

18-05-2015



Specifica Tecnica

Informazioni sul documento

Nome Documento	Specifica Tecnica
Versione	1.0.0
Stato	<i>Formale</i>
Uso	<i>Esterno</i>
Data Creazione	18-05-2015
Data Ultima Modifica	18-05-2015
Redazione	Fossa Manuel, Petrucci Mauro
Approvazione	Tollot Pietro
Verifica	Gabelli Pietro
Lista distribuzione	<i>LateButSafe</i>
	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Proponente Zucchetti S.p.a.



Tab 1: Versionamento del documento

Versione	Autore	Data	Descrizione
1.0.0	Tollot Pietro	13-04-2015	Approvazione del documento
0.7.0	Petrucci Mauro	08-04-2015	Apportate le modifiche segnalate dal verificatore Fossa Manuel
0.3.0	Petrucci Mauro	25-03-2015	Aggiunta dei contenuti
0.2.0	Fossa Manuel	24-03-2015	Aggiunta dei contenuti
0.1.0	Busetto Matteo	20-03-2015	Stesura dello scheletro del documento



pre-RR

Tab 2: Storico ruoli pre-RR

Indice

1	Introduzione	7
1.1	Scopo del documento	7
1.2	Scopo del Prodotto	7
1.3	Glossario	7
1.4	Riferimenti	7
1.4.1	Normativi	7
1.4.2	Informativi	7
2	Strumenti	9
2.1	HTML	9
2.2	JavaScript	9
2.3	jQuery	10
2.4	Angular.js	10
2.5	Node.js	10
2.6	MongoDB	10
2.7	Impress.js	10
3	Design Pattern	11
3.1	MVC	11
3.2	Singleton	11
3.2.1	Premi::Model::Command::Invoker	11
3.2.2	Premi::Model::Presentazione::SlideShow	11
3.3	Utility	12
3.4	Builder	12
3.4.1	Premi::Model::Builder	12
3.5	Command	12
3.5.1	Premi::Model::Command	13
3.6	Iterator	13
3.7	Template Method	13
3.7.1	Premi::Model::Inserimento	14
3.8	Strategy	14
3.8.1	Premi::Model::Modifica	14
4	Descrizione architetturale	15
4.1	Metodo e formalismi	15
4.2	Architettura generale	15
4.2.1	Model	15
4.2.2	View	15
4.2.3	ViewModel	15
5	Descrizione dei singoli componenti	17
5.1	Model	17
5.1.1	Premi::Model::Inserimento	17
5.1.1.1	Premi::Model::Inserimento::Insertter	17



Università degli studi di Padova - 2014/2015

	5.1.6.8	Premi::Model::Builder::ConcreteVideoBuilder	36
5.1.7		Premi::Model::Caricamento	36
	5.1.7.1	Premi::Model::Caricamento::Uploader	37
	5.1.7.2	Premi::Model::Caricamento::ConcreteSvgUploader	37
	5.1.7.3	Premi::Model::Caricamento::ConcreteImageUploader	38
	5.1.7.4	Premi::Model::Caricamento:: ConcreteVideoUploader	38
	5.1.7.5	Premi::Model::Caricamento:: ConcreteAudioUploader	38
5.2		Controller	40
	5.2.1	Premi::Controller::Presentazione	40
		5.2.1.1 Premi::Controller::Presentazione::EditController	40
		5.2.1.2 Premi::Controller::Presentazione::HomeController	41
		5.2.1.3 Premi::Controller::Presentazione::ExecutionController	41
5.3		View	42
	5.3.1	Premi::View::Pages::IndexPage	42
	5.3.2	Premi::View::Pages::Home	43
	5.3.3	Premi::View::Pages::Manifest	43
	5.3.4	Premi::View::Pages::Profile	44
	5.3.5	Premi::View::Pages::Execution	44
	5.3.6	Premi::View::Pages::Edit	45
	5.3.7	Premi::View::Pages::EditDesktop	45
	5.3.8	Premi::View::Pages::EditMobile	46

Sommario

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



1.1 Scopo del documento

1.2 Scopo del Prodotto

1.3 Glossario

1.4 Riferimenti

1.4.1 Normativi

- ### 1.4.2 Informativi

- **Design Patterns: Elements of Reusable Object-Oriented Software**, 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/>;
- MongoDB: <http://docs.mongodb.org/manual/>;

- Node.js: <https://nodejs.org/documentation/>;
- Angular.js: <https://docs.angularjs.org/tutorial>;
- jQuery: <http://api.jquery.com/> ;
- Impress.js: <https://github.com/bartaz/impress.js/>.



2.1 HTML

- **Vantaggi:**

- **Svantaggi:**

- ## 2.2 JavaScript

Università degli studi di Padova - 2014/2015



2.7 Impress.js

Impress.js una libreria open-source che permette di visualizzare i 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 obbligatori relativi all'esecuzione senza dover scrivere ingenti quantità di codice aggiuntivo. Si è deciso inoltre di integrare nel framework alcune funzioni in modo da rispondere a tutti i requisiti obbligatori relativi all'esecuzione.

3 Design Pattern

(da integrare con figure di applicabilità in premi e classi che utilizzano quei design pattern)

3.1 MVC

- **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 Controller che invoca i metodi delle classi presenti nel Model.

3.2 Singleton

- **Descrizione:** è un design pattern di tipo creazionale utilizzato per imporre ad alcune classi di avere solamente un'istanza. Questo design pattern garantisce la consistenza dello stato dell'applicazione che si andrà a sviluppare.
Quando una classe implementa il pattern Singleton, la classe stessa si occupa di controllare il numero di istanze costruite (di solito tramite campi dati statici) e di fornire l'accesso ad esse tramite metodi appositi.
- **Scopo dell'utilizzo:** viene usato il pattern Singleton per le classi che devono avere un'unica istanza durante l'esecuzione dell'applicazione;
- **Contesto d'utilizzo:** le classi che devono avere un'unica istanza sono:
 - Invoker;
 - SlideShow.

3.2.1 Premi::Model::Command::Invoker

La prima volta in cui avviene una modifica al piano della presentazione `Premi::Controller::Edit` invoca il metodo `Invoker::getInstance()` che, tramite costruttore privato, costruisce `Invoker` e lo restituisce al controller che può quindi dargli in pasto i comandi per l'implementazione del Design Pattern Command.

3.2.2 Premi::Model::Presentazione::SlideShow

La prima volta in cui viene caricata una presentazione, `Premi::Model::Presentazione::Loader` invoca il metodo `SlideShow::getInstace()` che, tramite costruttore privato, costruisce `SlideShow`.

3.3 Utility

- ???

3.4 Builder

- **Descrizione:** Il design pattern Builder separa la costruzione di un oggetto complesso dalla sua rappresentazione cosicché il processo di costruzione stesso possa creare diverse rappresentazioni.
L'algoritmo per la creazione di un oggetto complesso è indipendente dalle varie parti che costituiscono l'oggetto e da come vengono assemblate. Ciò ha l'effetto immediato di rendere più semplice la classe, permettendo a una classe builder separata di focalizzarsi sulla corretta costruzione di un'istanza e lasciando che la classe originale si concentri sul funzionamento degli oggetti. Un builder permette anche di costruire un oggetto passo-passo, cosa che si può verificare quando si fa il parsing di un testo o si ottengono i parametri da un'interfaccia interattiva.
- **Scopo dell'utilizzo:** viene usato il pattern Builder per separare la costruzione di un oggetto dalla sua rappresentazione e poter riusare il processo di costruzione per creare rappresentazioni differenti;
- **Contesto d'utilizzo:** viene utilizzato per il caricamento delle presentazioni e degli oggetti in essi presenti.

3.4.1 Premi::Model::Builder

Il package `Premi::Model::Builder` implementa il Design Pattern Builder. `Premi::Model::Presentazione::Load` costruisce il `Premi::Model::Builder::Director` che ricopre il ruolo di director del Design Pattern. `Loader` passa a `Director` i parametri di istanziazione dell'oggetto di una sottoclasse di `Premi::Model::Presentazione::SlideShowElement`. `Director` invoca il costruttore della classe concreta di `AbstractBuilder` per istanziare un oggetto della sottoclasse di `SlideShowElement`, invoca quindi i metodi di `build` per settarne i campi.

3.5 Command

- **Descrizione:** viene utilizzato quando c'è la necessità di disaccoppiare l'invocazione di un comando dai suoi dettagli implementativi, separando colui che invoca il comando da colui che esegue l'operazione.

Tale operazione viene realizzata attraverso questa catena: Client->Invocatore->Ricevitore

Il Client non è tenuto a conoscere i dettagli del comando ma il suo compito è solo quello di chiamare il metodo dell' Invocatore che si occuperà di intermediare l'operazione. L'Invocatore ha l'obiettivo di incapsulare, nascondere i dettagli della chiamata come nome del metodo e parametri. Il Ricevitore utilizza i parametri ricevuti per eseguire l'operazione
- **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.



Il package `Premi::Model::Command` implementa il pattern `Command`, tuttavia il client è esterno al package ed è individuabile nella classe `Premi::Controller::Presentazione::Edit`, che invoca il costruttore delle sottoclassi di `Premi::Model::Command::AbstractCommand` e dà l'oggetto creato in pasto a `Premi::Model::Command::Invoker`, che rappresenta, appunto, la componente invoker del pattern e che mette l'oggetto della sottoclasse di `AbstractCommand` in un contenitore denominato `undo`, invoca quindi il metodo `Invoker::execute()` che a sua volta esegue concretamente il comando.

3.6 Iterator

- ### 3.7 Template Method

- Prima viene creata una classe che fornisce i passi base di un algoritmo. Questi passi sono implementati usando metodi astratti. Successivamente, le sottoclassi cambiano i metodi astratti per implementare l'algoritmo. Così facendo l'algoritmo generale è mantenuto valido, ma i passi concreti possono essere cambiati dalle sottoclassi.

- **Scopo dell'utilizzo:** il pattern Template Method viene usato per definire la struttura di un algoritmo e lasciare alle sottoclassi la definizione di alcune parti usate;
- **Contesto d'utilizzo:** viene utilizzato per l'inserimento e la rimozione degli elementi.



Le classi del package `Premi::Model::Inserimento` implementano il pattern `Template`. La classe astratta `Premi::Model::Inserimento::Inserter` definisce i metodi astratti per l'inserimento di elementi nella presentazione e descrive i passi dell'algoritmo comuni per l'inserimento di tutti i tipi di elementi. Le sottoclassi di `Inserter` specificano i metodi:

- In maniera del tutto simile al package Inserimento è organizzato il package Premi::Model::Eliminazione

- **Descrizione:** Strategy è un Design Pattern che permette che il comportamento di un algoritmo sia deciso a runtime. Il pattern Strategy definisce una famiglia di algoritmi, incapsula ogni algoritmo e rende gli algoritmi intercambiabili all'interno di quella famiglia. Strategy permette all'algoritmo di cambiare indipendentemente dal client che lo utilizza;
- **Scopo dell'utilizzo:** il pattern Strategy viene usato per isolare più algoritmi che svolgono la stessa funzione dal codice che esegue la funzione;
- **Contesto d'utilizzo:** viene utilizzato per la modifica degli elementi.

Il package `Premi::Model::Modifica` implementa il design pattern Strategy. `Premi::Model::Modifica::Editor` fornisce un metodo astratto `editElement` che viene definito in modo diverso in ognuna delle sue sottoclassi e invocato al momento della costruzione. In maniera del tutto simile a `Premi::Model::Modifica` è organizzato il package `Premi::Model::Caricamento`.



4.1 Metodo e formalismi

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

Per i diagrammi di Package, classi e attività verrà usata la notazione UML 2.(DA AGGIUNGERE INDICE).

Il prodotto si presenta suddiviso in tre parti distinte: Model, View e ViewModel. Si è quindi cercato di implementare il design pattern architetturale MVVM 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 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.

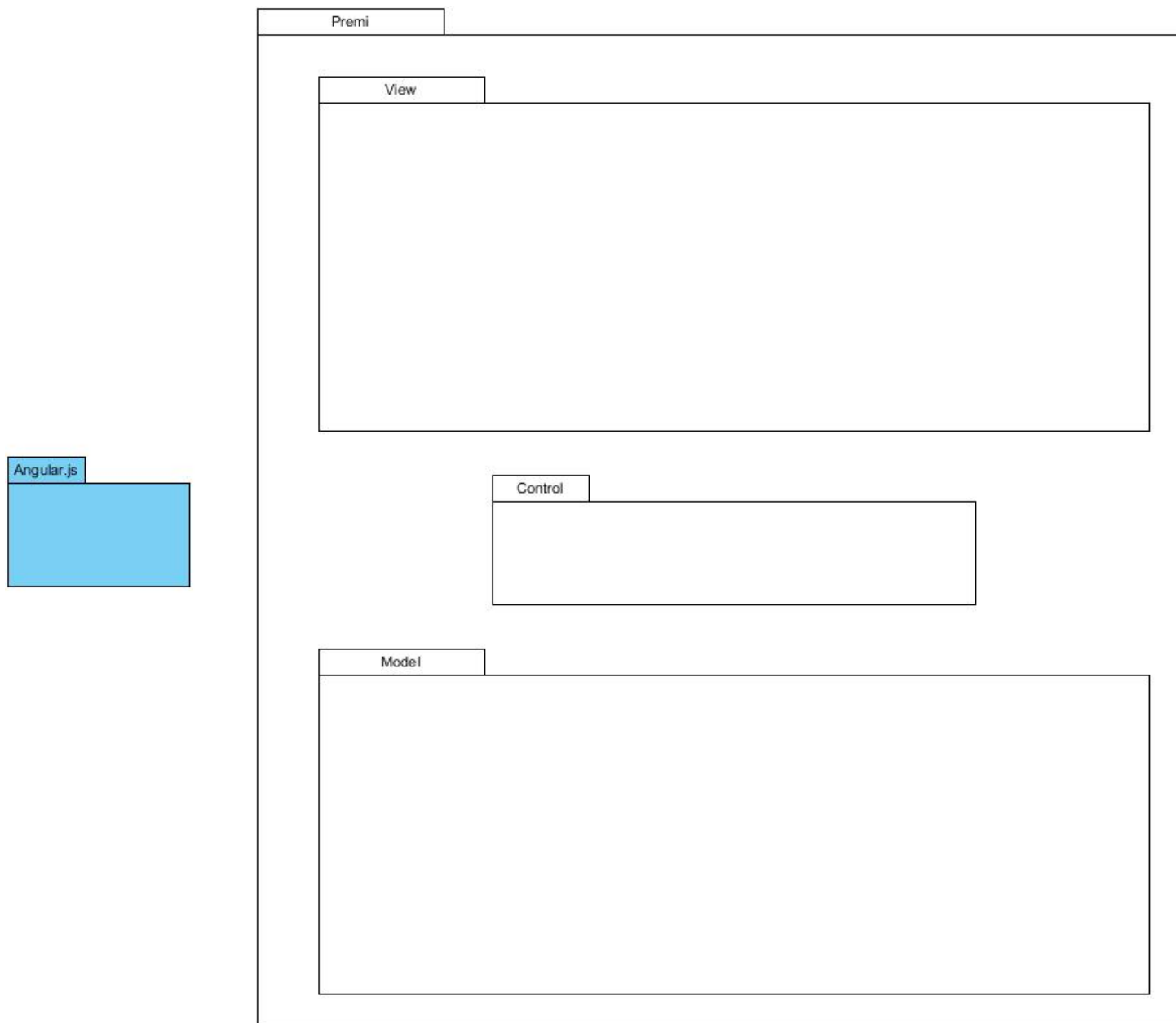


Fig 1: Architettura generale del sistema

5 Descrizione dei singoli componenti

5.1 Model

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

Relazioni d'uso di altre componenti: ??????????????????????.

Package contenuti:

- Premi::Model::Inserimento;
- Premi::Model::Rimozione;
- Premi::Model::Modifica;
- Premi::Model::Command;
- Premi::Model::Invoker;
- Premi::Model::Builder;
- Premi::Model::Presentazione;
- Premi::Model::MongoHandler.

5.1.1 Premi::Model::Inserimento

Tipo, obiettivo e funzione del componente: All'interno di questo Package viene implementato il Design Pattern template per l'inserimento di nuovi elementi nella presentazione.

Relazioni d'uso di altre componenti: Il package è in relazione con Premi::Model::Command da cui riceve i segnali e i parametri di inserimento dell'elemento. Inoltre comunica con il package Premi::Model::Presentazione, istanziando gli oggetti delle sottoclassi di SlideShowElement e inserendoli in SlideShow.

5.1.1.1 Premi::Model::Inserimento::Inserter

Tipo, obiettivo e funzione del componente: Classe astratta definita per l'implementazione del Design Pattern template, per l'inserimento di elementi all'interno di una presentazione.

Relazioni d'uso di altre componenti:

- `Premi::Model::Command::ConcreteConcreteInsertCommand` -> utilizza i metodi messi a disposizione da `Inserter` e concretizzati dalle sue sottoclassi che a loro volta invocano le funzioni della classe `Premi::Model::Presentazione::SlideShow` per l'impostazione dei campi relativi.

Interfacce con e relazioni d'uso e da altre componenti: Definisce le operazioni primitive astratte che le classi concrete sottostanti andranno a sovraccaricare e implementa il metodo template che rappresenta lo scheletro dell'algoritmo per l'inserimento di un elemento nella presentazione. È il componente receiver del Design Pattern Command.

Sottoclassi:



- #### 5.1.1.2 Premi::Controller::Presentazione::Inserimento::ConcreteTextInserter

Relazioni d'uso di altre componenti:

- Interfacce con e relazioni d'uso e da altre componenti:** Viene invocato per inserire elementi testuali in una presentazione.

- Premi::Model::Inserimento::Inserter.

Tipo, obiettivo e funzione del componente: Classe che rappresenta un algoritmo di inserimento di un elemento frame all'interno di una presentazione. È uno dei componenti concreti del Design Pattern Template.

Relazioni d'uso di altre componenti:

- Interfacce con e relazioni d'uso e da altre componenti:** Viene invocato per inserire elementi di tipo frame in una presentazione.

- Premi::Model::Inserimento::Inserter.

Tipo, obiettivo e funzione del componente: Classe che rappresenta un algoritmo di inserimento di un elemento svg all'interno di una presentazione. È uno dei componenti concreti del Design Pattern Template.

Relazioni d'uso di altre componenti:



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



Interfacce con e relazioni d'uso e da altre componenti: Viene invocato per inserire elementi di tipo audio in una presentazione.

Classi ereditate:

- Premi::Model::Inserimento::Inserter.

5.1.2 Premi::Model::Eliminazione

Tipo, obiettivo e funzione del componente: All'interno di questo Package viene implementato il Design Pattern template per l'eliminazione di elementi dalla presentazione.

Relazioni d'uso di altre componenti: Il package è in relazione con Premi::Model::Command da cui riceve i segnali e i parametri di eliminazione dell'elemento. Inoltre comunica con il package Premi::Model::Presentazione, rimuovendo dall'oggetto di classe SlideShow gli oggetti delle sottoclassi di SlideShowElement e distruggendoli.

5.1.2.1 Premi::Model::Eliminazione::Remover

Tipo, obiettivo e funzione del componente: Classe astratta definita per l'implementazione del Design Pattern template, per l'eliminazione di elementi all'interno di una presentazione.

Relazioni d'uso di altre componenti:

- Premi::Model::Command::ConcreteConcreteRemoveCommand -> utilizza i metodi messi a disposizione da Remover e concretizzati dalle sue sottoclassi che a loro volta invocano le funzioni della classe.

Interfacce con e relazioni d'uso e da altre componenti: Definisce le operazioni primitive astratte che le classi concrete sottostanti andranno a sovraccaricare o definire e implementa il metodo template che rappresenta lo scheletro dell'algoritmo per l'eliminazione di un elemento nella presentazione.

È il componente receiver del Design Pattern Command.

Sottoclassi:

- Premi::Model::Eliminazione::ConcreteTextRemover;
- Premi::Model::Eliminazione::ConcreteFrameRemover;
- Premi::Model::Eliminazione::ConcreteSvgRemover;
- Premi::Model::Eliminazione::ConcreteImageRemover;
- Premi::Model::Eliminazione::ConcreteVideoRemover;
- Premi::Model::Eliminazione::ConcreteAudioRemover.

5.1.2.2 Premi::Model::Eliminazione:: ConcreteTextRemover

Tipo, obiettivo e funzione del componente: Classe che implementa un algoritmo di eliminazione di un elemento testuale all'interno di una presentazione. È uno dei componenti concreti del Design Pattern Template.

Relazioni d'uso di altre componenti:



- Premi::Model::ConcreteRemoveCommand -> invoca i metodi per eliminare un elemento di tipo testo dalla presentazione.

Interfacce con e relazioni d'uso e da altre componenti: Viene invocato per eliminare elementi testuali in una presentazione.

Classi ereditate:

- Premi::Model::Eliminazione::Remover.

5.1.2.3 Premi::Model::Eliminazione:: ConcreteFrameRemover

Tipo, obiettivo e funzione del componente: Classe che implementa un algoritmo di eliminazione di un elemento di tipo frame all'interno di una presentazione. È uno dei componenti concreti del Design Pattern Template.

Relazioni d'uso di altre componenti:

- Premi::Model::ConcreteRemoveCommand -> invoca i metodi per eliminare un elemento di tipo frame dalla presentazione.

Interfacce con e relazioni d'uso e da altre componenti: Viene invocato per eliminare elementi di tipo frame da una presentazione.

Classi ereditate:

- Premi::Model::Eliminazione::Remover.

5.1.2.4 Premi::Model::Eliminazione:: ConcreteSVGtRemover

Tipo, obiettivo e funzione del componente: Classe che implementa un algoritmo di eliminazione di un elemento di tipo SVG all'interno di una presentazione. È uno dei componenti concreti del Design Pattern Template.

Relazioni d'uso di altre componenti:

- Premi::Model::ConcreteRemoveCommand -> invoca i metodi per eliminare un elemento di tipo SVG dalla presentazione.

Interfacce con e relazioni d'uso e da altre componenti: Viene invocato per eliminare elementi SVG da una presentazione.

Classi ereditate:

- Premi::Model::Eliminazione::Remover.

5.1.2.5 Premi::Model::Eliminazione:: ConcreteImageRemover

Tipo, obiettivo e funzione del componente: Classe che implementa un algoritmo di eliminazione di un elemento immagine all'interno di una presentazione. È uno dei componenti concreti del Design Pattern Template.

Relazioni d'uso di altre componenti:

- Premi::Model::ConcreteRemoveCommand -> invoca i metodi per eliminare un elemento di tipo immagine dalla presentazione.



Classi ereditate:

- #### 5.1.2.6 Premi::Model::Eliminazione:: ConcreteVideoRemover

Relazioni d'uso di altre componenti:

- Classi ereditate:**

- #### 5.1.2.7 Premi::Model::Eliminazione:: ConcreteAudioRemover

Relazioni d'uso di altre componenti:

- Classi ereditate:**

- ### 5.1.3 Premi::Model::Modifica

Relazioni d'uso di altre componenti: Il package è in relazione con Premi::Model::Command da cui riceve i segnali e i parametri di modifica dell'elemento. Inoltre comunica con il package Premi::Model::Presentazione, modificando nell'oggetto di classe SlideShow gli oggetti delle sottoclassi di SlideShowElement.

5.1.3.1 Premi::Model::Modifica::Editor

Tipo, obiettivo e funzione del componente: Interfaccia per la componente strategy del Design Pattern Strategy per la selezione dell'algoritmo di modifica della presentazione.

Relazioni d'uso di altre componenti:

- `Premi::Model::Command::ConcreteEditCommand` -> Invoca i costruttori delle sottoclassi di `Editor`;
- `Premi::Model::Presentazione::SlideShow` <- scorre gli elementi dei membri contenitori all'interno di `SlideShow` per trovare l'elemento da modificare;
- `Premi::Model::Presentazione::SlideShowElement` <- invoca le funzioni della classe `SlideShowElement` per modificare opportunamente i campi dell'elemento.

Interfacce con e relazioni d'uso e da altre componenti: Permette di selezionare dinamicamente ed in modo estensibile l'algoritmo di modifica della presentazione.

Sottoclassi:

- Premi::Model::Modifica::Editor::EditorPosition;
- Premi::Model::Modifica::Editor::EditorSize;
- Premi::Model::Modifica::Editor::EditorContent;
- Premi::Model::Modifica::Editor::EditorRotate;
- Premi::Model::Modifica::Editor::EditorColor;
- Premi::Model::Modifica::Editor::EditorShape.

5.1.3.2 Premi::Model::Modifica::EditorPosition

Tipo, obiettivo e funzione del componente: Classe concreta del Design Pattern Strategy per la modifica dei campi inerenti alla posizione di un elemento della presentazione.

Relazioni d'uso di altre componenti:

- `Premi::Model::Command::ConcreteEditCommand` -> Invoca il costruttore di `EditorPosition`;
- `Premi::Model::Presentazione::SlideShow` <- scorre gli elementi dei membri contenitori all'interno di `SlideShow` per trovare l'elemento da modificare;
- `Premi::Model::Presentazione::SlideShowElement` <- invoca le funzioni della classe `SlideShowElement` per modificare opportunamente i campi relativi alla posizione dell'elemento.

Interfacce con e relazioni d'uso e da altre componenti: Premi::Model::Command::ConcreteEditComm
invoca il costruttore e la funzione di esecuzione dell'operazione di modifica, EditorPosition in-
vocherà quindi i metodi di modifica delle coordinate forniti all'interno della sottoclasse di Pre-
mi::Model::Presentazione::SlideShowElement di cui fa parte l'oggetto da modificare.

Classi ereditate:

- Premi::Model::Modifica::Editor.



5.1.3.3 Premi::Model::Modifica::EditorSize

Tipo, obiettivo e funzione del componente: Classe concreta del Design Pattern Strategy per la modifica dei campi inerenti alla dimensione di un elemento della presentazione.

Relazioni d'uso di altre componenti:

- Premi::Model::Command::ConcreteEditCommand -> Invoca il costruttore di EditorSize;
- Premi::Model::Presentazione::SlideShow<- scorre gli elementi dei membri contenitori all'interno di SlideShow per trovare l'elemento da modificare;
- Premi::Model::Presentazione::SlideShowElement <- invoca le funzioni della classe SlideShowElement per modificare opportunamente i campi relativi alla dimensione dell'elemento.

Interfacce con e relazioni d'uso e da altre componenti: Premi::Model::Command::ConcreteEditCommand invoca il costruttore e la funzione di esecuzione dell'operazione di modifica, EditorSize invocherà quindi i metodi di modifica delle dimensioni forniti all'interno della sottoclasse di Premi::Model::Presentazione::SlideShowElement di cui fa parte l'oggetto da modificare.

Classi ereditate:

- Premi::Model::Modifica::Editor.

5.1.3.4 Premi::Model::Modifica::EditorRotate

Tipo, obiettivo e funzione del componente: Classe concreta del Design Pattern Strategy per la modifica dei campi inerenti all'inclinazione di un elemento della presentazione.

Relazioni d'uso di altre componenti:

- Premi::Model::Command::ConcreteEditCommand -> Invoca il costruttore di EditorRotate;
- Premi::Model::Presentazione::SlideShow<- scorre gli elementi dei membri contenitori all'interno di SlideShow per trovare l'elemento da modificare;
- Premi::Model::Presentazione::SlideShowElement <- invoca le funzioni della classe SlideShowElement per modificare opportunamente i campi relativi all'inclinazione dell'elemento.

Interfacce con e relazioni d'uso e da altre componenti: Premi::Model::Command::ConcreteEditCommand invoca il costruttore e la funzione di esecuzione dell'operazione di modifica, EditorPosition invocherà quindi i metodi di modifica dell'inclinazione forniti all'interno della sottoclasse di Premi::Model::Presentazione::SlideShowElement di cui fa parte l'oggetto da modificare.

Classi ereditate:

- Premi::Model::Modifica::Editor.



Tipo, obiettivo e funzione del componente: Classe concreta del Design Pattern Strategy per la modifica dei campi inerenti al contenuto di un elemento di tipo testuale della presentazione.

- `Premi::Model::Command::ConcreteEditCommand` -> Invoca il costruttore di `EditorContent`;
- `Premi::Model::Presentazione::SlideShow` <- scorre gli elementi del membro contenitore all'interno di `SlideShow` per trovare l'elemento testuale da modificare;
- `Premi::Model::Presentazione::SlideShowElement` <- invoca le funzioni della classe `SlideShowElement` per modificare opportunamente i campi relativi alla contenuto dell'elemento testuale.

- Premi::Model::Modifica::Editor.

Tipo, obiettivo e funzione del componente: Classe concreta del Design Pattern Strategy per la modifica dei campi inerenti alla forma di un elemento SVG della presentazione.

- `Premi::Model::Command::ConcreteEditCommand` -> Invoca il costruttore di `EditorShape`;
- `Premi::Model::Presentazione::SlideShow` <- scorre gli elementi dei membri contenitori all'interno di `SlideShow` per trovare l'elemento SVG da modificare;
- `Premi::Model::Presentazione::SlideShowElement` <- invoca le funzioni della classe `SlideShowElement` per modificare opportunamente i campi relativi alla forma dell'elemento SVG.

- Premi::Model::Modifica::Editor.



5.1.3.7 Premi::Model::Modifica::EditorColor

Tipo, obiettivo e funzione del componente: Classe concreta del Design Pattern Strategy per la modifica dei campi inerenti al colore di un elemento SVG della presentazione.

Relazioni d'uso di altre componenti:

- Premi::Model::Command::ConcreteEditCommand -> Invoca il costruttore di EditorColor;
- Premi::Model::Presentazione::SlideShow<- scorre gli elementi del membro contenitore all'interno di SlideShow per trovare l'elemento SVG da modificare;
- Premi::Model::Presentazione::SlideShowElement <- invoca le funzioni della classe SlideShowElement per modificare opportunamente i campi relativi alla forma dell'elemento SVG.

Interfacce con e relazioni d'uso e da altre componenti: Premi::Model::Command::ConcreteEditCommand invoca il costruttore e la funzione di esecuzione dell'operazione di modifica, EditorShape invocherà quindi i metodi di modifica della forma forniti dalla classe Premi::Model::Presentazione::SVG.

Classi ereditate:

- Premi::Model::Modifica::Editor.

5.1.4 Premi::Model::Command

Tipo, obiettivo e funzione del componente: All'interno di questo Package viene implementato il Design Pattern command, utile per la gestione di funzioni di annullamento e ripristino.

Relazioni d'uso di altre componenti: All'interno del Model, il package è in relazione con Premi::Model::Inserimento, Premi::Model::Eliminazione e Premi::Model::Modifica. Il package comunica, inoltre, con il controller, infatti le sue classi sono generate da Premi::Controller::Presentazione::Editor.

5.1.4.1 Premi::Model::Invoker

Tipo, obiettivo e funzione del componente: È componente invoker del Design Pattern Command, il suo scopo è tenere traccia delle modifiche atomiche apportate alla presentazione (modifica di elemento, eliminazione di elemento e inserimento di elemento) per poter implementare le funzioni di annulla/ripristina.

Relazioni d'uso di altre componenti:

- Premi::Controller::MobileEdit->crea un oggetto di una sottoclasse di Premi::Model::Command::AbstractCommand, passandolo all'Invoker che lo esegue e lo inserisce nello stack "undo", richiama il metodo che svuota lo stack "redo".
Può inoltre invocare il metodo "unexecute" dell'Invoker che provvede a richiamare il metodo undo del comando sulla cima dello stack "undo" e a spostarlo quindi nello stack "redo". Alternativamente invoca il metodo "redo" dell'Invoker che provvede a eseguire il comando sulla cima dello stack "redo" e a spostarlo quindi nello stack "undo";
- Premi::Controller::DesktopEdit->si comporta in modo analogo a MobileEdit;



- Interfacce con e relazioni d'uso e da altre componenti:** Viene invocato per effettuare le operazioni di modifica alla presentazione, a sua volta invoca una classe derivata da `Premi::Model::Command` per eseguire materialmente il comando. Quando un comando viene eseguito, `Invoker` lo salva in un array `$undo[]`, insieme ai parametri necessari a riportare la presentazione allo stato precedente.

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



5.1.4.4 Premi::Model::Command::ConcreteRemoveCommand

Tipo, obiettivo e funzione del componente: È classe concreta del Design Pattern Command, rappresenta un comando per rimuovere un elemento dall'oggetto SlideShow.

Relazioni d'uso di altre componenti:

- Premi::Controller::Presentazione::EditController -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker; Premi::Model::Invoker -> esegue il comando o ne invoca il metodo di annullamento;
- Premi::Model::Eliminazione::Remover <- invoca la classe concreta del template per l'eliminazione di un elemento.

Interfacce con e relazioni d'uso e da altre componenti: Viene utilizzata per gestire i Signal riguardanti l'eliminazione di un elemento ed invocare i corretti metodi del Model.

Classi ereditate:

- Premi::Model::Command::AbstractCommand.

5.1.4.5 Premi::Model::Command::ConcreteEditCommand

Tipo, obiettivo e funzione del componente: È classe concreta del Design Pattern Command, rappresenta un comando per modificare un elemento elemento nell'oggetto SlideShow.

Relazioni d'uso di altre componenti:

- Premi::Controller::Presentazione::EditController -> invoca il costruttore della classe e passa l'oggetto così creato all'Invoker; Premi::Model::Invoker -> esegue il comando o ne invoca il metodo di annullamento;
- Premi::Model::Modifica::Editor <- invoca la classe concreta del design pattern Strategy per la modifica di un elemento.

Interfacce con e relazioni d'uso e da altre componenti: Viene utilizzata per gestire i Signal riguardanti la modifica di un nuovo elemento ed invocare i corretti metodi del Model;

Classi ereditate:

- Premi::Model::Command::AbstractCommand.

5.1.5 Premi::Model::Presentazione

Tipo, obiettivo e funzione del componente: Di questo package fanno parte le classi degli elementi della presentazione e la classe che definisce la presentazione stessa. Si tratta del package centrale del software.

Relazioni d'uso di altre componenti: Premi::Model::Presentazione è in comunicazione con

- Premi::Model::Inserimento, i cui oggetti durante la modifica della presentazione istanziano oggetti di tipo SlideShowElement;
- Premi::Model::Eliminazione, i cui oggetti rimuovono da SlideShow gli oggetti di tipo SlideShowElement e li distruggono;

- `Premi::Model::Modifica`, i cui oggetti invocano metodi degli oggetti `SlideShowElement` che ne impostano i campi;
- `Premi::Controller::Presentazione::EditController` o `Premi::Controller::Presentazione::ExecutionController` invocano il costruttore di `Loader`;
- al momento del caricamento della presentazione gli oggetti di `Premi::Model::Presentazione::SlideShowElement` sono invece costruiti dalle classi del package `Premi::Model::Builder`.

5.1.5.1 Premi::Model::Presentazione::Loader

Tipo, obiettivo e funzione del componente: Questa classe si occupa della costruzione dell'oggetto SlideShow quando l'utente carica una presentazione o quando l'utente crea una nuova presentazione.

Relazioni d'uso di altre componenti:

- `Premi::Controller::EditController-> costruisce Loader;`
- `Premi::Controller::ExecutionController-> costruisce Loader;`
- `Premi::Model::SlideShow <-` invoca il metodo di `SlideShow::getInstance` che chiama il costruttore privato di `SlideShow`, eventualmente invoca le funzioni dell'oggetto `SlideShow` per inserire gli oggetti delle sottoclassi di `SlideShowElement` restituiti da `Premi::Model::Builder::Director;`
- `Premi::Model::Builder::Director <-` `Loader` passa a `Director` i parametri degli oggetti da costruire.

Interfacce con e relazioni d'uso e da altre componenti: il controller invoca il costruttore di Loader. Quando non passa alcun argomento Loader costruisce un oggetto di classe `Premi::Model::SlideShow`; altrimenti passa l'id dell'oggetto json che descrive la presentazione, Loader recupera i dati e costruisce un oggetto `SlideShow`, passa a `Premi::Model::Builder::Director` i parametri degli oggetti da istanziare contenuti dentro all'oggetto json che descrive la presentazione. Director restituisce gli oggetti istanziati e Loader invoca i metodi di `SlideShow` per inserire tali oggetti nei rispettivi contenitori.

5.1.5.2 Premi::Model::Presentazione::SlideShow

Tipo, obiettivo e funzione del componente: Classe implementata con il Design Pattern Singleton, contiene tutte le impostazioni della presentazione caricata, gli oggetti in essa presenti e i metodi per settarli o inserirne di nuovi. Gli oggetti della presentazione si trovano all'interno di contenitori divisi per classe. Tramite un iteratore è possibile scorrere detti contenitori.

Relazioni d'uso di altre componenti:

- Premi::Model::Presentazione::Loader -> invoca il metodo `SlideShow::getInstance()`, che a sua volta invoca il costruttore privato di `SlideShow`.



5.1.5.3 Premi::Model::Presentazione::SlideShowElement

Relazioni d'uso di altre componenti:

- Interfacce con e relazioni d'uso e da altre componenti:** Premi::Model::Inserimento::Inserter
istanzia oggetti di sottoclassi di SlideShowElement e li inserisce nei membri contenitori all'in-
terno di Premi::Model::Presentazione::SlideShow

- Premi::Model::Presentazione:: Text;
- Premi::Model::Presentazione:: Frame;
- Premi::Model::Presentazione:: Image;
- Premi::Model::Presentazione::SVG;
- Premi::Model::Presentazione::Audio;
- Premi::Model::Presentazione::Video.

5.1.5.4 Premi::Model::Presentazione::Text

Relazioni d'uso di altre componenti:

- `Premi::Model::Inserimento::ConcreteTextInserter` -> invoca il costruttore di `Text` e inserisce l'oggetto nel membro contenitore all'interno dell'oggetto della classe `Premi::Model::Presentazione`;
- `Premi::Model::Eliminazione::ConcreteTextRemover` -> rimuove l'oggetto `Text` dal membro contenitore all'interno di `Premi::Model::Presentazione::SlideShow`, ne invoca quindi il distruttore;



- Premi::Model::Modifica::ConcreteTextEditor -> invoca i metodi che impostano i campi dell'oggetto Text.

Interfacce con e relazioni d'uso e da altre componenti: Gli oggetti della classe Text vengono istanziati da Premi::Model::Inserimento::ConcreteTextInserter o da Premi::Model::Builder::ConcreteTextInserter e inseriti nei membri contenitori all'interno di Premi::Model::Presentazione::SlideShow.

Classi ereditate:

- Premi::Model::Presentazione::SlideShowElement.

5.1.5.5 Premi::Model::Presentazione::Frame

Tipo, obiettivo e funzione del componente: Gli oggetti della classe Frame rappresentano gli elementi di tipo frame della presentazione.

Relazioni d'uso di altre componenti:

- Premi::Model::Inserimento::ConcreteFrameInserter -> invoca il costruttore di Frame e inserisce l'oggetto nel membro contenitore all'interno dell'oggetto della classe Premi::Model::Presentazione::SlideShow.
- Premi::Model::Eliminazione::ConcreteFrameRemover -> rimuove l'oggetto Frame dal membro contenitore all'interno di Premi::Model::Presentazione::SlideShow, ne invoca quindi il distruttore;
- Premi::Model::Modifica::ConcreteFrameEditor -> invoca i metodi che impostano i campi dell'oggetto Frame.

Interfacce con e relazioni d'uso e da altre componenti: Gli oggetti della classe Frame vengono istanziati da Premi::Model::Inserimento::ConcreteFrameInserter o da Premi::Model::Builder::ConcreteFrameInserter e inseriti nei membri contenitori all'interno di Premi::Model::Presentazione::SlideShow.

Classi ereditate:

- Premi::Model::Presentazione::SlideShowElement.

5.1.5.6 Premi::Model::Presentazione::Image

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

Relazioni d'uso di altre componenti:

- Premi::Model::Inserimento::ConcreteImageInserter -> invoca il costruttore di Image e inserisce l'oggetto nel membro contenitore all'interno dell'oggetto della classe Premi::Model::Presentazione::SlideShow.
- Premi::Model::Eliminazione::ConcreteImageRemover -> rimuove l'oggetto Image dal membro contenitore all'interno di Premi::Model::Presentazione::SlideShow, ne invoca quindi il distruttore;
- Premi::Model::Modifica::ConcreteImageEditor -> invoca i metodi che impostano i campi dell'oggetto Image.

Interfacce con e relazioni d'uso e da altre componenti: Gli oggetti della classe Image vengono istanziati da Premi::Model::Inserimento::ConcreteImageInserter o da Premi::Model::Builder::ConcreteImageInserter e inseriti nei membri contenitori all'interno di Premi::Model::Presentazione::SlideShow.

Classi ereditate:

- Premi::Model::Presentazione::SlideShowElement.



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



5.1.5.9 Premi::Model::Presentazione::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:

- Premi::Model::Inserimento::ConcreteVideoInserter -> invoca il costruttore di Video e inserisce l'oggetto nel membro contenitore all'interno dell'oggetto della classe Premi::Model::Presentazione::SlideShowElement.
- Premi::Model::Eliminazione::ConcreteVideoRemover -> rimuove l'oggetto Video dal membro contenitore all'interno di Premi::Model::Presentazione::SlideShowElement, ne invoca quindi il distruttore;
- Premi::Model::Modifica::ConcreteFrameEditor -> invoca i metodi che impostano i campi dell'oggetto Video.

Interfacce con e relazioni d'uso e da altre componenti: Gli oggetti della classe Video vengono istanziati da Premi::Model::Inserimento::ConcreteVideoInserter o da Premi::Model::Builder::ConcreteVideoInserter e inseriti nei membri contenitori all'interno di Premi::Model::Presentazione::SlideShowElement.

Classi ereditate:

- Premi::Model::Presentazione::SlideShowElement.

5.1.6 Premi::Model::Builder

Tipo, obiettivo e funzione del componente: Il package Premi::Model::Builder implementa il Design Pattern builder. Il package ha come scopo principale la creazione degli oggetti delle sottoclassi di Premi::Model::Presentazione::SlideShowElement al momento del caricamento della presentazione nel programma.

Relazioni d'uso di altre componenti: È in relazione con il package Premi::Model::Presentazione che ne costruisce il Director, e con il package Premi::Model::Presentazione, delle cui classi, sottoclassi di SlideShowElement, costruisce gli oggetti.

5.1.6.1 Premi::Model::Builder::Director

Tipo, obiettivo e funzione del componente: Implementazione della parte Director del Design Pattern Builder. Fornisce a Premi::Model::Presentazione::Loader gli oggetti della presentazione.

Relazioni d'uso di altre componenti:

- Premi::Model::Presentazione::Loader -> costruisce Director, passandogli i parametri per la costruzione degli oggetti delle sottoclassi di Premi::Model::Presentazione::SlideShowElement.
- Premi::Model::Builder::AbstractBuilder <- invoca il costruttore degli oggetti delle sottoclassi di AbstractBuilder che genereranno i file delle sottoclassi di Premi::Model::Presentazione::SlideShowElement.

Interfacce con e relazioni d'uso e da altre componenti: Viene invocato da Premi::Model::Presentazione::Loader. È costruito con design pattern singleton. Restituisce gli oggetti di tipo Premi::Model::Presentazione::SlideShowElement a Premi::Model::Presentazione::Loader.



Tipo, obiettivo e funzione del componente: Classe astratta del Design Pattern Builder.

Interfacce con e relazioni d'uso e da altre componenti: Gli oggetti delle sottoclassi della classe AbstractBuilder vengono istanziati da Premi::Model::Builder::Director e hanno lo scopo di istanziare gli oggetti delle sottoclassi di Premi::Model::Presentazione::SlideShowElement.

Sottoclassi:

- ### 5.1.6.3 Premi::Model::Builder::ConcreteTextBuilder

Tipo, obiettivo e funzione del componente: Classe concreta del Design Pattern Builder. Ha lo scopo di istanziare gli oggetti di classe `Premi::Model::Presentazione::Text` al momento del caricamento della presentazione e di passarli all'oggetto di classe `Director`.

Relazioni d'uso di altre componenti:

- Premi::Model::Builder::Director -> istanzia gli oggetti passando i parametri ricevuti da Premi::Model::Presentazione::Loader;
- Premi::Model::Presentazione::Text <- istanzia gli oggetti della classe Text e li restituisce al director.

Interfacce con e relazioni d'uso e da altre componenti: Gli oggetti della classe ConcreteTextBuilder vengono istanziati da Premi::Model::Builder::Director e hanno lo scopo di istanziare gli oggetti delle sottoclassi di Premi::Model::Presentazione::Text e di passarli al Director.

Classi ereditate:

- Premi::Model::Builder::AbstractBuilder.

5.1.6.4 Premi::Model::Builder::ConcreteFrameBuilder

Tipo, obiettivo e funzione del componente: Classe concreta del Design Pattern Builder. Ha lo scopo di istanziare gli oggetti di classe Premi::Model::Presentazione::Frame al momento del caricamento della presentazione e di passarli all'oggetto di classe Director.

Relazioni d'uso di altre componenti:

- Premi::Model::Builder::Director -> istanzia gli oggetti passando i parametri ricevuti da Premi::Model::Presentazione::Loader;



- `Premi::Model::Presentazione::Frame <-` istanza gli oggetti della classe `Frame` e li restituisce al director.

Interfacce con e relazioni d'uso e da altre componenti: Gli oggetti della classe `ConcreteFrameBuilder` vengono istanziati da `Premi::Model::Builder::Director` e hanno lo scopo di istanziare gli oggetti delle sottoclassi di `Premi::Model::Presentazione::Frame` e di passarli al `Director`.

Classi ereditate:

- `Premi::Model::Builder::AbstractBuilder`.

5.1.6.5 `Premi::Model::Builder::ConcreteImageBuilder`

Tipo, obiettivo e funzione del componente: Classe concreta del Design Pattern Builder. Ha lo scopo di istanziare gli oggetti di classe `Premi::Model::Presentazione::Image` al momento del caricamento della presentazione e di passarli all'oggetto di classe `Director`.

Relazioni d'uso di altre componenti:

- `Premi::Model::Builder::Director ->` istanzia gli oggetti passando i parametri ricevuti da `Premi::Model::Presentazione::Loader`;
- `Premi::Model::Presentazione::Image <-` istanza gli oggetti della classe `Image` e li restituisce al director.

Interfacce con e relazioni d'uso e da altre componenti: Gli oggetti della classe `ConcreteImageBuilder` vengono istanziati da `Premi::Model::Builder::Director` e hanno lo scopo di istanziare gli oggetti delle sottoclassi di `Premi::Model::Presentazione::Image` e di passarli al `Director`.

Classi ereditate:

- `Premi::Model::Builder::AbstractBuilder`.

5.1.6.6 `Premi::Model::Builder::ConcreteSVGBuilder`

Tipo, obiettivo e funzione del componente: Classe concreta del Design Pattern Builder. Ha lo scopo di istanziare gli oggetti di classe `Premi::Model::Presentazione::SVG` al momento del caricamento della presentazione e di passarli all'oggetto di classe `Director`.

Relazioni d'uso di altre componenti:

- `Premi::Model::Builder::Director ->` istanzia gli oggetti passando i parametri ricevuti da `Premi::Model::Presentazione::Loader`;
- `Premi::Model::Presentazione::SVG <-` istanza gli oggetti della classe `SVG` e li restituisce al director.

Interfacce con e relazioni d'uso e da altre componenti: Gli oggetti della classe `ConcreteSVGBuilder` vengono istanziati da `Premi::Model::Builder::Director` e hanno lo scopo di istanziare gli oggetti delle sottoclassi di `Premi::Model::Presentazione::SVG` e di passarli al `Director`.

Classi ereditate:

- `Premi::Model::Builder::AbstractBuilder`.



Tipo, obiettivo e funzione del componente: Classe concreta del Design Pattern Builder. Ha lo scopo di istanziare gli oggetti di classe Premi::Model::Presentazione::Audio al momento del caricamento della presentazione e di passarli all'oggetto di classe Director.

- Premi::Model::Builder::Director -> istanzia gli oggetti passando i parametri ricevuti da Premi::Model::Presentazione::Loader;
- Premi::Model::Presentazione::Audio <- istanzia gli oggetti della classe Audio e li restituisce al director.

Classi ereditate:

- Tipo, obiettivo e funzione del componente:** Classe concreta del Design Pattern Builder. Ha lo scopo di istanziare gli oggetti di classe Premi::Model::Presentazione::Video al momento del caricamento della presentazione e di passarli all'oggetto di classe Director.

- `Premi::Model::Builder::Director` -> istanzia gli oggetti passando i parametri ricevuti da `Premi::Model::Presentazione::Loader`;
- `Premi::Model::Presentazione::Video` <- istanzia gli oggetti della classe `Video` e li restituisce al `director`.

Classi ereditate:

- Tipo, obiettivo e funzione del componente:** All'interno di questo Package viene implementato il Design Pattern strategy per il caricamento di nuovi elementi nella presentazione.

Relazioni d'uso di altre componenti: Il package è in relazione con Premi::Controller::MobileEdit, Premi::Controller::DesktopEdit e [[ControllerUtente]] dai quali riceve i segnali e i parametri di caricamento dell'elemento.



Tipo, obiettivo e funzione del componente: Classe astratta definita per l'implementazione del Design Pattern strategy, per il caricamento di elementi all'interno dello spazio personale di un utente.

- `Premi::Controller::MobileEdit` e `Premi::Controller::DesktopEdit` -> costruiscono un oggetto di una sottoclasse di `Uploader` e utilizzano i metodi da questi messi a disposizione per caricare il file nel server, inoltre costruiscono un oggetto della classe `Premi::Model::Command::InsertCommand` e lo danno in pasto a `Premi::Model::Command::Invoker`;

Interfacce con e relazioni d'uso e da altre componenti: Definisce le operazioni primitive astratte che le classi concrete sottostanti andranno a sovraccaricare e implementa il metodo strategy che rappresenta lo scheletro dell'algoritmo per il caricamento di un elemento nella presentazione.

- Premi::Model::Caricamento::ConcreteSvgUploader;
- Premi::Model::Caricamento::ConcreteImageUploader;
- Premi::Model::Caricamento::ConcreteVideoUploader;
- Premi::Model::Caricamento::ConcreteAudioUploader.

Tipo, obiettivo e funzione del componente: Classe che rappresenta un algoritmo di caricamento di un elemento svg all'interno dello spazio personale di un utente. È uno dei componenti concreti del Design Pattern Strategy.

- `Premi::Controller::MobileEdit` e `Premi::Controller::DesktopEdit` -> costruiscono un oggetto di classe `ConcreteSvgUploader` e utilizzano i metodi da questo messi a disposizione per caricare il file nel server, inoltre costruiscono un oggetto della classe `Premi::Model::Command::InsertCommand` e lo danno in pasto `Premi::Model::Command::Invoker`;
- `((ControllerUtente))` -> costruisce un oggetto di classe `ConcreteSvgUploader` e utilizza i metodi da questi messi a disposizione per caricare il file nel server.

Interfacce con e relazioni d'uso e da altre componenti: Viene invocato per caricare elementi SVG nello spazio personale di un utente ed eventualmente inserirlo automaticamente nello spazio personale di un utente ed eventualmente inserirlo automaticamente in una presentazione..

- Premi::Model::Caricamento::Uploader.



Tipo, obiettivo e funzione del componente: Classe che rappresenta un algoritmo di caricamento di un elemento immagine all'interno dello spazio personale di un utente. È uno dei componenti concreti del Design Pattern Strategy.

- `Premi::Controller::MobileEdit` e `Premi::Controller::DesktopEdit` -> costruiscono un oggetto di classe `ConcreteImageUploader` e utilizzano i metodi da questo messi a disposizione per caricare il file nel server, inoltre costruiscono un oggetto della classe `Premi::Model::Command::InsertCommand` e lo danno in pasto `Premi::Model::Command::Invoker`;
- (`ControllerUtente`) -> costruisce un oggetto di classe `ConcreteImageUploader` e utilizza i metodi da questi messi a disposizione per caricare il file nel server.

Interfacce con e relazioni d'uso e da altre componenti: Viene invocato per caricare elementi di tipo immagine nello spazio personale di un utente ed eventualmente inserirlo automaticamente in una presentazione..

- Premi::Model::Caricamento::Uploader.

Tipo, obiettivo e funzione del componente: Classe che rappresenta un algoritmo di caricamento di un elemento video all'interno dello spazio personale di un utente. È uno dei componenti concreti del Design Pattern Strategy.

- `Premi::Controller::MobileEdit` e `Premi::Controller::DesktopEdit` -> costruiscono un oggetto di classe `ConcreteVideoUploader` e utilizzano i metodi da questo messi a disposizione per caricare il file nel server, inoltre costruiscono un oggetto della classe `Premi::Model::Command::InsertCommand` e lo danno in pasto `Premi::Model::Command::Invoker`;
- `((ControllerUtente))` -> costruisce un oggetto di classe `ConcreteVideoUploader` e utilizza i metodi da questi messi a disposizione per caricare il file nel server.

Interfacce con e relazioni d'uso e da altre componenti:Viene invocato per caricare elementi di tipo video nello spazio personale di un utente ed eventualmente inserirlo automaticamente in una presentazione.

- Premi::Model::Caricamento::Uploader.

Tipo, obiettivo e funzione del componente: Classe che rappresenta un algoritmo di caricamento di un elemento di tipo audio all'interno dello spazio personale di un utente. È uno dei componenti concreti del Design Pattern Strategy.

Università degli studi di Padova - 2014/2015



- Interfacce con e relazioni d'uso e da altre componenti:** Viene invocato per caricare elementi di tipo audio nello spazio personale di un utente ed eventualmente inserirlo automaticamente in una presentazione..

- Premi::Model::Caricamento::Uploader.

5.2 Controller

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

Relazioni d'uso di altre componenti: il componente è costituito dai package Presentazione e Utente, comunica con il Model per rendere possibile la gestione del profilo e la gestione delle presentazioni da parte dell'utente.

Package contenuti:

- Premi::Controller::Presentazione
- Premi::Controller::Utente

5.2.1 Premi::Controller::Presentazione

Tipo, obiettivo e funzione del componente: fanno parte di questo package tutte le classi di controller con cui interagiscono le pagine dedicate alla gestione delle presentazioni.

Relazioni d'uso di altre componenti: il package comunica con la view ricevendo chiamate da Premi::View::Pages::MobileEdit, Premi::View::Pages::DesktopEdit, Premi::View::Pages::Execution e Premi::View::Pages::Home. Comunica, invece, con il model inviando segnali e chiamate ai package Premi::Model::Inserimento, Premi::Model::Eliminazione, Premi::Model::Modifica, Premi::Model::Command, Premi::Model::Builder e alle classi Premi::Model::Invoker, Premi::Model::MongoHan

5.2.1.1 Premi::Controller::Presentazione::EditController

Tipo, obiettivo e funzione del componente: Lo scopo di questa classe è di gestire i segnali delle pagine Premi::View::Pages::DesktopEdit e Premi::View::Pages::MobileEdit verso il model.

Relazioni d'uso di altre componenti:

- `Premi.View.Pages.DesktopEdit` e `Premi.View.Pages.MobileEdit` -> costruiscono `EditController`, ne invocano i metodi passando i parametri degli oggetti modificati;
- `Premi::Model::Command <- EditController` costruisce un comando e lo dà in pasto a `Premi::Model::Invoker`;
- `Premi::Model::Invoker <- EditController` costruisce l'oggetto di classe `Invoker`. Invoca il metodo `execute()` di `Invoker`, passando come parametro un oggetto di classe `Command` oppure invoca il metodo `unexecute()` di `Invoker`;
- `Premi::Model::Presentazione::Loader <- EditController` costruisce l'oggetto di classe `Loader`, passando come parametro i riferimenti alla presentazione da caricare.

Interfacce con e relazioni d'uso e da altre componenti: La pagina DesktopEdit o la pagina MobileEdit invia a EditController un segnale comunicando l'avvenuta modifica o la rimozione di un elemento della presentazione o l'inserimento di un nuovo elemento. EditController istanzia un oggetto di classe Premi::Model::Command e lo dà in pasto a Premi::Model::Invoker. Eventualmente EditController può semplicemente annullare il comando appena eseguito invocando il metodo unexecute di Invoker. La pagina web può, inoltre richiedere il caricamento di una presentazione o la creazione di una nuova presentazione a EditController, che, tramite



5.2.1.2 Premi::Controller::Presentazione::HomeController

Relazioni d'uso di altre componenti:

- Interfacce con e relazioni d'uso e da altre componenti:** La pagina Home costruisce HomeController e richiede l'elenco delle presentazioni dell'utente.

Relazioni d'uso di altre componenti:

- Interfacce con e relazioni d'uso e da altre componenti:** La pagina Execution costruisce ExecutionController per caricare la presentazione.



Premi::View::Pages

IndexPage	Home	DesktopEdit
Profile	Execution	MobileEdit

Powered By Visual Paradigm Community Edition

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

5.3.1 Premi::View::Pages::IndexPage

Tipo, obiettivo e funzione del componente: la classe IndexPage definisce la struttura, e la conseguente visualizzazione, della pagina web che consente ad un utente di effettuare login e registrazione al sistema e di passare alla visualizzazione della classe Loader.

Relazioni d'uso di altre componenti: la classe IndexPage utilizza i metodi messi a disposizione dalla classe (((((((((((((((((CONTROLLER LOGIN))))))))))))), contenuta nel package Controller, per verificare i dati inseriti durante la fase di autenticazione, per inviare i dati relativi alla registrazione e per visualizzare eventuali errori emersi nella fase di autenticazione/registrazione.

Interfacce con e relazioni d'uso e da altre componenti: i metodi implementati nella classe `IndexPage` sono i seguenti:

- `IndexPage::Login` invia al controller i dati della login e, se corretti, manda alla pagina Home;
- `IndexPage::Subscribe` invia al controller i dati della registrazione e, se corretti, manda alla pagina Home;
- `IndexPage::Manifest` manda alla pagina Manifest.

Attività svolte e dati trattati: la classe definisce la struttura della pagina web che consente agli utenti di autenticarsi e registrarsi al sistema. Essa resta in attesa che un utente inserisca i dati necessari per l'autenticazione o la registrazione al sistema oppure che l'utente decida di andare nella pagina Loader.



Tipo, obiettivo e funzione del componente: la classe Home definisce la struttura, e la conseguente visualizzazione, della pagina web che mostra ad un utente le presentazioni presenti sul server e i comandi principali di gestione del profilo e gestione presentazioni.

- ((CONTROLLER ELIMINAZIONE PRESENT.)) per l'eliminazione delle presentazioni dal server;
- (((((((((CONTROLLER SCARICAMENTO MANIFEST)))))))) per scaricare una presentazione in locale;
- (((((((((CONTROLLER LOGOUT)))))))) per effettuare il logout.

- Home::Delete invia al controller l'id della presentazione da eliminare;
- Home::Download invia al controller l'id della presentazione da scaricare in locale;
- Home::Execute manda alla pagina Execution con l'id della presentazione da eseguire
- Home::NewSlideShow manda alla pagina SlideShowEdit con la richiesta di una nuova presentazione;
- Home::EditSlideShow manda alla pagina SlideShowEdit con l'id della presentazione da modificare;
- Home::Logout manda al controller la richiesta di logout e

Tipo, obiettivo e funzione del componente: la classe Manifest definisce la struttura, e la conseguente visualizzazione, della pagina web che mostra ad un utente le presentazioni scaricate in locale e da la possibilità di eseguirle.

- `Manifest::ExecuteManifest` esegue la presentazione selezionata utilizzando la pagina html già presente in locale e il framework impress.js;
- `Manifest::DeleteManifest` elimina la presentazione salvate in locale;



5.3.4 Premi::View::Pages::Profile

- ((((((CONTROLLER INSERIMENTO FILE MEDIA)))))) per il caricamento di file media nel server;
- ((((((CONTROLLER ELIMINAZ. FILE MEDIA)))))) per la loro eliminazione dal server;
- ((((((CONTROLLER PASSWORD)))))) per la modifica della password;
- ((((((CONTROLLER RINOMINA FILE MEDIA)))))) per rinominarli.

- Profile::ChangePassword invia al controller la nuova password;
- Profile::UploadMedia invia al controller le informazioni sul nuovo file media caricato sul server;
- Profile::DeleteMedia invia al controller l'id del file media da eliminare;
- Profile::RenameMedia invia al controller l'id e il nuovo nome del file media.

5.3.5 Premi::View::Pages::Execution

Interfacce con e relazioni d'uso e da altre componenti: i metodi implementati nella classe Execution sono gestiti dal framework Impress.js con l'aggiunta e la modifica delle seguenti 4 funzioni all'interno del framework:

- Execution::Next va al frame successivo della presentazione;
- Execution::Prev va al frame precedente;
- Execution::Bookmark va al frame con bookmark successivo.

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

5.3.6 Premi::View::Pages::Edit

Tipo, obiettivo e funzione del componente: la classe Edit è divisa in due sottoclassi, che sono visualizzazioni di pagine web diverse a seconda del dispositivo dalla quale viene visualizzata, Desktop o Mobile.

5.3.7 Premi::View::Pages::EditDesktop

Tipo, obiettivo e funzione del componente: la classe EditDesktop definisce la struttura, e la conseguente visualizzazione, della pagina web che mostra da dispositivo desktop ad un utente l'editor di modifica di una presentazione.

Relazioni d'uso di altre componenti: mandati principali di gestione del profilo e gestione presentazioni.

Relazioni d'uso di altre componenti: la classe Home, utilizza i metodi messi a disposizione dalle seguenti classi presenti nel package Controller:

- (((((((((CONTROLLER CARICA EDITOR)))))))) per caricare la presentazione da modificare;
- (((((((((CONTROLLER INSERIMENTO)))))))) per l'inserimento di nuovi elementi;
- (((((((((CONTROLLER SPOSTAMENTO)))))))) per lo spostamento di nuovi elementi;
- (((((((((CONTROLLER ELIMINAZIONE)))))))) per l'eliminazione elementi;
- (((((((((CONTROLLER MODIFICA ELEMENTI)))))))) per le modifiche effettuate agli elementi ;
- (((((((((CONTROLLER MODIFICA PERCORSO)))))))) per cambiare il percorso della presentazione.

Interfacce con e relazioni d'uso e da altre componenti: i metodi implementati nella classe EditDesktop sono i seguenti:

- EditDesktop::InsertFrame invia al controller la richiesta di inserimento di un nuovo frame e le coordinate di posizione;
- EditDesktop::InsertMedia invia al controller la richiesta di inserimento di un nuovo file media, le sue informazioni e le coordinate di posizione;



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



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