QUIZZIPEDIA

team404swe@gmail.com

Specifica Tecnica 2.0

Informazioni sul documento	
Nome Documento	Specifica Tecnica 2.0
Versione	2.0
Uso	Esterno
Data Creazione	22 aprile 2016
Data Ultima Modifica	4 giugno 2016
Redazione	A. Multineddu - L. Alessio - D. Bortot
Verifica	Alex Beccaro
Approvazione	Martin V. Mbouenda
Committente	Zucchetti SPA
Lista di distribuzione	Prof. Vardanega Tullio TEAM404

Registro delle modifiche

Versione	Autore	Data	Descrizione
1.0.5	Davide Bortot (Progettista)	01/06/2016	Ampliamento sezione "Risorse necessarie" (§4) con MongoDB, Meteor, Openshift.
1.0.4	Luca Alessio (Analista)	01/06/2016	Ampliamento sezione "Tecnologie utilizzate" (aggiunti paragrafi AngularJS, Meteor, MongoDB).
1.0.3	Davide Bortot (Progettista)	30/05/2016	Riformattamento del documento e ristrutturazione dei file LaTeX.
1.0.2	Davide Bortot (Progettista)	27/05/2016	Correzione e contestualizzazione dei design pattern individuati in §9.
1.0.1	Luca Alessio (Analista)	24/05/2016	Correzione diagrammi di attività e revisione sintattica di diversi paragrafi.
1.0	Martin Mbouenda (responsabile)	16/05/2016	Approvazione del documento.
0.1.0	A. Beccaro (Verificatore)	13/05/2016	Verifica completa del documento.
0.0.19	A. Multineddu (Progettista)	12/05/2016	Stesura della tracciamento requisiti - componenti".
0.0.18	Davide Bortot (Progettista)	11/05/2016	Stesura della tracciamento componenti - requisiti".
0.0.17	Luca Alessio (Progettista)	12/05/2016	Termine stesura sezione diagrammi e revisione/ampliamento di vari paragrafi
0.0.16	Davide Bortot (Progettista)	10/05/2016	Stesura della sezione "Design pattern utilizzati".
0.0.15	Davide Bortot (Progettista)	08/05/2016	Inseriti grafici dei Diagrammi di Sequenza. Aggiunta tecnologia "Materialize" alla sezione "Tecnologie e strumenti utilizzati".
0.0.14	Luca Alessio (Progettista)	07/05/2016	Stesura sezione "Diagrammi di Sequenza" per la parte descrittiva.
0.0.13	Davide Bortot (Progettista)	06/05/2016	Integrara la sezione "Diagrammi delle classi del Presenter" con l'aggiunta dei grafici UML.
0.0.12	Andrea Multineddu (Progettista)	06/05/2016	Aggiunte immagini sezione "Diagrammi delle attività".
0.0.11	Davide Bortot (Progettista)	05/05/2016	Completata la parte descrittiva della sezione "Diagrammi delle classi del Presenter". Riadattamento allo stile del documento di §4.3. Inseriti grafici UML dei Package del Model e della View.
0.0.10	Luca Alessio (Progettista)	04/05/2016	Stesura sezione "Diagrammi delle attività" e correzioni varie su altre sezioni.
0.0.9	Davide Bortot (Progettista)	03/05/2016	Stesura sezione "Package della componente Presenter", inserito rispettivo grafico UML, e prima stesura di "Diagrammi delle classi del Presenter".
0.0.8	Andrea Multineddu (Progettista)	01/05/2016	Prima stesura sezioni "Package della componente Model" e "Diagrammi delle classi del Model".

0.0.7	Luca Alessio (Progettista)	30/04/2016	Prima stesura sezioni "Package della componente View" e "Diagrammi delle classi del View".
0.0.6	Davide Bortot (Progettista)	28/04/2016	Stesura delle sezioni §2.2, §2.3, §2.4 con la descrizione d'alto livello delle componenti MVP.
0.0.5	Luca Alessio (Progettista)	27/04/2016	Stesura parziale sezione "Tecnologie e strumenti utilizzati"
0.0.4	Davide Bortot (Progettista)	25/04/2016	Ampliata la sezione "Architettura generale del sistema".
0.0.3	Luca Alessio (Progettista)	24/04/2016	Stesura parziale sezione "Architettura generale del sistema"
0.0.2	Davide Bortot (Progettista)	23/04/2016	Strutturazione iniziale del documento e stesura sezione introduttiva
0.0.1	Marco Crivellaro (Progettista)	22/04/2016	Creazione documento.

Indice

Elenco delle figure

Elenco delle tabelle

Sommario

Questo documento, redatto dal gruppo **Team404**, contiene la descrizione dell'architettura software sulla quale verrà sviluppato il progetto Quizzipedia, commissionato da Zucchetti S.p.A.. Scopo del documento è quello di illustrare le scelte progettuali che il gruppo ha deciso di seguire per realizzare il prodotto.

1 Introduzione

1.1 Scopo del documento

Lo scopo della Specifica Tecnica è quello di illustrare le scelte progettuali che il gruppo ha deciso di seguire nella realizzazione del prodotto. Vengono presentati, pur restando ad alto livello, la gerarchia dei package, le loro interazioni e le principali classi in essi contenute, specificandone lo scopo e l'utilizzo. Verranno presentati inoltre i design pattern utilizzati nell'architettura.

1.2 Scopo del prodotto

Il progetto **Quizzipedia** ha come obiettivo lo sviluppo di un sistema software basato su tecnologie Web (Javascript$_G$, Node.js$_G$, HTML5$_G$, CSS3$_G$) che permetta la creazione, gestione e fruizione di questionari. Il sistema dovrà quindi poter archiviare i questionari suddivisi per argomento, le cui domande dovranno essere raccolte attraverso uno specifico linguaggio di markup (Quiz Markup Language) d'ora in poi denominato QML$_G$. In un caso d'uso a titolo esemplificativo, un "esaminatore" dovrà poter costruire il proprio questionario scegliendo tra le domande archiviate, ed il questionario così composto sarà presentato e fruibile all' "esaminando", traducendo l'oggetto QML in una pagina HTML$_G$, tramite un'apposita interfaccia web. Il sistema presentato dovrà inoltre poter proporre questionari preconfezionati e valutare le risposte fornite dall'utente finale.
Per un'analisi più precisa ed approfondita del progetto si rimanda al documento "*analisi_dei_requisiti_3.0.pdf*".

1.3 Glossario

Viene allegato un glossario nel file "*glossario_3.0.pdf*" nel quale viene data una definizione a tutti i termini che in questo documento appaiono con il simbolo '$_G$' a pedice.

1.4 Riferimenti

1.4.1 Normativi

- Capitolato d'appalto Quizzipedia:
  http://www.math.unipd.it/~tullio/IS-1/2015/Progetto/C5.pdf
- Norme di Progetto: "*norme_di_progetto_3.0.pdf*"

1.4.2 Informativi

- Corso di Ingegneria del Software anno 2015/2016:
  http://www.math.unipd.it/~tullio/IS-1/2015/
- Regole del progetto didattico:
  http://www.math.unipd.it/~tullio/IS-1/2015/Dispense/PD01.pdf http://www.math.unipd.it/~tullio/IS-1/2015/Progetto/

2 Specifiche del prodotto

I contenuti della specifica saranno presentati seguendo l'approccio top-down: dalla descrizione macroscopica del sistema si scenderà sempre più in dettaglio passando alla descrizione delle singole componenti. Verranno inoltre descritti i design pattern utilizzati e come essi sono stati applicati. Per agevolare la comprensione si è scelto di utilizzare i diagrammi dei package, delle classi, di attività e di sequenza, descritti attraverso lo standard UML 2.0.

2.1 Architettura generale del sistema

Il sistema Quizzipedia è di tipo *client-server*; il *client* fornisce all'utente un'interfaccia web su browser per la creazione e fruizione di questionari, mentre il lato *server* si occupa di gestire e salvare i dati su DBMS PostgreSQL. La base di dati raccoglie principalmente quesiti memorizzati in QML, che su richiesta verranno elaborati da un interprete. Il suddetto interprete processa l'input QML, precedentemente validato nella forma da un parser apposito, traducendolo in linguaggio HTML5 visualizzabile da browser. Nella realizzazione del sistema Quizzipedia verrà adottato il design pattern *Model View Presenter* nella sua variante *Passive View*.

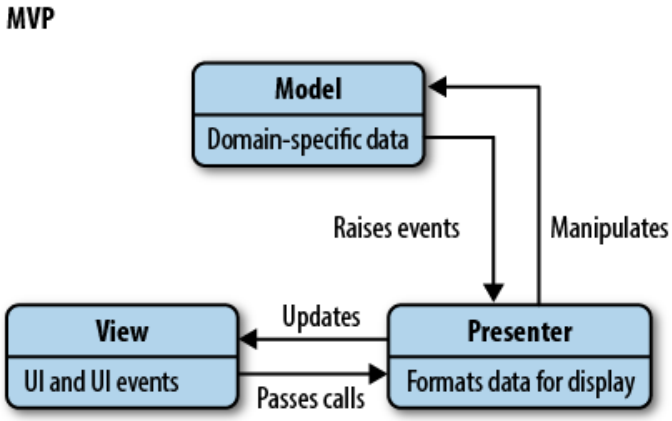


Figura 1: Il design pattern Model View Presenter (Passive View)

- **Model**: definisce l'organizzazione dei dati e ne specifica le modalità di accesso. Nel sistema Quizzipedia è situato nella parte server che opera sulla sottostante base di dati. Fa parte del Model anche la componente *Parser* che opera sui dati prima che essi vengano salvati nel DBMS.
- **View (Passive)**: rappresenta l'interfaccia grafica presentata all'utilizzatore, la quale visualizza i dati e cattura le interazioni dell'utente. L'aggettivo "Passive" indica che la View non è responsabile del proprio aggiornamento al variare del Model, compito che ricade sul Presenter.
- **Presenter**: controlla la View e ne gestisce il comportamento in reazione alle interazioni dell'utente, interagisce di conseguenza col Model per ottenere i dati necessari. Una volta ottenuti i dati si preoccupa di aggiornare la View. Nel sistema Quizzipedia implementa la parte logica, affiancata a quella grafica, del Client. Realizza un totale disaccoppiamento tra Model e View, controllandone i flussi di comunicazione.

Altre possibili architetture che sono state prese in considerazione sono definite dai design pattern *Model View Controller (MVC)* con *Front Controller*, *Model View Presenter (MVP)*

nelle varianti *Presentation Model* e *Supervising Controller*, e il design pattern *Model View ViewModel (MVVM)*.

- La prima pone il Controller, componente simile al Presenter, nella parte server del sistema. Quest'opzione è stata scartata per evitare di aumentare troppo la complessità del lato server, ponendo invece il Presenter dal lato client, così da redistribuire responsabilità e carico di lavoro.
- Il *Presentation Model* invece è affine al design pattern scelto, ma impone che sia la componente View ad aggiornarsi autonomamente al variare del Model. Proprio per questo motivo si è deciso di scartarla in favore del Passive View: per disaccoppiare completamente le componenti Model e View, e per alleggerire ulteriormente quest'ultima concentrandone tutta la parte logica nel Presenter.
- Il *Supervising Controller* propone che la View si aggiorni autonomamente nel caso di piccole modifiche (tramite data-binding col Model), lasciando le manipolazioni più complicate al Presenter; è stato scartato in favore del disaccoppiamento totale tra View e Model.
- Il pattern *MVVM* prevede di creare per ogni View un ViewModel, che rappresenta tutte le informazioni e i comportamenti della corrispondente View. La View si limita infatti, a visualizzare graficamente quanto esposto dal ViewModel, a riflettere in esso i suoi cambi di stato oppure ad attivarne dei comportamenti (tramite data-binding). Tale architettura è indicata per applicazioni particolarmente dinamiche in cui View e Model devono essere costantemente aggiornati. Non è questo il caso di Quizzipedia.

2.2 Descrizione del componente Model

Il Model, situato nella parte server del sistema svolge le seguenti funzioni:

- Interagisce con un database PostgreSQL nel quale vengono salvati i dati del sistema (ad es. domande, statistiche, utenti). Fornisce quindi adeguate funzionalità di salvataggio e caricamento da database di tali dati.
- Al momento del salvataggio di una nuova domanda esegue il *parsing* del codice QML tramite un componente chiamato *Parser*. Se la domanda è sintatticamente corretta può essere salvata nel database.
- Offre un'interfaccia logica di accesso al Presenter attraverso la quale richiedere dati e operazioni su di essi.

2.3 Descrizione del componente View

La componente View rappresenta l'interfaccia grafica che visualizza i dati del Model e inoltra i comandi dell'utente (o gli eventi da esso generati) al Presenter che si occuperà di gestire tali richieste sui dati interagendo col Model; la View si occupa quindi solamente della rappresentazione grafica dei dati e non ha alcun contatto diretto con essi. Essendo un'interfaccia web verrà realizzata tramite HTML5 e CSS3 per le parti statiche e con l'utilizzo di Javascript per le parti dinamiche. Per assicurare uno stile coerente tra le pagine web e migliorare l'adattabilità a piattaforme mobile verrà utilizzato il framework *Materialize*.

2.4 Descrizione del componente Presenter

Il Presenter ricopre tre ruoli fondamentali: recepire ed elaborare gli input dell'utente, comunicare col Model, ed aggiornare il View con i dati ottenuti. Per poterlo fare possiede le seguenti caratteristiche:

- Conosce i riferimenti alle altre due componenti. Il Presenter è l'unica componente che conosce entrambe le altre e facendo da singolo tramite tra Model e View permette il loro totale disaccoppiamento;
- E' in grado di elaborare gli input della View e tradurli in azioni sul Model. Viceversa ad ogni modifica del Model si preoccupa di aggiornare di conseguenza la View.
- E' responsabile della traduzione delle domande dal formato QML a formato HTML visualizzabile da browser, tramite un componente *Interpreter*. Tale funzionalità viene richiesta ogni qualvolta il Presenter richiede e riceve dal Model una domanda in formato QML.
- Possiede dei gestori che possano modificare l'aspetto della View in reazione all'interazione dell'utente o ai dati ricevuti dal Model. Al suo interno il Presenter contiene delle classi che in risposta ad un evento, quale l'interazione dell'utente con la View o il ricevimento di una risposta dal Model, modificano l'aspetto della GUI presentata.
- Gestisce la somministrazione di un questionario ad un utente, domanda dopo domanda, fino alla consegna e valutazione.

3 Tecnologie e strumenti utilizzati

3.1 HTML5

Linguaggio di markup per la progettazione di pagine Web. Richiesto espressamente nel capitolato per la creazione dell'interfaccia utente (mi pare, devo controllare).

- **Utilizzo**: viene utilizzato per creare la GUI che permette all'utente di accedere al sistema mediante browser.
- **Vantaggi**: favorisce la portabilità su diversi dispositivi (desktop, smartphone, tablet...) e browser. Sono punti a favore anche l'elevata compatibilità con tecnologie quali CSS3 e Javascript.
- **Svantaggi**: essendo HTML5 un linguaggio non ancora standard il rischio nel suo utilizzo è l'instabilità dei tag utilizzati. Un tag oggi accettato potrebbe essere modificato in un futuro prossimo, rendendo la visualizzazione dell'interfaccia utente dipendente dal stabilità dei tag utilizzati.

3.2 CSS3

Principale linguaggio usato per la formattazione di pagine HTML.

- **Utilizzo**: Viene utilizzato per formattare il codice HTML, ovvero creare fogli di stile che permettono allo sviluppatore di modificare alcuni aspetti grafici della pagina Web.
- **Vantaggi**: Richiede un minor sforzo di interpretazione da parte del browser ed è leggero da scaricare.
- **Svantaggi**: Può presentare problemi di compatibilità con browser meno recenti.

3.3 Materialize

Materialize è un framework CSS basato sull'idea di design "Material Design" di Google.

- **Utilizzo**: verrà utilizzato per la formattazione delle pagine web.
- **Vantaggi**: un tale framework aiuta l'applicazione a mantenere uno stile coerente in tutte le sue parti, e mette a disposizione una vasta scelta ci classi CSS preconfezionate, esonerando lo sviluppatore dalla necessita di definirne di proprie. Aumenta inoltra l'adattabilità (responsiveness) delle pagine HTML a diversi supporti (desktop, mobile, etc.).
- **Svantaggi**: necessità di un breve tempo di studio per impararne le classi e il corretto utilizzo.

3.4 Javascript

E' un linguaggio di scripting debolmente orientato agli oggetti, utilizzato nelle applicazioni Web. Viene interpretato all'interno del browser. Permette di definire funzionalità simili a quelle offerte da C++ e Java, quali cicli e strutture di controllo. Viene solitamente affiancato a pagine statiche HTML per poter gestire i contenuti dinamicamente, offrendo funzionalità che il linguaggio di markup non può offrire.

- **Utilizzo**: nel sistema Quizzipedia Javascript sarà ampiamente utilizzato. La quasi totalità delle componenti del Presenter sarà realizzata in Javascript, per gestire la parte logica dell'applicazione gli elementi dinamici dell'interfaccia.
- **Vantaggi**: eseguito lato Client (sul browser) non sovraccarica il Server per l'esecuzione di richieste complesse. Permette un maggior livello d'interattività dell'interfaccia utente.
- **Svantaggi**: per script sorgenti molto corposi, può risultare oneroso in termini di tempo lo scaricamento dei contenuti. Deve, inoltre, fare affidamento ad un linguaggio che possa fisicamente effettuare transazioni di dati quando lo script esegue operazioni su oggetti remoti (eg: database). E' altresì un linguaggio non tipizzato, quindi occorre porre attenzione ai tipi delle variabili che non sono dichiarati, ma variano dinamicamente.
- **Variabili e oggetti**: le variabili se sono dichiarate all'interno di una funzione sono visibili solo all'interno di essa; se sono invece esterne sono globali. Vengono dichiarate con la keyword *var* o semplicemente assegnando loro un valore. Ogni elemento in JavaScript è un tipo primitivo o un oggetto. Gli oggetti sono entità dotate di unicità (sono uguali solo a sé stessi) e identificabili con vettori associativi, che associano nomi di proprietà a valori.

3.5 Node.js

Node.js è un framework event-driven per il motore JavaScript V8, relativo all'utilizzo server-side di Javascript.

- **Utilizzo**: Tecnologia imposta dal proponente per la gestione della base di dati, e in generale della parte server, su cui poggia il sistema Quizzipedia.
- **Vantaggi**: Node.js offre prestazioni eccellenti per lo scripting server-side una velocità superiore rispetto alla concorrenza. Il modello event-driven su cui si basa si adatta inoltre molto bene agli scopi del progetto. Secondariamente, questa tecnologia cross-platform ha suscitato molto di più l'interesse del team che se ne intende avvalere rispetto a Tomcat, l'altra tecnologia proposta dal proponente.
- **Svantaggi**: Il design del linguaggio impedisce alcune ottimizzazioni delle operazioni e la gestione dei tipi non è confortevole come in altri concorrenti.

3.6 MongoDB

MongoDB è un DBMS non relazionale, orientato ai documenti. Classificato come un database di tipo NoSQL, MongoDB si allontana dalla struttura tradizionale basata su tabelle dei database relazionali in favore di documenti in stile JSON con schema dinamico rendendo l'integrazione di dati di alcuni tipi di applicazioni più facile e veloce.

- **Utilizzo**: Base di dati con lo scopo di memorizzare domande e altri dati necessari al funzionamento del sistema.
- **Vantaggi**: I dati non sono ristretti da alcun tipo di schema, il sistema è facilmente scalabile in caso di necessità, il livello di consistenza dei dati può essere definito a piacere, tecnologia open-source semplice da padroneggiare, in particolare il gruppo è già familiare con la sintassi JSON usata da MongoDB per la memorizzazione delle informazioni.
- **Svantaggi**: Minore flessibilità per quanto riguarda la formulazione delle query e elevata dimensione dei dati su server rispetto a tecnologie più tradizionali.

3.7 AngularJS

AngularJS è un framework strutturale per la costruzione di applicazioni web. Permette di estendere la sintassi HTML con componenti della propria applicazione mantenendo comunque una stretta separazione tra i dati e la loro rappresentazione rispettando il pattern MVC.

- **Utilizzo**: Verrà utilizzato come renderer dei template della user interface di cui si occupa Meteor.
- **Vantaggi**: Fornisce la possibilità di creare Single Page Application in modo pulito e mantenibile facendo utilizzo di dependency injection e rispettando il principio di separazione degli interessi. Le componenti costruite con AngularJS sono facilmente riusabili e testabili. Elevata compatibilità con diversi tipi di browser.
- **Svantaggi**: Essendo un framework scritto interamente in Javascript, le applicazioni costruite in Angular non sono molto robuste sotto il punto di vista della sicurezza inoltre, in caso l'utente disabiliti Javascript tutto il funzionamento dell'applicazione verrebbe compromesso.

3.8 Meteor

Meteor è una piattaforma full-stack Javascript per la costruzione di mobile e web apps. Semplifica la creazione di applicazioni real time offrendo un ecosistema completo atto al loro sviluppo e alla loro fruizione.

- **Utilizzo**: Questa tecnologia contribuirà alla realizzazione del cuore principale dell'applicazione Quizzipedia.
- **Vantaggi**: Framework abbastanza semplice da imparare ad utilizzare, ideale per lo sviluppo di applicazioni web real time, sia front-end che back-end vengono codificati principalmente usando solo Javascript, presenza di package che semplificano e velocizzano lo sviluppo dell'applicazione, alta scalabilità del progetto.
- **Svantaggi**: Il modo in cui alcuni componenti core della tecnologia si interfacciano potrebbe limitare la libertà degli sviluppatori. Esperienze precedenti con la tecnologia scarse o nulle dei componenti del Team 404, tempi abbastanza considerevoli dovranno essere impiegati nel conseguimento di una buona padronanza del framework.

4 Risorse necessarie

Dato che il progetto sarà sviluppato con le tecnologie di cui sopra, per lo sviluppo il gruppo avrà bisogno di risorse quali:

- un server integrato in un hosting Web che supporti un database MongoDB, scripting Node.js e il deployment di un'applicazione basata su Meteor. E' stato scelto il servizio di hosting OpenShift.
- un buon IDE per lo sviluppo web che faciliti la creazione di codice HTML, CSS e Javascript (ad es. NetBeans, Eclipse, Aptana Studio).
- un editor UML per la creazione dei diagrammi UML di questo documento e della Definizione di Prodotto. Sono stati usati Astah e StarUML.
- scaricare e installare il framework Meteor per consentire lo sviluppo e il testing dell'applicazione in locale.

5 Diagrammi dei packages

5.1 Package della componente Model

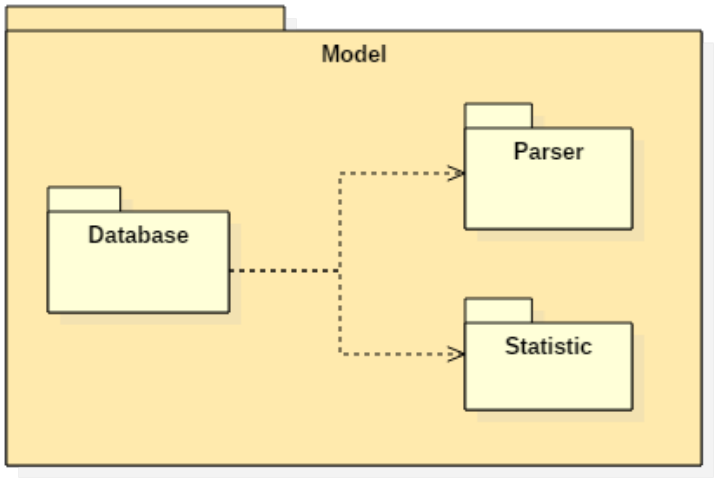


Figura 2: Package della componente Model

Il package per il componente Model del pattern architetturale MVP contiene i seguenti sotto packages:

- **Database:** questo package si occuperà di gestire le richieste in arrivo dal Presenter e avviserà quest'ultimo ogni qualvolta si verifica un cambiamento dei dati salvati nel sistema tramite appositi eventi; Per fare ciò si avvale delle classi:
  - *Database*
  - *QuestionModifiedEvent*
  - *QuestionRemovedEvent*
  - *QuizModifiedEvent*
  - *QuizRemovedEvent*
  - *UserCreatedEvent*
  - *UserModifiedEvent*
  - *UserRemovedEvent*

  e delle seguenti interfacce:
  - *ModelEvent*
- **Parser:** questo package fornisce funzionalità per il controllo sintattico rispetto a QML; Per fare ciò si avvale delle classi:
  - *Parser*
- **Statistic:** questo package fornisce classi per il raccoglimento delle statistiche relative ai quesiti, questionari ed utenti; Per fare ciò si avvale delle classi:
  - *Statistics*

5.2 Package della componente View

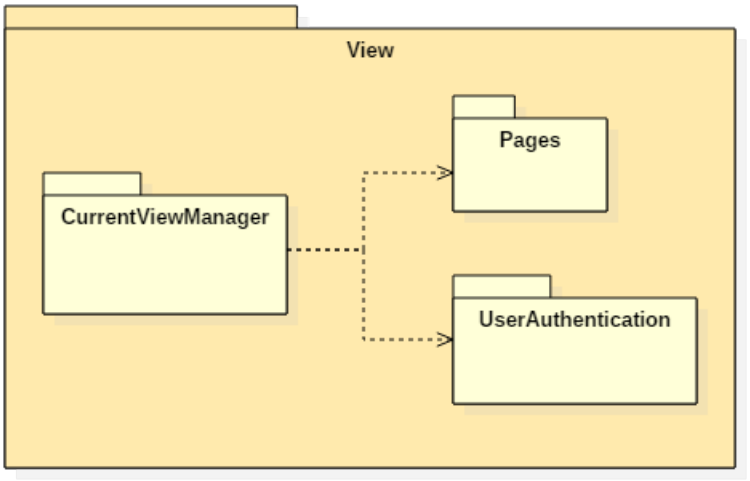
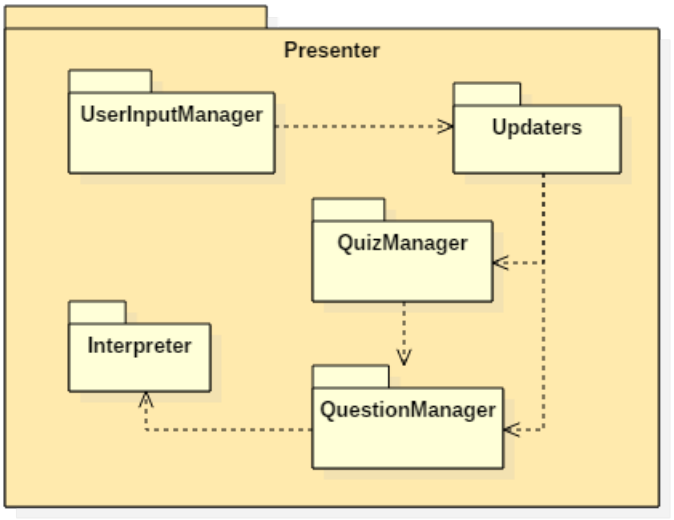


Figura 3: Package della componente View

Il package per il componente View del pattern architetturale MVP contiene i seguenti sotto packages:

- **CurrentViewManager:** questo sotto package ha lo scopo di definire lo stato attuale del sistema Quizzipedia; per fare ciò si avvale delle classi:
  - *CurrentView*
  - *Sender*
- **Pages:** contiene la classe astratta *Page*, che rappresenta una specifica situazione del sistema (ovvero una pagina web del sito), più tutte le sue derivazioni concrete:
  - *MainPage*
  - *CategoryListPage*
  - *QuizListPage*
  - *QuizExecutionPage*
  - *QuizManagementPage*
  - *ViewTutorialPage*
- **UserAuthentication:** contiene la classe *User* che raggruppa le funzionalità di autenticazione di un utente generico sulla piattaforma Quizzipedia; come *Pages* anche *UserAuthentication* è un package d'appoggio per la classe *CurrentViewManager*::*CurrentView*.

5.3 Package della componente Controller



Gli elementi del package collaborano e interagiscono con l'obiettivo comune di trattare il flusso di dati tra View e Model e mantenere aggiornato lo stato di entrambi.
Il package del Presenter contiene i seguenti sub-packages:

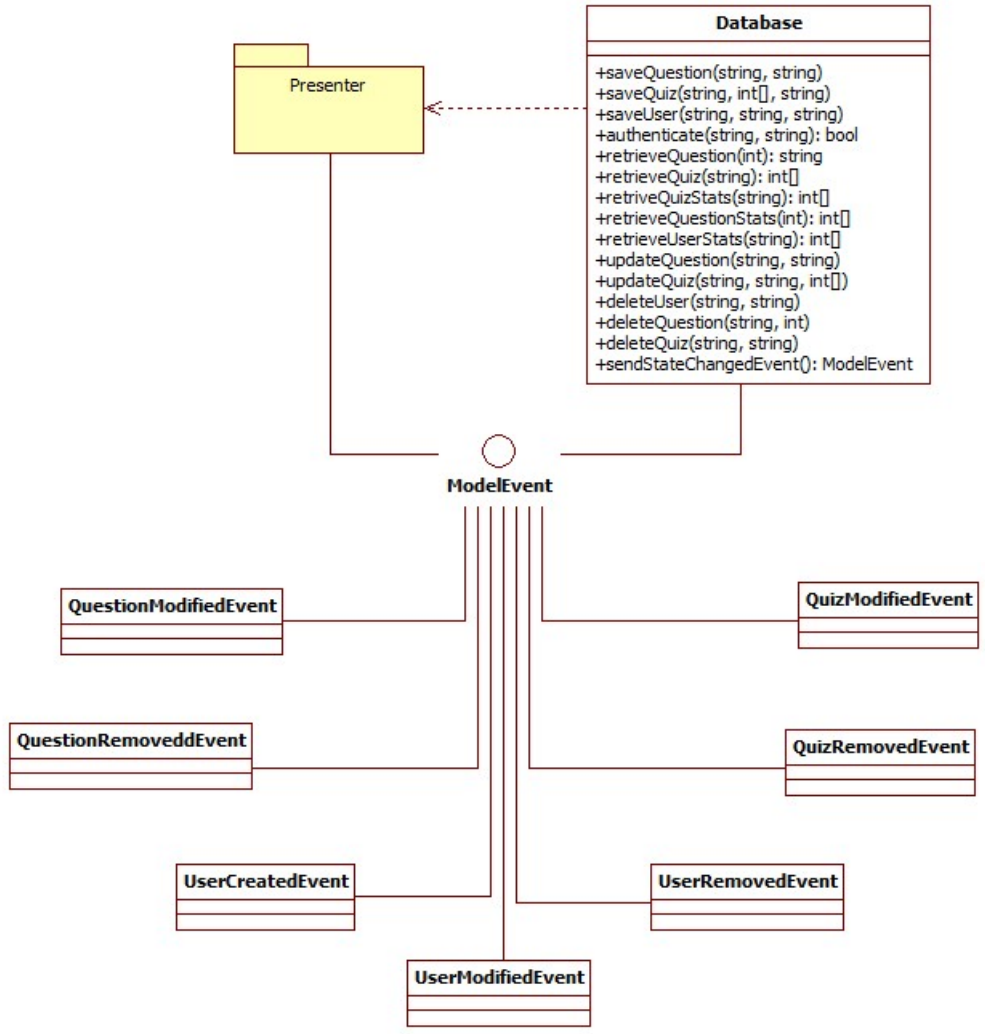
- **UserInputManager**: Il package *UserInputManager* gestisce gli input degli utenti ricevuti dalla View, realizzando la logica dell'applicazione web. Per svolgere il proprio compito si interfaccia con il package *Updaters* per aggiornare e interagire con View e Model.
  Il package contiene al suo interno le classi:
  - *InputManager*
  - *Input*
  - *CreateQuiz*
  - *AddQuestion*
  - *RemoveQuestion*
  - *SaveQuiz*
  - *CreateQuestion*
  - *SaveQuestion*
  - *DeleteQuestion*
  - *ChooseQuiz*
  - *StartQuiz*
  - *NextQuestion*
  - *PreviousQuestion*
  - *EndQuiz*
  - *ViewProfile*
  - *UpdateProfile*
  - *ViewQMLTutorial*
  - *ViewQuizList*

- **Updaters**: Il package *Updaters* ha la responsabilità di aggiornare View e Model in risposta agli input utente. per svolgere i suoi compiti collabora con i packages *QuizManger* e *QuestionManager*. Il package contiene al suo interno le classi:
  - *Updater*
  - *ModelUpdater*
  - *ViewUpdater*
- **QuestionManager**: Il package *QuestionManager* espone le funzionalità necessarie alla creazione e gestione di domande. Per permettere al package di essere estendibile in futuro con domande in diversi formati, le classi al suo interno sono organizzate seguendo il pattern *Abstract Factory*.
  Il package contiene al suo interno le classi:
  - *QuestionFactory*
  - *Question*
  - *QML2HTMLFactory*
  - *HTMLQuestion*
  - *Translator*
- **QuizManager**: Il Package *QuizManager* espone le funzionalità necessarie alla creazione e gestione di quiz, collaborando con il package *QuestionManager*. Per disaccoppiare la creazione di un quiz dalle domande che lo compongono, la classi all'interno del package seguono la struttura del design pattern *Builder*.
  Il package contiene al suo interno le classi:
  - *QuizDirector*
  - *QuizBuilder*
  - *Quiz*
- **Interpreter**: Il package *Interpreter* è responsabile della traduzione di testo QML in codice HTML5 visualizzabile da browser. Per permettere al package di essere estendibile in futuro con nuovi tipi di "Interpreter", le classi al suo interno sono organizzate seguendo il pattern *Abstract Factory* .
  Il package contiene al suo interno le seguenti classi:
  - *Interpreter*
  - *InterpreterFactory*
  - *QMLInterpreterFactory*
  - *QMLInterpreter*
  - *QML2HTMLInterpreter*

6 Diagrammi delle classi

6.1 Diagrammi delle classi del Model

6.1.1 Package Database



6.1.1.1 Classe Database

- **Funzione del componente:** la classe permettera' l'inserimento, la lettura, la modifica e la rimozione dei dati all'interno del database
- **Relazioni d'uso di altri componenti:** interagisce con il Presenter, inviando o ricevendo dati sulla base delle richieste di quest'ultimo
- **Attivita' svolte e dati trattati:** ogni metodo della classe consente l'inserimento, la lettura, la modifica e la rimozione di dati dal database, in seguito alla quale si possono verificare eventi per notificare al Presenter

6.1.1.2 Interfaccia ModelEvent

- **Funzione del componente:** l'interfaccia identifichera' dei particolari eventi che richiedono l'intervento del Presenter
- **Relazioni d'uso di altri componenti:** questa interfaccia verra implementata da classi che andranno a specificare un particolare tipo di evento

6.1.1.3 Classe QuestionModifiedEvent

- **Funzione del componente:** la classe specifica la modifica di un quesito nel database
- **Relazioni d'uso di altri componenti:** implementa l'interfaccia ModelEvent
- **Attivita' svolte e dati trattati:** segnala al Presenter il verificarsi della modifica di un quesito

6.1.1.4 Classe QuestionRemovedEvent

- **Funzione del componente:** la classe specifica la rimozione di un quesito nel database
- **Relazioni d'uso di altri componenti:** implementa l'interfaccia ModelEvent
- **Attivita' svolte e dati trattati:** segnala al Presenter il verificarsi della rimozione di un quesito

6.1.1.5 Classe QuizModifiedEvent

- **Funzione del componente:** la classe specifica la modifica di un quiz nel database
- **Relazioni d'uso di altri componenti:** implementa l'interfaccia ModelEvent
- **Attivita' svolte e dati trattati:** segnala al Presenter il verificarsi della modifica di un quiz

6.1.1.6 Classe QuizRemovedEvent

- **Funzione del componente:** la classe specifica la rimozione di un quiz nel database
- **Relazioni d'uso di altri componenti:** implementa l'interfaccia ModelEvent
- **Attivita' svolte e dati trattati:** segnala al Presenter il verificarsi della rimozione di un quiz

6.1.1.7 Classe UserCreatedEvent

- **Funzione del componente:** la classe specifica l'aggiunta di un nuovo utente nel database
- **Relazioni d'uso di altri componenti:** implementa l'interfaccia ModelEvent
- **Attivita' svolte e dati trattati:** segnala al Presenter il verificarsi dell'aggiunta di un nuovo utente

6.1.1.8 Classe UserModifiedEvent

- **Funzione del componente:** la classe specifica la modifica di un utente nel database
- **Relazioni d'uso di altri componenti:** implementa l'interfaccia ModelEvent
- **Attivita' svolte e dati trattati:** segnala al Presenter il verificarsi della modifica di un utente

6.1.1.9 Classe UserRemovedEvent

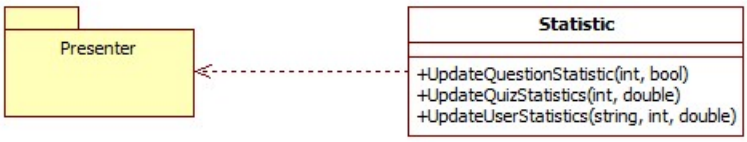
- **Funzione del componente:** la classe specifica la rimozione di un utente nel database
- **Relazioni d'uso di altri componenti:** implementa l'interfaccia ModelEvent
- **Attivita' svolte e dati trattati:** segnala al Presenter il verificarsi della rimozione di un utente

6.1.2 Package Parser

6.1.2.1 Classe Parser

- **Funzione del componente:** controlla che il testo fornito risulti corretto secondo la sintassi QML
- **Attivita' svolte e dati trattati:** il Parser controlla che il testo fornito in input rispetta la sintassi QML e fornisce in caso di errore un messaggio avvertendo l'utente di dove si trova l'errore e la tipologia

6.1.3 Package Statistiche

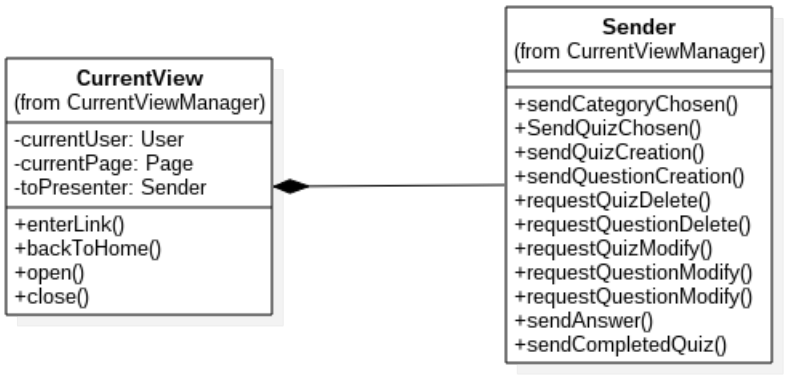


6.1.3.1 Classe Statistic

- **Funzione del componente:** questa classe fornisce funzionalita' per il raccoglimento delle statistiche all'interno di Quizzipedia
- **Attivita' svolte e dati trattati:** la classe aggiornera' le statistiche relative ai quesiti, questionari ed utenti. Per i quesiti verranno indicati il numero di volte che e' stato proposto e il numero di risposte corrette. Per i questionari verranno indicati le valutazioni medie ottenute dagli utenti e il numero di volte che e' stato proposto. Per gli utenti verranno indicati la valutazione migliore e la media dei tentativi eseguiti su singolo quiz

6.2 Diagrammi delle classi della View

6.2.1 Package CurrentViewManager



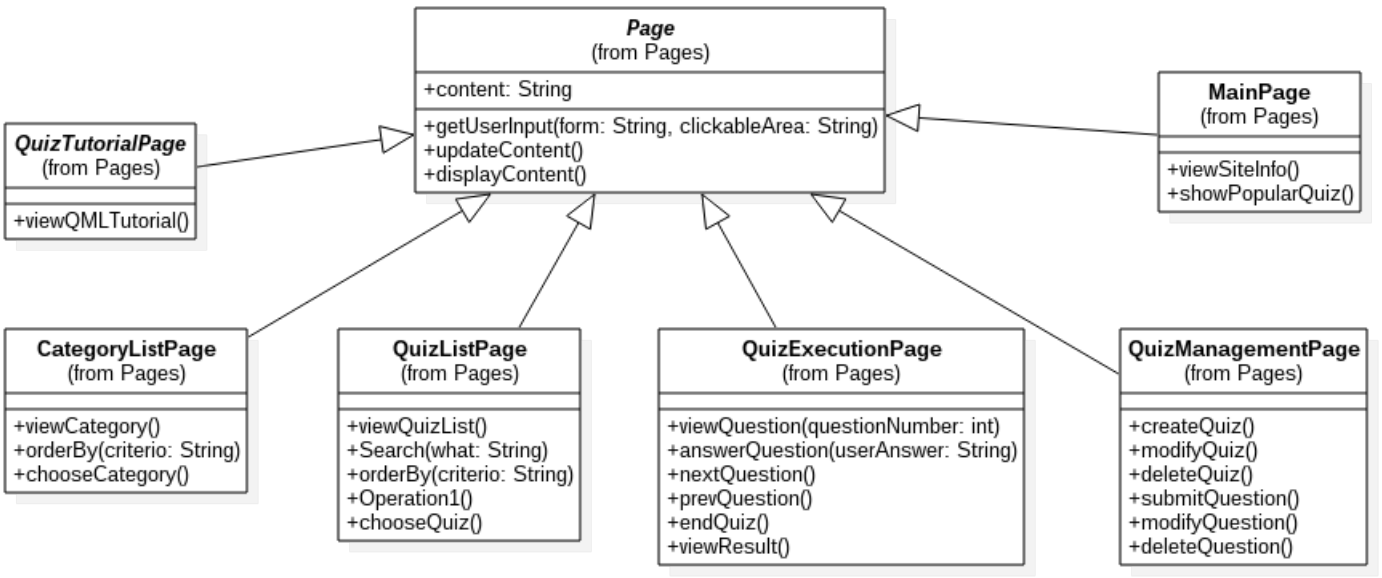
6.2.1.1 Classe CurrentView

- **Funzione del componente:** lo scopo di questa classe è quello di rappresentare la vista del sistema attualmente disponibile all'utente e di gestirne eventuali cambiamenti
- **Relazioni d'uso di altri componenti:** toPresenter, currentPage e currentUser sono parametri di tipi appartenenti ad altri package percui c'è una relazione di dipendenza verso i rispettivi packages Sender, Pages ed UserAuthentication
- **Attività svolte e dati trattati:** sono presenti dei parametri per la memorizzazione dell'utente corrente, della pagina corrente e per l'invio e la ricezione di informazioni dal Presenter. La funzionalità principale offerta è enterLink (nome da rivedere) che, come suggerisce il nome, permette all'utente di spostarsi nell'applicazione cambiando pagina. Sono presenti altre funzionalità minori che gestiscono l'avvio e la chiusura della sessione e il ritorno alla homepage.

6.2.1.2 Classe Sender

- **Funzione del componente:** inoltra le richieste dell'utente al presenter
- **Relazioni d'uso di altri componenti:** viene utilizzata all'interno di CurrentViewManager::CurrentView in risposta agli eventi che accadono sull'interfaccia web in seguito alle azioni compiute dall'utente; l'output dei metodi di questa classe viene poi processato dalle classi del package Presenter::UserInputManager.
- **Attività svolte e dati trattati:** ogni metodo di questa classe corrisponde ad una situazione che si verifica, attraverso un rispettivo metodo, in una classe derivata da Pages::Page; la classe conosce il tipo Presenter::UserInputManager::Input, e genera messaggi per ogni evento passando oggetti di tipo Input al Presenter.

6.2.2 Package Pages



6.2.2.1 Classe Page

- **Funzione del componente:** rappresenta una pagina web
- **Relazioni d'uso di altri componenti:** classe astratta che viene concretizzata dalle sue classi derivate (MainPage, CategoryListPage, QuizListPage,QuizTutorialPage, QuizExecutionPage, QuizManagementPage)
- **Attività svolte e dati trattati:** le funzionalità offerte dalla classe sono compiti fondamentali che una pagina web deve svolgere ovvero ricevere gli input dell'utente (quali click su link o dati inseriti in un form che poi andranno passati al presenter) e aggiornarsi quando nuovi dati sono disponibili

6.2.2.2 Classe MainPage

- **Funzione del componente:** mostra la pagina principale a cui l'utente arriva entrando nella piattaforma Quizzipedia
- **Relazioni d'uso di altri componenti:** concretizza la classe astratta Page da cui è diretta discendente (e viene usata da CurrentView come currentPage)
- **Attività svolte e dati trattati:** permette l'autenticazione/registrazione dell'utente nel sistema, da una panoramica del sistema generale all'utente

6.2.2.3 Classe CategoryListPage

- **Funzione del componente:** pagina che elenca gli argomenti tra i quali l'utente può scegliere e fornisce operazioni di ordinamento sulla vista
- **Relazioni d'uso di altri componenti:** concretizza la classe astratta Page da cui è diretta discendente (e viene usata da CurrentView come currentPage) +

- **Attività svolte e dati trattati:** le funzionalità di questa classe permettono la visualizzazione delle varie categorie, l'ordinamento per diversi criteri (alfabetico, categorie più visitate, ultimi questionari disponibili...) e la scelta di una tra di esse

6.2.2.4 Classe QuizListPage

- **Funzione del componente:** pagina che elenca i questionari (su uno stesso argomento) tra i quali l'utente può scegliere e fornisce operazioni di ordinamento sulla vista
- **Attività svolte e dati trattati:** le funzionalità di questa classe permettono la visualizzazione di informazioni sui vari questionari, l'ordinamento per diversi criteri (alfabetico, più visitato, novità...) e la scelta di uno tra essi
- **Attività svolte e dati trattati:** le funzionalità di questa classe permettono la visualizzazione di informazioni sui vari questionari, l'ordinamento per diversi criteri (alfabetico, più visitato, novità...) e la scelta di uno tra essi

6.2.2.5 Classe QuizExecutionPage

- **Funzione del componente:** questa classe rappresenta il punto focale del sistema Quizzipedia ovvero la parte in cui l'utente svolge i questionari scelti
- **Relazioni d'uso di altri componenti:** concretizza la classe astratta Page da cui è diretta discendente (e viene usata da CurrentView come currentPage). Il questionario visualizzato è quello scelto nella precedente pagina QuizListPage.
- **Attività svolte e dati trattati:** l'utente può navigare tra le domande del questionario nell'ordine che preferisce, dare le proprie risposte e, al termine del questionario, visualizzarne il risultato; per ognuna di queste funzionalità è presente un metodo della classe

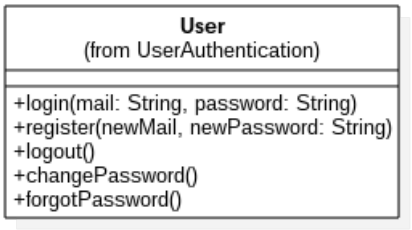
6.2.2.6 Classe QuizManagementPage

- **Funzione del componente:** gestisce creazione, modifica ed eliminazione di singole domande e interi questionari
- **Relazioni d'uso di altri componenti:** concretizza la classe astratta Page da cui è diretta discendente (e viene usata da CurrentView come currentPage)
- **Attività svolte e dati trattati:** le funzionalità offerte dalla classe consentono creazione, modifca ed eliminazione di singoli quesiti e di interi questionari

6.2.2.7 Classe QuizTutorialPage

- **Funzione del componente:** questa pagina visualizza un breve manuale che spiega l'uso e la sintassi del linguaggio QML (vedi analisi_dei_requisiti_2.0.pdf)
- **Relazioni d'uso di altri componenti:** concretizza la classe astratta Page da cui è diretta discendente (e viene usata da CurrentView come currentPage)
- **Attività svolte e dati trattati:** questa classe svolge solamente una semplice attività di visualizzazione di informazioni

6.2.3 Package UserAuthentication



6.2.3.1 Classe User

- **Funzione del componente:** classe che rappresenta il singolo utilizzatore attuale del sistema nella propria sessione
- **Relazioni d'uso di altri componenti:** User viene utilizzato dalla classe CurrentViewManager::CurrentView
- **Attività svolte e dati trattati:** questa classe prevede le funzionalità basilari per l'autenticazione dell'utente all'interno del sistema come registrazione, login e logout ed altre operazioni secondarie come il cambio o il recupero della password smarrita.

6.3 Diagrammi delle classi del Controller

6.3.1 Package UserInputManager

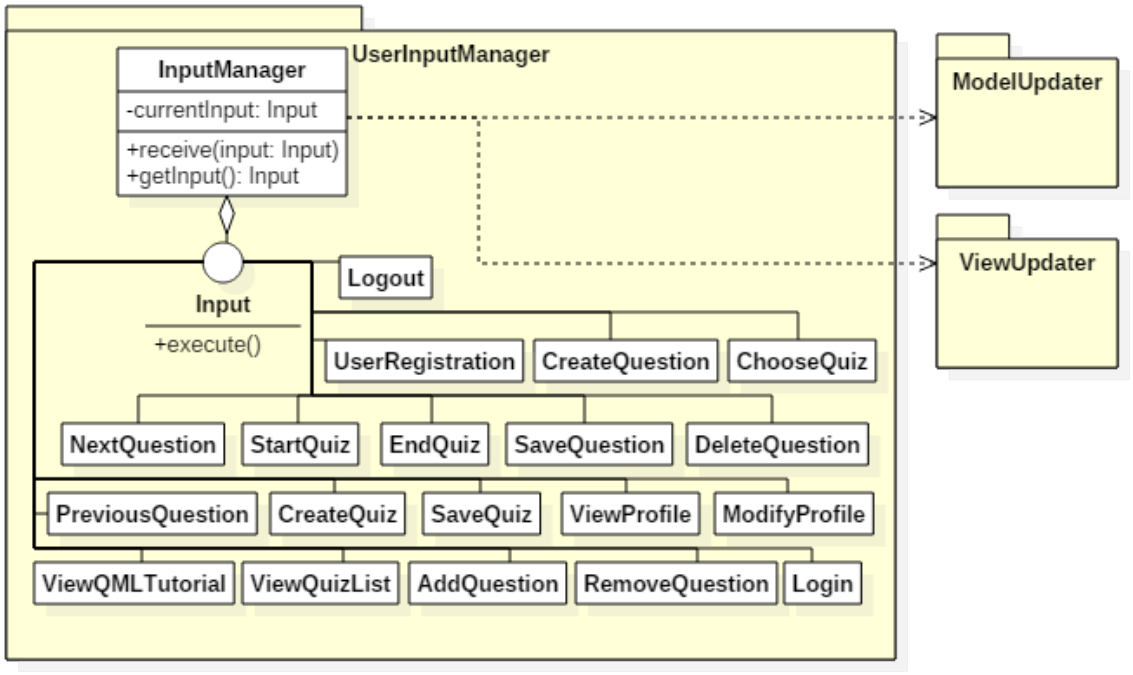


Figura 4: Diagramma delle classi del package UserInputManager

6.3.1.1 Classe InputManager

La classe implementa, con riferimento al pattern *Command*, il componente Invoker del Command. In questo modo il trattamento degli input viene disaccoppiato dalle componenti che ne eseguiranno le richieste.

- **Funzione del componente**: riceve gli input dalla View e delega alle componenti del package Updaters la loro risoluzione.
- **Relazione d'uso di altre componenti**: si interfaccia con il package Updaters (che sono i suoi *Receivers*) e con il package View dal quale riceve gli input.
- **Attività svolte e dati trattati**: questa classe possiede operazioni per ricevere e indirizzare ai giusti componenti gli input che le arrivano dalla View.

6.3.1.2 Interfaccia Input

L'interfaccia implementa l'interfaccia di esecuzione delle richieste; con riferimento al pattern *Command*, essa corrisponde alla classe Command. Quest'implementazione permette di aggiungere facilmente nuovi tipi di Input (concretizzazioni del Command).

- **Funzione del componente**: interfaccia di base degli input.

- **Relazione d'uso di altre componenti**: viene concretizzata in più sottoclassi. Oggetti delle sue sottoclassi vengono creati da View::Sender. Le sottoclassi concrete s'interfacciano con Updaters::ViewUpdater e Updaters::ModelUpdater.
- **Attività svolte e dati trattati**: definisce il contratto di ogni classe di tipo Input. Le classi concrete s'interfacciano con le classi del package Updaters in modo specializzato per ogni input.
- **Concretizzazioni**: Attualmente l'interfaccia Input viene concretizzata nelle seguenti sottoclassi (paragonabili, con riferimento al pattern *Command*, a classi di tipo *ConcreteCommand*), così definite:
  - *CreateQuiz*: richiede l'interfaccia per la creazione di un nuovo questionario.
  - *AddQuestion*: richiede di aggiungere una domanda al questionario in creazione.
  - *RemoveQuestion*: richiede di rimuovere una domanda precedentemente aggiunta nel questionario in creazione.
  - *SaveQuiz*: richiede di salvare un nuovo questionario appena creato.
  - *CreateQuestion*: richiede l'interfaccia per la creazione di una nuova domanda.
  - *SaveQuestion*: richiede di salvare nel sistema una nuova domanda.
  - *DeleteQuestion*: richiede di cancellare dal sistema una domanda precedentemente creata.
  - *ChooseQuiz*: richiede di caricare e presentare all'utente il quiz scelto.
  - *StartQuiz*: richiede di iniziare lo svolgimento di un quiz scelto.
  - *NextQuestion*: richiede la domanda successiva in un quiz.
  - *PreviousQuestion*: richiede la domanda precedente in un quiz.
  - *EndQuiz*: richiede di consegnare un quiz svolto completamente o in parte.
  - *ViewProfile*: richiede la pagina di visualizzazione del profilo utente.
  - *UpdateProfile*: richiede di modificare le informazioni nel profilo utente.
  - *ViewQMLTutorial*: richiede la visualizzazione del manuale utente QML.
  - *ViewQuizList*: richiede la visualizzazione di una lista di quiz.

6.3.2 Package Updaters

6.3.2.1 Interfaccia Updater

- **Funzione del componente**: definisce le operazioni di base che andranno implementate nelle sottoclassi.
- **Relazione d'uso di altre componenti**: viene concretizzata in ViewUpdater e ModelUpdater.
- **Attività svolte e dati trattati**: nessuno.

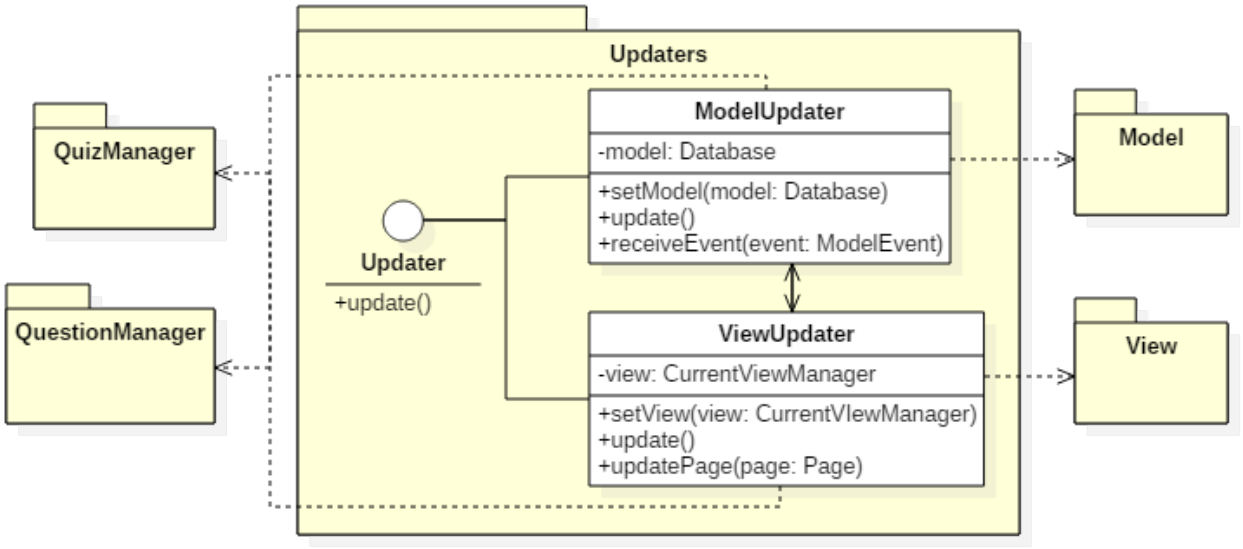


Figura 5: Diagramma delle classi del package Updaters

6.3.2.2 Classe ModelUpdater

- **Funzione del componente**: inoltra al Model i dati da salvare su database. Riceve eventi dal Database.
- **Relazione d'uso di altre componenti**: collabora con le classi ViewUpdater, QuizManager e QuestionManager e s'interfaccia con la classe Database del Model esterna al package Presenter.
- **Attività svolte e dati trattati**: possiede un riferimento al Model e ne riceve gli eventi; tali eventi possono comportare l'aggiornamento della View, in tal caso collabora con la classe ViewUpdater. E' il *Receiver* degli input che modificano il Model. Alcuni input possono richiedere la sua collaborazione con le classi QuizManager e QuestionManager.

6.3.2.3 Classe ViewUpdater

- **Funzione del componente**: è responsabile dell'aggiornamento della View rispetto agli input utente e rispetto alle modifiche del Model. E' il *Receiver* degli input che modificano la View.
- **Relazione d'uso di altre componenti**: s'interfaccia con le classi ViewUpdater, QuizManager e QuestionManager e con le classi del package View.
- **Attività svolte e dati trattati**: la componente può ricevere input che non necessitano di interazione col Model; in tal caso provvede direttamente all'aggiornamento della View. Se è richiesta interazione col Model essa interagisce con la classe ModelUpdater. Alcuni input possono richiedere la sua collaborazione con le classi QuizManager e QuestionManager.

6.3.3 Package QuestionManager

Il package fa uso del design pattern *Abstract Factory*.

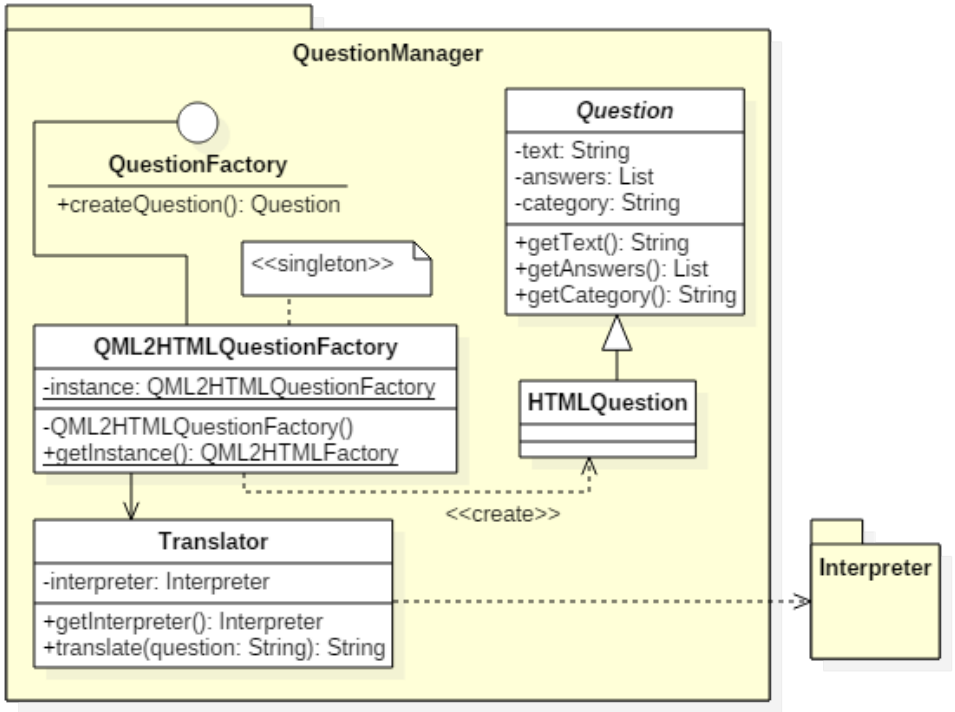


Figura 6: Diagramma delle classi del package QuestionManager

6.3.3.1 Interfaccia QuestionFactory

- **Funzione del componente**: interfaccia di base delle Factory di tipi Question.
- **Relazione d'uso di altre componenti**: può essere concretizzata in diversi tipi di QuestionFactory.
- **Attività svolte e dati trattati**: definisce il contratto delle factory Question, cioè le operazioni di costruzione di Question che saranno definite in ogni concretizzazione.

6.3.3.2 Classe astratta Question

- **Funzione del componente**: classe di base del tipo Question.
- **Relazione d'uso di altre componenti**: può essere concretizzata in diversi tipi di Question.
- **Attività svolte e dati trattati**: definisce il contratto generale delle Question, cioè le loro operazioni e dati.

6.3.3.3 Classe QML2HTMLQuestionFactory

La classe QML2HTMLQuestionFactory è un *singleton*.

- **Funzione del componente**: crea oggetti di tipo HTMLQuestion.
- **Relazione d'uso di altre componenti**: è concretizzazione della classe QuestionFactory. Crea oggetti HTMLQuestion. Si interfaccia con la classe Translator.
- **Attività svolte e dati trattati**: crea su richiesta oggetti di tipo HTMLQuestion, delegando la traduzione del codice QML alla classe Translator.

6.3.3.4 Classe HTMLQuestion

- **Funzione del componente**: rappresenta una domanda in formato HTML.
- **Relazione d'uso di altre componenti**: è concretizzazione di Question.
- **Attività svolte e dati trattati**: eredita e specializza le funzionalità dell'interfaccia Question.

6.3.3.5 Classe Translator

- **Funzione del componente**: s'interfaccia con le componenti del package Interpreter per la traduzione di quesiti QML in HTML.
- **Relazione d'uso di altre componenti**: collabora con le interfacce Interpreter e InterpreterFactory.
- **Attività svolte e dati trattati**: riceve dalla classe ViewUpdater le richieste di traduzione e il codice QML da tradurre. Attraverso la factory InterpreterFactory (una sua concretizzazione) costruisce un Interpreter concreto e lo utilizza per la traduzione del codice. L'esito della traduzione viene reso disponibile a ViewUpdater.

6.3.4 Package QuizManager

Il package fa uso del design pattern *Builder*.

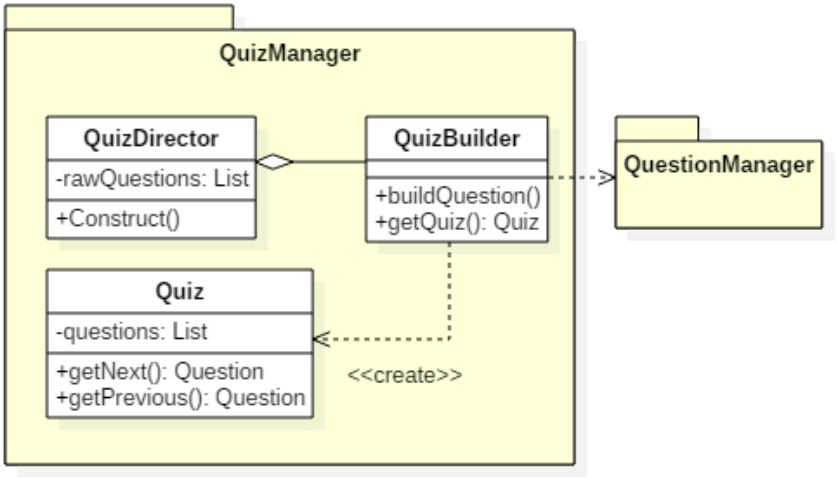


Figura 7: Diagramma delle classi del package QuizManager

6.3.4.1 Classe QuizDirector

- **Funzione del componente**: è responsabile della costruzione di un oggetto di tipo Quiz.
- **Relazione d'uso di altre componenti**: utilizza la classe QuizBuilder.
- **Attività svolte e dati trattati**: la classe, a partire da un set di domande (in questo caso in QML), utilizza il Builder per costruire un Quiz.ì

6.3.4.2 Classe QuizBuilder

- **Funzione del componente**: Costruisce domanda per domanda un questionario.
- **Relazione d'uso di altre componenti**: interagisce con il package QuestionManager.
- **Attività svolte e dati trattati**: Costruisce passo pe r passo un oggetto di tipo Quiz. Si appoggia al package QuestionManager per la creazione delle singole domande.

6.3.4.3 Classe Quiz

- **Funzione del componente**: la classe rappresenta un Quiz, una raccolta di domande.
- **Relazione d'uso di altre componenti**: nessuna.
- **Attività svolte e dati trattati**: la classe possiede i dati e fornisce le operazioni necessaria alla fruizione di un Quiz.

6.3.5 Package Interpreter

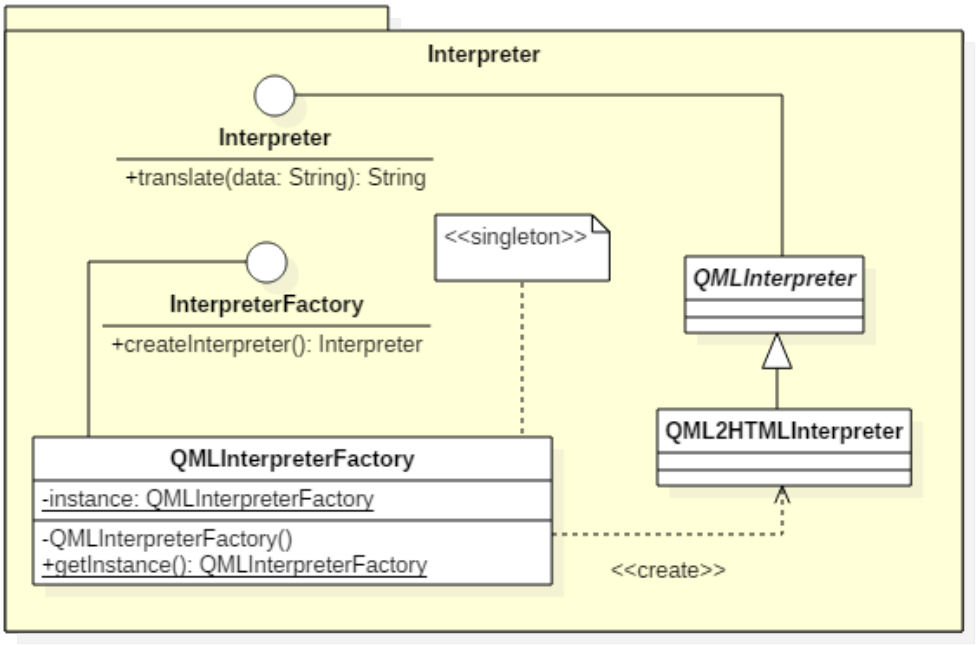


Figura 8: Diagramma delle classi del package Interpreter

6.3.5.1 Interfaccia Interpreter

- **Funzione del componente**: interfaccia di base del tipo Interpreter.
- **Relazione d'uso di altre componenti**: può essere concretizzata in diversi tipi di Interpreter. Viene riferita dalla classe Translator.
- **Attività svolte e dati trattati**: definisce il contratto degli Interpreter, cioè le operazioni di traduzione che saranno definite in ogni concretizzazione.

6.3.5.2 Interfaccia InterpreterFactory

- **Funzione del componente**: interfaccia di base delle Factory di tipi Interpreter.
- **Relazione d'uso di altre componenti**: può essere concretizzata in diversi tipi di InterpreterFactory. Viene riferita dalla classe Translator.
- **Attività svolte e dati trattati**: definisce il contratto delle factory, cioè le operazioni di costruzione di Interpreter che saranno definite in ogni concretizzazione.

6.3.5.3 Classe QMLInterpreterFactory

La classe QMLInterpreterFactory è un *singleton*.

- **Funzione del componente**: crea oggetti di tipo QMLInterpreter.

- **Relazione d'uso di altre componenti**: è concretizzazione della classe InterpreterFactory. Crea oggetti QMLInterpreter.
- **Attività svolte e dati trattati**: crea su richiesta oggetti di tipo QMLInterpreter.

6.3.5.4 Classe QMLInterpreter

- **Funzione del componente**: classe astratta che rappresenta gli Interpreter che traducono codice QML in un altro formato.
- **Relazione d'uso di altre componenti**: è sottotipo di Interpreter. Può essere concretizzata in più tipi di QMLInterpreter.
- **Attività svolte e dati trattati**: definisce il contratto dei QMLInterpreter, cioè le operazioni di traduzione da QML verso altri linguaggi.

6.3.5.5 Classe QML2HTMLInterpreter

- **Funzione del componente**: traduce codice QML in codice HTML.
- **Relazione d'uso di altre componenti**: è concretizzazione di QMLInterpreter.
- **Attività svolte e dati trattati**: riceve in input domande in QML e le traduce in codice HTML visualizzabile da browser.

7 Diagrammi di attività

7.1 Creazione Questionario

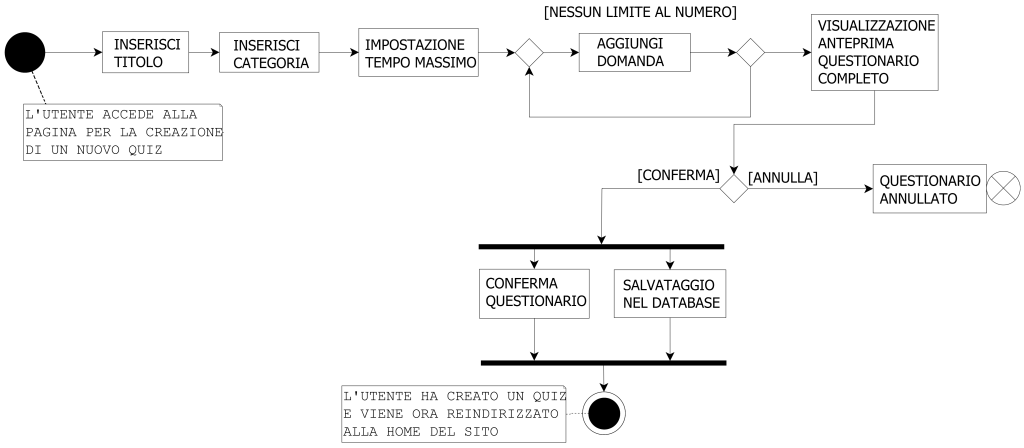


Figura 9: Diagramma di attività che descrive la creazione di un questionario

- **Precondizioni:** l'utente è autenticato nel sistema Quizzipedia e accede all'area del sito dedicata alla creazione di questionari.
- **Postcondizioni:** l'utente ha creato con successo (o ha annullato la creazione di) un questionario su un argomento a piacere contenente domande già disponibili o create precedentemente da lui stesso. L'utente viene reindirizzato alla pagina principale. Il questionario creato può ora essere svolto da altri utenti del sistema Quizzipedia.
- **Descrizione:** l'utente inizialmente assegna un titolo al questionario, sceglie l'argomento trattato tra le categorie presenti nel sistema e assegna un tempo limite entro il quale il questionario deve essere completato. A questo punto l'utente può iniziare ad inserire domande nel questionario: di volta in volta può scegliere tra le domande già disponibili sull'argomento (domande precaricate dal team404 e domande create da altri utenti del sistema) o tra le domande create da lui stesso (vedi diagramma d'attività in sezione successiva). Quando l'utente ha terminato di inserire le domande nel proprio questionario ne verrà presentata un'anteprima. L'utente ha infine la possibilità di annullare la creazione del questionario (e quindi tornare alla schermata precedente perdendo però i dati immessi in precedenza) o di confermarla con conseguente pubblicazione nel sito dove sarà disponibile agli altri utenti per la risoluzione.

7.2 Creazione Domanda

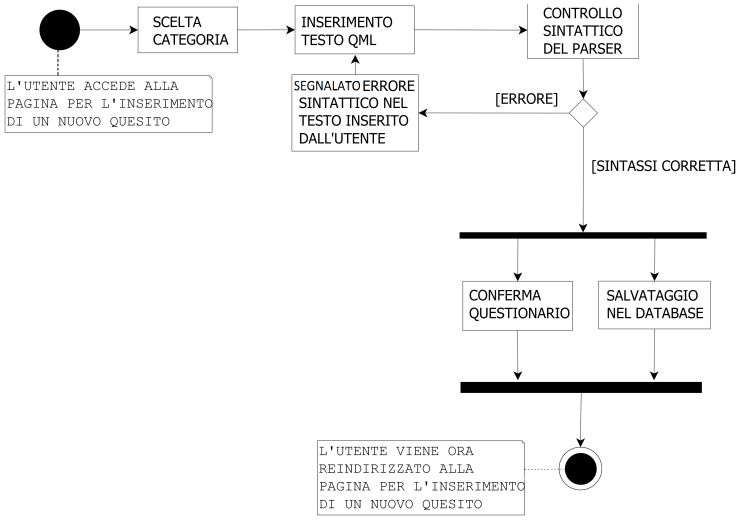


Figura 10: Diagramma di attività sulla creazione di una domanda

- **Precondizioni:** l'utente è autenticato nel sistema Quizzipedia e accede all'area del sito dedicata alla gestione (creazione, eliminazione e modifica) dei propri singoli quesiti.
- **Postcondizioni:** l'utente ha creato con successo una nuova domanda. La nuova domanda sarà disponibile a tutti gli utenti qualora essi decidano di creare un questionario appartenente alla stessa categoria. L'utente viene reindirizzato alla pagina di gestione questionari.
- **Descrizione:** l'utente inizialmente sceglie una categoria (argomento trattato) per la domanda, successivamente procede con la stesura del codice QML in un'area di testo apposita. Qualora l'utente abbia dubbi sull'uso della sintassi è presente nella pagina un link al tutorial interno al sistema. Quando l'utente ha terminato l'inserimento del codice può sottoporlo al sistema che lo valuterà. Viene eseguito il parsing del codice inserito per controllare la presenza di anomalie nella sintassi, se sono presenti errori ciò viene segnalato all'utente che ha la possibilità di correggere il codice errato altrimenti la domanda viene considerata valida e il sistema procede alla sua memorizzazione nel sistema (con conseguente notifica positiva all'utente). Al termine del procedimento la domanda sarà disponibile a tutti gli utenti registrati durante la fase di creazione di questionari (inerenti allo stesso argomento della domanda).

7.3 Compilazione Questionario

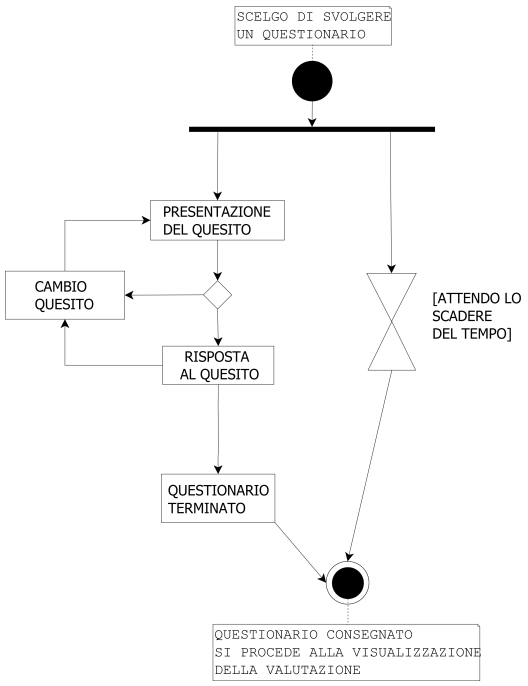


Figura 11: Diagramma di attività sulla compilazione di un questionario

- **Precondizioni:** Durante le sue ultime interazioni con il sistema Quizzipedia, l'utente ha scelto un argomento e un questionario inerente a quell'argomento che è intenzionato a svolgere. Alternativamente l'utente arriva dall'esterno seguendo un link ad un preciso questionario.
- **Postcondizioni:** L'utente ha terminato lo svolgimento del questionario (ha risposto a tutte le domande oppure è scaduto il tempo limite). L'utente viene reindirizzato ad una pagina contenente i risultati della sua prestazione cognitiva.
- **Descrizione:** Al momento dell'inizio del questionario (coincidente con il momento in cui l'utente accede alla pagina del questionario) il tempo limite inizia a scorrere, l'utente dovrà rispondere al maggior numero di domande possibili entro lo scadere del tempo (prefissato dal creatore del questionario, vedi sezioni precedenti). Viene presentato un singolo quesito per volta all'utente. L'utente può dare una risposta oppure cambiare domanda. L'utente ha la libertà di scorrere le domande del questionario e di svolgerle in qualsiasi ordine. Se l'utente ha dato una risposta a tutte le domande può consegnare il questionario. Se il tempo limite scade il questionario verrà immediatamente consegnato anche se ancora incompleto. Al momento della consegna il questionario verrà valutato dal sistema che informerà poi l'utente sull'esito della sua performance mediante un redirect ad una apposita pagina contente statistiche e correzioni.

7.4 Scelta Questionario

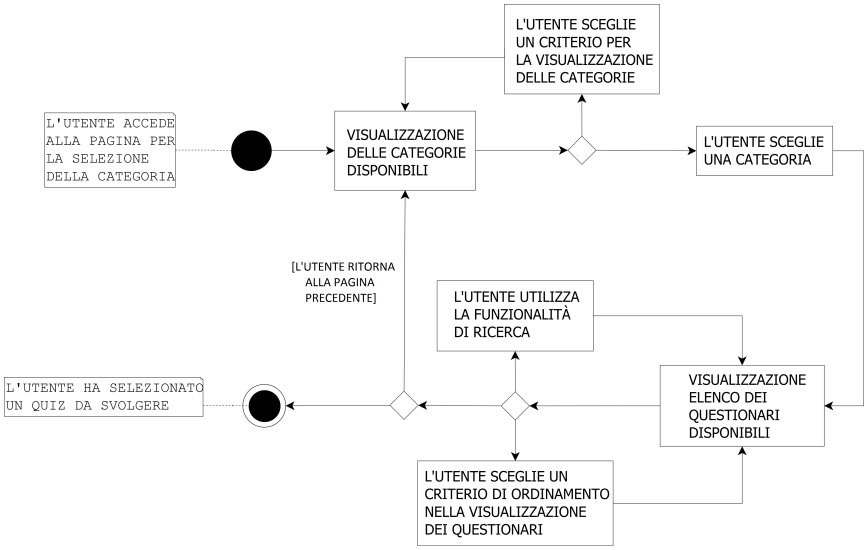


Figura 12: Diagramma di attività sulla scelta di un questionario

- **Precondizioni:** L'utente, intenzionato a svolgere un questionario, accede alla pagina adibita alla selezione dell'argomento.
- **Postcondizioni:** L'utente ha scelto un questionario da svolgere e viene reindirizzato alla pagina contenente tale questionario dove potrà cominciare a svolgerlo (vedi diagramma "Compilazione Questionario").
- **Descrizione:** Inizialmente viene presentato all'utente un elenco di tutti gli argomenti in cui sono suddivisi i questionari del sistema Quizzipedia. L'utente ha la possibilità di ordinare le categorie secondo vari criteri (ordine alfabetico e categorie più recenti) prima di passare alla scelta effettiva di una tra di esse. All'utente viene ora proposta una lista, anch'essa ordinabile secondo vari criteri (alfabetico, cronologico e autore), contenente tutti i questionari disponibili relativi all'argomento scelto in precedenza. Eventualmente l'utente può effettuare una ricerca qualora stesse cercando uno specifico questionario. L'utente può quindi scegliere definitivamente un questionario da svolgere oppure tornare indietro alla pagina di selezione categorie.

8 Diagrammi di sequenza

8.1 Salvataggio di una Domanda

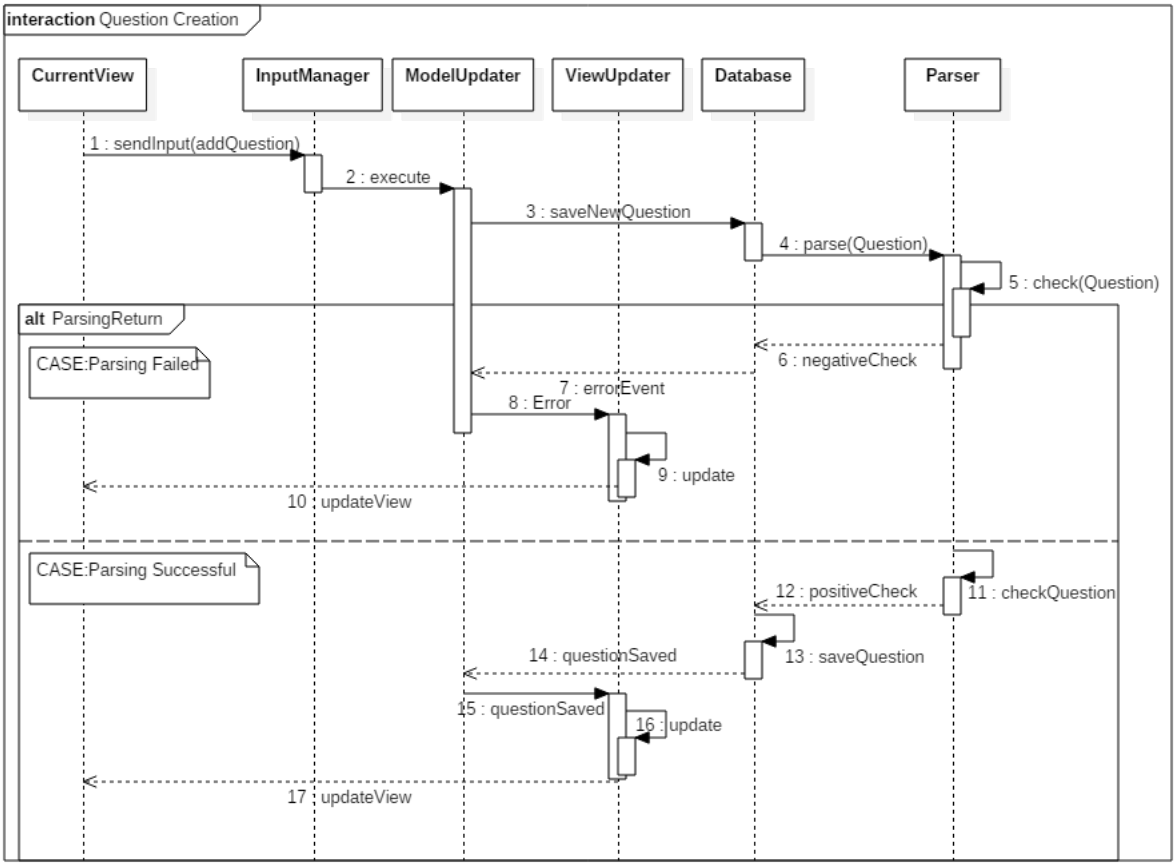


Figura 13: Diagramma di sequenza del salvataggio di una domanda

- **Precondizioni:** l'utente è autenticato nel sistema Quizzipedia e accede all'area del sito dedicata alla gestione (creazione, eliminazione e modifica) dei propri singoli quesiti.

- **Postcondizioni:** l'utente ha creato con successo una nuova domanda. La nuova domanda sarà disponibile a tutti gli utenti qualora essi decidano di creare un questionario appartenente alla stessa categoria. L'utente viene reindirizzato alla pagina di gestione questionari.

- **Descrizione:** l'utente compila il form per la creazione della nuova domanda (al cui interno ricordiamo è situato un campo adibito alla stesura del codice QML) e sottomette i dati al sistema. L'evento viene notificato al Presenter che chiama il Parser situato nel Model passandogli l'input di testo QML. All'interno del Parser viene effettuata la valutazione del codice che può avere due differenti esiti e portare a diversi scenari di conseguenza. In caso il Parser dia un esito negativo, ciò è semplicemente notificato al Presenter

e di conseguenza alla View che si aggiornerà rendendo noto all'utente il fallimento della procedura di creazione. In caso contrario, ovvero il testo QML inserito dall'utente viene considerato valido, il Model provvede a salvare la nuova domanda nel database (e anche di ciò verrà notificato l'utente con un aggiornamento nella View).

8.2 Visualizzazione e scelta di un Questionario

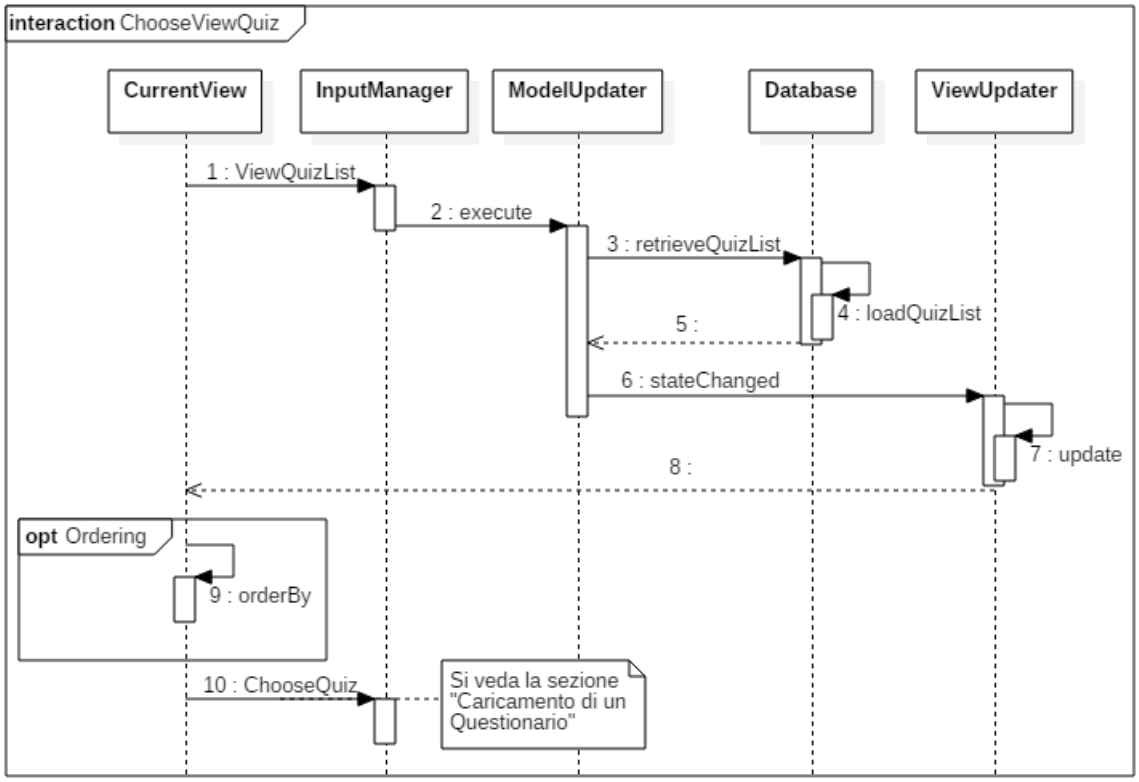


Figura 14: Diagramma di sequenza sulla visualizzazione e scelta di un questionario

- **Precondizioni:** L'utente, intenzionato a svolgere un questionario, accede alla pagina adibita alla selezione dell'argomento.

- **Postcondizioni:** L'utente ha scelto un questionario da svolgere e viene reindirizzato alla pagina contenente tale questionario.

- **Descrizione:** La classe CurrentView della View interagisce con la classe InputManager del Presenter richiedendo un elenco di questionari da visualizzare. La richiesta viene inoltrata al Model che procede a recuperare i dati richiesti e a restituirli alla View. Opzionalmente l'utente ha la possibilità di ordinare i questionari su schermo secondo vari criteri (alfabetico o cronologico). L'utente ha poi la possibilità di scegliere un questionario e il flusso di eventi si collegherà con quello espresso nel diagramma della sezione "Caricamento di un Questionario".

8.3 Caricamento di un Questionario

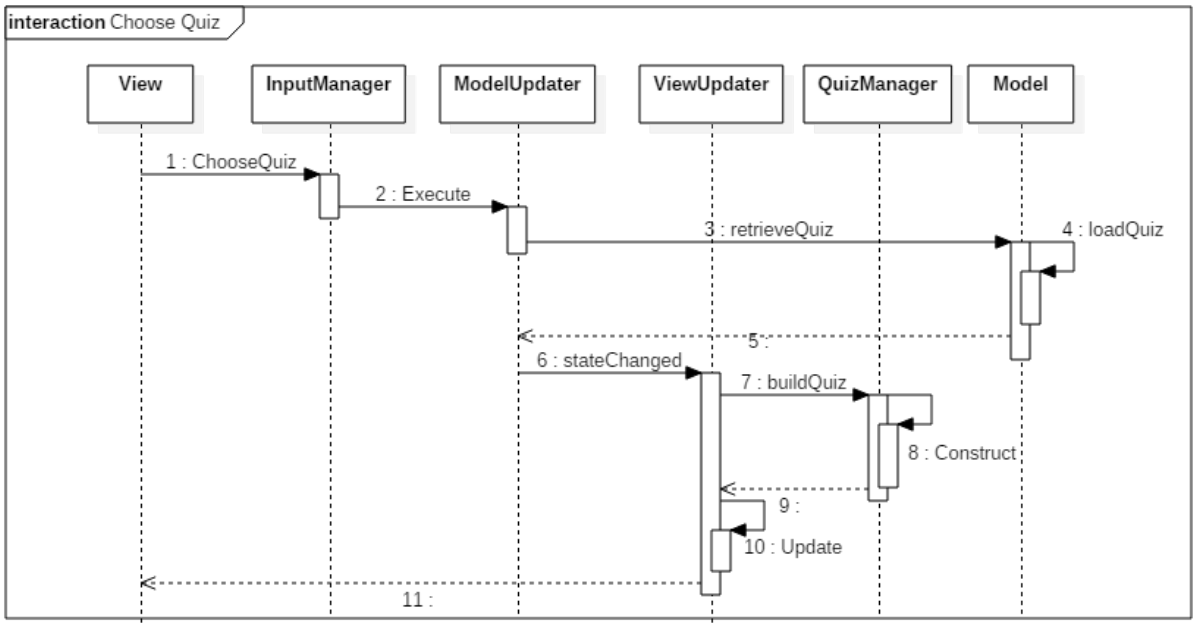


Figura 15: Diagramma di sequenza sul caricamento di un questionario

- **Precondizioni:** L'utente sta entrando in una pagina contenente un questionario che verrà caricato on demand.

- **Postcondizioni:** Il questionario è stato correttamente caricato e viene visualizzato a schermo pronto per la risoluzione.

- **Descrizione:** L'input iniziale che parte dalla View al Presenter contiene l' identificativo del questionario scelto dall'utente. Il Presenter inoltra successivamente questa informazione al Model che si occuperà di estrarre dal database i dati necessari alla costruzione del questionario (effettuata poi dal QuizManager). Una volta assemblato il questionario è pronto per essere somministrato all'utente (il cui diagramma di sequenza è riportato nella sezione "Svolgimento di un Questionario").

8.4 Traduzione di una domanda

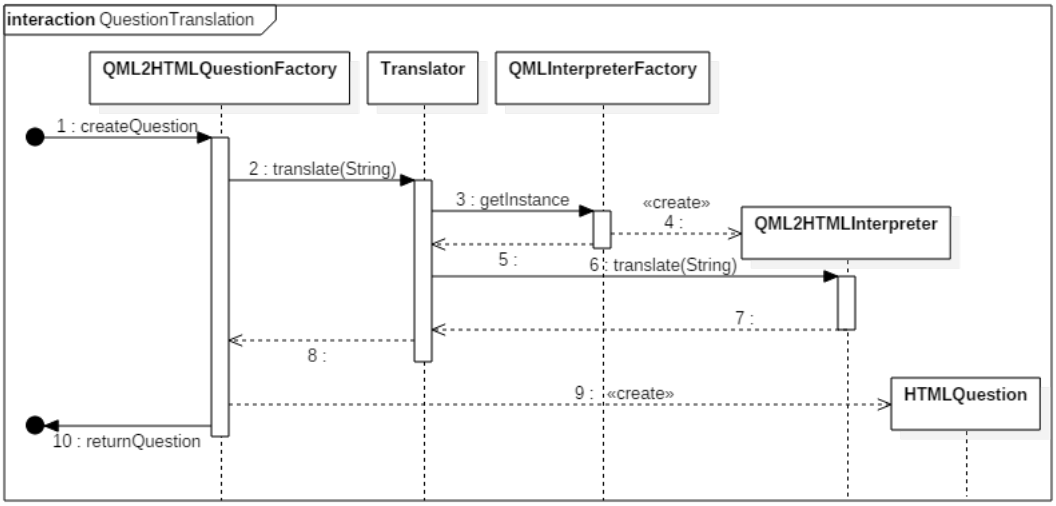


Figura 16: Diagramma di sequenza sulla traduzione di una domanda QML

- **Precondizioni:** Viene richiesta la visualizzazione su schermo di una domanda.
- **Postcondizioni:** La domanda viene convertita dal precedente formato QML al formato HTML interpretabile dal browser.
- **Descrizione:** alla richiesta di una domanda viene invocata una factory di Question, in questo caso una factory che a partire da testo QML crea domande in HTML. La factory delega la traduzione alla classe Translator; quest'ultima s'interfaccia col package Interpreter, richiedendo l'unica istanza di QMLInterpreterFactory, la quale crea un oggetto QML2HTMLInterpreter. A questo punto Translator può richiedere la traduzione del codice QML all'Interpreter ottenuto, e ritornare il risultato alla QML2HTMLFactory. Infine viene creata la domanda in formato HTML e restituita al richiedente iniziale.

8.5 Svolgimento di un Questionario

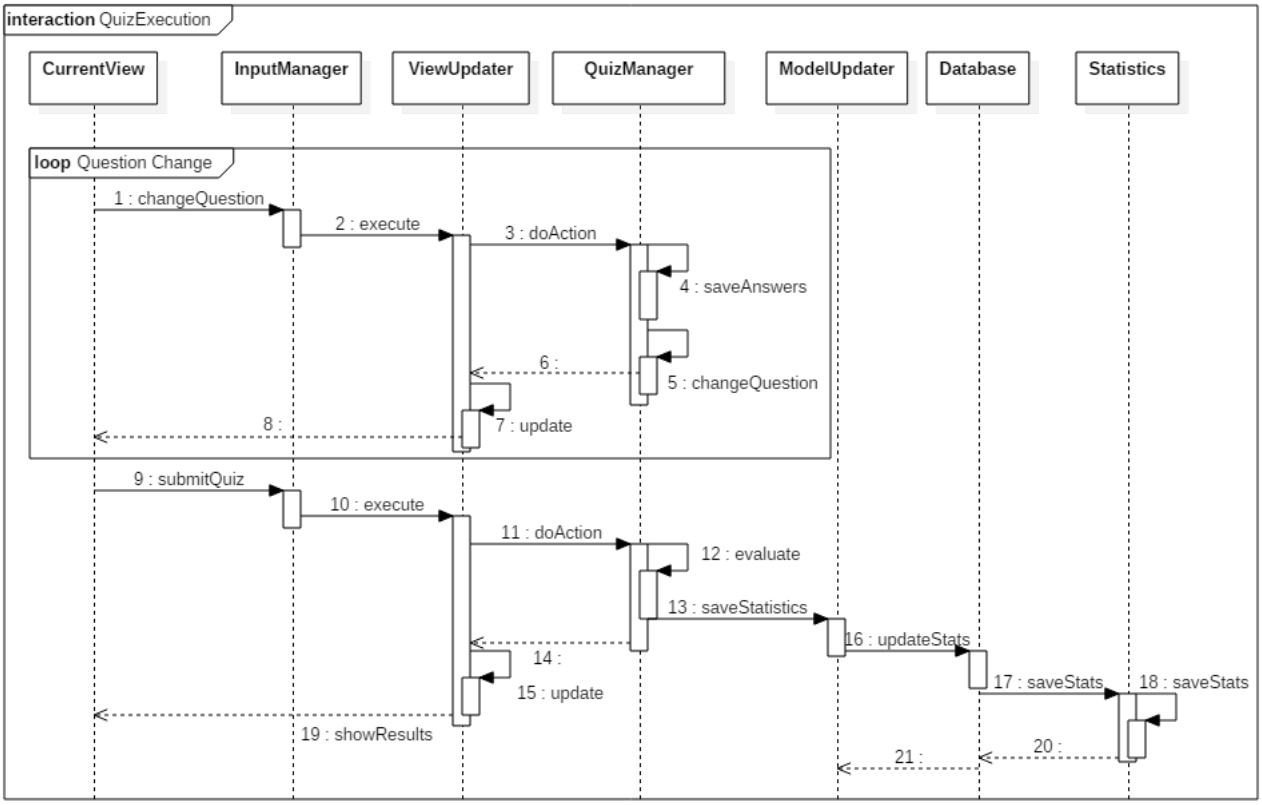


Figura 17: Diagramma di sequenza sullo svolgimento di un questionario

- **Precondizioni:** L'utente è intenzionato a svolgere un questionario ed è entrato nella pagina corrispondente a tale questionario. Inizialmente tutte le domande risultano senza risposta.

- **Postcondizioni:** L'utente ha svolto il questionario e il sistema, dopo aver compiuto la valutazione, comunica i risultati della prova.

- **Descrizione:** Come descritto nel diagramma (evento loop Question Change), l'utente ha la possibilità di scorrere più volte le domande del questionario in modo da risolverle nell'ordine in cui si sente più confortevole. Ogni volta che l'utente richiede la visualizzazione di un'altra domanda il sistema memorizza la risposta fornita dall'utente alla domanda precedente (se questa risposta era effettivamente stata data). Quando un utente ha risposto a tutte le domande può passare alla consegna del questionario, ciò porta al salvataggio delle statistiche della prestazione corrente nella base di dati e al successivo resoconto di valutazione finale che viene esposto all'utente.

9 Design pattern utilizzati

9.1 Abstract Factory

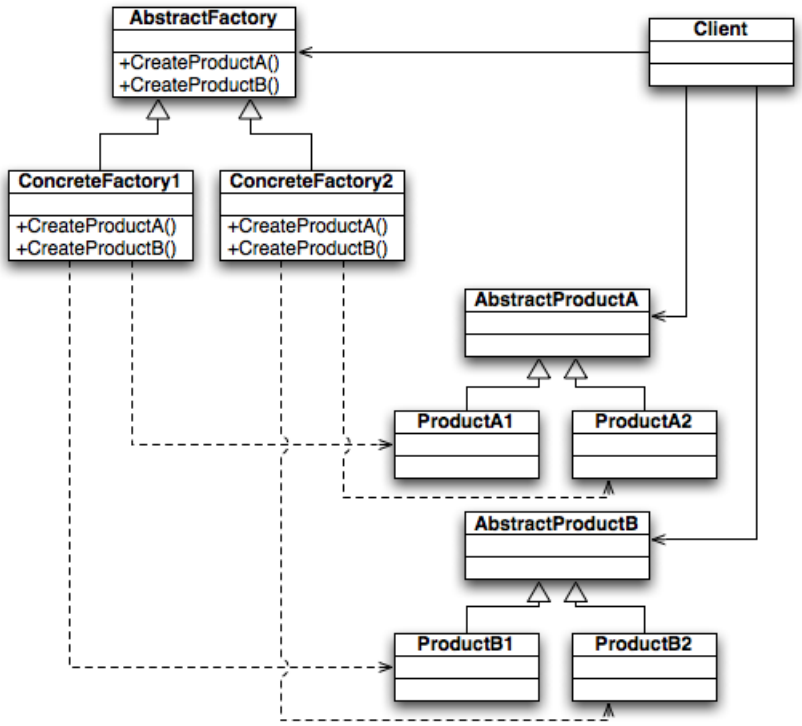


Figura 18: Design Pattern Abstract Factory

- **Scopo**: fornire al Client un'interfaccia per creare famiglie di prodotti, senza dover esplicitare il nome concreto delle classi a cui si riferisce. La factory, quindi, incapsula le responsabilità per la creazione degli oggetti prodotto.
- **Applicabilità**:
  - quando si necessita di rendere indipendente un sistema dalla creazione dei suoi componenti, dalla loro rappresentazione e composizione.
  - quando si necessita la configurazione di un sistema scegliendo tra più tipologie.
  - quando si devono gestire famiglie di prodotti correlati tra loro e progettati per essere usati insieme.
  - quando si vuole rivelare solamente l'interfaccia delle classi all'interno di una libreria.
- **Conseguenze**:
  - isola le classi concrete, controllandone la creazione. La creazione è delegata alla classe astratta e i Client manipolano solamente interfacce non conoscendo i nomi dei prodotti nascosti
  - permette di modificare facilmente la famiglia di prodotti utilizzati poiché la classe concreta appare una sola volta nel programma

- promuove la coerenza nell'utilizzo dei prodotti poiché sono stati progettati per essere utilizzati insieme (si usano oggetti di una sola famiglia per volta)
- risulta invece difficile l'inserimento di nuove tipologie di prodotto poichè richiederebbe la modifica della classe di Abstract Factory e di conseguentemente il cambiamento di tutte le sottoclassi (il problema può essere evitato mediante una tecnica di implementazione della classe astratta, ma in modo meno flessibile e poco sicuro.

- **Utilizzo**: nel sistema Quizzipedia il design pattern è stato utilizzato nei package Interpreter e QuestionManager. Entrambi si avvalgono del pattern per rendere il sistema indipendente dalla creazione delle rispettive classi concrete e rendere i package aperti all'estensione tramite la definizione di nuovi tipi "Interpreter" e "Question".

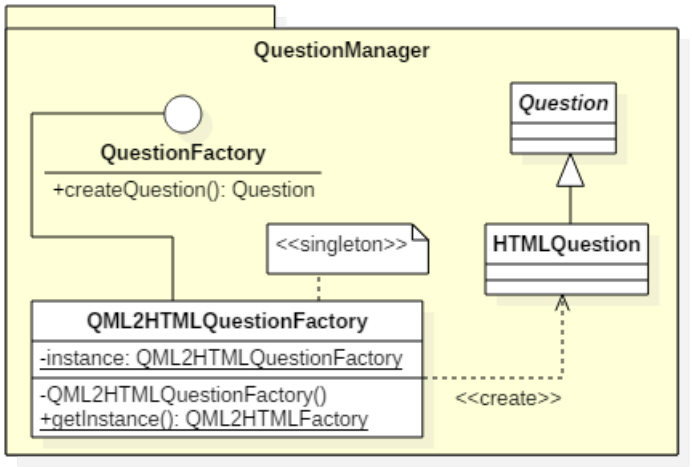


Figura 19: Abstract Factory in QuestionManager

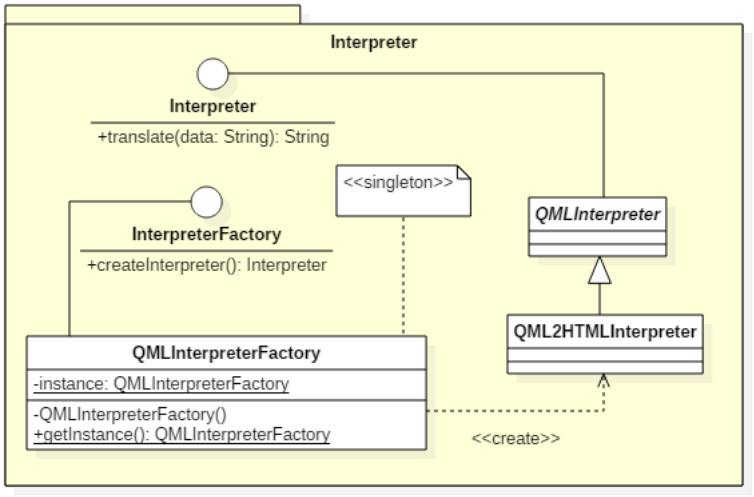


Figura 20: Abstract Factory in Interpreter

9.2 Singleton

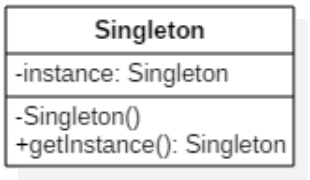


Figura 21: Design Pattern Singleton

- **Scopo**: assicurare l'esistenza di una sola istanza di un oggetto, offrendo un punto globale di accesso a tale istanza.
- **Applicabilità**:
  - quando deve essere creata una sola istanza di una classe e deve esistere un punto di accesso ad essa noto a tutti Client
  - quando l'istanza deve poter essere estesa e i Client devono essere in grado di utilizzarne le istanze senza apportare modifiche al proprio codice
- **Conseguenze**:
  - l'accesso all'unica istanza è controllato
  - permette di aver un miglior uso delle variabili globali, riducendo lo spazio dei nomi senza inquinarlo con variabili utilizzate per memorizzare riferimenti ad istanze
  - maggior flessibilità per quanto riguarda le operazioni di classe, ma maggior difficoltà nelle modifiche del progetto per utilizzare più istanze. Impossibilità di sovrascrivere le funzioni di classe poiché statiche e non virtuali.
- **Utilizzo**: nel sistema Quizzipedia il design pattern è stato utilizzato per le classi QML2HTMLQuestionFactory e QMLInterpreterFactory. Usato in collaborazione con l'*Abstract Factory*, il pattern assicura che la costruzione degli oggetti Interpreter e Question abbia un unico punto d'accesso nelle classi sopracitate.

9.3 Builder

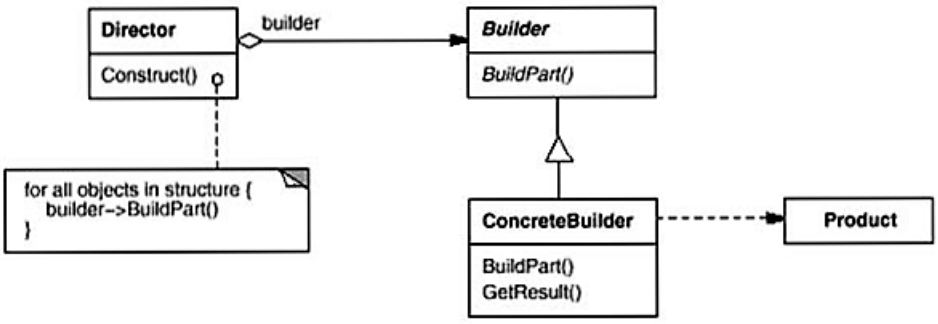


Figura 22: Design Pattern Builder

- **Scopo**: Separa la costruzione di un oggetto complesso dalla sua rappresentazione.
- **Applicabilità**:
  - La procedura di creazione di un oggetto complesso deve essere indipendente dalle parti che compongono l'oggetto
  - Il processo di costruzione deve permettere diverse rappresentazioni per l'oggetto da costruire.
- **Conseguenze**:
  - Facilita le modifiche alla rappresentazione interna di un prodotto
  - Isola il codice dedicato alla costruzione di un prodotto dalla sua rappresentazione
  - Consente un controllo migliore del processo di costruzione
- **Utilizzo**: nel sistema Quizzipedia il design pattern è stato utilizzato nel package QuizManager per rendere la creazione di un Quiz indipendente dalle domande che lo compongono.

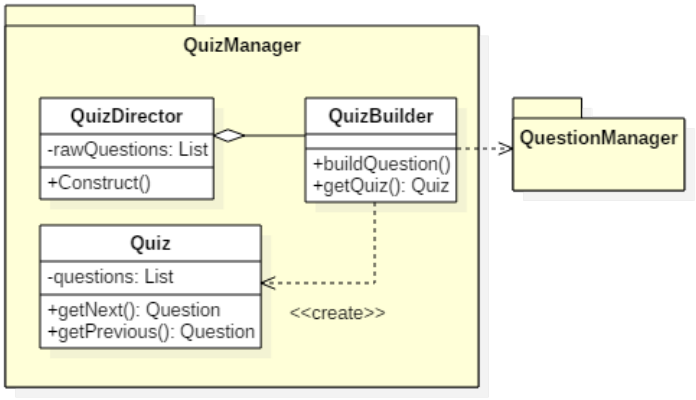


Figura 23: Builder in QuizManager

9.4 Command

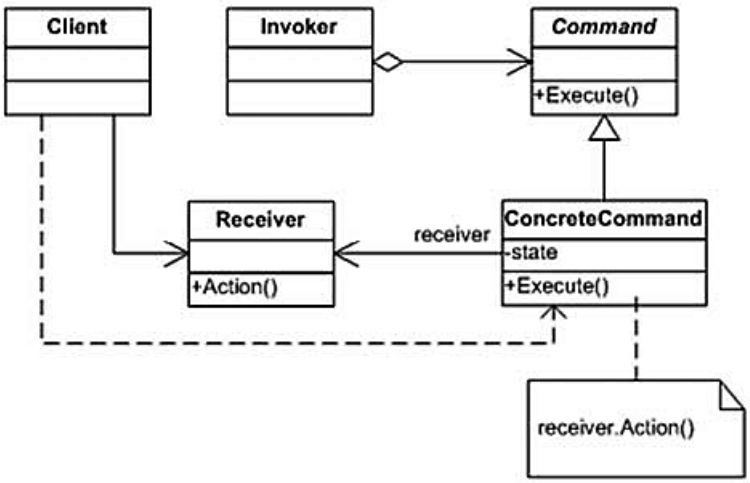


Figura 24: Design Pattern Command

- **Scopo**: Incapsulare una richiesta in un oggetto, cosicché i client sia indipendenti dalle richieste
- **Applicabilità**:
  - Necessità di gestire richieste di cui non si conoscono i particolari
  - Parametrizzazione di oggetti sull'azione da eseguire
  - Specificare, accodare ed eseguire richieste molteplici volte
  - Supporto ad operazioni di Undo e Redo e a transazioni.
- **Conseguenze**:
  - Accoppiamento "lasco" tra oggetto invocante e quello che porta a termine l'operazione
  - I command possono essere estesi
  - I command possono essere composti e innestati
  - È facile aggiungere nuovi comandi, le classi esistenti non devono essere modificate.
- **Utilizzo**: nel sistema Quizzipedia il design pattern è stato utilizzato nel package Presenter, per disaccoppiare la ricezione degli input da parte della classe InputManager dalla loro risoluzione da parte di altre classi del Presenter e del Model. Il pattern permette inoltre di aggiungere facilmente nuovi *Command* al sistema, estendendo l'interfaccia *Input* del package UserInputManager. Sarà in questo modo più facile estendere le funzionalità offerte all'utente dal sistema Quizzipedia.

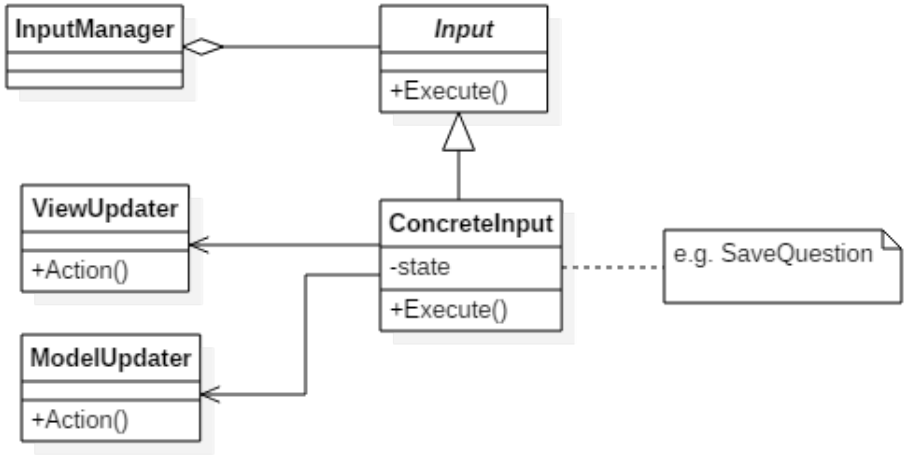


Figura 25: Design Pattern Command nel Presenter

10 Tracciamento

10.1 Mappatura componenti–requisiti

Tabella 2: Mappatura dei componenti sui requisiti

Nome componente	Codice requisito
Model::Database::Database	F 1.2.1.2 F 1.2.2 F 1.2.3 F 2.3.1.1 F 3 F 3.1.1 F 3.2 F 4 F 4.5 F 5 F 5.2.2 F 6.2.2
Model::Database::QuestionCreatedEvent	F 5.2.2.1
Model::Database::QuestionModifiedEvent	Non tracciabile
Model::Database::QuestionRemovedEvent	Non tracciabile
Model::Database::QuizCreatedEvent	F 6.2.2.1
Model::Database::QuizModifiedEvent	Non tracciabile
Model::Database::QuizRemovedEvent	Non tracciabile
Model::Database::UserCreatedEvent	Non tracciabile

Tabella 2: Mappatura dei componenti sui requisiti

Nome componente	Codice requisito
Model::Database::UserModifiedEvent	Non tracciabile
Model::Database::UserRemovedEvent	Non tracciabile
Model::Parser::Parser	F 5.2.1
Model::Statistic::Statistics	Non tracciabile
View::CurrentViewManager::CurrentView	Non tracciabile
View::CurrentViewManager::Sender	Non tracciabile
View::Pages::MainPage	F 1 F 1.1 F 1.1.1 F 1.1.2 F 1.1.2.1 F 1.2 F 1.2.1.1 F 1.2.1.2 F 2 F 2.1 F 2.1.1 F 2.1.2 F 2.2 F 2.2.1 F 2.2.2.2 F 2.3 F 2.3.1.1
View::Pages::CategoryListPage	F 3.1 F 3.1.1 F 3.1.2 F 3.1.3
View::Pages::QuizListPage	F 3 F 3.1 F 3.1.1 F 3.1.2 F 3.1.3 F 3.2 F 3.2.1 F 3.2.2 F 3.2.3
View::Pages::QuizExecutionPage	F 4.1 F 4.2.1 F 4.2.2 F 4.2.3 F 4.2.4 F 4.2.4.1 F 4.2.5

Tabella 2: Mappatura dei componenti sui requisiti

Nome componente	Codice requisito
	F 4.2.5.1 F 4.2.6 F 4.2.6.1 F 4.2 F 4.3 F 4.4 F 4.5.1 F 4.5.2.1 F 4.5.2.1 F 4.5.2.1.1 F 4.5.2.1.2 F 4.5.2.1.3 F 4.6.2 F 4.7
View::Pages::QuizManagementPage	Non traccibile
View::Pages::ViewTutorialPage	Non tracciabile
View::UserAuthentication::User	F 7
Presenter::UserInputManager::InputManager	F 1 F 1.2.1 F 1.2.1.1 F 2.3 F 2.3.1 F 3 F 3.1.1 F 3.2.1 F 4 F 4.5 F 5.2 F 7
Presenter::UserInputManager::CreateQuiz	F 6 F 6.2.1
Presenter::UserInputManager::AddQuestion	F 6.1 F 6.1.3.2
Presenter::UserInputManager::RemoveQuestion	F 6.1 F 6.1.3.2
Presenter::UserInputManager::SaveQuiz	F 6.2.2
Presenter::UserInputManager::CreateQuestion	F 5 F 5.2.1
Presenter::UserInputManager::SaveQuestion	F 5.2.2
Presenter::UserInputManager::DeleteQuestion	F 5
Presenter::UserInputManager::Choosequiz	F 3.2.1
Presenter::UserInputManager::StartQuiz	F 4

Tabella 2: Mappatura dei componenti sui requisiti

Nome componente	Codice requisito
Presenter::UserInputManager::NextQuestion	F 4.1 F 4.4
Presenter::UserInputManager::PreviousQuestion	F 4.1 F 4.4
Presenter::UserInputManager::EndQuiz	F 4.5
Presenter::UserInputManager::Login	F 4.5
Presenter::UserInputManager::Logout	F 4.5
Presenter::UserInputManager::UserRegistration	F 4.5
Presenter::UserInputManager::ViewProfile	non tracciabile
Presenter::UserInputManager::UpdateProfile	non tracciabile
Presenter::UserInputManager::ViewQMLTutorial	non tracciabile
Presenter::UserInputManager::ViewQuizList	F 3.1.1 F 3.2
Presenter::Updaters::ModelUpdater	F 1.2.2 F 5.2.2 F 6.2.2
Presenter::Updaters::ViewUpdater	F 1.1.2.1 F 1.2.1.1 F 1.2.1.2 F 2.3.1.1 F 3.1.1 F 3.2.1 F 4.5.1 F 4.5.2.1 F 5.2.1.1 F 5.2.1.2 F 5.2.2.1 F 5.2.2.2 F 6.2.1.1 F 6.2.1.2 F 6.2.2.1 F 6.2.2.2
Presenter::QuestionManager::QML2HTMLFactory	F 8 F 5
Presenter::QuestionManager::HTMLQuestion	F 5
Presenter::QuestionManager::Translator	F 8
Presenter::QuizManager::QuizDirector	F 6
Presenter::QuizManager::QuizBuilder	F 6
Presenter::QuizManager::Quiz	F 5
Presenter::Interpreter::QMLInterpreterFactory	F 8

Tabella 2: Mappatura dei componenti sui requisiti

Nome componente	Codice requisito
Presenter::Interpreter::QML2HTMLInterpreter	F 8

Tabella 2: Mappatura dei componenti sui requisiti

10.2 Mappatura requisiti – componenti

Tabella 3: Mappatura dei requisiti sui componenti

Codice Requisito	Nome Componente
F 1	View::CurrentViewManager::CurrentView View::UserAuthentication::User
F 1.1	View::CurrentViewManager::CurrentView View::UserAuthentication::User
F 1.1.1	View::CurrentViewManager::CurrentView View::UserAuthentication::User
F 1.1.2	View::CurrentViewManager::CurrentView View::UserAuthentication::User
F 1.1.2.1	View::CurrentViewManager::CurrentView View::UserAuthentication::User
F 1.2	View::CurrentViewManager::CurrentView View::UserAuthentication::User
F 1.2.1	View::CurrentViewManager::CurrentView View::UserAuthentication::User
F 1.2.1.1	View::CurrentViewManager::CurrentView View::UserAuthentication::User
F 1.2.1.2	View::CurrentViewManager::CurrentView View::UserAuthentication::User Presenter::UserInputManager::Input Presenter::UserInputManager::UserRegistration Presenter::UserInputManager::InputManager Presenter::Updaters::ModelUpdater Presenter::Updaters::ViewUpdater Model::Database
F 1.2.1.3	View::CurrentViewManager::CurrentView View::UserAuthentication::User
F 1.2.2	View::CurrentViewManager::CurrentView View::UserAuthentication::User Presenter::UserInputManager::Input Presenter::UserInputManager::UserRegistration Presenter::UserInputManager::InputManager Presenter::Updaters::ModelUpdater Presenter::Updaters::ViewUpdater Model::Database
F 1.2.3	View::CurrentViewManager::CurrentView View::UserAuthentication::User Presenter::UserInputManager::Input Presenter::UserInputManager::UserRegistration Presenter::UserInputManager::Login

Tabella 3: Mappatura dei requisiti sui componenti

Codice Requisito	Nome Componente
	Presenter::UserInputManager::InputManager Presenter::Updaters::ModelUpdater Presenter::Updaters::ViewUpdater Model::Database
F 2	View::CurrentViewManager::CurrentView View::UserAuthentication::User Presenter::UserInputManager::Input Presenter::UserInputManager::Login Presenter::UserInputManager::InputManager Presenter::Updaters::ModelUpdater Presenter::Updaters::ViewUpdater Model::Database
F 2.1	View::CurrentViewManager::CurrentView View::UserAuthentication::User
F 2.1.1	View::CurrentViewManager::CurrentView View::UserAuthentication::User
F 2.1.2	View::CurrentViewManager::CurrentView View::UserAuthentication::User
F 2.3	View::CurrentViewManager::CurrentView View::UserAuthentication::User
F 2.3.1	View::CurrentViewManager::CurrentView View::UserAuthentication::User
F 2.3.1.1	View::CurrentViewManager::CurrentView View::UserAuthentication::User Presenter::UserInputManager::Input Presenter::UserInputManager::Login Presenter::UserInputManager::InputManager Presenter::Updaters::ModelUpdater Presenter::Updaters::ViewUpdater Model::Database
F 2.3.2	View::CurrentViewManager::CurrentView View::UserAuthentication::User Presenter::UserInputManager::Input Presenter::UserInputManager::Login Presenter::UserInputManager::InputManager Presenter::Updaters::ModelUpdater Presenter::Updaters::ViewUpdater Model::Database
F 3	View::CurrentViewManager::CurrentView View::Pages::QuizListPage
F 3.1	View::CurrentViewManager::CurrentView View::Pages::QuizListPage
F 3.1.1	View::CurrentViewManager::CurrentView

Tabella 3: Mappatura dei requisiti sui componenti

Codice Requisito	Nome Componente
	View::CurrentViewManager::CurrentView View::Pages::QuizListPage Presenter::UserInputManager::Input Presenter::UserInputManager::ViewQuizList Presenter::UserInputManager::InputManager Presenter::Updaters::ViewUpdater Presenter::Updaters::ModelUpdater Model::Database
F 3.1.2	View::CurrentViewManager::CurrentView View::Pages::QuizListPage
F 3.2	View::CurrentViewManager::CurrentView View::Pages::QuizListPage
F 3.2.1	View::CurrentViewManager::CurrentView View::Pages::QuizListPage Presenter::UserInputManager::Input Presenter::UserInputManager::ChooseQuiz Presenter::UserInputManager::InputManager Presenter::Updaters::ModelUpdater Presenter::Updaters::ViewUpdater Presenter::QuizManager::QuizDirector Presenter::QuizManager::QuizBuilder Presenter::QuizManager::Quiz Presenter::QuizManager::QuizDirector Model::Database
F 3.2	View::CurrentViewManager::CurrentView View::Pages::QuizListPage
F 4	View::CurrentViewManager::CurrentView View::Pages::QuizListPage View::Pages::QuizExecutionPage
F 5	View::CurrentViewManager::CurrentView Presenter::UserInputManager::CreateQuestion Presenter::UserInputManager::SaveQuestion Presenter::Updaters::ModelUpdater Presenter::QuestionManager::QML2HTMLFactory Presenter::QuestionManager::HTMLQuestion Presenter::QuestionManager::Translator Presenter::Interpreter::QMLInterpreterFactory Presenter::Interpreter::QML2HTMLInterpreter Model::Database
F 6	View::CurrentViewManager::CurrentView Presenter::UserInputManager::InputManager Presenter::UserInputManager::CreateQuestion Presenter::UserInputManager::SaveQuestion Presenter::Updaters::ModelUpdater

Tabella 3: Mappatura dei requisiti sui componenti

Codice Requisito	Nome Componente
	Presenter::QuestionManager::QML2HTMLFactory Presenter::QuestionManager::HTMLQuestion Presenter::QuestionManager::Translator Presenter::Interpreter::QMLInterpreterFactory Presenter::Interpreter::QML2HTMLInterpreter Presenter::QuizManager::QuizDirector Presenter::QuizManager::QuizBuilder Presenter::QuizManager::Quiz Presenter::QuizManager::QuizDirector Model::Database
F 7	Presenter::UserInputManager::InputManager View::CurrentViewManager::CurrentView View::UserAthentication::User Presenter::UserInputManager::Logout Presenter::Updaters::ViewUpdater Model::Database

Tabella 3: Mappatura dei requisiti sui componenti