



## Definizione di prodotto

### Informazioni sul documento

<b>Versione</b>	1.0.0
<b>Data di Creazione</b>	2017-03-13
<b>Data ultima modifica</b>	2017-04-07
<b>Stato</b>	Approvato
<b>Redazione</b>	Jordan Gottardo Giovanni Prete
<b>Verifica</b>	Leonardo Brutesco
<b>Approvazione</b>	Daniel De Gaspari
<b>Uso</b>	Esterno
<b>Lista di distribuzione</b>	Professor Tullio Vardanega Professor Riccardo Cardin <i>Zephyrus</i> <i>RiskApp</i>
<b>Email di riferimento</b>	zephyrus.swe@gmail.com

## Registro delle modifiche

Versione	Data	Autore	Ruolo	Descrizione
1.0.0	2017-04-08	Daniel De Gaspari	<i>Responsabile</i>	Approvazione del documento
0.3.0	2017-04-08	Leonardo Brutesco	<i>Verificatore</i>	Verifica del documento
0.2.2	2017-04-07	Jordan Gottardo	<i>Progettista</i>	Stesura sezione 5
0.3.2	2017-03-27	Leonardo Brutesco	<i>Verificatore</i>	Verifica del documento in forma parziale
0.3.1	2017-03-26	Giovanni Prete	<i>Progettista</i>	Stesura appendice A
0.3.0	2017-03-24	Leonardo Brutesco	<i>Verificatore</i>	Verifica del documento in forma parziale
0.2.2	2017-03-23	Jordan Gottardo	<i>Progettista</i>	Stesura nella sezione 4 delle sottosezioni relative agli asset
0.2.1	2017-03-23	Giovanni Prete	<i>Progettista</i>	Corretti gli errori grammaticali e sintattici segnalati dal verificatore nelle sezione 4 in "ViewPkg:: Sidebar-Pkg:: ContentPkg:: InsertNodeContent"
0.2.0	2017-03-22	Leonardo Brutesco	<i>Verificatore</i>	Verifica del documento in forma parziale
0.1.2	2017-03-21	Jordan Gottardo	<i>Progettista</i>	Stesura nella sezione 4 delle sottosezioni relative agli asset
0.1.1	2017-03-16	Giovanni Prete	<i>Progettista</i>	Corretti gli errori grammaticali e sintattici segnalati dal verificatore nelle sezione 4 in "StorePkg::ProcessPkg::Asset"
0.1.0	2017-03-15	Leonardo Brutesco	<i>Verificatore</i>	Verifica del documento in forma parziale
0.0.4	2017-03-14	Jordan Gottardo	<i>Progettista</i>	Stesura nella sezione 4 delle sottosezioni relative alle interazioni con la mappa
0.0.3	2017-03-21	Giovanni Prete	<i>Progettista</i>	Modificato "Scopo del documento" in sezione 1
0.0.2	2017-03-21	Giovanni Prete	<i>Progettista</i>	Stesura sezioni 1, 2 e 3
0.0.1	2017-03-21	Giovanni Prete	<i>Progettista</i>	Creazione template e indice

## Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Scopo del documento . . . . .	1
1.2	Scopo del prodotto . . . . .	1
1.3	Glossario . . . . .	1
1.4	Riferimenti . . . . .	1
1.4.1	Riferimenti normativi . . . . .	1
1.4.2	Riferimenti informativi . . . . .	1
<b>2</b>	<b>Tecnologie utilizzate</b>	<b>3</b>
2.1	Introduzione . . . . .	3
2.2	CSS3 . . . . .	4
2.3	HTML5 . . . . .	4
2.4	JavaScript ES6 . . . . .	5
2.5	JSON . . . . .	5
2.6	JSX . . . . .	6
2.7	Node.js . . . . .	6
2.8	OpenLayers . . . . .	7
2.9	Open Street Map . . . . .	7
2.10	React . . . . .	8
2.11	ReactColor . . . . .	9
2.12	React-Redux . . . . .	9
2.13	React Toolbox . . . . .	9
2.14	Redux . . . . .	10
2.15	SVG . . . . .	10
<b>3</b>	<b>Descrizione architettura</b>	<b>11</b>
3.1	Introduzione . . . . .	11
3.2	Server . . . . .	11
3.2.1	Chiamate REST . . . . .	11
3.2.1.1	Customer . . . . .	11
3.2.1.2	Graph . . . . .	12
3.2.1.3	Asset . . . . .	12
3.2.1.4	Node . . . . .	12
3.2.1.5	Edge . . . . .	12
3.3	Client . . . . .	13
3.3.1	Design architettonicale di DeGeOP . . . . .	13
<b>4</b>	<b>Componenti</b>	<b>14</b>
4.1	Introduzione . . . . .	14
4.2	DeGeOP . . . . .	14
4.2.1	Informazioni sul package . . . . .	14
4.3	DeGeOP:StorePkg . . . . .	16
4.3.1	Informazioni sul package . . . . .	16
4.4	DeGeOP:StorePkg::StoreContentsPkg . . . . .	18
4.4.1	Informazioni sul package . . . . .	18
4.4.2	Classi . . . . .	18
4.4.2.1	Customer . . . . .	18
4.4.2.2	Options . . . . .	19

4.4.2.3	StoreDeGeOP . . . . .	20
4.5	DeGeOP::StorePkg::ProcessPkg . . . . .	21
4.5.1	Informazioni sul package . . . . .	21
4.5.2	Classi . . . . .	21
4.5.2.1	Asset . . . . .	21
4.5.2.2	Edge . . . . .	24
4.5.2.3	ExitNode . . . . .	25
4.5.2.4	MachineNode . . . . .	25
4.5.2.5	Node . . . . .	27
4.5.2.6	Process . . . . .	28
4.5.2.7	QueueNode . . . . .	29
4.5.2.8	ResourceNode . . . . .	30
4.5.2.9	SourceNode . . . . .	30
4.6	DeGeOP::StorePkg::PolygonPkg . . . . .	32
4.6.1	Informazioni sul package . . . . .	32
4.6.2	Classi . . . . .	32
4.6.2.1	ConcretePolygon . . . . .	32
4.6.2.2	ConcretePolygonFactory . . . . .	33
4.6.2.3	Coordinate . . . . .	33
4.6.2.4	Polygon . . . . .	34
4.6.2.5	PolygonFactory . . . . .	34
4.7	DeGeOP::ReducerPkg . . . . .	35
4.7.1	Informazioni sul package . . . . .	35
4.7.2	Classi . . . . .	35
4.7.2.1	AssetReducer . . . . .	35
4.7.2.2	EdgeReducer . . . . .	36
4.7.2.3	NodeReducer . . . . .	36
4.7.2.4	OptionReducer . . . . .	37
4.7.2.5	Reducer . . . . .	37
4.8	DeGeOP::CallManagerPkg . . . . .	39
4.8.1	Informazioni sul package . . . . .	39
4.8.2	Classi . . . . .	39
4.8.2.1	Request . . . . .	39
4.8.2.2	Server . . . . .	40
4.9	DeGeOP::ActionPkg . . . . .	42
4.9.1	Informazioni sul package . . . . .	42
4.9.2	Classi . . . . .	43
4.9.2.1	EdgeAction . . . . .	43
4.10	DeGeOP::ActionPkg::ActionElementsPkg . . . . .	44
4.10.1	Informazioni sul package . . . . .	44
4.10.2	Classi . . . . .	44
4.10.2.1	Action . . . . .	44
4.10.2.2	AssetAction . . . . .	45
4.10.2.3	NodeAction . . . . .	45
4.10.2.4	OptionAction . . . . .	45
4.11	DeGeOP::ActionPkg::ActionCreatorsPkg . . . . .	47
4.11.1	Informazioni sul package . . . . .	47
4.11.2	Classi . . . . .	47
4.11.2.1	AssetActionCreator . . . . .	47

4.11.2.2	EdgeActionCreator . . . . .	48
4.11.2.3	NodeActionCreator . . . . .	48
4.11.2.4	OptionActionCreator . . . . .	49
4.12	DeGeOP::ViewPkg . . . . .	50
4.12.1	Informazioni sul package . . . . .	50
4.13	DeGeOP::ViewPkg::DeGeOPViewPkg . . . . .	51
4.13.1	Informazioni sul package . . . . .	51
4.13.2	Classi . . . . .	52
4.13.2.1	DeGeOPView . . . . .	52
4.14	DeGeOP::ViewPkg::MapComponentsPkg . . . . .	55
4.14.1	Informazioni sul package . . . . .	55
4.14.2	Classi . . . . .	55
4.14.2.1	ButtonWrapper . . . . .	55
4.14.2.2	MapWrapper . . . . .	56
4.14.2.3	MessageWrapper . . . . .	57
4.14.2.4	PolygonOperationWrapper . . . . .	58
4.15	DeGeOP::ViewPkg::SidebarPkg . . . . .	59
4.15.1	Informazioni sul package . . . . .	59
4.15.2	Classi . . . . .	60
4.15.2.1	HomeSidebar . . . . .	60
4.15.2.2	InsertAssetSidebar . . . . .	60
4.15.2.3	InsertEdgeSidebar . . . . .	61
4.15.2.4	InsertNodeSidebar . . . . .	61
4.15.2.5	ViewAssetSidebar . . . . .	62
4.15.2.6	ViewEdgeSidebar . . . . .	62
4.15.2.7	ViewNodeSidebar . . . . .	63
4.16	DeGeOP::ViewPkg::SidebarPkg::ContentPkg . . . . .	64
4.16.1	Informazioni sul package . . . . .	64
4.16.2	Classi . . . . .	65
4.16.2.1	AbstractContent . . . . .	65
4.16.2.2	InsertAssetContent . . . . .	65
4.16.2.3	InsertEdgeContent . . . . .	66
4.16.2.4	InsertNodeContent . . . . .	66
4.16.2.5	ViewAssetContent . . . . .	67
4.16.2.6	ViewEdgeContent . . . . .	67
4.16.2.7	ViewNodeContent . . . . .	68
4.17	DeGeOP::ViewPkg::SidebarPkg::ButtonsPkg . . . . .	69
4.17.1	Informazioni sul package . . . . .	70
4.17.2	Classi . . . . .	70
4.17.2.1	AbstractButtons . . . . .	70
4.17.2.2	ThreeButtons . . . . .	70
4.17.2.3	TwoButtons . . . . .	71
<b>5</b>	<b>Tracciamento</b>	<b>72</b>
5.1	Tracciamento classi-requisiti . . . . .	72
5.2	Tracciamento requisiti-classi . . . . .	83
<b>A</b>	<b>Descrizione design pattern</b>	<b>98</b>
A.1	Introduzione . . . . .	98
A.2	Pattern Creazionali . . . . .	99

A.2.1	Factory Method . . . . .	99
A.2.1.1	Descrizione . . . . .	99
A.2.1.2	Contestualizzazione . . . . .	99
A.3	Pattern Architetturali . . . . .	100
A.3.1	Redux . . . . .	100
A.3.1.1	Descrizione . . . . .	100
A.3.1.2	Contestualizzazione . . . . .	100

## 1 Introduzione

### 1.1 Scopo del documento

Lo scopo del documento è definire la progettazione sia ad alto livello e sia di dettaglio del prodotto DeGeOP.

Si precisa che viene riportato solo ed esclusivamente quanto verrà sviluppato effettivamente (tecnologie, componenti, classi, metodi, campi dati, tracciamento, design pattern).

### 1.2 Scopo del prodotto

Lo scopo del prodotto consiste nella creazione di un'interfaccia web contenente una mappa geografica su cui potranno essere rappresentati:

- il processo produttivo aziendale;
- gli scenari di danno;
- i risultati dell'analisi dei rischi.

Il prodotto verrà utilizzato da agenti assicuratori per l'inserimento delle informazioni utili allo svolgimento dell'analisi dei rischi dell'assicurando.

L'interfaccia dovrà essere in grado di connettersi ai sistemi preesistenti di *RiskApp* per la memorizzazione e gestione dei dati inseriti. A causa del requisito di integrabilità, che è stato deciso di soddisfare, l'applicazione da sviluppare sarà parte integrante dell'attuale applicazione del proponente.

### 1.3 Glossario

Allo scopo di rendere più semplice e chiara la comprensione dei documenti viene allegato il *Glossario v2.0.0*, nel quale verranno raccolte le spiegazioni di terminologia tecnica o ambigua, abbreviazioni ed acronimi. Per evidenziare un termine presente in tale documento, esso verrà marcato con il pedice *G*. Solo la prima occorrenza del termine in ogni sezione sarà marcata per non appesantire la lettura del documento.

Tutti i termini del glossario evidenziati sono link ipertestuali al glossario stesso; affinché funzionino correttamente è necessario che la posizione delle directory e dei file forniti non venga alterata.

### 1.4 Riferimenti

#### 1.4.1 Riferimenti normativi

- *Norme di progetto v3.0.0*.

#### 1.4.2 Riferimenti informativi

- **capitolatoG d'appalto C3: DeGeOP: A Designer and Geo-localizer Web App for Organizational Plants.** Reperibile all'indirizzo:  
<http://www.math.unipd.it/~tullio/IS-1/2016/Progetto/C3.pdf> ;  
• *Analisi dei requisiti v3.0.0*;

- **Guide to the Software Engineering Body of Knowledge: IEEE Computer Society. Software Engineering Coordinating Committee (Versione 2004):**
  - **Chapter 3:** Software Design;
- **slide del corso di Ingegneria del Software**  
<http://www.math.unipd.it/~tullio/IS-1/2016/>;
- **Design Patterns** - Elementi per il riuso di software a oggetti - Gamma, Helm, Johnson, Vlissides;
- **Documentazione di Redux.** Reperibile all'indirizzo:  
<http://redux.js.org/>;
- **Documentazione di React.** Reperibile all'indirizzo:  
<https://facebook.github.io/react/>;
- **Documentazione di REST.** Reperibile all'indirizzo:  
[https://en.wikipedia.org/wiki/Representational\\_state\\_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer).

## 2 Tecnologie utilizzate

### 2.1 Introduzione

In questa sezione vengono descritte le tecnologie su cui si basa lo sviluppo del progetto. Per ognuna di esse verrà indicato l'ambito di utilizzo della tecnologia, i vantaggi e eventuali svantaggi che ne derivano. La scelta delle tecnologie non è stata vincolata in alcun modo dal proponente, anche se alcune decisioni a riguardo sono state prese tenendo conto delle tecnologie da loro già utilizzate.

Tecnologia	Utilizzo
CSS3	Linguaggio per la formattazione delle pagine web
HTML5	Linguaggio per la costruzione di pagine web
JavaScript ES6	Linguaggio principale in cui è sviluppata l'applicazione
JSON	Formato dati utilizzato per lo scambio di informazioni
JSX	Estensione <i>JavaScript_G</i> per l'integrazione del codice <i>HTML_G</i>
Node.js	Ambiente operativo per utilizzare JavaScript lato server
OpenLayers	Libreria per la gestione della mappa e del grafo
Open Street Map	Libreria per la fornitura di mappe in formato vettoriale
React	Costruzione dell'interfaccia grafica
ReactColor	Libreria per gestire la paletta di colori
React-Redux	Libreria per interfacciare <i>React_G</i> con Redux
React Toolbox	Libreria che implementa la specifica di Material Design
Redux	Libreria per l'implementazione dell'architettura
SVG	Scalable Vector Graphics

Tabella 1: Panoramica generale delle tecnologie usate nel progetto

## 2.2 CSS3

<b>Descrizione</b>	È un linguaggio utilizzato per la presentazione di documenti HTML. Lo standard viene definito dal <a href="#">W3C</a> .
<b>Vantaggi</b>	- assicura maggiore manutenibilità e riutilizzo grazie alla separazione tra presentazione e struttura; - raccomandato dal W3C.
<b>Svantaggi</b>	- specifica ufficiale non completata; - non supportato pienamente da tutti i browser.
<b>Utilizzo</b>	Viene utilizzato per definire il layout dell'applicazione.

## 2.3 HTML5

<b>Descrizione</b>	È un <a href="#">linguaggio di markup</a> utilizzato per definire la struttura delle pagine web. Lo standard viene definito dal W3C.
<b>Vantaggi</b>	- fornisce un set di <a href="#">tag</a> più vasto rispetto alle vecchie versioni, con nuovi tag semanticci; - raccomandato dal W3C.
<b>Svantaggi</b>	- non supportato pienamente da tutti i browser.
<b>Utilizzo</b>	Viene utilizzato per definire il layout dell'applicazione.

## 2.4 JavaScript ES6

<b>Descrizione</b>	JavaScript è un linguaggio di scripting orientato agli oggetti e agli eventi, utilizzato principalmente nella programmazione Web lato client. Le caratteristiche più importanti di questo linguaggio sono:
	<ul style="list-style-type: none"> <li>• <b>eventi:</b> quando l'utente interagisce con la pagina Web in vari modi, come ad esempio mouse e tastiera, viene generato un evento; JavaScript gestisce tali eventi, i quali possono avviare un'azione registrata in un gestore di eventi;</li> <li>• <b>tipizzazione dinamica:</b> il programmatore non è tenuto a specificare il tipo degli oggetto che utilizza;</li> <li>• <b>paradigma a protipi:</b> stile di programmazione orientato ad oggetti in cui l'ereditarietà è implementata tramite il riuso di oggetti esistenti, basandosi sul loro prototipo.</li> </ul>
	In particolare, il <i>gruppoG</i> si baserà sull'utilizzo della specifica JavaScript ES6, che definisce significativi cambiamenti sintattici per la scrittura di applicazioni complesse in modo più semplice.
<b>Vantaggi</b>	<ul style="list-style-type: none"> <li>- facilità di utilizzo;</li> <li>- larga disponibilità di documentazione;</li> <li>- conoscenza pregressa del linguaggio;</li> <li>- maggior supporto da parte dei browser rispetto alle alternative, come ad esempio ActionScript.</li> </ul>
<b>Svantaggi</b>	<ul style="list-style-type: none"> <li>- variazione di interpretazione a seconda del browser;</li> <li>- la tipizzazione dinamica è frequentemente fonte di errori.</li> </ul>
<b>Utilizzo</b>	JavaScript è il linguaggio base con cui si svilupperà l'applicazione <i>DeGeOP</i> . Di conseguenza è anche il linguaggio utilizzato maggiormente dalle librerie esterne da noi sfruttate.

## 2.5 JSON

<b>Descrizione</b>	Formato dati utilizzato per lo scambio di informazioni tra il client (ovvero il nostro prodotto) e il server (ovvero il prodotto di <i>RiskApp</i> ).
<b>Vantaggi</b>	<ul style="list-style-type: none"> <li>- standard per lo scambio di dati;</li> <li>- meno verboso di alternative come XML.</li> </ul>
<b>Utilizzo</b>	Viene utilizzato per lo scambio di dati tra l'applicazione <i>DeGeOPe</i> e il server di <i>RiskApp</i> .

## 2.6 JSX

<b>Descrizione</b>	JSX è un linguaggio orientato agli oggetti staticamente tipizzato. È un'estensione di JavaScript. I file in linguaggio JSX vengono poi tradotti in JavaScript.
<b>Vantaggi</b>	<ul style="list-style-type: none"><li>- permette di utilizzare tag in stile HTML all'interno delle componenti React;</li><li>- facilità di utilizzo;</li><li>- viene compilato, quindi permette di scoprire gli errori a tempo di compilazione;</li><li>- il suo utilizzo in combinazione con React è altamente consigliato;</li><li>- maggiore leggibilità.</li></ul>
<b>Utilizzo</b>	Viene utilizzato come sintassi all'interno di React.

## 2.7 Node.js

<b>Descrizione</b>	Ambiente operativo per utilizzare JavaScript in ambito server.
<b>Vantaggi</b>	<ul style="list-style-type: none"><li>- combinato con npm permette di creare un ambiente di sviluppo molto facilitato.</li></ul>
<b>Utilizzo</b>	Viene utilizzato per far avviare la nostra applicazione.

## 2.8 OpenLayers

<b>Descrizione</b>	E' una libreria JavaScript per visualizzare mappe interattive nei browser web. OpenLayers offre <a href="#">API</a> ai programmati per poter accedere a diverse fonti d'informazioni cartografiche in Internet: mappe del progetto OpenStreetMap, mappe sotto licenze non-libere (Google Maps, Bing, Yahoo), Web Feature Service, ecc. E' coperto da licenza BSD.
<b>Vantaggi</b>	- buona documentazione; - maggiori funzionalità e flessibilità rispetto ai concorrenti, come ad esempio Leaflet.
<b>Svantaggi</b>	- più pesante di alcuni concorrenti, come Leaflet.
<b>Utilizzo</b>	Viene utilizzato per gestire la mappa.

## 2.9 Open Street Map

<b>Descrizione</b>	E' una libreria JavaScript per fornire informazioni geografiche in formato vettoriale.
<b>Vantaggi</b>	- utile alla nostra applicazione in quanto mostra il perimetro di molti edifici.
<b>Svantaggi</b>	- mancanza della vista satellitare.
<b>Utilizzo</b>	OpenStreetMap viene utilizzato in modo indiretto dalla libreria OpenLayers per fornire una vista vettoriale nella mappa dell'applicazione.

## 2.10 React

<b>Descrizione</b>	E' una libreria JavaScript <i>open source</i> mantenuta da Facebook e Instagram utile alla costruzione di interfacce grafiche. Per fare ciò, React utilizza componenti indipendenti e riusabili che ereditano dalla classe base astratta React.Component. Le componenti devono implementare il metodo <i>render()</i> che si occupa di rappresentare la <i>componente</i> sul browser. Le caratteristiche più importanti di questa libreria sono:
	<ul style="list-style-type: none"><li>• <b>One-way-data-flow:</b> meccanismo tramite il quale le proprietà (un insieme di valori immutabili passato al render di un componente) non possono essere direttamente modificate. Queste proprietà possono però essere modificate da una <i>callback</i>;</li><li>• <b>Virtual DOM:</b> virtualizzazione operata da React per effettuare un re-rendering efficiente dei componenti. Consiste in:<ul style="list-style-type: none"><li>- replicare il DOM in memoria;</li><li>- individuare le differenze tra il DOM reale e il DOM virtuale;</li><li>- aggiornare le informazioni del DOM reale sulla base delle differenze precedentemente individuate.</li></ul></li><li>• utilizzo di JSX.</li></ul>
<b>Vantaggi</b>	<ul style="list-style-type: none"><li>- facile da testare in quanto il DOM virtuale è implementato interamente in JavaScript;</li><li>- agevola il riuso del codice grazie all'uso delle componenti, le quali possono essere combinate e collegate tra loro;</li><li>- gestione automatica degli aggiornamenti dell'interfaccia grafica.</li></ul>
<b>Svantaggi</b>	<ul style="list-style-type: none"><li>- per far dialogare le componenti, React suggerisce di portare lo stato nelle componenti di più alto livello possibile. In questo modo le componenti di più basso livello sono prevalentemente presentazionali, mentre le componenti di più alto livello risultano essere cariche di informazioni;</li><li>- manca di librerie per la gestione del model perché si occupa solamente della costruzione dell'interfaccia grafica.</li></ul>
<b>Utilizzo</b>	React viene utilizzata per la costruzione dell'interfaccia grafica dell'applicazione.

## 2.11 ReactColor

<b>Descrizione</b>	E' una libreria JavaScript per creare una paletta di colori RGB.
<b>Vantaggi</b>	- buona documentazione.
<b>Utilizzo</b>	Viene utilizzata per la creazione di un color picker.

## 2.12 React-Redux

<b>Descrizione</b>	Libreria che facilita l'integrazione tra Redux e React.
<b>Vantaggi</b>	- facilita l'integrazione tra Redux e React.
<b>Utilizzo</b>	Le classi JavaScript vengono passate ad una funzione della libreria per ottenere una nuova classe che sfrutta React-Redux.

## 2.13 React Toolbox

<b>Descrizione</b>	E' una libreria JavaScript composta da un insieme di componenti React che implementano la specifica del Material Design di Google.
<b>Vantaggi</b>	- vasta varietà di elementi grafici; - elementi grafici e temi facilmente personalizzabili; - ben documentata; - aiuta la separazione tra presentazione e contenuto grazie all'utilizzo usa i moduli <a href="#">CSS_G</a> al posto dello stile inline, a differenza ad esempio di Material-UI.
<b>Utilizzo</b>	React Toolbox viene utilizzata per implementare alcune le componenti grafiche secondo la specifica del Material Design.

## 2.14 Redux

<b>Descrizione</b>	Libreria per l'implementazione dell'architettura che si occupa di gestire le interazioni tra la business logic e la presentazione. Per fare ciò:
	<ul style="list-style-type: none"><li>• implementa un <i>design pattern</i> architettonicale da usare il sostituzione a MVC, come descritto in <a href="#">Redux</a>;</li><li>• offre delle API apposite per la gestione degli elementi del design pattern descritto al punto precedente.</li></ul>
<b>Vantaggi</b>	- integrabile facilmente con React; - largamente utilizzato; - ben documentata.
<b>Utilizzo</b>	Redux viene utilizzato per implementare l'architettura di <i>DeGeOP</i> .

## 2.15 SVG

<b>Descrizione</b>	Standard per la scrittura di immagini in formato vettoriale.
<b>Vantaggi</b>	- standard aperto.
<b>Utilizzo</b>	SVG viene utilizzato per aggiungere oggetti personalizzati alla mappa.

## 3 Descrizione architettura

### 3.1 Introduzione

Il prodotto *DeGeOP*, creato dal *gruppo<sub>G</sub>* *Zephyrus*, sarà integrabile nell'attuale applicazione del proponente *RiskApp*.

Per esporre l'architettura dell'applicazione si procederà con approccio top-down, partendo cioè da una visione generale delle componenti che distinguono il sistema, per poi analizzare in dettaglio la conformazione di tali componenti.

Il sistema attuale di *RiskApp* presenta un'architettura client-server.

### 3.2 Server

*DeGeOP* si conserverà al server di *RiskApp* esclusivamente utilizzando le *API<sub>G</sub>* REST fornite dal proponente stesso e riportate in questa sezione. Il gruppo non ha quindi accesso all'implementazione del server di *RiskApp*.

#### 3.2.1 Chiamate REST

L'interfaccia REST proposta da *RiskApp* fornisce l'accesso alle seguenti entità:

- Customer
- Graph
- *Asset<sub>G</sub>*
- Node
- Edge

Per ognuna di queste è possibile fare una chiamata REST usando i verbi http. (GET, OPTION, POST, PUT, UPDATE, DELETE)

Tutte queste entità sono identificate univocamente da uno uuid formato come una stringa di cifre esadecimale con la seguente struttura:

XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX

Il verbo OPTION viene utilizzato all'interno del server di Riskapp per ottenere informazioni sulla struttura (nomi dei campi dati e relativi tipi di dato) delle entità ricevute a seguito di una chiamata GET o necessarie per le chiamate POST e PUT. Inoltre per determinati campi, la chiamata OPTION restituisce tutti i possibili valori, nel caso siano limitati al lato server, per il campo stesso.

##### 3.2.1.1 Customer

- **descrizione:** l'entità customer contiene le informazioni del cliente;
- **chiamata:** /mitsuko/v01/customer/<uuid>/uuid/
  - **operazioni utilizzate:** GET.

### 3.2.1.2 Graph

- **descrizione:** l'entità grafo proposta da *RiskApp* contiene le informazioni relative al processo produttivo e ai suoi *nodi<sub>G</sub>*;
- **chiamata:** /mitsuko/v01/graph/<uuid>/uuid/
  - **operazioni utilizzate:** GET.

### 3.2.1.3 Asset

- **descrizione:** l'entità asset rappresenta un asset del cliente con le relative informazioni;
- **chiamata:** /mitsuko/v01/customer/<uuid>/asset/new/
  - **operazioni utilizzate:** POST.
- **chiamata:** /mitsuko/v01/asset/<uuid>/uuid/
  - **operazioni utilizzate:** GET, PUT, OPTIONS, UPDATE, DELETE.

### 3.2.1.4 Node

- **descrizione:** l'entità node rappresenta un elemento di interesse strategico all'interno di un asset;
- **chiamata:** /mitsuko/v01/graph/<uuid>/node/new/
  - **operazioni utilizzate:** POST.
- **chiamata:** /mitsuko/v01/node/<uuid>/uuid/
  - **operazioni utilizzate:** GET, PUT, OPTIONS, UPDATE, DELETE.

### 3.2.1.5 Edge

- **descrizione:** l'entità edge rappresenta un collegamento fra due nodi;
- **chiamata:** /mitsuko/v01/graph/<uuid>/edge/new/
  - **operazioni utilizzate:** POST.
- **chiamata:** /mitsuko/v01/edge/<uuid>/uuid/
  - **operazioni utilizzate:** GET, PUT, OPTIONS, UPDATE, DELETE.

### 3.3 Client

Il proponente ha fornito l'ambiente di sviluppo contenente la loro attuale applicazione, denominata in figura come "RiskApp", su cui il gruppo potrà integrare DeGeOP. Il prodotto sarà sviluppato come una single-page accessibile cliccando sulla scheda di menu "Process and analysis".

#### 3.3.1 Design architetturale di DeGeOP

E' stato scelto di utilizzare il design architetturale Redux (per maggiori informazioni su questo pattern, si veda l'appendice A.3.1). Per una migliore gestione e uso di questo pattern si è deciso di rafforzare l'uso classico di Redux, incapsulando le varie componenti in una struttura a classi. Rispetto all'architettura base, ovvero Redux, sono quindi presenti altre componenti:

- **ActionCreators**: struttura il cui compito è creare le "azioni", ossia le operazioni che si occupano di interagire con lo *store<sub>G</sub>*. Le actions come fornite da Redux permettono di definire azioni potenzialmente invalide o la cui esecuzione potrebbe portare ad errori. È quindi nata la necessità di incapsulare il concetto di azione e devolvere il compito della sua creazione ad una *componente<sub>G</sub>* propria;
- **Reducer<sub>G</sub>**: classe di utilità che accetta *action<sub>G</sub>* generiche e le reindirizza alle giuste implementazioni per gestire l'azione in analisi. Anche in questo caso l'incapsulazione è stata adottata per fornire una migliore interfaccia di utilizzo dei reducer.

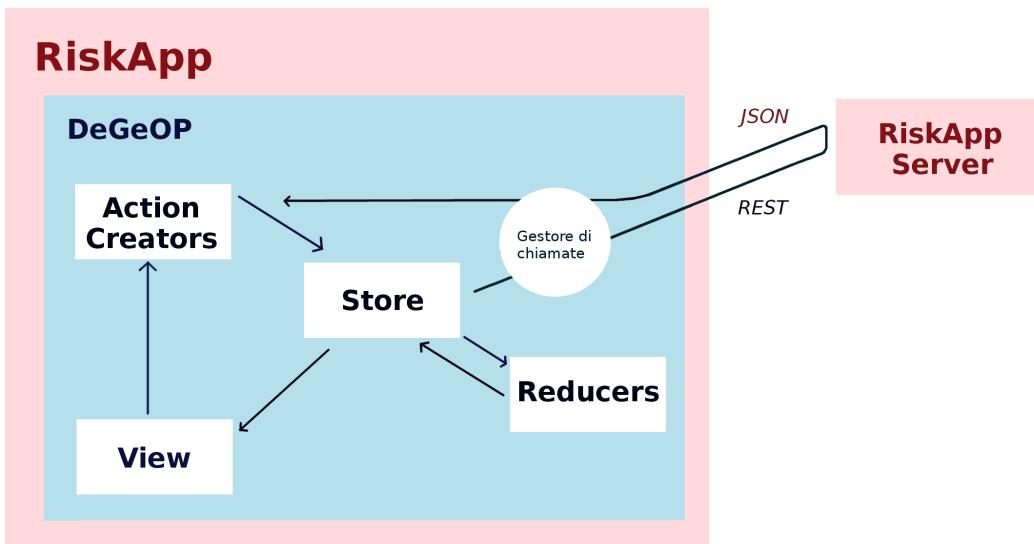


Figura 1: Architettura di base

## 4 Componenti

### 4.1 Introduzione

Partendo da quanto descritto nel documento di *Specifica Tecnica*, è stato fatto uno studio approfondito delle tecnologie da utilizzare, in modo da poter scendere ad un livello di dettaglio tale da permettere di definire con chiarezza e precisione (seppure mantenendo la sintassi UML) la codifica dell'applicazione da parte dei programmatore.

Questo si nota ed esempio all'interno del *ViewPkg*: l'utilizzo di React, ha permesso:

- buon riutilizzo di codice, consentendo di ridurre fortemente il numero di componenti presentazionali che erano state progettate ad alto livello;
- semplificazione della procedura di costruzione dell'interfaccia grafica, grazie al principio di React "lifting state up", che consente la strutturazione dell'interfaccia in maniera fortemente modulare dal punto di vista puramente presentazionale, ma centralizzata dal punto di vista delle informazioni.

### 4.2 DeGeOP

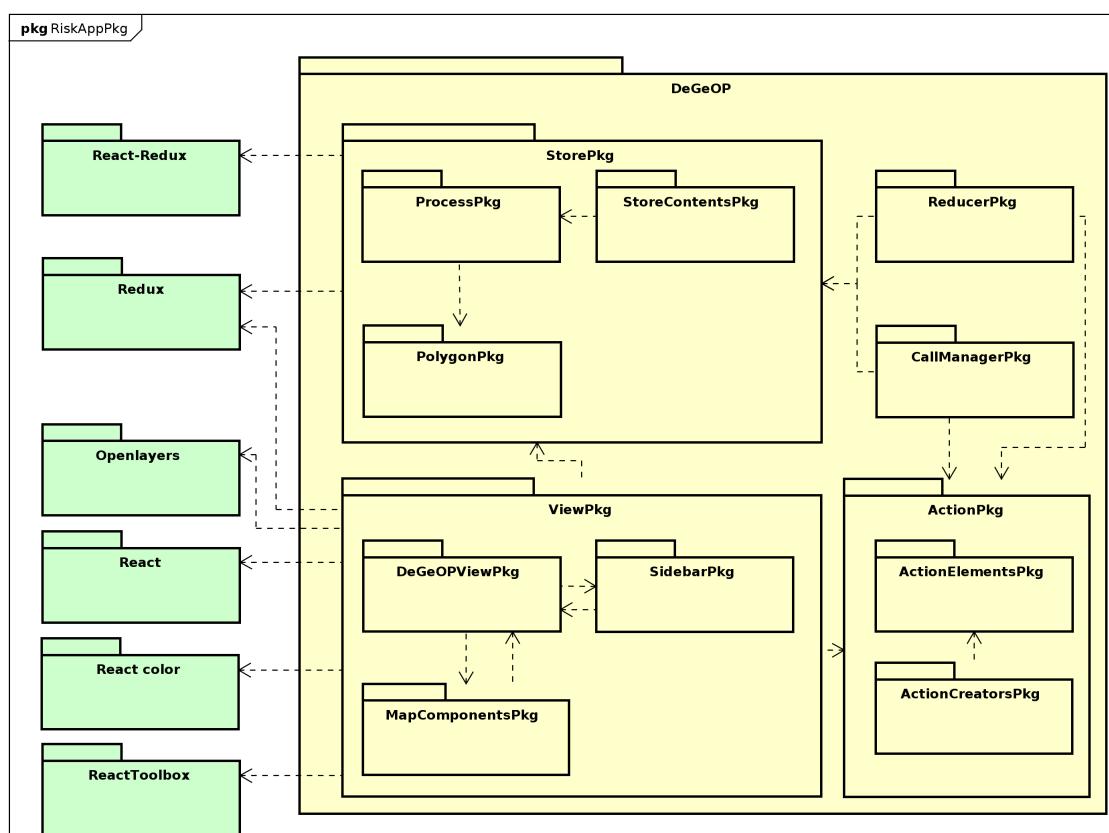


Figura 2: Schema componente DeGeOP

#### 4.2.1 Informazioni sul package

- **descrizione:** racchiude tutte le componenti necessarie per il front-end del prodotto;
- **package contenuti:**

- DeGeOP::[ActionPkg](#);
- DeGeOP::[CallManagerPkg](#);
- DeGeOP::[ReducerPkg](#);
- DeGeOP::[StorePkg](#);
- DeGeOP::[ViewPkg](#).

### 4.3 DeGeOP::StorePkg

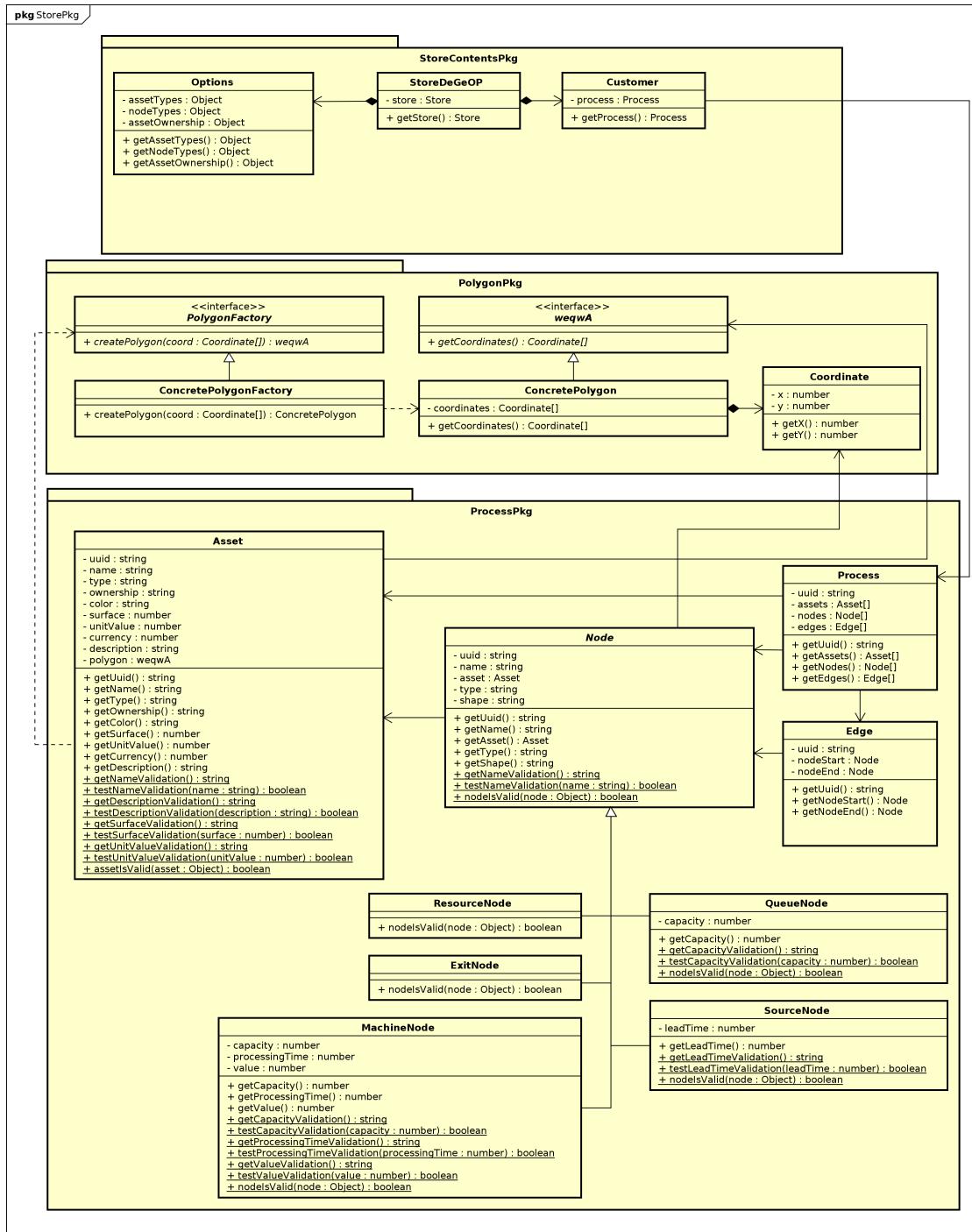


Figura 3: Schema componente DeGeOP::StorePkg

#### 4.3.1 Informazioni sul package

- **descrizione**: racchiude le componenti utilizzate per la memorizzazione e rappresentazione dei dati;
- **padre**: [DeGeOP](#);

- **package contenuti:**

- StorePkg::[PolygonPkg](#);
- StorePkg::[ProcessPkg](#);
- StorePkg::[StoreContentsPkg](#).

- **interazioni con altri package:**

- IN CallManagerPkg: subscribe sullo store;
- IN ReducerPkg: applicazione di cambiamenti di stato;
- IN ViewPkg: subscribe sullo store;
- OUT React-Redux: utilizzo di Provider per evitare di passare lo store come proprietà alle componenti React;
- OUT Redux: creazione Store utilizzando il metodo createStore().

## 4.4 DeGeOP::StorePkg::StoreContentsPkg

### 4.4.1 Informazioni sul package

- **descrizione:** racchiude le componenti che implementano il concetto di store dell'architettura Redux;
- **padre:** [StorePkg](#);
- **interazioni con altri package:**
  - OUT AnalysisPkg: riferimento ad analisi di danno;
  - OUT ProcessPkg: riferimento a processo;
  - OUT React-Redux: utilizzo del Provider;
  - OUT Redux: creazione store.
- **classi contenute:**
  - Customer;
  - Options;
  - StoreDeGeOP.

### 4.4.2 Classi

#### 4.4.2.1 Customer

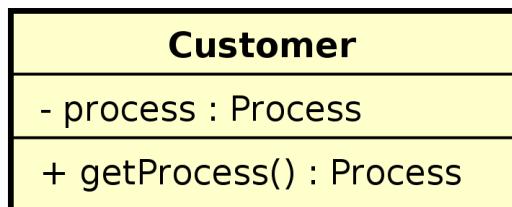


Figura 4: Diagramma classe Customer

- **descrizione:** rappresenta l'assicurando;
- **utilizzo:** viene utilizzato nello Store per memorizzare l'assicurando;
- **attributi:**
  - process : Process
    - \* rappresenta il processo produttivo dell'assicurando.
- **metodi:**
  - +getProcess() : Process
    - il metodo restituisce il processo produttivo dell'assicurando

#### 4.4.2.2 Options

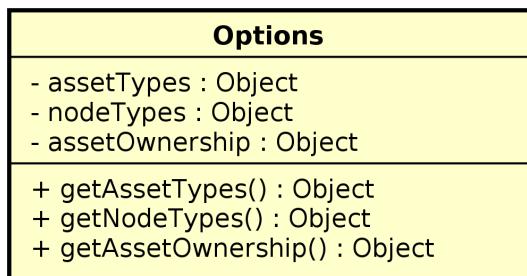


Figura 5: Diagramma classe Options

- **descrizione:** classe contenente le associazioni tra codice e valore dei campi dati degli altri elementi dello store;
- **utilizzo:** viene utilizzata per associare codice e valore dei campi dati degli altri elementi dello store;
- **attributi:**
  - -assetOwnership : Object
    - \* associazione chiave valore delle tipologie di soggetti di appartenenza dell'asset.
  - -assetTypes : Object
    - \* associazione chiave valore delle tipologie di materiale dell'asset.
  - -nodeTypes : Object
    - \* associazione chiave valore delle tipologie di materiale del nodo.
- **metodi:**
  - +getAssetOwnership() : Object
    - il metodo ritorna l'oggetto assetOwnership
  - +getAssetTypes() : Object
    - il metodo ritorna l'oggetto assetTypes
  - +getNodeTypes() : Object
    - il metodo ritorna l'oggetto nodeTypes
- **relazioni con altre classi:**
  - IN OptionReducer;
  - IN StoreDeGeOP.

#### 4.4.2.3 StoreDeGeOP



Figura 6: Diagramma classe StoreDeGeOP

- **descrizione:** rappresenta una classe che incapsula uno Store creato utilizzando Redux;
- **utilizzo:** viene utilizzato per memorizzare lo stato dell'applicazione. Le componenti che effettuano il subscribe sullo Store verranno notificate ad ogni cambiamento di stato dello Store;
- **attributi:**
  - -customer : Customer
    - \* rappresenta il cliente che gestiamo.
  - -store : Object
    - \* rappresenta l'oggetto Store creato utilizzando Redux.
- **metodi:**
  - +getStore() : Object
    - il metodo restituisce lo store creato utilizzando Redux
- **relazioni con altre classi:**
  - IN Reducer;
  - OUT Options.

## 4.5 DeGeOP::StorePkg::ProcessPkg

### 4.5.1 Informazioni sul package

- **descrizione:** racchiude le componenti necessarie alla rappresentazione del processo produttivo dell'assicurando;
- **padre:** [StorePkg](#);
- **interazioni con altri package:**
  - IN StoreContentsPkg: riferimento a processo;
  - OUT PolygonPkg: riferimento ad un poligono.
- **classi contenute:**
  - Asset;
  - Edge;
  - ExitNode;
  - MachineNode;
  - Node;
  - Process;
  - QueueNode;
  - ResourceNode;
  - SourceNode.

### 4.5.2 Classi

#### 4.5.2.1 Asset



Figura 7: Diagramma classe Asset

- **descrizione:** rappresenta un fabbricato di interesse per il processo produttivo dell'assicurando;
- **utilizzo:** sono contenuti all'interno di Process;
- **attributi:**
  - -color : string
    - \* indica il colore dell'asset.
  - -currency : number
    - \* indica la valuta utilizzata.
  - -description : string
    - \* rappresenta la descrizione dell'asset.
  - -name : string
    - \* rappresenta il nome dell'asset.
  - -ownership : string
    - \* rappresenta l'appartenenza dell'asset.
  - -surface : number
    - \* rappresenta la dimensione, in mq, dell'asset.
  - -type : string
    - \* rappresenta la tipologia dell'asset.
  - -unitValue : string
    - \* indica il valore economico dell'asset.
  - -uuid : string
    - \* rappresenta l'identificativo dell'asset (uuid).
- **metodi:**
  - +assetIsValid(asset) : boolean
    - il metodo testa che i parametri con cui l'asset sta per essere creato siano validi
    - \* asset : Object
      - oggetto contenente i parametri di un Asset che dovrà essere validato.
  - +getColor() : string
    - il metodo ritorna il colore dell'asset
  - +getCurrency() : number
    - il metodo ritorna la valuta utilizzata per l'asset
  - +getDescription() : string
    - il metodo ritorna la descrizione dell'asset
  - +getDescriptionValidation() : string
    - il metodo ritorna l'espressione regolare che indica una stringa non contenente le seguenti caratteristiche: vuota; più lunga di 5000 caratteri; inizia e/o finisce con uno spazio; contiene caratteri speciali diversi dall'apostrofo
  - +getName() : string
    - ritorna il nome dell'asset

- `+getNameValidation() : string`  
il metodo ritorna l'espressione regolare che indica una stringa non contenente le seguenti caratteristiche: più lunga di 50 caratteri; inizia e/o finisce con uno spazio; contiene caratteri speciali.
- `+getOwnership() : string`  
il metodo ritorna l'appartenenza dell'asset
- `+getSurface() : number`  
il metodo ritorna la dimensione dell'asset
- `+getSurfaceValidation() : string`  
il metodo ritorna l'espressione regolare che indica una stringa non contenente le seguenti caratteristiche: vuota; più lunga di 5 cifre per la parte intera; più di 2 per la parte decimale;
- `+getType() : string`  
il metodo ritorna la tipologia dell'asset
- `+getUnitValue() : string`  
il metodo ritorna il valore economico dell'asset
- `+getUnitValueValidation() : string`  
il metodo ritorna l'espressione regolare che indica una stringa non contenente le seguenti caratteristiche: vuota; più lunga di 5 cifre per la parte intera; più di 2 per la parte decimale
- `+getUuid() : string`  
il metodo ritorna l'uuid dell'asset
- `+testDescriptionValidation(description) : boolean`  
il metodo ritorna true solamente se il valore ricevuto in input, come parametro, è ritenuto valido.
  - \* `description : string`  
rappresenta la descrizione dell'asset.
- `+testNameValidation(name) : boolean`  
il metodo ritorna true solamente se il valore ricevuto in input, come parametro, è ritenuto valido.
  - \* `name : string`  
rappresenta il nome dell'asset.
- `+testSurfaceValidation(surface) : boolean`  
il metodo ritorna true solamente se il valore ricevuto in input, come parametro, è ritenuto valido.
  - \* `surface : number`  
rappresenta la dimensione, in mq, dell'asset.
- `+testUnitValueValidation(unitValue) : boolean`  
il metodo ritorna true solamente se il valore ricevuto in input, come parametro, è ritenuto valido.
  - \* `unitValue : number`  
indica il valore economico dell'asset.

- **relazioni con altre classi:**

- IN AssetReducer.

#### 4.5.2.2 Edge

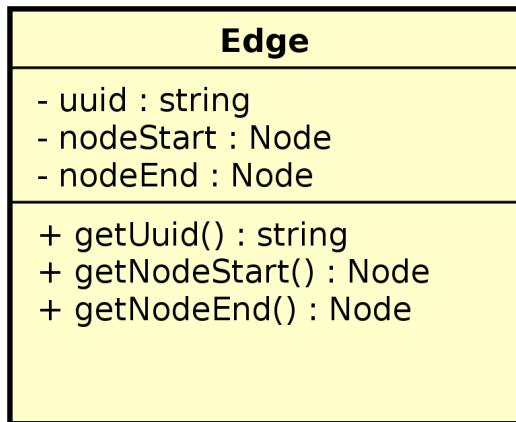


Figura 8: Diagramma classe Edge

- **descrizione:** rappresenta un arco che collega due nodi tra di loro; un arco indica che i nodi sono in correlazione tra di loro;
- **utilizzo:** è contenuto all'interno di Process;
- **attributi:**
  - -nodeEnd : Node
    - \* rappresenta il nodo di arrivo.
  - -nodeStart : Node
    - \* rappresenta il nodo di partenza.
  - -uuid : string
    - \* rappresenta il codice identificativo dell'arco (uuid).
- **metodi:**
  - +getNodeEnd() : Node
    - il metodo restituisce il nodo di arrivo
  - +getNodeStart() : Node
    - il metodo restituisce il nodo di partenza
  - +getUuid() : string
    - il metodo restituisce il codice identificativo dell'arco (uuid)
- **relazioni con altre classi:**
  - IN EdgeReducer.

#### 4.5.2.3 ExitNode

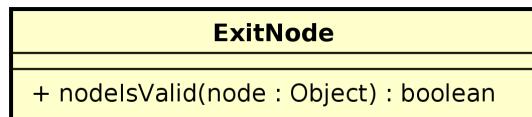


Figura 9: Diagramma classe ExitNode

- **descrizione:** rappresenta un nodo di tipo Uscita;
- **utilizzo:** è contenuto all'interno di Process;
- **metodi:**
  - +nodeisValid() : boolean  
il metodo testa che i parametri con cui il nodo sta per essere creato siano validi

#### 4.5.2.4 MachineNode

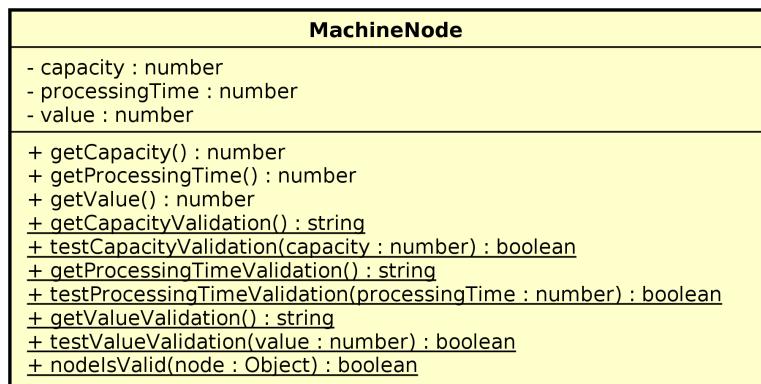


Figura 10: Diagramma classe MachineNode

- **descrizione:** rappresenta un nodo di tipo Macchina;
- **utilizzo:** è contenuto all'interno di Process;
- **attributi:**
  - -capacity : number
    - \* rappresenta la capacità del nodo.
  - -processingTime : number
    - \* rappresenta il tempo di processo del nodo.
  - -value : number
    - \* rappresenta il valore del nodo.
- **metodi:**

- `+getCapacity() : number`  
il metodo ritorna la capacità del nodo
- `+getCapacityValidation() : string`  
il metodo ritorna l'espressione regolare che indica una stringa non contenente le seguenti caratteristiche: vuota; più lunga di 5 cifre per la parte intera; più di 2 per la parte decimale;
- `+getProcessingTime() : number`  
il metodo ritorna il tempo di processo del nodo
- `+getProcessingTimeValidation() : string`  
il metodo ritorna l'espressione regolare che indica una stringa non contenente le seguenti caratteristiche: vuota; più lunga di 5 cifre per la parte intera; più di 2 per la parte decimale;
- `+getValue() : number`  
il metodo ritorna il valore del nodo
- `+getValueValidation() : string`  
il metodo ritorna l'espressione regolare che indica una stringa non contenente le seguenti caratteristiche: vuota; più lunga di 20 cifre per la parte intera; più di 2 per la parte decimale
- `+nodeisValid(node) : boolean`  
il metodo testa se i parametri con cui il nodo sta per essere creato sono validi
  - \* `node : Object`  
oggetto che rappresenta i parametri con cui il nodo sta per essere creato.
- `+testCapacityValidation() : boolean`  
il metodo testa se la capacità del nodo valida
- `+testProcessingTimeValidation(processingTime) : boolean`  
il metodo testa se il tempo di processo valida
  - \* `processingTime : number`  
rappresenta il tempo di processo del nodo.
- `+testValueValidation(value) : boolean`  
il metodo testa se il valore del nodo valida
  - \* `value : number`  
rappresenta il valore del nodo.

#### 4.5.2.5 Node

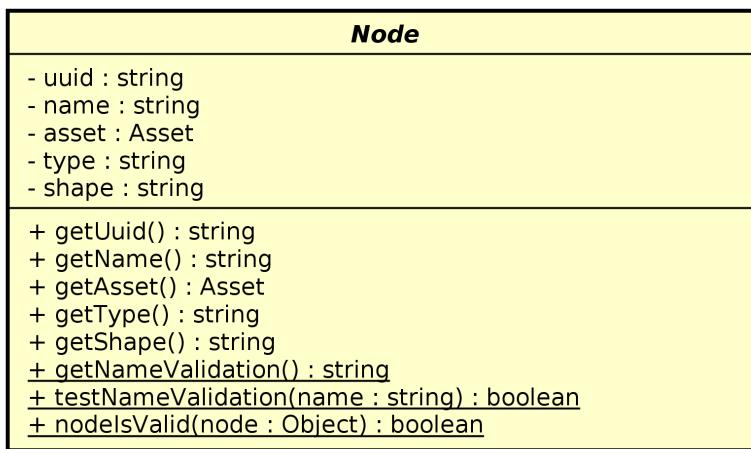


Figura 11: Diagramma classe Node

- **descrizione:** rappresenta un nodo contenuto all'interno di un Asset ;
- **utilizzo:** è contenuto all'interno di Process;
- **attributi:**
  - -asset : Asset
    - \* rappresenta l'asset a cui il nodo appartiene.
  - -name : string
    - \* rappresenta il nome del nodo.
  - -shape : string
    - \* rappresenta la forma del nodo.
  - -type : string
    - \* rappresenta la classe del nodo.
  - -uuid : string
    - \* rappresenta il codice identificativo del nodo (uuid).
- **metodi:**
  - +getAsset() : Asset
    - il metodo ritorna l'asset di appartenenza del nodo
  - +getName() : string
    - il metodo ritorna il codice identificativo del nodo
  - +getNameValidation() : string
    - il metodo ritorna l'espressione regolare che indica una stringa non contenente le seguenti caratteristiche: vuota; più lunga di 50 caratteri; inizia e/o finisce con uno spazio; contiene caratteri speciali;
  - +getShape() : string
    - il metodo ritorna la forma del nodo

- `+getType() : string`  
il metodo ritorna la classe del nodo
- `+getUuid() : string`  
il metodo ritorna il codice identificativo del nodo (uuid)
- `+nodesisValid() : boolean`  
il metodo testa che i parametri con cui il nodo sta per essere creato siano validi
- `+testNameValidation(name) : boolean`  
il metodo ritorna true solamente se il valore ricevuto in input, come parametro, è ritenuto valido.
  - \* `name : string`  
rappresenta il nome del nodo.

- **relazioni con altre classi:**

- IN `NodeReducer`.

#### 4.5.2.6 Process

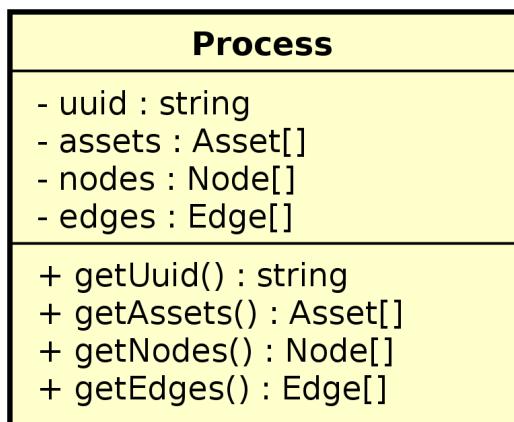


Figura 12: Diagramma classe Process

- **descrizione:** rappresenta un processo produttivo dell'azienda dell'assicurando;
- **utilizzo:** è memorizzato nello Store;
- **attributi:**
  - `-assets : Asset[]`
    - \* rappresenta gli asset facenti parti del processo produttivo.
  - `-edges : Edge []`
    - \* rappresenta gli archi che effettuano i collegamenti nel processo produttivo.
  - `-nodes : Node []`
    - \* rappresenta i nodi facenti parti del processo produttivo.
  - `-uuid : string`
    - \* rappresenta l'identificativo del processo (uuid).

- **metodi:**

- `+getAssets() : Asset[ ]`  
il metodo ritorna gli asset facenti parti del processo produttivo
- `+getEdges() : Edge [ ]`  
il metodo ritorna gli archi che effettuano i collegamenti nel processo produttivo
- `+getNodes() : Node [ ]`  
il metodo ritorna i nodi facenti parti del processo produttivo
- `+getUuid() : string`  
il metodo ritorna il codice identificativo (uuid) del processo

#### 4.5.2.7 QueueNode

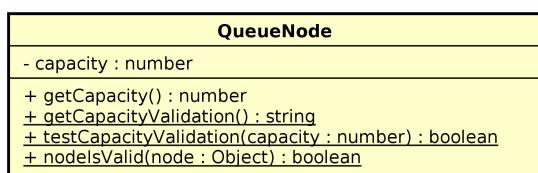


Figura 13: Diagramma classe QueueNode

- **descrizione:** rappresenta un nodo di tipo Coda;

- **utilizzo:** è contenuto all'interno di Process;

- **attributi:**

- `-capacity : number`  
\* rappresenta la capacità del nodo coda.

- **metodi:**

- `+getCapacity() : number`  
il metodo ritorna la capacità del nodo
- `+getCapacityValidation() : string`  
il metodo ritorna l'espressione regolare che indica una stringa non contenente le seguenti caratteristiche: vuota; più lunga di 5 cifre per la parte intera; più di 2 per la parte decimale;
- `+nodeisValid(node : Object)` : boolean  
il metodo testa che i parametri con cui il nodo sta per essere creato siano validi
  - \* `node : Object`  
oggetto che rappresenta i parametri con cui il nodo sta per essere creato.
- `+testCapacityValidation(capacity : number) : boolean`  
il metodo ritorna true solamente se il valore ricevuto in input, come parametro, è ritenuto valido.
  - \* `capacity : number`  
rappresenta la capacità del nodo.

#### 4.5.2.8 ResourceNode

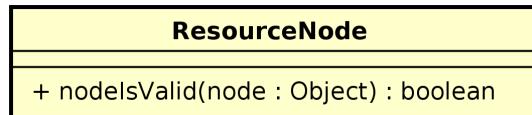


Figura 14: Diagramma classe ResourceNode

- **descrizione:** rappresenta un nodo di tipo Risorsa;
- **utilizzo:** è contenuto all'interno di Process;
- **metodi:**
  - +nodeisValid() : boolean  
il metodo testa che i parametri con cui il nodo sta per essere creato siano validi

#### 4.5.2.9 SourceNode

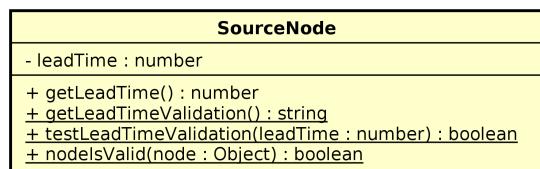


Figura 15: Diagramma classe SourceNode

- **descrizione:** rappresenta un nodo di tipo Sorgente;
- **utilizzo:** è contenuto all'interno di Process;
- **attributi:**
  - -leadTime : Object
    - \* rappresenta il tempo di approvvigionamento del nodo.
- **metodi:**
  - +getLeadTime() : number  
il metodo ritorna il tempo di approvvigionamento del nodo
  - +getLeadTimeValidation() : string  
il metodo ritorna l'espressione regolare che indica una stringa non contenente le seguenti caratteristiche: vuota; più lunga di 5 cifre per la parte intera; più di 2 per la parte decimale;
  - +nodeisValid(node) : boolean  
il metodo testa se i parametri con cui il nodo sta per essere creato sono validi
    - \* node : Object  
oggetto che rappresenta i parametri con cui il nodo sta per essere creato.

- `+testLeadTimeValidation(leadTime) : boolean`
  - il metodo testa se il tempo di approvvigionamento è valido
  - \* `leadTime : number`
  - rappresenta il tempo di approvvigionamento.

## 4.6 DeGeOP::StorePkg::PolygonPkg

### 4.6.1 Informazioni sul package

- **descrizione:** racchiude le componenti necessarie alla rappresentazione dell'area degli asset e degli scenari di danno;
- **padre:** [StorePkg](#);
- **interazioni con altri package:**
  - IN AnalysisPkg: riferimento ad un poligono;
  - IN ProcessPkg: riferimento ad un poligono.
- **classi contenute:**
  - [ConcretePolygon](#);
  - [ConcretePolygonFactory](#);
  - [Coordinate](#);
  - [Polygon](#);
  - [PolygonFactory](#).

### 4.6.2 Classi

#### 4.6.2.1 ConcretePolygon

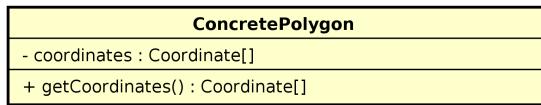


Figura 16: Diagramma classe ConcretePolygon

- **descrizione:** rappresenta un poligono;
- **utilizzo:** viene istanziata da [ConcretePolygonFactory](#); viene utilizzata in Asset e Scenario;
- **attributi:**
  - coordinates : Coordinate[]  
\* rappresenta le coordinate che vanno a delineare il poligono.
- **metodi:**
  - +getCoordinates() : Coordinate[]  
il metodo restituisce le coordinate che vanno a delineare il poligono

#### 4.6.2.2 ConcretePolygonFactory

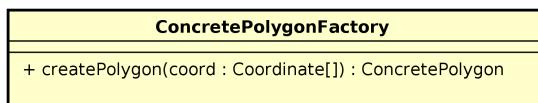


Figura 17: Diagramma classe ConcretePolygonFactory

- **descrizione:** gestisce la creazione concreta dei poligoni;
- **utilizzo:** implementazione di PolygonFactory; è la classe concreta da istanziare per gestire la creazione di un poligono;
- **metodi:**
  - `+createPolygon() : ConcretePolygon`  
il metodo si occupa della costruzione del poligono

#### 4.6.2.3 Coordinate

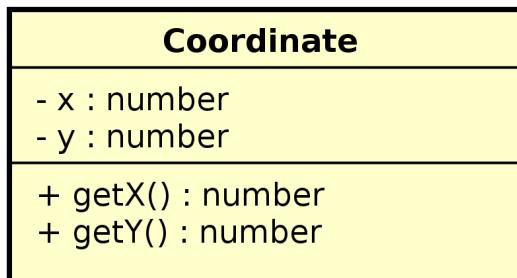


Figura 18: Diagramma classe Coordinate

- **descrizione:** rappresenta una coordinata geografica;
- **utilizzo:** è utilizzata all'interno di Polygon per delimitarne i suoi vertici;
- **attributi:**
  - `-x : number`
    - \* rappresenta la latitudine di una coordinata geografica.
  - `-y : number`
    - \* rappresenta la longitudine di una coordinata geografica.
- **metodi:**
  - `+getX() : number`  
restituisce la latitudine della coordinata geografica
  - `+getY() : number`  
restituisce la longitudine della coordinata geografica

#### 4.6.2.4 Polygon

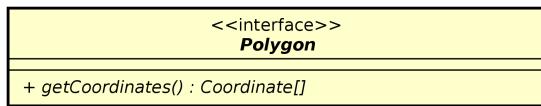


Figura 19: Diagramma classe Polygon

- **descrizione:** interfaccia che rappresenta il poligono;
- **utilizzo:** fornisce i metodi del poligono;
- **metodi:**
  - `+getCoordinates() : Coordinate[]`  
il metodo restituisce un array di coordinate

#### 4.6.2.5 PolygonFactory



Figura 20: Diagramma classe PolygonFactory

- **descrizione:** interfaccia che si occupa della costruzione dei poligoni;
- **utilizzo:** viene usata dalle classi Scenario e Asset per la costruzione dei poligoni;
- **metodi:**
  - `+createPolygon(coord) : Polygon`  
il metodo si occupa della costruzione del poligono
    - \* `coord : Coordinate[]`  
raccoglie le coordinate necessarie alla costruzione dei poligoni .

## 4.7 DeGeOP::ReducerPkg

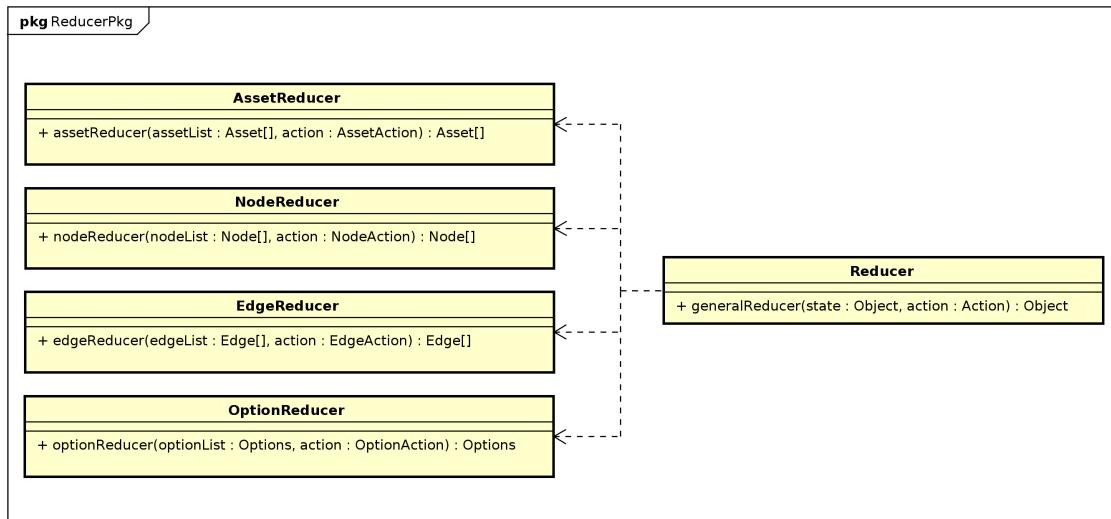


Figura 21: Schema componente DeGeOP::ReducerPkg

### 4.7.1 Informazioni sul package

- **descrizione:** racchiude le componenti necessarie all'implementazione dei reducer secondo l'architettura Redux;
- **padre:** [DeGeOP](#);
- **interazioni con altri package:**
  - OUT ActionPkg: utilizzo di azioni ;
  - OUT StorePkg: applicazione di cambiamenti di stato.
- **classi contenute:**
  - AssetReducer;
  - EdgeReducer;
  - NodeReducer;
  - OptionReducer;
  - Reducer.

### 4.7.2 Classi

#### 4.7.2.1 AssetReducer

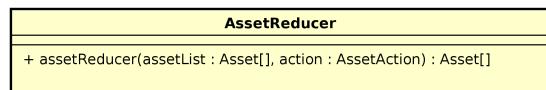


Figura 22: Diagramma classe AssetReducer

- **descrizione:** rappresenta il reducer dell'asset;

- **utilizzo:** il suo metodo gestisce le operazioni sullo store riguardanti l'asset;
- **metodi:**
  - `+assetReducer(action, assetList) : Asset[ ]`  
il metodo gestisce le operazioni sullo store riguardanti l'asset e ritorna la nuova lista di asset ottenuta a seguito della modifica
    - \* `action : AssetAction`  
rappresenta un'azione che descrive i cambiamenti da effettuare sullo stato.
    - \* `assetList : Asset[ ]`  
rappresenta una lista di asset, che sono il vecchio stato.
- **relazioni con altre classi:**
  - OUT Asset.

#### 4.7.2.2 EdgeReducer

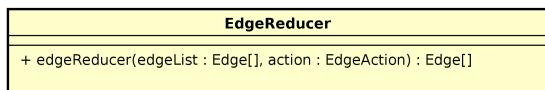


Figura 23: Diagramma classe EdgeReducer

- **descrizione:** rappresenta il reducer dell'arco;
- **utilizzo:** il suo metodo gestisce le operazioni sullo store riguardanti gli archi;
- **metodi:**
  - `+edgeReducer(action, edgeList) : Edge[ ]`  
il metodo gestisce le operazioni sullo store riguardanti gli archi e ritorna la nuova lista di archi ottenuta a seguito della modifica
    - \* `action : EdgeAction`  
rappresenta un'azione che descrive i cambiamenti da effettuare sullo stato.
    - \* `edgeList : Edge[ ]`  
rappresenta una lista di archi, che sono il vecchio stato.
- **relazioni con altre classi:**
  - OUT Edge.

#### 4.7.2.3 NodeReducer

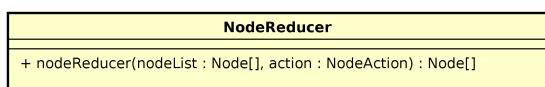


Figura 24: Diagramma classe NodeReducer

- **descrizione:** rappresenta il reducer del nodo;

- **utilizzo:** il suo metodo gestisce le operazioni sullo store riguardanti i nodi;
- **metodi:**
  - `+nodeReducer(action, nodeList) : Node [ ]`  
il metodo gestisce le operazioni sullo store riguardanti i nodi e ritorna la nuova lista di nodi ottenuta a seguito della modifica
    - \* `action : EdgeAction`  
rappresenta un'azione che descrive i cambiamenti da effettuare sullo stato.
    - \* `nodeList : Node [ ]`  
rappresenta una lista di nodi, che sono il vecchio stato.
- **relazioni con altre classi:**
  - OUT Node.

#### 4.7.2.4 OptionReducer

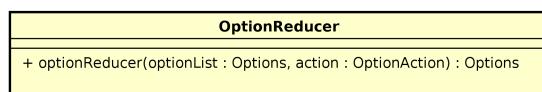


Figura 25: Diagramma classe OptionReducer

- **descrizione:** rappresenta il reducer delle option;
- **utilizzo:** il suo metodo gestisce le operazioni sullo store riguardanti l'asset;
- **metodi:**
  - `+optionReducer(action, optionList) : Options`  
il metodo gestisce le operazioni sullo store riguardanti l'asset e ritorna la nuova lista di asset ottenuta a seguito della modifica
    - \* `action : OptionAction`  
rappresenta un'azione che descrive i cambiamenti da effettuare sullo stato.
    - \* `optionList : Options`  
rappresenta la lista delle option.
- **relazioni con altre classi:**
  - IN Reducer;
  - OUT Options.

#### 4.7.2.5 Reducer

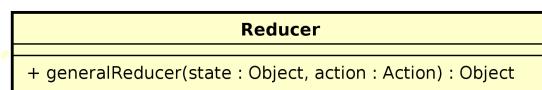


Figura 26: Diagramma classe Reducer

- **descrizione:** accetta una action generica in input e la reindirizza al giusto reducer;

- **utilizzo:** il suo metodo viene utilizzato per catturare un'azione e generare un nuovo stato sullo Store;
- **metodi:**
  - `+generalReducer() : Object`  
Esegue l'azione sullo stato corrente dello store e ne ritorna il nuovo stato
- **relazioni con altre classi:**
  - OUT OptionReducer;
  - OUT StoreDeGeOP.

## 4.8 DeGeOP::CallManagerPkg

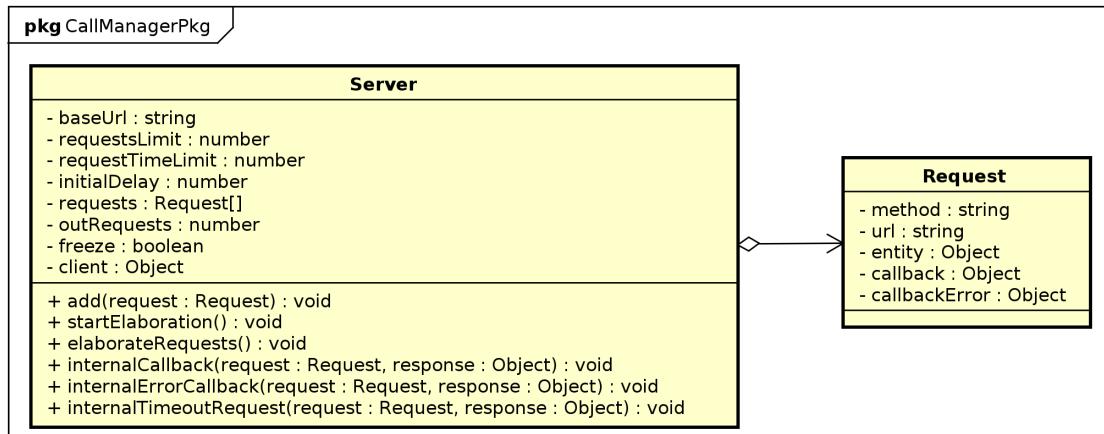


Figura 27: Schema componente DeGeOP::CallManagerPkg

### 4.8.1 Informazioni sul package

- **descrizione:** racchiude le componenti necessarie alla comunicazione dei dati verso il server;
- **padre:** [DeGeOP](#);
- **interazioni con altri package:**
  - OUT ActionPkg: dispatch di azioni;
  - OUT StorePkg: subscribe sullo store.
- **classi contenute:**
  - Request;
  - Server.

### 4.8.2 Classi

#### 4.8.2.1 Request

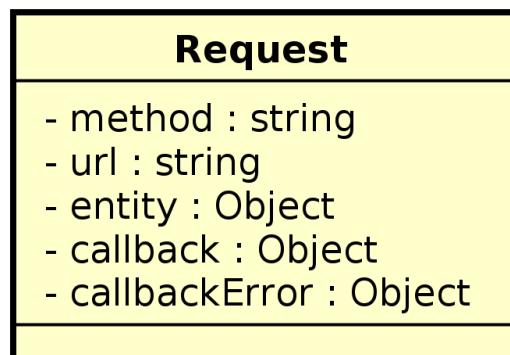


Figura 28: Diagramma classe Request

- **descrizione:** rappresenta i dati necessari per eseguire una richiesta di tipologia REST;
- **utilizzo:** è utilizzata all'interno di Server per gestire la coda delle richieste ;
- **attributi:**
  - callback : Object
    - \* rappresenta la funzione che viene invocata se la richiesta è andata a buon fine.
  - callbackError : Object
    - \* rappresenta la funzione che viene invocata se la richiesta non è andata a buon fine.
  - entity : Object
    - \* rappresenta il payload della richiesta.
  - method : string
    - \* rappresenta il verbo http con cui si esegue la richiesta.
  - url : string
    - \* rappresenta l'url verso cui eseguire la richiesta.

#### 4.8.2.2 Server

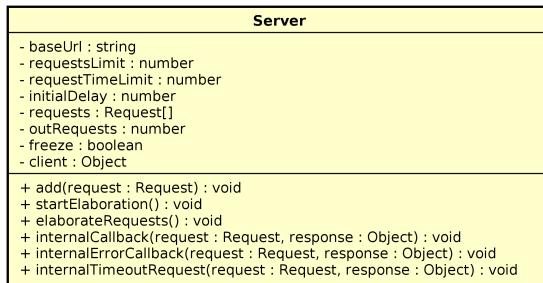


Figura 29: Diagramma classe Server

- **descrizione:** rappresenta il gestore delle chiamate REST;
- **utilizzo:** è utilizzato come buffer per gestire le richieste in uscita e le risposte ricevute;
- **attributi:**
  - baseUrl : string
    - \* URL http verso cui devono essere inviate le richieste.
  - client : Object
    - \* gestisce l'invio delle richieste e la ricezione delle risposte.
  - freeze : boolean
    - \* viene utilizzata per bloccare il server in caso di collegamento mancante.
  - initialDelay : number
    - \* rappresenta il tempo in millisecondi passato il quale una richiesta scaduta viene eseguita nuovamente.

- `-outRequests : number`
  - \* rappresenta il numero delle richieste attualmente in attesa di risposta.
- `-requests : Request[]`
  - \* lista delle richieste da evadere.
- `-requestsLimit : number`
  - \* rappresenta il numero massimo di richieste eseguite ancora in attesa di risposta.
- `-timeLimitRequest : number`
  - \* rappresenta il tempo in millisecondi dopo cui considerare la richiesta scaduta.

- **metodi:**

- `+add(request) : void`
  - aggiunge una richiesta alla coda del server
  - \* `request : Request`
    - rappresenta la richiesta da aggiungere alla coda server.
- `+elaborateRequests() : void`
  - elabora la coda della richieste fino a quando ci sono elementi presenti
- `+internalCallback(request, response) : void`
  - esegue la chiamata alla funzione di callback fornita dalla richiesta nel caso in cui questa abbia ricevuto una risposta senza errori
  - \* `request : Request`
    - rappresenta la richiesta che è stata inviata.
  - \* `response : Object`
    - oggetto che rappresenta la risposta alla richiesta inviata.
- `+internalErrorCallback(request, response) : void`
  - esegue la chiamata alla funzione di callback fornita dalla richiesta nel caso in cui questa abbia ricevuto una risposta con errori
  - \* `request : Request`
    - rappresenta la richiesta che è stata inviata.
  - \* `response : Object`
    - oggetto che rappresenta la risposta alla richiesta inviata.
- `+internalTimeoutRequest(response) : void`
  - gestisce la richiesta nel caso in cui questa abbia non abbia ricevuto alcuna risposta entro il tempo di timeout
  - \* `response : Object`
    - oggetto che rappresenta la risposta alla richiesta inviata.
- `+startElaboration() : void`
  - inizia l'esecuzione delle richieste in attesa di essere evase

## 4.9 DeGeOP::ActionPkg

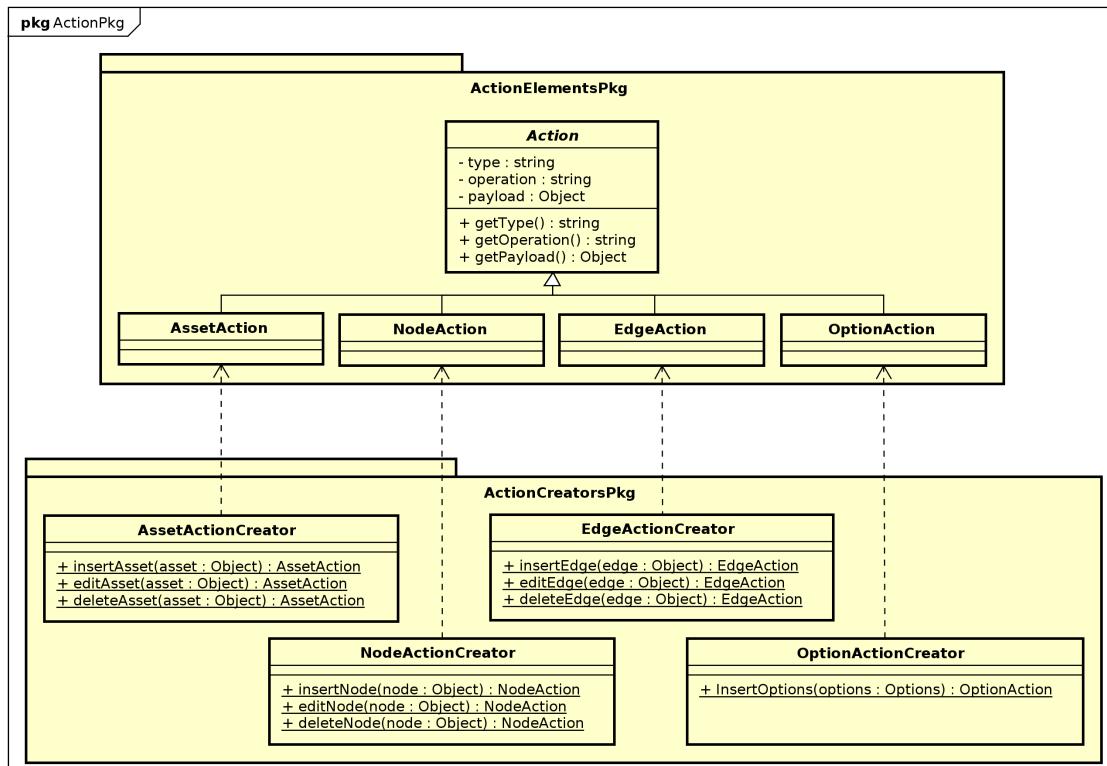


Figura 30: Schema componente DeGeOP::ActionPkg

### 4.9.1 Informazioni sul package

- **descrizione:** racchiude le componenti utilizzate per implementare le action dell'architettura Redux. Le action vengono create e ne viene fatto il dispatch verso lo store. Un reducer gestirà una action per produrre un cambiamento di stato sullo store;
- **padre:** [DeGeOP](#);
- **package contenuti:**
  - [ActionPkg::ActionCreatorsPkg](#);
  - [ActionPkg::ActionElementsPkg](#).
- **interazioni con altri package:**
  - IN CallManagerPkg: dispatch di azioni;
  - IN ReducerPkg: utilizzo di azioni ;
  - IN ViewPkg: dispatch di azioni.
- **classi contenute:**
  - [EdgeAction](#).

#### 4.9.2 Classi

##### 4.9.2.1 EdgeAction



Figura 31: Diagramma classe EdgeAction

- **descrizione:** rappresenta un'azione relativa agli archi;
- **utilizzo:** l'azione viene creata da un apposito ActionCreator per essere poi inviata ad un reducer.

## 4.10 DeGeOP::ActionPkg::ActionElementsPkg

### 4.10.1 Informazioni sul package

- **descrizione:** racchiude le componenti che rappresentano effettivamente le azioni;
- **padre:** [ActionPkg](#);
- **interazioni con altri package:**
  - IN ActionCreatorsPkg: creazione di azioni.
- **classi contenute:**
  - Action;
  - AssetAction;
  - NodeAction;
  - OptionAction.

### 4.10.2 Classi

#### 4.10.2.1 Action

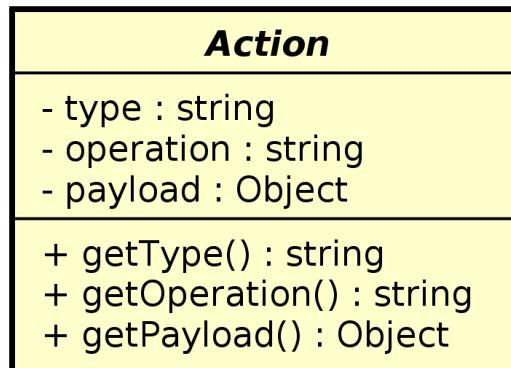


Figura 32: Diagramma classe Action

- **descrizione:** una classe astratta che rappresenta una generica azione di cui può essere fatto il dispatch;
- **utilizzo:** i suoi membri vengono usati dai reducer per completare una azione;
- **attributi:**
  - operation : string
    - \* rappresenta l'operazione da eseguire.
  - payload : Object
    - \* rappresenta l'oggetto che descrive il cambiamento apportato dall'azione.
  - type : string
    - \* rappresenta la tipologia di elemento su cui eseguire l'azione.

- **metodi:**

- `+getOperation() : string`  
il metodo ritorna l'operazione eseguita dall'azione
- `+getPayload() : Object`  
il metodo ritorna il payload dell'oggetto
- `+getType() : string`  
il metodo ritorna il tipo dell'azione

#### 4.10.2.2 AssetAction

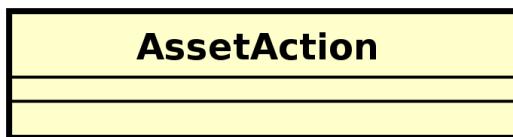


Figura 33: Diagramma classe AssetAction

- **descrizione:** rappresenta un'azione relativa agli asset;
- **utilizzo:** l'azione viene creata da un apposito ActionCreator per essere poi inviata ad un reducer.

#### 4.10.2.3 NodeAction



Figura 34: Diagramma classe NodeAction

- **descrizione:** rappresenta un'azione relativa ai nodi;
- **utilizzo:** l'azione viene creata da un apposito ActionCreator per essere poi inviata ad un reducer.

#### 4.10.2.4 OptionAction



Figura 35: Diagramma classe OptionAction

- **descrizione:** rappresenta un'azione relativa all'oggetto option;
- **utilizzo:** l'azione viene creata da un apposito ActionCreator per essere poi inviata ad un reducer;
- **relazioni con altre classi:**
  - IN OptionActionCreator.

## 4.11 DeGeOP::ActionPkg::ActionCreatorsPkg

### 4.11.1 Informazioni sul package

- **descrizione:** racchiude le componenti che gestiscono la creazione delle azioni;
- **padre:** ActionPkg;
- **interazioni con altri package:**
  - OUT ActionElementsPkg: creazione di azioni.
- **classi contenute:**
  - AssetActionCreator;
  - EdgeActionCreator;
  - NodeActionCreator;
  - OptionActionCreator.

### 4.11.2 Classi

#### 4.11.2.1 AssetActionCreator

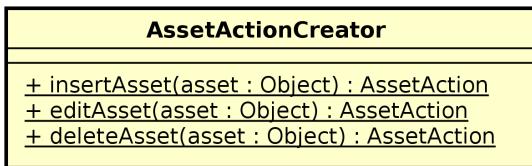


Figura 36: Diagramma classe AssetActionCreator

- **descrizione:** rappresenta la factory di azioni relative agli asset;
- **utilizzo:** i suoi metodi sono chiamati dalla View e dal CallManager per la creazione di azioni relative agli asset;
- **metodi:**
  - `+deleteAsset(asset) : AssetAction`  
il metodo crea l'azione relativa all'eliminazione dell'asset ricevuto in input  
\* asset : Object  
oggetto contenente i parametri di un Asset che dovrà essere eliminato.
  - `+editAsset(asset) : AssetAction`  
il metodo crea l'azione relativa alla modifica di un asset  
\* asset : Object  
oggetto contenente i parametri di un Asset che dovrà essere modificato nello store.
  - `+insertAsset(asset) : AssetAction`  
il metodo crea l'azione relativa all'inserimento di un nuovo asset  
\* asset : Object  
oggetto contenente i parametri di un Asset che dovrà essere inserito nello store.

#### 4.11.2.2 EdgeActionCreator

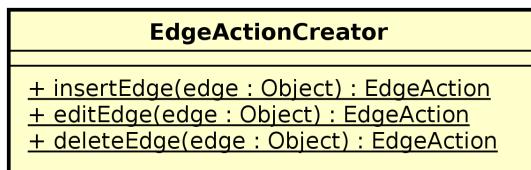


Figura 37: Diagramma classe EdgeActionCreator

- **descrizione:** rappresenta la factory di azioni relative agli archi;
- **utilizzo:** i suoi metodi sono chiamati dalla View e dal CallManager per la creazione di azioni relative agli archi;
- **metodi:**
  - `+deleteEdge(edge : Object)`  
il metodo crea l'azione relativa all'eliminazione di un arco
    - \* `edge : Object`  
oggetto contenente i parametri di un arco che dovrà essere eliminato.
  - `+editEdge(edge : Object)`  
il metodo crea l'azione relativa alla modifica di un arco
    - \* `edge : Object`  
oggetto contenente i parametri di un arco che dovrà essere modificato nello store.
  - `+insertEdge(edge : Object)`  
il metodo crea l'azione relativa all'inserimento di un nuovo arco
    - \* `edge : Object`  
oggetto contenente i parametri di un arco che dovrà essere inserito nello store.

#### 4.11.2.3 NodeActionCreator

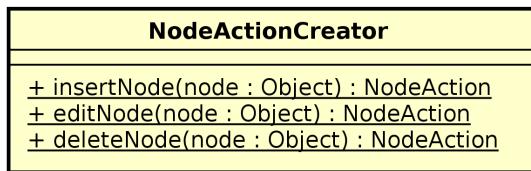


Figura 38: Diagramma classe NodeActionCreator

- **descrizione:** rappresenta la factory di azioni relative ai nodi;
- **utilizzo:** i suoi metodi sono chiamati dalla View e dal CallManager per la creazione di azioni relative ai nodi;
- **metodi:**

- `+deleteNode(node) : NodeAction`  
il metodo crea l'azione relativa all'eliminazione di un nodo
  - \* `node : Object`  
oggetto contenente i parametri di un nodo che dovrà essere eliminato.
- `+editNode(node) : NodeAction`  
il metodo crea l'azione relativa alla modifica di un nodo
  - \* `node : Object`  
oggetto contenente i parametri di un nodo che dovrà essere modificato nello store.
- `+insertNode(node) : NodeAction`  
il metodo crea l'azione relativa all'inserimento di un nuovo nodo
  - \* `node : Object`  
oggetto contenente i parametri di un nodo che dovrà essere inserito nello store.

#### 4.11.2.4 OptionActionCreator

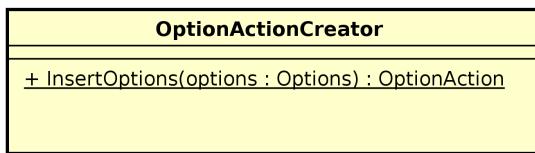


Figura 39: Diagramma classe OptionActionCreator

- **descrizione:** rappresenta la factory di azioni relative agli asset;
- **utilizzo:** i suoi metodi sono chiamati dalla View e dal CallManager per la creazione di azioni relative agli asset;
- **metodi:**
  - `+insertOptions() : OptionAction`  
il metodo crea l'azione relativa all'inserimento dell'oggetto options nello store
- **relazioni con altre classi:**
  - OUT OptionAction.

## 4.12 DeGeOP::ViewPkg

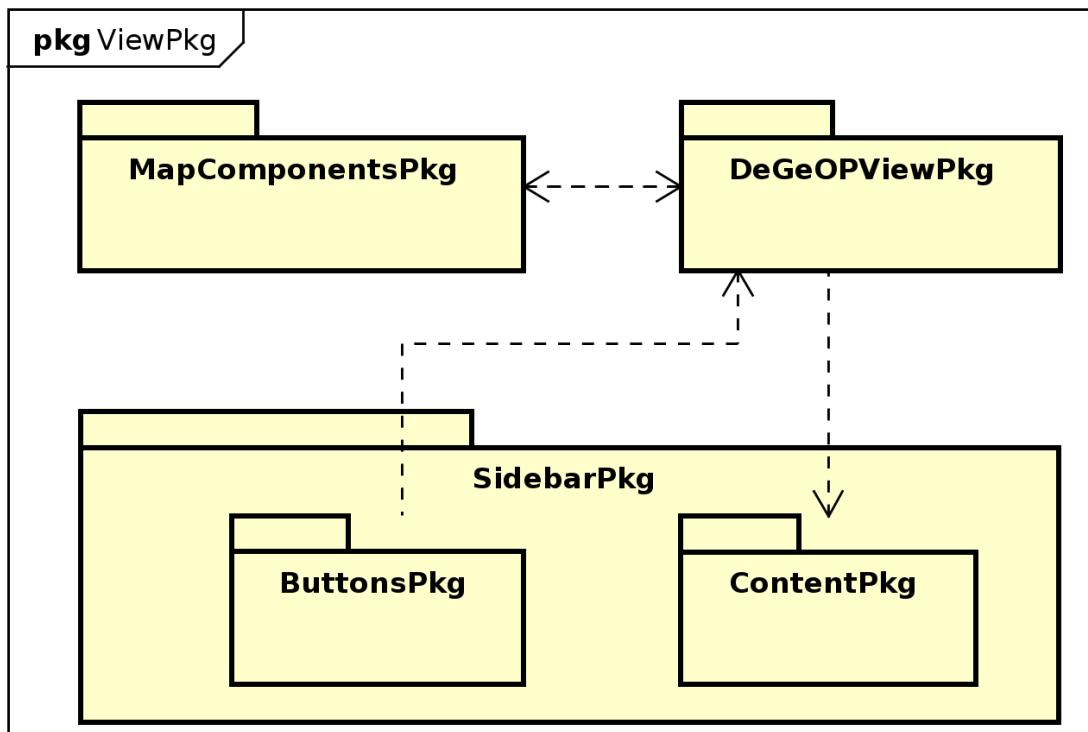


Figura 40: Schema componente DeGeOP::ViewPkg

### 4.12.1 Informazioni sul package

- **descrizione:** racchiude le componenti per la visualizzazione dell'interfaccia utente;
- **padre:** [DeGeOP](#);
- **package contenuti:**
  - [ViewPkg::DeGeOPViewPkg](#);
  - [ViewPkg::MapComponentsPkg](#);
  - [ViewPkg::SidebarPkg](#).
- **interazioni con altri package:**
  - OUT ActionPkg: dispatch di azioni;
  - OUT Alexa voice service: gestore vocale;
  - OUT Hammer: gestione gesture ;
  - OUT Openlayers: gestione mappa;
  - OUT React: utilizzo componenti react;
  - OUT ReactToolbox: utilizzo componenti material design;
  - OUT Redux: utilizzo metodo dispatch;
  - OUT StorePkg: subscribe sullo store.

### 4.13 DeGeOP::ViewPkg::DeGeOPViewPkg

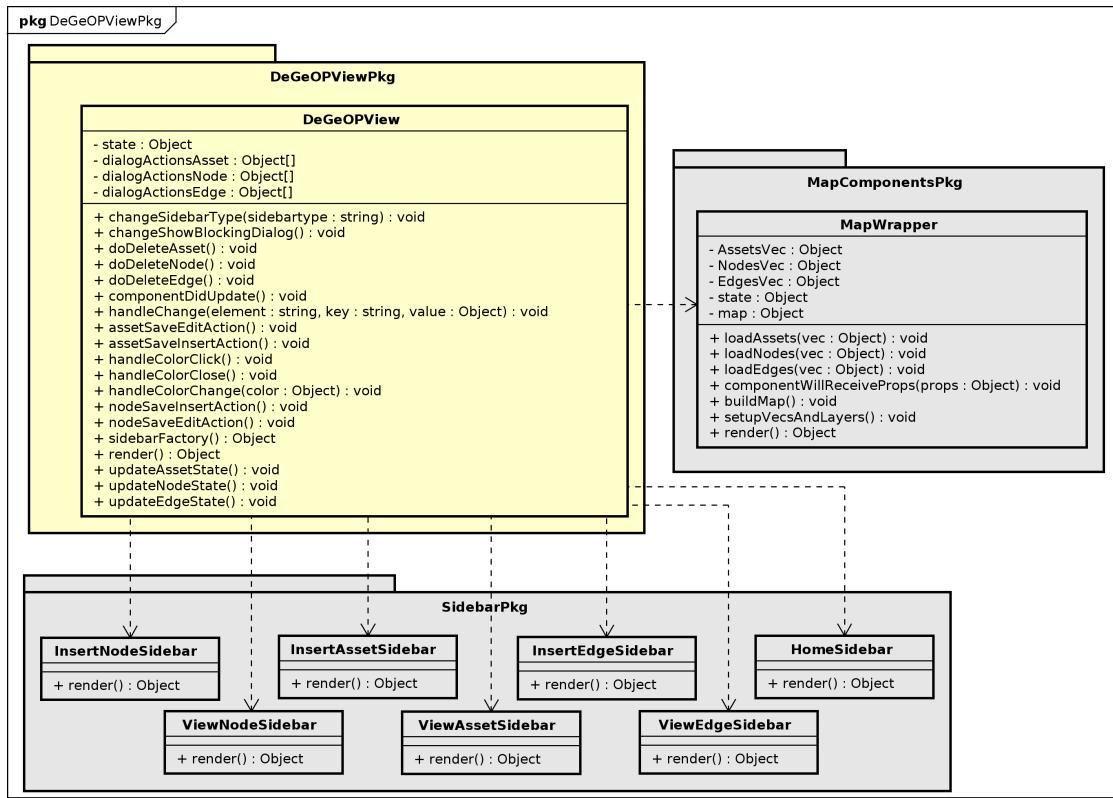


Figura 41: Schema componente DeGeOP::ViewPkg::DeGeOPViewPkg

#### 4.13.1 Informazioni sul package

- **descrizione:** racchiude la componente principale della view;
- **padre:** [ViewPkg](#);
- **interazioni con altri package:**
  - IN MapComponentsPkg: utilizzo di componenti grafiche;
  - OUT MapComponentsPkg: utilizzo di componenti grafiche;
  - OUT SidebarPkg: utilizzo della sidebar.
- **classi contenute:**
  - DeGeOPView.

#### 4.13.2 Classi

##### 4.13.2.1 DeGeOPView

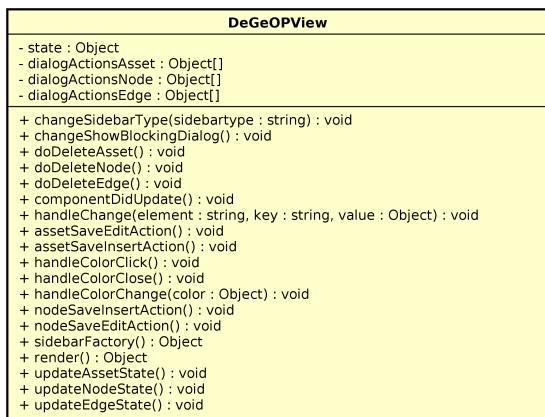


Figura 42: Diagramma classe DeGeOPView

- **descrizione:** rappresenta l'oggetto grafico radice, che comprende l'intera View del prodotto;
- **utilizzo:** i suoi metodi setter sono richiamati da ConcreteDeGeOPViewBuilder per impostare i suoi campi dati. La classe effettua il subscribe sullo Store per ricevere gli aggiornamenti ;
- **attributi:**
  - dialogActionsAsset : Object[]
    - \* contiene le funzioni richiamate durante la visualizzazione della schermata modale relativa agli asset.
  - dialogActionsEdge : Object[]
    - \* contiene le funzioni richiamate durante la visualizzazione della schermata modale relativa ai nodi.
  - dialogActionsNode : Object[]
    - \* contiene le funzioni richiamate durante la visualizzazione della schermata modale relativa ai nodi.
  - state : Object
    - \* rappresenta lo stato di DeGeOPView, ovvero i dati temporanei che l'utente inserisce prima che vengano inseriti nello store.
- **metodi:**
  - +assetSaveEditAction() : void
    - il metodo emette l'azione relativa alla modifica di un asset
  - +assetSaveInsertAction() : void
    - il metodo emette l'azione relativa all'inserimento di un asset
  - +changeShowBlockingDialog() : void
    - il metodo gestisce il cambiamento del booleano nello state di DeGeOPView che gestisce la visualizzazione della finestra modale

- `+changeSidebarType() : void`  
il metodo gestisce il cambiamento della stringa dello state di DeGeOPView che gestisce il tipo di sidebar visualizzata al momento
- `+componentDidUpdate() : void`  
il metodo viene richiamato quando lo state di DeGeOPView cambia; al suo interno vengono gestite le validazioni dei campi compilati dall'utente
- `+doDeleteAsset() : void`  
il metodo emette l'azione di eliminazione dell'asset attualmente selezionato e ne fa il dispatch verso lo store
- `+doDeleteEdge() : void`  
il metodo emette l'azione di eliminazione del nodo attualmente selezionato e ne fa il dispatch verso lo store
- `+doDeleteNode() : void`  
il metodo emette l'azione di eliminazione del nodo attualmente selezionato e ne fa il dispatch verso lo store
- `+handleChange(element, key, value) : void`  
il metodo gestisce il cambiamento di stato di DeGeOPView relativamente ai campi dati compilati dall'utente
  - \* `element : string`  
indica il campo dati dello stato di DeGeOPView da cambiare. Può assumere i valori: asset, node, edge, common.
  - \* `key : string`  
indica il campo dati dell'oggetto descritto da element che dovrà essere modificato.
  - \* `value : Object`  
indica il valore da inserire nell'oggetto descritto da element, nel campo dati descritto da key.
- `+handleColorChange(color) : void`  
il metodo gestisce il cambiamento nello state di DeGeOPView relativo al colore selezionato dalla palette colori
  - \* `color : Object`  
rappresenta il nuovo colore selezionato dalla palette colori.
- `+handleColorClick() : void`  
il metodo gestisce l'apertura e la chiusura della palette colori
- `+handleColorClose() : void`  
il metodo gestisce la chiusura della palette colori
- `+nodeSaveEditAction() : void`  
il metodo gestisce l'emissione dell'azione relativa alla modifica di un nodo
- `+nodeSaveInsertAction() : void`  
il metodo gestisce l'emissione dell'azione relativa all'inserimento di un nodo
- `+render() : Object`  
il metodo gestisce il render di DeGeOPView, composta da una sidebar e da un MapWrapper
- `+sidebarFactory() : Object`  
il metodo gestisce la creazione di una sidebar da renderizzare in base alla stringa attualmente specificata nello state di DeGeOPView

- `+updateAssetState() : void`  
il metodo imposta lo stato dell'asset in DeGeOPView con i dati dell'asset selezionato sulla mappa
- `+updateEdgeState() : void`  
il metodo imposta lo stato dell'arco in DeGeOPView con i dati dell'arco selezionato sulla mappa
- `+updateNodeState() : void`  
il metodo imposta lo stato del nodo in DeGeOPView con i dati del nodo selezionato sulla mappa

## 4.14 DeGeOP::ViewPkg::MapComponentsPkg

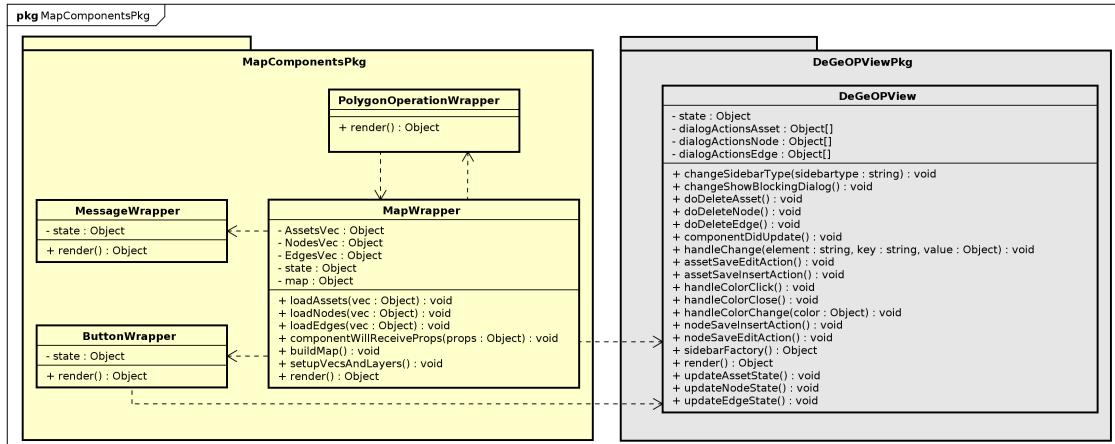


Figura 43: Schema componente DeGeOP::ViewPkg::MapComponentsPkg

### 4.14.1 Informazioni sul package

- descrizione:** racchiude le componenti relative alla mappa e ai pulsanti sopra di essa;
- padre:** [ViewPkg](#);
- interazioni con altri package:**
  - IN DeGeOPViewPkg: utilizzo di componenti grafiche;
  - OUT DeGeOPViewPkg: utilizzo di componenti grafiche;
  - OUT Openlayers: gestione mappa.
- classi contenute:**
  - ButtonWrapper;
  - MapWrapper;
  - MessageWrapper;
  - PolygonOperationWrapper.

### 4.14.2 Classi

#### 4.14.2.1 ButtonWrapper

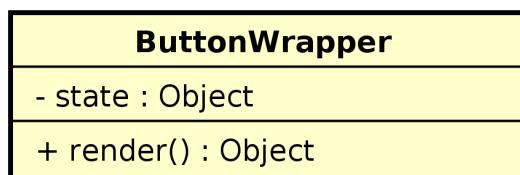


Figura 44: Diagramma classe ButtonWrapper

- descrizione:** rappresenta una classe wrapper per visualizzare una serie di pulsanti con cui è possibile eseguire varie operazioni;

- **utilizzo:** viene utilizzato per mostrare sulla mappa una serie di bottoni;
- **attributi:**
  - state : Object
    - \* rappresenta lo stato del ButtonWrapper.
- **relazioni con altre classi:**
  - IN MapWrapper.

#### 4.14.2.2 MapWrapper

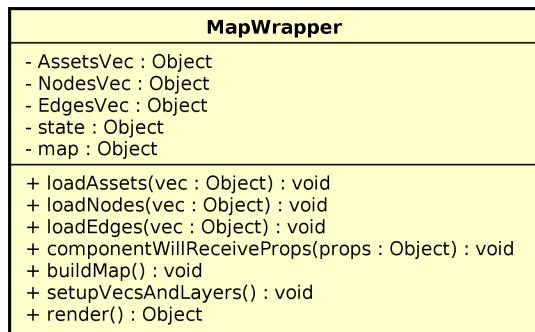


Figura 45: Diagramma classe MapWrapper

- **descrizione:** rappresenta una classe wrapper per visualizzare la mappa;
- **utilizzo:** viene utilizzata per visualizzare una mappa e permettere all'utente di interagire con essa;
- **attributi:**
  - assetsVec : Object
    - \* contiene gli oggetti geometrici relativi agli asset da disegnare sulla mappa.
  - edgesVec : Object
    - \* contiene gli oggetti geometrici relativi agli asset da disegnare sulla mappa.
  - map : Object
    - \* oggetto che contiene lo stato della mappa.
  - nodesVec : Object
    - \* contiene gli oggetti geometrici relativi ai nodi da disegnare sulla mappa.
  - state : Object
    - \* contiene lo stato della classe MapWrapper.
- **metodi:**
  - +buildMap() : void
    - il metodo costruisce la mappa

- `+componentWillReceiveProps(props) : void`  
il metodo viene richiamato quando la componente MapWrapper sta per ricevere nuovi props
  - \* `props : Object`  
props da ricevere.
- `+loadAssets(vec) : void`  
il metodo carica il vettore degli asset
  - \* `vec : Object`  
vettore degli asset da caricare.
- `+loadEdges(vec) : void`  
il metodo carica il vettore degli archi
  - \* `vec : Object`  
vettore degli archi da caricare.
- `+loadNodes(vec) : void`  
il metodo carica il vettore dei nodi
  - \* `vec : Object`  
vettore dei nodi da caricare.
- `+render() : Object`  
renderizza il MapWrapper
- `+setupVecsAndLayers() : void`  
il metodo prepara i vettori e i layers per la mappa

- **relazioni con altre classi:**

- IN PolygonOperationWrapper;
- OUT ButtonWrapper;
- OUT MessageWrapper;
- OUT PolygonOperationWrapper.

#### 4.14.2.3 MessageWrapper

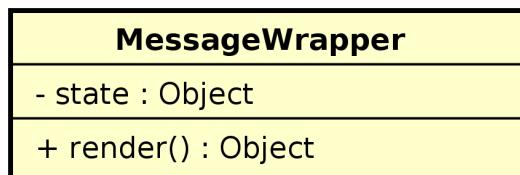


Figura 46: Diagramma classe MessageWrapper

- **descrizione:** rappresenta una classe wrapper per visualizzare un messaggio;
- **utilizzo:** viene utilizzato per mostrare messaggi di errore o di successo sulla mappa;
- **attributi:**
  - state : Object

\* rappresenta lo stato del MessageWrapper.

- **metodi:**

- +render() : Object  
renderizza il MessageWrapper

- **relazioni con altre classi:**

- IN MapWrapper.

#### 4.14.2.4 PolygonOperationWrapper

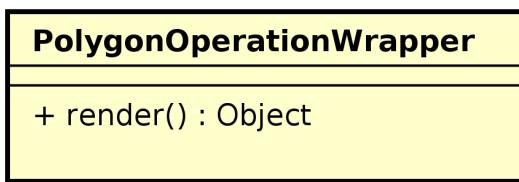


Figura 47: Diagramma classe PolygonOperationWrapper

- **descrizione:** rappresenta una classe wrapper per visualizzare un bottone con cui è possibile effettuare operazioni sul perimetro di un poligono sulla mappa;
- **utilizzo:** invocando i suoi metodi è possibile iniziare a disegnare il poligono su mappa oppure cancellare l'ultimo segmento disegnato;
- **metodi:**
  - +render() : Object  
renderizza il PolygonOperationWrapper
- **relazioni con altre classi:**
  - IN MapWrapper;
  - OUT MapWrapper.

## 4.15 DeGeOP::ViewPkg::SidebarPkg

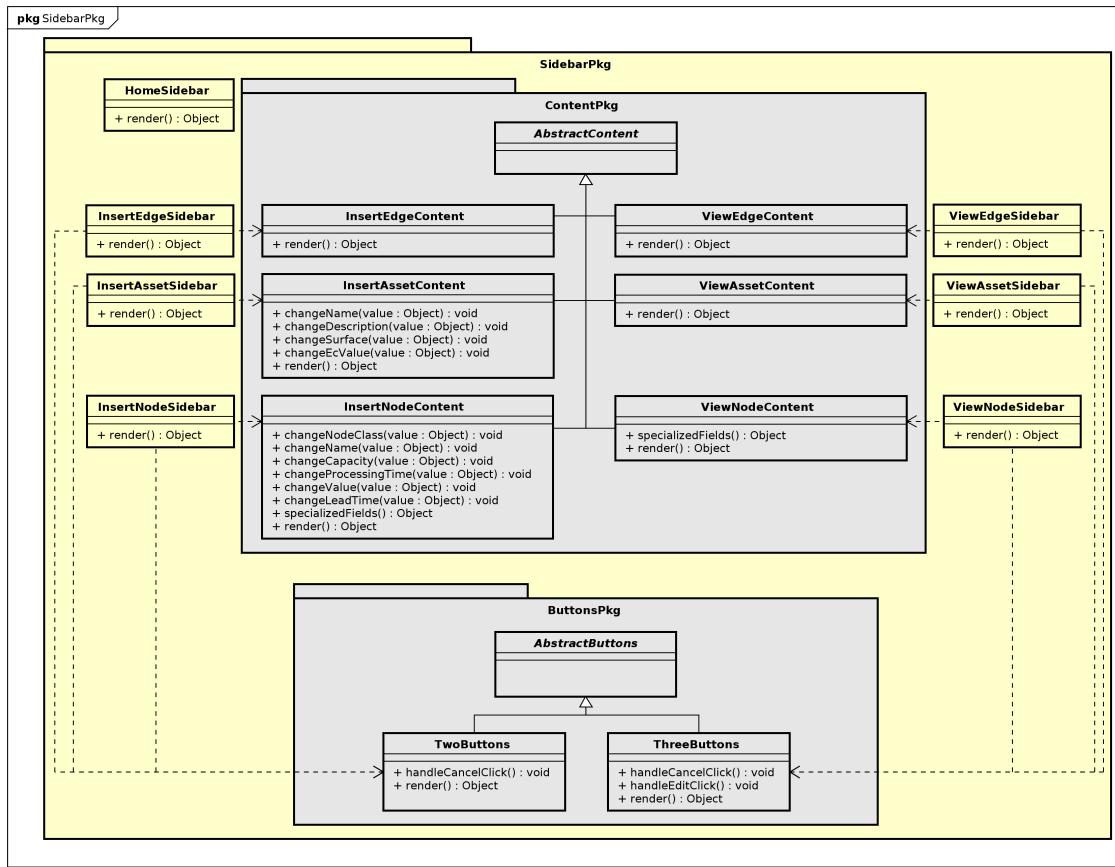


Figura 48: Schema componente DeGeOP::ViewPkg::SidebarPkg

### 4.15.1 Informazioni sul package

- **descrizione:** racchiude le componenti necessarie alla rappresentazione della sidebar;
- **padre:** [ViewPkg](#);
- **package contenuti:**
  - SidebarPkg:[ButtonsPkg](#);
  - SidebarPkg:[ContentPkg](#).
- **interazioni con altri package:**
  - IN DeGeOPViewPkg: utilizzo della sidebar.
- **classi contenute:**
  - HomeSidebar;
  - InsertAssetSidebar;
  - InsertEdgeSidebar;
  - InsertNodeSidebar;

- ViewAssetSidebar;
- ViewEdgeSidebar;
- ViewNodeSidebar.

#### 4.15.2 Classi

##### 4.15.2.1 HomeSidebar

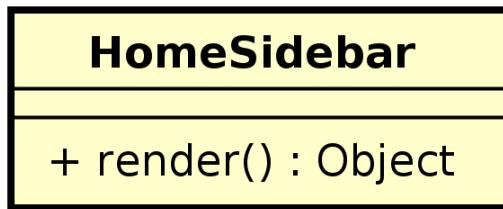


Figura 49: Diagramma classe HomeSidebar

- **descrizione:** rappresenta la sidebar di default ;
- **utilizzo:** renderizza una sidebar di default che dà il benvenuto all'utente;
- **metodi:**
  - +render() : Object  
il metodo renderizza la sidebar di benvenuto

##### 4.15.2.2 InsertAssetSidebar

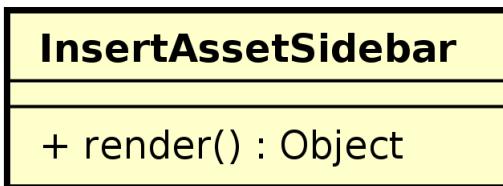


Figura 50: Diagramma classe InsertAssetSidebar

- **descrizione:** rappresenta una sidebar per l'inserimento e la modifica di un asset;
- **utilizzo:** renderizza una sidebar composta da contenuto in cui l'utente può compilare i dati e da buttoni che permettono di eseguire inserimenti e modifiche relative ad un asset;
- **metodi:**
  - +render() : Object  
il metodo renderizza la sidebar di inserimento e modifica di un asset, composta da un InsertAssetContent e da un TwoButtons

#### 4.15.2.3 InsertEdgeSidebar

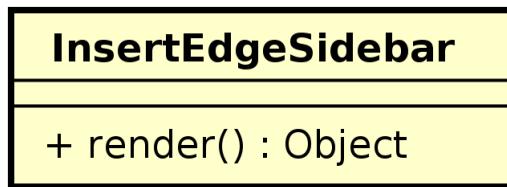


Figura 51: Diagramma classe InsertEdgeSidebar

- **descrizione:** rappresenta una sidebar per l'inserimento e la modifica di un arco;
- **utilizzo:** renderizza una sidebar composta da contenuto in cui l'utente può compilare i dati e da buttoni che permettono di eseguire inserimenti e modifiche relative ad un arco;
- **metodi:**
  - `+render() : Object`  
il metodo renderizza la sidebar di inserimento e modifica di un arco, composta da un `InsertEdgeContent` e da un `TwoButtons`

#### 4.15.2.4 InsertNodeSidebar

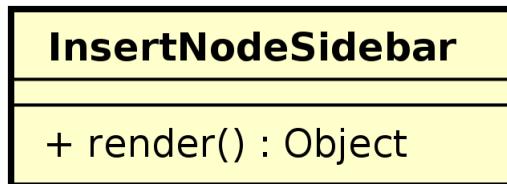


Figura 52: Diagramma classe InsertNodeSidebar

- **descrizione:** rappresenta una sidebar per l'inserimento e la modifica di un nodo;
- **utilizzo:** renderizza una sidebar composta da contenuto in cui l'utente può compilare i dati e da buttoni che permettono di eseguire inserimenti e modifiche relative ad un nodo;
- **metodi:**
  - `+render() : Object`  
il metodo renderizza la sidebar di inserimento e modifica di un nodo, composta da un `InsertNodeContent` e da un `TwoButtons`

#### 4.15.2.5 ViewAssetSidebar

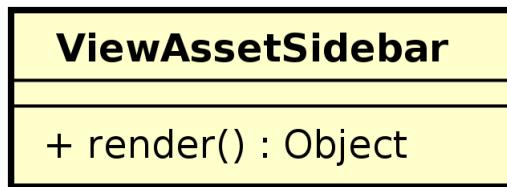


Figura 53: Diagramma classe ViewAssetSidebar

- **descrizione:** rappresenta una sidebar per la visualizzazione di un asset;
- **utilizzo:** renderizza una sidebar composta da contenuto in cui l'utente può visualizzare i dati e da buttoni che permettono di effettuare l'eliminazione dell'asset;
- **metodi:**
  - **+render() : Object**  
il metodo renderizza una sidebar per la visualizzazione di un asset, composta da un ViewAssetContent e da un ThreeButtons

#### 4.15.2.6 ViewEdgeSidebar

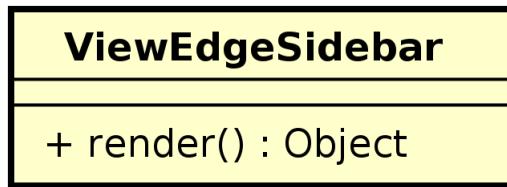


Figura 54: Diagramma classe ViewEdgeSidebar

- **descrizione:** rappresenta una sidebar per l'inserimento e la modifica di un arco;
- **utilizzo:** renderizza una sidebar composta da contenuto in cui l'utente può visualizzare i dati e da buttoni che permettono di effettuare l'eliminazione dell'arco;
- **metodi:**
  - **+render() : Object**  
il metodo renderizza una sidebar per la visualizzazione di un arco, composta da un ViewEdgeContent e da un ThreeButtons

#### 4.15.2.7 ViewNodeSidebar

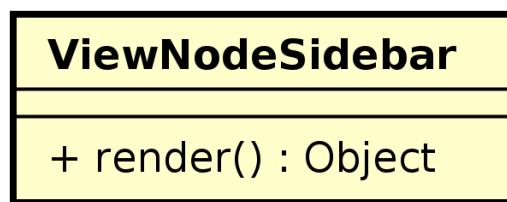


Figura 55: Diagramma classe ViewNodeSidebar

- **descrizione:** rappresenta una sidebar per la visualizzazione di un nodo;
- **utilizzo:** renderizza una sidebar composta da contenuto in cui l'utente può visualizzare i dati e da buttoni che permettono di effettuare l'eliminazione del nodo;
- **metodi:**
  - `+render() : Object`  
il metodo renderizza una sidebar per la visualizzazione di un arco, composta da un ViewNodeContent e da un ThreeButtons

## 4.16 DeGeOP::ViewPkg::SidebarPkg::ContentPkg

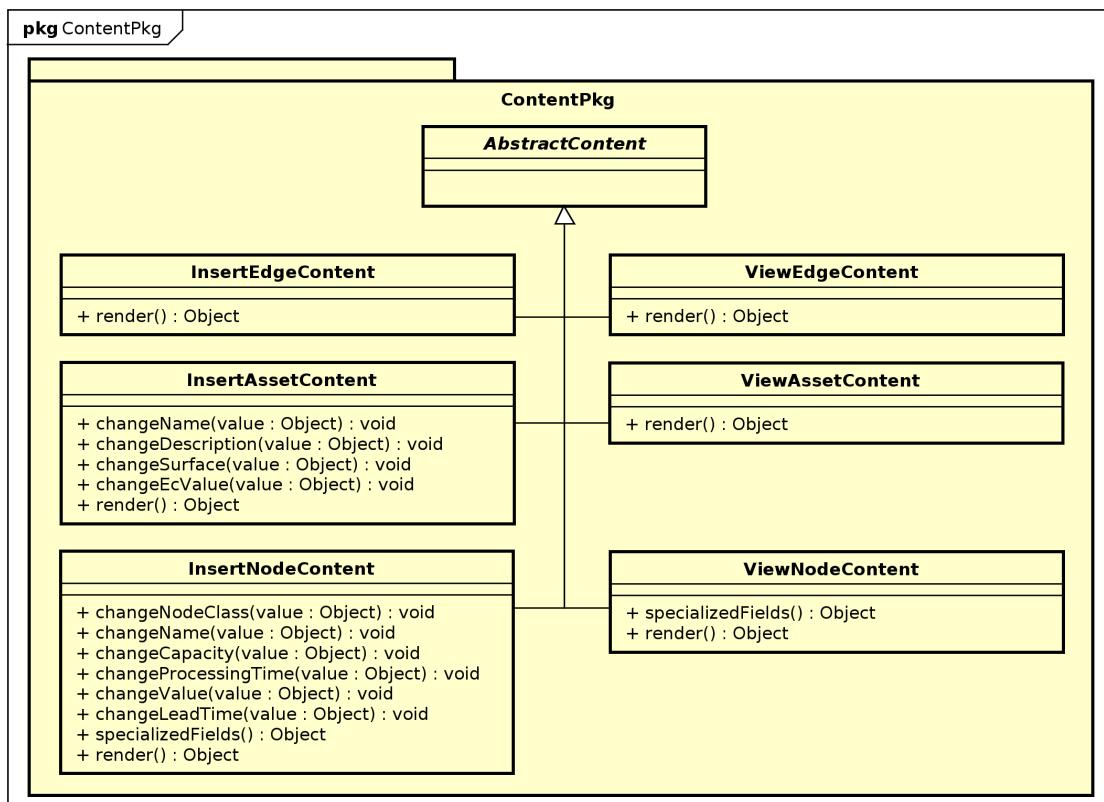


Figura 56: Schema componente DeGeOP::ViewPkg::SidebarPkg::ContentPkg

### 4.16.1 Informazioni sul package

- **descrizione:** racchiude le componenti che sono relative all'area informativa della Sidebar;
- **padre:** [SidebarPkg](#);
- **interazioni con altri package:**
  - IN FactorySidebarPkg: creazione contenuto della sidebar;
  - OUT React-color: visualizzazione palette colori.
- **classi contenute:**
  - **AbstractContent**;
  - **InsertAssetContent**;
  - **InsertEdgeContent**;
  - **InsertNodeContent**;
  - **ViewAssetContent**;
  - **ViewEdgeContent**;
  - **ViewNodeContent**.

#### 4.16.2 Classi

##### 4.16.2.1 AbstractContent



Figura 57: Diagramma classe AbstractContent

- **descrizione:** una classe d'interfaccia rappresentante il contenuto dell'area informativa nella sidebar;
- **utilizzo:** viene riferita da sidebar in quanto è una delle sue componenti.

##### 4.16.2.2 InsertAssetContent

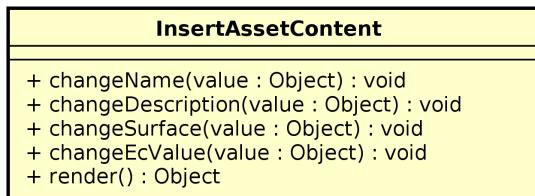


Figura 58: Diagramma classe InsertAssetContent

- **descrizione:** rappresenta il contenuto della sidebar relativa all'inserimento di un asset;
- **utilizzo:** viene creata da InsertAssetSidebarFactory ;
- **metodi:**
  - +changeDescription() : void  
delega il cambiamento dell'input della descrizione alla componente di livello più alto
  - +changeEcValue(value) : void  
delega il cambiamento dell'input del valore economico dell'asset alla componente di livello più alto
    - \* value : string  
valore attualmente contenuto nell'input del valore economico dell'asset.
  - +changeName(value) : void  
delega il cambiamento dell'input del nome dell'asset alla componente di livello più alto
    - \* value : Object  
valore attualmente contenuto nell'input del nome dell'asset.
  - +changeSurface() : void  
delega il cambiamento dell'input della superficie dell'asset alla componente di livello più alto

- `+render() : Object`  
renderizza il contenuto della sidebar dell'asset in modalità inserimento e modifica

#### 4.16.2.3 InsertEdgeContent

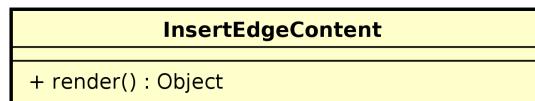


Figura 59: Diagramma classe InsertEdgeContent

- **descrizione:** rappresenta il contenuto della sidebar relativa all'inserimento di un arco;
- **utilizzo:** viene creata da `InsertEdgeSidebarFactory` ;
- **metodi:**
  - `+render() : Object`  
renderizza il contenuto della sidebar dell'arco in modalità inserimento e modifica

#### 4.16.2.4 InsertNodeContent

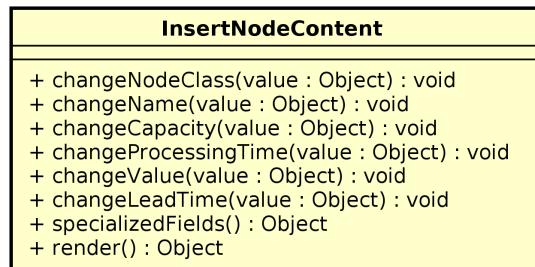


Figura 60: Diagramma classe InsertNodeContent

- **descrizione:** rappresenta il contenuto della sidebar relativa all'inserimento di un nodo;
- **utilizzo:** viene creata da `InsertNodeSidebarFactory` ;
- **metodi:**
  - `+changeCapacity() : void`  
delega il cambiamento della capacità del nodo alla componente di più alto livello
  - `+changeLeadTime(value) : void`  
delega il cambiamento del tempo di approvvigionamento del nodo alla componente di più alto livello
    - \* `value : string`  
valore che è contenuto nel dropdown della classe del nodo.
  - `+changeName(value) : void`  
delega il cambiamento del nome del nodo alla componente di più alto livello

- \* value : string  
valore che è contenuto nel dropdown della classe del nodo.
- +changeNodeClass(value) : void  
delega il cambiamento del dropdown del nome del nodo alla componente di più alto livello
  - \* value : string  
valore che è contenuto nel dropdown della classe del node.
- +changeProcessingTime(value) : void  
delega il cambiamento del valore del nodo alla componente di più alto livello
  - \* value : string  
valore che è contenuto nel dropdown della classe del nodo.
- +render() : Object  
renderizza il contenuto della sidebar del node in modalità inserimento e modifica
- +specializedFields() : Object  
metodo che gestisce i campi dati da visualizzare a seconda della tipologia del nodo

#### 4.16.2.5 ViewAssetContent



Figura 61: Diagramma classe ViewAssetContent

- **descrizione:** rappresenta il contenuto della sidebar relativa alla visualizzazione di un asset;
- **utilizzo:** viene creata da ViewAssetSidebarFactory ;
- **metodi:**
  - +render() : Object  
renderizza la parte di sidebar relativa alla visualizzazione dei campi dati di un asset

#### 4.16.2.6 ViewEdgeContent

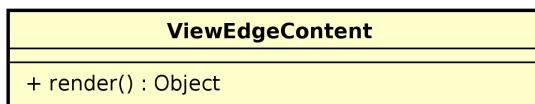


Figura 62: Diagramma classe ViewEdgeContent

- **descrizione:** rappresenta il contenuto della sidebar relativa alla visualizzazione di un arco;
- **utilizzo:** viene creata da ViewEdgeSidebarFactory ;
- **metodi:**

- `+render() : Object`  
renderizza la parte di sidebar relativa alla visualizzazione dei campi dati di un arco
- `+specializedFields() : Object`  
si occupa della visualizzazione dei campi dati a seconda della tipologia del nodo

#### 4.16.2.7 ViewNodeContent

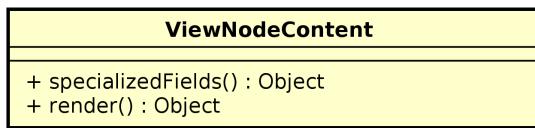


Figura 63: Diagramma classe ViewNodeContent

- **descrizione:** rappresenta il contenuto della sidebar relativa alla visualizzazione di un nodo;
- **utilizzo:** viene creata da `ViewNodeSidebarFactory` ;
- **metodi:**
  - `+render() : Object`  
renderizza la parte di sidebar relativa alla visualizzazione dei campi dati di un nodo

#### 4.17 DeGeOP::ViewPkg::SidebarPkg::ButtonsPkg

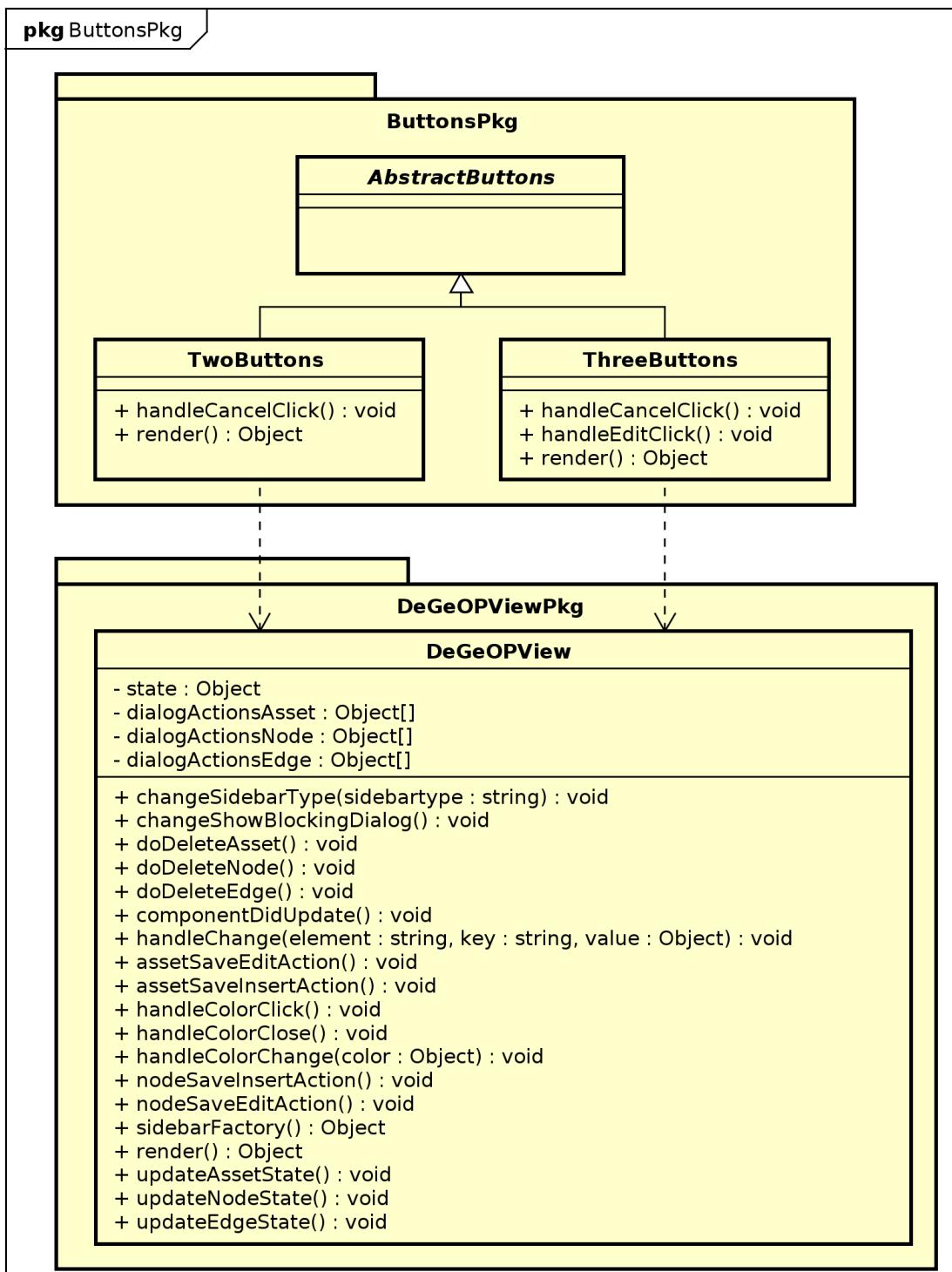


Figura 64: Schema componente DeGeOP::ViewPkg::SidebarPkg::ButtonsPkg

#### 4.17.1 Informazioni sul package

- **descrizione:** racchiude le componenti che sono relative all'area con i bottoni della Sidebar;
- **padre:** [SidebarPkg](#);
- **classi contenute:**
  - AbstractButtons;
  - ThreeButtons;
  - TwoButtons.

#### 4.17.2 Classi

##### 4.17.2.1 AbstractButtons

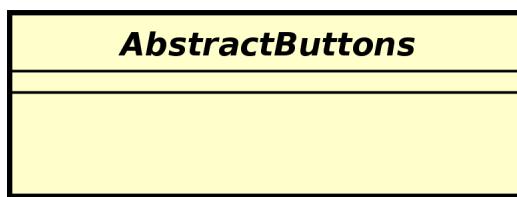


Figura 65: Diagramma classe AbstractButtons

- **descrizione:** una classe d'interfaccia rappresentante i bottoni inseriti nella sidebar;
- **utilizzo:** viene riferita da sidebar in quanto è una delle sue componenti.

##### 4.17.2.2 ThreeButtons

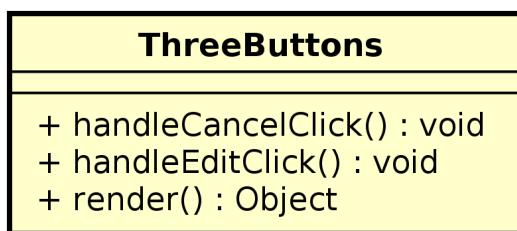


Figura 66: Diagramma classe ThreeButtons

- **descrizione:** classe che renderizza tre bottoni, uno di modifica, uno di eliminazione e uno di annullamento. Il bottone di annullamento provoca l'annullamento della selezione corrente. Il bottone di eliminazione provoca la comparsa di una finestra modale che chiede la conferma dell'eliminazione dell'elemento correntemente selezionato. Il bottone di modifica provoca l'avvio della modalità di modifica, con cui è possibile ridisegnare l'asset o cambiare i campi dati;
- **utilizzo:** viene utilizzata nelle sidebar di visualizzazione;

- **metodi:**

- `+handleCancelClick() : void`  
il metodo gestisce il click sul bottone di annullamento, ripristinando la sidebar di default
- `+handleEditClick() : void`  
il metodo imposta la sidebar di modifica dell'elemento attualmente selezionato
- `+render() : Object`  
renderizza tre buttoni, uno di modifica, uno di annullamento e uno di eliminazione

#### 4.17.2.3 TwoButtons

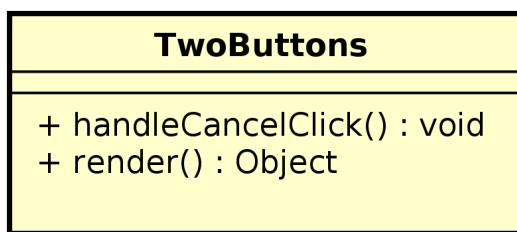


Figura 67: Diagramma classe TwoButtons

- **descrizione:** classe che renderizza due buttoni, uno di salvataggio e uno di annullamento. Il bottone di salvataggio è disabilitato fino a che tutti i campi della sidebar non sono stati compilati in modo corretto. Il bottone di annullamento provoca l'interruzione dell'inserimento dei dati;
- **utilizzo:** viene utilizzata nelle sidebar di inserimento;
- **metodi:**
  - `+handleCancelClick() : void`  
il metodo gestisce il click sul bottone di annullamento, ripristinando la sidebar di default
  - `+render() : Object`  
renderizza due buttoni, uno di annullamento e uno di salvataggio

## 5 Tracciamento

### 5.1 Tracciamento classi-requisiti

Classe	Requisito
Asset	ROF1 ROF1.3 ROF1.4 ROF1.5 ROF2 ROF4 ROF4.4 ROF5
AssetAction	ROF1 ROF1.5 ROF4 ROF4.4 ROF5
AssetActionCreator	ROF1 ROF1.5 ROF4 ROF4.4 ROF5 ROF5.1 ROF6.10
AssetReducer	ROF1 ROF1.5 ROF4 ROF4.4 ROF5 ROF5.1 ROF6.10

Classe	Requisito
ButtonWrapper	RFF16 RFF19.1.9 ROF1 ROF1.3 ROF1.4 ROF10 ROF11 ROF14 ROF15 ROF4 ROF4.3 ROF5 ROF6 ROF9 ROF9.2 ROF9.9
ConcretePolygon	ROF1.3 ROF1.4
ConcretePolygonFactory	RFF16 RFF19 ROF1 ROF4
Coordinate	ROF1.3 ROF1.4
Customer	RFF16 RFF20 RFF47 RFF48 RFF48.1 ROF1 ROF1.3 ROF1.4 ROF1.5 ROF10 ROF11 ROF14 ROF15 ROF4 ROF5 ROF6 ROF9

Classe	Requisito
DataToServer	RFF16
	RFF19
	RFF20
	RFF21
	ROF1
	ROF1.5
	ROF10
	ROF11
	ROF14
	ROF15
	ROF4
	ROF5
	ROF6
	ROF9

Classe	Requisito
DeGeOPView	RFF16 RFF19.4 RFF21 RFF24.4 RFF25 RFF25.1 RFF26 RFF26.1 RFF47 RFF48 ROF1 ROF1.3 ROF1.4 ROF1.5 ROF10 ROF11 ROF14 ROF15 ROF24 ROF24.1 ROF24.2 ROF24.3 ROF39 ROF4 ROF4.1.1 ROF4.1.2 ROF4.1.3 ROF4.2 ROF4.3 ROF4.5 ROF40 ROF41 ROF5 ROF6 ROF6.11 ROF9 ROF9.2 ROF9.9
Edge	ROF1.3 ROF1.4 ROF10 ROF11 ROF12 ROF14 ROF15 ROF5

Classe	Requisito
EdgeAction	ROF11 ROF14 ROF15
EdgeActionCreator	RFF14.5 ROF11 ROF14 ROF14.7 ROF15 ROF15.1
EdgeReducer	RFF14.5 ROF11 ROF11.5 ROF14 ROF14.1 ROF14.2 ROF15 ROF15.1
ExitNode	ROF1.3 ROF1.4 ROF10 ROF5

Classe	Requisito
InsertAssetContent	ROF1.2 ROF1.2.1 ROF1.2.1.1 ROF1.2.2 ROF1.2.2.1 ROF1.2.3 ROF1.2.3.1 ROF1.2.3.2 ROF1.2.3.3 ROF1.2.3.4 ROF1.2.3.5 ROF1.2.3.6 ROF1.2.4 ROF1.2.4.1 ROF1.2.4.2 ROF1.2.4.3 ROF1.2.4.4 ROF1.2.5 ROF1.2.5.1 ROF1.2.5.2 ROF1.2.6 ROF1.2.6.1 ROF1.2.7 ROF1.2.7.1 ROF1.6
InsertEdgeContent	RFF11.2 RFF11.3 RFF11.3.1 RFF11.3.1.1 RFF11.3.2 RFF11.6 RFF11.7 ROF1.6 ROF11.3.2.1 ROF11.4

Classe	Requisito
InsertNodeContent	ROF6.1 ROF6.12 ROF6.12.1 ROF6.12.1.1 ROF6.3 ROF6.3.1 ROF6.3.2 ROF6.3.3 ROF6.3.4 ROF6.3.5 ROF6.3.6 ROF6.4 ROF6.5 ROF6.5.2 ROF6.5.2.1 ROF6.5.3 ROF6.5.3.1 ROF6.5.4 ROF6.5.4.1 ROF6.6 ROF6.6.2 ROF6.6.2.1 ROF6.7 ROF6.7.2 ROF6.7.2.1 ROF6.8 ROF9.1
MachineNode	ROF1.3 ROF1.4 ROF10 ROF5

Classe	Requisito
MapWrapper	RFF16 RFF16.1.10 RFF16.1.7 RFF16.1.7.1 RFF16.1.8 RFF16.1.8.1 RFF16.1.9 RFF16.2 RFF16.3 RFF19.1.10 RFF19.1.7 RFF19.1.7.1 RFF19.1.8 RFF19.1.8.1 RFF19.1.9 RFF24.4 ROF1 ROF1.1 ROF1.1.1 ROF1.1.2 ROF1.1.3 ROF1.3 ROF1.5 ROF10 ROF11 ROF11.1 ROF11.1.1 ROF11.1.2 ROF14 ROF15 ROF24 ROF24.1 ROF24.2 ROF24.3 ROF4 ROF4.1 ROF4.3 ROF42 ROF43 ROF44 ROF5 ROF6 ROF6.1 ROF6.2 ROF9 ROF9.1 ROF9.1.1 ROF9.1.3 ROF9.2
Zephyrus - DeGeOP	ROF9.9

Classe	Requisito
MessageWrapper	RFF14.8 RFF16 RFF16.4 RFF19.4 RFF20 RFF22 RFF23 RFF47 RFF48 ROF1 ROF1.3 ROF1.5 ROF1.6 ROF10 ROF11 ROF14 ROF15 ROF15.1 ROF15.2 ROF20.1 ROF4 ROF4.3 ROF4.5 ROF5 ROF6 ROF6.11 ROF9 ROF9.11 ROF9.2 ROF9.9
Node	ROF1.3 ROF1.4 ROF10 ROF6 ROF7 ROF9 ROF9.10
NodeAction	ROF10 ROF6 ROF9 ROF9.10
NodeActionCreator	ROF10 ROF10.1 ROF6 ROF9 ROF9.10

Classe	Requisito
NodeReducer	ROF10 ROF10.1 ROF6 ROF9 ROF9.10
Polygon	RFF16 RFF19 ROF1 ROF1.5 ROF10 ROF15 ROF4 ROF5
PolygonFactory	RFF19
PolygonOperationWrapper	RFF16 RFF16.1.10 RFF16.1.7.1 RFF16.1.8 RFF16.1.8.1 RFF16.1.9 RFF19.1.7 RFF19.1.8 ROF1 ROF1.1 ROF1.2 ROF1.5 ROF10 ROF15 ROF20.2 ROF4 ROF4.1 ROF5
Process	ROF1.3 ROF1.4 ROF10 ROF15 ROF5
QueueNode	ROF1.3 ROF1.4 ROF10 ROF15 ROF5

Classe	Requisito
Reducer	RFF14.5 RFF16 ROF1 ROF1.5 ROF10 ROF10.1 ROF11 ROF11.5 ROF14 ROF14.1 ROF14.2 ROF15 ROF15.1 ROF4 ROF5 ROF5.1 ROF6 ROF6.10 ROF9
ResourceNode	ROF1.3 ROF1.4 ROF10 ROF15 ROF5
Sidebar	RFF16 ROF1 ROF1.3 ROF1.4 ROF1.5 ROF10 ROF11 ROF14 ROF15 ROF4 ROF4.3 ROF5 ROF6 ROF9 ROF9.2 ROF9.9
SourceNode	ROF1.3 ROF1.4 ROF10 ROF15 ROF5

Classe	Requisito
StoreDeGeOP	RFF20 RFF25 RFF25.1 RFF26 RFF26.1 RFF47 RFF48 RFF48.1 ROF1.3 ROF1.4 ROF10 ROF10.1 ROF11.5 ROF14.1 ROF14.2 ROF15 ROF5 ROF5.1 ROF6.10
ViewAssetContent	ROF2 ROF3 ROF42 ROF5.2 ROF6.10
ViewEdgeContent	ROF12 ROF13 ROF44
ViewNodeContent	ROF10.2 ROF43 ROF7 ROF8

Tabella 2: Tracciamento classe-requisiti

## 5.2 Tracciamento requisiti-classi

Requisito	Classe
RFF11.2	InsertEdgeContent
RFF11.3	InsertEdgeContent
RFF11.3.1	InsertEdgeContent
RFF11.3.1.1	InsertEdgeContent
RFF11.3.2	InsertEdgeContent

Requisito	Classe
RFF11.6	InsertEdgeContent
RFF11.7	InsertEdgeContent
RFF14.4	
RFF14.5	EdgeActionCreator EdgeReducer Reducer
RFF14.5.1	
RFF14.5.1.1	
RFF14.5.2	
RFF14.5.2.1	
RFF14.8	MessageWrapper
RFF16	ButtonWrapper ConcretePolygonFactory Customer DataToServer DeGeOPView MapWrapper MessageWrapper Polygon PolygonOperationWrapper Reducer Sidebar
RFF16.1	
RFF16.1.1	
RFF16.1.1.1	
RFF16.1.10	MapWrapper PolygonOperationWrapper
RFF16.1.11	
RFF16.1.2	
RFF16.1.2.1	
RFF16.1.3	
RFF16.1.3.1	
RFF16.1.3.10	
RFF16.1.3.11	
RFF16.1.3.2	
RFF16.1.3.3	

Requisito	Classe
RFF16.1.3.4	
RFF16.1.3.5	
RFF16.1.3.6	
RFF16.1.3.7	
RFF16.1.3.8	
RFF16.1.3.9	
RFF16.1.4	
RFF16.1.4.1	
RFF16.1.5	
RFF16.1.5.1	
RFF16.1.6	
RFF16.1.6.1	
RFF16.1.7	MapWrapper
RFF16.1.7.1	MapWrapper PolygonOperationWrapper
RFF16.1.8	MapWrapper PolygonOperationWrapper
RFF16.1.8.1	MapWrapper PolygonOperationWrapper
RFF16.1.9	MapWrapper PolygonOperationWrapper
RFF16.2	MapWrapper
RFF16.3	MapWrapper
RFF16.4	MessageWrapper
RFF17	
RFF18	
RFF19	ConcretePolygonFactory DataToServer Polygon PolygonFactory
RFF19.1	
RFF19.1.1	
RFF19.1.1.1	
RFF19.1.10	MapWrapper

Requisito	Classe
RFF19.1.11	
RFF19.1.2	
RFF19.1.2.1	
RFF19.1.3	
RFF19.1.3.11	
RFF19.1.4	
RFF19.1.4.1	
RFF19.1.5	
RFF19.1.5.1	
RFF19.1.6	
RFF19.1.6.1	
RFF19.1.7	MapWrapper PolygonOperationWrapper
RFF19.1.7.1	MapWrapper
RFF19.1.8	MapWrapper PolygonOperationWrapper
RFF19.1.8.1	MapWrapper
RFF19.1.9	ButtonWrapper MapWrapper
RFF19.2	
RFF19.3	
RFF19.4	DeGeOPView MessageWrapper
RFF20	Customer DataToServer MessageWrapper StoreDeGeOP
RFF21	DataToServer DeGeOPView
RFF22	MessageWrapper
RFF23	MessageWrapper
RFF24.4	DeGeOPView MapWrapper
RFF25	DeGeOPView StoreDeGeOP

Requisito	Classe
RFF25.1	DeGeOPView StoreDeGeOP
RFF26	DeGeOPView StoreDeGeOP
RFF26.1	DeGeOPView StoreDeGeOP
RFF45	
RFF46	
RFF47	Customer DeGeOPView MessageWrapper StoreDeGeOP
RFF48	Customer DeGeOPView MessageWrapper StoreDeGeOP
RFF48.1	Customer StoreDeGeOP
ROF1	Asset AssetAction AssetActionCreator AssetReducer ButtonWrapper ConcretePolygonFactory Customer DataToServer DeGeOPView MapWrapper MessageWrapper Polygon PolygonOperationWrapper Reducer Sidebar
ROF1.1	MapWrapper PolygonOperationWrapper
ROF1.1.1	MapWrapper
ROF1.1.2	MapWrapper PolygonOperationWrapper
ROF1.1.3	MapWrapper
ROF1.2	InsertAssetContent

Requisito	Classe
ROF1.2.1	InsertAssetContent
ROF1.2.1.1	InsertAssetContent
ROF1.2.2	InsertAssetContent
ROF1.2.2.1	InsertAssetContent
ROF1.2.3	InsertAssetContent
ROF1.2.3.1	InsertAssetContent
ROF1.2.3.2	InsertAssetContent
ROF1.2.3.3	InsertAssetContent
ROF1.2.3.4	InsertAssetContent
ROF1.2.3.5	InsertAssetContent
ROF1.2.3.6	InsertAssetContent
ROF1.2.4	InsertAssetContent
ROF1.2.4.1	InsertAssetContent
ROF1.2.4.2	InsertAssetContent
ROF1.2.4.3	InsertAssetContent
ROF1.2.4.4	InsertAssetContent
ROF1.2.5	InsertAssetContent
ROF1.2.5.1	InsertAssetContent
ROF1.2.5.2	InsertAssetContent
ROF1.2.6	InsertAssetContent
ROF1.2.6.1	InsertAssetContent
ROF1.2.7	InsertAssetContent
ROF1.2.7.1	InsertAssetContent

Requisito	Classe
ROF1.3	Asset ButtonWrapper ConcretePolygon Coordinate Customer DeGeOPView Edge ExitNode MachineNode MapWrapper MessageWrapper Node Process QueueNode ResourceNode Sidebar SourceNode StoreDeGeOP
ROF1.4	Asset ButtonWrapper ConcretePolygon Coordinate Customer DeGeOPView Edge ExitNode MachineNode Node Process QueueNode ResourceNode Sidebar SourceNode StoreDeGeOP

Requisito	Classe
ROF1.5	Asset AssetAction AssetActionCreator AssetReducer Customer DataToServer DeGeOPView MapWrapper MessageWrapper Polygon PolygonOperationWrapper Reducer Sidebar
ROF1.6	InsertAssetContent InsertEdgeContent MessageWrapper
ROF10	ButtonWrapper Customer DataToServer DeGeOPView Edge ExitNode MachineNode MapWrapper MessageWrapper Node NodeAction NodeActionCreator NodeReducer Polygon PolygonOperationWrapper Process QueueNode Reducer ResourceNode Sidebar SourceNode StoreDeGeOP
ROF10.1	NodeActionCreator NodeReducer Reducer StoreDeGeOP
ROF10.2	ViewNodeContent

Requisito	Classe
ROF11	ButtonWrapper Customer DataToServer DeGeOPView Edge EdgeAction EdgeActionCreator EdgeReducer MapWrapper MessageWrapper Reducer Sidebar
ROF11.1	MapWrapper
ROF11.1.1	MapWrapper
ROF11.1.2	MapWrapper
ROF11.3.2.1	InsertEdgeContent
ROF11.4	InsertEdgeContent
ROF11.5	EdgeReducer Reducer StoreDeGeOP
ROF12	Edge ViewEdgeContent
ROF13	ViewEdgeContent
ROF14	ButtonWrapper Customer DataToServer DeGeOPView Edge EdgeAction EdgeActionCreator EdgeReducer MapWrapper MessageWrapper Reducer Sidebar
ROF14.1	EdgeReducer Reducer StoreDeGeOP
ROF14.2	EdgeReducer Reducer StoreDeGeOP

Requisito	Classe
ROF14.3	
ROF14.6	
ROF14.7	EdgeActionCreator
ROF15	ButtonWrapper Customer DataToServer DeGeOPView Edge EdgeAction EdgeActionCreator EdgeReducer MapWrapper MessageWrapper Polygon PolygonOperationWrapper Process QueueNode Reducer ResourceNode Sidebar SourceNode StoreDeGeOP
ROF15.1	EdgeActionCreator EdgeReducer MessageWrapper Reducer
ROF15.2	MessageWrapper
ROF2	Asset ViewAssetContent
ROF20.1	MessageWrapper
ROF20.2	PolygonOperationWrapper
ROF24	DeGeOPView MapWrapper
ROF24.1	DeGeOPView MapWrapper
ROF24.2	DeGeOPView MapWrapper
ROF24.3	DeGeOPView MapWrapper
ROF3	ViewAssetContent

Requisito	Classe
ROF39	DeGeOPView
ROF4	Asset AssetAction AssetActionCreator AssetReducer ButtonWrapper ConcretePolygonFactory Customer DataToServer DeGeOPView MapWrapper MessageWrapper Polygon PolygonOperationWrapper Reducer Sidebar
ROF4.1	MapWrapper PolygonOperationWrapper
ROF4.1.1	DeGeOPView
ROF4.1.2	DeGeOPView
ROF4.1.3	DeGeOPView
ROF4.2	DeGeOPView
ROF4.2.1	
ROF4.2.2	
ROF4.2.2.1	
ROF4.2.3	
ROF4.2.3.6	
ROF4.2.4	
ROF4.2.4.4	
ROF4.2.5	
ROF4.2.5.2	
ROF4.2.6	
ROF4.2.6.1	
ROF4.2.7	
ROF4.2.7.1	

Requisito	Classe
ROF4.3	ButtonWrapper DeGeOPView MapWrapper MessageWrapper Sidebar
ROF4.4	Asset AssetAction AssetActionCreator AssetReducer
ROF4.5	DeGeOPView MessageWrapper
ROF40	DeGeOPView
ROF41	DeGeOPView
ROF42	MapWrapper ViewAssetContent
ROF43	MapWrapper ViewNodeContent
ROF44	MapWrapper ViewEdgeContent
ROF5	Asset AssetAction AssetActionCreator AssetReducer ButtonWrapper Customer DataToServer DeGeOPView Edge ExitNode MachineNode MapWrapper MessageWrapper Polygon PolygonOperationWrapper Process QueueNode Reducer ResourceNode Sidebar SourceNode StoreDeGeOP

Requisito	Classe
ROF5.1	AssetActionCreator AssetReducer Reducer StoreDeGeOP
ROF5.2	ViewAssetContent
ROF6	ButtonWrapper Customer DataToServer DeGeOPView MapWrapper MessageWrapper Node NodeAction NodeActionCreator NodeReducer Reducer Sidebar
ROF6.1	InsertNodeContent MapWrapper
ROF6.10	AssetActionCreator AssetReducer Reducer StoreDeGeOP ViewAssetContent
ROF6.11	DeGeOPView MessageWrapper
ROF6.12	InsertNodeContent
ROF6.12.1	InsertNodeContent
ROF6.12.1.1	InsertNodeContent
ROF6.2	MapWrapper
ROF6.3	InsertNodeContent
ROF6.3.1	InsertNodeContent
ROF6.3.2	InsertNodeContent
ROF6.3.3	InsertNodeContent
ROF6.3.4	InsertNodeContent
ROF6.3.5	InsertNodeContent
ROF6.3.6	InsertNodeContent
ROF6.4	InsertNodeContent

Requisito	Classe
ROF6.5	InsertNodeContent
ROF6.5.2	InsertNodeContent
ROF6.5.2.1	InsertNodeContent
ROF6.5.3	InsertNodeContent
ROF6.5.3.1	InsertNodeContent
ROF6.5.4	InsertNodeContent
ROF6.5.4.1	InsertNodeContent
ROF6.6	InsertNodeContent
ROF6.6.2	InsertNodeContent
ROF6.6.2.1	InsertNodeContent
ROF6.7	InsertNodeContent
ROF6.7.2	InsertNodeContent
ROF6.7.2.1	InsertNodeContent
ROF6.8	InsertNodeContent
ROF6.9	
ROF7	Node ViewNodeContent
ROF8	ViewNodeContent
ROF9	ButtonWrapper Customer DataToServer DeGeOPView MapWrapper MessageWrapper Node NodeAction NodeActionCreator NodeReducer Reducer Sidebar
ROF9.1	InsertNodeContent MapWrapper
ROF9.1.1	MapWrapper
ROF9.1.3	MapWrapper

Requisito	Classe
ROF9.10	Node NodeAction NodeActionCreator NodeReducer
ROF9.11	MessageWrapper
ROF9.12	
ROF9.12.1	
ROF9.12.1.1	
ROF9.2	ButtonWrapper DeGeOPView MapWrapper MessageWrapper Sidebar
ROF9.3	
ROF9.4	
ROF9.5	
ROF9.5.2	
ROF9.5.2.1	
ROF9.5.3	
ROF9.5.3.1	
ROF9.5.4	
ROF9.5.4.1	
ROF9.6	
ROF9.6.2	
ROF9.6.2.1	
ROF9.7	
ROF9.7.2	
ROF9.7.2.1	
ROF9.8	
ROF9.9	ButtonWrapper DeGeOPView MapWrapper MessageWrapper Sidebar

Tabella 3: Tracciamento requisito-classi

## A Descrizione design pattern

### A.1 Introduzione

I *design pattern* semplificano l'attività di progettazione, favorendo il riutilizzo del codice e rendendo l'architettura più manutenibile. I design pattern vengono definiti come soluzioni progettuali generali a problemi ricorrenti. Si tratta di una descrizione o dei modelli logici da applicare per la risoluzione di problemi che possono presentarsi durante la  $fase_G$  di progettazione. Esistono diversi design pattern e vengono suddivisi in base al problema da risolvere:

- **pattern creazionali:** nascondono i costruttori delle classi e espongono dei metodi al loro posto. In questo modo si possono utilizzare oggetti senza sapere come sono implementati;
- **pattern comportamentali:** forniscono soluzioni alle più comuni tipologie di interazione tra gli oggetti;
- **pattern architetturali:** operano ad un livello più elevato rispetto ad altri design pattern, ed esprimono schemi di base per impostare l'organizzazione strutturale di un sistema software. In questi schemi si descrivono sottosistemi predefiniti, i ruoli che essi assumono e le relazioni reciproche;
- **pattern strutturali:** consentono di riutilizzare degli oggetti esistenti fornendo agli utilizzatori un'interfaccia più adatta alle loro esigenze.

Sono stati utilizzati i seguenti design pattern:

- Factory Method (creazionale);
- Redux (architetturale).

Di seguito verranno mostrati i design pattern utilizzati e come vengono contestualizzati nel progetto *DeGeOP*. I diagrammi delle classi mostrati in queste comparazioni sono parziali e a scopo illustrativo. Per la loro visualizzazione completa si rimanda alla sezione ??.

## A.2 Pattern Creazionali

### A.2.1 Factory Method

#### A.2.1.1 Descrizione

Questo pattern permette di creare oggetti fornendo un'interfaccia per creare un oggetto, ma lascia che le sottoclassi decidano quale oggetto istanziare. Il pattern factory può essere utilizzato quando:

- la creazione di un oggetto preclude il suo riuso senza una significativa duplicazione di codice.
- la creazione di un oggetto richiede l'accesso ad informazioni o risorse che non dovrebbero essere contenute nella classe di composizione.
- la gestione del *ciclo di vita* degli oggetti gestiti deve essere centralizzata in modo da assicurare un comportamento coerente all'interno dell'applicazione.

#### A.2.1.2 Contestualizzazione

Questo pattern viene utilizzato nel package StorePkg::PolygonPkg. Sono state create le classi `ConcretePolygon`, che implementa l'interfaccia `Polygon`, e `ConcretePolygonFactory`, che implementa l'interfaccia `PolygonFactory`. `PolygonFactory` espone il metodo `CreatePolygon`, che viene sovrascritto dalla sua sottoclasse `ConcretePolygonFactory` consentendo quindi ai suoi utilizzatori, `Scenario` e `Asset`, la creazione dell'oggetto di tipo `Polygon`.

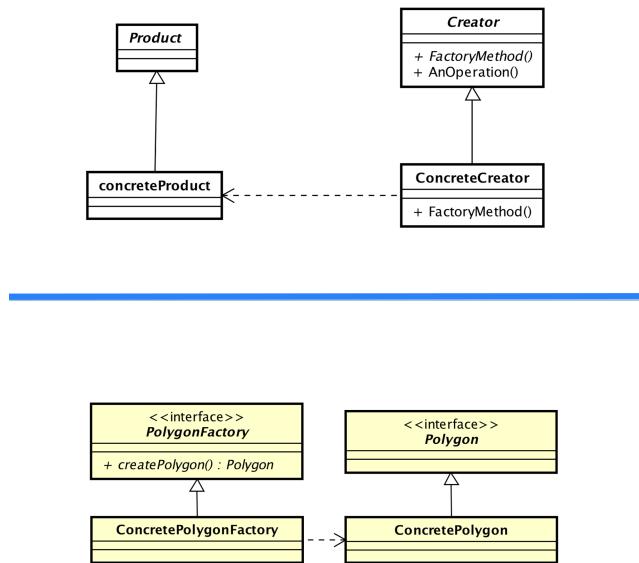


Figura 68: Factory Method e la sua contestualizzazione in DeGeOP

### A.3 Pattern Architetturali

#### A.3.1 Redux

##### A.3.1.1 Descrizione

Il pattern proposto dalla libreria Redux comprende 3 componenti:

- **Store**: è un unico oggetto globale read-only che contiene i dati da gestire, memorizzando l'intero stato del programma;
- **Actions**: definiscono le azioni che si occupano di aggiornare lo Store: l'unico modo di cambiare lo Store è emettere un'azione;
- **Reducer**: funzioni pure che, ricevuto un input uno stato e un'azione, restituiscono un nuovo stato modificato in base all'azione.

Dato uno Store e una funzione Reducer, lo stato del programma viene aggiornato in modo deterministico con i dati delle azioni. Il vero punto di forza di Redux è la gestione del flusso di dati in modo unidirezionale. Questo significa che tutti i dati dell'applicazione seguono lo stesso flusso, rendendo la logica dell'applicazione più predicable e facile da capire e implementare.

##### A.3.1.2 Contestualizzazione

Questo design pattern viene utilizzato per il design dell'architettura ad alto livello, come illustrato nella sezione [Descrizione architettura](#) e [Tecnologie utilizzate](#). L'intera applicazione nel suo insieme implementa questo design pattern.

Le classi [Action](#) all'interno del package ActionPkg e [Reducer](#) all'interno di ReducerPkg implementano rispettivamente le componenti Action e Reducer di questo design pattern.

L'unico modo per cambiare i dati contenuti all'interno del package StorePkg è emettere delle Actions.

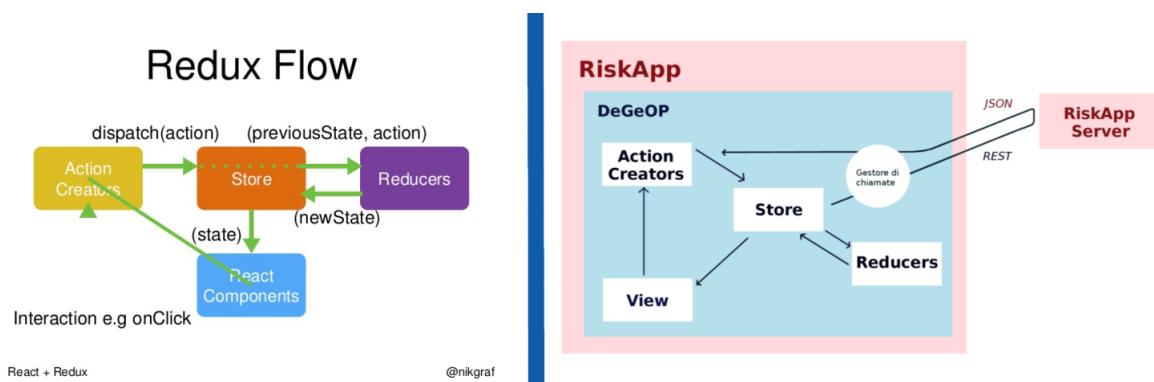


Figura 69: Design pattern proposto da Redux e la sua contestualizzazione in DeGeOP