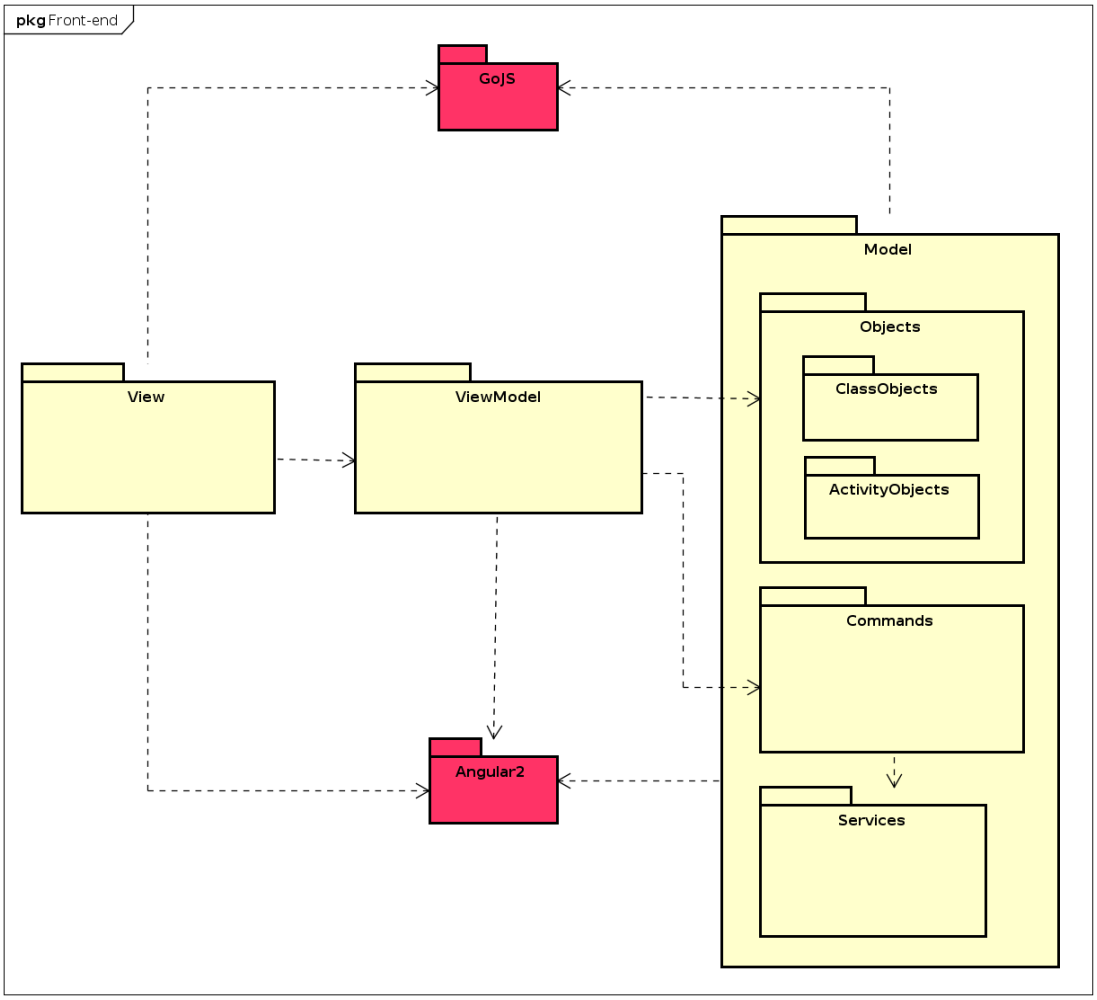

Figura 4: Diagramma dei package Front-end

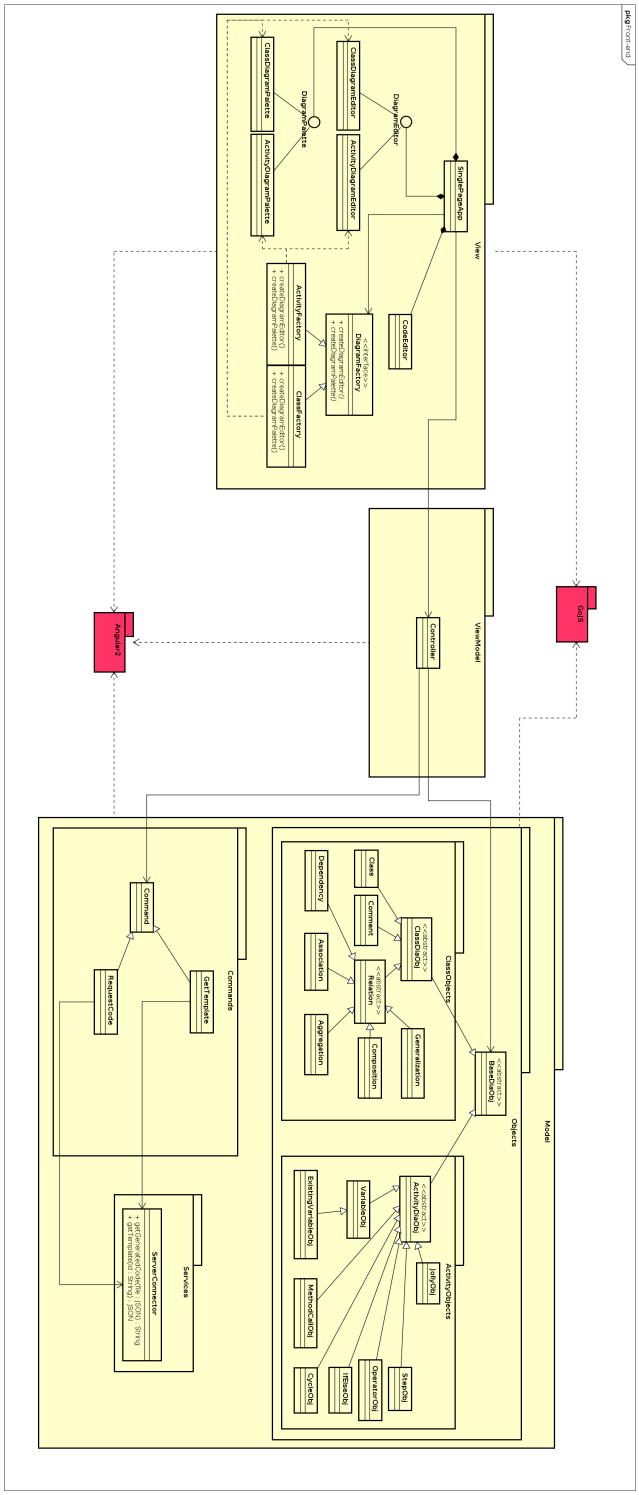

Figura 5: Diagramma delle classi Front-end

4.1 Descrizione packages e classi

4.1.1 Front-end::View

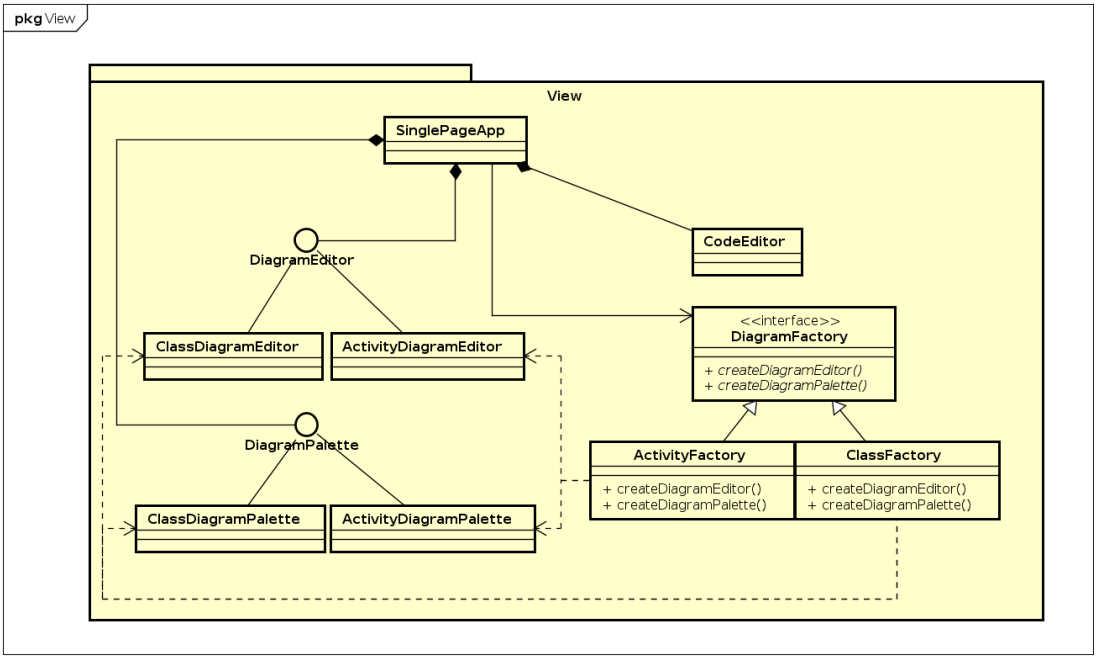

Figura 6: Componente View

4.1.1.1 Informazioni sul package

- **Descrizione**:
  Questo package raccoglie le classi che rappresentano l'editor dei diagrammi delle classi e delle attività, l'editor che permette di modificare il codice prodotto dai diagrammi precedentemente creati. Il package contiene inoltre le palette e i menù laterali che fanno parte della classe dell'editor dei diagrammi;
- **Framework esterni**:
  - Angular2
  - GoJS

4.1.1.2 Classi

4.1.1.2.1 SinglePageApp

- **Descrizione**:
  Questa classe gestisce la visualizzazione dell'interfaccia grafica di SWEDesigner. Contiene le classi riguardanti l'editor dei diagrammi e l'editor del codice generato. Rappresenta il Client del design pattern *Command*$_G$ implementato in `Front-end::Model::Commands`.
- **Utilizzo**:
  Viene utilizzata per generare e gestire l'interfaccia grafica della web-app. Si interfaccia con `Front-end::ViewModel::Controller` per la comunicazione con il `Front-end::Model`. Importa l'interfaccia `Front-end::View::DiagramFactory` per la generazione dell'interfaccia grafica dei diagrammi. Utilizza la libreria GoJS.
- **Relazioni con altre classi**:
  - OUT: `Front-end::ViewModel::Controller`
  - OUT: `Front-end::View::CodeEditor`
  - OUT: `Front-end::View::DiagramEditor`
  - OUT: `Front-end::View::DiagramPalette`
  - OUT: `Front-end::View::DiagramFactory`

4.1.1.2.2 CodeEditor

- **Descrizione**:
  Questa classe gestisce la visualizzazione dell'interfaccia grafica dell'editor del codice generato dal `Back-end`.
- **Utilizzo**:
  Viene utilizzata per generare e gestire l'interfaccia grafica dell'editor del codice generato.
- **Relazioni con altre classi**:
  - IN: `Front-end::View::SinglePageApp`

4.1.1.2.3 DiagramEditor

- **Descrizione**:
  Questa interfaccia rappresenta la struttura per ciascun editor di diagramma. Rappresenta l'interfaccia AbstractProduct del design pattern *Abstract Factory*$_G$.
- **Utilizzo**:
  Viene utilizzata per fornire un'interfaccia grafica comune agli editor dei diagrammi. È contenuta in `Front-end::View::SinglePageApp`. Utilizza la libreria GoJS.

- **Relazioni con altre classi**:
  - IN: `Front-end::View::SinglePageApp`
- **Sottoclassi**:
  - `Front-end::View::ClassDiagramEditor`
  - `Front-end::View::ActivityDiagramEditor`

4.1.1.2.4 ClassDiagramEditor

- **Descrizione**:
  Questa classe gestisce la visualizzazione dell'interfaccia grafica dell'editor del diagramma delle classi.
- **Utilizzo**:
  Viene utilizzata per generare e gestire l'interfaccia grafica dell'editor del diagramma delle classi. Utilizza la libreria GoJS.
- **Relazioni con altre classi**:
  - IN: `Front-end::View::ClassFactory`
- **Interfacce implementate**:
  - `Front-end::View::DiagramEditor`

4.1.1.2.5 ActivityDiagramEditor

- **Descrizione**:
  Questa classe gestisce la visualizzazione dell'interfaccia grafica dell'editor del diagramma delle attività.
- **Utilizzo**:
  Viene utilizzata per generare e gestire l'interfaccia grafica dell'editor del diagramma delle attività. Utilizza la libreria GoJS.
- **Relazioni con altre classi**:
  - IN: `Front-end::View::ActivityFactory`
- **Interfacce implementate**:
  - `Front-end::View::DiagramEditor`

4.1.1.2.6 DiagramPalette

- **Descrizione**:
  Questa interfaccia rappresenta la struttura per ciascuna palette del diagramma. Rappresenta l'interfaccia AbstractProduct del design pattern Abstract Factory.
- **Utilizzo**:
  Viene utilizzata per fornire un'interfaccia grafica comune alle palette dei diagrammi. È contenuta in `Front-end::View::SinglePageApp`. Utilizza la libreria GoJS.
- **Relazioni con altre classi**:
  - IN: `Front-end::View::SinglePageApp`
- **Sottoclassi**:
  - `Front-end::View::ClassDiagramPalette`
  - `Front-end::View::ActivityDiagramPalette`

4.1.1.2.7 ClassDiagramPalette

- **Descrizione**:
  Questa classe gestisce la visualizzazione dell'interfaccia grafica della palette del diagramma delle classi.
- **Utilizzo**:
  Viene utilizzata per generare e gestire l'interfaccia grafica della palette del diagramma delle classi. Utilizza la libreria GoJS.
- **Relazioni con altre classi**:
  - IN: `Front-end::View::ClassFactory`
- **Interfacce implementate**:
  - `Front-end::View::DiagramPalette`

4.1.1.2.8 ActivityDiagramPalette

- **Descrizione**:
  Questa classe gestisce la visualizzazione dell'interfaccia grafica delle palette del diagramma delle attività.
- **Utilizzo**:
  Viene utilizzata per generare e gestire l'interfaccia grafica della palette del diagramma delle attività. Utilizza la libreria GoJS.
- **Relazioni con altre classi**:
  - IN: `Front-end::View::ActivityFactory`

- **Interfacce implementate**:
  - `Front-end::View::DiagramPalette`

4.1.1.2.9 DiagramFactory

- **Descrizione**:
  Questa interfaccia rappresenta la struttura per le operazioni di creazione dei diagrammi. Rappresenta l'interfaccia AbstractFactory del design pattern Abstract Factory.
- **Utilizzo**:
  Viene utilizzata per fornire un interfaccia comune alle factory dei vari tipi di diagrammi. Utilizza la libreria GoJS.
- **Relazioni con altre classi**:
  - IN: `Front-end::View:SinglePageApp`
- **Sottoclassi**:
  - `Front-end::View::ClassFactory`
  - `Front-end::View::ActivityFactory`

4.1.1.2.10 ClassFactory

- **Descrizione**:
  Questa classe implementa le operazioni di creazione dei componenti del diagramma delle classi. Rappresenta la classe ConcreteFactory del design pattern Abstract Factory.
- **Utilizzo**:
  Viene utilizzata per instanziare i componenti concreti del diagramma delle classi, implementando l'interfaccia `Front-end::View::DiagramFactory`. Utilizza la libreria GoJS.
- **Relazioni con altre classi**:
  - OUT: `Front-end::View:ClassDiagramEditor`
  - OUT: `Front-end::View:ClassDiagramPalette`
- **Interfacce implementate**:
  - `Front-end::View::DiagramFactory`

4.1.1.2.11 ActivityFactory

- **Descrizione**:
  Questa classe implementa le operazioni di creazione dei componenti del diagramma delle attività. Rappresenta la classe ConcreteFactory del design pattern Abstract Factory.
- **Utilizzo**:
  Viene utilizzata per instanziare i componenti concreti del diagramma delle attività, implementando l'interfaccia `Front-end::View::DiagramFactory`. Utilizza la libreria GoJS.
- **Relazioni con altre classi**:
  - OUT: `Front-end::View:ActivityDiagramEditor`
  - OUT: `Front-end::View:ActivityDiagramPalette`
- **Interfacce implementate**:
  - `Front-end::View::DiagramFactory`

4.1.2 Front-end::ViewModel

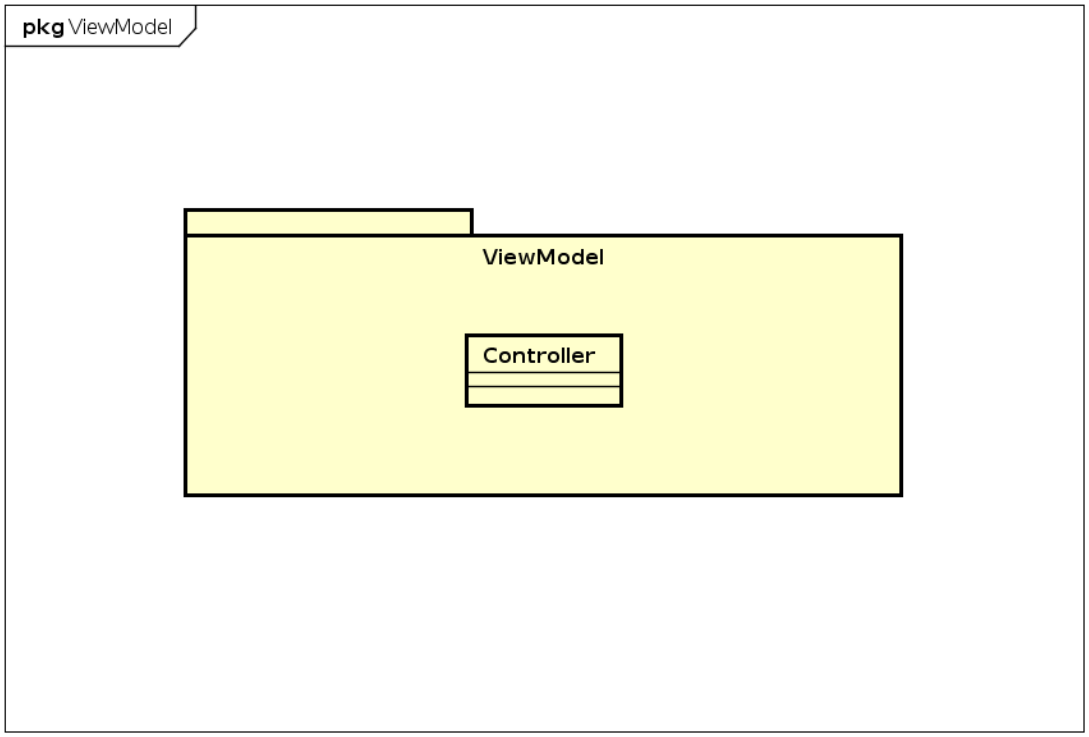

Figura 7: Componente FrontEnd::ViewModel

4.1.2.1 Informazioni sul package

- **Descrizione**:
  Questo package contiene tutte le classi necessarie a regolare la comunicazione e l'interazione tra i package `Front-end::View` e `Front-end::Model` e funge da responsabile per la gestione logica della `Front-end::View`. Fornisce quindi i dati dal modello in una forma che la vista può usare facilmente.
- **Framework esterni**:
  - Angular2

4.1.2.2 Classi

4.1.2.2.1 Controller

- **Descrizione**:
  Questa classe gestisce le richieste del client interfacciandosi con la parte del `Front-end::Model`. Rappresenta l'invoker del design pattern Command presente nel package `Front-end::Model::Commands`.
- **Utilizzo**:
  Viene utilizzata per processare le richieste di inserimento blocchi all'interno dei diagrammi ed eseguire i comandi presenti nel package `Front-end::Model::Commands`.
- **Relazioni con altre classi**:
  - IN: `Front-end::View::SinglePageApp`
  - OUT: `Front-end::Model::Objects::BaseDiaObj`
  - OUT: `Front-end::Model::Commands::Command`

4.1.3 Front-end::Model

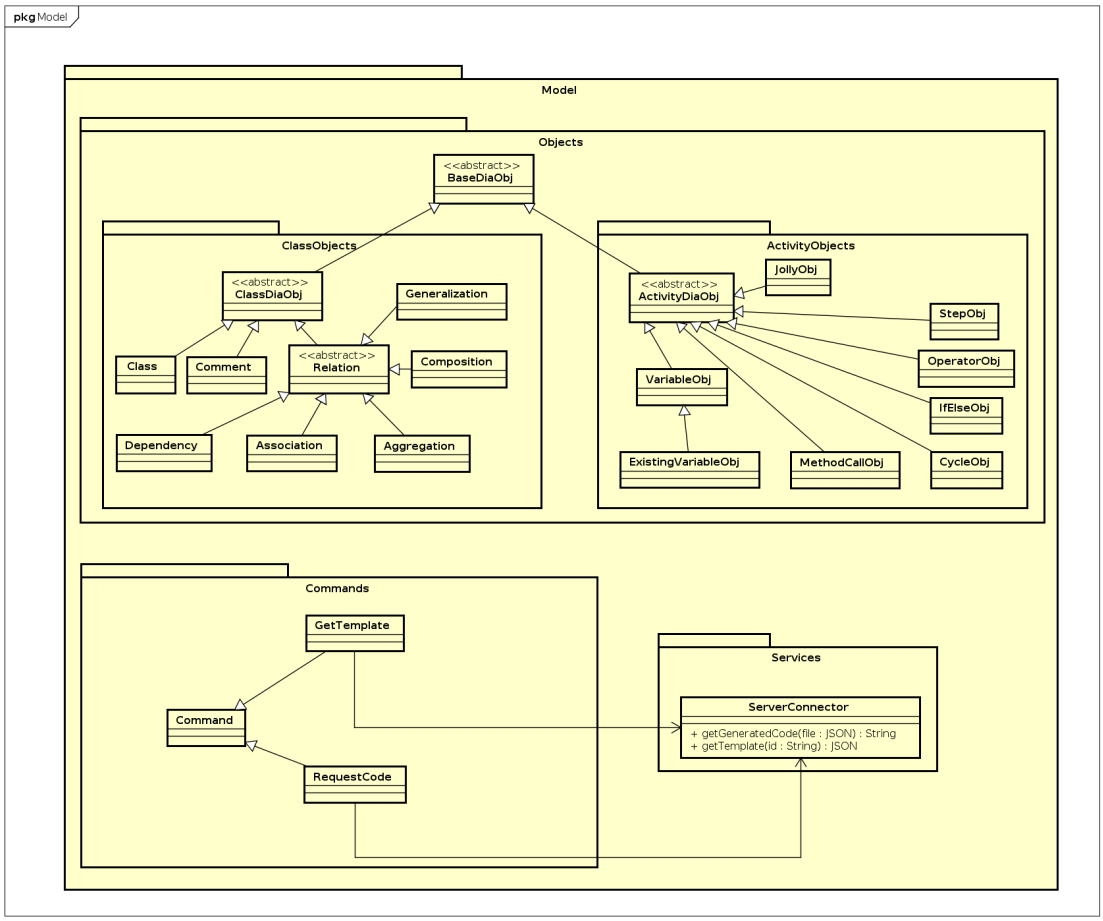

Figura 8: Componente FrontEnd::Model

4.1.3.1 Informazioni sul package

- **Descrizione**:
  Questo package contiene i modelli dei blocchi utilizzati dalla `Front-end::view`, i comandi che permettono la richiesta di generazione codice e richiesta di template con il package `Front-end::Model::Services` che gestisce l'iterazione con il back-end.
- **Package contenuti**:
  - Front-end::Model::Objects
  - Front-end::Model::Commands

  - Front-end::Model::Services

- **Framework esterni**:
  - GoJS
  - Angular2

4.1.4 Front-end::Model::Objects

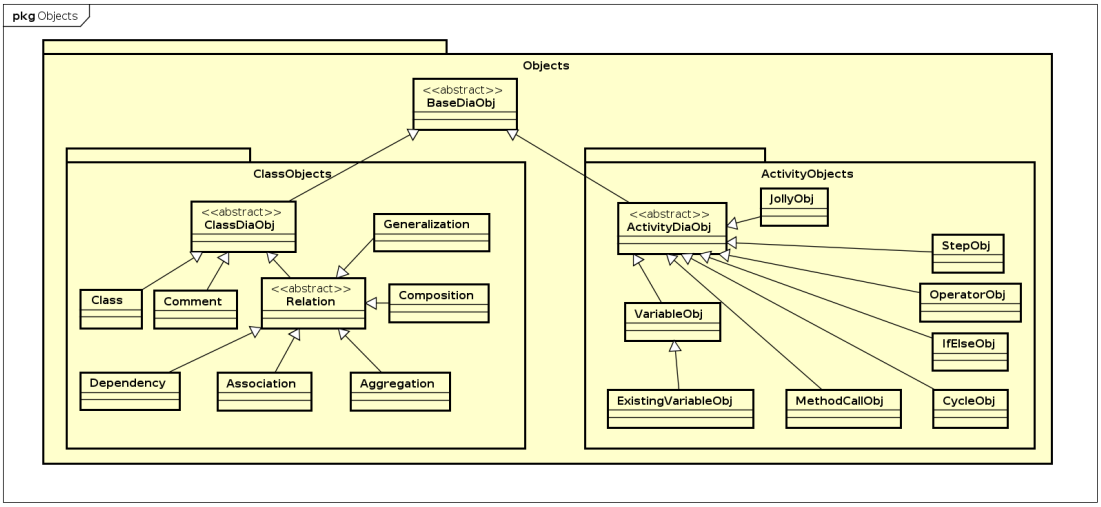

Figura 9: Componente FrontEnd::Model::Objects

4.1.4.1 Informazioni sul package

- **Descrizione**:
  Questo package contiene tutti i blocchi che costituiscono il diagramma delle classi `Front-End::Model::Objects::ClassObjects` ed il diagramma delle attività `Front-End::Model::Objects::ActivityObjects`.

- **Package contenuti**:
  - `Front-end::Model::Objects::ClassObjects`
  - `Front-end::Model::Objects::ActivityObjects`

- **Framework esterni**:
  - GoJS

4.1.4.2 Classi

4.1.4.2.1 BaseDiaObj

- **Descrizione**:
  Questa classe astratta rappresenta un contratto comune tra le classi `Front-end::Model::Objects::ClassObjects::ClassDiaObj` e `Front-end::Model::Objects::ActivityObjects::ActivityDiaObj`.
- **Utilizzo**:
  Viene utilizzata per rappresentare un oggetto base comune ai diagrammi delle classi e delle attività.
- **Relazioni con altre classi**:
  - IN: `Front-end::ViewModel::Controller`
- **Sottoclassi**:
  - `Front-end::Model::Objects::ClassObjects::ClassDiaObj`
  - `Front-end::Model::Objects::ClassObjects::ActivityDiaObj`

4.1.5 Front-end::Model::Objects::ClassObjects

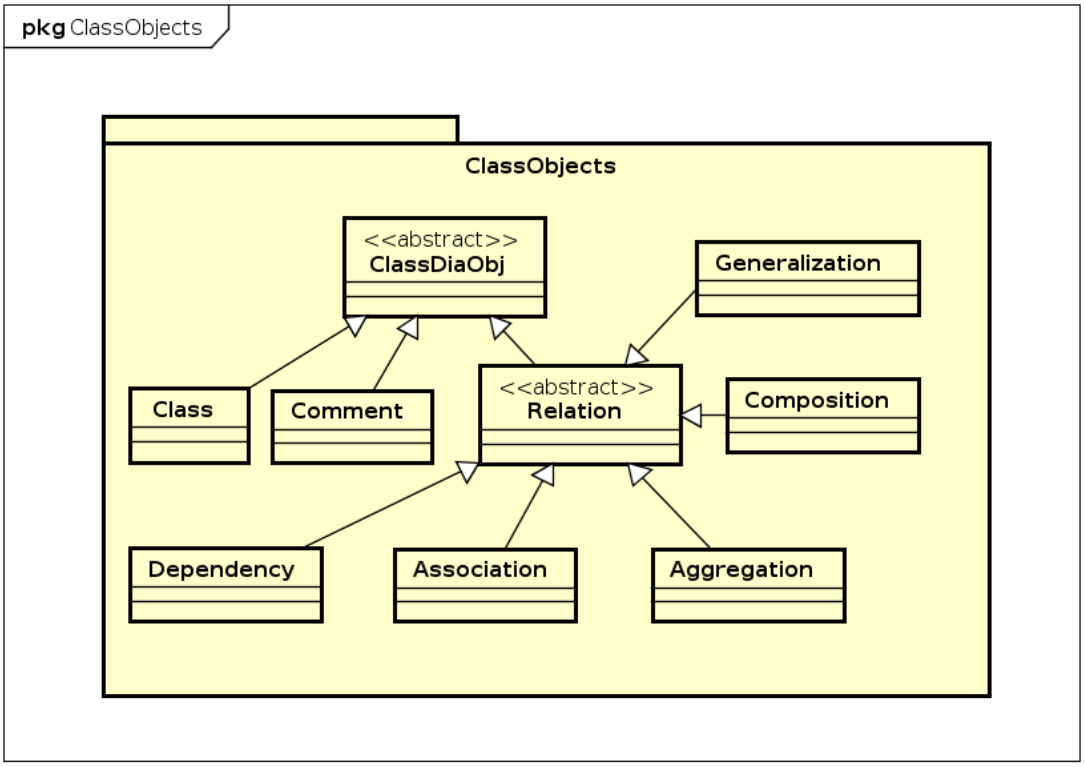

Figura 10: Componente FrontEnd::Model::Objects::ClassObjects

4.1.5.1 Informazioni sul package

- **Descrizione**:
  Questo package contiene tutti i blocchi che vanno a comporre il diagramma delle classi.
- **Framework esterni**:
  - GoJS

4.1.5.2 Classi

4.1.5.2.1 ClassDiaObj

- **Descrizione**:
  Questa classe astratta rappresenta un contratto comune tra le classi che rappresentano i blocchi del diagramma delle classi.

- **Utilizzo**:
  Viene utilizzata per rappresentare un oggetto base comune a tutti i blocchi del diagramma delle classi. Utilizza la libreria GoJS.
- **Classi ereditate**:
  - `Front-end::Model::Objects::BaseDiaObj`
- **Sottoclassi**:
  - `Front-end::Model::Objects::ClassObjects::Class`
  - `Front-end::Model::Objects::ClassObjects::Comment`
  - `Front-end::Model::Objects::ClassObjects::Relation`

4.1.5.2.2 Class

- **Descrizione**:
  Questa classe rappresenta un blocco classe del diagramma delle classi.
- **Utilizzo**:
  Viene utilizzata per istanziare oggetti classe all'interno del diagramma delle classi. Utilizza la libreria GoJS.
- **Classi ereditate**:
  - `Front-end::Model::Objects::ClassObjects::ClassDiaObj`

4.1.5.2.3 Comment

- **Descrizione**:
  Questa classe rappresenta un blocco commento del diagramma delle classi.
- **Utilizzo**:
  Viene utilizzata per istanziare oggetti commento all'interno del diagramma delle classi. Utilizza la libreria GoJS.
- **Classi ereditate**:
  - `Front-end::Model::Objects::ClassObjects::ClassDiaObj`

4.1.5.2.4 Relation

- **Descrizione**:
  Questa classe astratta rappresenta un contratto comune tra le classi che rappresentano le relazioni del diagramma delle classi.

- **Utilizzo**:
  Viene utilizzata per rappresentare un oggetto base comune a tutte le relazioni del diagramma delle classi. Utilizza la libreria GoJS.
- **Classi ereditate**:
  - `Front-end::Model::Objects::ClassObjects::ClassDiaObj`
- **Sottoclassi**:
  - `Front-end::Model::Objects::ClassObjects::Dependency`
  - `Front-end::Model::Objects::ClassObjects::Association`
  - `Front-end::Model::Objects::ClassObjects::Aggregation`
  - `Front-end::Model::Objects::ClassObjects::Composition`
  - `Front-end::Model::Objects::ClassObjects::Generalization`

4.1.5.2.5 Dependency

- **Descrizione**:
  Questa classe rappresenta una relazione di tipo dipendenza del diagramma delle classi.
- **Utilizzo**:
  Viene utilizzata per istanziare oggetti relazione di tipo dipendenza all'interno del diagramma delle classi. Utilizza la libreria GoJS.
- **Classi ereditate**:
  - `Front-end::Model::Objects::ClassObjects::Relation`

4.1.5.2.6 Association

- **Descrizione**:
  Questa classe rappresenta una relazione di tipo associazione del diagramma delle classi.
- **Utilizzo**:
  Viene utilizzata per istanziare oggetti relazione di tipo associazione all'interno del diagramma delle classi. Utilizza la libreria GoJS.
- **Classi ereditate**:
  - `Front-end::Model::Objects::ClassObjects::Relation`

4.1.5.2.7 Aggregation

- **Descrizione**:
  Questa classe rappresenta una relazione di tipo aggregazione del diagramma delle classi.
- **Utilizzo**:
  Viene utilizzata per istanziare oggetti relazione di tipo aggregazione all'interno del diagramma delle classi. Utilizza la libreria GoJS.
- **Classi ereditate**:
  - `Front-end::Model::Objects::ClassObjects::Relation`

4.1.5.2.8 Composition

- **Descrizione**:
  Questa classe rappresenta una relazione di tipo composizione del diagramma delle classi.
- **Utilizzo**:
  Viene utilizzata per istanziare oggetti relazione di tipo composizione all'interno del diagramma delle classi. Utilizza la libreria GoJS.
- **Classi ereditate**:
  - `Front-end::Model::Objects::ClassObjects::Relation`

4.1.5.2.9 Generalization

- **Descrizione**:
  Questa classe rappresenta una relazione di tipo generalizzazione del diagramma delle classi.
- **Utilizzo**:
  Viene utilizzata per istanziare oggetti relazione di tipo generalizzazione all'interno del diagramma delle classi. Utilizza la libreria GoJS.
- **Classi ereditate**:
  - `Front-end::Model::Objects::ClassObjects::Relation`

4.1.6 Front-end::Model::Objects::ActivityObjects

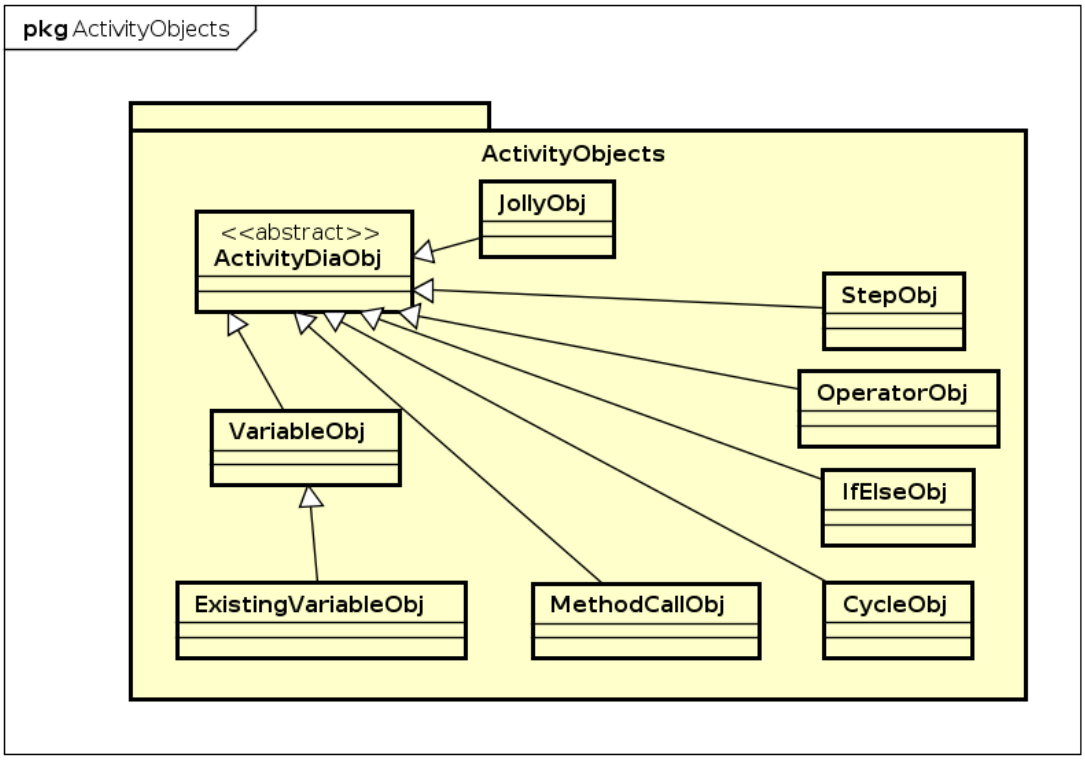

Figura 11: Componente FrontEnd::Model::Objects:ActivityObjects

4.1.6.1 Informazioni sul package

- **Descrizione**:
  Questo package contiene tutti i blocchi che vanno a comporre il diagramma delle attività.
- **Framework esterni**:
  - GoJS

4.1.6.2 Classi

4.1.6.2.1 ActivityDiaObj

- **Descrizione**:
  Questa classe astratta rappresenta un contratto comune tra le classi che rappresentano i blocchi del diagramma delle attività.

- **Utilizzo**:
  Viene utilizzata per rappresentare un oggetto base comune a tutti i blocchi del diagramma delle attività. Utilizza la libreria GoJS.
- **Classi ereditate**:
  - `Front-end::Model::Objects::BaseDiaObj`
- **Sottoclassi**:
  - `Front-end::Model::Objects::ActivityObjects::VariableObj`
  - `Front-end::Model::Objects::ActivityObjects::MethodCallObj`
  - `Front-end::Model::Objects::ActivityObjects::CycleObj`
  - `Front-end::Model::Objects::ActivityObjects::IfElseObj`
  - `Front-end::Model::Objects::ActivityObjects::OperatorObj`
  - `Front-end::Model::Objects::ActivityObjects::StepObj`
  - `Front-end::Model::Objects::ActivityObjects::JollyObj`

4.1.6.2.2 VariableObj

- **Descrizione**:
  Questa classe rappresenta un blocco variabile del diagramma delle attività.
- **Utilizzo**:
  Viene utilizzata per istanziare oggetti variabile all'interno del diagramma delle attività. Utilizza la libreria GoJS.
- **Classi ereditate**:
  - `Front-end::Model::Objects::ActivityObjects::ActivityDiaObj`
- **Sottoclassi**:
  - `Front-end::Model::Objects::ActivityObjects::ExistingVariableObj`

4.1.6.2.3 ExistingVariableObj

- **Descrizione**:
  Questa classe rappresenta un blocco variabile esistente del diagramma delle attività.
- **Utilizzo**:
  Viene utilizzata per istanziare oggetti variabile esistente all'interno del diagramma delle attività. Utilizza la libreria GoJS.

- **Classi ereditate**:
  - `Front-end::Model::Objects::ActivityObjects::VariableObj`

4.1.6.2.4 MethodCallObj

- **Descrizione**:
  Questa classe rappresenta un blocco chiamata del metodo del diagramma delle attività.
- **Utilizzo**:
  Viene utilizzata per istanziare oggetti chiamata del metodo all'interno del diagramma delle attività. Utilizza la libreria GoJS.
- **Classi ereditate**:
  - `Front-end::Model::Objects::ActivityObjects::ActivityDiaObj`

4.1.6.2.5 CycleObj

- **Descrizione**:
  Questa classe rappresenta un blocco ciclo del diagramma delle attività.
- **Utilizzo**:
  Viene utilizzata per istanziare oggetti ciclo all'interno del diagramma delle attività. Utilizza la libreria GoJS.
- **Classi ereditate**:
  - `Front-end::Model::Objects::ActivityObjects::ActivityDiaObj`

4.1.6.2.6 ifElseObj

- **Descrizione**:
  Questa classe rappresenta un blocco if/else del diagramma delle attività.
- **Utilizzo**:
  Viene utilizzata per istanziare oggetti if/else all'interno del diagramma delle attività. Utilizza la libreria GoJS.
- **Classi ereditate**:
  - `Front-end::Model::Objects::ActivityObjects::ActivityDiaObj`

4.1.6.2.7 OperatorObj

- **Descrizione**:
  Questa classe rappresenta un blocco operatore del diagramma delle attività.

- **Utilizzo**:
  Viene utilizzata per istanziare oggetti operatore all'interno del diagramma delle attività. Utilizza la libreria GoJS.
- **Classi ereditate**:
  - `Front-end::Model::Objects::ActivityObjects::ActivityDiaObj`

4.1.6.2.8 StepObj

- **Descrizione**:
  Questa classe rappresenta un blocco avanzamento del diagramma delle attività.
- **Utilizzo**:
  Viene utilizzata per istanziare oggetti avanzamento all'interno del diagramma delle attività. Utilizza la libreria GoJS.
- **Classi ereditate**:
  - `Front-end::Model::Objects::ActivityObjects::ActivityDiaObj`

4.1.6.2.9 JollyObj

- **Descrizione**:
  Questa classe rappresenta un blocco jolly del diagramma delle attività.
- **Utilizzo**:
  Viene utilizzata per istanziare oggetti jolly all'interno del diagramma delle attività. Utilizza la libreria GoJS.
- **Classi ereditate**:
  - `Front-end::Model::Objects::ActivityObjects::ActivityDiaObj`

4.1.7 Front-end::Model::Commands

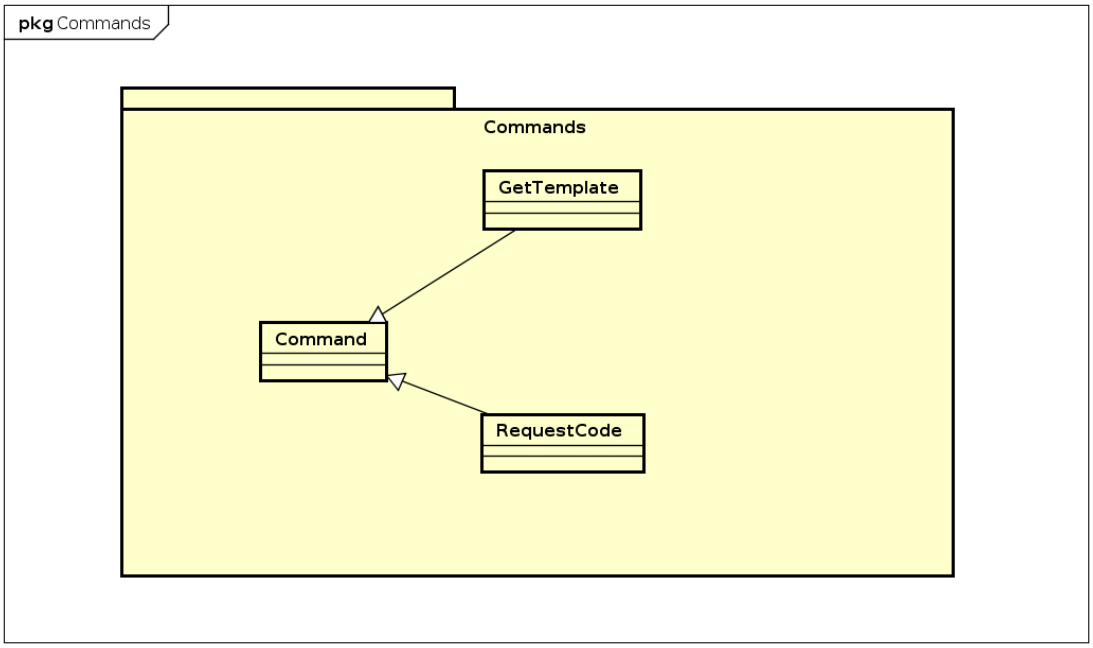

Figura 12: Componente FrontEnd::Model::Commands

4.1.7.1 Informazioni sul package

- **Descrizione**:
  Questo package contiene i comandi relativi al progetto di richiesta codice e richiesta template.
- **Framework esterni**:
  - Angular2

4.1.7.2 Classi

4.1.7.2.1 Command

- **Descrizione**:
  Questa classe rappresenta l'interfaccia per l'esecuzione di un comando che viene chiamato da `Front-end::ViewModel::Controller` quando l'utente decide di richiedere un template o la generazione di codice. Rappresenta il Command del design pattern Command.

- **Utilizzo**:
  Viene utilizzata da `Front-end::ViewModel::Controller`, che rappresenta l'invoker del design pattern Command, che chiederà al `Front-end::Model::Commands::Command` di eseguire le richieste in base all'input dell'utente.
- **Relazioni con altre classi**:
  - IN: `Front-end::ViewModel::Controller`
- **Sottoclassi**:
  - `Front-end::Model::Commands::RequestCode`
  - `Front-end::Model::Commands::GetTemplate`

4.1.7.2.2 RequestCode

- **Descrizione**:
  Questa classe rappresenta un comando di richiesta generazione codice del progetto richiesto dall'utente. Rappresenta il Concrete Command del design pattern Command.
- **Utilizzo**:
  Viene utilizzata da `Front-end::ViewModel::Controller`, per mezzo della classe `Front-end::Model::Commands::Command` per le operazioni legate alla richiesta di generazione del codice del progetto.
- **Relazioni con altre classi**:
  - OUT: `Front-end::Model::Services::ServerConnector`
- **Classi ereditate**:
  - `Front-end::Model::Commands::Command`

4.1.7.2.3 GetTemplate

- **Descrizione**:
  Questa classe rappresenta un comando di richiesta template richiesto dall'utente. Rappresenta il Concrete Command del design pattern Command.
- **Utilizzo**:
  Viene utilizzata da `Front-end::ViewModel::Controller`, per mezzo della classe `Front-end::Model::Commands::Command` per le operazioni legate alla richiesta di template.
- **Classi ereditate**:
  - `Front-end::Model::Commands::Command`

- **Relazioni con altre classi**:
  - OUT: `Front-end::Model::Services::ServerConnector`

4.1.8 Front-end::Model::Services

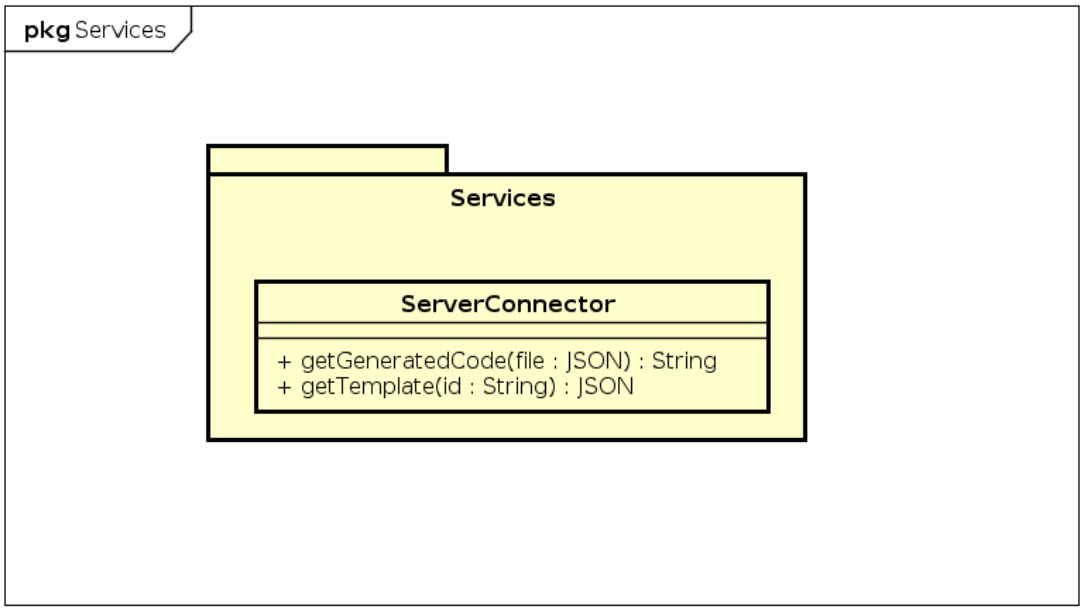

Figura 13: Componente FrontEnd::Model::Services

4.1.8.1 Informazioni sul package

- **Descrizione**:
  Questo package contiene tutte le classi necessarie per gestire la comunicazione tra Front-end e Back-end.
- **Framework esterni**:
  - Angular2

4.1.8.2 Classi

4.1.8.2.1 ServerConnector

- **Descrizione**:
  Questa classe gestisce le comunicazioni con l'applicazione back-end, sfruttando le

funzionalità `http` offerte dal framework Angular2. Rappresenta il Receiver del design pattern Command implementato in `Front-end::Model::Commands`.

- **Utilizzo**:
  Viene utilizzata per l'invio del file JSON contenente il progetto al `Back-end` e la ricezione del codice generato da quest'ultimo dopo aver effettuato la richiesta. Inoltre viene utilizzata per richiedere e ricevere i file JSON relativi ai template presenti nella libreria. Le richieste alla parte di back-end aderiscono ai principi REST.
- **Relazioni con altre classi**:
  - IN: `Front-end::Model::Commands::GetTemplate`
  - IN: `Front-end::Model::Commands::RequestCode`

5 Back-end

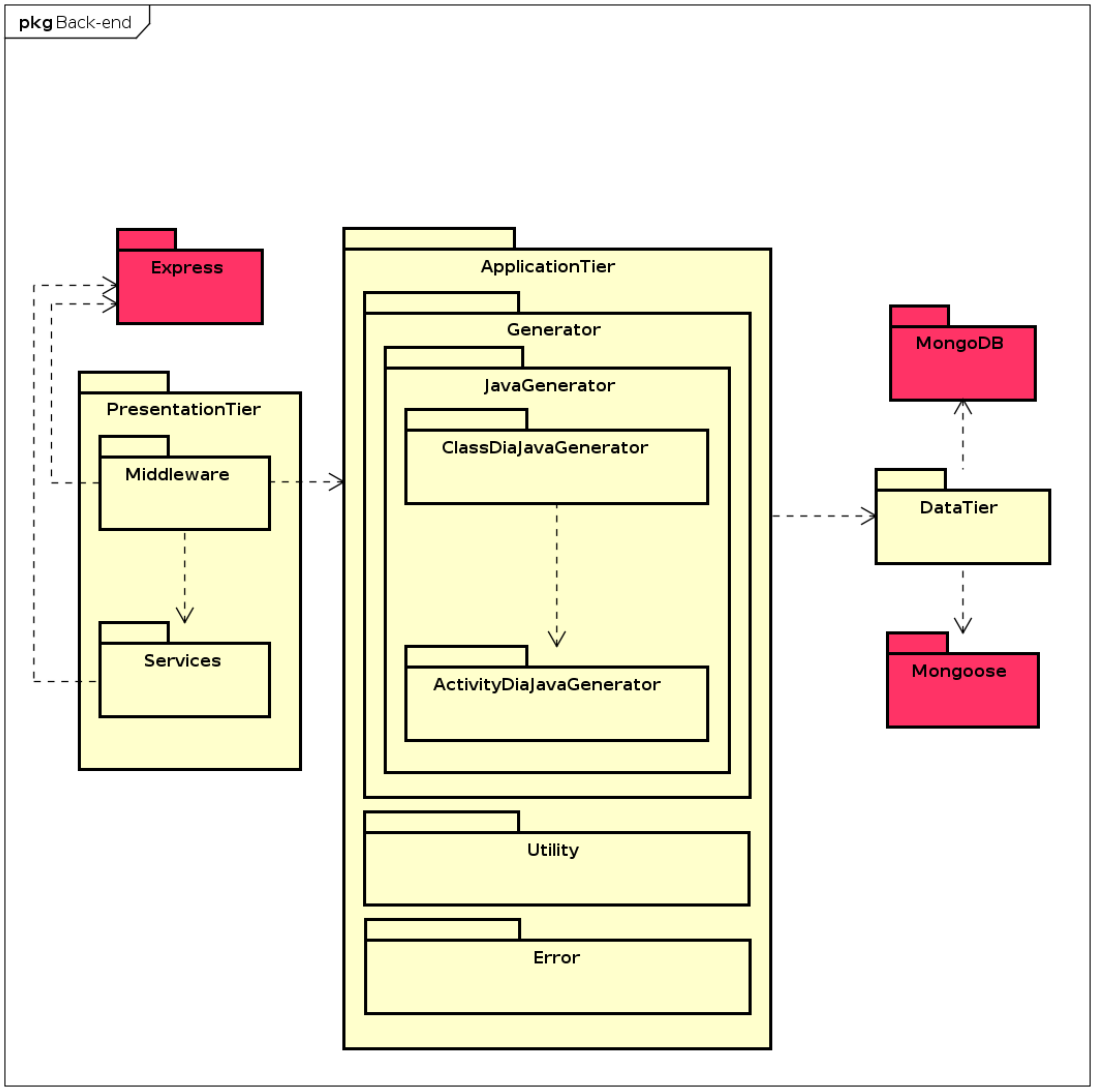

Figura 14: Diagramma dei package Back-end

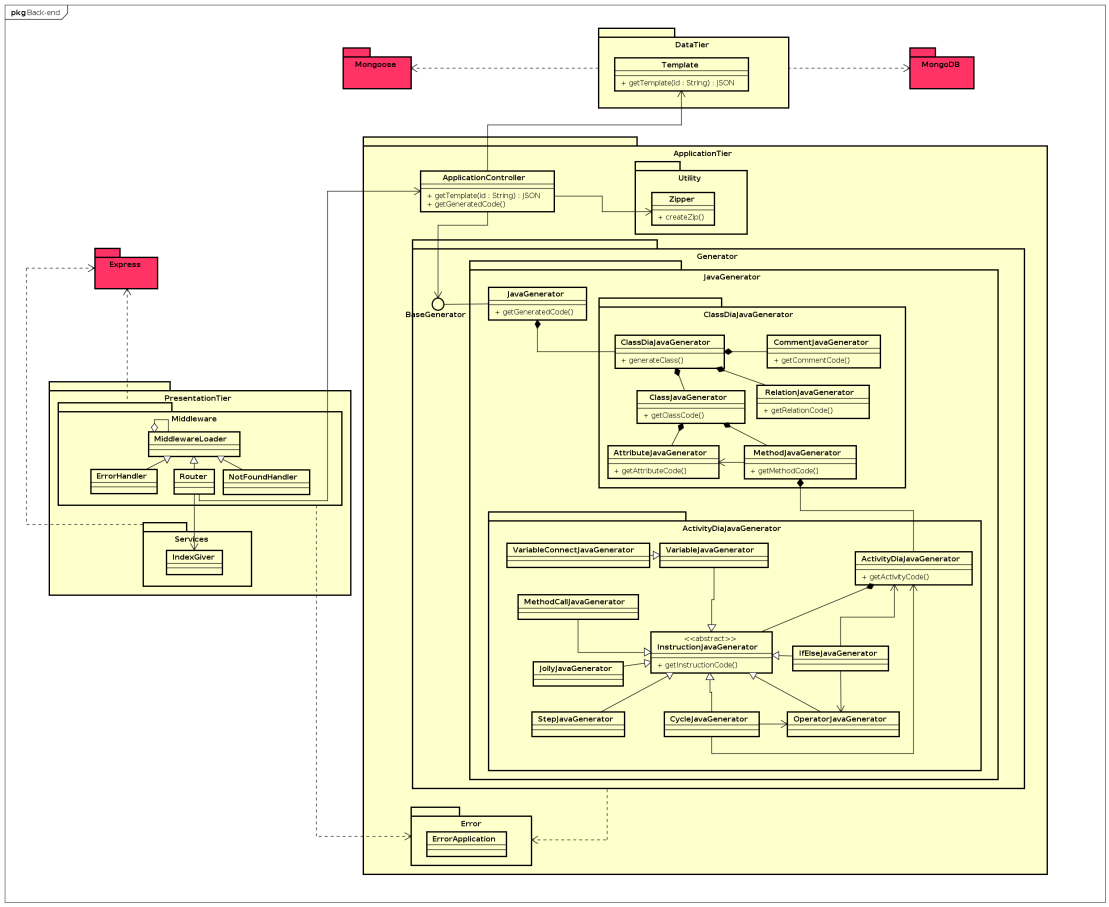

Figura 15: Diagramma delle classi Back-end

5.1 Descrizione packages e classi

5.1.1 Back-end::PresentationTier

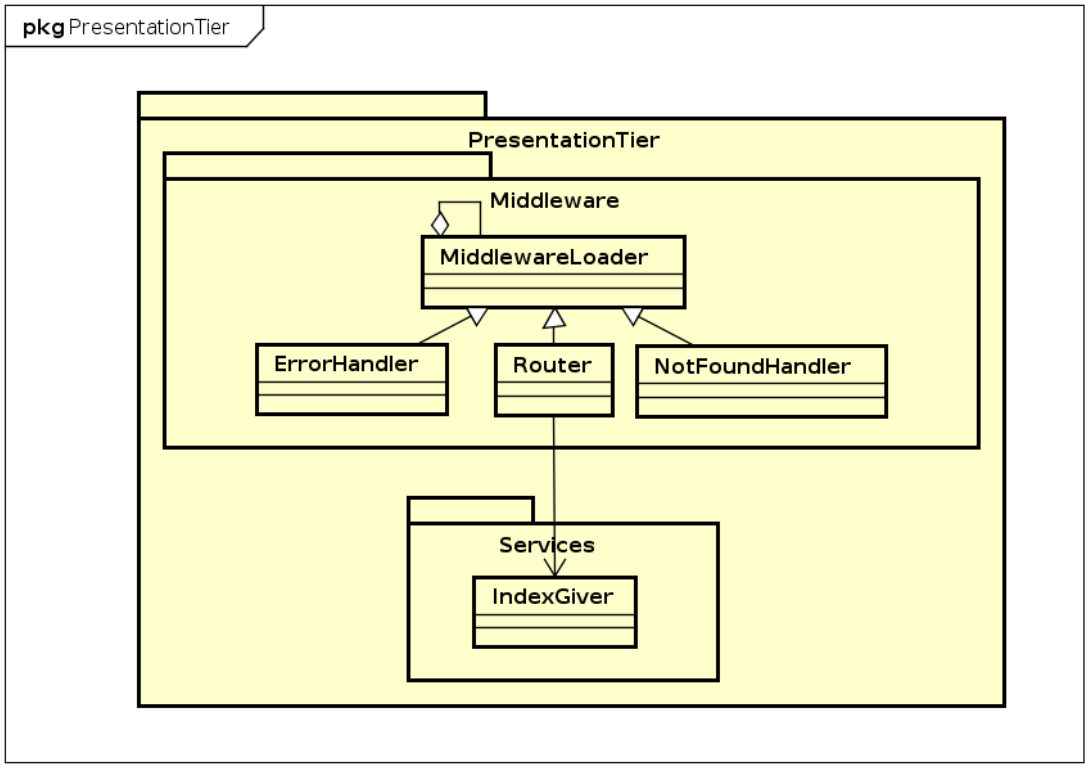

Figura 16: Componente Back-end::PresentationTier

5.1.1.1 Informazioni sul package

- **Descrizione**:
  Questo package contiene tutte le classi che gestiscono le richieste da parte del client. Costituisce la parte Presentation dell'architettura Three-tier del back-end.
- **Package contenuti**:
  - `Back-end::PresentationTier::Middleware`
  - `Back-end::PresentationTier::Services`
- **Framework esterni**:
  - Express

5.1.2 Back-end::PresentationTier::Middleware

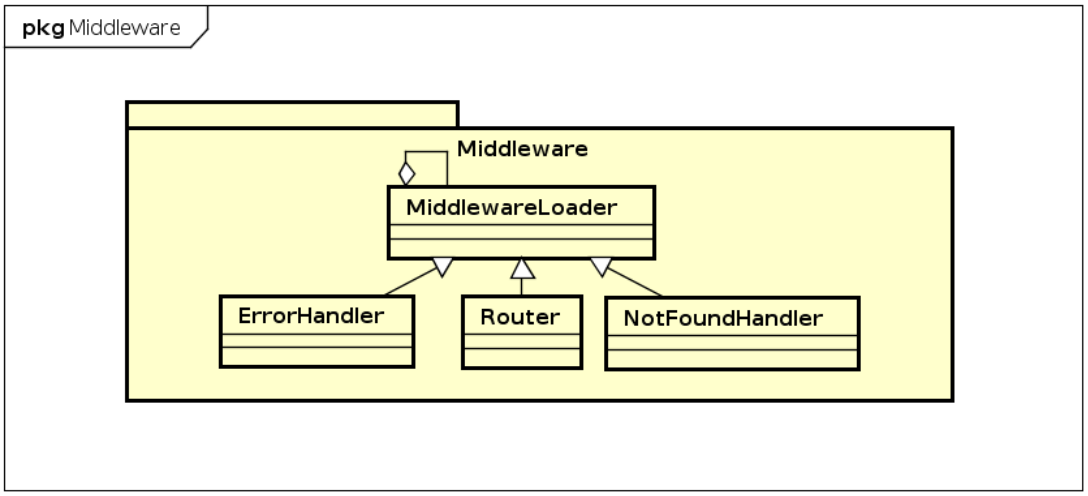

Figura 17: Componente Back-end::PresentationTier::Middleware

5.1.2.1 Informazioni sul package

- **Descrizione**:
  Questo package contiene le classi per la gestione delle comunicazioni tra server e client, e le classi per la gestione dei possibili errori nel momento di richiesta delle risorse.
- **Framework esterni**:
  - Express

5.1.2.2 Classi

5.1.2.2.1 MiddlewareLoader

- **Descrizione**:
  Questa classe definisce un'interfaccia comune per tutte le richieste REST provenienti dal client. È uno dei componenti ConcreteHandler del design pattern *Chain of responsibility*$_G$. Utilizza il framework Express.
- **Utilizzo**:
  Viene utilizzato per istanziare in modo nascosto all'applicazione tutti i middleware presenti nel componente `Back-end::PresentationTier::Middleware`.
- **Sottoclassi**:

- `Back-end::PresentationTier::Middleware::Router`
- `Back-end::PresentationTier::Middleware::ErrorHandler`
- `Back-end::PresentationTier::Middleware::NotFoundHandler`

5.1.2.2.2 ErrorHandler

- **Descrizione**:
  Questa classe gestisce gli errori generati nei precedenti middleware. Invia al client uno stato di risposta HTTP 500 (server error) con una descrizione dell'errore nel formato JSON. È uno dei componenti ConcreteHandler del Design Pattern Chain of responsibility.
- **Utilizzo**:
  Questo middleware viene utilizzato per ultimo nella catena di gestione delle richieste di Express, in modo da gestire tutti gli errori generati precedentemente.
- **Classi ereditate**:
  - `Back-end::PresentationTier::Middleware::MiddlewareLoader`

5.1.2.2.3 Router

- **Descrizione**:
  Classe che si occupa della richiesta di risorse. È uno dei componenti Handler del Design Pattern Chain of responsibility.
- **Utilizzo**:
  Si occupa di smistare la richiesta in base all'URI ricevuto e ad invocare l'opportuno metodo sulla classe `Back-end::ApplicationTier::ApplicationController`. Utilizza il framework Express.
- **Relazioni con altre classi**:
  - OUT: `Back-end::PresentationTier::Services::IndexGiver`
  - OUT: `Back-end::ApplicationTier::ApplicationController`
- **Classi ereditate**:
  - `Back-end::PresentationTier::Middleware::MiddlewareLoader`

5.1.2.2.4 NotFoundHandler

- **Descrizione**:
  Classe che si occupa della gestione dell'errore di pagina non trovata. È uno dei componenti ConcreteHandler del Design Pattern Chain of responsibility.

- **Utilizzo**:
  Viene utilizzata per generare una pagina 404 di errore nel caso in cui l'URI passato non corrisponda ad una risorsa presente nell'applicazione. Utilizza il framework Express.
- **Classi ereditate**:
  - `Back-end::PresentationTier::Middleware::MiddlewareLoader`

5.1.3 Back-end::PresentationTier::Services

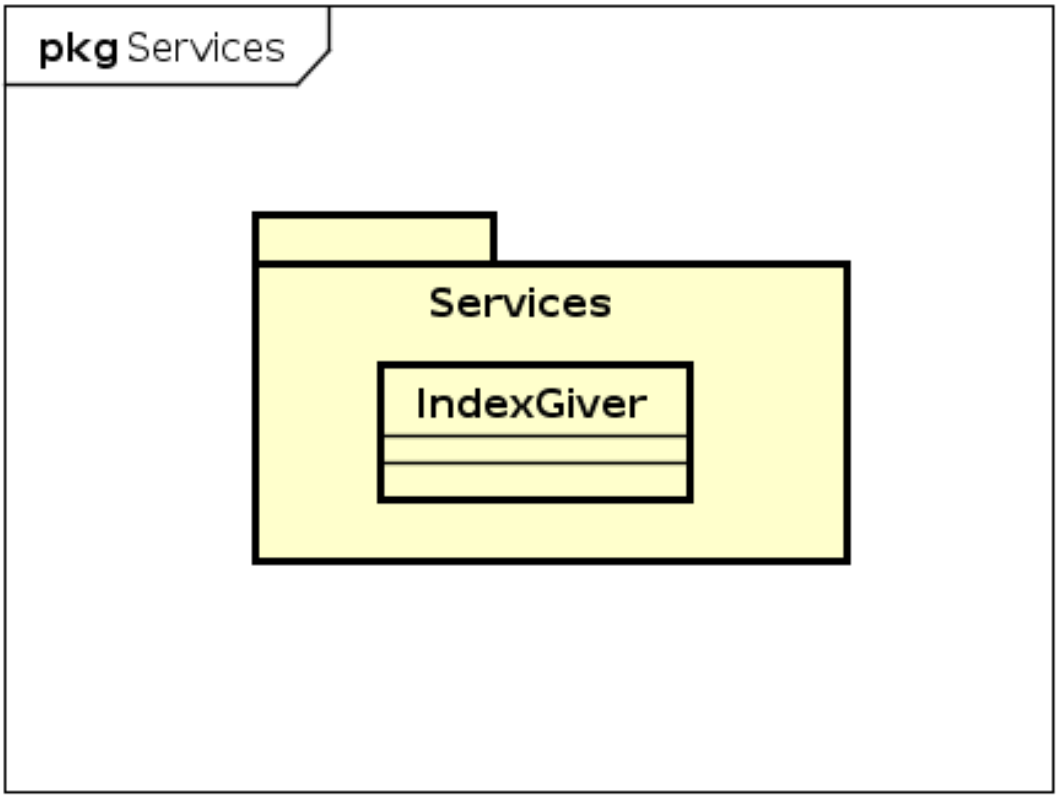

Figura 18: Componente Back-end::PresentationTier::Services

5.1.3.1 Informazioni sul package

- **Descrizione**:
  Questo package contiene la classe che gestisce la richiesta della index-page.
- **Framework esterni**:

- Express

5.1.3.2 Classi

5.1.3.2.1 IndexGiver

- **Descrizione**:
  Questa classe gestisce la richiesta della single page da parte del client.
- **Utilizzo**:
  Viene utilizzata per fornire al client la single page prelevandola dal server. Utilizza il framework Express.
- **Relazioni con altre classi**:
  - IN: `Back-end::PresentationTier::Middleware::Router`

5.1.4 Back-end::ApplicationTier

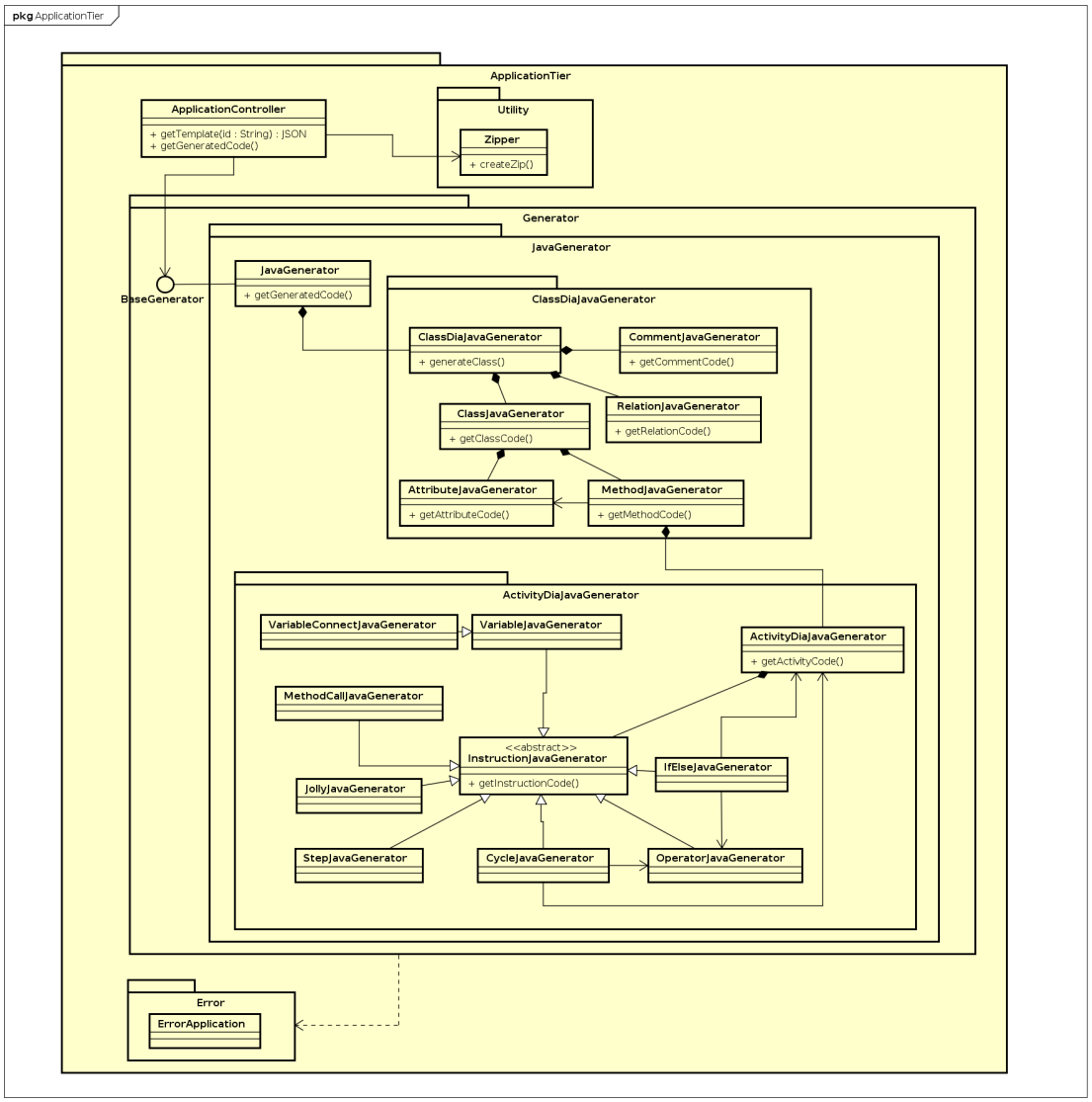

Figura 19: Componente Back-end::ApplicationTier

5.1.4.1 Informazioni sul package

- **Descrizione**:
  Questo package contiene tutte le componenti inerenti la business logic dell'applicazione Back-end, scritte in JavaScript.
- **Package contenuti**:

- Back-end::Generator
- Back-end::Utility
- Back-end::Error

5.1.4.2 Classi

5.1.4.2.1 ApplicationController

- **Descrizione**:
  Questa classe gestisce le comunicazioni tra i tre package della struttura three-tier.
- **Utilizzo**:
  Viene utilizzata per raccogliere le richieste del package `Back-end::PresentationTier` delegandole all'interfaccia `Back-end::ApplicationTier::Generator::BaseGenerator` per la generazione del codice e alla classe `Back-end::DataTier::Template` per la richiesta di dati dal database MongoDB. Inoltre si interfaccia con la classe `Back-end::ApplicationTier::Utility::Zipper` per la compressione dei dati ritornati.
- **Relazioni con altre classi**:
  - IN: `Back-end::PresentationTier::Middleware::Router`
  - OUT: `Back-end::ApplicationTier::Generator::BaseGenerator`
  - OUT: `Back-end::ApplicationTier::Utility::Zipper`
  - OUT: `Back-end::DataTier::Template`

5.1.5 Back-end::ApplicationTier::Generator

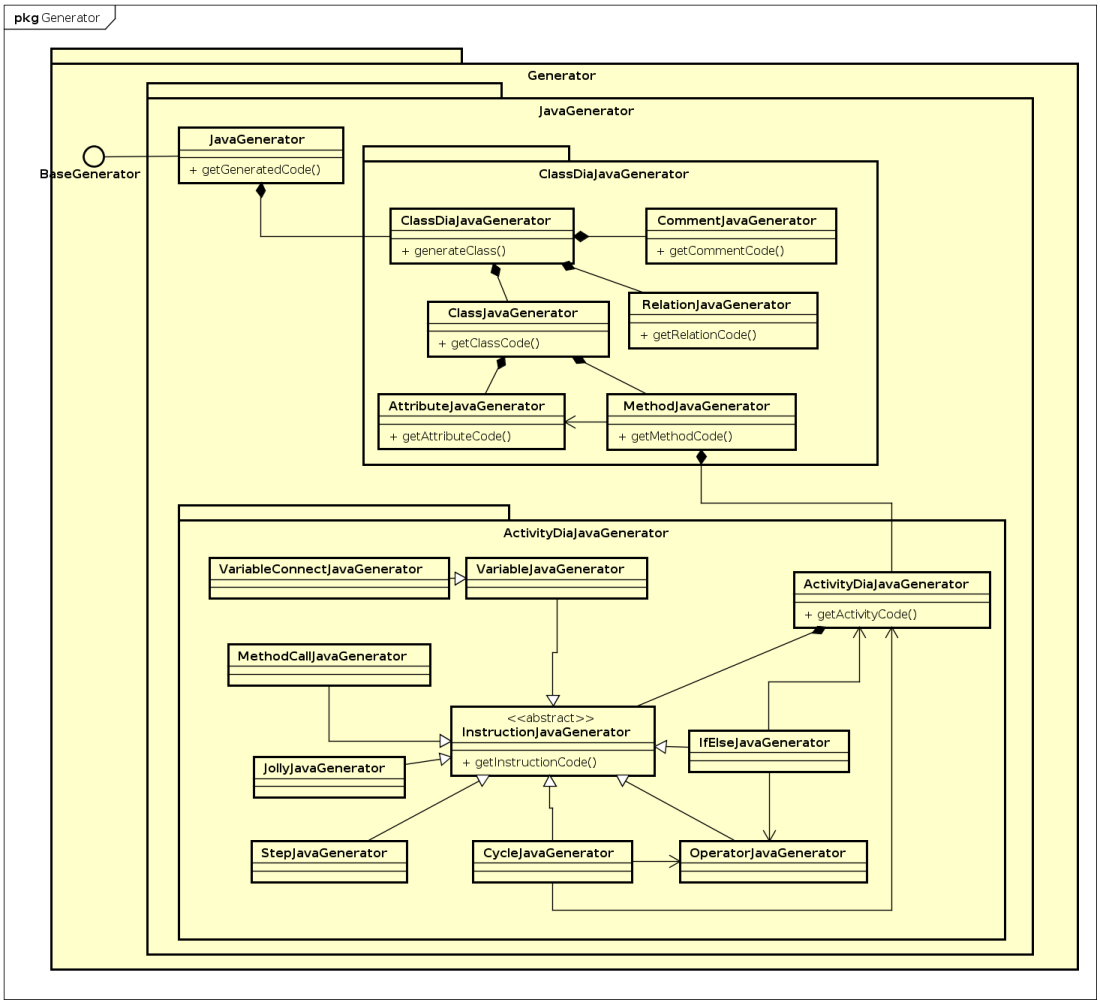

Figura 20: Componente Back-end::ApplicationTier::Generator

5.1.5.1 Informazioni sul package

- **Descrizione**:
  Questo package contiene le classi necessarie per trasformare gli oggetti JSON in codice sorgente di un un particolare linguaggio target. Tali linguaggi sono identificati da ulteriori package interni che possono essere aggiunti nel tempo.
- **Package contenuti**:
  - `Back-end::Generator::JavaGenerator`

5.1.5.2 Classi

5.1.5.2.1 BaseGenerator

- **Descrizione**:
  Questa interfaccia rappresenta un contratto comune a tutte le classi che generano codice. Rappresenta l'interfaccia Strategy del design pattern $Strategy_G$.
- **Utilizzo**:
  Viene utilizzata da `Back-end::ApplicationTier::ApplicationController` per invocare l'algoritmo di generazione codice nel linguaggio voluto attraverso le sue sottoclassi concrete.
- **Relazioni con altre classi**:
  - IN: `Back-end::ApplicationTier::ApplicationController`
- **Sottoclassi**:
  - `Back-end::ApplicationTier::Generator::JavaGenerator::JavaGenerator`

5.1.6 Back-end::ApplicationTier::Generator::JavaGenerator

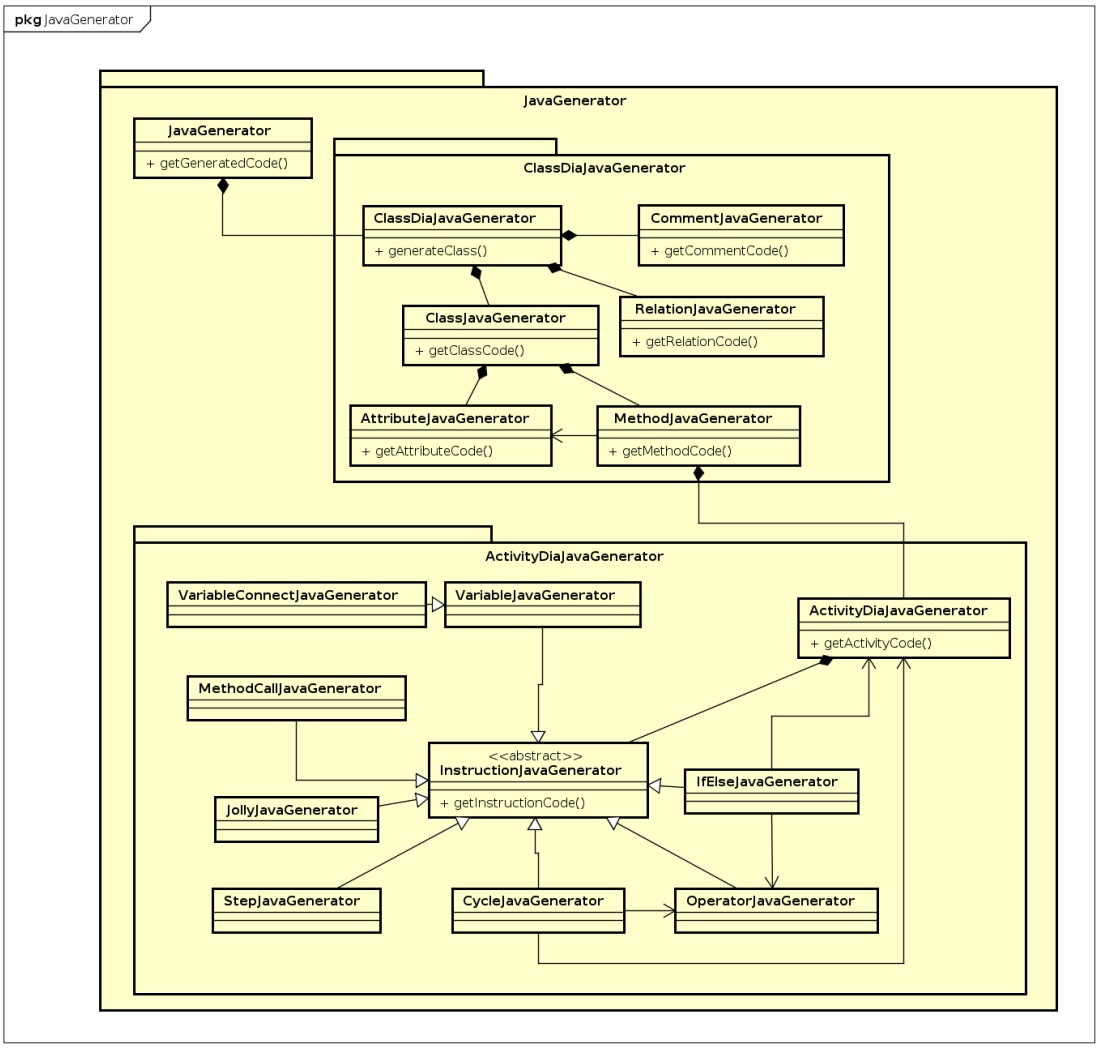

Figura 21: Componente Back-end::ApplicationTier::Generator::JavaGenerator

5.1.6.1 Informazioni sul package

- **Descrizione**:
  Questo package contiene le classi relative alla generazione di codice Java partendo dal file JSON ricavato dai diagrammi delle classi e delle attività.
- **Package contenuti**:
  - `Back-end::Generator::JavaGenerator::ClassDiaJavaGenerator`

- `Back-end::Generator::JavaGenerator::ActivityDiaJavaGenerator`

5.1.6.2 Classi

5.1.6.2.1 JavaGenerator

- **Descrizione**:
  Questa classe implementa l'interfaccia `Back-end::ApplicationTier::Generator::JavaGenerator::JavaGenerator` e si occupa della generazione di codice Java partendo dal file JSON ricavato dai diagrammi delle classi e delle attività. Rappresenta una classe ConcreteStrategy del design pattern *Strategy*.
- **Utilizzo**:
  Viene utilizzata da `Back-end::ApplicationTier::ApplicationController` per generare codice Java fornendo il file JSON proveniente dal client.
- **Relazioni con altre classi**:
  - OUT: `Back-end::ApplicationTier::Generator::JavaGenerator::ClassDiaJavaGenerator::ClassDiaJavagenerator`
- **Interfacce implementate**:
  - `Back-end::ApplicationTier::Generator::BaseGenerator`

5.1.7 Back-end::ApplicationTier::Generator::JavaGenerator ::ClassDiaJavaGenerator

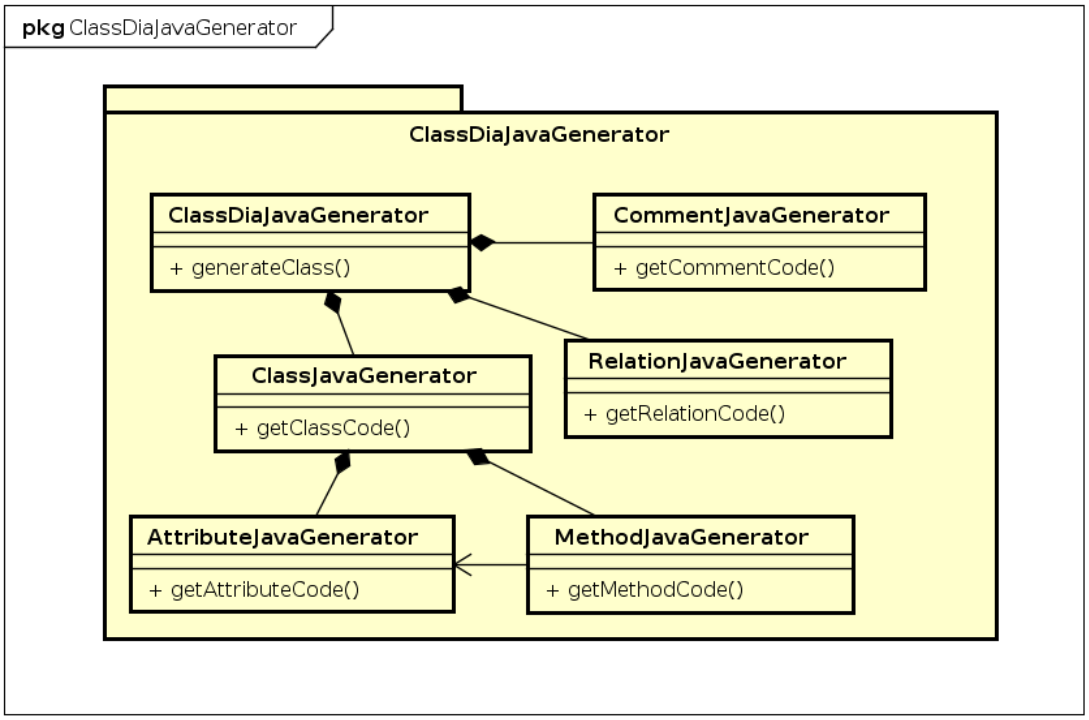

Figura 22: Componente Back-end::ApplicationTier::Generator::JavaGenerator:: ClassDiaJavaGenerator

5.1.7.1 Informazioni sul package

- **Descrizione**:
  Questo package contiene le classi relative alla generazione di codice Java partendo dal file JSON ricavato dai diagrammi delle classi.

5.1.7.2 Classi

5.1.7.2.1 ClassDiaJavaGenerator

- **Descrizione**:
  Questa classe si occupa della generazione di codice Java di un diagramma delle classi partendo dal file JSON.

- **Utilizzo**:
  Viene utilizzata da
  `Back-end::ApplicationTier::Generator::JavaGenerator::JavaGenerator` per la generazione del codice Java relativo alla porzione di file JSON contenente un diagramma delle classi.
- **Relazioni con altre classi**:
  - IN: `Back-end::ApplicationTier::Generator::JavaGenerator ::JavaGenerator`
  - OUT: `Back-end::ApplicationTier::Generator::JavaGenerator:: ClassDiaJavaGenerator::ClassJavaGenerator`
  - OUT: `Back-end::ApplicationTier::Generator::JavaGenerator:: ClassDiaJavaGenerator::RelationJavaGenerator`
  - OUT: `Back-end::ApplicationTier::Generator::JavaGenerator:: ClassDiaJavaGenerator::CommentJavaGenerator`

5.1.7.2.2 ClassJavaGenerator

- **Descrizione**:
  Questa classe si occupa della generazione del codice Java di una classe partendo dal file JSON.
- **Utilizzo**:
  Viene utilizzata da
  `Back-end::ApplicationTier::Generator::JavaGenerator ::ClassDiaJavaGenerator::ClassDiaJavaGenerator` per la generazione del codice Java relativo alla porzione di file JSON contenente una classe.
- **Relazioni con altre classi**:
  - IN: `Back-end::ApplicationTier::Generator::JavaGenerator:: ClassDiaJavaGenerator::ClassDiaJavaGenerator`
  - OUT: `Back-end::ApplicationTier::Generator::JavaGenerator:: ClassDiaJavaGenerator::AttributeJavaGenerator`
  - OUT: `Back-end::ApplicationTier::Generator::JavaGenerator:: ClassDiaJavaGenerator::MethodJavaGenerator`

5.1.7.2.3 AttributeJavaGenerator

- **Descrizione**:
  Questa classe si occupa della generazione del codice Java di un attributo partendo dal file JSON.

- **Utilizzo**:
  Viene utilizzata da `Back-end::ApplicationTier::Generator::JavaGenerator::ClassDiaJavaGenerator::ClassJavaGenerator` per la generazione del codice Java relativo alla porzione di file JSON contenente un attributo.
- **Relazioni con altre classi**:
  - IN: `Back-end::ApplicationTier::Generator::JavaGenerator::ClassDiaJavaGenerator::ClassJavaGenerator`
  - IN: `Back-end::ApplicationTier::Generator::JavaGenerator::ClassDiaJavaGenerator::MethodJavaGenerator`

5.1.7.2.4 MethodJavaGenerator

- **Descrizione**:
  Questa classe si occupa della generazione del codice Java di un metodo partendo dal file JSON.
- **Utilizzo**:
  Viene utilizzata da `Back-end::ApplicationTier::Generator::JavaGenerator::ClassDiaJavaGenerator::ClassJavaGenerator` per la generazione del codice Java relativo alla porzione di file JSON contenente un metodo. Utilizza `Back-end::ApplicationTier::Generator::JavaGenerator::AcitvityDiaJavaGenerator::ActivityDiaJavaGenerator` per la generazione del codice del diagramma delle attività associato al metodo.
- **Relazioni con altre classi**:
  - IN: `Back-end::ApplicationTier::Generator::JavaGenerator::ClassDiaJavaGenerator::ClassJavaGenerator`
  - OUT: `Back-end::ApplicationTier::Generator::JavaGenerator::ClassDiaJavaGenerator::AttributeJavaGenerator`
  - OUT: `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::ActivityDiaJavaGenerator`

5.1.7.2.5 RelationJavaGenerator

- **Descrizione**:
  Questa classe si occupa della generazione del codice Java di una relazione partendo dal file JSON.
- **Utilizzo**:
  Viene utilizzata da `Back-end::ApplicationTier::Generator::JavaGenerator::ClassDiaJavaGenerator::ClassDiaJavaGenerator` per la generazione del codice Java relativo alla porzione di file JSON contenente una relazione tra classi.

- **Relazioni con altre classi**:
  - IN: `Back-end::ApplicationTier::Generator::JavaGenerator::ClassDiaJavaGenerator::ClassDiaJavaGenerator`

5.1.7.2.6 CommentJavaGenerator

- **Descrizione**:
  Questa classe si occupa della generazione del codice Java di un commento partendo dal file JSON.
- **Utilizzo**:
  Viene utilizzata da `Back-end::ApplicationTier::Generator::JavaGenerator::ClassDiaJavaGenerator::ClassDiaJavaGenerator` per la generazione del codice Java relativo alla porzione di file JSON contenente un commento.
- **Relazioni con altre classi**:
  - IN: `Back-end::ApplicationTier::Generator::JavaGenerator::ClassDiaJavaGenerator::ClassDiaJavaGenerator`

5.1.8 Back-end::ApplicationTier::Generator::JavaGenerator ::ActivityDiaJavaGenerator

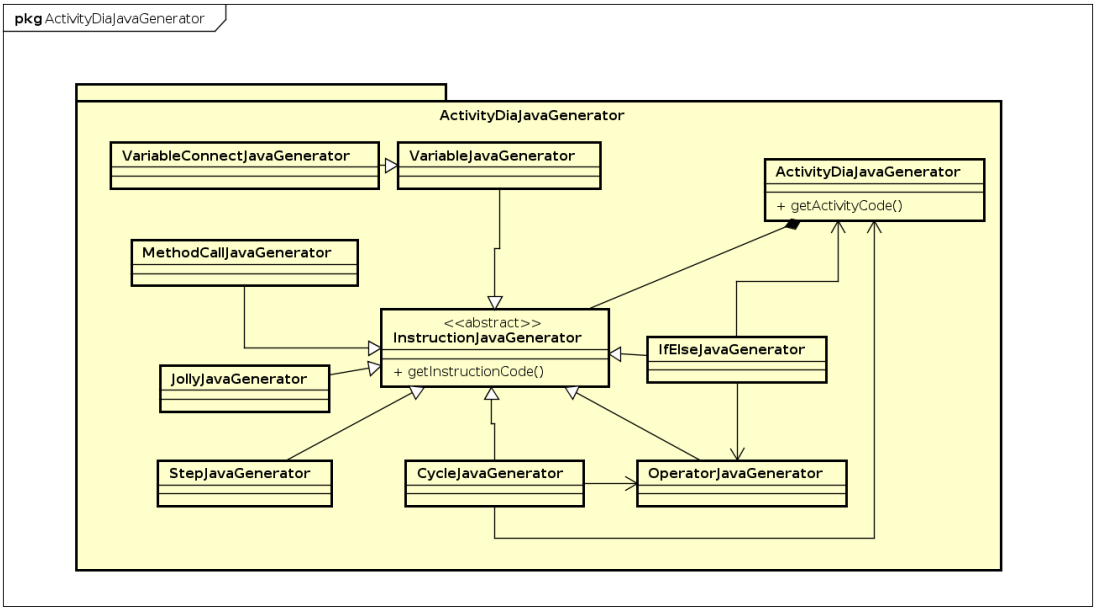

Figura 23: Componente Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator

5.1.8.1 Informazioni sul package

- **Descrizione**:
  Questo package contiene le classi relative alla generazione di codice Java partendo dal file JSON ricavato dai diagrammi delle attività.

5.1.8.2 Classi

5.1.8.2.1 ActivityDiaJavaGenerator

- **Descrizione**:
  Questa classe si occupa della generazione di codice Java di un diagramma delle attività partendo dal file JSON.
- **Utilizzo**:
  Viene utilizzata da `Back-end::ApplicationTier::Generator::JavaGenerator::ClassDiaJavaGenerator::MethodJavaGenerator` per la generazione del codice Java relativo alla porzione di file JSON contenente un diagramma delle attività corrispondente al corpo di un metodo di una classe definita. Viene inoltre utilizzata per la generazione del codice Java relativo al corpo dei cicli e degli if/else in quanto essi vengono trattati come dei sotto-diagrammi delle attività.
- **Relazioni con altre classi**:
  - IN: `Back-end::ApplicationTier::Generator::JavaGenerator::ClassDiaJavaGenerator::MethodJavaGenerator`
  - IN: `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::CycleJavaGenerator`
  - IN: `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::IfElseJavaGenerator`
  - OUT: `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::InstructionJavaGenerator`

5.1.8.2.2 InstructionJavaGenerator

- **Descrizione**:
  Questa classe astratta rappresenta un contratto comune a tutte le classi che generano codice Java relativo a blocchi del diagramma delle attività.
- **Utilizzo**:
  Viene utilizzata per rappresentare un oggetto base comune a tutte le classi che rappresentano istruzioni all'interno di un diagramma delle attività.
- **Relazioni con altre classi**:

  - IN: `Back-end::ApplicationTier::Generator::JavaGenerator ::ActivityDiaJavaGenerator::ActivityDiaJavaGenerator`

- **Sottoclassi**:
  - `Back-end::ApplicationTier::Generator::JavaGenerator ::ActivityDiaJavaGenerator::VariableJavaGenerator`
  - `Back-end::ApplicationTier::Generator::JavaGenerator ::ActivityDiaJavaGenerator::MethodCallJavaGnerator`
  - `Back-end::ApplicationTier::Generator::JavaGenerator ::ActivityDiaJavaGenerator::JollyJavaGenerator`
  - `Back-end::ApplicationTier::Generator::JavaGenerator ::ActivityDiaJavaGenerator::StepJavaGenerator`
  - `Back-end::ApplicationTier::Generator::JavaGenerator ::ActivityDiaJavaGenerator::CycleJavaGenerator`
  - `Back-end::ApplicationTier::Generator::JavaGenerator ::ActivityDiaJavaGenerator::OperatorJavaGenerator`
  - `Back-end::ApplicationTier::Generator::JavaGenerator ::ActivityDiaJavaGenerator::IfElseJavaGenerator`

5.1.8.2.3 VariableJavaGenerator

- **Descrizione**:
  Questa classe si occupa della generazione del codice Java di un blocco variabile del diagramma delle attività partendo dal file JSON.
- **Utilizzo**:
  Viene utilizzata da `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::ActivityDiaJavaGenerator` per la generazione del codice Java relativo alla porzione di file JSON contenente un blocco variabile del diagramma delle attività.
- **Classi ereditate**:
  - `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::InstructionJavaGenerator`
- **Sottoclassi**:
  - `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::VariableConnectJavaGenerator`

5.1.8.2.4 VariableConnectJavaGenerator

- **Descrizione**:
  Questa classe si occupa della generazione del codice Java di un blocco connessione variabile esistente del diagramma delle attività partendo dal file JSON.
- **Utilizzo**:
  Viene utilizzata da `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::ActivityDiaJavaGenerator` per la generazione del codice Java relativo alla porzione di file JSON contenente un blocco connessione variabile esistente del diagramma delle attività.
- **Classi ereditate**:
  - `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::VariableJavaGenerator`

5.1.8.2.5 MethodCallJavaGenerator

- **Descrizione**:
  Questa classe si occupa della generazione del codice Java di un blocco chiamata del metodo del diagramma delle attività partendo dal file JSON.
- **Utilizzo**:
  Viene utilizzata da `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::ActivityDiaJavaGenerator` per la generazione del codice Java relativo alla porzione di file JSON contenente un blocco chiamata del metodo del diagramma delle attività.
- **Classi ereditate**:
  - `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::InstructionavaGenerator`

5.1.8.2.6 JollyavaGenerator

- **Descrizione**:
  Questa classe si occupa della generazione del codice Java di un blocco jolly del diagramma delle attività partendo dal file JSON.
- **Utilizzo**:
  Viene utilizzata da `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::ActivityDiaJavaGenerator` per la generazione del codice Java relativo alla porzione di file JSON contenente un blocco jolly del diagramma delle attività.
- **Classi ereditate**:

  - `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::InstructionavaGenerator`

5.1.8.2.7 StepJavaGenerator

- **Descrizione**:
  Questa classe si occupa della generazione del codice Java di un blocco avanzamento del diagramma delle attività partendo dal file JSON.
- **Utilizzo**:
  Viene utilizzata da `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::ActivityDiaJavaGenerator` per la generazione del codice Java relativo alla porzione di file JSON contenente un blocco avanzamento del diagramma delle attività.
- **Classi ereditate**:
  - `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::InstructionavaGenerator`

5.1.8.2.8 CycleJavaGenerator

- **Descrizione**:
  Questa classe si occupa della generazione del codice Java di un blocco ciclo del diagramma delle attività partendo dal file JSON.
- **Utilizzo**:
  Viene utilizzata da `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::ActivityDiaJavaGenerator` per la generazione del codice Java relativo alla porzione di file JSON contenente un blocco ciclo del diagramma delle attività.
- **Relazioni con altre classi**
  - OUT: `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::ActivityDiaJavaGenerator`
  - OUT: `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::OperatorJavaGenerator`
- **Classi ereditate**:
  - `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::InstructionavaGenerator`

5.1.8.2.9 OperatorJavaGenerator

- **Descrizione**:
  Questa classe si occupa della generazione del codice Java di un blocco operatore del diagramma delle attività partendo dal file JSON.
- **Utilizzo**:
  Viene utilizzata da `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::ActivityDiaJavaGenerator` per la generazione del codice Java relativo alla porzione di file JSON contenente un blocco operatore del diagramma delle attività.
- **Relazioni con altre classi**
  - IN: `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::CycleJavaGenerator`
  - IN: `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::IfElseJavaGenerator`
- **Classi ereditate**:
  - `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::InstructionavaGenerator`

5.1.8.2.10 IfElseJavaGenerator

- **Descrizione**:
  Questa classe si occupa della generazione del codice Java di un blocco if/else del diagramma delle attività partendo dal file JSON.
- **Utilizzo**:
  Viene utilizzata da `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::ActivityDiaJavaGenerator` per la generazione del codice Java relativo alla porzione di file JSON contenente un blocco if/else del diagramma delle attività.
- **Relazioni con altre classi**
  - OUT: `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::ActivityDiaJavaGenerator`
  - OUT: `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::OperatorJavaGenerator`
- **Classi ereditate**:
  - `Back-end::ApplicationTier::Generator::JavaGenerator::ActivityDiaJavaGenerator::InstructionavaGenerator`

5.1.9 Back-end::ApplicationTier::Utility

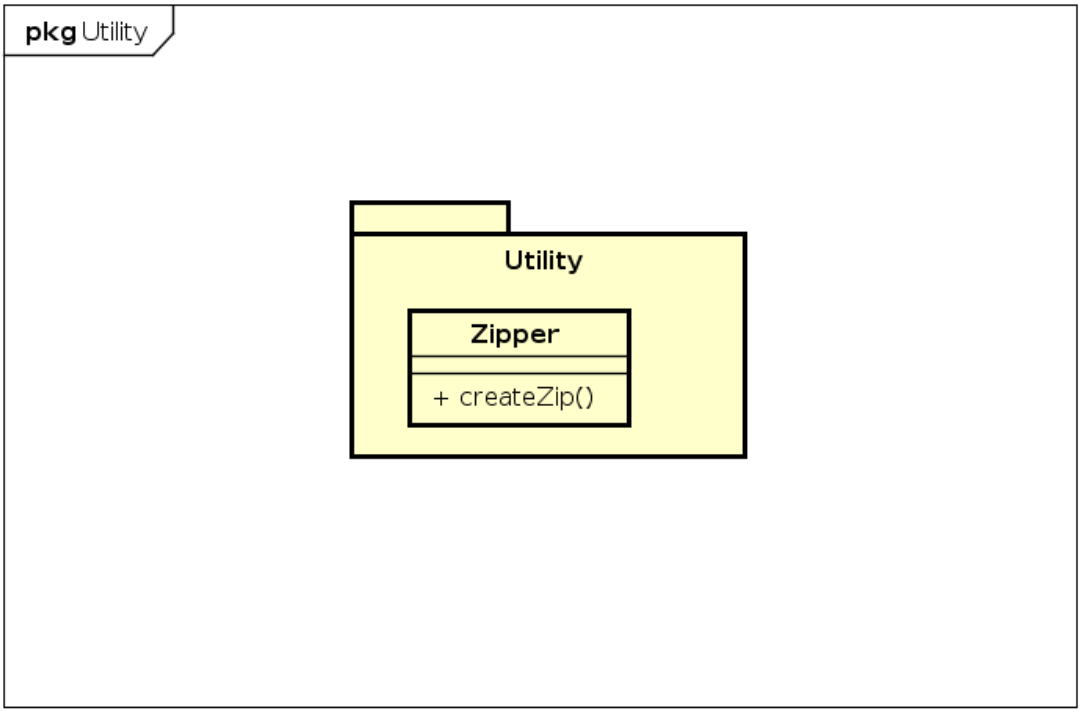

Figura 24: Componente Back-end::ApplicationTier::Utility

5.1.9.1 Informazioni sul package

- **Descrizione**:
  Questo package contiene la classe che gestisce la compressione del progetto.

5.1.9.2 Classi

5.1.9.2.1 Zipper

- **Descrizione**:
  Questa classe si occupa di effettuare la compressione dei file che le vengono forniti.
- **Utilizzo**:
  Viene utilizzata da `Back-end::ApplicationTier::ApplicationController` per comprimere i file generati da `Back-end::ApplicationTier::Generator::BaseGenerator`.
- **Relazioni con altre classi**:

– IN: `Back-end::ApplicationTier::Utility::Zipper`

5.1.10 Back-end::ApplicationTier::Error

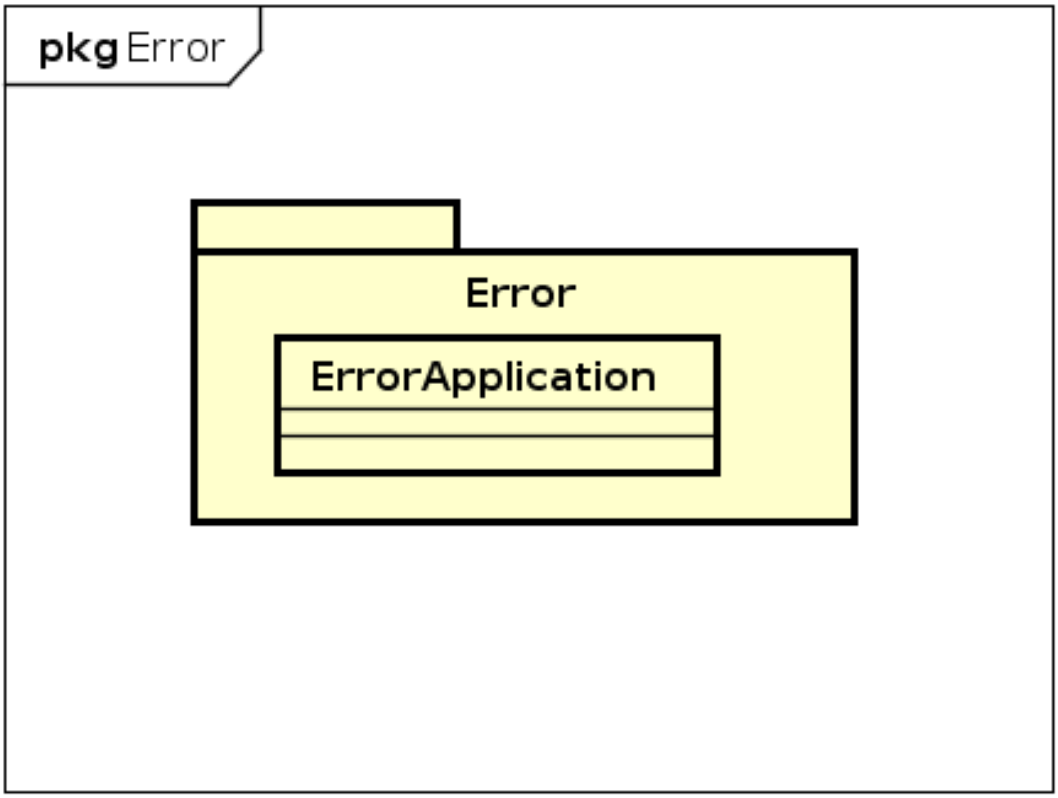

Figura 25: Componente Back-end::ApplicationTier::Error

5.1.10.1 Informazioni sul package

- **Descrizione**:
  Questo package contiene la classe che gestisce la visualizzazione degli errori.

5.1.10.2 Classi

5.1.10.2.1 ErrorApplication

- **Descrizione**:
  Questa classe rappresenta un errore che può verificarsi nel `Back-end` .

- **Utilizzo**:
  Viene utilizzata da tutte le classi presenti all'interno del package `Back-end` per rappresentare un errore generato, identificandolo tramite codice, nome e descrizione.

5.1.11 Back-end::DataTier

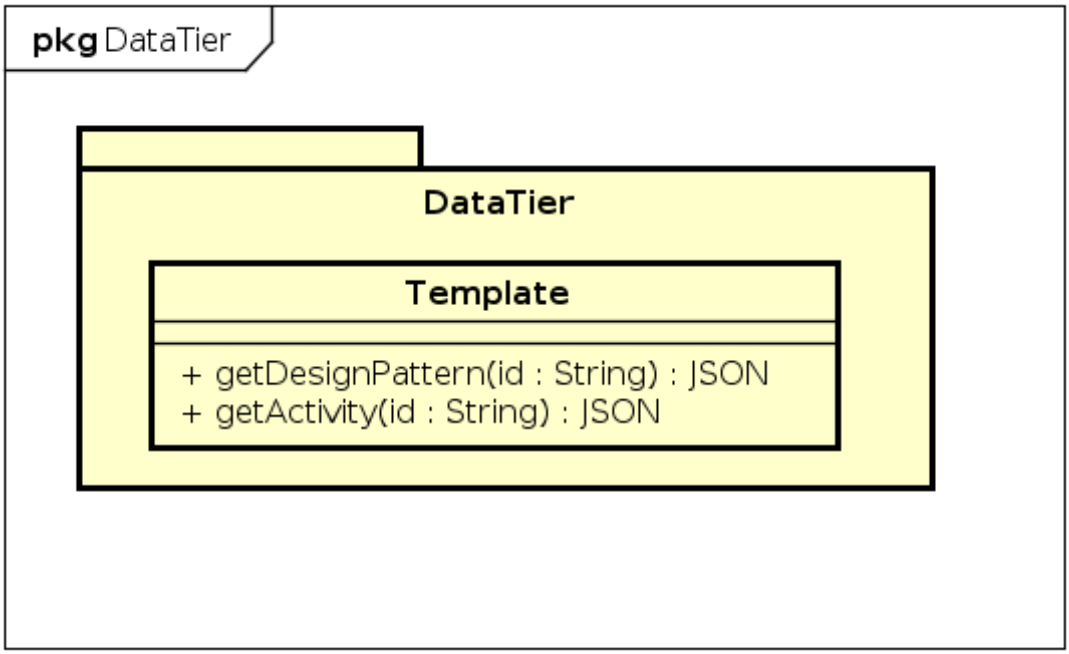

Figura 26: Componente Back-end::DataTier

5.1.11.1 Informazioni sul package

- **Descrizione**:
  Questo package contiene la classe che gestisce il recupero dei dati di libreria interfacciandosi con un database MongoDB, tramite la libreria Mongoose. Costituisce la parte Presentation dell'architettura Three-tier del back-end.
- **Framework esterni**:
  - MongoDB
  - Mongoose

5.1.11.2 Classi

5.1.11.2.1 Template

- **Descrizione**:
  Questa classe gestisce la richiesta di un template JSON da parte del client richiedendolo al Database.
- **Utilizzo**:
  Viene utilizzato per gestire la richiesta di un template JSON da parte di `Back-end::ApplicationTier::ApplicationController` richiedendolo al database MongoDB utilizzando la libreria Mongoose.
- **Relazioni con altre classi**:
  - IN: `Back-end::ApplicationTier::ApplicationController`

6 Design pattern

Un design pattern è una descrizione o modello logico da applicare per la risoluzione di un problema che può presentarsi in diverse situazioni durante le fasi di progettazione e sviluppo del software, ancor prima della definizione dell'algoritmo risolutivo della parte computazionale. Oltre al problema descrive anche soluzioni eleganti per la sua risoluzione. I design pattern facilitano l'attività di progettazione, la riusabilità e la manutenibilità. I design pattern si possono dividere in quattro fondamentali categorie:

- **Design Pattern Architetturali**: esprimono degli schemi di base per impostare l'organizzazione strutturale di un sistema software;
- **Design Pattern Creazionali**: forniscono un'astrazione del processo di instanziazione degli oggetti;
- **Design Pattern Strutturali**: si occupano delle modalità di composizione di classi e oggetti per formare strutture complesse;
- **Design Pattern Comportamentali**: si occupano di algoritmi e dell'assegnazione di responsabilità tra oggetti collaboranti.

Per approfondire i design pattern utilizzati nel progetto si faccia riferimento all'appendice A. In seguito verranno descritti i design pattern implementati e il loro utilizzo.

6.1 Design pattern architetturali

Il gruppo ha deciso di separare l'architettura del *Front-end* da quella di *Back-end* vedendo entrambe le parti come due applicazioni distinte ma comunicanti tra loro. Abbiamo utilizzato per la parte di *Front-end* il design pattern MVVM che rispecchia l'architettura adottata dal framework Angular2 e per la parte di *Back-end* è stato scelto di usare il design pattern Three-tier.

6.1.1 MVVM

- **Scopo**: MVVM facilita la separazione dello sviluppo della GUI_G, dallo sviluppo della *business logic*$_G$ o meglio del Model. La View-Model è un convertitore di valori, che ha la responsabilità di convertire gli Objects ricevuti dal Model in modo da poter essere gestiti e riprodotti facilmente dalla View.
- **Utilizzo**: Viene utilizzato nella parte di *Front-end* lasciando la gestione della logica dell'applicazione al *Back-end*. La comunicazione con il Back-end avviene tramite il package `Front-end::Model::Services` che gestisce la richiesta di generazione codice e la richiesta di template già pronti da inserire nell'editor. La parte di Model ha al suo interno la business-logic con tutti i relativi blocchi implementabili nell'editor e tutti i comandi che possono essere usati dal client. La View-Model

si occupa di gestire la comunicazione tra la parte di Model e la View. È il design pattern utilizzato dal *framework* Angular2, con il quale viene sviluppato il *Front-end*.

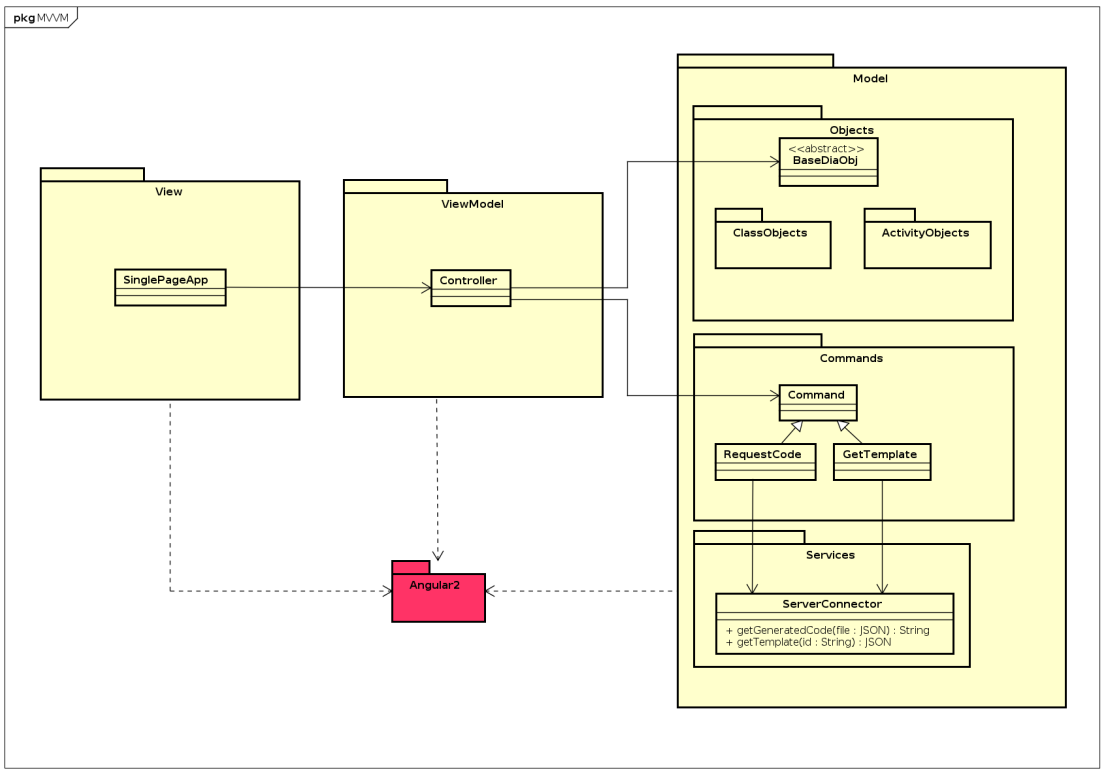

Figura 27: Contestualizzazione MVVM

6.1.2 Three-tier

- **Scopo**: Lo schema Three-tier può essere definito un design pattern, e presenta diverse analogie con il pattern Model-View-Controller. Three-tier prevede la suddivisione dell'architettura in tre strati dedicati rispettivamente all'interfaccia (Presentation-tier), alla logica (Application-tier) e alla gestione dei dati persistenti (Data-tier). L'interfaccia rappresenta il client della parte logica, e la parte logica rappresenta il client della parte dei dati persistenti. Tutti e tre gli strati utilizzano interfacce ben definite conferendo scalabilità e manutenibilità all'applicazione.
- **Utilizzo**: Viene utilizzato nella parte di *Back-end*. La comunicazione con la parte di *Front-end* avviene tramite lo strato Presentation-tier che nel nostro caso non è proprio uno strato di presentazione grafica ma si occupa di ricevere richieste o inviare risorse al client-side utilizzando il framework Express. Nello strato di Application-tier è gestita tutta la parte di business-logic che implementa l'algorit-

mo di generazione codice, la gestione degli errori e la funzionalità di compressione file. L'ultimo strato si occupa della gestione dei dati persistenti; è presente l'intero database che raccoglie tutti i template che possono essere richiesti dall'utente nella View del *Front-end* per velocizzare e automatizzare il più possibile la stesura del diagramma delle classi e delle attività. Lo strato Data-tier utilizza la libreria Mongoose.

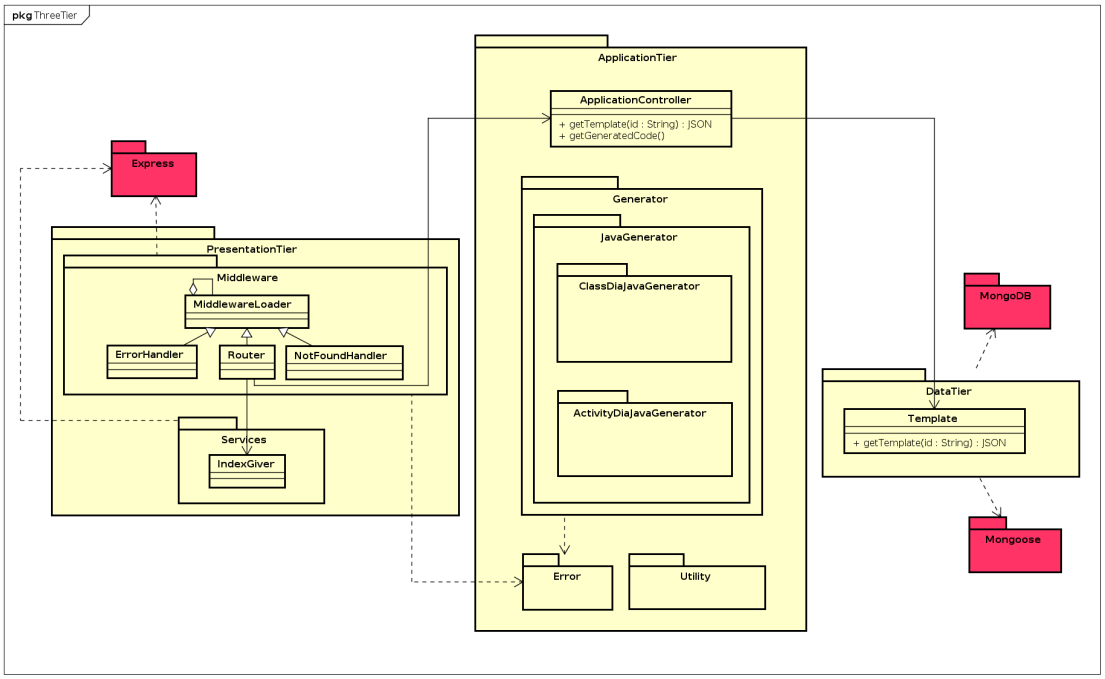

Figura 28: Contestualizzazione Three-tier

6.1.3 Middleware

- **Scopo**: Abbiamo scelto di utilizzare questo design pattern per fornire un *intermediario* tra i componenti del *Front-end* e i componenti del *Back-end*. Questo pattern in generale è utilizzato nello sviluppo e nella gestione di sistemi distribuiti complessi.
- **Utilizzo**: È presente all'interno dello strato Presentation-tier appartenente al design pattern architetturale Three-tier. Viene utilizzato da Express per fornire una libreria di funzioni comuni. Definisce una serie di livelli (o funzioni) per gestire le varie richieste client-side e richiamare i rispettivi handler o funzioni di routing. Tutti i componenti del middleware sono collegati l'uno con l'altro e ricevono a turno una richiesta in ingresso, finchè uno di questi non decide di partire con l'elaborazione per poi chiamare la funzione *next*. Come si può notare è molto legato

a Chain of Responsibility che verrà descritto in seguito. Tutti i componenti di Express vengono utilizzati con il metodo *use*. Nella progettazione architetturale è utilizzato nel package `Back-end::PresentationTier::Middleware`.

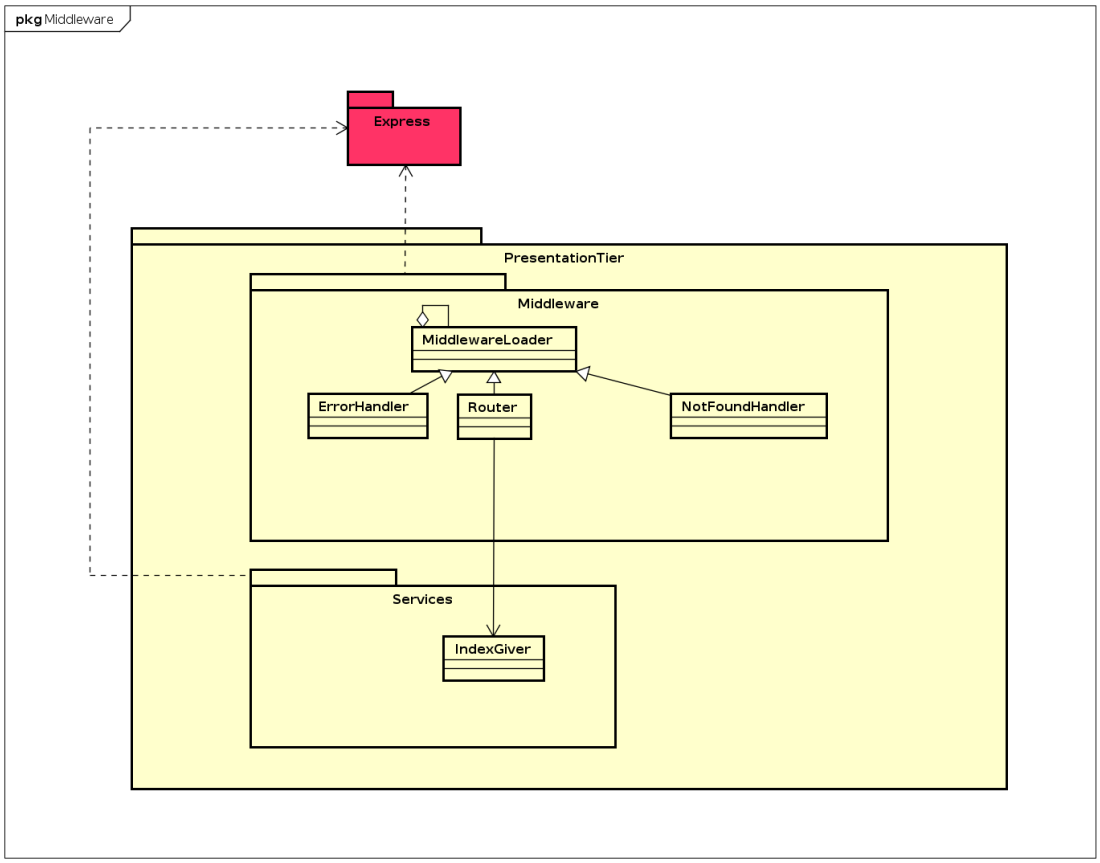

Figura 29: Contestualizzazione Middleware

6.2 Design pattern creazionali

6.2.1 Abstract factory

- **Scopo**: Indirizza il problema della creazione di un oggetto alle relative sottoclassi senza specificarne la classe esatta. Fornisce un'interfaccia per creare l'oggetto, ma lascia che le sottoclassi decidano quale oggetto istanziare.
- **Utilizzo**: Il pattern viene utilizzato dal client per richiedere le istanze dei componenti relativi ad uno dei due diagrammi. La classe `Front-end::View::DiagramFactory` rappresenta l'interfaccia Abstract Factory del design pattern per le operazioni che creano dei prodotti astratti. `Front-end::View::ClassFactory` e

`Front-end::View::ActivityFactory` rappresentano le classi ConcreteFactory del design pattern che implementano le funzioni di istanziazione dei prodotti. Le classi `Front-end::View::DiagramEditor` e `Front-end::View::DiagramPalette` rappresenta le interfacce AbstractProduct del design pattern e dichiarano un'interfaccia per i tipi degli oggetti prodotti. Le classi `Front-end::View::ClassDiagramEditor`, `Front-end::View::ActivityDiagramEditor`, `Front-end::View::ClassDiagramPalette` e `Front-end::View::ActivityDiagrampalette` rappresentano le classi Product del design pattern e definiscono un oggetto prodotto creato dalle corrispondenti factory concrete, implementate dall'interfaccia AbstractProduct.

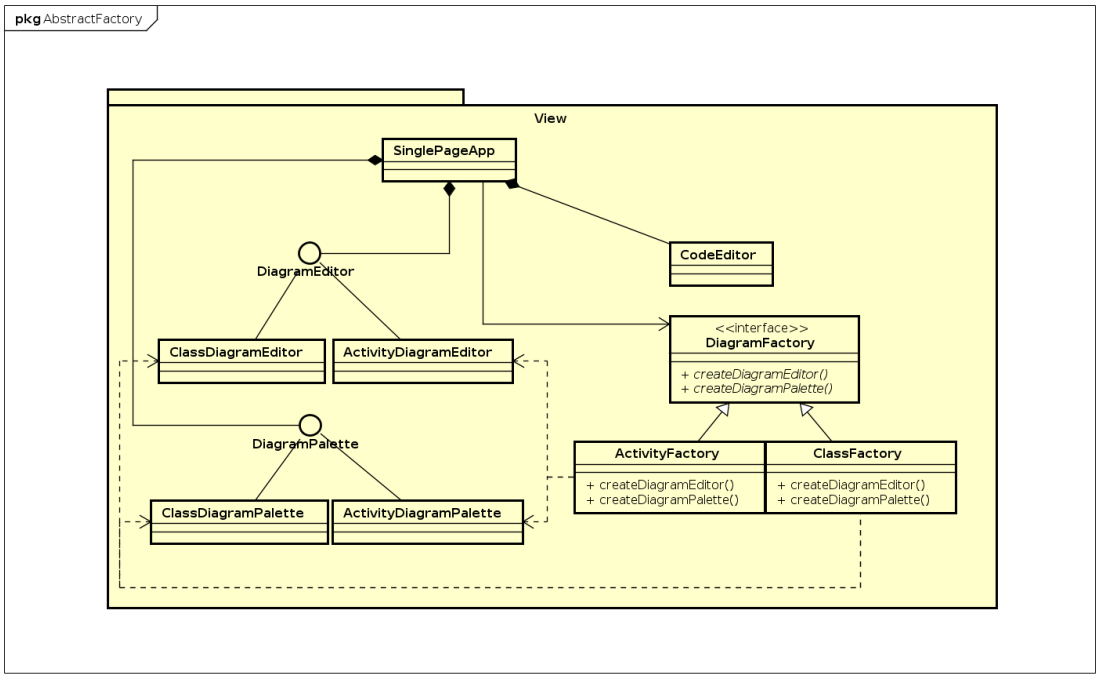

Figura 30: Contestualizzazione AbstractFactory

6.3 Design pattern comportamentali

6.3.1 Command

- **Scopo**: Permette di isolare la porzione di codice che effettua un'azione dal codice che ne richiede l'esecuzione; l'azione è incapsulata nell'oggetto command. L'obiettivo è di rendere variabile l'operazione del client senza però conoscere i dettagli dell'operazione stessa.
- **Utilizzo**: Il pattern viene utilizzato nella parte logica del *Front-end*, precisamente nel package `Front-end::Model::Commands`. La classe `Front-end::View::SinglePageApp`

rappresenta il Client del design pattern che si occupa di creare un ConcreteCommand e settare il suo Receiver. La classe `Front-end::Model::Commands::Command` rappresenta il Command del design pattern che dichiara un'interfaccia per l'esecuzione delle operazioni. La classe `Front-end::ViewModel::Controller` rappresenta l'invoker del design pattern che chiede al Command di eseguire la richiesta. Le classi `Front-end::Model::Commands::GetTemplate` e `Front-end::Model::Commands::RequestCode` rappresentano i ConcreteCommand che definiscono un legame tra il Receiver e l'azione. Infine il Receiver è implementato dalla classe `Front-end::Model::Services::ServerConnector`, che sa come eseguire le operazioni associate all'esecuzione della richiesta di template o di generazione codice comunicando con il server.

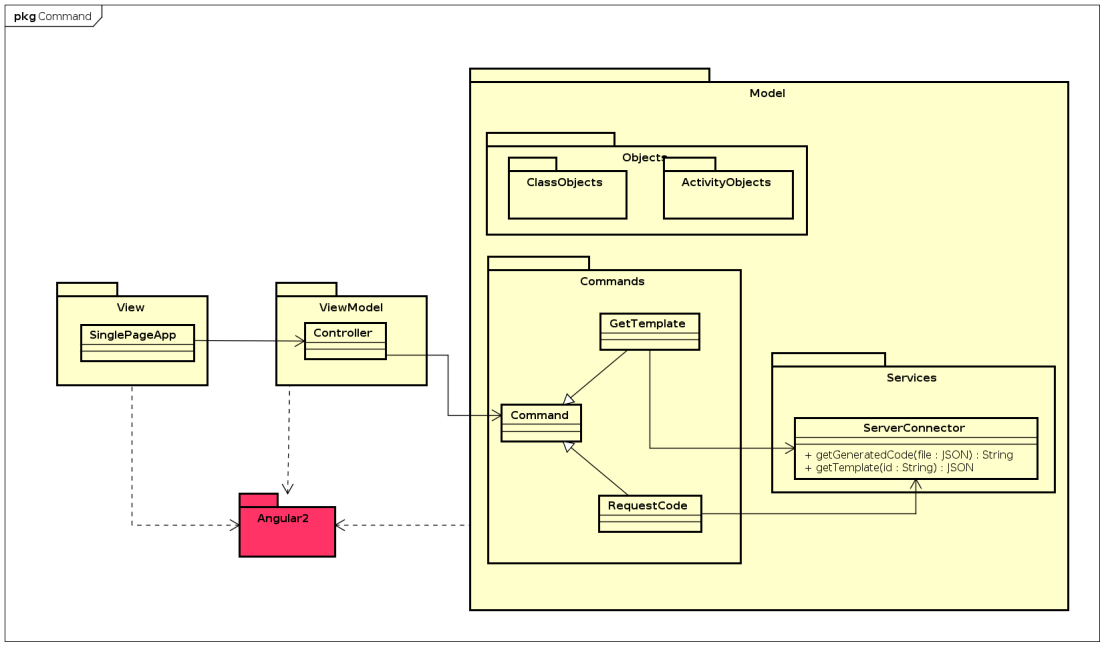

Figura 31: Contestualizzazione Command

6.3.2 Chain of responsibility

- **Scopo**: Permette ad un oggetto a cui viene effettuata una richiesta di eseguire le richieste di più oggetti. In questo modo si può ottenere una separazione tra il richiedente della richiesta e l'oggetto destinatario. Tutti gli oggetti destinatari sono collegati tra di loro. Ogni nodo della catena se può esaudisce la richiesta altrimenti delega l'azione al nodo successivo. La catena viene attraversata finchè un nodo non esegue l'azione richiesta dal mittente.
- **Utilizzo**: Express usa chain of responsibility per la gestione dei middleware e del routing. Come già accennato, Chain of Responsibility è molto legato al pattern

Middleware. Viene utilizzato all'interno del package `Back-end::PresentationTier::Middleware`. La classe `Back-end::PresentationTier::Middleware::MiddlewareLoader` gestisce la richiesta scorrendo tutta la lista delle sottoclassi e richiamando il metodo *next* finchè una di queste non può soddisfarla.

6.3.3 Strategy

- **Scopo**: Permette di definire una famiglia di algoritmi e renderli intercambiabili, in modo che possano cambiare indipendentemente dal client che ne fa uso. In un progetto software che si focalizza sulla scalabilità ed evoluzione è fondamentale poter effettuare modifiche alle procedure in maniera meno intrusiva possibile.
- **Utilizzo**: Viene utilizzato all'interno del package `Back-end::ApplicationTier::Generator` in modo da poter permettere l'inserimento di altri algoritmi di generazione di codice senza modificare la classe `Back-end::ApplicationTier::ApplicationController` che ne fa uso. Questo permette un'espansione della parte logica del *Back-end* in maniera semplice, ottimizzando lo sviluppo del software in un'ottica di massima scalabilità.

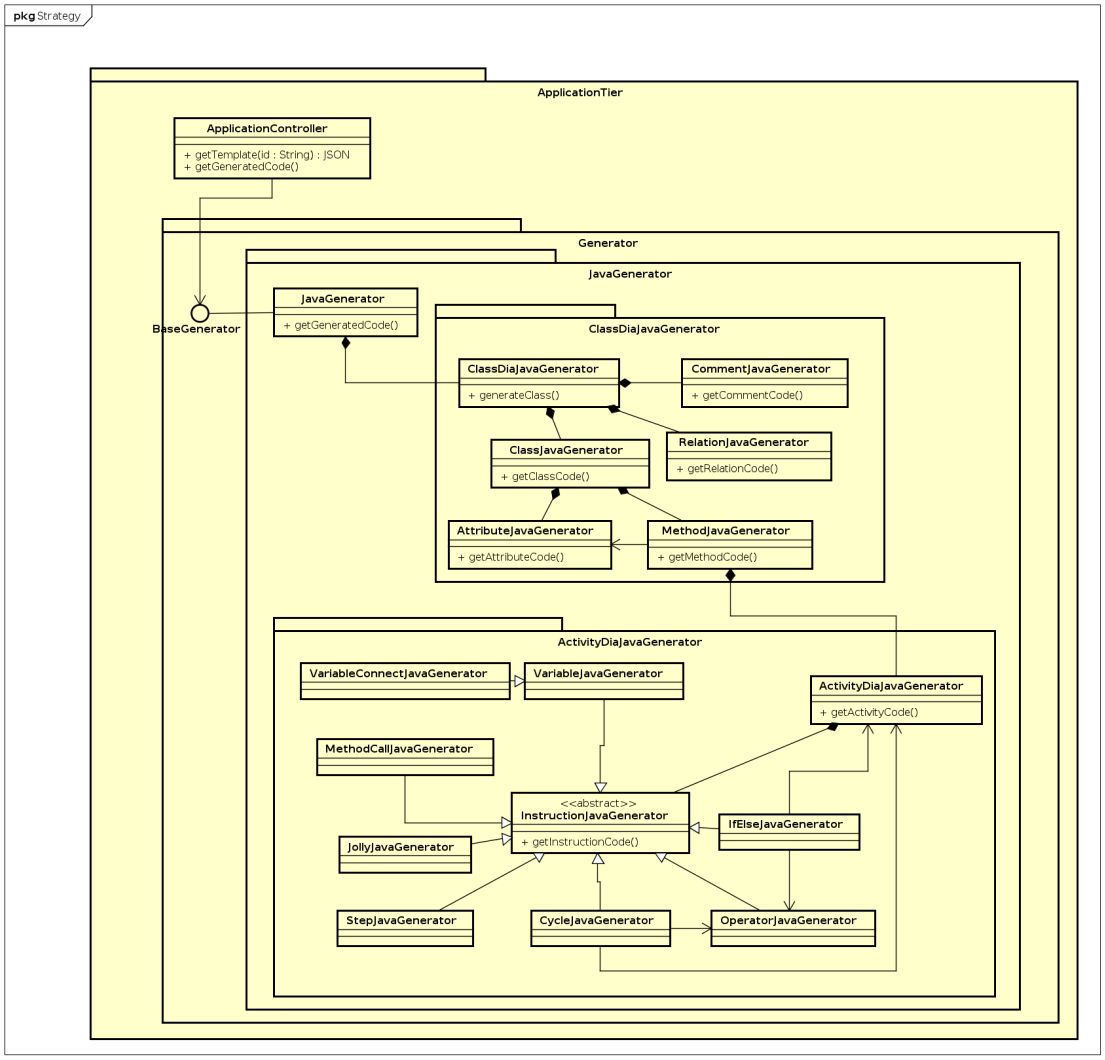

Figura 32: Contestualizzazione Strategy

7 Diagrammi delle attività

Vengono riportati in seguito i diagrammi delle attività realizzati durante la progettazione architetturale, i quali servono a descrivere le interazioni che l'utente ha con il sistema SWEDesigner. Abbiamo suddiviso i diagrammi in due categorie:

- quelli che raffigurano le azioni che l'utente può compiere all'interno del diagramma delle classi;
- quelli che raffigurano le azioni che l'utente può compiere all'interno del diagramma delle attività.

Per il diagramma delle classi e per quello delle attività sono stati individuati tre diagrammi delle attività che riguardano rispettivamente: inserimento di un nuovo blocco, modifica di un blocco e rimozione di un blocco.

7.1 Workflow

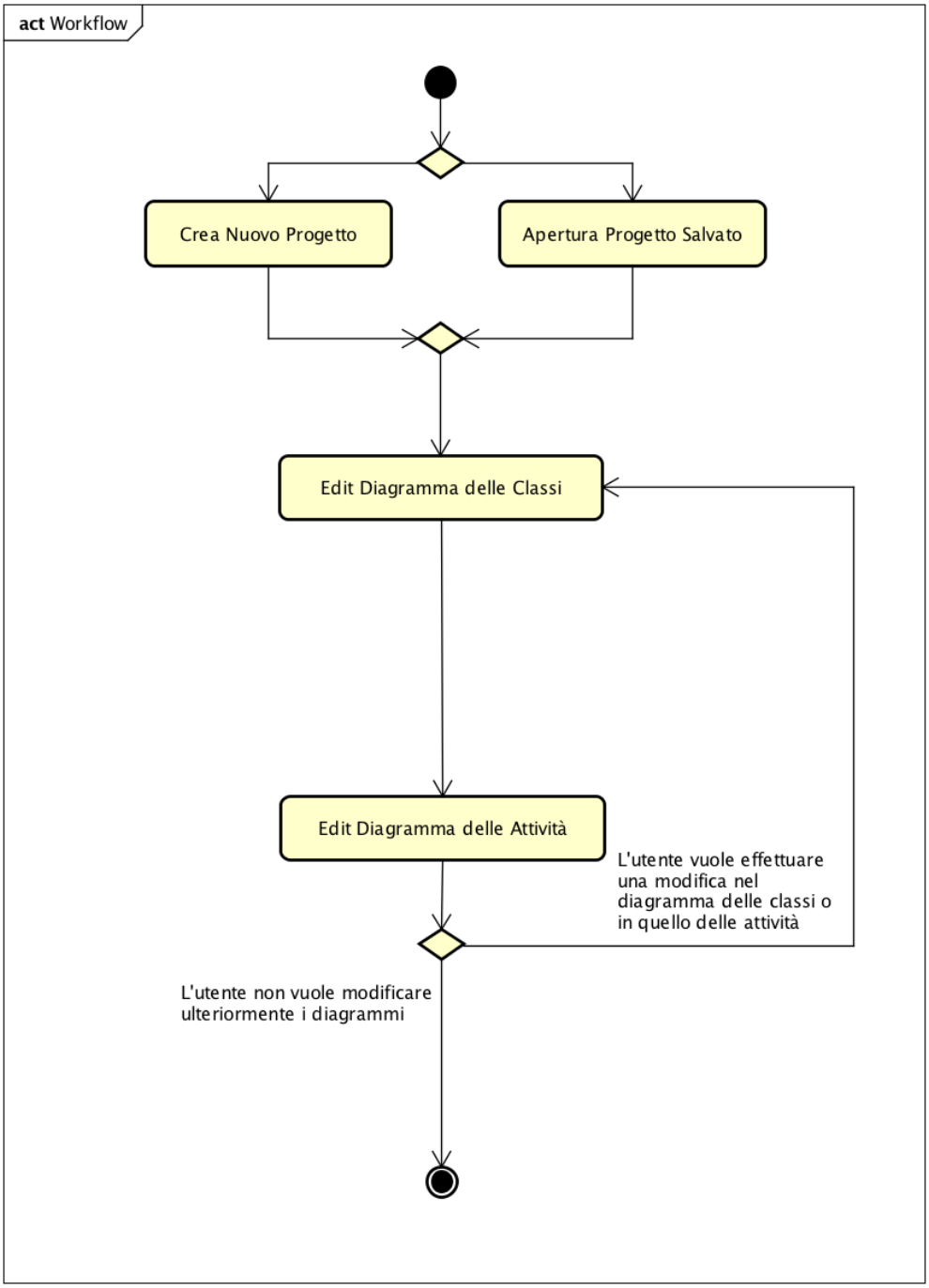

Figura 33: Diagramma delle attività Workflow

Tramite SWEDesigner l'utente può, dopo aver creato o caricato un nuovo progetto, modificare il diagramma delle classi, quello delle attività oppure entrambi.

7.2 Inserimento blocco diagramma delle classi

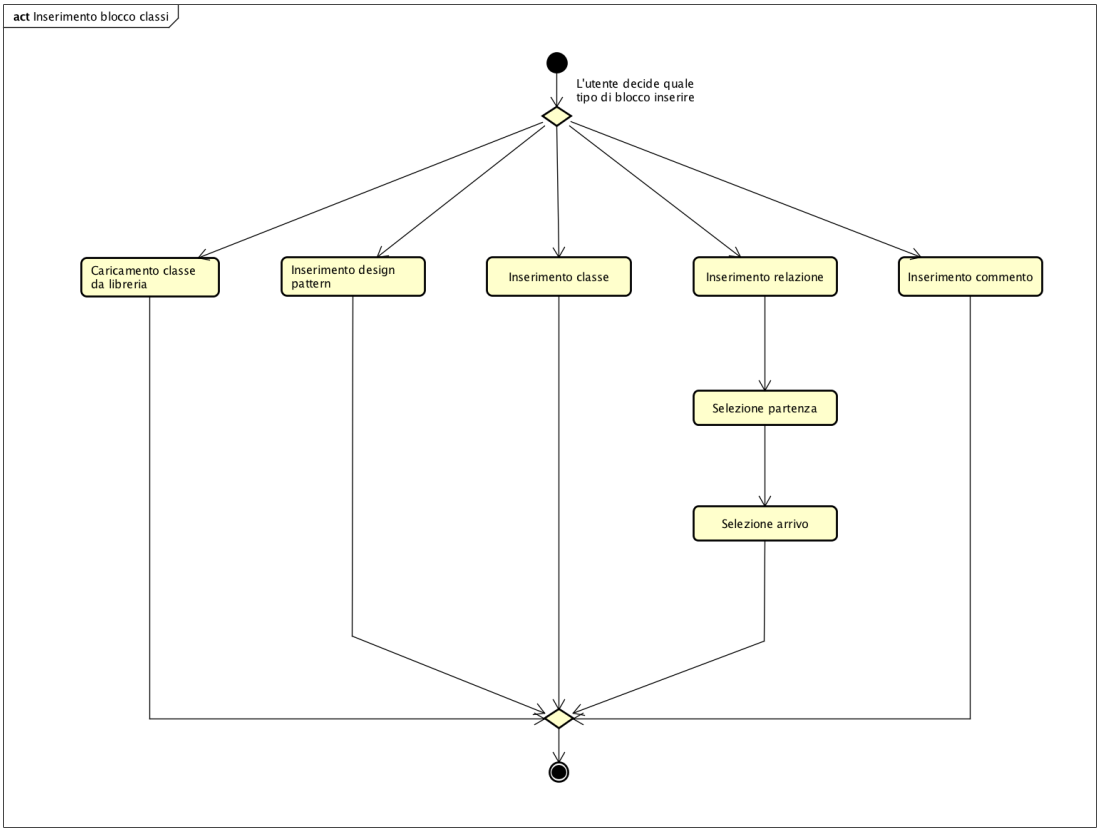

Figura 34: Diagramma attività inserimento blocco in diagramma delle classi

Quando l'utente vuole inserire un nuovo blocco nel diagramma delle classi ha a disposizione quattro tipologie di blocco:

- **Caricamento classe da libreria**: l'utente può inserire una classe caricandola dalla libreria;
- **Design pattern**: l'utente può inserire un design pattern scegliendolo tra quelli disponibili nella libreria (Adapter, Decorator, Facade, Singleton, Builder, Abstract Factory, Command, Iterator, Observer, Strategy, Template, MVC);
- **Classe**: l'utente può inserire una nuova classe vuota, dove il nome, la priorità, ed il colore vengono inizializzati a valore di default;

- **Relazione**: l'utente può inserire una nuova relazione tramite il form di inserimento relazione. In seguito selezionerà la partenza e l'arrivo di tale relazione;
- **Commento**: l'utente può inserire un nuovo commento vuoto tramite il form di inserimento commento.

7.3 Modifica blocco diagramma delle classi

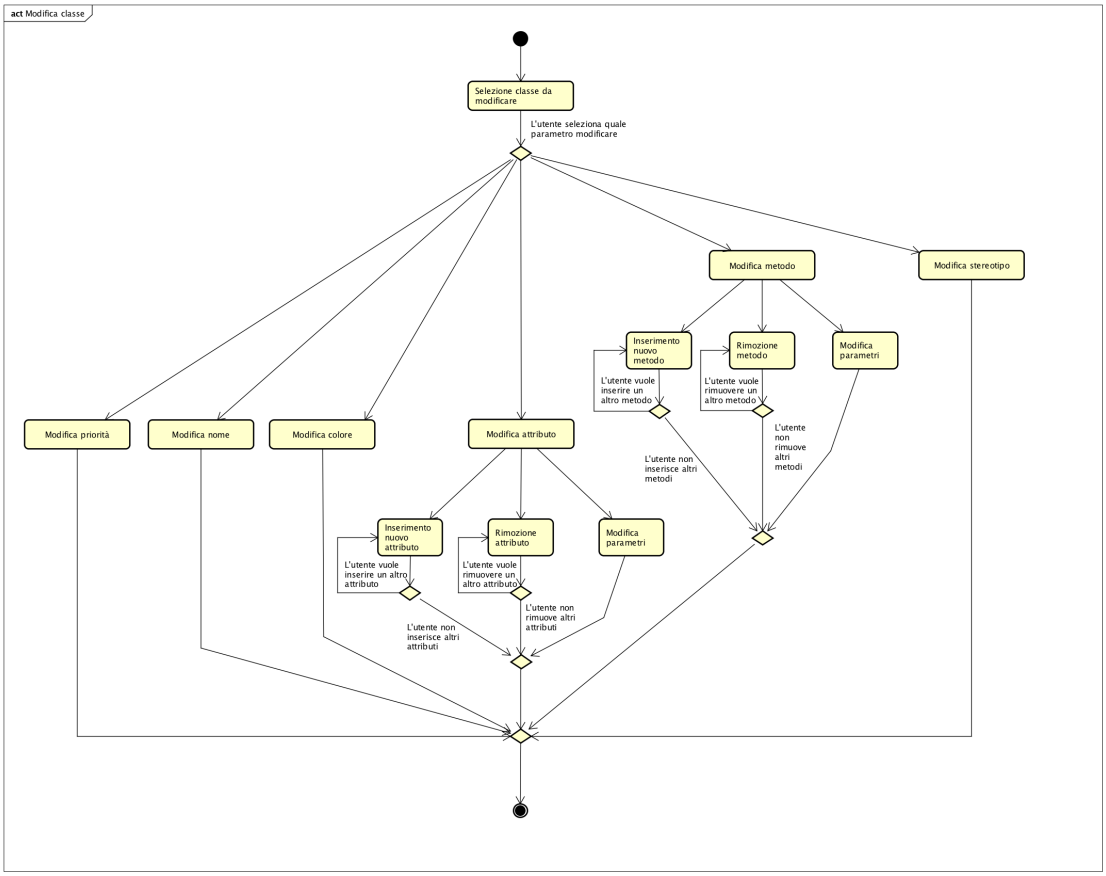

Figura 35: Diagramma attività modifica blocco in diagramma delle classi

Quando l'utente vuole modificare una classe del diagramma delle classi può modificare:

- **Priorità**: l'utente può modificare o inserire la priorità della classe tramite il form per l'inserimento della priorità che poi verrà graficamente visualizzata;
- **Nome**: l'utente può modificare il nome di una classe tramite il form per la modifica del nome che poi verrà graficamente visualizzato;

- **Colore**: l'utente può modificare il colore di una classe scegliendolo tra quelli disponibili;
- **Attributo**: l'utente può modificare un attributo in una classe, inserendolo come nuovo, rimuovendolo o modificandone i parametri;
- **Metodo**: l'utente può modificare un metodo di una classe, inserendolo come nuovo, rimuovendolo o modificandone i parametri;
- **Stereotipo**: l'utente può modificare uno stereotipo di una classe tramite il form per la modifica dello stereotipo che poi verrà graficamente visualizzato.

7.4 Rimozione blocco diagramma delle classi

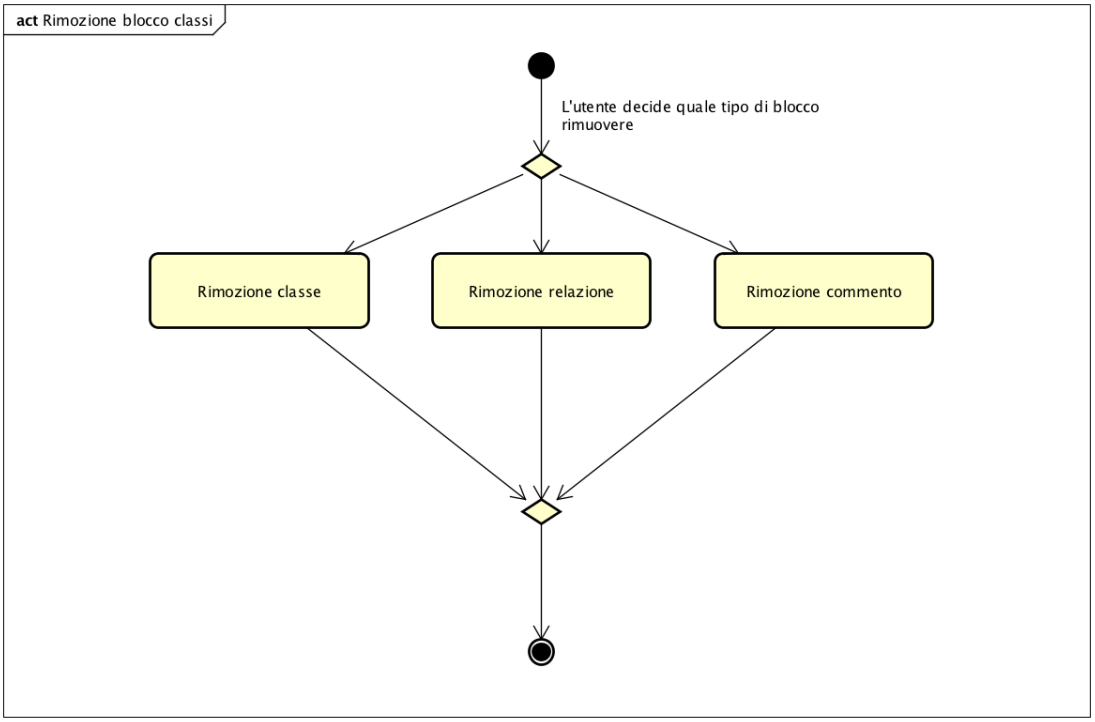

Figura 36: Diagramma attività rimozione blocco in diagramma delle classi

L'utente può decidere di rimuovere un blocco all'interno del diagramma delle classi. Suddetto blocco può essere una classe, una relazione oppure un commento. Nel caso in cui si è tentato di rimuovere una classe con relazioni il sistema fornirà un messaggio d'errore.

7.5 Inserimento blocco diagramma delle attività

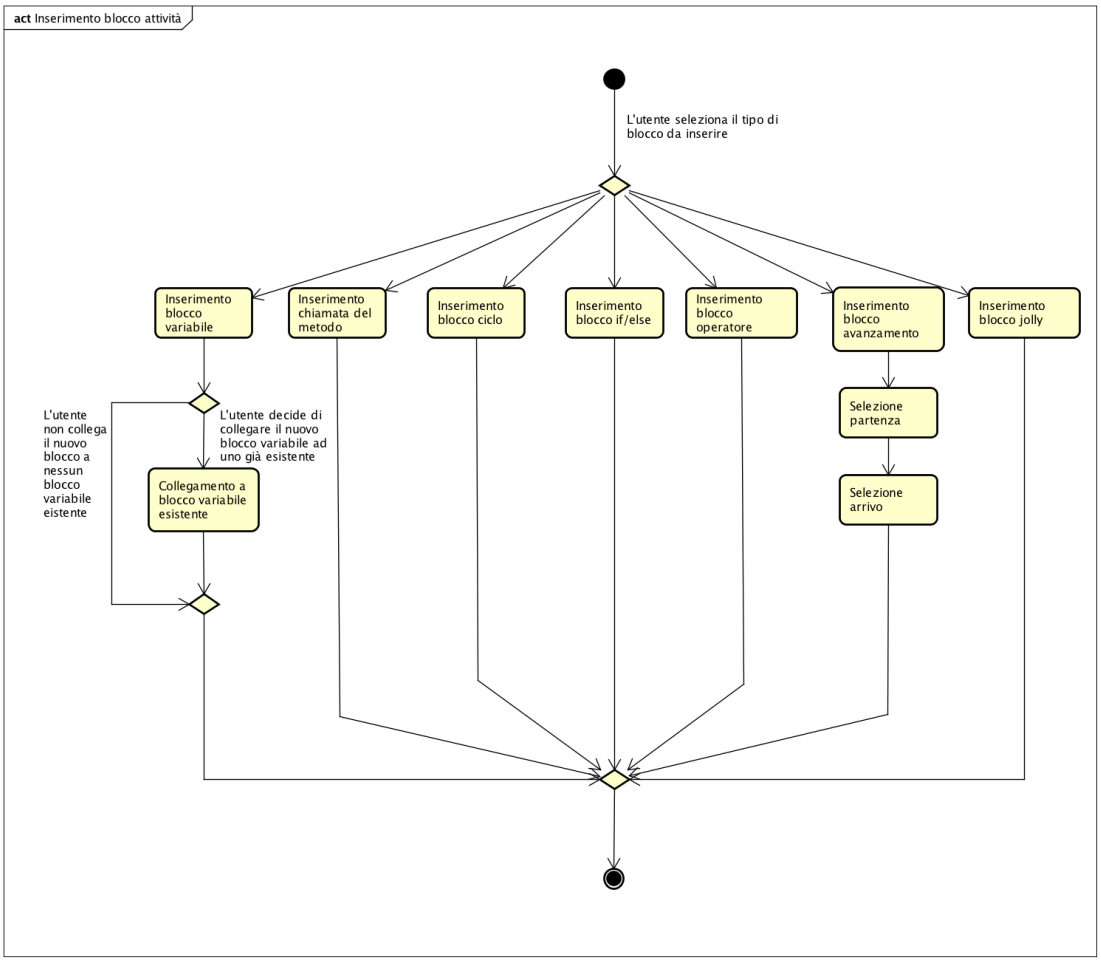

Figura 37: Diagramma attività inserimento blocco in diagramma delle attività

Quando l'utente vuole inserire un nuovo blocco all'interno del diagramma delle attività ha a disposizione sette tipologie di blocco:

- **Blocco variabile**: l'utente può inserire un nuovo blocco variabile nel diagramma delle attività con la possibilità di collegarlo ad un blocco variabile già esistente per fare in modo che entrambi i blocchi si riferiscano alla stessa variabile;
- **Blocco chiamata del metodo**: l'utente può inserire un nuovo blocco chiamata del metodo nel diagramma delle attività che verrà visualizzato con il campo relativo al metodo da invocare vuoto;

- **Blocco ciclo**: l'utente può inserire un nuovo blocco ciclo nel diagramma delle attività che verrà visualizzato con la condizione ed il corpo del ciclo vuoti;
- **Blocco If/else**: l'utente può inserire un nuovo blocco if/else nel diagramma delle attività che verrà visualizzato con la condizione ed il corpo vuoti;
- **Blocco operatore**: l'utente può inserire un nuovo blocco operatore nel diagramma delle attività che verrà visualizzato con un operatore di default. L'elemento di sinistra, di destra ed il risultato vengono impostati a vuoti;
- **Blocco avanzamento**: l'utente può inserire un nuovo blocco avanzamento nel diagramma delle attività selezionando il blocco di partenza e di arrivo del blocco di avanzamento;
- **Blocco jolly**: l'utente può inserire un nuovo blocco jolly nel diagramma delle attività che verrà visualizzato con la descrizione vuota.

7.6 Modifica blocco diagramma delle attività

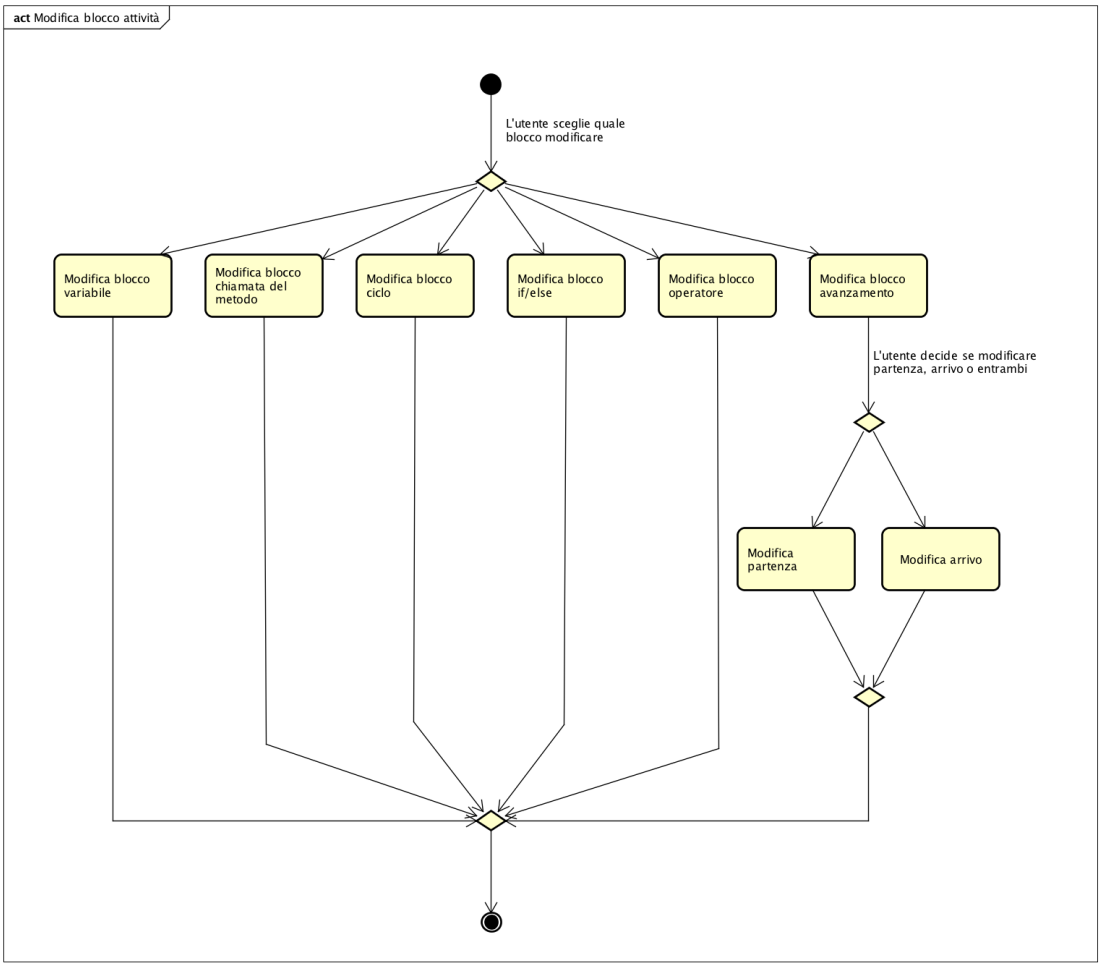

Figura 38: Diagramma attività modifica blocco in diagramma delle attività

Quando l'utente vuole modificare un blocco del diagramma delle attività può modificare:

- **Blocco variabile**: l'utente può modificare un blocco variabile esistente del diagramma delle attività e la modifica verrò poi visualizzata dal sistema;
- **Blocco chiamata del metodo**: l'utente può modificare un blocco chiamata del metodo del diagramma delle attività cambiando il metodo da invocare, i parametri di invocazione o entrambi;
- **Blocco ciclo**: l'utente può modificare un blocco ciclo del diagramma delle attività cambiando la condizione del ciclo, il corpo o entrambi;

- **Blocco if/else**: l'utente può modificare un blocco if/else del diagramma delle attività cambiando la condizione del if, il corpo del if oppure il corpo del else;
- **Blocco operatore**: l'utente può modificare un blocco operatore del diagramma delle attività cambiando l'elemento sinistra, l'elemento destra oppure l'elemento a cui assegnare il risultato dell'operazione;
- **Blocco avanzamento**: l'utente può modificare un blocco avanzamento del diagramma delle attività cambiando il blocco di partenza, il blocco di arrivo o entrambi.

7.7 Rimozione blocco diagramma delle attività

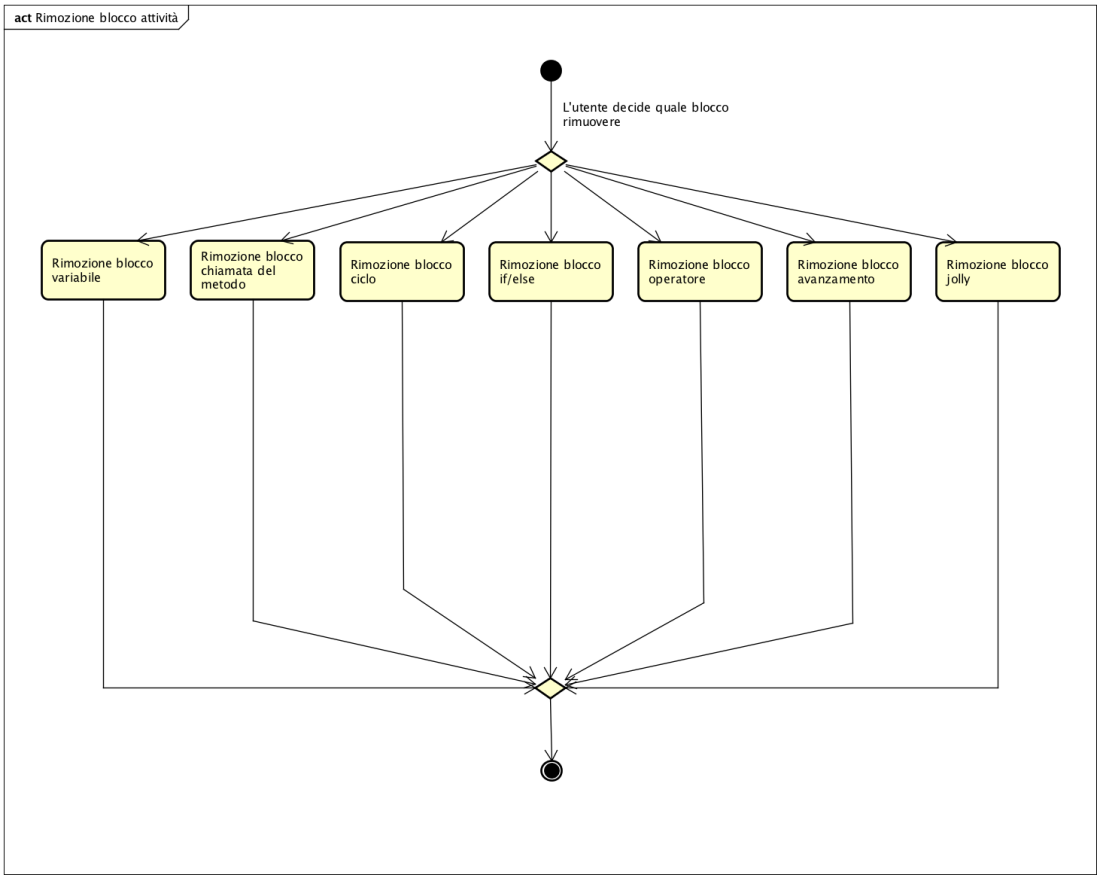

Figura 39: Diagramma delle attività rimozione blocco in diagramma delle attività

L'utente può decidere di rimuovere un blocco all'interno del diagramma delle attività. Suddetto blocco può essere un blocco variabile, un blocco chiamata del metodo, un

blocco ciclo, un blocco if/else, un blocco operatore, un blocco avanzamento o un blocco jolly.

8 Diagrammi di sequenza

In questa sezione sono riportati i diagrammi di sequenza per le due principali funzionalità del back-end: la generazione del codice a partire dal file JSON generato dal front-end e l'interrogazione del database per la restituzione degli oggetti presenti in libreria. Vengono riportati i diagrammi di sequenza a differenza di quelli delle attività per evidenziare maggiormente le interazioni fra le classi.

8.1 Generazione Codice

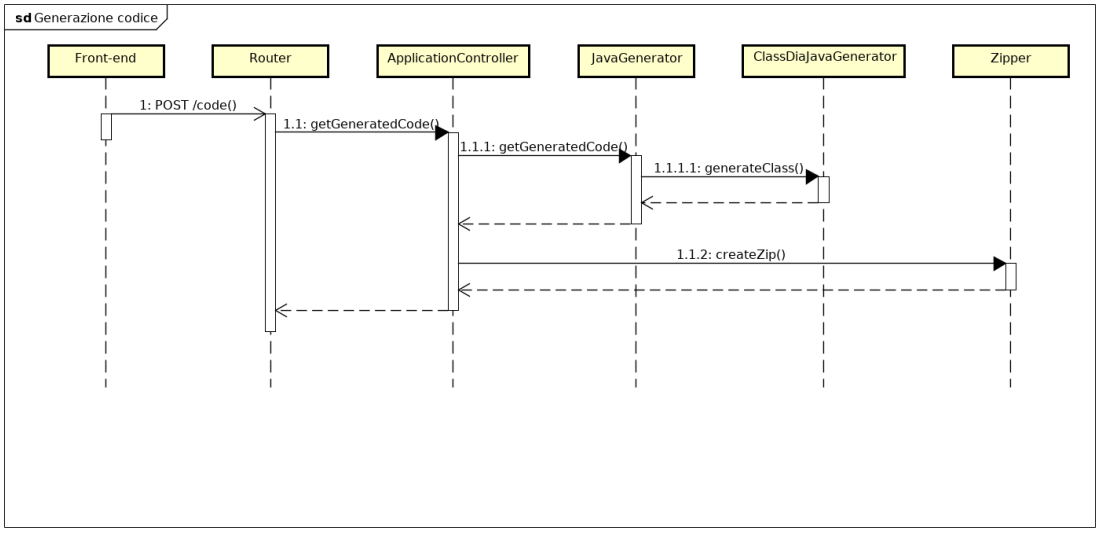

Figura 40: Diagramma di Sequenza Generazione Codice

A seguito di una richiesta HTML di generazione codice da parte del front-end il back-end invoca il metodo di generazione del codice che analizza il diagramma delle classi generando il codice opportuno. Viene successivamente generato un file compresso contenente tutti i file contenenti il codice delle classi generate.

8.2 Generazione Codice Classe

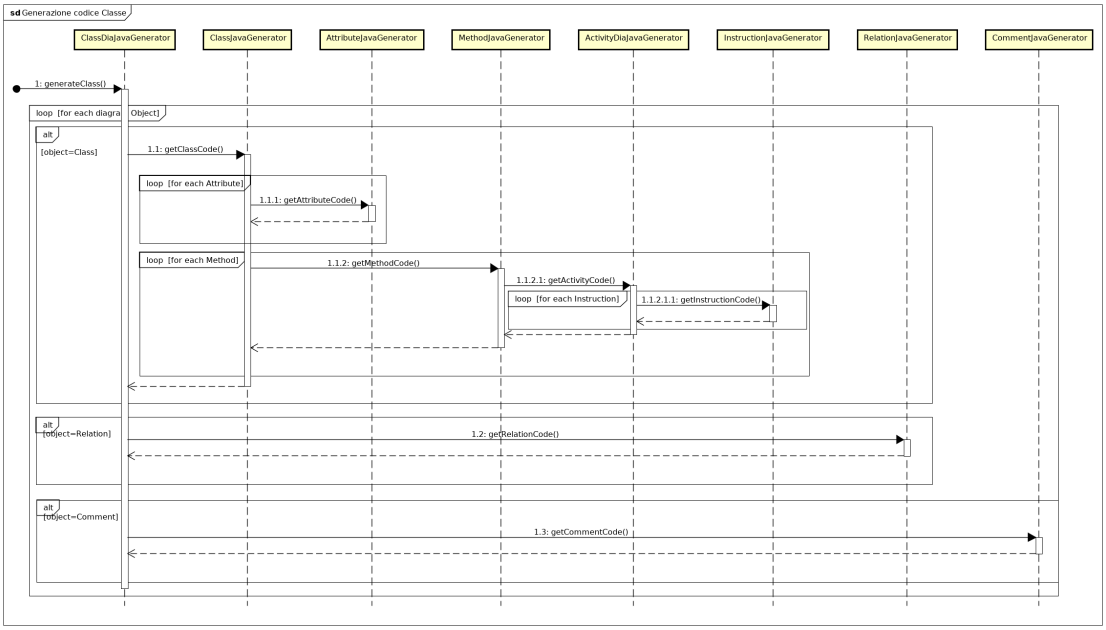

Figura 41: Diagramma di Sequenza Generazione Codice Classe

A seguito di una invocazione del metodo di generazione del codice di un diagramma delle classi viene invocato il metodo che attraversa il diagramma generando il codice di ciascuno degli oggetti che incontra.

Oggetto incontrato	Azione
Classe	• Generazione del codice per ciascun attributo; • Generazione del codice per ciascun metodo (invocando la generazione del codice per il diagramma delle attività associato).
Relazione	Generazione del codice per la relazione
Commento	Generazione del codice per il commento

Tabella 3: Diagramma di sequenza - Generazione codice classe

8.3 Ottenimento template

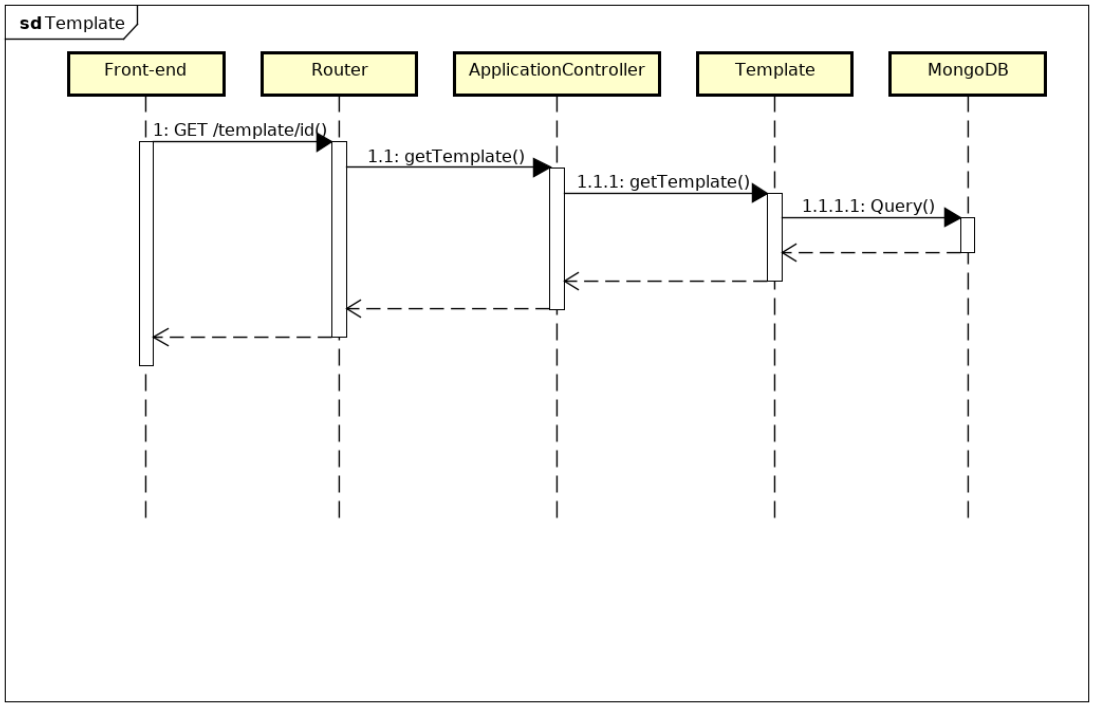

Figura 42: Diagramma di Sequenza Ottenimento template

A seguito di una richiesta HTML di ottenimento template (corredata di id del template) da parte del front-end il back-end invoca il metodo di ritorno del template. Questo metodo interroga MongoDB e ritorna il file JSON contenente il template di id richiesto.

9 Database

9.1 Descrizione

Il server dell'applicazione si appoggia su un database non relazionale MongoDB dove sono contenuti i template. Questi possono essere richiesti dall'utente per automatizzare il più possibile la creazione del diagramma delle classi e delle attività. L'architettura è molto semplice, si basa su una sola tabella dove saranno raccolti una serie di file JSON. Tali file rappresentano dei template di alcuni design pattern più ricorrenti, di classi specifiche già pronte con tutti i metodi definiti e di alcuni diagrammi delle attività che possono andare a strutturare il corpo di metodi creati dall'utente. Il Database viene gestito dalla classe `Back-end::DataTier::Template` che esegue la richiesta di un template tramite delle $query_G$ utilizzano la libreria Mongoose.

9.2 Struttura

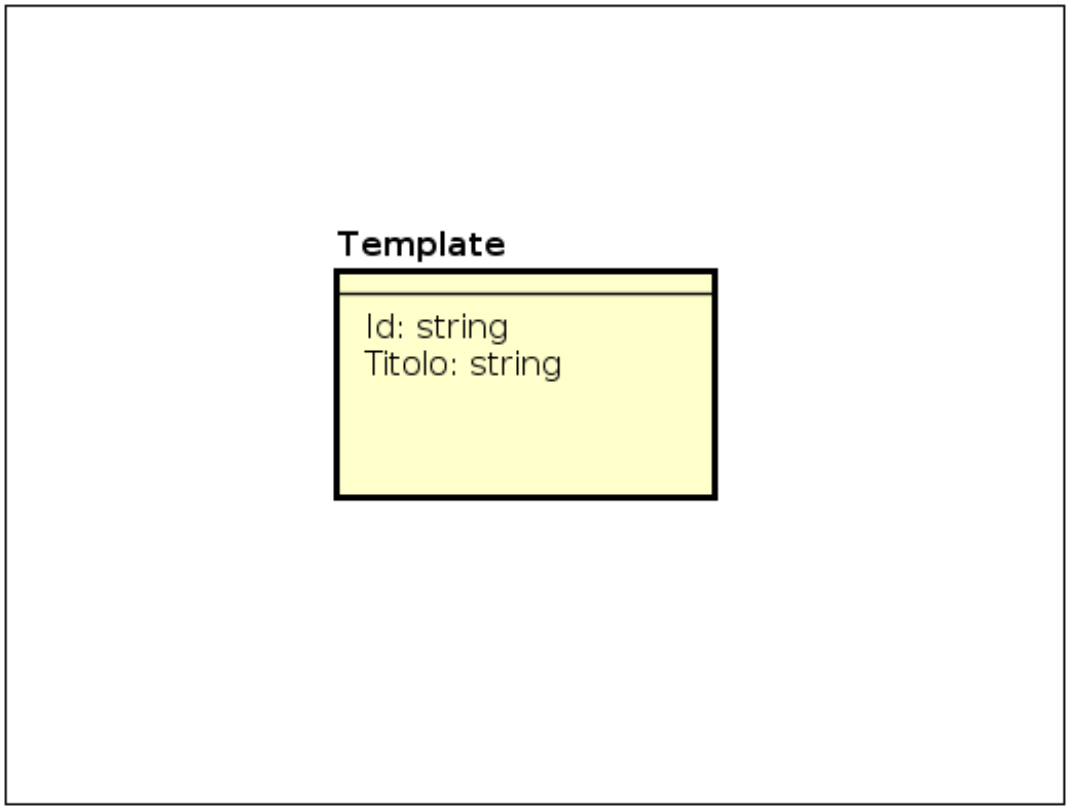

Figura 43: Tabella Template del database

I JSON all'interno della tabella Template saranno distinguibili tramite due campi:

- **Id**
- **Titolo**

10 Tracciamento

10.1 Classi-Requisiti

Classe	Requisiti
Front-end::View::SinglePageApp	R1O0 R1O2 R1O4 R1O5 R1O6.3 R1O7
Front-end::View::ClassDiagramEditor	R1O4 R1O4.1
Front-end::View::DiagramPalette	R1O4 R1O4.1 R1O5
Front-end::View::ClassDiagramPalette	R1O4 R1O4.1
Front-end::Model::Objects::BaseDiaObj	R1O4.1
Front-end::Model::Objects::ClassObjects::ClassDiaObj	R1O4.1
Front-end::View::ClassFactory	R1O4.1
Front-end::View::ClassDiagramEditor	R1O4.4 R1O4.4.1 R1O4.4.2 R1D4.4.3 R1O4.4.5 R1O4.4.5.1 R1O4.4.5.2 R1O4.4.5.3 R1O4.4.5.4 R1O4.4.8 R1O4.4.8.5 R1O4.4.9 R1O4.4.10 R1O4.5 R1O4.6 R1O4.7 R1O4.8 R1O4.9
Front-end::ViewModel::Controller	R1O4.4 R1O5 R1O6

Front-end::Model::Objects::ClassObjects::Dependency	R1O4.7.2
Front-end::Model::Objects::ClassObjects::Association	R1O4.7.2
Front-end::Model::Objects::ClassObjects::Aggregation	R1O4.7.2
Front-end::Model::Objects::ClassObjects::Composition	R1O4.7.2
Front-end::Model::Objects::ClassObjects::Generalization	R1O4.7.2
Front-end::Model::Objects::ClassObjects::Comment	R1D4.9 R1D4.11
Front-end::Model::Objects::ActivityObjects::ActivityDiaObj	R1O5
Front-end::View::ActivityDiagramEditor	R1O5 R1O5.2 R1O5.2.1.1 R1O5.2.1.2 R1O5.2.1.3 R1O5.2.1.4 R1O5.2.1.5 R1O5.2.1.6 R1O5.2.1.7 R1O5.2.2 R1O5.2.2.1 R1O5.2.2.2 R1O5.2.2.3 R1O5.2.2.4 R1O5.2.2.5 R1O5.2.2.6 R1O5.2.2.7 R1O5.2.3 R1O5.3
Front-end::View::ActivityDiagramPalette	R1O5
Front-end::View::ActivityFactory	R1O5
Front-end::Model::Objects::ActivityObjects::VariableObj	R1O5.2.1.1
Front-end::Model::Objects::ActivityObjects::MethodCallObj	R1O5.2.1.2
Front-end::Model::Objects::ActivityObjects::CycleObj	R1O5.2.1.3
Front-end::Model::Objects::ActivityObjects::IfElseObj	R1O5.2.1.4
Front-end::Model::Objects::ActivityObjects::OperatorObj	R1O5.2.1.5
Front-end::Model::Objects::ActivityObjects::StepObj	R1O5.2.1.6
Front-end::Model::Objects::ActivityObjects::JollyObj	R1O5.2.1.7
Front-end::View::CodeEditor	R1O6
Front-end::Model::Commands::Command	R1O6 R1O6.3 R1O7

	R1O9
Front-end::Model::Commands::RequestCode	R1O6
Back-end::ApplicationTier::Generator:: JavaGenerator::JavaGenerator	R1O9
Back-end::ApplicationTier::Generator::JavaGenerator ::ClassDiaJavaGenerator	R1O9
Back-end::ApplicationTier::Generator::JavaGenerator ::ActivityDiaJavaGenerator::ActivityDiaJavaGenerator	R1O9

Tabella 4: Tracciamento Classi-Requisiti

10.2 Requisiti-Classi

Requisito	Classi
R1O0	Front-end::View::SinglePageApp
R1O2	Front-end::View::SinglePageApp
R1O4	Front-end::View::ClassDiagramEditor Front-end::View::ClassFactory Front-end::View::DiagramPalette Front-end::View::ClassDiagramPalette Front-end::View::SinglePageApp
R1O4.1	Front-end::Model::Objects::BaseDiaObj Front-end::View::DiagramPalette Front-end::Model::Objects::ClassObjects::ClassDiaObj Front-end::View::ClassDiagramPalette Front-end::Model::Objects::ClassObjects::ClassFactory Front-end::View::ClassDiagramEditor
R1O4.4	Front-end::View::ClassDiagramEditor Front-end::ViewModel::Controller
R1O4.4.1	Front-end::View::ClassDiagramEditor
R1O4.4.2	Front-end::View::ClassDiagramEditor
R1D4.4.3	Front-end::View::ClassDiagramEditor
R1O4.4.5	Front-end::View::ClassDiagramEditor
R1O4.4.5.1	Front-end::View::ClassDiagramEditor
R1O4.4.5.2	Front-end::View::ClassDiagramEditor
R1O4.4.5.3	Front-end::View::ClassDiagramEditor
R1O4.4.5.4	Front-end::View::ClassDiagramEditor
R1O4.4.8	Front-end::View::ClassDiagramEditor
R1O4.4.8.5	Front-end::View::ClassDiagramEditor
R1O4.4.9	Front-end::View::ClassDiagramEditor
R1O4.4.10	Front-end::View::ClassDiagramEditor
R1O4.5	Front-end::View::ClassDiagramEditor
R1O4.6	Front-end::View::ClassDiagramEditor
R1O4.7	Front-end::View::ClassDiagramEditor
R1O4.7.2	Front-end::Model::Objects::ClassObjects::Dependency Front-end::Model::Objects::ClassObjects::Association Front-end::Model::Objects::ClassObjects::Aggregation Front-end::Model::Objects::ClassObjects::Composition Front-end::Model::Objects::ClassObjects::Generalization
R1O4.8	Front-end::View::ClassDiagramEditor
R1D4.9	Front-end::View::ClassDiagramEditor Front-end::Model::Objects::ClassObjects::Comment
R1D4.11	Front-end::Model::Objects::ClassObjects::Comment

R1O5	Front-end::Model::Objects::ActivityObjects::ActivityDiaObj Front-end::View::ActivityDiagram Front-end::View::ActivityFactory Front-end::View::ActivityDiagramPalette Front-end::View::SinglePageApp Front-end::ViewModel::Controller
R1O5.2	Front-end::View::ActivityDiagramEditor
R1O5.2.1.1	Front-end::View::ActivityDiagramEditor Front-end::Model::Objects::ActivityObjects::VariableObj
R1O5.2.1.2	Front-end::View::ActivityDiagramEditor Front-end::Model::Objects::ActivityObjects::MethodCallObj
R1O5.2.1.3	Front-end::View::ActivityDiagramEditor Front-end::Model::Objects::ActivityObjects::CycleObj
R1O5.2.1.4	Front-end::View::ActivityDiagramEditor Front-end::Model::Objects::ActivityObjects::IfElseObj
R1O5.2.1.5	Front-end::View::ActivityDiagramEditor Front-end::Model::Objects::ActivityObjects::OperatorObj
R1O5.2.1.6	Front-end::View::ActivityDiagramEditor Front-end::Model::Objects::ActivityObjects::StepObj
R1O5.2.1.7	Front-end::View::ActivityDiagramEditor Front-end::Model::Objects::ActivityObjects::JollyObj
R1O5.2.2	Front-end::View::ActivityDiagramEditor
R1O5.2.2.1	Front-end::View::ActivityDiagramEditor
R1O5.2.2.2	Front-end::View::ActivityDiagramEditor
R1O5.2.2.3	Front-end::View::ActivityDiagramEditor
R1O5.2.2.4	Front-end::View::ActivityDiagramEditor
R1O5.2.2.5	Front-end::View::ActivityDiagramEditor
R1O5.2.2.6	Front-end::View::ActivityDiagramEditor
R1O5.2.2.7	Front-end::View::ActivityDiagramEditor
R1O5.2.3	Front-end::View::ActivityDiagramEditor
R1O5.3	Front-end::View::ActivityDiagramEditor
R1O6	Front-end::View::CodeEditor Front-end::ViewModel::Controller Front-end::Model::Commands::Command Front-end::Model::Commands::RequestCode
R106.3	Front-end::View::SinglePageApp Front-end::Model::Commands::Command
R107	Front-end::View::SinglePageApp Front-end::Model::Commands::Command
R1O9	Back-end::ApplicationTier::Generator

	::JavaGenerator::JavaGenerator Back-end::ApplicationTier::Generator::JavaGenerator ::ClassDiaJavaGenerator Back-end::ApplicationTier::Generator::JavaGenerator ::ActivityDiaJavaGenerator::ActivityDiaJavaGenerator Front-end::Model::Commands::Command

Tabella 5: Tracciamento Requisiti-Classi

10.3 Componenti-Requisiti

Componente	Requisiti
Front-end	
Front-end::View	R1O0 R1F1 R1O3 R1O4 R1O4.1 R1O4.3 R1O4.4 R1O4.4.10 R1O4.5 R1O4.6 R1O4.8 R1D4.9 R1O5 R1O5.1 R1O5.2 R1O5.2.1 R1O5.2.1.1 R1O5.2.1.2 R1O5.2.1.3 R1O5.2.1.4 R1O5.2.1.5 R1O5.2.1.6 R1O5.2.1.7 R1O5.3 R1O5.4 R1O6 R1O7 R1O7.1 R1F7.2 R1F8
Front-end::ViewModel	R1O4 R1O5 R1O6
Front-end::Model	R1O3 R1O4 R1D4.2 R1O4.6 R1D4.9 R1O5

	R1O5.2.1 R1O5.2.1.1 R1O5.2.1.2 R1O5.2.1.3 R1O5.2.1.4 R1O5.2.1.5 R1O5.2.1.6 R1O6 R1O9
Front-end::Model::Objects::ClassObjects	R1O4.3 R1O4.4 R1O4.5 R1O4.6 R1O4.9
Front-end::Model::Objects::ActivityObjects	R1O5.1 R1O5.2 R1O5.3
Front-end::Model::Services	R1O5.4
Front-end::Model::Commands	R1D4.2 R1O6 R1O7 R1O7.1
Back-end	
Back-end::PresentationTier::Middleware	R1F3.1.1 R1O3.2.1 R1O4.5.1 R1O6.1 R1O7.1.1 R1F7.2.1
Back-end::ApplicationTier::Generator	R1O6
Back-end::ApplicationTier::Generator::JavaGenerator	R1O9
Back-end::ApplicationTier::Generator::JavaGenerator ::ActivityDiaJavaGenerator	R1O9
Back-end::ApplicationTier::Utility	R1O7.1
Back-end::ApplicationTier::Error	R1F3.1.1 R1O3.2.1 R1O4.5.1 R1O6.1 R1O7.1.1 R1F7.2.1
Back-end::DataTier	R1O5.4

Tabella 6: Tracciamento Componenti-Requisiti

10.4 Requisiti-Componenti

Requisiti	Componenti
R1O0	Front-end::View
R1F1	Front-end::View
R1O3	Front-end::View Front-end::Model
R1F3.1.1	Back-end::PresentationTier::Middleware Back-end::ApplicationTier::Error
R1O3.2.1	Back-end::PresentationTier::Middleware Back-end::ApplicationTier::Error
R1O4	Front-end::View Front-end::Model Front-end::ViewModel
R1O4.1	Front-end::View
R1D4.2	Front-end::Model Front-end::Model::Commands
R1O4.3	Front-end::View Front-end::Model::Objects::ClassObjects
R1O4.4	Front-end::View Front-end::Model::Objects::ClassObjects
R1O4.5	Front-end::View Front-end::Model::Objects::ClassObjects
R1O4.5.1	Back-end::PresentationTier::Middleware Back-end::ApplicationTier::Error
R1O4.6	Front-end::View Front-end::Model::Objects::ClassObjects
R1O4.8	Front-end::View
R1D4.9	Front-end::View Front-end::Model::Objects::ClassObjects
R1O5	Front-end::View Front-end::Model Front-end::ViewModel
R1O5.1	Front-end::View Front-end::Model::Objects::ActivityObjects
R1O5.2	Front-end::View Front-end::Model::Objects::ActivityObjects
R1O5.2.1	Front-end::View Front-end::Model
R1O5.2.1.1	Front-end::View Front-end::Model
R1O5.2.1.2	Front-end::View

	Front-end::Model
R1O5.2.1.3	Front-end::View Front-end::Model
R1O5.2.1.4	Front-end::View Front-end::Model
R1O5.2.1.5	Front-end::View Front-end::Model
R1O5.2.1.6	Front-end::View Front-end::Model
R1O5.2.1.7	Front-end::View Front-end::Model
R1O5.3	Front-end::View Front-end::Model::Objects::ActivityObjects
R1O5.4	Front-end::View Front-end::Model::Services Back-end::DataTier
R1O6	Front-end::View Front-end::Model Front-end::ViewModel Front-end::Model::Commands Back-end::ApplicationTier::Generator
R1O6.1	Back-end::PresentationTier::Middleware Back-end::ApplicationTier::Error
R1O7	Front-end::View Front-end::Model::Commands
R1O7.1	Front-end::View Front-end::Model::Commands Back-end::ApplicationTier::Utility
R1O7.1.1	Back-end::PresentationTier::Middleware Back-end::ApplicationTier::Error
R1F7.2	Front-end::View
R1F7.2.1	Back-end::PresentationTier::Middleware Back-end::ApplicationTier::Error
R1F8	Front-end::View
R1O9	Front-end::Model Back-end::ApplicationTier::Generator::JavaGenerator Back-end::ApplicationTier::Generator::JavaGenerator ::ActivityDiaJavaGenerator

Tabella 7: Tracciamento Requisiti-Componenti

A Descrizione design pattern

A.1 Design pattern architetturali

A.1.1 MVVM

Model-View-ViewModel (MVVM) è un pattern per l'implementazione di interfacce utente. Esso divide un'applicazione software in tre parti interconnesse, in modo da separare nettamente la rappresentazione interna dei dati dal modo in cui essa viene presentata all'utente. Il componente centrale, il modello, consiste di dati business, regole, logica e funzioni. Una View può essere qualsiasi output dell'informazione, come ad esempio un testo o un diagramma. Si possono avere molteplici View della stessa informazione. La View è in grado di gestire eventi, eseguire operazioni ed effettuare il data-binding. La terza parte, ViewModel è un Model esteso con funzionalità per la manipolazione dei dati e per l'interazione con la View. Oltre a dividere l'applicazione in queste tre componenti, MVVM si occupa anche di definire le interazioni tra esse:

- Un ViewModel fa da intermediario tra la vista e il modello, ed è responsabile per la gestione della logica della vista. In genere, il ViewModel interagisce con il modello invocando metodi nelle classi del modello. Il ViewModel fornisce quindi i dati dal modello in una forma che la vista può usare facilmente;
- Un Model è una implementazione del modello di dominio (domain model) della applicazione che include un modello di dati insieme con la logica di business e di validazione.
- Una View è responsabile della definizione della struttura, il layout e l'aspetto di ciò che l'utente vede sullo schermo.

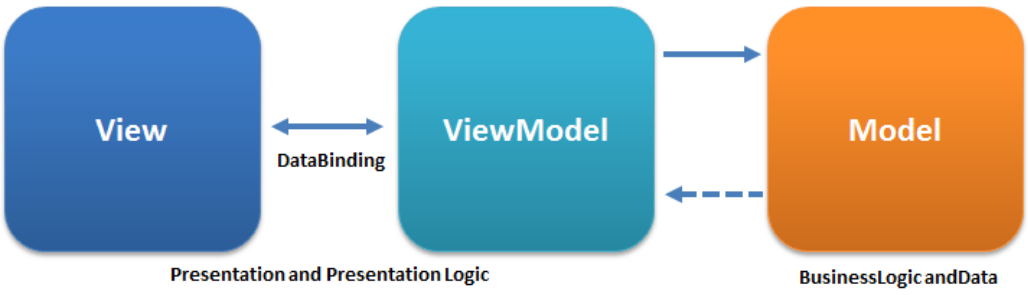

Figura 44: Struttura logica di Model-View-ViewModel

A.1.2 Three-tier

Un'architettura three-tier si fonda su tre strati distinti che compongono il sistema:

- **Presentazion-tier** che si occupa di presentare i dati all'utente e delle interazioni con esso;
- **Logic-tier** che elabora i dati in ingresso dal presentation tier per produrre un risultato;
- **Data-tier** che si occupa della persistenza dei dati.

  Gli strati comunicano solo con quelli adiacenti, garantendo massima separazione tra le componenti.

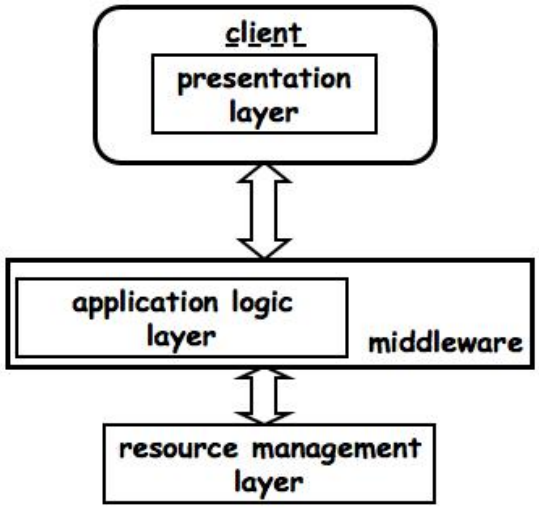

Figura 45: Struttura logica di Three-tier

A.1.3 Middleware

Il Middleware è uno strato software che si interpone tra l'applicazione software e il sistema operativo per semplificarne le comunicazioni e la gestione di input/output. Viene solitamente utilizzato in applicazioni distribuite e facilita l'interoperabilità, fornendo servizi che permettono la comunicazione tra applicazioni di sistemi operativi diversi. La distinzione tra lo strato software del sistema operativo è, per alcune entità, arbitraria; può infatti accadere che il Middleware fornisca dei servizi abitualmente attribuibili a un sistema operativo. I primi utilizzi di Middleware risalgono agli anni '80, come soluzione

ai problemi di comunicazione tra applicazioni nuove e meno recenti. I servizi Middleware forniscono un set di interfacce che permette a un'applicazione di:

- localizzare facilmente applicazioni o servizi in una rete;
- filtrare dati per renderli *user friendly*$_G$ oppure anonimizzarli per renderli pubblicabili, proteggendone la *privacy*;
- essere indipendente dai servizi di rete;
- essere affidabile e sempre disponibile.

Si tratta quindi di funzionalità leggermente più specializzate da quelle normalmente offerte da un sistema operativo. L'avvento del web ha avuto una forte ripercussione sulla diffusione dei software di Middleware. Essi hanno infatti permesso l'accesso sicuro da remoto a database locali. I tipi di Middleware sono:

- **Message-Oriented Middleware (MOM)**: sono Middleware dove le notifiche degli eventi vengono spedite come messaggi tra sistemi o componenti. I messaggi inviati al client vengono memorizzati fintanto che non vengono gestiti, nel frattempo il client può svolgere altro lavoro;
- **Enterprise messaging system**: è un tipo di Middleware che facilita il passaggio di messaggi tra sistemi diversi o componenti in formato standard, spesso utilizzando servizi web o *XML*$_G$;
- **Message broker**: è parte dell entreprise messaging system. Accoda, duplica, traduce e spedisce messaggi a sistemi o componenti diverse;
- **Enterprise Service Bus**: è definito come qualche tipo di Middleware integrato che supporta sia MOM che dei servizi web;
- **Intelligent Middleware**: gestisce il processamento in tempo reale di grandi volumi di segnali che trasforma in informazioni di business. Particolarmente adatto per architetture scalabili e distribuite;
- **Content-Centric Middleware**: questo tipo di Middleware fornisce una semplice astrazione con la quale le applicazioni possono inoltrare richieste per contenuti univocamente identificati, senza occuparsi su come e dove vanno ottenuti.

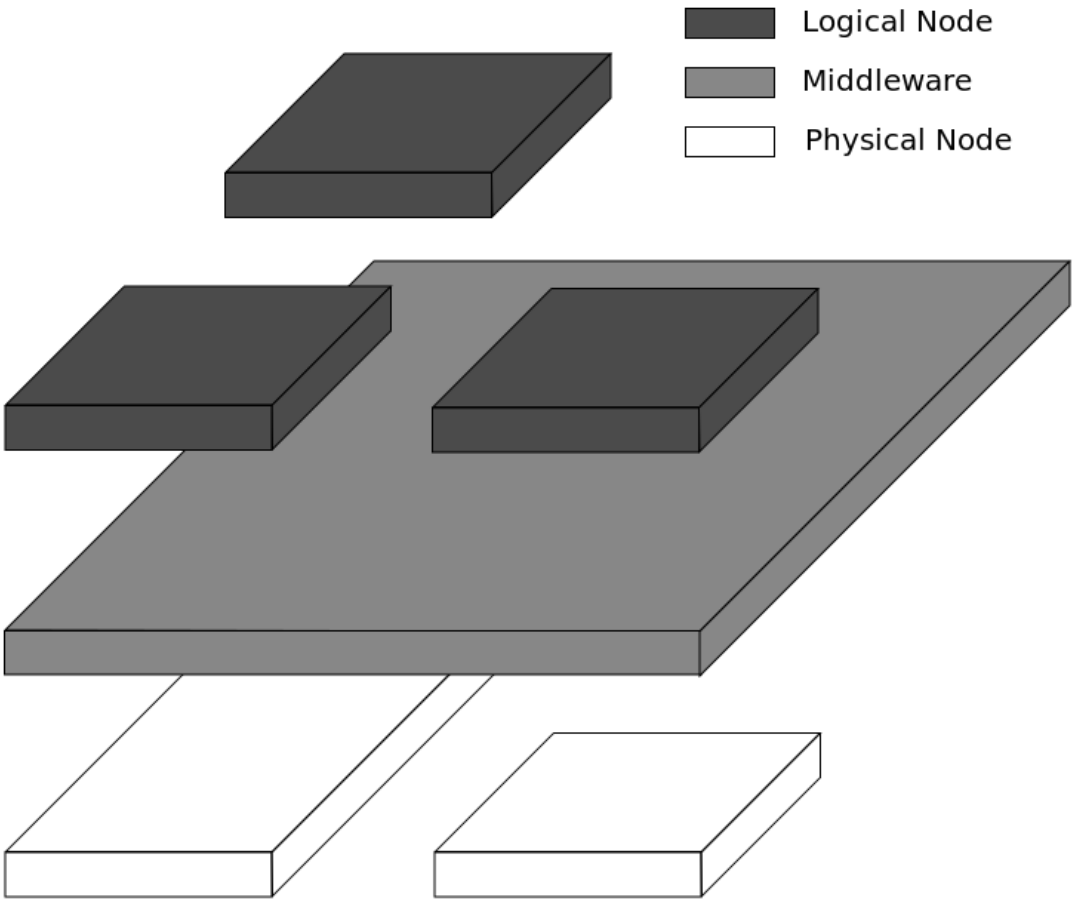

Figura 46: Struttura logica di Middleware

A.2 Design pattern creazionali

A.2.1 Abstract factory

L'Abstract Factory fornisce un'interfaccia per creare famiglie di oggetti connessi o dipendenti tra loro, in modo che non ci sia necessità da parte dei client di specificare i nomi delle classi concrete all'interno del proprio codice.

In questo modo si permette che un sistema sia indipendente dall'implementazione degli oggetti concreti e che il client, attraverso l'interfaccia, utilizzi diverse famiglie di prodotti.

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;
- si vuole che i prodotti che sono organizzati in famiglie siano vincolati ad essere utilizzati con prodotti della stessa famiglia;
- si vuole fornire una libreria di classi mostrando solo le interfacce e nascondendo le implementazioni.

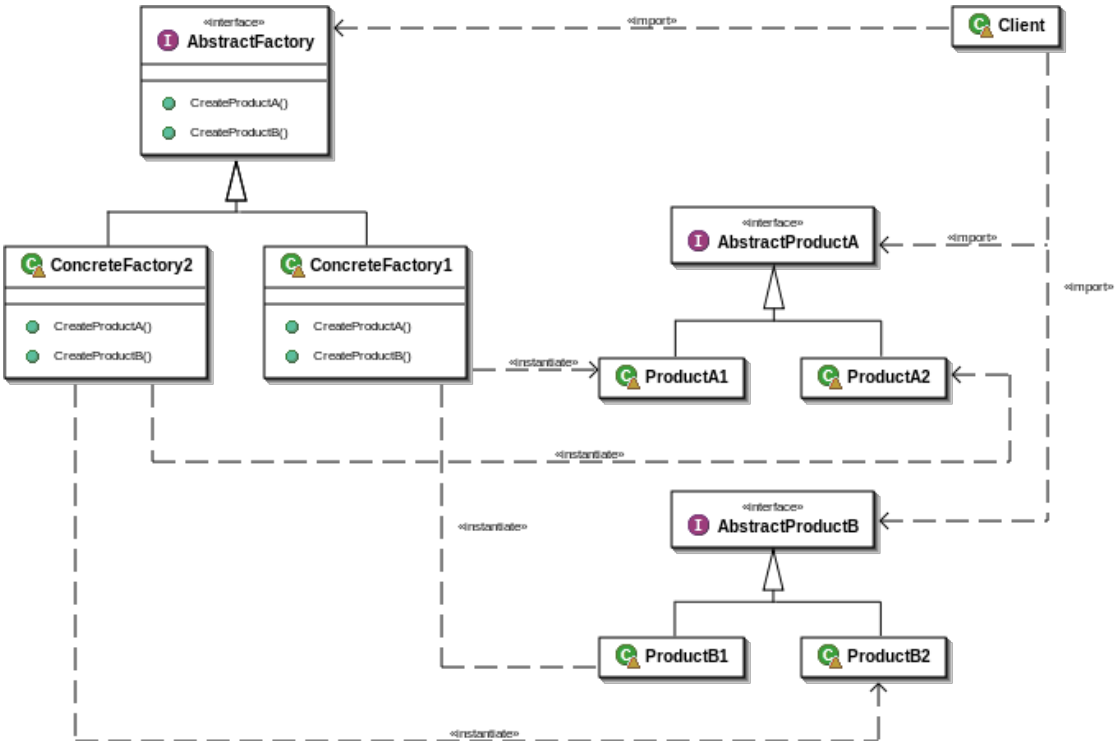

Figura 47: Logica di Abstract factory

Dall'immagine si evidenzia:

- **AbstractFactory** dichiara l'interfaccia per le operazioni che creano prodotti astratti;
- **ConcreteFactory** implementa le operazioni per creare i prodotti concreti;
- **AbstractProduct** dichiara l'interfaccia per un tipo di prodotto;
- **ConcreteProduct** implementa *AbstractProduct* e definisce l'oggetto prodotto che deve essere creato dalla factory concreta corrispondente;

- **Client** utilizza solo le interfacce dichiarate da *AbstractFactory* e *AbstractProduct*.

A.3 Design pattern comportamentali

A.3.1 Chain of responsibility

Il chain of responsibility è un pattern comportamentale che permette di separare i sender dai receiver delle richieste. La richiesta attraversa una catena di oggetti per essere intercettata solo quando raggiunge il proprio gestore. Viene utilizzato quando non è possibile determinare staticamente il receiver oppure l'insieme di oggetti gestori cambia dinamicamente a runtime. Le richieste vengono dette implicite poiché il sender non ha alcuna conoscenza sull'identità del ricevente. Per permettere alla richiesta di attraversare la catena e per rimanere implicita, ogni receiver condivide un'interfaccia comune per gestire le richieste ed accedere al proprio successore. La gerarchia che vorrà inviare richieste dovrà avere una superclasse che dichiara un metodo handler generico.

L'utilizzo di questo pattern comporta una serie di conseguenze:

- Ridotto accoppiamento: gli oggetti non sono a conoscenza di chi gestirà la richiesta ma sanno solo che verrà gestita in modo appropriato. Inoltre non bisognerà manutenere i riferimenti a tutti i possibili riceventi;
- Aggiunge flessibilità nell'assegnamento delle responsabilità degli oggetti: è possibile distribuire le responsabilità tra gli oggetti a runtime modificandone la gerarchia. Staticamente è possibile usare il subclassing per specializzare i gestori;
- Non c'è garanzia che la request venga gestita, questo può avvenire quando la catena non è stata costruita in modo rigoroso.

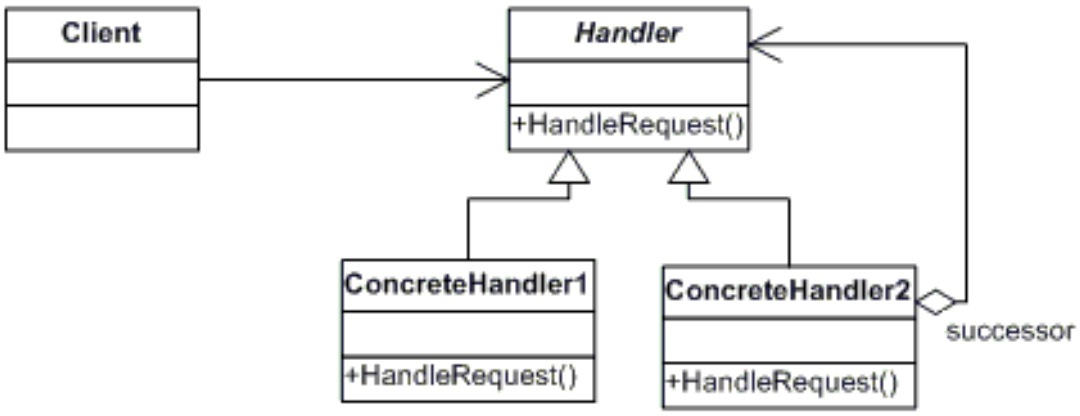

Figura 48: Struttura del Chain of responsibility

A.3.2 Strategy

Strategy ha come scopo quello di definire una famiglia di algoritmi, incapsularli e renderli intercambiabili. Permette agli algoritmi di variare indipendentemente dal client che ne fa uso.
È opportuno usare il pattern strategy nei seguenti casi:

- Molte classi correlate differiscono fra loro solo per il comportamento. Strategy fornisce un modo per configurare una classe con un comportamento scelto fra tanti;
- Sono necessarie più varianti di un algoritmo. Per esempio, è possibile definire più algoritmi con bilanciamenti diversi fra occupazione in memoria, velocità di esecuzione, ecc. Possiamo usare il pattern Strategy quando queste varianti sono implementate sotto forma di gerarchia di classi di algoritmi;
- Un algoritmo usa una struttura dati che non dovrebbe essere resa nota ai client. Il pattern strategy può essere usato per evitare di esporre strutture dati complesse e specifiche dell'algoritmo;
- Una classe definisce molti comportamenti che compaiono all'interno di scelte condizionali multiple. Al posto di molte scelte condizionali si suggerisce di spostare i blocchi di codice correlati in una classe Strategy dedicata.

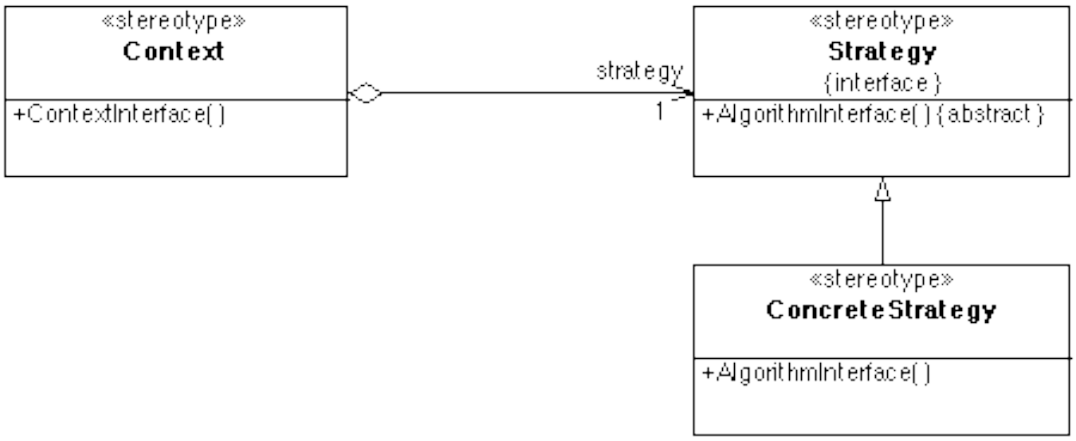

Figura 49: Struttura logica di Strategy

A.3.3 Command

Il command pattern è uno dei design pattern che permette di isolare la porzione di codice che effettua un'azione (eventualmente molto complessa) dal codice che ne richiede l'ese-

cuzione. L'azione è incapsulata nell'oggetto Command. L'obiettivo è rendere variabile l'azione del client senza però conoscere i dettagli dell'operazione stessa. Altro aspetto importante è che il destinatario della richiesta può non essere deciso staticamente all'atto dell'istanziazione del Command ma dev'essere ricavato 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.

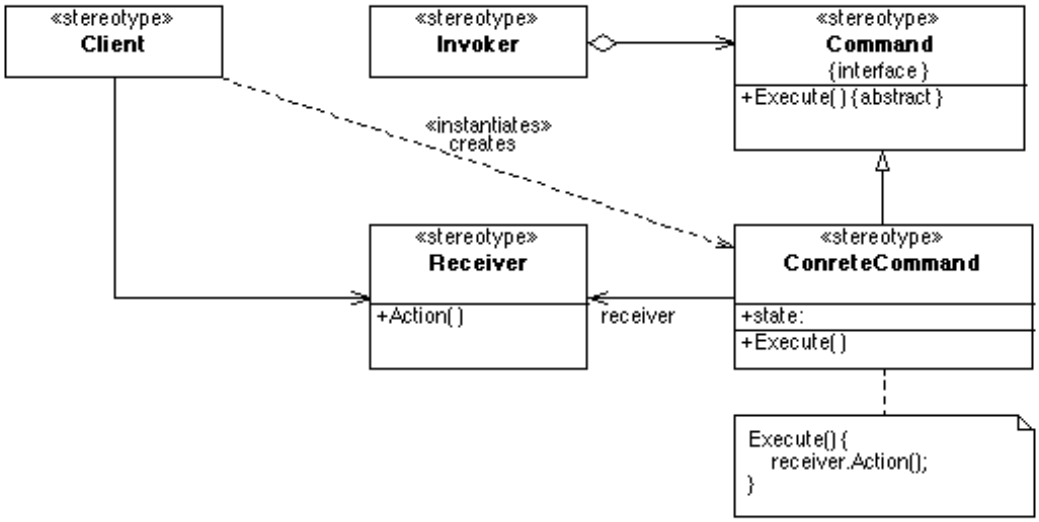

Figura 50: Struttura logica di Command