

09-09-2015



Specifica Tecnica

Informazioni sul documento

Nome Documento	Specifica Tecnica
Versione	4.0.0
Stato	<i>Formale</i>
Uso	<i>Esterno</i>
Data Creazione	18-05-2015
Data Ultima Modifica	09-09-2015
Redazione	Venturelli Giovanni,Petrucci Mauro,Busetto Matteo,Tollot Pietro
Approvazione	Fossa Manuel
Verifica	Gabelli Pietro
Lista distribuzione	<i>LateButSafe</i> Prof. Tullio Vardanega Prof. Riccardo Cardin Proponente Zucchetti S.p.a.



Registro delle modifiche

Tab 1: Versionamento del documento

Versione	Autore	Data	Descrizione
4.0.0	Fossa Manuel	09-09-2015	Approvazione del documento
3.0.1	Gabelli Pietro	09-09-2015	Aggiunta dei contenuti: HomeOffline, ExecutionOffline, HomeOfflineController, OfflineExecutionController; tracciamenti
3.0.0	Busetto Matteo	09-09-2015	Aggiornati diagrammi di sequenza, CommandPackage; aggiunta appendice: Pattern
3.0.0	Venturelli Giovanni	09-09-2015	Rimossa sezione SVG
3.0.0	Venturelli Giovanni	08-09-2015	Aggiornato Invoker, abstractCommand
3.0.0	Tollot Pietro	08-09-2015	Divisione server/MVC; aggiornato NodeServer; aggiornati diagrammi backEnd
3.0.0	Gabelli Pietro	22-08-2015	Approvazione del documento
2.5.0	Gabelli Pietro	19-08-2015	Rimozione componenti di ApacheServer
2.4.0	Tollot Pietro	02-07-2015	Modifica schema backEndProgettazione
2.3.0	Tollot Pietro	27-06-2015	Aggiornamento schemi di Authenitcation, Loader, Register, accessControll, fileServerRelation, mongoRelation, nodeAPI, serverRelation; modifica capitolo Model::MongoRelations
2.2.0	Fossa Manuel	22-06-2015	Aggiornamento schema View; aggiornamento capitolo View::Pages
2.1.0	Gabelli Pietro	17-06-2015	Aggiornamento contenuti: architettura da MVP a MVC e Controller
2.0.0	Venturelli Giovanni	16-06-2015	Approvazione Documento



Questo documento è distribuito sotto licenza [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License](https://creativecommons.org/licenses/by-nc-sa/3.0/).


$$\mathbf{RR} \rightarrow \mathbf{RP}$$

Tab 2: Storico ruoli RR \rightarrow RP

Tab 3: Storico ruoli RP \rightarrow RQTab 4: Storico ruoli RQ \rightarrow RA

Indice

1	Introduzione	9
1.1	Scopo del documento	9
1.2	Scopo del Prodotto	9
1.3	Glossario	9
1.4	Riferimenti	9
1.4.1	Normativi	9
1.4.2	Informativi	9
2	Strumenti	11
2.1	HTML	11
2.2	JavaScript	11
2.3	jQuery	11
2.4	MEAN	12
2.4.1	MongoDB	12
2.4.2	Express.js	12
2.4.3	AngularJS	12
2.4.4	Node.js	12
2.5	Impress.js	12
3	Descrizione architetturale	13
3.1	Metodo e formalismi	13
3.2	Architettura generale	13
3.2.1	Model	13
3.2.2	View	13
3.2.3	Controller	13
3.2.4	nodeApi	14
4	Descrizione dei singoli componenti	15
4.1	Model	15
4.1.1	Model::SlideShow	15
4.1.2	Model::SlideShow::SlideShowActions	15
4.1.3	InsertEditRemove	16
4.1.4	Model::SlideShow::SlideShowActions::Command	19
4.1.4.1	Invoker	19
4.1.4.2	AbstractCommand	20
4.1.4.3	ConcreteTextInsertCommand	21
4.1.4.4	ConcreteFrameInsertCommand	22
4.1.4.5	ConcreteImageInsertCommand	22
4.1.4.6	ConcreteAudioInsertCommand	23
4.1.4.7	ConcreteVideoInsertCommand	23
4.1.4.8	ConcreteBackgroundInsertCommand	24
4.1.4.9	ConcreteTextRemoveCommand	24
4.1.4.10	ConcreteFrameRemoveCommand	25
4.1.4.11	ConcreteImageRemoveCommand	25

4.1.4.12	ConcreteAudioRemoveCommand	26
4.1.4.13	ConcreteVideoRemoveCommand	26
4.1.4.14	ConcreteEditSizeCommand	27
4.1.4.15	ConcreteEditPositionCommand	27
4.1.4.16	ConcreteEditColorCommand	28
4.1.4.17	ConcreteEditBackgroundCommand	28
4.1.4.18	ConcreteEditRotationCommand	29
4.1.4.19	ConcreteEditFontCommand	29
4.1.4.20	Classe ConcreteEditContentCommand	30
4.1.4.21	Classe ConcreteEditBookmarkCommand	30
4.1.4.22	Classe ConcretePortaAvantiCommand	31
4.1.4.23	Classe ConcretePortaDietroCommand	31
4.1.4.24	Classe ConcreteAddToMainPathCommand	32
4.1.4.25	Classe concreteRemoveFromMainPathCommand	32
4.1.4.26	Classe concreteNewChoicePathCommand	33
4.1.4.27	Classe concreteDeleteChoicePathCommand	33
4.1.4.28	Classe ConcreteRemoveFromChoicePathCommand	34
4.1.4.29	Classe ConcreteAddToChoicePathCommand	34
4.1.5	Model::SlideShow::SlideShowElements	35
4.1.5.1	SlideShowElement	35
4.1.5.2	Text	36
4.1.5.3	Frame	36
4.1.5.4	Image	37
4.1.5.5	Audio	37
4.1.5.6	Video	38
4.1.6	Background	38
4.1.7	Model::serverRelations	39
4.1.8	Model::serverRelations::accessControl	40
4.1.8.1	Authentication	40
4.1.8.2	Registration	41
4.1.9	Model::serverRelations:loader	41
4.1.9.1	Loader	41
4.1.9.2	FileServerRelation	41
4.1.9.3	MongoRelation	42
4.2	View	43
4.2.1	View::Pages	43
4.2.2	View::Pages::Index	43
4.2.3	View::Pages::Login	44
4.2.4	View::Pages::Registrazione	44
4.2.5	View::Pages::Home	44
4.2.6	View::Pages::Profile	44
4.2.7	View::Pages::Execution	45
4.2.8	View::Pages::Edit	45
4.2.9	View::Pages::HomeOffline	45
4.2.10	View::Pages::ExecutionOffline	45
4.3	Controller	46



Questo documento è distribuito sotto licenza [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License](https://creativecommons.org/licenses/by-nc-sa/3.0/).



1	Architettura generale del sistema	14
2	InsertEditRemove	16
3	Command Package	19
4	SlideShowElements	35
5	diagramma package Model::serverRelations	39
6	accessControl	40
7	serverRelation::Loader	41
8	View	43
9	REST API nodeApi	52
10	Attività Principali	57
11	Gestione Presentazioni	58
12	Caricare File	58
13	Modificare Presentazione da Desktop	59
14	Modificare Presentazione da Mobile	59
15	Gestire Sfondo	60
16	Inserire Elemento	61
17	Modificare Elemento	61
18	Modificare Frame	62
19	Modificare SVG	62
20	Modificare Testo	63
21	Model View Controller	75
22	Diagramma delle classi del package Command	76
23	diagramma di sequenza del Pattern Command	77

1	Versionamento del documento	1
2	Storico ruoli RR -> RP	3
3	Storico ruoli RP -> RQ	3
4	Storico ruoli RQ -> RA	3
5	Tracciamento Componenti-Requisiti	65
6	Tracciamento Requisiti-Componenti	69

Sommario

Il presente documento contiene la specifica tecnica delle componenti che costituiscono il prodotto software Premi.

1 Introduzione

1.1 Scopo del documento

Il presente documento ha lo scopo di definire la progettazione ad alto livello del Progetto_g Premi.

Verrà presentata l'architettura generale secondo la quale saranno organizzate le varie componenti Software_r e saranno descritti i Design Pattern utilizzati.

1.2 Scopo del Prodotto

Lo scopo del Progetto_g è la realizzazione un Software_g per la creazione ed esecuzione di presentazioni multimediali favorendo l'uso di tecniche di storytelling e visualizzazione non lineare dei contenuti.

1.3 Glossario

Al fine di evitare ogni ambiguità di linguaggio e massimizzare la comprensione dei documenti, i termini tecnici, di dominio, gli acronimi e le parole che necessitano di essere chiarite, sono riportate nel documento [Glossario_v.3.0.0.pdf](#). Ogni occorrenza di vocaboli presenti nel Glossario è marcata da una “g” minuscola in pedice.

1.4 Riferimenti

1.4.1 Normativi

- Capitolato d'appalto C4: Premi: Software_g di presentazione “better than Prezi”
<http://www.math.unipd.it/~tullio/IS-1/2014/Progetto/C4.pdf>;
- Norme di Progetto_g: [NormeDiProgetto_v.4.0.0.pdf](#);
- Analisi dei Requisiti_g: [AnalisiDeiRequisiti_v.4.0.0.pdf](#);
- Piano di qualifica: [PianoDiQualifica_v.4.0.0.pdf](#);
- Piano di Progetto_g: [PianoDiProgetto_v.4.0.0.pdf](#).

1.4.2 Informativi

- **Design Patterns: Elements of Reusable Object-Oriented Software_g**, Addison Wesley, 1995;
- Descrizione dei Design Pattern
http://sourcemaking.com/design_patterns;
- Ingegneria del Software_g - Ian Sommerville - 9a Edizione (2010):
- Slide del docente per l'anno accademico 2014/2015 reperibili al sito
<http://www.math.unipd.it/~tullio/IS-1/2014/>;

- MEAN: <http://www.mean.io/>; MEAN Web Development, Amos Q. Haviv, 2014;
- MongoDB: <http://docs.mongodb.org/manual/>;
- Angular.js: <https://docs.angularjs.org/tutorial>;
- Express.js: <http://expressjs.com/>;
- Node.js: <https://nodejs.org/documentation/>;
- jQuery: <http://api.jquery.com/> ;
- Impress.js: <https://github.com/bartaz/impress.js/>.



2 Strumenti

Si è deciso di utilizzare HTML5 e CSS3 per la presentazione grafica dell'applicazione WEB_g. HTML5 è uno standard da settembre 2014 e permette una più semplice integrazione di contenuti multimediali.

- **Multi piattaforma:** poiché l'applicazione deve essere disponibile sia su dispositivi Desktop_g che mobile HTML5 permette la creazione di strutture responsive in grado di adattarsi alle dimensioni dello schermo;
- **Integrazione con linguaggi di scripting:** con HTML5 c'è una maggiore integrazione con i linguaggi di scripting come JavaScript_g questo permetterà di rendere l'applicazione dinamica;
- **Nessuna installazione:** il fatto che l'applicazione sia sviluppata con tecnologie WEB_g quali HTML permetterà all'utente finale di poter utilizzare il prodotto senza doverlo scaricare e installare.

- **Browser:** è possibile che i Browser_g meno recenti abbiano difficoltà ad interpretare correttamente le informazioni contenute nelle pagine, rendendo difficile, se non impossibile, l'utilizzo dell'applicazione con questo linguaggio.



2.4 MEAN

MEAN è uno stack di Software_g Javascript, libero ed open source per costruire siti WEB_g dinamici ed applicazioni WEB_g. È una combinazione di MongoDB, Express.js ed Angular.js, eseguita su Node.js.

2.4.1 MongoDB

MongoDB è un database NoSQL open source orientato ai documenti, facilmente scalabile e ad alte prestazioni. Si allontana dalla struttura tradizionale basata su tabelle dei database relazionali, in favore di documenti in stile JSON con schema dinamico; questo rende l'integrazione di dati più semplice e facile in alcuni tipi d'applicazioni.

2.4.2 Express.js

Express.js è un Framework_g per applicazioni WEB_g Node.js, disegnato per costruire applicazioni WEB_g single-page, multi-page o ibride. È costruito sopra il modulo Connect di Node.js e fa uso della sua architettura middleware; nel nostro sistema è utilizzato in particolar modo per la gestione dei path da cui sono offerti i servizi per l'interfacciamento con il database Mongo.

2.4.3 AngularJS

AngularJS, è un Framework_g per applicazioni WEB_g, open-source, mantenuto da Google e da una comunità di sviluppatori e corporations. Mira a semplificare lo sviluppo ed il test di applicazioni single-page fornendo un Framework_g per l'architettura model-view-whatever lato-client.

Il Framework_g AngularJS come prima cosa legge la pagina HTML, che ha al suo interno degli attributi Tag_g personalizzati; Angular interpreta questi attributi come direttive per legare parti di input o di output della pagina ad un modello che è rappresentato da variabili Javascript standard. Il valore di queste variabili Javascript può essere impostato manualmente all'interno del Codice_g, oppure ricavato da Risorse_g JSON statiche o dinamiche.

2.4.4 Node.js

Node.js è un ambiente di esecuzione open source e cross-platform per applicazioni lato Server_g; le applicazioni Node.js sono scritte in linguaggio Javascript. Node.js fornisce un'architettura scalabile orientata agli eventi grazie alla sua natura asincrona. Node.js usa il motore Javascript V8 di Google per eseguire Codice_g, ed una larga percentuale dei moduli base è scritta in Javascript.

2.5 Impress.js

Impress.js è un Framework_g open source che permette di visualizzare i Tag_g div di una pagina HTML come passi di una presentazione. Si è deciso di affidare la visualizzazione della presentazione a questa libreria in quanto permette di conseguire quasi tutti i Requisiti_g obbligatori relativi all'esecuzione senza dover scrivere ingenti quantità di Codice_g aggiuntivo. Si è deciso inoltre di integrare nel Framework_g alcune funzioni_g in modo da rispondere a tutti i Requisiti_g obbligatori relativi all'esecuzione.



3.1 Metodo e formalismi

- Tipo;
- Funzione_g;
- Classi o interfacce estese;
- Interfacce implementate;
- Relazioni con altre classi.

Per i diagrammi di Package, classi e attività verrà usata la notazione UML 2.0.

Il prodotto si presenta suddiviso in due parti distinte, una parte che verrà eseguita localmente all'utente ed una parte server. La parte remota è formata da un server con tecnologia NodeJs che comunica direttamente con un database MongoDB. La parte locale si presenta suddivisa in tre parti distinte: Model, View e Controller. Per la parte locale si è quindi cercato di implementare il design pattern architetturale MVC in modo da garantire un basso livello di accoppiamento. In figura 1 viene riportato il diagramma dei package, in seguito vengono elencate le componenti dell'applicativo con le relative caratteristiche e funzionalità generali, per una trattazione più approfondita si rimanda alle sezioni specifiche dei componenti.

Contiene la rappresentazione dei dati, l'implementazione dei metodi da applicare ad essi e lo stato di questi ultimi; costituisce il cuore del Software_g e risulta di fatto totalmente indipendente dagli altri due strati.

Contiene tutti gli elementi della GUI, comprese le interfacce di comunicazione con le librerie grafiche esterne. Si limita a passare gli input inviati dall'utente allo strato che sta sotto di lei, il Controller, demandandone a quest'ultimo la gestione.

E' il punto di incontro tra la View e il Model: i dati ricevuti da quest'ultimo sono elaborati per essere presentati alla View.

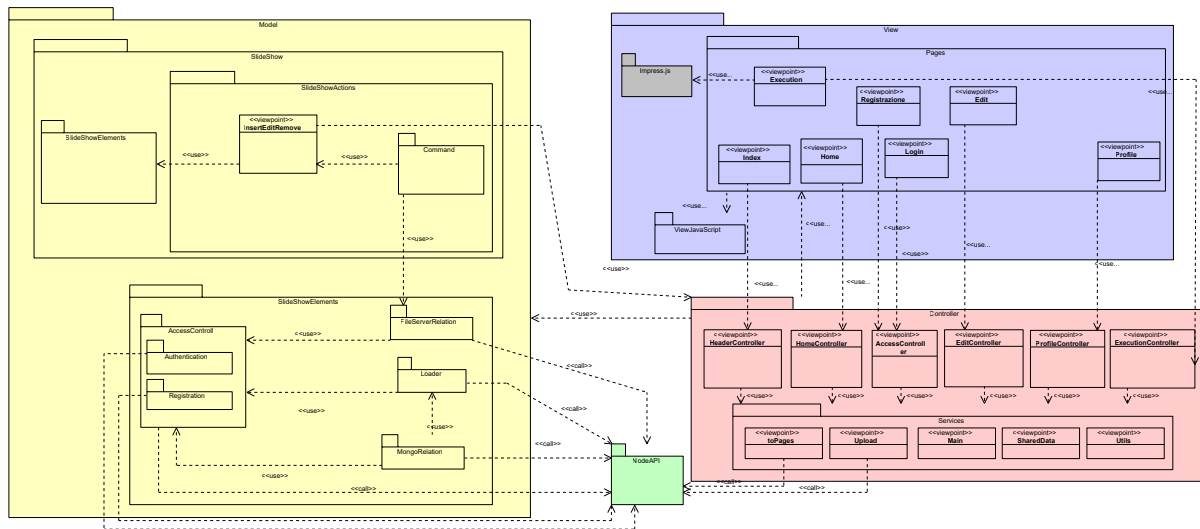


Fig 1: Architettura generale del sistema

3.2.4 nodeApi

Si tratta della parte remota del sistema rispetto all'utente. La Api segue i principi di una API REST e raccoglie funzionalità di interazione con la base dati MongoDB e di caricamento e gestione dei file multimediali che l'utente può inserire nelle presentazioni.



Ogni componente appartiene al package Premi, quindi lo scope sarà Premi:<componente>.

Tipo, obiettivo e funzione del componente: questo Package è la parte Model dell'architettura MVC.

Package contenuti:

- #### 4.1.1 Model::SlideShow

Relazioni d'uso di altre componenti: il package è in relazione con Controller da cui riceve le chiamate relative a inserimento, eliminazione e modifica degli elementi_g.

Tipo, obiettivo e funzione del componente: all'interno di questo Package si trovano le classi che si occupano della costruzione, dell'inserimento, della rimozione e della modifica degli elementi della presentazione.

Relazioni d'uso di altre componenti: il package è in relazione con

Model::SlideShow::SlideShowActions::Command che ne invoca le funzioni, passando i relativi parametri per l'inserimento, la rimozione e la modifica degli elementi. Tutti i componenti seguenti appartengono al package SlideShowActions, quindi lo scope sarà Model::SlideShow::SlideShowActions::<componente>.

4.1.3 InsertEditRemove

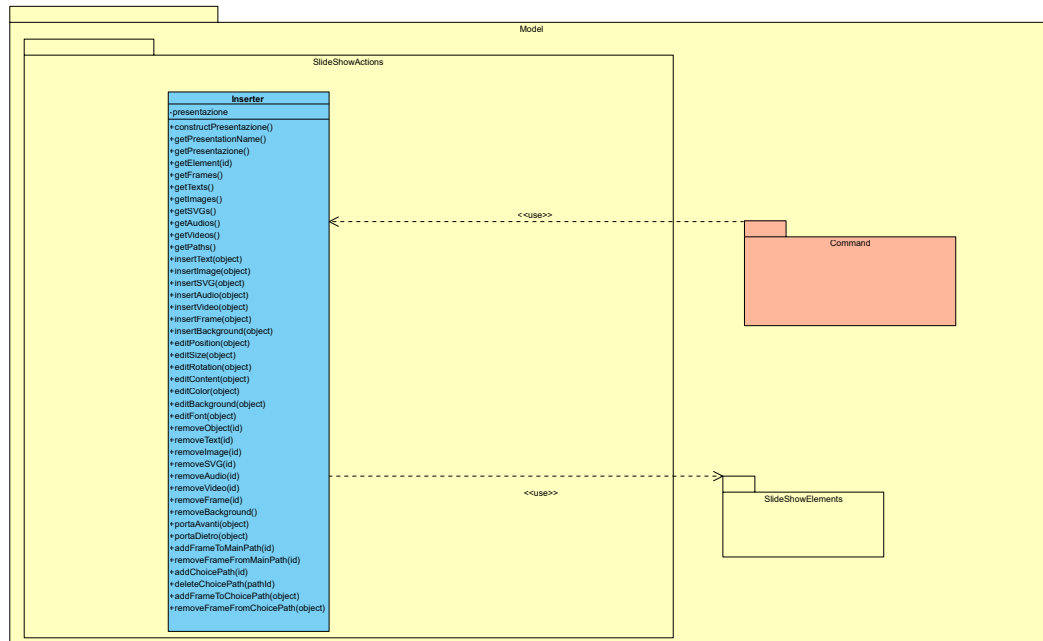


Fig 2: InsertEditRemove

Tipo, obiettivo e funzione del componente: classe statica che offre i metodi destinati all'inserimento, eliminazione e modifica degli elementi, all'interno di una presentazione.

Interfacce con e relazioni d'uso e da altre componenti: è il componente receiver del Design Pattern Command.

Relazioni d'uso di altre componenti:

- `Model::SlideShow::SlideShowActions::Command::ConcreteEditSizeCommand` -> invoca il metodo `editSize()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteEditPositionCommand` -> invoca il metodo `editPosition()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteEditRotationCommand` -> invoca il metodo `editRotation()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteEditColorCommand` -> invoca il metodo `editColor()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteEditFontCommand` -> invoca il metodo `editFont()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteEditBackgroundCommand` -> invoca il metodo `editBackground()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteEditBookmarkCommand` -> aggiorna il valore di `Bookmarkg`;

- `Model::SlideShow::SlideShowActions::Command::ConcretePortaAvantiCommand` -> invoca il metodo `portaAvanti()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcretePortaDietroCommand` -> invoca il metodo `portaDietro()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteAddToMainPathCommand` -> invoca il metodo `addFrameToMainPath()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteRemoveFromMainPathCommand` -> invoca il metodo `removeFrameFromMainPath()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteNewChoicePathCommand` -> invoca il metodo `addChoicePath()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteDeleteChoicePathCommand` -> invoca il metodo `deleteChoicePath()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteAddToChoicePathCommand` -> invoca il metodo `addFrameToChoicePath()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteRemoveFromChoicePathCommand` -> invoca il metodo `removeFrameFromChoicePath()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteTextInsertCommand` -> invoca il metodo `insertText()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteFrameInsertCommand` -> invoca il metodo `insertFrame()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteImageInsertCommand` -> invoca il metodo `insertImage()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteAudioInsertCommand` -> invoca il metodo `insertAudio()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteVideoInsertCommand` -> invoca il metodo `insertVideo()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteBackgroundInsertCommand` -> invoca il metodo `insertBackground()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteTextRemoveCommand` -> invoca il metodo `removeText()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteFrameRemoveCommand` -> invoca il metodo `removeFrame()` messo a disposizione da `insertEditRemove`;
- `Model::SlideShow::SlideShowActions::Command::ConcreteImageRemoveCommand` -> invoca il metodo `removeImage()` messo a disposizione da `insertEditRemove`;



- Questo documento è distribuito sotto licenza [Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License](https://creativecommons.org/licenses/by-nc-sa/3.0/).



- #### 4.1.4 Model::SlideShow::SlideShowActions::Command

Relazioni d'uso di altre componenti: all'interno del Model, il package è in relazione con

- #### 4.1.4.1 Invoker

Relazioni d'uso di altre componenti:

- `Model::EditController->crea un oggetto di una sottoclasse di Model::SlideShow::SlideShowActions::Command::AbstractCommand` passandolo all'Invoker che ne invoca il metodo `execute()` e lo inserisce nello stack "undostack", richiama il metodo che svuota lo stack "redostack".
Può inoltre invocare il metodo "undo()" dell'Invoker che provvede a richiamare il metodo `undoaction()` del comando sulla cima dello stack "undostack" e a spostarlo quindi nello stack "redostack". Alternativamente invoca il metodo "redo()" dell'Invoker che provvede a invocare il metodo `doaction()` del comando sulla cima dello stack "redostack" e a spostarlo quindi nello stack "undostack";
- `Model::SlideShow::SlideShowActions::Command::AbstractCommand <- Invoker` invoca il metodo `doaction()` dell'oggetto della sottoclasse di `AbstractCommand`. Alternativamente invoca il metodo `undoaction()`.

Interfacce con e relazioni d'uso e da altre componenti: viene invocato per effettuare le operazioni di modifica alla presentazione, a sua volta invoca i metodi `doaction()` o `undoaction()` di una classe derivata da `Model::SlideShow::SlideShowActions::Command::AbstractCommand` per eseguire materialmente il comando. Quando un comando viene eseguito, `Invoker` lo salva in un array `$undostack` [1].

4.1.4.2 AbstractCommand

Tipo, obiettivo e funzione del componente: è classe astratta del Design Pattern Command, è classe base per i comandi di modifica, inserimento ed eliminazione.

Relazioni d'uso di altre componenti:

- **Model::Invoker** -> esegue materialmente il comando, richiamandone il metodo `doaction()`; inoltre provvede ad annullare l'ultima operazione invocandone il metodo `undoaction()`.

Interfacce con e relazioni d'uso e da altre componenti:Viene utilizzata per applicare un generico parametro di trasformazione ad un oggetto della presentazione, questo parametro verrà poi specificato dalle classi concrete.

Sottoclassi:

- ConcreteTextInsertCommand;
- ConcreteFrameInsertCommand;
- ConcreteImageInsertCommand;
- ConcreteAudioInsertCommand;
- ConcreteVideoInsertCommand;
- ConcreteBackgroundInsertCommand;
- ConcreteTextRemoveCommand;
- ConcreteFrameRemoveCommand;

- ConcreteImageRemoveCommand;
- ConcreteAudioRemoveCommand;
- ConcreteVideoRemoveCommand;
- ConcreteEditSizeCommand;
- ConcreteEditPositionCommand;
- ConcreteEditRotationCommand;
- ConcreteEditColorCommand;
- ConcreteEditBackgroundCommand;
- ConcreteEditFontCommand;
- ConcreteEditContentCommand;
- ConcreteEditBookmarkCommand;
- ConcretePortaAvantiCommand;
- ConcretePortaDietroCommand;
- ConcreteAddToMainPathCommand;
- ConcreteRemoveFromMainPathCommand;
- ConcreteNewChoicePathCommand;
- ConcreteDeleteChoicePathCommand;
- ConcreteAddToChoicePathCommand;
- ConcreteRemoveFromChoicePathCommand;

4.1.4.3 ConcreteTextInsertCommand

Tipo, obiettivo e funzione del componente: è classe concreta del Design Pattern Command, rappresenta un comando per inserire un nuovo Elemento_g testuale nella presentazione.

Relazioni d'uso di altre componenti:

- `Controller::EditController ->` invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- `Model::SlideShow::SlideShowActions::Command::Invoker ->` invoca il metodo `doaction()` del comando e lo inserisce nel campo `dati undostack` e ne setta il valore del campo `dati booleano executed` a `true`, o ne invoca il metodo di annullamento `undoaction()` e lo inserisce nel campo `dati redostack`;
- `Model::SlideShow::SlideShowActions::InsertEditRemove <-` invoca il metodo `insertText(...)` della classe statica per l'inserimento di un `Elementog`;



- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.4 ConcreteFrameInsertCommand

Tipo, obiettivo e funzione del componente: è classe concreta del Design Pattern Command, rappresenta un comando per inserire un nuovo `Elementog Frameg` nella presentazione.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> invoca il metodo `doaction()` del comando e lo inserisce nel campo dati `undostack` e ne setta il valore del campo dati booleano `executed` a `true`, o ne invoca il metodo di annullamento `undoaction()` e lo inserisce nel campo dati `redostack`;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` <- invoca il metodo `insertFrame(...)` della classe statica per l'inserimento di un `Elementog Frameg` nella presentazione;
- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.5 ConcreteImageInsertCommand

Tipo, obiettivo e funzione del componente: è classe concreta del Design Pattern Command, rappresenta un comando per inserire un nuovo `Elementog immagine` nella presentazione.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> invoca il metodo `doaction()` del comando e lo inserisce nel campo dati `undostack` e ne setta il valore del campo dati booleano `executed` a `true`, o ne invoca il metodo di annullamento `undoaction()` e lo inserisce nel campo dati `redostack`;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` <- invoca il metodo `insertImage(...)` della classe statica per l'inserimento di un `Elementog immagine` nella presentazione;



- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.6 ConcreteAudioInsertCommand

Tipo, obiettivo e funzione del componente: è classe concreta del Design Pattern Command, rappresenta un comando per inserire un nuovo `Elementog` audio nella presentazione.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> invoca il metodo `doaction()` del comando e lo inserisce nel campo dati `undostack` e ne setta il valore del campo dati booleano `executed` a `true`, o ne invoca il metodo di annullamento `undoaction()` e lo inserisce nel campo dati `redostack`;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` <- invoca il metodo `insertAudio(...)` della classe statica per l'inserimento di un `Elementog` audio nella presentazione;
- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Interfacce con e relazioni d'uso e da altre componenti: viene utilizzata per gestire le richieste di inserimento di un nuovo `Elementog` Audio.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.7 ConcreteVideoInsertCommand

Tipo, obiettivo e funzione del componente: è classe concreta del Design Pattern Command, rappresenta un comando per inserire un nuovo `Elementog` video nella presentazione.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> invoca il metodo `doaction()` del comando e ne setta il valore del campo dati booleano `executed` a `true`, o ne invoca il metodo di annullamento `undoaction()` e lo inserisce nel campo dati `redostack`;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` <- invoca il metodo `insertVideo(...)` della classe statica per l'inserimento di un `Elementog` video nella presentazione;



- Classi ereditate:

- #### 4.1.4.8 ConcreteBackgroundInsertCommand

Relazioni d'uso di altre componenti:

- Classi ereditate:**

- #### 4.1.4.9 ConcreteTextRemoveCommand

Relazioni d'uso di altre componenti:

- Università degli studi di Padova - 2014/2015



- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.10 ConcreteFrameRemoveCommand

Tipo, obiettivo e funzione del componente: è classe concreta del Design Pattern Command, rappresenta un comando per rimuovere un `Elementog Frameg` dalla presentazione.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> invoca il metodo `doaction()` del comando e lo inserisce nel campo dati `undostack` e ne setta il valore del campo dati booleano `executed` a `true`, o ne invoca il metodo di annullamento `undoaction()` e lo inserisce nel campo dati `redostack`;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` <- invoca il metodo `removeFrame(...)` della classe statica per la rimozione di un `Elementog Frameg` dalla presentazione;
- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.11 ConcreteImageRemoveCommand

Tipo, obiettivo e funzione del componente: è classe concreta del Design Pattern Command, rappresenta un comando per rimuovere un `Elementog immagine` dalla presentazione.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> Invoker invoca il metodo `doaction()` del comando e lo inserisce nel campo dati `undostack` e ne setta il valore del campo dati booleano `executed` a `true`, o ne invoca il metodo di annullamento `undoaction()` e lo inserisce nel campo dati `redostack`;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` <- invoca il metodo `removeImage(...)` della classe statica per l'eliminazione di un `Elementog immagine` dalla presentazione;



- Premi::Controller::EditController <- l'oggetto invoca il metodo appropriato di EditController quando viene invocato il metodo doaction() e il campo dati booleano executed ha valore true, o quando viene invocato il metodo undoaction().

Classi ereditate:

- Model::SlideShow::SlideShowActions::Command::AbstractCommand.

4.1.4.12 ConcreteAudioRemoveCommand

Tipo, obiettivo e funzione del componente: è classe concreta del Design Pattern Command, rappresenta un comando per rimuovere un Elemento_g audio dalla presentazione.

Relazioni d'uso di altre componenti:

- Controller::EditController -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- Model::SlideShow::SlideShowActions::Command::Invoker -> Invoker invoca il metodo doaction() del comando e lo inserisce nel campo dati undostack e ne setta il valore del campo dati booleano executed a true, o ne invoca il metodo di annullamento undoaction() e lo inserisce nel campo dati redostack;
- Model::SlideShow::SlideShowActions::InsertEditRemove <- invoca il metodo removeAudio(...) della classe statica per l'eliminazione di un Elemento_g immagine dalla presentazione;
- Premi::Controller::EditController <- l'oggetto invoca il metodo appropriato di EditController quando viene invocato il metodo doaction() e il campo dati booleano executed ha valore true, o quando viene invocato il metodo undoaction().

Classi ereditate:

- Model::SlideShow::SlideShowActions::Command::AbstractCommand.

4.1.4.13 ConcreteVideoRemoveCommand

Tipo, obiettivo e funzione del componente: è classe concreta del Design Pattern Command, rappresenta un comando per rimuovere un Elemento_g video dalla presentazione.

Relazioni d'uso di altre componenti:

- Controller::EditController -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- Model::SlideShow::SlideShowActions::Command::Invoker -> Invoker invoca il metodo doaction() del comando e lo inserisce nel campo dati undostack e ne setta il valore del campo dati booleano executed a true, o ne invoca il metodo di annullamento undoaction() e lo inserisce nel campo dati redostack;
- Model::SlideShow::SlideShowActions::InsertEditRemove <- invoca il metodo removeVideo(...) della classe statica per l'eliminazione di un Elemento_g video dalla presentazione;



- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.14 ConcreteEditSizeCommand

Tipo, obiettivo e funzione del componente: è classe concreta del Design Pattern Command, rappresenta un comando per modificare le dimensioni di un `Elementog` della presentazione.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> Invoker invoca il metodo `doaction()` del comando e lo inserisce nel campo dati `undostack` e ne setta il valore del campo dati booleano `executed` a `true`, o ne invoca il metodo di annullamento `undoaction()` e lo inserisce nel campo dati `redostack`;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` <- il comando invoca il metodo `editSize(...)` della classe statica per la modifica dei campi dati relativi alle dimensioni dell'oggetto nella presentazione;
- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.15 ConcreteEditPositionCommand

Tipo, obiettivo e funzione del componente: è classe concreta del Design Pattern Command, rappresenta un comando per modificare la posizione di un `Elementog` della presentazione.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> Invoker invoca il metodo `doaction()` del comando e lo inserisce nel campo dati `undostack` e ne setta il valore del campo dati booleano `executed` a `true`, o ne invoca il metodo di annullamento `undoaction()` e lo inserisce nel campo dati `redostack`;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` <- il comando invoca il metodo `editPosition(...)` della classe statica per la modifica dei campi dati relativi alla posizione dell'oggetto nella presentazione;



- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.16 ConcreteEditColorCommand

Tipo, obiettivo e funzione del componente: è classe concreta del Design Pattern Command, rappresenta un comando per modificare il colore di un `Elementog` della presentazione.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'`Invoker`;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> `Invoker` invoca il metodo `doaction()` del comando e lo inserisce nel campo dati `undostack` e ne setta il valore del campo dati booleano `executed` a `true`, o ne invoca il metodo di annullamento `undoaction()` e lo inserisce nel campo dati `redostack`;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` <- il comando invoca il metodo `editColor(...)` della classe statica per la modifica del campo dati relativo al colore dell'oggetto della presentazione;
- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.17 ConcreteEditBackgroundCommand

Tipo, obiettivo e funzione del componente: è classe concreta del Design Pattern Command, rappresenta un comando per modificare lo sfondo di un `Elementog` `Frameg` della presentazione.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'`Invoker`;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> `Invoker` invoca il metodo `doaction()` del comando e lo inserisce nel campo dati `undostack` e ne setta il valore del campo dati booleano `executed` a `true`, o ne invoca il metodo di annullamento `undoaction()` e lo inserisce nel campo dati `redostack`;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` <- il comando invoca il metodo `editBackground(...)` della classe statica per la modifica del campo dati relativo allo sfondo dell'oggetto della presentazione;



- Premi::Controller::EditController <- l'oggetto invoca il metodo appropriato di EditController quando viene invocato il metodo doaction() e il campo dati booleano executed ha valore true, o quando viene invocato il metodo undoaction().

Classi ereditate:

- Model::SlideShow::SlideShowActions::Command::AbstractCommand.

4.1.4.18 ConcreteEditRotationCommand

Tipo, obiettivo e funzione del componente: è classe concreta del Design Pattern Command, rappresenta un comando per modificare l'orientamento di un Elemento_g della presentazione.

Relazioni d'uso di altre componenti:

- Controller::EditController -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- Model::SlideShow::SlideShowActions::Command::Invoker -> Invoker invoca il metodo doaction() del comando e lo inserisce nel campo dati undostack e ne setta il valore del campo dati booleano executed a true, o ne invoca il metodo di annullamento undoaction() e lo inserisce nel campo dati redostack;
- Model::SlideShow::SlideShowActions::InsertEditRemove <- il comando invoca il metodo editRotation(...) della classe statica per la modifica del campo dati relativo all'orientamento dell'oggetto della presentazione;
- Premi::Controller::EditController <- l'oggetto invoca il metodo appropriato di EditController quando viene invocato il metodo doaction() e il campo dati booleano executed ha valore true, o quando viene invocato il metodo undoaction().

Classi ereditate:

- Model::SlideShow::SlideShowActions::Command::AbstractCommand.

4.1.4.19 ConcreteEditFontCommand

Tipo, obiettivo e funzione del componente: è classe concreta del Design Pattern Command, rappresenta un comando per modificare il carattere di un Elemento_g testuale della presentazione.

Relazioni d'uso di altre componenti:

- Controller::EditController -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- Model::SlideShow::SlideShowActions::Command::Invoker -> Invoker invoca il metodo doaction() del comando e lo inserisce nel campo dati undostack e ne setta il valore del campo dati booleano executed a true, o ne invoca il metodo di annullamento undoaction() e lo inserisce nel campo dati redostack;



- `Model::SlideShow::SlideShowActions::InsertEditRemove` <- il comando invoca il metodo `editColor(...)` della classe statica per la modifica dei campi dati relativi al `Fontg` dell'oggetto testuale della presentazione;
- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.20 Classe `ConcreteEditContentCommand`

È classe concreta del Design Pattern Command, serve a assegnare del testo ad un `Elementog` della presentazione.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> invoca il metodo `doaction()` del comando e lo inserisce nel campo dati `undostack` e ne setta il valore del campo dati booleano `executed` a `true`, o ne invoca il metodo di annullamento `undoaction()` e lo inserisce nel campo dati `redostack`;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` <- il comando invoca il metodo `editContent(spec)` della classe statica per la modifica dei campi dati relativi testo dell'oggetto della presentazione;
- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.21 Classe `ConcreteEditBookmarkCommand`

È classe concreta del Design Pattern Command, applica un `Bookmarkg` ad un `Elementog` della presentazione.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> invoca il metodo `doaction()` del comando e lo inserisce nel campo dati `undostack` e ne setta il valore del campo dati booleano `executed` a `true`, o ne invoca il metodo di annullamento `undoaction()` e lo inserisce nel campo dati `redostack`;



- `Model::SlideShow::SlideShowActions::InsertEditRemove` <- il comando invoca il metodo `updateBookmark(...)` della classe statica per la modifica del campo relativo al `Bookmarkg` dell'`Elementog` della presentazione;
- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.22 Classe ConcretePortaAvantiCommand

È classe concreta del Design Pattern Command, sposta all'indietro l'`Elementog` su cui è stata invocata.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'`Invoker`;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> invoca il metodo `doaction()` del comando e lo inserisce nel campo dati `undostack` e ne setta il valore del campo dati booleano `executed` a `true`, o ne invoca il metodo di annullamento `undoaction()` e lo inserisce nel campo dati `redostack`;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` <- il comando invoca il metodo `portaAvanti()` della classe statica per la modifica del campo `z-index` degli oggetti della presentazione;
- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.23 Classe ConcretePortaDietroCommand

È classe concreta del Design Pattern Command, sposta all'indietro l'`Elementog` su cui è stata invocata.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'`Invoker`;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> invoca il metodo `doaction()` del comando e lo inserisce nel campo dati `undostack` e ne setta il valore del campo dati booleano `executed` a `true`, o ne invoca il metodo di annullamento `undoaction()` e lo inserisce nel campo dati `redostack`;



- `Model::SlideShow::SlideShowActions::InsertEditRemove` <- il comando invoca il metodo `portaDietro()` della classe statica per la modifica del campo z-index degli oggetti della presentazione;
- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.24 Classe ConcreteAddToMainPathCommand

È classe concreta del Design Pattern Command, inserisce in `Frameg` su cui è chiamata nella posizione specificata, all'interno del Percorso_g principale.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> invoca il metodo `doaction()` del comando e lo inserisce nel campo dati `undostack` e ne setta il valore del campo dati booleano `executed` a `true`, o ne invoca il metodo di annullamento `undoaction()` e lo inserisce nel campo dati `redostack`;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` <- il comando invoca il metodo `addFrameToMainPath(...)` della classe statica per la modifica dell'oggetto relativo al Percorso_g principale della presentazione;
- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.25 Classe concreteRemoveFromMainPathCommand

È classe concreta del Design Pattern Command, rimuove dal Percorso_g principale della presentazione il `Frameg` su cui è stata chiamata.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> invoca il metodo `doaction()` del comando e lo inserisce nel campo dati `undostack` e ne setta il valore del campo dati booleano `executed` a `true`, o ne invoca il metodo di annullamento `undoaction()` e lo inserisce nel campo dati `redostack`;



- `Model::SlideShow::SlideShowActions::InsertEditRemove` <- il comando invoca il metodo `removeFrameFromMainPath(...)` della classe statica per la modifica dell'oggetto relativo al Percorso_g principale della presentazione;
- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.26 Classe concreteNewChoicePathCommand

È classe concreta del Design Pattern Command, riceve l'id del Frame_g da cui parte il nuovo Percorso_g scelta.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> invoca il metodo `doaction()` del comando e lo inserisce nel campo dati `undostack` e ne setta il valore del campo dati booleano `executed` a `true`, o ne invoca il metodo di annullamento `undoaction()` e lo inserisce nel campo dati `redostack`;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` <- il comando invoca il metodo `addChoicePath(...)` della classe statica per la modifica dell'oggetto relativo ai percorsi_g della presentazione;
- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.27 Classe concreteDeleteChoicePathCommand

È classe concreta del Design Pattern Command, rimuove dal Percorso_g scelta il Frame_g su cui è stata chiamata.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> invoca il metodo `doaction()` del comando e lo inserisce nel campo dati `undostack` e ne setta il valore del campo dati booleano `executed` a `true`, o ne invoca il metodo di annullamento `undoaction()` e lo inserisce nel campo dati `redostack`;



- `Model::SlideShow::SlideShowActions::InsertEditRemove` <- il comando invoca il metodo `deleteChoicePath(...)` della classe statica per la modifica dell'oggetto relativo al Percorso_g principale della presentazione;
- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.28 Classe ConcreteRemoveFromChoicePathCommand

È classe concreta del Design Pattern Command, rimuove dal Percorso_g scelta il Frame_g su cui è stata chiamata.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> invoca il metodo `removeFrameFromChoicePath(...)` della classe statica per la modifica dell'oggetto relativo al Percorso_g scelta della presentazione;
- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.4.29 Classe ConcreteAddToChoicePathCommand

È classe concreta del Design Pattern Command, aggiunge il Frame_g su cui è chiamata ad un Percorso_g scelta.

Relazioni d'uso di altre componenti:

- `Controller::EditController` -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker;
- `Model::SlideShow::SlideShowActions::Command::Invoker` -> invoca il metodo `addFrameToChoicePath(...)` della classe statica per la modifica dell'oggetto relativo al Percorso_g scelta della presentazione;
- `Premi::Controller::EditController` <- l'oggetto invoca il metodo appropriato di `EditController` quando viene invocato il metodo `doaction()` e il campo dati booleano `executed` ha valore `true`, o quando viene invocato il metodo `undoaction()`.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::Command::AbstractCommand`.

4.1.5 Model::SlideShow::SlideShowElements

Tutti i componenti seguenti appartengono al package SlideShowElements, quindi lo scope sarà Model::SlideShow::SlideShowElements::<componente>.

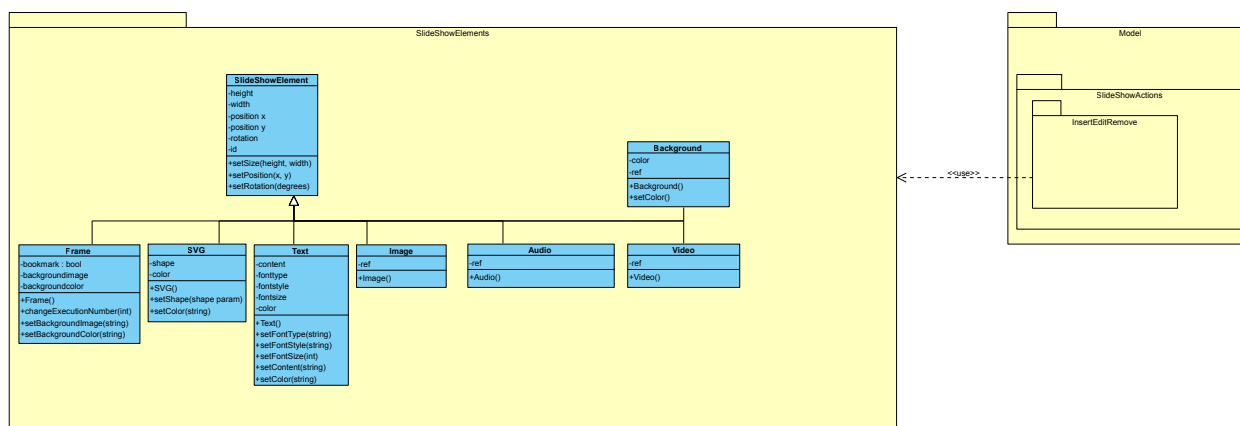


Fig 4: SlideShowElements

Tipo, obiettivo e funzione del componente: di questo package fanno parte le classi degli elementig della presentazione e la classe che definisce la presentazione stessa.

Relazioni d'uso di altre componenti: Model::SlideShow::SlideShowElements è in comunicazione con

- Model::SlideShow::SlideShowActions::Insert, i cui oggetti durante la modifica della presentazione istanziano oggetti di tipo SlideShowElement;
- Model::Remove, i cui oggetti rimuovono gli elementig di tipo SlideShowElements;
- Model::SlideShow::SlideShowActions::EditElements, i cui oggetti invocano metodi degli oggetti SlideShowElement che ne impostano i campi.

4.1.5.1 SlideShowElement

Tipo, obiettivo e funzione del componente: gli oggetti della classe SlideShowElement rappresentano gli elementig della presentazione.

Relazioni d'uso di altre componenti:

- Model::SlideShow::SlideShowActions::InsertEditRemove -> invoca il costruttore delle sottoclassi di SlideShowElements;
- Model::SlideShow::SlideShowActions::InsertEditRemove -> gli oggetti delle sue sottoclassi richiamano le funzionig delle sottoclassi di SlideShowElement che gestiscono l'impostazione dei campi dati;
- Model::SlideShow::SlideShowActions::InsertEditRemove -> gli oggetti delle sue sottoclassi rimuovono dai contenitori di SlideShow gli oggetti di classe SlideShowElement e ne richiamano i distruttori.



Model::SlideShow::SlideShowActions::InsertEditRemove istanzia oggetti di sottoclassi di SlideShowElement.

- `Model::SlideShow::Text;`
- `Model::SlideShow::Frameg;`
- `Model::SlideShow::Image;`
- `Model::SlideShow::Audio;`
- `Model::SlideShow::Video;`
- `Model::SlideShow::Background.`

Tipo, obiettivo e funzione del componente: gli oggetti della classe Text rappresentano gli elementi_g di tipo testuale della presentazione.

- `Model::SlideShow::SlideShowActions::InsertEditRemove` -> invoca il costruttore di `Text` e inserisce l'oggetto nella presentazione;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` -> rimuove l'oggetto `Text` dalla presentazione;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` -> invoca i metodi che modificano i campi dati dell'oggetto.

Classi ereditate:

- `Model::SlideShow::SlideShowActions::InsertEditRemove` -> invoca il costruttore di `Frameg` e inserisce l'oggetto nella presentazione;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` -> rimuove l'oggetto `Frameg` dalla presentazione;

- `Model::SlideShow::SlideShowActions::InsertEditRemove` -> invoca i metodi che modificano i campi dati dell'oggetto.

Interfacce con e relazioni d'uso e da altre componenti: gli oggetti della classe `Frame_g` vengono istanziati da `Model::SlideShow::SlideShowActions::InsertEditRemove` e inseriti nella presentazione.

Classi ereditate:

- `Model::SlideShow::SlideShowElement`.

4.1.5.4 Image

Tipo, obiettivo e funzione del componente: gli oggetti della classe Image rappresentano gli elementi_g di tipo immagine della presentazione.

Relazioni d'uso di altre componenti:

- `Model::SlideShow::SlideShowActions::InsertEditRemove` -> invoca il costruttore di `Image` e inserisce l'oggetto nella presentazione;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` -> rimuove l'oggetto `Image` dalla presentazione;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` -> invoca i metodi che modificano i campi dati dell'oggetto;

Interfacce con e relazioni d'uso e da altre componenti: gli oggetti della classe Image vengono istanziati da Model::SlideShow::SlideShowActions::InsertEditRemove e inseriti nella presentazione.

Classi ereditate:

- `Model::SlideShow::SlideShowElement`.

4.1.5.5 Audio

Tipo, obiettivo e funzione del componente: gli oggetti della classe Audio rappresentano gli elementi_g di tipo audio della presentazione.

Relazioni d'uso di altre componenti:

- `Model::SlideShow::SlideShowActions::InsertEditRemove` -> invoca il costruttore di `Audio` e inserisce l'oggetto nella presentazione;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` -> rimuove l'oggetto `Audio` dalla presentazione;
- `Model::SlideShow::SlideShowActions::InsertEditRemove` -> invoca i metodi che modificano i campi dati dell'oggetto.

Interfacce con e relazioni d'uso e da altre componenti: gli oggetti della classe Audio vengono istanziati da Model::SlideShow::SlideShowActions::InsertEditRemove e inseriti nella presentazione.

Classi ereditate:

- Model::SlideShow::SlideShowElements::SlideShowElement.



4.1.5.6 Video

Tipo, obiettivo e funzione del componente: gli oggetti della classe Video rappresentano gli elementi di tipo video della presentazione.

Relazioni d'uso di altre componenti:

- Model::SlideShow::SlideShowActions::InsertEditRemove -> invoca il costruttore di Video e inserisce l'oggetto nella presentazione;
- Model::SlideShow::SlideShowActions::InsertEditRemove -> rimuove l'oggetto Video dalla presentazione;
- Model::SlideShow::SlideShowActions::InsertEditRemove -> invoca i metodi che modificano i campi dati dell'oggetto.

Interfacce con e relazioni d'uso e da altre componenti: gli oggetti della classe Video vengono istanziati da Model::SlideShow::SlideShowActions::InsertEditRemove e inseriti nella presentazione.

Classi ereditate:

- Model::SlideShow::SlideShowElements::SlideShowElement.

4.1.6 Background

Tipo, obiettivo e funzione del componente: gli oggetti della classe Background rappresentano lo sfondo della presentazione.

Relazioni d'uso di altre componenti:

- Model::SlideShow::SlideShowActions::InsertEditRemove -> invoca il costruttore di Background e inserisce l'oggetto nella presentazione;
- Model::SlideShow::SlideShowActions::InsertEditRemove -> rimuove l'oggetto Video dalla presentazione;
- Model::SlideShow::SlideShowActions::InsertEditRemove -> invoca i metodi che modificano i campi dati dell'oggetto.

Interfacce con e relazioni d'uso e da altre componenti: gli oggetti della classe Background vengono istanziati da Model::SlideShow::SlideShowActions::InsertEditRemove e inseriti nella presentazione.

Classi ereditate:

- Model::SlideShow::SlideShowElements::SlideShowElement.

```

    usecaseDiagram
        package serverRelation {
            usecase loader
            usecase fileServerRelation
            usecase mongoRelation
            usecase accessControl
        }
        package server {
            usecase Controller
            usecase Server
        }
        package client {
            usecase slideShow
        }

        loader --> fileServerRelation : <<use>>
        loader --> mongoRelation : <<use>>
        loader --> accessControl : <<use>>
        fileServerRelation --> mongoRelation : <<use>>
        mongoRelation --> accessControl : <<use>>
        Controller --> loader : <<use>>
        Controller --> fileServerRelation : <<use>>
        Controller --> mongoRelation : <<use>>
        Controller --> Server : <<use>>
        Controller --> accessControl : <<use>>
        slideShow --> loader : <<use>>
        slideShow --> Controller : <<use>>
        slideShow --> Server : <<use>>
    
```

The diagram illustrates the relationships between various components in a file management system. The components are organized into three packages: **serverRelation** (yellow), **server** (green), and **client** (orange). The **serverRelation** package contains four use cases: **loader**, **fileServerRelation**, **mongoRelation**, and **accessControl**. The **server** package contains two use cases: **Controller** and **Server**. The **client** package contains one use case: **slideShow**. The relationships are as follows: **loader** uses **fileServerRelation**, **mongoRelation**, and **accessControl**. **fileServerRelation** uses **mongoRelation**. **mongoRelation** uses **accessControl**. **Controller** uses **loader**, **fileServerRelation**, **mongoRelation**, **Server**, and **accessControl**. **slideShow** uses **loader**, **Controller**, and **Server**. All relationships are represented by dashed arrows with the stereotype **<<use>>**.

Tipo, obiettivo e funzione del componente: package, racchiude le funzionalità del sistema che interagiscono con i servizi offerti dal Server_g nodeJs per l'interazione con la base dati MongoDB e la gestione dei File_g multimediali in Cloud

- relazioni verso **Server** del quale si utilizzano i servizi RESTfull;
- relazioni da **Controller** per il recupero o la creazione di una nuova presentazione dal database MongoDB al caricamento delle pagine HTML;
- relazioni da **Model::SlideShow** che utilizza la rappresentazione locale della presentazione.

4.1.8.2 Registration

Tipo, obiettivo e funzione del componente: Classe, fornisce le funzionalità di registrazione.

Relazioni d'uso di altre componenti:

- relazione verso **Server** per la registrazione dell'utente presso il database MongoDB
- relazione da **Controller** da cui riceve in input i parametri dell'utente per la registrazione

4.1.9 Model::serverRelations:loader

Loader
-toInsert : object -toUpdate : object -toDelete : object
+update() : bool +addInsert(idElement : string) : bool +addUpdate(idElement : string) : bool +addDelete(idElement : string) : bool

Fig 7: serverRelation::Loader

Tipo, obiettivo e funzione del componente: package, racchiude le funzioni di recupero o creazione di una presentazione dal Server_g attraverso i servizi offerti dalla Api, una volta ottenuta la presentazione e' esposta per le modifiche provenienti da altri package nel Model

Relazioni d'uso di altre componenti:

- relazione verso **Server** per la modifica della presentazione nel database MongoDB
- relazione da **Model::SlideShow::InsertEditRemove** a cui viene esposta la rappresentazione della presentazione locale per essere modificata

4.1.9.1 Loader

Tipo, obiettivo e funzione del componente: Classe la cui Funzione_g è esporre una interfaccia per la sincronizzazione delle modifiche della presentazione nel model verso il server

Relazioni d'uso di altre componenti:

- relazione verso **Server** per il recupero della presentazione dal database MongoDB
- relazione da **Model::SlideShow::InsertEditRemove** a cui viene esposta la rappresentazione della presentazione locale per essere modificata

4.1.9.2 FileServerRelation

Tipo, obiettivo e funzione del componente: Classe che si interfaccia con il Server_g per l'upload, la gestione e il recupero di informazioni dei File_g multimediali presenti nello spazio dell'utente

Relazioni d'uso di altre componenti:

- dipendenza verso **Server** per recupero informazioni sui File_g e upload e gestione di nuovi File_g verso nello spazio utente
- dipendenza verso **accessControll** per il recupero del token per accedere ai servizi protetti del Server_g
- dipendenza da **Controller** da cui vengono chiamati i metodi esposti

4.1.9.3 MongoRelation

Tipo, obiettivo e funzione del componente: Classe che si interfaccia con il Server_g per la gestione delle presentazioni salvate in formato json su un database MongoDB **Relazioni d'uso di altre componenti:**

- dipendenza verso **Server** per l'interazione con il database MongoDB
- dipendenza verso **accessControll** per il recupero del token per accedere ai servizi protetti del Server_g
- dipendenza da **Loader** da cui vengono chiamati i metodi esposti

4.2 View

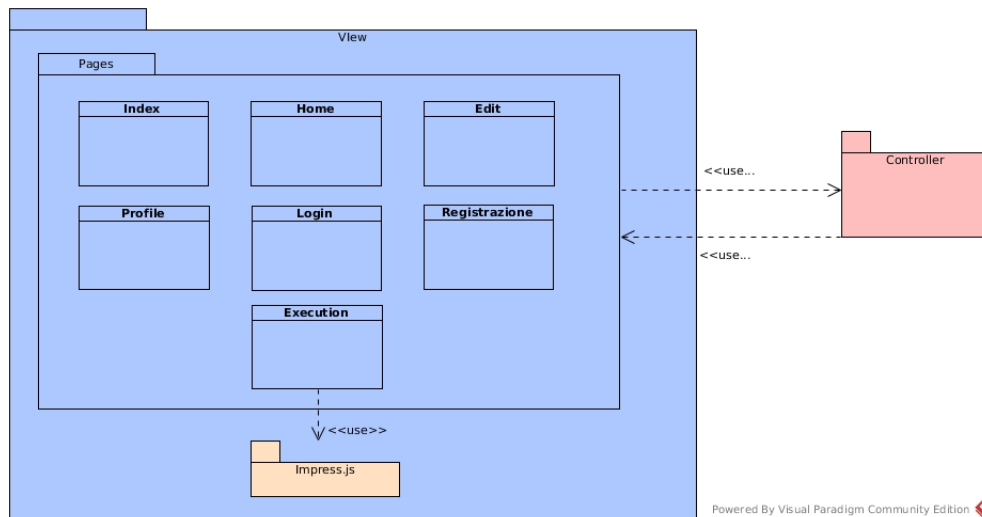


Fig 8: View

Tipo, obiettivo e funzione del componente: questo livello costituisce l'interfaccia del Software_g utilizzabile dagli utenti mediante pagine WEB_g.

Relazioni d'uso di altre componenti: il componente è costituito dal package Pages e comunica con il Controller per rendere possibile la gestione del proprio profilo, la gestione delle presentazioni e per controllare i dati in transito per il sistema, dovuti all'interazione dell'utente con lo stesso e la comunicazione con il Controller.

4.2.1 View::Pages

Tipo, obiettivo e funzione del componente: questo package costituisce le pagine fisiche del sistema, realizzate in HTML.

Relazioni d'uso di altre componenti: il componente comunica con il package Premi:-Controller per l'utilizzo delle funzioni_g presenti all'interno dello stesso per l'interazione dell'utente con il sito.

4.2.2 View::Pages::Index

Tipo, obiettivo e funzione del componente: la classe Index definisce la struttura, e la conseguente visualizzazione, della pagina WEB_g che consente ad un utente di effettuare Login_g e registrazione al sistema.

Relazioni d'uso di altre componenti: la classe Index utilizza i metodi messi a disposizione dalla classe Controller::HeaderController per effettuare il Logout_g o il reindirizzamento alle pagine Home e Profile.

Attività svolte e dati trattati: la classe definisce la struttura della pagina WEB_g comune a tutte le altre pagine.



4.2.3 View::Pages::Login

Tipo, obiettivo e funzione del componente: la classe Login_g definisce la struttura, e la conseguente visualizzazione, della pagina WEB_g che consente ad un utente di effettuare il Login_g al sistema.

Relazioni d'uso di altre componenti: la classe Login_g utilizza i metodi messi a disposizione dalla classe Controller::AuthenticationController per verificare i dati inseriti, per inviare i dati relativi alla Login_g e per visualizzare eventuali errori.

Attività svolte e dati trattati: la classe definisce la struttura della pagina WEB_g che consente agli utenti di autenticarsi al sistema. Essa resta in attesa che un utente inserisca i dati necessari per l'autenticazione al sistema.

4.2.4 View::Pages::Registrazione

Tipo, obiettivo e funzione del componente: la classe Registrazione definisce la struttura, e la conseguente visualizzazione, della pagina WEB_g che consente ad un utente di effettuare la registrazione al sistema.

Relazioni d'uso di altre componenti: la classe Registrazione utilizza i metodi messi a disposizione dalla classe Controller::AuthenticationController per verificare i dati inseriti, per inviare i dati relativi alla registrazione e per visualizzare eventuali errori.

Attività svolte e dati trattati: la classe definisce la struttura della pagina WEB_g che consente agli utenti di registrarsi al sistema. Essa resta in attesa che un utente inserisca i dati necessari per la registrazione al sistema.

4.2.5 View::Pages::Home

Tipo, obiettivo e funzione del componente: la classe Home definisce la struttura, e la conseguente visualizzazione, della pagina WEB_g che mostra ad un utente le presentazioni presenti sul Server_g e i comandi principali per gestirle.

Relazioni d'uso di altre componenti: la classe Home utilizza i metodi messi a disposizione dalla classe Controller::HomeController per l'eliminazione delle presentazioni dal Server_g, la loro rinominazione o la creazione di una nuova.

Attività svolte e dati trattati: la classe definisce la struttura della pagina WEB_g che consente agli utenti di visualizzare una lista delle proprie presentazioni, crearne di nuove, modificarle, eliminarle, scaricarle, eseguirle o modificarle.

4.2.6 View::Pages::Profile

Tipo, obiettivo e funzione del componente: la classe Profile definisce la struttura della pagina WEB_g che consente agli utenti di modificare i propri dati di profilo e gestire i File_g media caricati nel Server_g.

Relazioni d'uso di altre componenti: la classe Profile utilizza i metodi messi a disposizione dalla classe Controller::ProfileController, per la modifica della password.

Attività svolte e dati trattati: la classe Profile definisce la struttura, e la conseguente visualizzazione, della pagina WEB_g che mostra ad un utente i dati del proprio profilo e la possibilità di modificarli.

4.2.7 View::Pages::Execution

Tipo, obiettivo e funzione del componente: la classe Execution definisce la struttura, e la conseguente visualizzazione, della pagina WEB_g che mostra ad un utente l'esecuzione di una presentazione.

Relazioni d'uso di altre componenti: questa classe è gestita dal Framework_g esterno Impress.js; utilizza i metodi messi a disposizione della classe Controller::ExecutionController per creare la pagina che verrà eseguita da Impress.js.

Attività svolte e dati trattati: la classe definisce la struttura della pagina WEB_g che consente agli utenti di eseguire la presentazione spostandosi con la tastiera avanti e indietro, passare al capitolo successivo oppure selezionare un nuovo Percorso_g.

4.2.8 View::Pages::Edit

Tipo, obiettivo e funzione del componente: la classe Edit definisce la struttura, e la conseguente visualizzazione, della pagina WEB_g che mostra l'Editor_g di modifica di una presentazione.

Relazioni d'uso di altre componenti: la classe Edit utilizza i metodi messi a disposizione dalla classe Controller::EditController per caricare la presentazione da modificare, per l'inserimento di nuovi elementi_g, per il loro spostamento ed eliminazione, per le modifiche effettuate agli elementi_g e per cambiare il Percorso_g della presentazione.

Attività svolte e dati trattati: la classe definisce la struttura della pagina WEB_g che consente agli utenti di modificare una presentazione (inserendo, spostando, modificando o eliminando elementi_g), cambiare il Percorso_g e assegnare Bookmark_g ai Frame_g.

La classe dovrà predisporre delle apposite funzioni_g JavaScript_g per la gestione degli elementi_g nella view.

4.2.9 View::Pages::HomeOffline

Tipo, obiettivo e funzione del componente: la classe Manifest definisce la struttura, e la conseguente visualizzazione, della pagina WEB_g che mostra all'utente le presentazioni salvate in locale, permettendone l'esecuzione.

Attività svolte e dati trattati: la classe definisce la struttura della pagina WEB_g che consente agli utenti di visualizzare le presentazioni salvate in locale e permette la loro esecuzione.

4.2.10 View::Pages::ExecutionOffline

Tipo, obiettivo e funzione del componente: la classe Execution definisce la struttura, e la conseguente visualizzazione, della pagina WEB_g che mostra ad un utente l'esecuzione di una presentazione salvata in locale.

Relazioni d'uso di altre componenti: questa classe è gestita dal Framework_g esterno Impress.js; utilizza i metodi messi a disposizione della classe Controller::OfflineController per creare la pagina che verrà eseguita da Impress.js.

Attività svolte e dati trattati: la classe definisce la struttura della pagina WEB_g che consente agli utenti di eseguire la presentazione spostandosi con la tastiera avanti e indietro, passare al capitolo successivo oppure selezionare un nuovo Percorso_g.

4.3 Controller

Tipo, obiettivo e funzione del componente: fanno parte di questo livello i package che gestiscono i segnali e le chiamate effettuati dalla view.

Relazioni d'uso di altre componenti: comunica con il Model per rendere possibile la gestione del profilo e la gestione delle presentazioni da parte dell'utente.

4.3.1 Controller::EditController

Tipo, obiettivo e funzione del componente: lo scopo di questa classe è di gestire i segnali e le chiamate provenienti dalla pagina View::Pages::Edit.

Relazioni d'uso di altre componenti:

- Tutte le seguenti classi, appartenenti al package `Model::SlideShow::SlideShowActions::Command`:
 - `Invoker` <- `EditController` costruisce l'oggetto `Invoker`, gli passa un oggetto di classe `Command` eseguendo e annullando tale comando;
 - `ConcreteTextInsertCommand` <- `EditController` costruisce un comando e lo dà in pasto a `Invoker`;
 - `ConcreteFrameInsertCommand` <- `EditController` costruisce un comando e lo dà in pasto a `Invoker`;
 - `ConcreteImageInsertCommand` <- `EditController` costruisce un comando e lo dà in pasto a `Invoker`;
 - `ConcreteSVGInsertCommand` <- `EditController` costruisce un comando e lo dà in pasto a `Invoker`;
 - `ConcreteAudioInsertCommand` <- `EditController` costruisce un comando e lo dà in pasto a `Invoker`;
 - `ConcreteVideoInsertCommand` <- `EditController` costruisce un comando e lo dà in pasto a `Invoker`;
 - `ConcreteBackgroundInsertCommand` <- `EditController` costruisce un comando e lo dà in pasto a `Invoker`;
 - `ConcreteTextRemoveCommand` <- `EditController` costruisce un comando e lo dà in pasto a `Invoker`;
 - `ConcreteFrameRemoveCommand` <- `EditController` costruisce un comando e lo dà in pasto a `Invoker`;
 - `ConcreteImageRemoveCommand` <- `EditController` costruisce un comando e lo dà in pasto a `Invoker`;
 - `ConcreteSVGRemoveCommand` <- `EditController` costruisce un comando e lo dà in pasto a `Invoker`;
 - `ConcreteAudioRemoveCommand` <- `EditController` costruisce un comando e lo dà in pasto a `Invoker`;
 - `ConcreteVideoRemoveCommand` <- `EditController` costruisce un comando e lo dà in pasto a `Invoker`;



- Interfacce con e relazioni d'uso e da altre componenti:** EditController richiama le funzioni JavaScript fornite da View::Pages::Edit per la modifica della view. Successivamente istanzia un oggetto di una sottoclasse di Command e lo dà in pasto a Invoker e successivamente richiama il metodo corretto di Loader per il salvataggio nel database. Nel caso di un annullamento di una modifica o di un suo ripristino, EditController richiama il metodo undo() (o redo()) di Invoker il quale a sua volta, richiama il metodo corretto di EditController per l'aggiornamento della view.



4.3.2 Controller::ExecutionController

Tipo, obiettivo e funzione del componente: lo scopo di questa classe è di gestire i segnali provenienti dalla pagina View::Pages::Execution.

Relazioni d'uso di altre componenti:

- Controller::Services::SharedData -> ExecutionController ricava la presentazione corrente da SharedData;
- Controller::Services::toPages <- Quando la view invia una richiesta di reindirizzamento alla pagina View::Pages::Home o View::Pages::Edit, HeaderComponent invoca il metodo appropriato di toPages.

Interfacce con e relazioni d'uso e da altre componenti: la view invia a ExecutionController una richiesta di reindirizzamento ad una pagina oppure per ricavare la presentazione corrente. ExecutionController richiama il metodo appropriato di toPages, se la richiesta è un reindirizzamento, oppure di SharedData.

4.3.3 Controller::HomeOfflineController

Tipo, obiettivo e funzione del componente: lo scopo di questa classe è di gestire i segnali provenienti dalla pagina View::Pages::HomeOffline.

Relazioni d'uso di altre componenti:

- Controller::HomeController <- HomeController invoca i metodi necessari al salvataggio in locale di presentazioni ed eliminazione di presentazioni già salvate in locale.

Interfacce con e relazioni d'uso e da altre componenti: HomeController invia a HomeOfflineController richieste di salvare una presentazione in locale o di eliminarne una di quelle già presenti. HomeOfflineController invoca HomeController per ricavare le presentazioni salvate in locale. HomeController richiama il metodo appropriato di mongoRelation per soddisfarla.

4.3.4 Controller::OfflineExecutionController

Tipo, obiettivo e funzione del componente: lo scopo di questa classe è di gestire i segnali provenienti dalla pagina View::Pages::ExecutionOffline.

Interfacce con e relazioni d'uso e da altre componenti: la view invia a ExecutionController richiesta di generare la presentazione richiesta.

4.3.5 Controller::HeaderController

Tipo, obiettivo e funzione del componente: lo scopo di questa classe è di gestire i segnali e le chiamate provenienti dalla pagina View::Pages::Index.

Relazioni d'uso di altre componenti:

- Controller::Services::Main <- Quando la view invia una richiesta di Logout_g, HeaderComponent invoca il metodo per la deautenticazione fornito da Main;

- `Controller::Services::toPages` <- Quando la view invia una richiesta di reindirizzamento alla pagina `View::Pages::Home` o `View::Pages::Profile`, `HeaderController` invoca il metodo appropriato di `toPages`.

Interfacce con e relazioni d'uso e da altre componenti: la view invia a HeaderComponent una richiesta di Logout_g o di reindirizzamento ad una pagina. HeaderComponent richiama il metodo per il Logout_g di Main, oppure, se è un reindirizzamento, richiama il metodo appropriato di toPages.

4.3.6 Controller::AuthenticationController

Tipo, obiettivo e funzione del componente: lo scopo di questa classe è di gestire i segnali e le chiamate provenienti dalle pagine View::Pages::Login_g e View::Pages::Registrazione.

Relazioni d'uso di altre componenti:

- Controller::Services::Main <- Quando la view invia una richiesta di autenticazione, Logout o registrazione, HeaderComponent invoca il metodo corretto fornito da Main;

Interfacce con e relazioni d'uso e da altre componenti: la view invia a Authentication-Controller una richiesta di registrazione o autenticazione. AuthenticationController richiama il metodo appropriato di Main.

4.3.7 Controller::ProfileController

Tipo, obiettivo e funzione del componente: lo scopo di questa classe è di gestire i segnali e le chiamate della pagina View::Pages::Profile.

Relazioni d'uso di altre componenti:

- `Controller::Services::Main` <- Quando la view invia una richiesta di cambio della password, viene invocato il metodo per il cambio della password di Main.

Interfacce con e relazioni d'uso e da altre componenti: la pagina Profile invia a ProfileController la richiesta di cambio password. ProfileController richiama il metodo appropriato di Main.

4.3.8 Controller::HomeController

Tipo, obiettivo e funzione del componente: lo scopo di questa classe è di gestire i segnali e le chiamate provenienti dalla pagina View::Pages::Home.

Relazioni d'uso di altre componenti:

- Model::serverRelation::mongoRelation <- HomeController invoca i metodi necessari per il recupero di tutte le presentazioni dell'utente, la creazione di una nuova, la rinominazione o la cancellazione di una presentazione.

Interfacce con e relazioni d'uso e da altre componenti: la pagina Home invia a HomeController una richiesta. HomeController, in base al tipo di richiesta (creazione nuova presentazione, rinominazione, eliminazione o ottenimento della lista delle presentazioni) richiama il metodo appropriato di mongoRelation per soddisfarla.

4.3.9 Controller::Services

Tipo, obiettivo e funzione del componente: lo scopo di questa classe è di gestire le principali funzioni_g dell'applicazione, a partire dall'autenticazione fino ad arrivare all'upload dei File_g nel Server_g.

Relazioni d'uso di altre componenti: comunica con il Model per svolgere le operazioni necessarie.

4.3.9.1 Services::toPages

Tipo, obiettivo e funzione del componente: lo scopo di questa classe è di gestire i reindirizzamenti alle pagine corrette.

Relazioni d'uso di altre componenti:

- `/private` <- `toPages` invia una richiesta http al `Serverg`, il quale controlla l'esistenza del token per le pagine in cui è richiesta l'autenticazione.

Interfacce con e relazioni d'uso e da altre componenti: toPages invia una richiesta http al Server_g per il reindirizzamento alla pagina corretta. Nel caso in cui la pagina richieda di essere autenticati, viene inviato anche il token di sessione per verificare l'effettiva autenticazione.

4.3.9.2 Services::Upload

Tipo, obiettivo e funzione del componente: lo scopo di questa classe è di permettere l'upload dei File_g media nel Server_g.

Relazioni d'uso di altre componenti:

- `/private/api/files/image/[filename]` <- Upload invia una richiesta http al Server_g per effettuare l'upload del File_g immagine filename;
- `/private/api/files/video/[filename]` <- Upload invia una richiesta http al Server_g per effettuare l'upload del File_g video filename;
- `/private/api/files/audio/[filename]` <- Upload invia una richiesta http al Server_g per effettuare l'upload del File_g audio filename.

Interfacce con e relazioni d'uso e da altre componenti: Upload invia una richiesta http al Server_g per il caricamento di un File_g media nel Server_g, inviando anche il token di sessione per verificare l'effettiva autenticazione.

4.3.9.3 Services::Main

Tipo, obiettivo e funzione del componente: lo scopo di questa classe è di permettere le funzioni di base dell'applicazione, tra cui l'autenticazione al Server.

Relazioni d'uso di altre componenti:

- `Model::serverRelation::accessControl::Authentication` <- Main richiama `Authentication` per inviare una richiesta di autenticazione o di `Logoutg` al `Serverg`;
- `Model::serverRelation::accessControl::Registration` <- Main richiama `Registration` per inviare una richiesta di registrazione di un nuovo utente al `Serverg`;

- `Model::serverRelation::accessControl::ChangePassword` <- Main richiama `ChangePassword` per inviare una richiesta di cambio password al `Serverg`.

Interfacce con e relazioni d'uso e da altre componenti: Main richiama il metodo corretto di accessControl in modo da inviare una richiesta http al Server_g per effettuare l'autenticazione, la registrazione o il cambio della password.

4.3.9.4 Services::SharedData

Tipo, obiettivo e funzione del componente: lo scopo di questa classe è di mantenere in memoria la presentazione corrente.

Relazioni d'uso di altre componenti:

- `Model::serverRelation::mongoRelation` <- `SharedData` richiama `mongoRelation` per ottenere la presentazione corrente.

Interfacce con e relazioni d'uso e da altre componenti: SharedData richiama il metodo corretto di mongoRelation in modo da inviare una richiesta http al Server_g per ottenere la presentazione voluta.

4.3.9.5 Services::Utils

Tipo, obiettivo e funzione del componente: lo scopo di questa classe è definire delle funzioni utili a tutta l'applicazione.

Relazioni d'uso di altre componenti: data la sua natura, non comunica con nessun package.

5 REST API nodeApi

Il seguente diagramma delle classi è stato esteso con le primitive:

- «**Resource**» : rappresenta una Risorsa_g associata ad un certo URL_g a cui sono disponibili dei servizi
- «**Node**» : rappresenta una parte di URL_g a cui non sono disponibili servizi ma è utile per suddividere quest'ultimi
- «**Server**» : rappresenta la radice dei servizi offerti dal Server_g
- «**Path**» : indica una aggiunta in coda all' URL_g attuale per raggiungere una nuova Risorsa_g o nodo
- «**Middleware**» : indica un middleware, un insieme di funzionalità chiamate ogni qualvolta si accede a Risorse_g attraversando questo Elemento_g

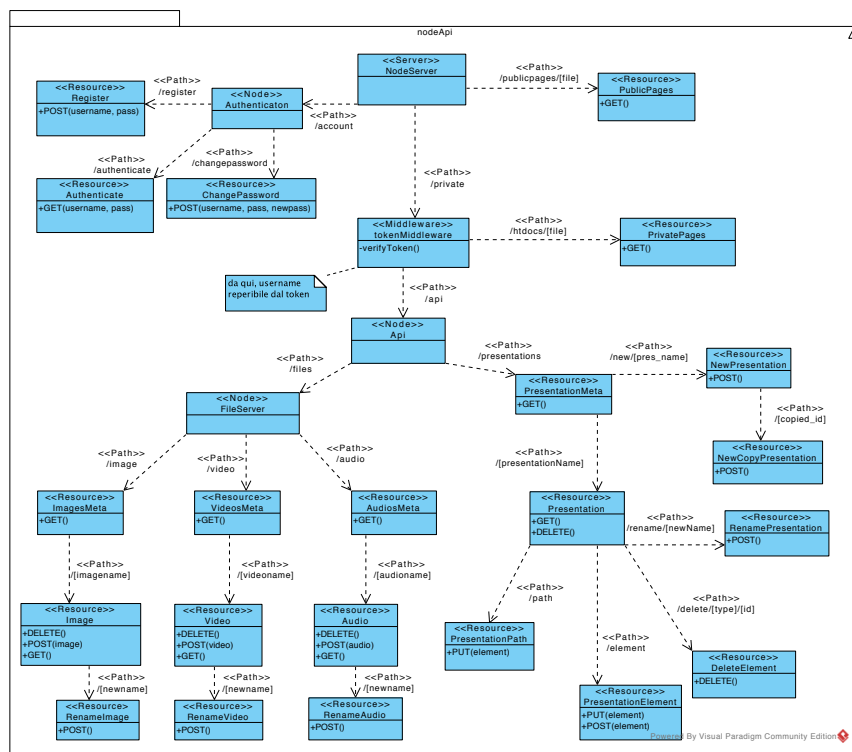


Fig 9: REST API nodeApi

- **NodeServer:** radice dei servizi offerti dal Server_g:
 1. Server_g per pagine HTML e File_g statici associati
 2. servizi di autenticazione stateless
 3. servizi di upload e reperimento File_g statici multimediali per utente
 4. servizi di interazione con MongoDB per salvataggio persistente delle presentazioni

- Register:

- **POST** /account/register
 - * **descrizione:** inserisce nuovo utente in MongoDB, crea una nuova collezione 'presentations'+username, crea le cartelle per i File_g utente

- **Authenticate:**

- **GET** /account/authenticate
 - * **descrizione:** verifica se username e password sono corretti e ritorna un token per l'accesso ai servizi protetti

- **ChangePassword:**

- **POST** /account/changepassword
 - * **descrizione:** verifica la correttezza di username e password e modifica questa'ultima con la nuova

- **PublicPages:**

- **GET** /publicpages/[file]
 - * **descrizione:** se presente [file] nella cartella /public_html del Server_g ritorna il File_g stesso

- **tokenMiddleware:** verifica che il token passato nel campo Authorization dell' Header sia valido, dal token ricava lo username dell'utente

- PrivatePages:

- **GET** /private/htdocs/[file]
 - * **descrizione:** se presente [file] nella cartella /private_html del Server_g ritorna il File_g stesso

- **PresentationMeta:**

- **GET** /private/api/presentations
 - * **descrizione:** cerca in MongoDB nella collezione associata alle presentazioni dell'utente, ritorna un array i cui elementi_g sono i campi meta delle presentazioni dell'utente

- **NewPresentation:**

- **POST** /private/api/presentations/new/[presentationName]
 - * **descrizione:** se non esiste già crea una nuova presentazione con il nome [presentationName]

- NewCopyPresentation:

- **POST** /private/api/presentations/new/[newPresentationName]/[oldPresentationName]

* **descrizione:** crea una nuova presentazione con nome [newPresentationName]
dalla presentazione con titolo [oldPresentationName]

- **Presentation:**

- **GET** /private/api/presentations/[presentationName]

* **descrizione:** recupera se presente la presentazione dell'utente associata al titolo passato nell'URL_g

- **DELETE** /private/api/presentations/[presentationName]

* **descrizione:** elimina se presente la presentazione dell'utente associata al titolo passato nell'URL_g

- **RenamePresentation:**

- **POST** /private/api/presentations/{presentationName}/rename/{newname}

* **descrizione:** rinomina se presente la presentazione dell'utente associata al titolo passato nell'URL_g con il nome [newname]

- **PresentationElement:**

- **POST** /private/api/presentations/[presentationName]/element

- * **descrizione:** inserisce nella presentazione dell'utente individuata da [presentationName] l'oggetto element passato nel body della richiesta

- **PUT** /private/api/presentations/{presentationName}/element

- * **descrizione:** sostituisce nella presentazione dell'utente l'Elemento_g passato nel body della richiesta

- **DeleteElement:**

- **DELETE** /private/api/presentations/[presentationName]/delete/[type/[id element]]

* **descrizione:** elimina dalla presentazione con il titolo [presentationName] l'Elemento_g con identificativo [idElement]

- **PresentationPath:**

- **PUT** /private/api/presentations/[presentationName]/path

- * **descrizione:** aggiorna l'elemento paths dalla presentazione con il titolo [presentationName] con il nuovo elemento paths passato come argomento nel corpo della richiesta

- **GetImage:**

- **GET** /files/[user]/image/[imagename]

* **descrizione:** ritorna il File_g[imagenam] nella cartella /users/[username]/image

- **GetAudio:**

- **GET** /files/[user]/audio/[audioname]



- * **descrizione:** ritorna il File_g [audioname] nella cartella /users/[username]/audios
- **GetVideo:**
 - **GET** /files/[user]/video/[videoname]
 - * **descrizione:** ritorna il File_g [videoname] nella cartella /users/[username]/videos
- **ImagesMeta:**
 - **GET** /private/api/files/image
 - * **descrizione:** ritorna un array con i nomi dei File_g immagine dell'utente
- **Image:**
 - **POST** /private/api/files/image/[imagename]
 - * **descrizione:** caricare da locale un nuovo File_g immagine nella cartella /users/[username]/images
 - **DELETE** /private/api/files/image/[imagename]
 - * **descrizione:** elimina il File_g immagine [imagename] dalla cartella /users/[username]/images
- **RenameImage:**
 - **POST** /private/api/files/image/[imagename]/[newname]
 - * **descrizione:** rinomina il File_g immagine [imagename] con [newname] nella cartella /users/[username]/images
- **AudiosMeta:**
 - **GET** /private/api/files/audio
 - * **descrizione:** ritorna un array con i nomi dei File_g audio dell'utente
- **Audio:**
 - **POST** /private/api/files/audio/[audioname]
 - * **descrizione:** caricare da locale un nuovo File_g immagine nella cartella /users/[username]/audios
 - **DELETE** /private/api/files/audio/[audioname]
 - * **descrizione:** elimina il File_g audio [audioname] dalla cartella /users/[username]/audios
- **RenameAudio:**
 - **POST** /private/api/files/audio/[audioname]/[newname]
 - * **descrizione:** rinomina il File_g audio [audioname] con [newname] nella cartella /users/[username]/audios
- **VideosMeta:**
 - **GET** /private/api/files/video

* **descrizione:** ritorna un array con i nomi dei File_g video dell'utente

- **Video:**

- **POST** /private/api/files/video/[videoname]

- * **descrizione:** caricare da locale un nuovo File_g immagine nella cartella /users/[username]/videos

- **DELETE** /private/api/files/video/[videoname]

* **descrizione:** elimina il File_g video [videoname] dalla cartella /users/[username]/videos

- **RenameVideo:**

- **POST** /private/api/files/video/[videoname]/[newname]

* **descrizione:** rinomina il File_g video [videoname] con [newname] nella cartella /users/[username]/videos

6 Diagrammi di attività

Vengono ora illustrati i diagrammi di attività che descrivono le interazioni dell'utente con Premi. È stato disegnato un diagramma ad alto livello che descrive le attività possibili, le quali vengono poi illustrate tramite dei sotto-diagrammi specifici.

6.1 Attività Principali

L'utente una volta aperto il Software_g Premi potrà loggarsi, registrarsi oppure accedere alla pagina per visualizzare le presentazioni scaricare in locale. Dopodiché l'utente potrà decidere se modificare la propria password, gestire, modificare o eseguire le proprie presentazioni oppure gestire il proprio profilo.

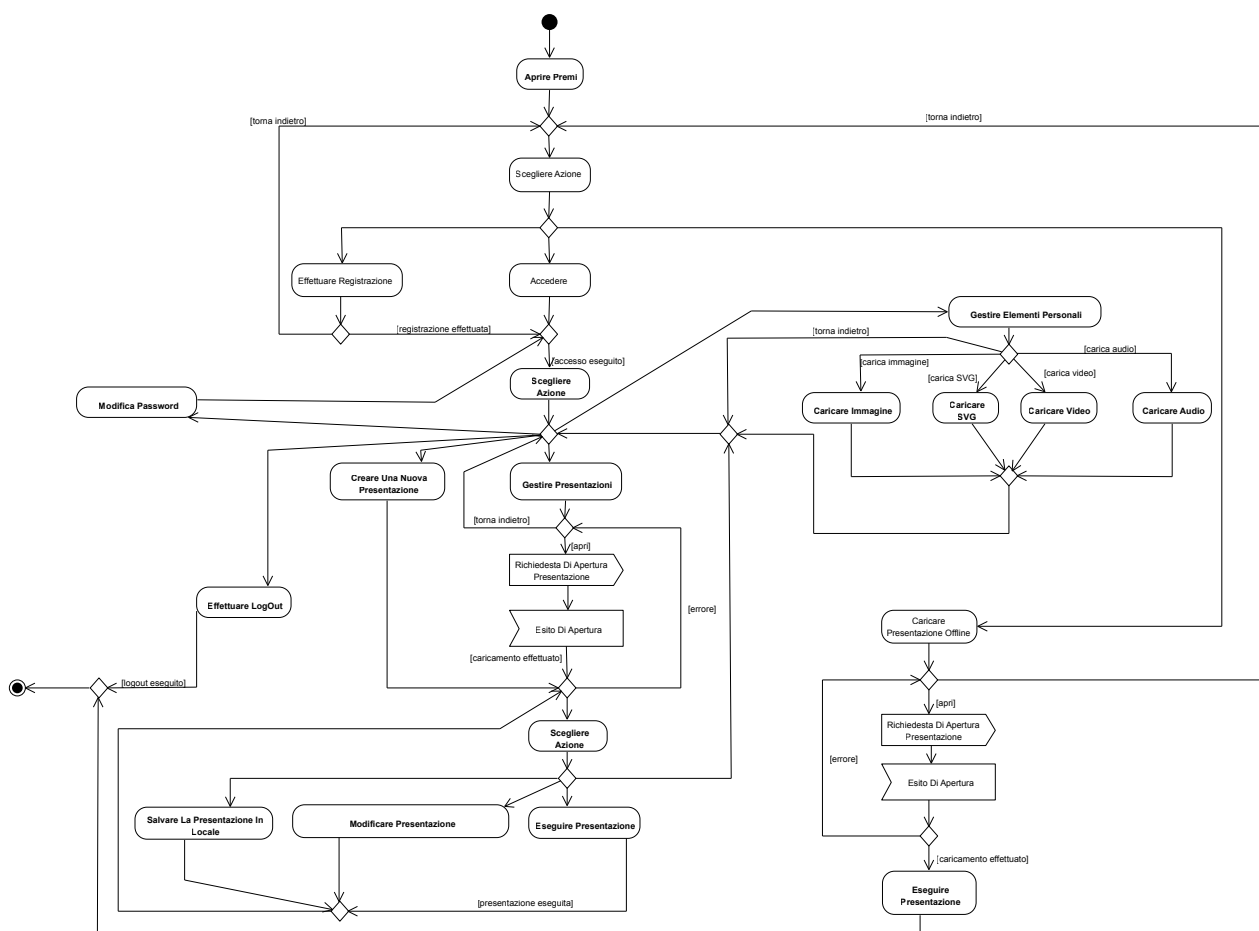


Fig 10: Attività Principali

6.1.1 Gestione presentazioni

L'utente una volta scelto di gestire le proprie presentazioni potrà rinominarle, aprirle o eliminarle.

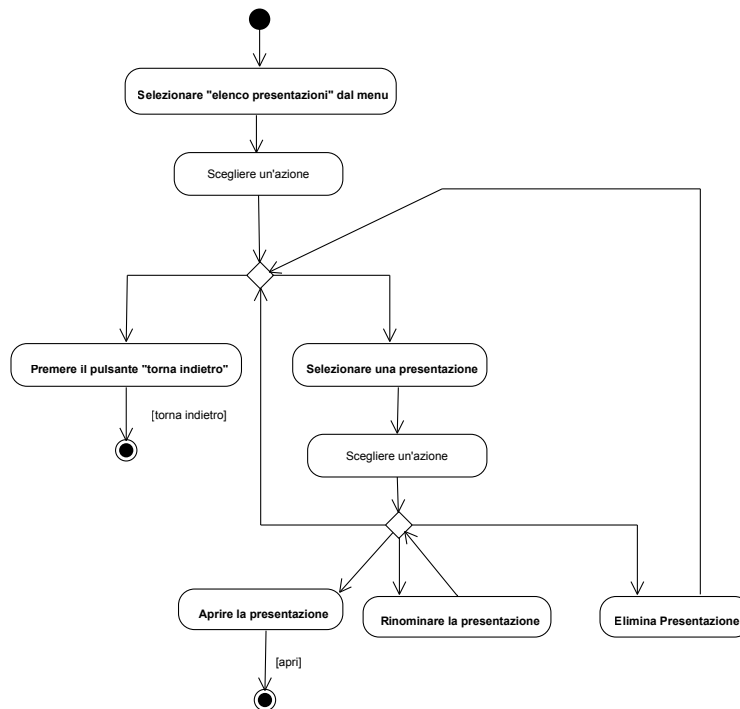


Fig 11: Gestione Presentazioni

6.1.2 Caricare File

L'utente una volta scelto di gestire il proprio profilo potrà caricare nuovi File_g all'interno del proprio spazio sul Server_g.

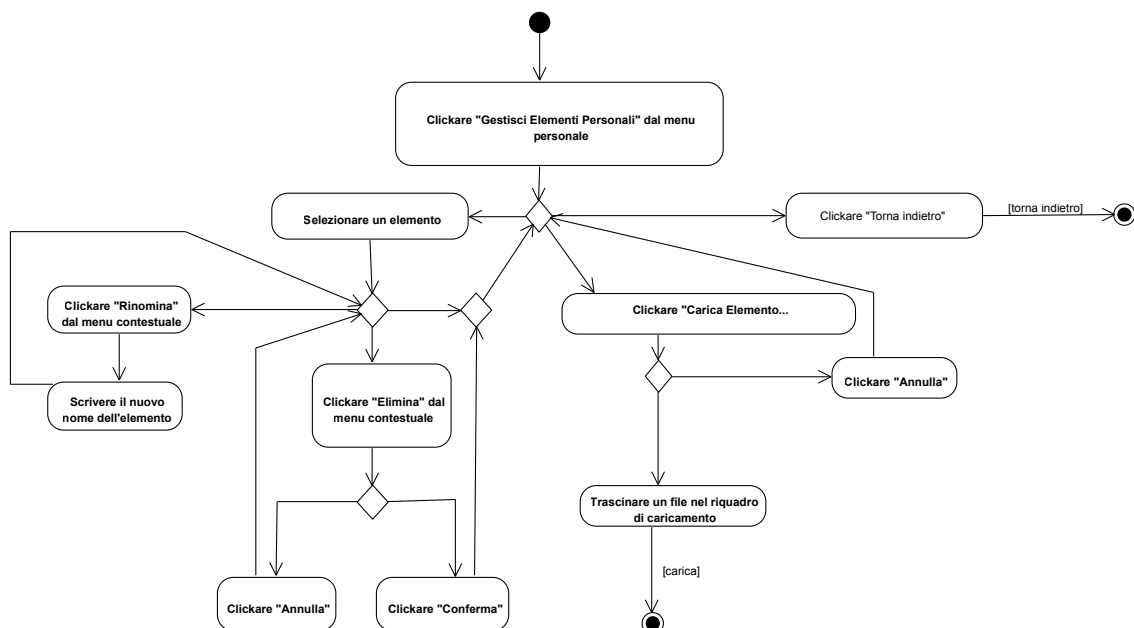


Fig 12: Caricare File

6.1.3 Modificare Presentazione da Desktop

L'utente una volta scelto di modificare una presentazione da mobile potrà decidere che tipo di modifiche apportare.

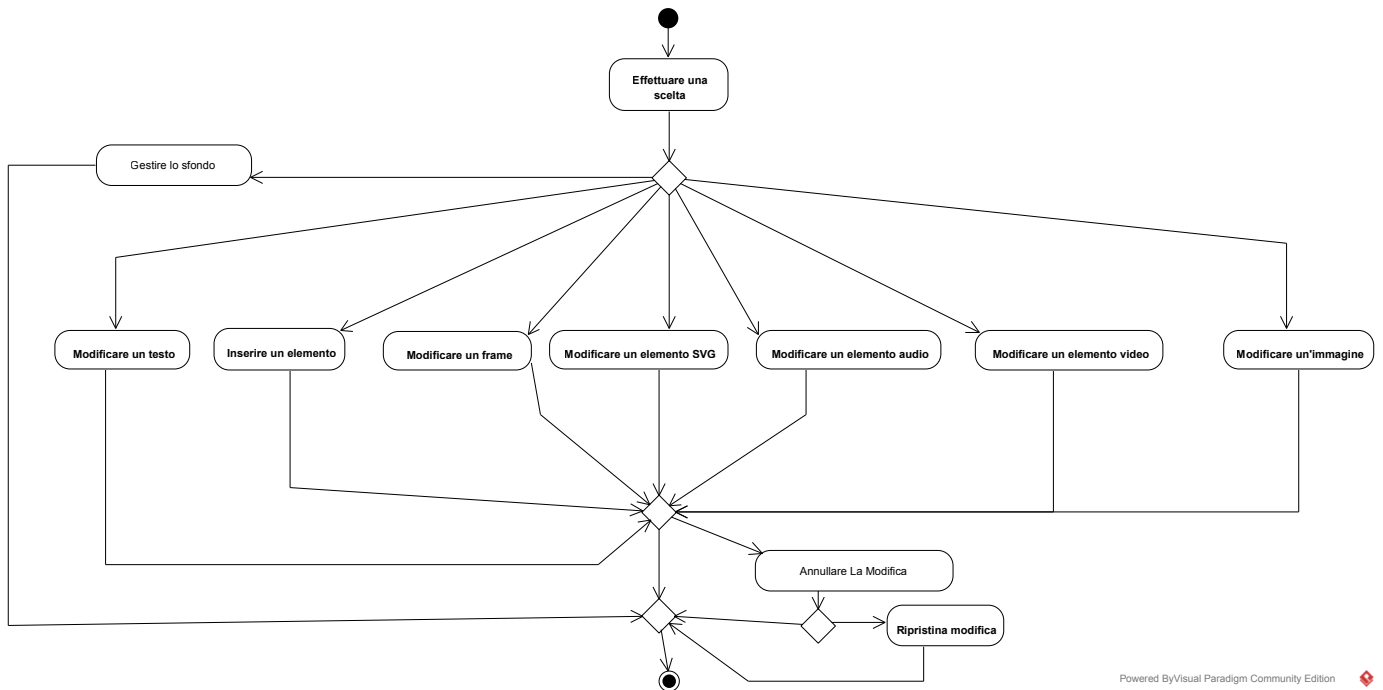


Fig 13: Modificare Presentazione da Desktop

6.1.4 Modificare Presentazione da Mobile

L'utente una volta scelto di modificare una presentazione da mobile potrà decidere che tipo di modifiche apportare.

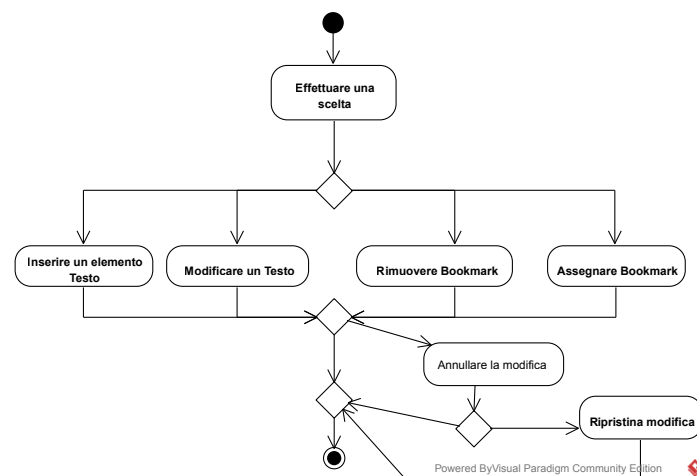


Fig 14: Modificare Presentazione da Mobile

6.1.5 Gestire Sfondo

L'utente una volta scelto di modificare una presentazione da mobile potrà decidere di apportare una modifica allo sfondo.



Fig 15: Gestire Sfondo

6.1.6 Inserire Elemento

L'utente una volta scelto di modificare una presentazione da mobile potrà decidere di inserire un nuovo Elemento_g sul piano della presentazione_g.

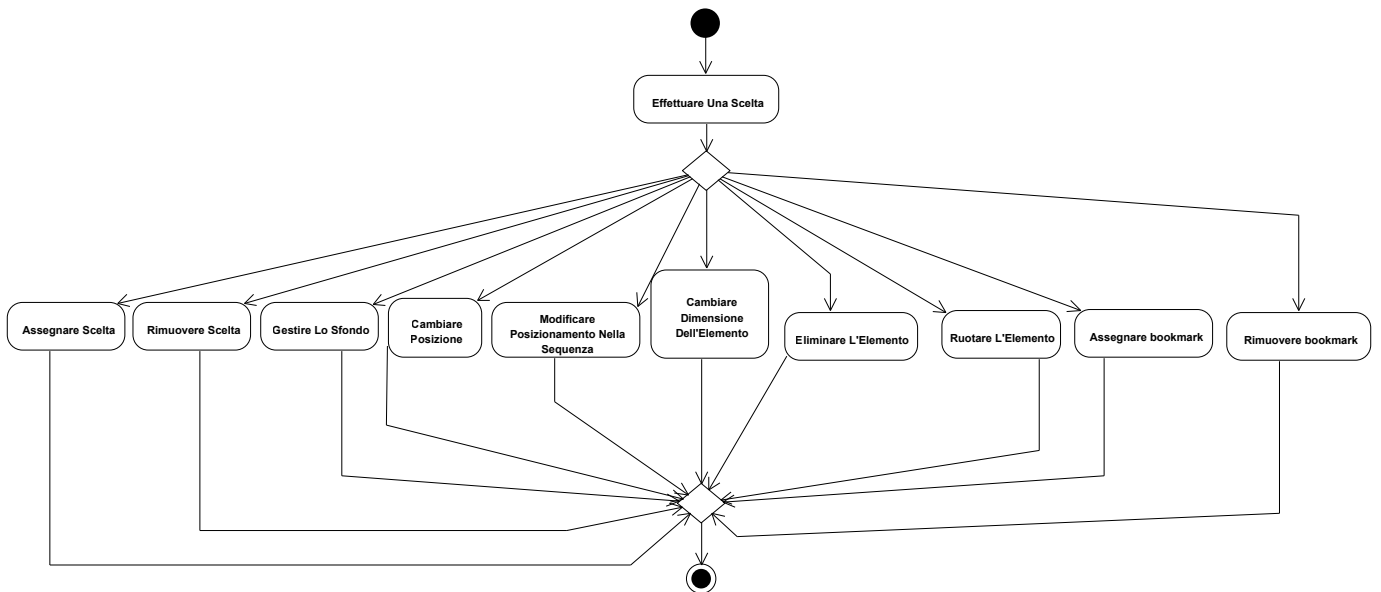


Fig 18: Modificare Frame

6.1.9 Modificare SVG

L'utente una volta scelto di modificare una presentazione da mobile potrà decidere di modificare un SVG selezionato.

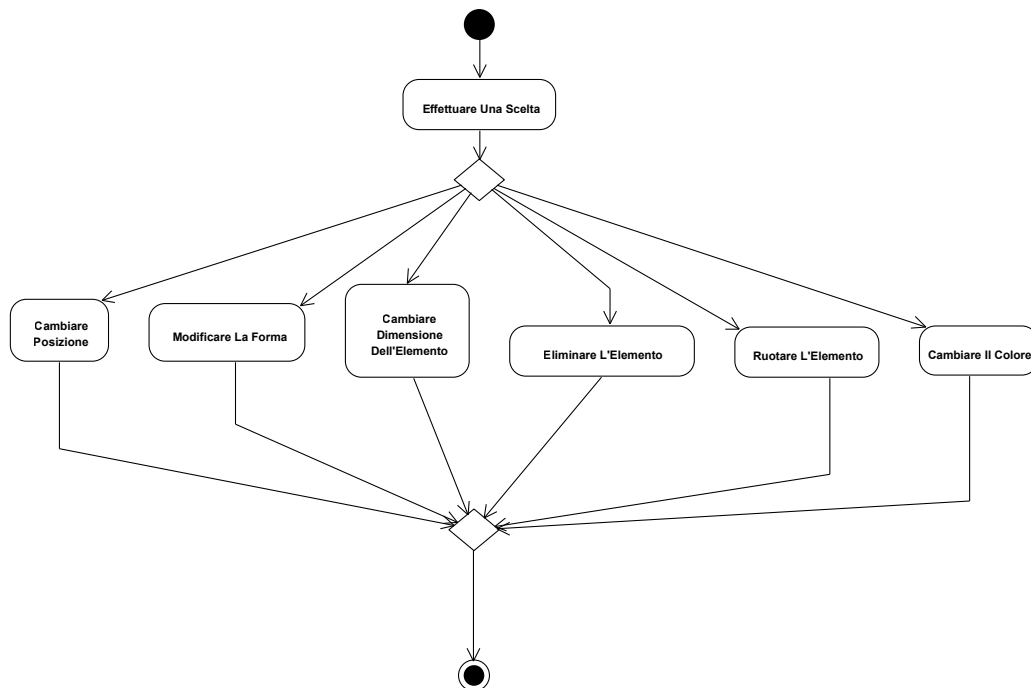


Fig 19: Modificare SVG



L'utente una volta scelto di modificare una presentazione da mobile potrà decidere di modificare un testo selezionato.





Per la parte di modifica delle presentazioni verranno utilizzati Javascript e il Framework_g Angular.js per lo spostamento in tempo reale degli elementi_g delle presentazioni. Infine è stato inoltre considerato l'utilizzo della tecnologia HTML5 Manifest per la gestione delle presentazioni offline.

8 Tracciamento dei Componenti coi Requisiti

8.1 Tracciamento Componenti-Requisiti

Tab 5: Tracciamento Componenti-Requisiti

Componente	Requisiti
Controller	RF 1, RF 3, RF 1.1, RF 1.2, RF 3.1, RF 3.2, RF 7, RF 7.1, RF 7.1.1, RF 7.4, RF 7.7, RF 7.7.1, RF 7.7.4, RF 7.7.7, RF 7.7.10, RF 7.7.13, RF 7.7.19, RF 7.7.25, RF 7.7.16, RF 7.7.28, RF 7.7.31, RF 7.7.34, RF 7.7.37, RF 7.7.40, RF 7.7.43, RF 7.10, RF 7.16, RF 7.13, RF 7.19, RF 7.19.1, RF 7.19.4, RF 7.19.10, RF 7.19.13, RF 7.28, RF 7.31, RF 7.34, RF 61, RF 61.1, RF 61.1.1, RF 61.1.4, RF 61.1.7, RF 61.1.10, RF 61.1.13, RF 61.1.16, RF 61.1.16.1, RF 61.1.16.4, RF 61.1.16.7, RF 61.1.16.10, RF 61.4, RF 61.4.1, RF 61.4.4, RF 61.4.7, RF 61.4.10, RF 61.7, RF 61.10, RF 61.4.10.1, RF 61.4.10.4, RF 61.4.10.7, RF 61.4.10.10, RF 1, RF 3, RF 1.1, RF 1.2, RF 3.1, RF 3.2, RF 10, RF 49, RF 3.2, RF 7, RF 64, RF 19, RF 34, RF 13, RF 43, RF 16, RF 17
- >Services	RF 1, RF 3, RF 1.1, RF 1.2, RF 3.1, RF 3.2, RF 7, RF 7.7.7, RF 7.7.13, RF 7.13
Model	
- >serverRelation	
- - >accessControl	

Componente	Requisiti
- - - >Authentication	RF 3, RF 43, RF 3.1, RF 3.2, RF 64
- - - >Registration	RF 1, RF 1.1, RF 1.2
- - >fileServerRelation	RF 16, RF 12, RF 17, RF 37
- - >Loader	RF 7, RF 7.1, RF 7.4, RF 7.7, RF 7.7.1, RF 7.7.4, RF 7.7.7, RF 7.7.10, RF 7.7.13, RF 7.7.19, RF 7.7.25, RF 7.7.16, RF 7.7.28, RF 7.7.34, RF 7.10, RF 7.16, RF 7.13, RF 7.19, RF 7.19.1, RF 7.37, RF 7.40, RF 7.40.1, RF 7.40.4, RF 7.43, RF 7.46, RF 7.7.46
- - >mongoRelation	RF 4, RF 7.1, RF 7.7.7, RF 7.7.13, RF 7.16, RF 7.13, RF 7.19.1, RF 7.19.4, RF 7.19.10, RF 7.19.13, RF 19, RF 34, RF 7.37, RF 35
- >SlideShow	
- - >SlideShowActions	
- - - >Command	RF 7.7.25, RF 7.19.4, RF 7.7.13, RF 7.43, RF 7.13, RF 7.43, RF 7.7.43, RF 7.22, RF 7.25, RF 10.5, RF 10.8, RF 7.7.4, RF 7.7.40, RF 7.16, RF 7.40.4, RF 7.19.13, RF 7.7.4, RF 7.7.19, RF 7.46, RF 7.7.46, RF 7.7.10, RF 7.7.16, RF 7.1, RF 7.1.1, RF 7.10, RF 7.7.7, RF 7.43, RF 7.7.25, RF 7.4, RF 7.7.28, RF 7.19.13, RF 7.37, RF 7.43, RF 7.7.1, RF 7.43, RF 7.7.13, RF 7.43, RF 55, RF 58

Università degli studi di Padova - 2014/2015



Università degli studi di Padova - 2014/2015

8.2 Tracciamento Requisiti-Componenti

Tab 6: Tracciamento Requisiti-Componenti

Requisito	Componenti
RF 1	View::Pages::Index, Model::serverRelation::accessControl::Registration, Controller
RF 1.1	View::Pages::Index, Model::serverRelation::accessControl::Registration, Controller
RF 1.2	View::Pages::Index, Model::serverRelation::accessControl::Registration, Controller
RF 3	View::Pages::Index, Model::serverRelation::accessControl::Authentication, Controller
RF 3.1	View::Pages::Index, Model::serverRelation::accessControl::Authentication, Controller
RF 3.2	View::Pages::Index, Model::serverRelation::accessControl::Authentication, Controller
RF 4	Model::serverRelation::mongoRelation
RF 7	View::Pages::Home, View::Pages::Edit, Model::serverRelation::Loader, Controller
RF 7.1	View::Pages::Edit, Model::SlideShow::SlideShowActions::InsertEditRemove, Model::SlideShow::SlideShowActions::Command, Model::SlideShow::SlideShowElements, Model::serverRelation::Loader, Controller, Model::serverRelation::mongoRelation
RF 7.1.1	View::Pages::Edit, Model::SlideShow::SlideShowActions::InsertEditRemove, Model::SlideShow::SlideShowActions::Command, Controller
RF 7.4	View::Pages::Edit, Model::serverRelation::Loader, Controller
RF 7.7	View::Pages::Edit, Model::serverRelation::Loader, Controller
RF 7.7.1	View::Pages::Edit, Model::SlideShow::SlideShowActions::InsertEditRemove, Model::SlideShow::SlideShowActions::Command, Model::SlideShow::SlideShowElements, Model::serverRelation::Loader, Controller:
RF 7.7.4	View::Pages::Edit, Model::SlideShow::SlideShowActions::Command, Model::serverRelation::Loader, Model::SlideShow::SlideShowActions::InsertEditRemove, Controller

Università degli studi di Padova - 2014/2015



Università degli studi di Padova - 2014/2015

Requisito	Componenti
RF 10.1	
RF 10.4	
RF 10.5	Model::SlideShow::SlideShowActions::Command
RF 10.8	Model::SlideShow::SlideShowActions::Command
RF 12	Model::serverRelation::fileServerRelation
RF 13	View::Pages::Profile, Controller
RF 16	View::Pages::Profile, Controller, Model::serverRelation::fileServerRelation
RF 17	View::Pages::Profile, Controller, Model::serverRelation::fileServerRelation
RF 19	View::Pages::Home, Controller, Model::serverRelation::mongoRelation
RF 25	
RF 31	
RF 34	View::Pages::Home, Controller, Model::serverRelation::mongoRelation
RF 35	Model::serverRelation::mongoRelation
RF 36	
RF 37	Model::serverRelation::fileServerRelation
RF 43	View::Pages::Profile, Model::serverRelation::accessControl::Authentication, Controller
RF 46	
RF 49	View::Pages::Home, Controller
RF 52	View::Pages::Manifest
RF 55	Model::SlideShow::SlideShowActions::Command
RF 58	Model::SlideShow::SlideShowActions::Command
RF 61	View::Pages::Manifest, View::Pages::Execution, Controller
RF 61.1	View::Pages::Execution, Controller
RF 61.1.1	View::Pages::Execution, Controller
RF 61.1.4	View::Pages::Execution, Controller
RF 61.1.7	View::Pages::Execution, Controller
RF 61.1.10	View::Pages::Execution, Controller
RF 61.1.13	View::Pages::Execution, Controller

Requisito	Componenti
RF 61.1.16	View::Pages::Execution, Model::SlideShow::SlideShowElements, Controller
RF 61.1.16.1	View::Pages::Execution, Controller
RF 61.1.16.4	View::Pages::Execution, Controller
RF 61.1.16.7	View::Pages::Execution, Controller
RF 61.1.16.10	View::Pages::Execution, Controller
RF 61.4	View::Pages::Execution, Controller
RF 61.4.1	View::Pages::Execution, Controller
RF 61.4.4	View::Pages::Execution, Controller
RF 61.4.7	View::Pages::Execution, Controller
RF 61.4.10	View::Pages::Execution, Controller
RF 61.4.10.1	View::Pages::Execution, Controller
RF 61.4.10.4	View::Pages::Execution, Controller
RF 61.4.10.7	View::Pages::Execution, Controller
RF 61.4.10.10	View::Pages::Execution, Controller
RF 61.7	View::Pages::Execution, Controller
RF 61.10	View::Pages::Execution, Controller
RF 64	View::Pages::Home, Model::serverRelation::accessControl::Authentication, Controller
RF 67	
RF 67.1	
RF 67.4	
RF 67.7	
RF 67.10	
RF 67.13	
RF 70	



Università degli studi di Padova - 2014/2015

A Design Pattern e Pattern Architeturali

A.1 MVC

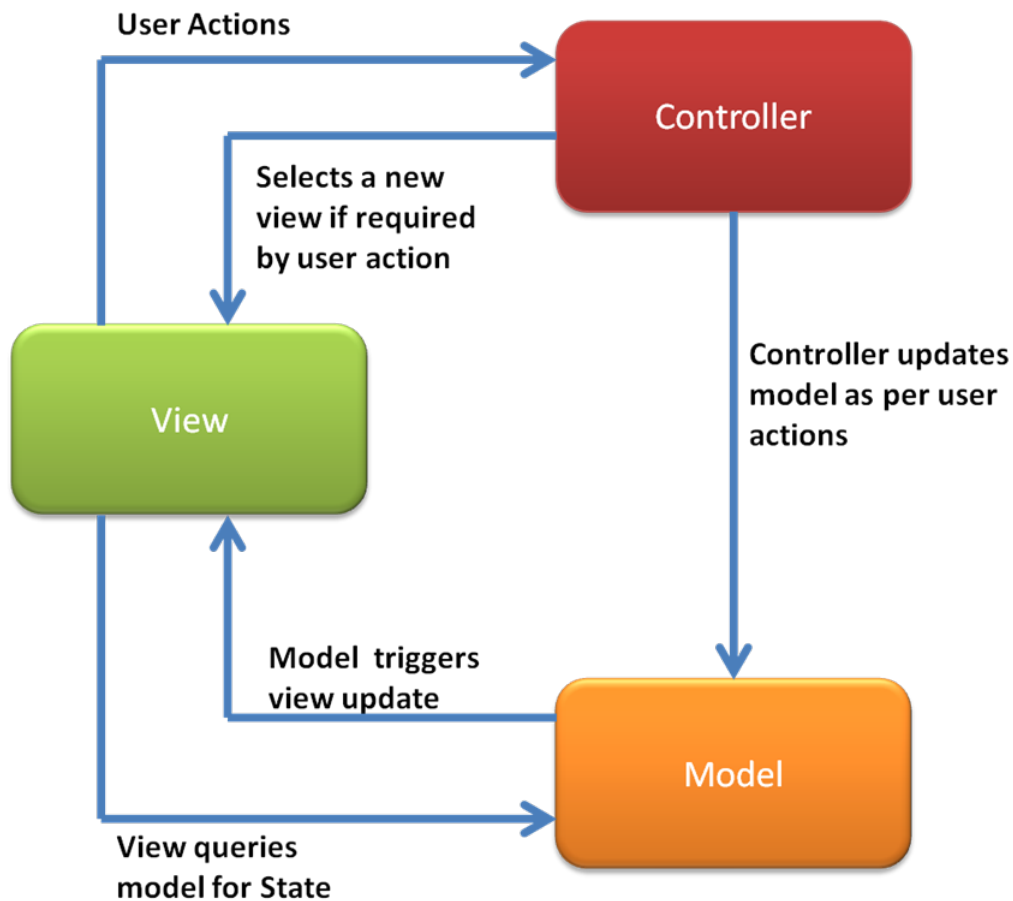


Fig 21: Model View Controller

- **Scopo dell'utilizzo:** è stato scelto il pattern MVC per separare la logica dell'applicazione dalla rappresentazione grafica;
- **Contesto d'utilizzo:** il pattern MVC viene utilizzato per l'architettura generale dell'applicazione. Ogni modifica effettuata dall'utente sulla View viene inviata al Model tramite il Controller e viceversa.

A.2 Command

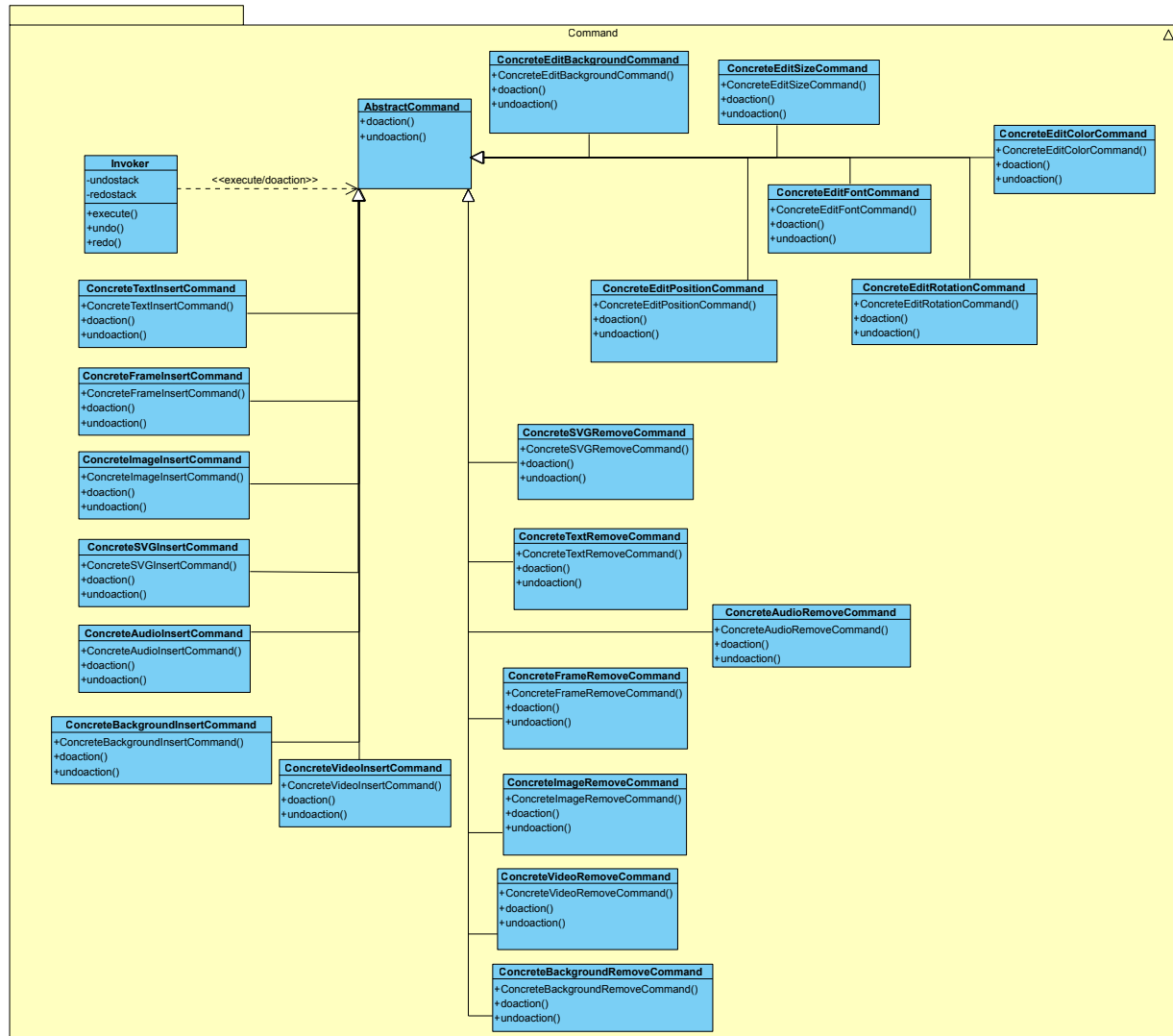


Fig 22: Diagramma delle classi del package Command



- **Scopo dell'utilizzo:** si è scelto di utilizzare il pattern Command perché poter accodare o mantenere uno storico delle operazioni e gestire operazioni cancellabili;
- **Contesto d'utilizzo:** viene utilizzato in fase di modifica delle presentazioni.

A.2.1 Premi::Model::SlideShow::SlideShowActions::Command

Premi::Controller::Presentazione::Edit può invocare il metodo `unexecute()` di `Invoker` che a sua volta invoca il metodo `AbstractCommand::undoCommand()` nell'ultimo oggetto inserito nel membro contenitore `undo`. Questo metodo esegue le operazioni necessarie per annullare tutte le modifiche apportate dal comando. Quindi `Invoker` toglie il comando dal contenitore `undo` e lo inserisce nel contenitore `redo`. Quando `Premi::Controller::Presentazione::Edit` invoca il metodo



Invoker::execute(), l'oggetto Invoker esegue il comando e lo sposta nuovamente dal membro contenitore redo e lo mette nel membro undo.