404NotFound

Premi: better than Prezi.

Specifica Tecnica

Versione	1.0
Redazione	Vegro Federico Cossu Mattia Camborata Marco Manuto Monica Rettore Andrea Gobbo Ismaele De Lazzari Enrico
Verifica	Manuto Monica Rettore Andrea
Responsabile	Gobbo Ismaele
Uso	Esterno
Stato	Formale
Ultima modifica	XXX
Lista di distribuzione	404NotFound prof. Tullio Vardanega prof. Riccardo Cardin Zucchetti S.p.a.

Registro delle modifiche

Versione	Autore	Data	Descrizione
1.0	Autore vers 1.0	12-01-2015	Scrittura versione finale ecc.
0.1	Autore vers 0.1	19-12-2014	Stesura scheletro ecc.

Tabella 1: Storico versioni del documento.

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del documento

Questo documento definisce la progettazione ad alto livello di Premi. Viene prima descritta la struttura generale del sistema e successivamente vengono analizzate le varie componenti software in relazione alle loro attività principali. Segue poi la descrizione delle tecnologie e dei Design Pattern$_G$ utilizzati, e un mockup$_G$ dell'interfaccia grafica lato utente.

1.2 Scopo del prodotto

Lo scopo del progetto è la realizzazione di un software di presentazione di slide non basato sul modello di PowerPoint$_G$, sviluppato in tecnologia HTML5$_G$ e che funzioni sia su desktop che su dispositivo mobile. Il software dovrà permettere la creazione da parte dell'autore e la successiva presentazione del lavoro, fornendo effetti grafici di supporto allo storytelling e alla creazione di mappe mentali.

1.3 Glossario

Al fine di evitare ogni ambiguità relativa al linguaggio e ai termini utilizzati nei documenti formali tutti i termini e gli acronimi presenti nel seguente documento che necessitano di definizione saranno seguiti da una "G" in pedice e saranno riportati in un documento esterno denominato Glossario.pdf. Tale documento accompagna e completa il presente e consiste in un listato ordinato di termini e acronimi con le rispettive definizioni e spiegazioni.

2 Definizione del Prodotto

2.1 Metodo e formalismo di specifica

Verrà qui esposta l'architettura di Premi ad alto livello seguendo un approccio top-down$_G$: verranno prima descritti i package$_G$ e le loro dipendenze e successivamente le singole classi contenute al loro interno. I diagrammi delle classi e dei package$_G$ seguono il formalismo UML$_G$2.0 e la struttura dei package segue una prassi ("best practice$_G$") di AngularJS$_G$ che propone una suddivisione dei componenti per funzionalità dell'applicazione in alternativa alla classica suddivisione Model-View-Controller$_G$, più difficile da mantenere per applicazioni di medie o grandi dimensioni. Per ulteriori approfondimenti consultare la guida al sito scotch.io oppure il tutorial di urigo:angular-meteor.
Si illustreranno poi i Design Pattern utilizzati nella fase di progettazione ad alto livello e si descriveranno le interazioni dell'utente con l'applicazione attraverso i diagrammi di attività$_G$.

2.2 Presentazione dell'architettura generale del sistema

I componenti sono stati suddivisi prima in base al loro contributo a specifiche funzionalità del software e solo successivamente per appartenenza ai ruoli del pattern MVC$_G$. Questo aumenta la chiarezza espositiva dei diagrammi, evita la creazione di package$_G$ contenenti un numero eccessivo di classi e aiuta a compiere verifiche mirate a singoli componenti.
È importante specificare che il framework AngularJS unisce view e controller attraverso una dichiarazione esterna a entrambi, che fa parte del meccanismo detto di *routing* o di reindirizzamento dell'utente; view e controller inoltre non sanno di essere collegati tra loro e comunicano attraverso un oggetto chiamato *$scope*. Questo rende l'architettura sia di tipo Model-View-Controller$_G$ che di tipo Model-View-ViewModel$_G$.
Per motivi di leggibilità $scope e routing non verranno rappresentati in modo esplicito nei diagrammi dei package e delle classi di questo documento, ma sono comunque da considerarsi impliciti nelle dipendenze tra i view e controller dei componenti.

3 Diagrammi dei Package

Di seguito vengono descritti componenti principali del sistema e le loro dipendenze.

——————————pkg.jpg

L'applicazione è costituita da un solo package$_G$ principale chiamato `Premi`; al suo interno sono presenti:

- `Premi` racchiude tutti i package$_G$ e la view e il controller principali;
- `Premi.UserManager` è il package$_G$ di gestione dei dati dell'utente;
- `Premi.Viewer` racchiude gli elementi necessari alla visualizzazione della presentazione nei vari contesti previsti;
- `Premi.Presentation` racchiude la struttura generale della presentazione;

- `Premi.PresentationManager` contiene gli elementi necessari alla gestione delle presentazioni da parte dell'utente;
- `Premi.Editor` è il package$_G$ dedicato alla modifica interna delle presentazioni; possiede al suo interno tre ulteriori package$_G$:
  - `Premi.Editor.FrameEditor` si occupa di creare, modificare o cancellare i Frame$_G$ contenuti nella presentazione;
  - `Premi.Editor.InfographicEditor` posiziona Frame$_G$ o altri elementi all'interno di un poster;
  - `Premi.Editor.TrailsEditor` ordina i Frame per la creazione di uno o più percorsi di presentazione.

4 Descrizione dei singoli componenti

Tipo, obiettivo e funzione del componente
Relazioni d'uso di altre componenti
Interfacce con le relazioni di uso da altre componenti

4.1 Premi

——————————premi-class.jpg
—-template——

4.1.1 Premi.

- **Nome:**
- **Tipo:** class
- **Package:** Premi.
- **Descrizione:**
- **Relazioni con altri componenti:**

—/template——

4.1.2 Premi.Utility

Nome: Utility

Tipo: class

Package: Premi

Descrizione: classe statica di utilità che fornisce strumenti per interagire con la base di dati

4.1.3 Premi.View

- **Nome:** View
- **Tipo:** class
- **Package:** Premi
- **Descrizione:** view generale che funge da sfondo per le altre view

4.1.4 Premi.PremiCtrl

- **Nome:** PremiCtrl
- **Tipo:** class
- **Package:** Premi
- **Descrizione:** controller generale che modifica la view principale in base alle scelte dell'utente
- **Relazioni con altri componenti:** la classe altera l'aspetto di `Premi.View` caricando le view di cui l'utente ha bisogno

4.2 Premi.UserManager

——————————premi-class.jpg

4.2.1 Premi.UserManager.User

——————————UserManager-class.jpg (mostra le relazioni che possiede ESTERNE al package)

- **Nome:** User
- **Tipo:** class
- **Package:** Premi.UserManger
- **Descrizione:** classe che definisce un utente e fornisce le funzionalità necessarie alla sua creazione, al suo login e ad un eventuale cambio di password
- **Relazioni con altri componenti:** si collega a `Premi.Utility` per le interazioni con la base di dati

4.2.2 Premi.UserManager.View

- **Nome:** View
- **Tipo:** class
- **Package:** Premi.UserManger
- **Descrizione:** vista generale del package `UserManager`. Puo' contenere le view abbinate alle funzionalità di `User`

4.2.3 Premi.UserManager.UserManagerCtrl

- **Nome:** UserManagerCtrl
- **Tipo:** class
- **Package:** Premi.UserManager
- **Descrizione:** controller generale del package `UserManager` dedicato a fornire alla view strumenti per l'interazione con l'utente
- **Relazioni con altri componenti:** si collega a `Premi.UserManager.View` per mostrare i dati dell'utente che va a prelevare tramite Premi.UserManager.User

4.2.4 Premi.UserManager.LoginView

- **Nome:** LoginView
- **Tipo:** class
- **Package:** Premi.UserManager
- **Descrizione:** view che permette all'utente di effettuare la login

4.2.5 Premi.UserManager.LoginController

- **Nome:** LoginController
- **Tipo:** class
- **Package:** Premi.UserManager
- **Descrizione:** fornisce gli strumenti necessari alla view per effettuare la login
- **Relazioni con altri componenti:** con `Premi.UserManager.LoginView` e con `Premi.UserManager.User` per trasmettere le informazioni relative al login

4.2.6 Premi.UserManager.RegistrationView

- **Nome:** RegistrationView
- **Tipo:** class
- **Package:** Premi.UserManager
- **Descrizione:** view che permette all'utente di registrarsi nel sistema

4.2.7 Premi.UserManager.RegistrationController

- **Nome:** RegistrationController
- **Tipo:** class
- **Package:** Premi.UserManager
- **Descrizione:** fornisce gli strumenti necessari alla view per registrare l'utente
- **Relazioni con altri componenti:** con `Premi.UserManager.RegistrationView` e con `Premi.UserManager.User` per trasmettere le informazioni relative alla registrazione

4.2.8 Premi.UserManager.ChangePasswordView

- **Nome:** ChangePasswordView
- **Tipo:** class
- **Package:** Premi.UserManager
- **Descrizione:** view che permette all'utente di cambiare la propria password

4.2.9 Premi.UserManager.ChangePasswordController

- **Nome:** ChangePassword
- **Tipo:** class
- **Package:** Premi.UserManager
- **Descrizione:** fornisce gli strumenti necessari alla view per cambiare la password dell'utente
- **Relazioni con altri componenti:** con `Premi.UserManager.ChangePasswordView` e con `Premi.UserManager.User` per trasmettere le informazioni relative al cambio password

4.3 Premi.Viewer

——————————viewer-class.jpg

4.3.1 Premi.Viewer.View

- **Nome:** View
- **Tipo:** class
- **Package:** Premi.Viewer
- **Descrizione:** view che mostra la presentazione all'utente, offrendo funzionalità dipendenti dalla visibilità(pubblica o privata) o dal contesto in cui si trova(presentazione live)

4.3.2 Premi.ViewerCtrl

- **Nome:** ViewerCtrl
- **Tipo:** class
- **Package:** Premi.Viewer
- **Descrizione:** abilita funzionalità nella view in base all'utente
- **Relazioni con altri componenti:** con `Premi.Viewer.View`, con `Premi.UserManager.User` per verificare se l'utente è il proprietario della presentazione, e con la libreria esterna `ImpressJs` per aggiungere animazioni alla presentazione

4.4 Premi.Presentation

————————-presentation-class.jpg

4.4.1 Premi.Presentation.GraphicObject

- **Nome:** GraphicObject
- **Tipo:** *abstract class*
- **Package:** Premi.Presentation
- **Descrizione:** rappresenta gli oggetti grafici nella presentazione

4.4.2 Premi.Presentation.GoContent

- **Nome:** GoContent
- **Tipo:** *abstract class*
- **Package:** Premi.Presentation
- **Descrizione:** rappresenta gli oggetti grafici con contenuto di una presentazione
- **Relazioni con altri componenti:** estende `Premi.Presentation.GraphicObject`

4.4.3 Premi.Presentation.Text

- **Nome:** Text
- **Tipo:** class
- **Package:** Premi.Presentation
- **Descrizione:** rappresenta un'area di testo nella presentazione
- **Relazioni con altri componenti:** estende `Premi.Presentation.GoContent`

4.4.4 Premi.Presentation.Image

- **Nome:** Image
- **Tipo:** class
- **Package:** Premi.Presentation
- **Descrizione:** rappresenta un'immagine nella presentazione
- **Relazioni con altri componenti:** estende `Premi.Presentation.GoContent`

4.4.5 Premi.Presentation.Shape

- **Nome:** Shape
- **Tipo:** class
- **Package:** Premi.Presentation
- **Descrizione:** rappresenta una figura nella presentazione
- **Relazioni con altri componenti:** estende `Premi.Presentation.GoContent`

4.4.6 Premi.Presentation.Frame

- **Nome:** Frame
- **Tipo:** class
- **Package:** Premi.Presentation
- **Descrizione:** rappresenta un frame nella presentazine
- **Relazioni con altri componenti:** estende `Premi.Presentation.GraphicObject` e possiede un insieme di oggetti `Premi.Presentation.GoContent`

4.4.7 Premi.Presentation.Trail

- **Nome:** Trail
- **Tipo:** class
- **Package:** Premi.Presentation
- **Descrizione:** rappresenta un percorso nella presentazione
- **Relazioni con altri componenti:** possiede una lista di riferimenti ad oggetti di tipo `Premi.Presentation.Frame`, ma non è in possesso degli oggetti stessi

———————-dipendenze di presentation.jpg CITARE L'IMMAGINE DELLE DIPENDENZE CON DI PRESENTATION Completo-Dettagliato.jpg

4.4.8 Premi.Presentation.Presentation

- **Nome:** Presentation
- **Tipo:** class
- **Package:** Premi.Presentation
- **Descrizione:** rappresenta una presentazione
- **Relazioni con altri componenti:** possiede una lista di oggetti `premi.Presentation.Frame`, identifica l'appartenenza di una presentazione ad un determinato `Premi.UserManager.User` e si collega alla base di dati tramite la classe statica `Premi.Utility`

4.5 Premi.PresentationManager

——-usermanager-classe.jpg

4.5.1 Premi.PresentationManager.Export

- **Nome:** Export
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** permette di esportare la presentazione in formato poster
- **Relazioni con altri componenti:** preleva le informazioni da `Premi.Presentation.Presentat` per costruire il poster

4.5.2 Premi.PresentationManager.Portable

- **Nome:** Portable
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** permette di rendere portabile una presentazione, e quindi di essere indipendente dal software Premi e di essere caricata come pagina HTML da un browser
- **Relazioni con altri componenti:**

4.5.3 Premi.PresentationManager.View

- **Nome:** View
- **Tipo:** class
- **Package:** Premi.PresentationController
- **Descrizione:** è la view generale delle operazioni che l'utente puo' effettuare sul

4.5.4 Premi.PresentationManager.PresentationManagerCtrl

- **Nome:** PresentationManagerCtrl
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** fornisce alla view gli strumenti necessari alla gestione delle presentazioni
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.PresentationManager.Vie`

4.5.5 Premi.PresentationManager.RemoveView

- **Nome:** RemoveView
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** mostra una finestra di conferma eliminazione della presentazione selezionata

4.5.6 Premi.PresentationManager.RemoveController

- **Nome:** RemoveController
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** fornisce alla view gli strumenti per rimuovere la presentazione o annullare l'azione
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.PresentationManager.Rem` e sfrutta le funzionalità di `Premi.Presentation.Presentation` per l'eliminazione della presentazione

4.5.7 Premi.PresentationManager.PublicView

- **Nome:** PublicView
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** mostra una finestra di conferma per rendere pubblica o privata una presentazione

4.5.8 Premi.PresentationManager.PublicController

- **Nome:** PublicController
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** fornisce alla view gli strumenti per rendere pubblica o privata una presentazione
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.PresentationManager.Pub` e sfrutta le funzionalità di `Premi.Presentation.Presentation` per accedere alla base di dati

4.5.9 Premi.PresentationManager.EditTitleDescrView

- **Nome:** EditTitleDescrView
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** mostra una finestra per la modifica del titolo e della descrizone della presentazione

4.5.10 Premi.PresentationManager.EditTitleDescrController

- **Nome:** EditTitleDescrController
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** fornisce alla view la possibilità di modificare titolo e descrizione della presentazione
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.PresentationManager.Edi` e sfrutta le funzionalità di `Premi.Presentation.Presentation` per accedere alla base di dati

4.5.11 Premi.PresentationManager.NewView

- **Nome:** NewView
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** mostra una finestra per la creazione di una nuova presentazione

4.5.12 Premi.PresentationManager.NewController

- **Nome:** NewController
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** fornisce alla view gli strumenti per creare una nuova presentazione
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.PresentationManager.New` e sfrutta le funzionalità di `Premi.Presentation.Presentation` per accedere alla base di dati

4.5.13 Premi.PresentationManager.ListPresView

- **Nome:** ListPresView
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** mostra all'utente la lista delle sue presentazioni, e bottoni aggiuntivi per accedere alle altre funzionalità del package

4.5.14 Premi.PresentationManager.ListPresController

- **Nome:** ListPresController
- **Tipo:** class
- **Package:** Premi.PresentationManager
- **Descrizione:** fornisce alla view una lista preformata delle presentazioni dell'utente
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.PresentationManager.Lis` e sfrutta le funzionalità di `Premi.Presentation.Presentation` per recuperare la lista delle presentazioni

4.6 Premi.Editor

4.6.1 Premi.Editor.View

- **Nome:** View
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** view generica dell'editor che funge da contenitore

4.6.2 Premi.Editor.EditorCtrl

- **Nome:** EditorCtrl
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** fornisce alla view le funzionzalità per la modifica di una presentazione
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Editor.EditorView` e sfrutta `Premi.Presentation.Presentation` per recuperare la presentazione dalla base di dati

4.6.3 Premi.Editor.TextView

- **Nome:** TextView
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** mostra le opzioni per l'aggiunta di un'area di testo

4.6.4 Premi.Editor.TextController

- **Nome:** TextController
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** fornisce alla view le funzionalità per creare e modificare un'area di testo
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Editor.TextView`

4.6.5 Premi.Editor.ShapeView

- **Nome:** ShapeView
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** mostra le opzioni per l'aggiunta di uno shape

4.6.6 Premi.Editor.ShapeController

- **Nome:** ShapeController
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** fornisce alla view le funzionalità per creare e modificare uno shape
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Editor.ShapeView`

4.6.7 Premi.Editor.ImageView

- **Nome:** ImageView
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** mostra le opzioni per l'aggiunta di un'immagine

4.6.8 Premi.Editor.ImageController

- **Nome:** ImageController
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** fornisce alla view le funzionalità per aggiungere un'immagine
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Editor.ImageView`

4.6.9 Premi.Editor.StyleView

- **Nome:** StyleView
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** permette la modifica dello stile di un Frame oppure dello sfondo dell'infografica

4.6.10 Premi.Editor.StyleController

- **Nome:** StyleController
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** fornisce alla view le funzionalità per la modifica dello stile
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Editor.StyleView`

4.6.11 Premi.Editor.FrameListView

- **Nome:** FrameListView
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** view di base per la rappresentazione di una lista dei Frame contenuti all'interno della presentazione

4.6.12 Premi.Editor.FrameListController

- **Nome:** FrameListController
- **Tipo:** class
- **Package:** Premi.Editor
- **Descrizione:** fornisce alla view le funzionalità per rappresentazione della lista dei Frame
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Editor.FrameListView`

4.7 Premi.Editor.FrameEditor

4.7.1 Premi.Editor.FrameEditor.View

- **Nome:** View
- **Tipo:** class
- **Package:** Premi.Editor.FrameEditor.View
- **Descrizione:** è la view generale della fase di modifica dei Frame

4.7.2 Premi.Editor.FrameEditor.FrameEditorCtrl

- **Nome:** FrameEditorCtrl
- **Tipo:** class
- **Package:** Premi.Editor.FrameEditor.FrameEditorCtrl
- **Descrizione:** fornisce alla view le funzionalità per la gestione delle altre view contenute al suo interno
- **Relazioni con altri componenti:** è il controller dedicato di `Premi.Editor.FrameEditor.Fram`

4.7.3 Premi.

- **Nome:**
- **Tipo:** class
- **Package:** Premi.
- **Descrizione:**
- **Relazioni con altri componenti:**

4.7.4 Premi.

- **Nome:**
- **Tipo:** class
- **Package:** Premi.
- **Descrizione:**
- **Relazioni con altri componenti:**

4.7.5 Premi.

- **Nome:**
- **Tipo:** class
- **Package:** Premi.
- **Descrizione:**
- **Relazioni con altri componenti:**

5 Diagrammi delle attività

6 Stime di fattibilità e di bisogno di risorse

7 Tracciamento della relazione componenti - requisiti

8 Descrizione delle tecnologie utilizzate

9 Design Pattern

9.1 Design Pattern Architetturali

9.1.1 MVC - Model View Controller

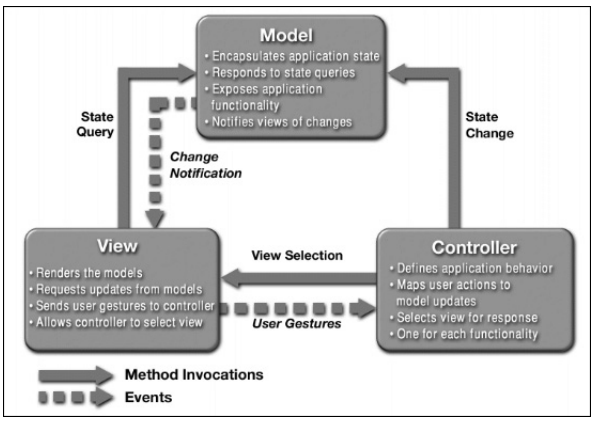

Figura 1: Diagramma del design pattern MVC

- **Descrizione:** Il design pattern$_G$ MVC permette un disaccoppiamento totale della View dalle logiche di manipolazione del Modello tramite l'introduzione di un componente, il Controller, che funga da intermediario e da coordinatore in risposta alle interazioni con l'utente. Si individuano tre componenti:
  - Model: dati di business e regole di accesso;
  - View: rappresentazione grafica. Visualizza i dati contenuti nel model e raccoglie gli input dell'utente;
  - Controller: reazioni della UI agli input utente. Interagisce con il model in base ai comandi dell'utente (attraverso la View);
- **Motivazione:** Lo scopo di molte applicazioni è quello di recuperare dati e visualizzarli in maniera opportuna a seconda delle esigenze degli utenti. Poiché il flusso chiave di informazione avviene tra il dispositivo su cui sono memorizzati i dati e l'interfaccia utente, si è portati a legare insieme queste due parti per ridurre la quantità di codice e migliorare le performance dell'applicazione. Questo approccio, apparentemente naturale, presenta alcuni problemi significativi; uno di questi è che l'interfaccia utente tende a cambiare più in fretta rispetto al sistema di memorizzazione dei dati. C'è la necessità, quindi, di rendere modulari le funzionalità dell'interfaccia utente in maniera tale da poter facilmente modificare le singole parti. L'intento del pattern MVC è di disaccoppiare il più possibile tra loro le parti dell'applicazione adibite al controllo, all'accesso ai dati e alla presentazione, apportando diversi vantaggi:
  - indipendenza tra i business data (model) la logica di presentazione (view) e quella di controllo (controller);

- separazione dei ruoli e delle relative interfacce;
- viste diverse per il medesimo model;
- semplice il supporto per nuove tipologie di client: bisogna scrivere la vista ed il controller appropriati riutilizzando il model esistente.

- **Applicablità:** Il pattern MVC può essere utilizzato nei seguenti casi:
  - Quando si vuole trattare un gruppo di oggetti come un oggetto singolo;
  - Quando si vuole disaccoppiare View e Model instaurando un protocollo di sottoscrizione e notifica tra loro;
  - Quando si vogliono agganciare più View a un Model per fornire più rappresentazioni del Model stesso.

9.1.2 MVVM - Model View VievModel

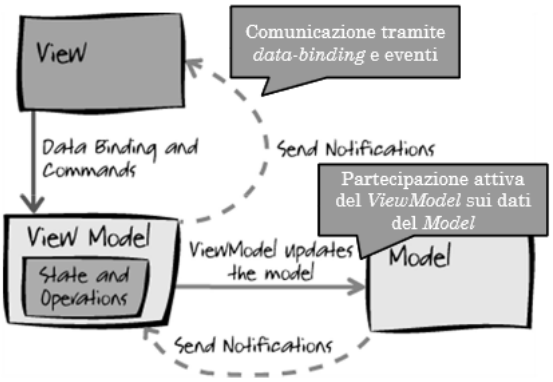

Figura 2: Diagramma del design pattern MVVM

- **Descrizione:** Il design pattern$_G$ MVVM è una variante del pattern MVC che propone un ruolo più attivo della View, la quale è in grado di gestire eventi, eseguire operazioni ed effettuare il data-binding. In questo contesto, quindi, alcune delle funzionalità del Controller vengono inglobate nella View, la quale si appoggia su un'estensione del Model: il ViewModel. Come per il pattern MVC, anche quì si individuano tre componenti:
  - Model: dati di business e regole di accesso;
  - View: rappresentazione grafica. Visualizza i dati contenuti nel model e raccoglie gli input dell'utente;
  - ViewModel: Model esteso con funzionalità per la manipolazione dei dati e per l'interazione con la View.

- **Motivazione:** Il cuore del funzionamento di questo pattern è la creazione di un componente (ViewModel) che rappresenta, in modo astratto, tutte le informazioni e i comportamenti della corrispondente View; quest'ultima si limita a

visualizzare graficamente quanto esposto dal ViewModel, a riflettere i propri cambi di stato nel ViewModel stesso oppure ad attivare suoi comportamenti. L'uso di questo pattern richiede la presenza, nella tecnologia di UI, di un'ottimo meccanismo di data-binding (tipicamente bidirezionale) o, in alternativa, di un strato di sincronizzazione fra la View e il ViewModel. E' compito del ViewModel, però, offrire alla View una "superficie esterna" il più possibile ben fruibile, in modo che la sincronizzazione dello stato possa essere fatta senza introdurre logiche decisionali che rendano necessario un test specifico.

- **Applicablità:** Il pattern MVVM può essere utilizzato nei seguenti casi:
  - Quando si vuole trattare un gruppo di oggetti come un oggetto singolo;
  - Quando si vuole disaccoppiare View e Model instaurando un protocollo di sottoscrizione e notifica tra loro;

9.1.3 Dependecy Injection

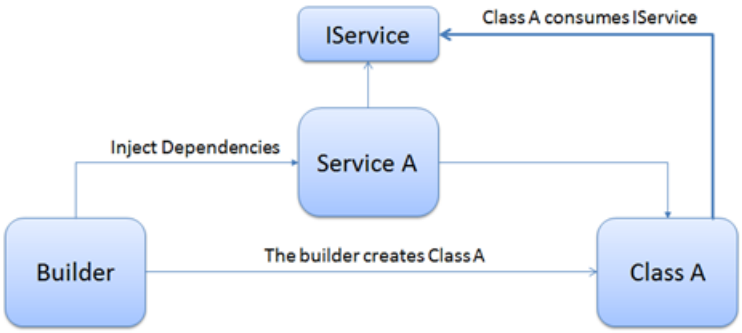

Figura 3: Diagramma del design pattern Dependency Injection

- **Descrizione:** Il design pattern$_G$ Dependecy Injection ha lo scopo di semplificare lo sviluppo e migliorare la testabilità del software, permettendo la separazione del comportamento di una componente dalla risoluzione delle sue dipendenze; Il pattern Dependency Injection coinvolge almeno tre elementi:
  - una componente *dipendente*;
  - la dichiarazione delle *dipendenze* del componente, definite come *interface contracts*;
  - un *injector* (chiamato anche *provider* o *container*) che crea, a richiesta, le istanze delle classi che implementano delle *dependency interfaces*.
- **Motivazione:** Il collegamento di due o più componenti in modo esplicito ne aumenta l'accoppiamento, causando una scarsa manutenibilità del software e complicando le fasi di unit testing. Inoltre un componente soggetto a dipendenze risulta meno predisposto al riutilizzo dello stesso. La dependency injection prende il controllo su tutti gli aspetti di creazione degli oggetti e delle loro dipendenze. Normalmente, senza l'utilizzo di questa tecnica, se un oggetto necessita

di accedere ad un particolare servizio, l'oggetto stesso si prende la responsabilità di gestirlo, o avendo un diretto riferimento al servizio, o individuandolo con un Service Locator che gli restituisce un riferimento ad una specifica implementazione del servizio. Con l'utilizzo della dependency injection, l'oggetto ha in sé solamente una proprietà che può ospitare un riferimento a quel servizio e, quando l'oggetto viene istanziato, un riferimento ad una implementazione di questo servizio gli viene iniettata dal framework$_G$ esterno, senza che il programmatore che crea l'oggetto sappia nulla sul posizionamento del servizio o altri dettagli dello stesso.

- **Applicablità:** Il pattern Dependency Injection può essere utilizzato nei seguenti casi:
  - Quando si ha la necessità di collegare più componenti cercando di minimizzare il livello di accoppiamento;
  - Quando si lavora su progetti basati sul Test Driven.

9.2 Design Pattern Creazionali

9.2.1 Factory Method

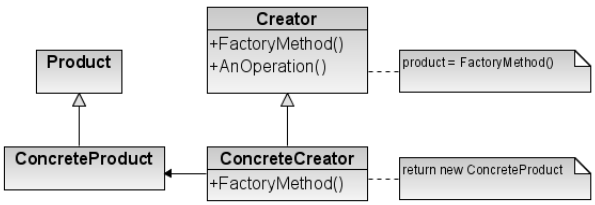

Figura 4: Diagramma del design pattern Factory Method

- **Descrizione:** il design pattern$_G$ Factory Method indirizza il problema della creazione di oggetti senza specificarne l'esatta classe, fornendo un'interfaccia per creare un oggetto, ma lasciando che le sottoclassi decidano quale oggetto istanziare. Definisce un'interfaccia (Creator) per ottenere una nuova istanza di un oggetto (Product). Delega ad una classe derivata (ConcreteCreator) la scelta di quale classe istanziare (ConcreteProduct);
- **Motivazione:** la creazione di un oggetto può, spesso, richiedere processi complessi la cui collocazione all'interno della classe di composizione potrebbe non essere appropriata. Esso può, inoltre, comportare duplicazione di codice, richiedere informazioni non accessibili alla classe di composizione, o non fornire un sufficiente livello di astrazione. Il Factory Method indirizza questi problemi definendo un metodo separato per la creazione degli oggetti. Tale metodo può essere ridefinito dalle sottoclassi per definire il tipo derivato di prodotto che verrà effettivamente creato;
- **Applicablità:** Il pattern Factory Method si puù utilizzare nei seguenti casi:

- Quando si desidera che la creazione di un oggetto non precluda il suo riuso senza una significativa duplicazione di codice;
- Quando si desidera che la creazione di un oggetto non richieda l'accesso ad informazioni o risorse che non dovrebbero essere contenute nella classe di composizione;
- Quando si desidera che la gestione del ciclo di vita degli oggetti gestiti debba essere centralizzata in modo da assicurare un comportamento consistente all'interno dell'applicazione.

10 Mockup Interfaccia [FACOLTATIVO]