QUIZZIPEDIA

team404swe@gmail.com

Specifica Tecnica 3.0

Informazioni sul documento	
Nome Documento	Specifica Tecnica 3.0
Versione	3.0
Uso	Esterno
Data Creazione	22 aprile 2016
Data Ultima Modifica	23 luglio 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
2.0.2	Andrea Multinedu (Programmatore)	15/07/2016	Modifica diagrammi di attività (fig. 9 - 10)
2.0.1	Luca Alessio (Verificatore)	13/07/2016	Apportamento correzioni rilevate durante la RQ: ampliamento spiegazione svantaggi AngularJS, revisione di alcune scelte architetturali e correzione di altri errori minori.
2.0	Luca Alessio (Responsabile)	25/06/2016	Approvazione del documento.
1.1.1	Davide Bortot (Progettista)	23/06/2016	Correzione errori rilevati in seguito a verifica.
1.1	Marco Crivellaro (Progettista)	22/06/2016	Verifica del documento.
1.0.19	Davide Bortot (Progettista)	21/06/2016	Aggiunta sezione "Linguaggio QML" (§5).
1.0.18	Davide Bortot (Progettista)	21/06/2016	Caricati i diagrammi di sequenza aggiornati alla nuova architettura (§10).
1.0.17	Luca Alessio (Analista)	20/06/2016	Termine stesura e aggiornamento sezione "Tracciamento", inizio aggiornamento sezione "Diagrammi di sequenza".
1.0.16	Davide Bortot (Progettista)	19/06/2016	Aggiunta la classe Router al ViewModel (aggiunte in §6.2.2 e §8.2.5) e il rispettivo tracciamento in §21.1.
1.0.15	Luca Alessio (Analista)	18/06/2016	Inizio revisione tracciamento
1.0.14	Luca Alessio (Analista)	16/06/2016	Revisione e correzione sezioni Model e View
1.0.13	Davide Bortot (Progettista)	11/06/2016	Eliminati ed inglobati nel package "Controllers" i packages "QuizManager" e "QuestionManager", ed eliminati i relativi design pattern in §11. Riadattato il package Interpreter (§8.2.4). Aggiunto il design pattern Publish-Subscribe (§11.3).
1.0.12	Andrea Multinedu (Progettista)	09/06/2016	Definizione delle subpackage del Model "Database"(§7.1.1) e "Publishers" (§7.1.4).
1.0.11	Andrea Multinedu (Progettista)	08/06/2016	Ridefinizione del package Model (§6.1).
1.0.10	Davide Bortot (Progettista)	05/06/2016	Definizione delle classi del ViewModel "Controllers" (§8.2.1), "Subscribers" (§82.2) e "Methods" (§8.2.3).
1.0.9	Davide Bortot (Progettista)	04/06/2016	Prima definizione delle classi del ViewModel, con conseguente aggiornamento di §6.2.2 e §8.2.
1.0.8	Davide Bortot (Progettista)	03/06/2016	Prima rivisitazione ed adattamento di §5.2.2, divenuto package "ViewModel".
1.0.7	Davide Bortot (Progettista)	02/06/2016	Rivisitate le sezioni §2.2, §2.3, §2.4 in seguito alla redifinizione di §2.1.

1.0.6	Davide Bortot (Progettista)	02/06/2016	Riscritta la sezione "Architettura generale del sistema" (§2.1) con l'introduzione dei framework scelti e dell'adozione del pattern MVVM.
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_4.0.pdf*".

1.3 Glossario

Viene allegato un glossario nel file "*glossario_4.0.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_4.0.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/
- Framework Meteor:
  https://www.meteor.com/
- Framework AngularJs:
  https://www.angularjs.org/

- Framework Materialize:
  http://www.materializecss.com/
- NodeJs:
  https://www.nodejs.org/
- MongoDB:
  https://www.mongodb.com/
- Cloud hosting OpenShift:
  https://www.openshift.com
- Specifica HTML5 del W3C:
  http://www.w3schools.com/html/
- Specifica CSS3 del W3C:
  http://www.w3schools.com/css/

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 MongoDB. La base di dati raccoglie quesiti memorizzati in QML, che su richiesta verranno elaborati da un interprete, questionari (insiemi di quesiti) e i dati degli utenti registrati nel sistema. Il suddetto interprete processa l'input QML, precedentemente validato nella forma da un parser apposito, traducendolo in linguaggio HTML5 visualizzabile da browser. Il sistema verrà sviluppato utilizzando il framework full-stack Meteor, adottando AngularJS come tecnologia per il rendering e il *data-binding* dei componenti dell'interfaccia utente. Nella realizzazione di *Quizzipedia* verrà adottato il design pattern *Model View ViewModel*.

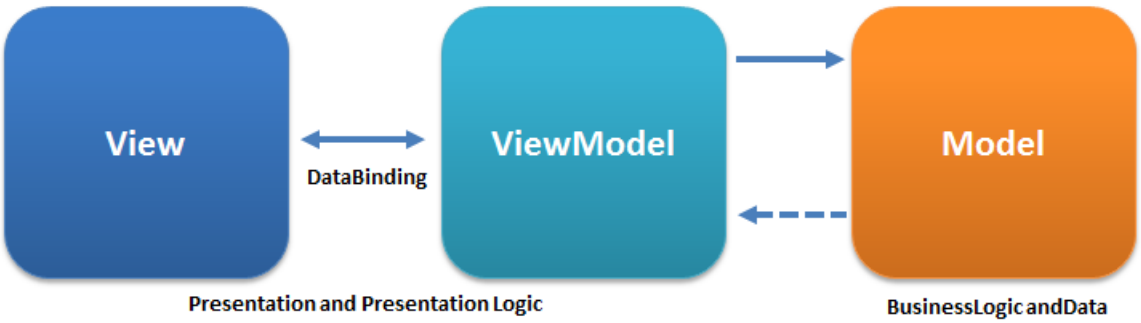


Figura 1: Il design pattern Model View ViewModel

- **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**: rappresenta l'interfaccia grafica presentata all'utilizzatore, la quale visualizza i dati e cattura le interazioni dell'utente. Nel pattern *MVVM* è puramente dichiarativa, definita attraverso linguaggi di markup.
- **ViewModel**: è la proiezione di una parte del Model per una View; realizza il *data-binding* con gli elementi della View e implementa la logica dell'applicazione. Realizza il disaccoppiamento totale dalla componente View dal Model.

Altre possibili architetture sono state prese in considerazione, ma i framework scelti, pensati per la creazione di applicazioni single-page reattive, hanno indirizzato la scelta verso il pattern *MVVM*, che si adatta particolarmente all'utilizzo di AngularJS e al sistema di comunicazione client-server implementato da Meteor (publish-subscribe).

2.2 Descrizione del componente Model

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

- Interagisce con un database MongoDB nel quale vengono salvati i dati del sistema (ad es. domande, quiz, 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 ViewModel attraverso la quale richiedere dati e operazioni su di essi, tramite funzionalità del framework Meteor come il *Publishing* e i *Methods*.

2.3 Descrizione del componente View

La componente View, situata nella parte client del sistema, rappresenta l'interfaccia grafica che visualizza i dati del ViewModel e offre le funzionalità del sistema all'utente; 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 template AngularJS per le parti dinamiche. Per assicurare uno stile coerente tra le pagine web e migliorare l'adattabilità a piattaforme mobile verrà utilizzato il framework CSS *Materialize*.

2.4 Descrizione del componente ViewModel

Il ViewModel, situato nella parte client del sistema, ricopre tre ruoli fondamentali: realizza il *data-binding* con i componenti della View, cattura ed elabora gli input degli utenti e s'interfaccia col Model per richiedere e inviare dati. Per svolgere le proprie funzioni possiede le seguenti caratteristiche:

- Conosce i riferimenti alle altre due componenti. Il ViewModel 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 e sulla View stessa tramite il meccanismo del *data-binding*.
- 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 ViewModel richiede e riceve dal Model una domanda in formato QML.
- Richiede dati al Model tramite la tecnica *publish-subscribe* di Meteor.
- 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 della ViewModel 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**: A livello di codifica, i concetti di scope inheritance e dynamic scoping possono spesso sovrapporsi producendo del codice che non si comporta sempre allo stesso modo a run time e complicandone di conseguenza la verifica. Il data binding tra diversi moduli non è ottimizzato al meglio e ciò potrebbe ripercuotersi sulle prestazioni dell'applicazione. Inoltre, 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 Linguaggio QML

Il linguaggio QMl è lo strumento scelto per definire le domande del sistema Quizzipedia. Ogni domanda avrà sempre:

- un tipo
- un teso della domanda
- un insieme di risposte, alcune giuste, alcune errate

Per rappresentare queste informazioni in modo semplice e intuitivo si è deciso di utilizzare la seguente sintassi:

- *Definizione della domanda*: `<question> *Contenuto* <fine>`
  i marcatori `<question>` e `<fine>` definiscono i confini della definizione di una domanda
- *Definizione del tipo:* `{TIPO}`
  la sintassi `{TIPO}` , inserita all inizio della definizione della domanda, ne identifica univocamente il tipo. La dicitura `"TIPO"` sarà una coppia di lettere che identifica il tipo della domanda, ad esempio:
  - **VF**: domanda Vero o Falso
  - **MU**: domanda a Risposta Multipla con unica risposta esatta
  - **MX**: domanda a Risposta Multipla con più risposte esatte
- *Definizione del testo*: `=> *Testo della domanda*`
  la sintassi => posta dopo la definizione del tipo, delimita la definizione del testo della domanda.
- *Definizione delle risposte*: cambia da domanda a domanda
  Le risposte saranno una lista di elementi del tipo

  ```
  {} Risposta1
  {} Risposta2
  {X} Risposta3
  {} Risposta4
  ```

  dove l'elemento `{X}` contiene la risposta esatta, nel caso delle domande a risposta multipla.

  Si potrà avere ad esempio una lista per domande Vero o Falso

  ```
  {V} Risposta1
  {F} Risposta2
  ```

Viene riportato un esempio completo di domanda a risposta multipla con più risposte complete, che utilizza tutti gli elementi di cui sopra:

```
 <question> {MX}
=>scegli i frutti.
{} mais
{X} mango
{} fagioli
{X} pera
<fine>
```

6 Diagrammi dei packages

6.1 Server

Nella parte *Server* del sistema lavorano gli elementi del Model.

6.1.1 Package della componente Model

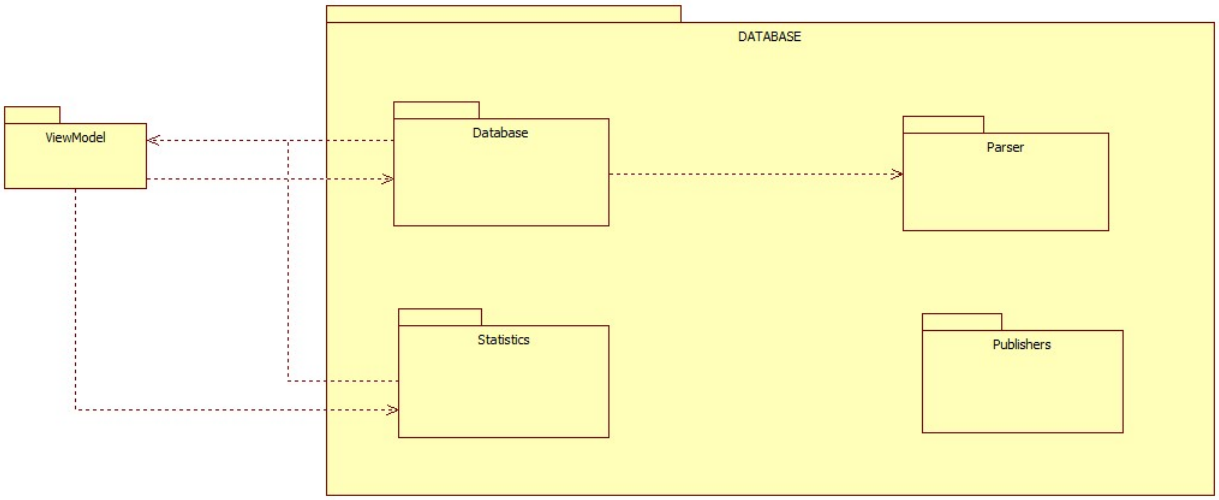


Figura 2: Package della componente Model

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 dalla ViewModel e avviserà quest'ultima ogni qualvolta si verifichi un cambiamento dei dati salvati nel sistema; per fare ciò si avvale delle classi:
  - *UserManager*
  - *QuizManager*
  - *QuestionManager*
- **Parser:** questo package fornisce funzionalità per il controllo sintattico rispetto a QML; Per fare ciò si avvale delle classi:
  - *Parser*
- **Statistics:** questo package fornisce classi per il raccoglimento delle statistiche relative ai quesiti, questionari ed utenti; Per fare ciò si avvale delle classi:
  - *Statistics*
- **Publishers:** questo package fornisce classi per la pubblicazione delle collezioni presenti sul database:
  - *QuizPublisher*
  - *QuestionPublisher*

6.2 Client

Nella parte *Client* del sistema risiedono i componenti dalla View e del ViewModel.

6.2.1 Package della componente View

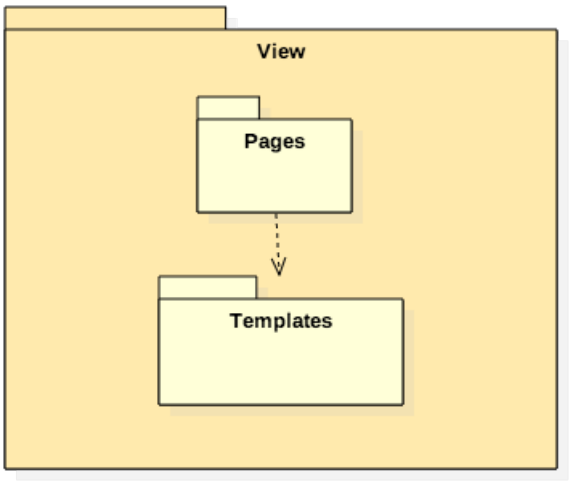
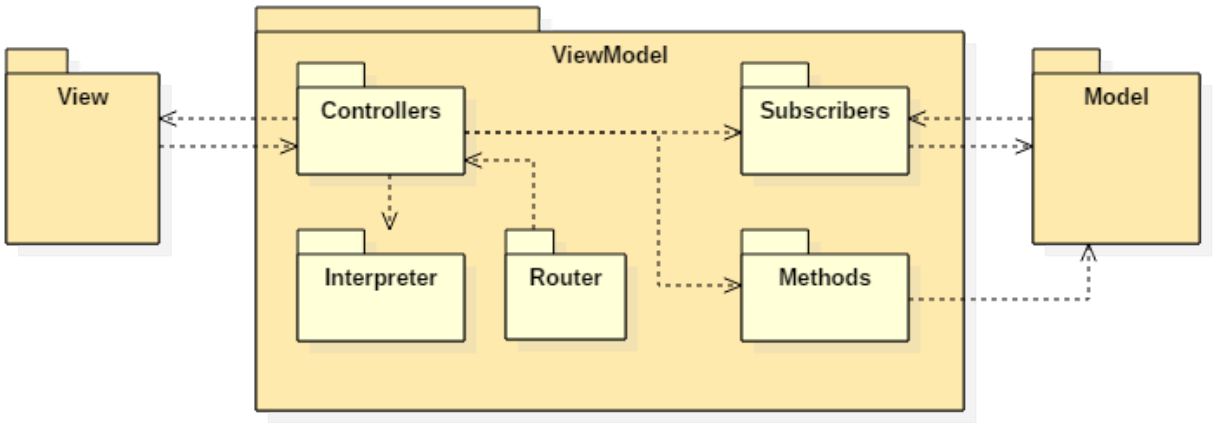


Figura 3: Package della componente View

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

- **Pages:** contiene la classe astratta *Page*, che rappresenta una generica pagina del sistema, più tutte le sue derivazioni concrete, una per ogni pagina prevista. Le pagine possono utilizzare al loro interno dei template per visualizzare situazioni standard gestite da una componente nel package Templates.
- **Templates:** contiene le classi che rappresentano le componenti utilizzabili dalle pagine per visualizzare situazioni o modelli di dati simili tra loro. Si ha così un disaccoppiamento di quali elementi una pagina contiene da come questi sono realizzati.

6.2.2 Package della componente ViewModel



Gli elementi del package collaborano e interagiscono con l'obiettivo comune di realizzare il data-binding con i componenti della View e interagire col Model per richiedere e aggiornare i dati. Il package del ViewModel contiene i seguenti sub-packages:

- **Controllers**: Il package contiene tutti i *Controller* del sistema Quizzipedia. Ogni *Controller* implementa le funzionalità associate ad un template della View. Il package contiene al suo interno le classi:
  - *NewQuestionController*
  - *NewQuizController*
  - *QuizManagementController*
  - *QuizListController*
  - *QuizDetailsController*
  - *DeleteQuestionController*
  - *DeleteQuizController*
  - *QuestionsManagementController*
  - *QMLEditorController*
- **Subscribers**: Il package contiene le classi necessarie ad effettuare il *subscribing* delle collezioni di dati pubblicate dal *server*. Il package contiene al suo interno le classi:
  - *QuestionsSubscriber*
  - *QuizSubscriber*
  - *UsersSubscriber*
- **Methods**: Il package contiene le classi necessarie all'utilizzo dei *Methods* di Meteor nel sistema, che permettono al *client* di richiedere modifiche ai dati al *server*. Il package contiene al suo interno le classi:
  - *QuestionMethods*
  - *QuizMethods*

  - *UserMethods*

- **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*

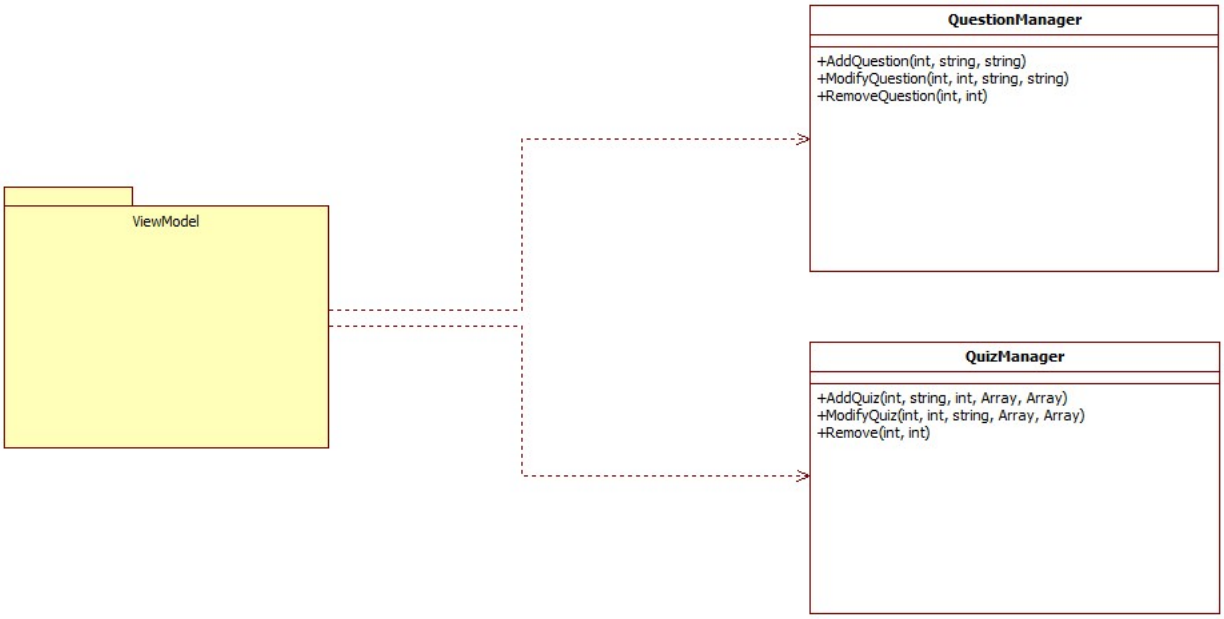
- **Router**: Il package implementa il routing e la visualizzazione dinamica dei template dei sistema, fornendo ad una single-page application le caratteristiche tipiche di un sistema web server-side. Contiene la classe:

  - *Router*

7 Diagrammi delle classi del Server

7.1 Diagrammi delle classi del Model

7.1.1 Package Database



7.1.1.1 Classe QuizManager

- **Funzione del componente:** la classe permettera' l'inserimento, la lettura e la rimozione di questionari all'interno della collezione
- **Relazioni d'uso di altri componenti:** interagisce con la ViewModel, gestendo le richieste per inserire, modificare o eliminare quiz

7.1.1.2 Classe QuestionManager

- **Funzione del componente:** la classe permettera' l'inserimento, la lettura e la rimozione di singoli quesiti all'interno della collezione
- **Relazioni d'uso di altri componenti:** interagisce con la ViewModel, gestendo le richieste per inserire, modificare o eliminare quesiti

7.1.2 Package Parser

7.1.2.1 Classe Parser

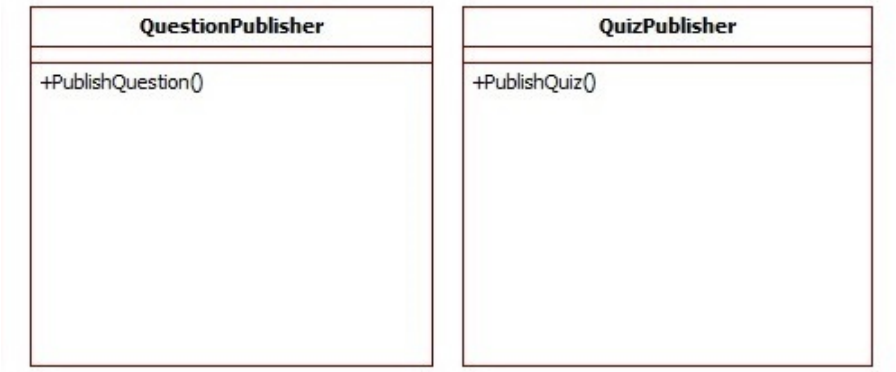
- **Funzione del componente:** controlla che il testo fornito risulti corretto secondo la sintassi QML
- **Attivita' svolte e dati trattati:** il Parser controlla che il testo fornito in input rispetta la sintassi QML e fornisce in caso di errore un messaggio avvertendo l'utente di dove si trova l'errore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

7.1.3 Package Statistics

7.1.3.1 Classe Statistics

- **Funzione del componente:** questa classe fornisce funzionalita' per il raccoglimento delle statistiche sulle prestazioni degli utenti del sistema
- **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

7.1.4 Package Publishers



7.1.4.1 Classe QuizPublisher

- **Funzione del componente:** questa classe fornisce funzionalita' per la pubblicazione dei quiz

- **Attivita' svolte e dati trattati:** questa classe permette ad un utente di accedere alla collezione dei quiz

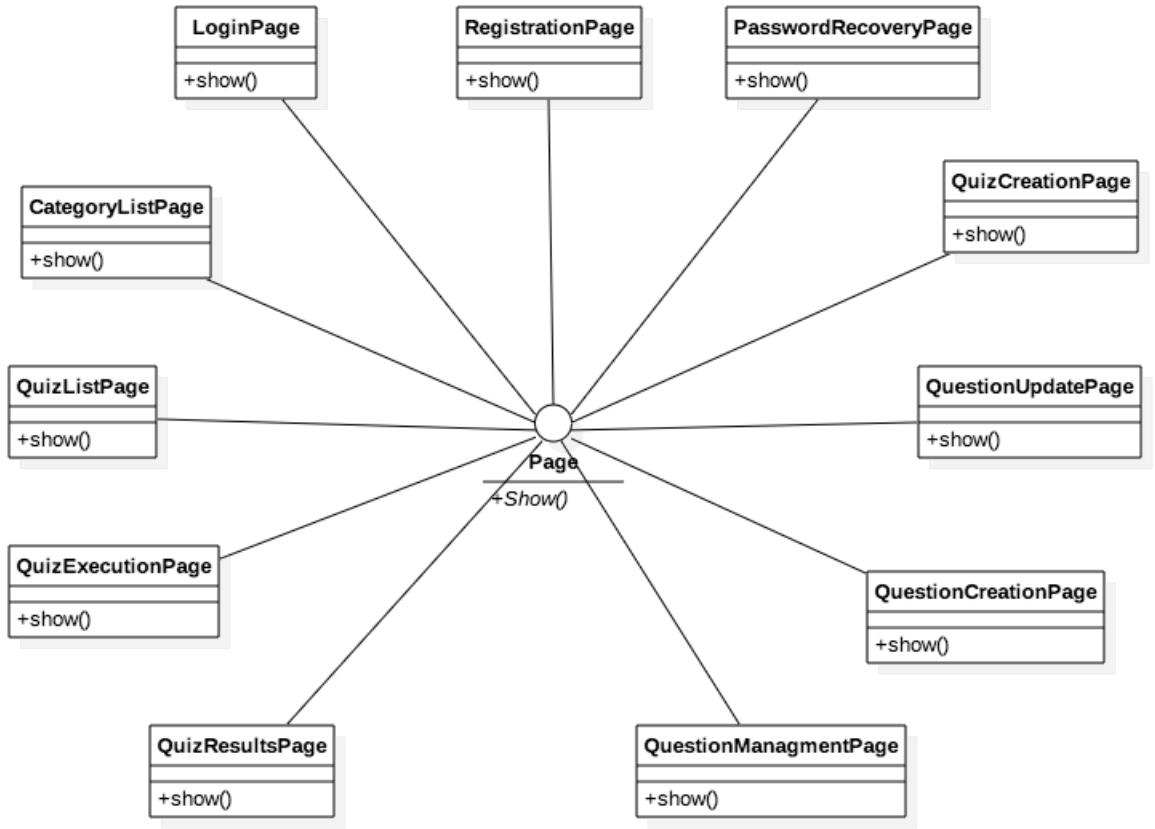
7.1.4.2 Classe QuestionPublisher

- **Funzione del componente:** questa classe fornisce funzionalita' per la pubblicazione dei quesiti
- **Attivita' svolte e dati trattati:** questa classe permette ad un utente di accedere alla collezione dei quesiti

8 Diagrammi delle classi del Client

8.1 Diagrammi delle classi della View

8.1.1 Package Pages



8.1.1.1 Interfaccia Page

- **Funzione del componente:** rappresenta una pagina web
- **Relazioni d'uso di altri componenti:** L'interfaccia Page viene concretizzata dalle sue classi derivate, una rappresentativa per ogni pagina dell'applicazione
- **Attività svolte e dati trattati:** fornisce il contratto di base che una pagina deve soddisfare (visualizzare il proprio contenuto in maniera semplice e fruibile per l'utente). Fornisce inoltre un tipo che gli altri package possono utilizzare per riferirsi ad una qualsiasi pagina indipendentemente dalla sua concretizzazione e favorendo quindi l'estendibilità del codice.

8.1.1.2 Classe LoginPage

- **Funzione del componente:** visualizza il form di autenticazione e permette il login dell'utente. Fornisce inoltre un link alla pagina di registrazione e uno alla pagina di recupero della password

- **Relazioni d'uso di altri componenti:** concretizza l'interfaccia Page da cui è diretta discendente e utilizza il template LoginForm
- **Attività svolte e dati trattati:** permette l'autenticazione dell'utente nel sistema

8.1.1.3 Classe RegistrationPage

- **Funzione del componente:** visualizza il form di registrazione. Fornisce inoltre un link alla pagina di login
- **Relazioni d'uso di altri componenti:** concretizza l'interfaccia Page da cui è diretta discendente e utilizza il template RegistrationForm
- **Attività svolte e dati trattati:** permette la registrazione dell'utente nel sistema

8.1.1.4 Classe PasswordRecoveryPage

- **Funzione del componente:** visualizza il form per il recupero della password
- **Relazioni d'uso di altri componenti:** concretizza l'interfaccia Page da cui è diretta discendente e utilizza il template PasswordRecoveryForm
- **Attività svolte e dati trattati:** permette il recupero della password dimenticata tramite l'invio di una email

8.1.1.5 Classe CategoryListPage

- **Funzione del componente:** visualizza la lista delle categorie
- **Relazioni d'uso di altri componenti:** concretizza l'interfaccia Page da cui è diretta discendente
- **Attività svolte e dati trattati:** permette la selezione di una categoria che porta ad una pagina in cui vengono visualizzati tutti i questionari della categoria scelta

8.1.1.6 Classe QuizListPage

- **Funzione del componente:** visualizza una lista di questionari
- **Relazioni d'uso di altri componenti:** concretizza l'interfaccia Page da cui è diretta discendente e utilizza il template QuizList
- **Attività svolte e dati trattati:** permette la selezione di un questionario che porta alla pagina di compilazione del questionario scelto

8.1.1.7 Classe QuizExecutionPage

- **Funzione del componente:** visualizza un questionario (una domanda alla volta) e tutti i dati relativi (tempo rimasto, numero domande, ecc...)
- **Relazioni d'uso di altri componenti:** concretizza l'interfaccia Page da cui è diretta discendente e utilizza il template QuestionCompilation
- **Attività svolte e dati trattati:** permette la compilazione di un questionario

8.1.1.8 Classe QuizResultsPage

- **Funzione del componente:** visualizza i risultati del questionario appena compilato
- **Relazioni d'uso di altri componenti:** concretizza l'interfaccia Page da cui è diretta discendente e utilizza il template QuizResults
- **Attività svolte e dati trattati:** permette la compilazione di un questionario

8.1.1.9 Classe QuestionManagmentPage

- **Funzione del componente:** visualizza la lista delle domande create dall'utente
- **Relazioni d'uso di altri componenti:** concretizza l'interfaccia Page da cui è diretta discendente e utilizza il template QuestionList
- **Attività svolte e dati trattati:** permette l'accesso alle pagine di creazione di una nuova domanda e modifica di una domanda esistente. Permette inoltre l'eliminazione di una domanda

8.1.1.10 Classe QuestionCreationPage

- **Funzione del componente:** visualizza il form di creazione di una nuova domanda
- **Relazioni d'uso di altri componenti:** concretizza l'interfaccia Page da cui è diretta discendente e utilizza il template QuestionForm
- **Attività svolte e dati trattati:** permette la creazione di una nuova domanda

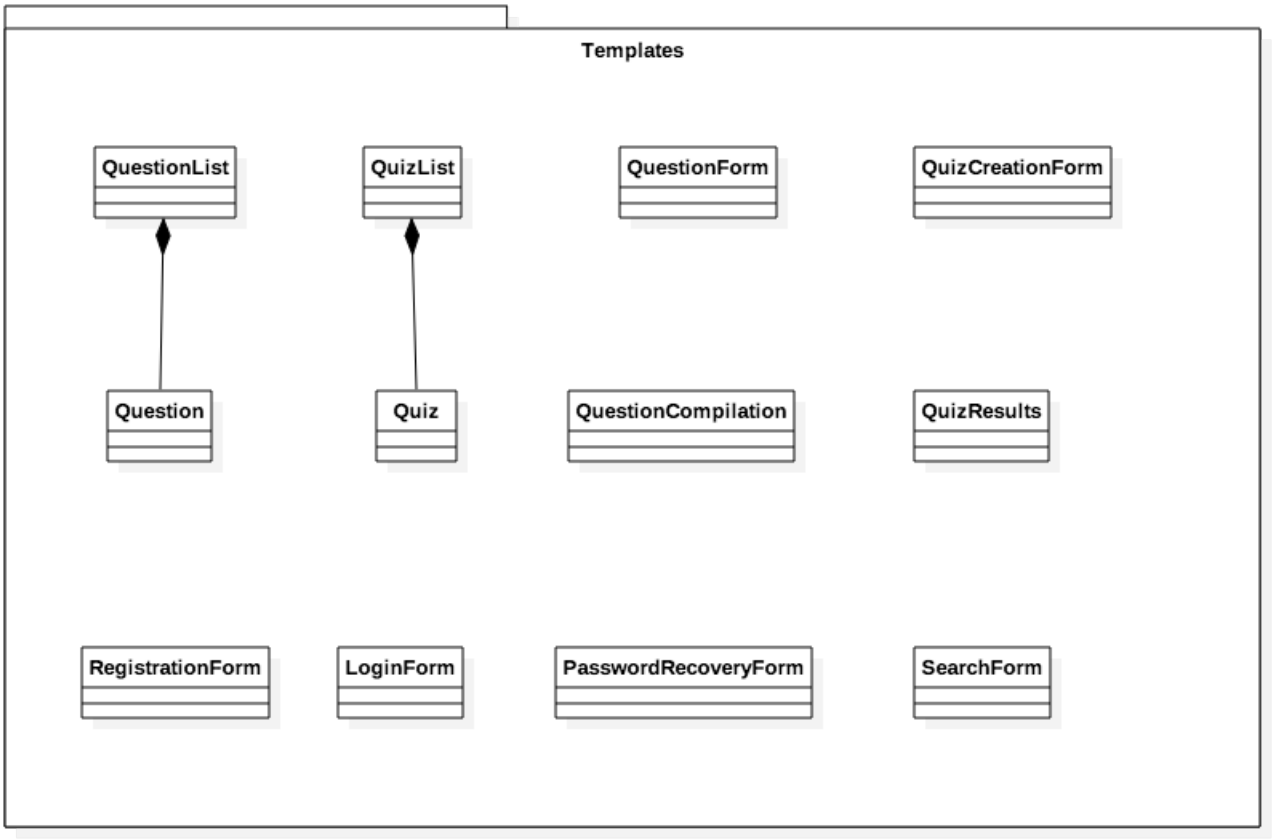
8.1.1.11 Classe QuestionUpdatePage

- **Funzione del componente:** visualizza il form di modifica di una domanda
- **Relazioni d'uso di altri componenti:** concretizza l'interfaccia Page da cui è diretta discendente e utilizza il template QuestionForm
- **Attività svolte e dati trattati:** permette la modifica di una domanda

8.1.1.12 Classe QuizCreationPage

- **Funzione del componente:** visualizza il form di creazione di un nuovo questionario
- **Relazioni d'uso di altri componenti:** concretizza l'interfaccia Page da cui è diretta discendente e utilizza il template QuizCreationForm
- **Attività svolte e dati trattati:** permette la creazione di un nuovo questionario

8.1.2 Package Templates



8.1.2.1 Classe QuestionList

- **Funzione del componente:** visualizza una lista di domande
- **Relazioni d'uso di altri componenti:** composta da Question

8.1.2.2 Classe Question

- **Funzione del componente:** visualizza una domanda inserita in una lista
- **Relazioni d'uso di altri componenti:** usata solo da QuestionList per creare la lista

8.1.2.3 Classe QuizList

- **Funzione del componente:** visualizza una lista di quiz
- **Relazioni d'uso di altri componenti:** composta da Quiz

8.1.2.4 Classe Quiz

- **Funzione del componente:** visualizza un quiz inserito in una lista
- **Relazioni d'uso di altri componenti:** usata solo da QuizList per creare la lista

8.1.2.5 Classe QuestionCompilation

- **Funzione del componente:** visualizza una domanda e ne permette la compilazione

8.1.2.6 Classe QuestionForm

- **Funzione del componente:** visualizza il form per i dati di una domanda. Utilizzabile sia per la creazione che per la modifica della domanda (se viene modificata una domanda già esistente nei campi vengono inseriti i valori attuali)

8.1.2.7 Classe QuizCreation

- **Funzione del componente:** visualizza il form di creazione di un questionario

8.1.2.8 Classe QuizResults

- **Funzione del componente:** visualizza i risultati ottenuti in seguito alla compilazione di un quiz

8.1.2.9 Classe SearchForm

- **Funzione del componente:** visualizza un form per l'inserimento dei dati desiderati e l'avvio della ricerca

8.1.2.10 Classe RegistrationForm

- **Funzione del componente:** visualizza un form per la registrazione di un nuovo utente

8.1.2.11 Classe LoginForm

- **Funzione del componente:** visualizza un form per l'autenticazione di un utente

8.1.2.12 Classe PasswordRecoveryForm

- **Funzione del componente:** visualizza un form per il recupero della password dimenticata

8.2 Diagrammi delle classi del ViewModel

8.2.1 Package Controllers

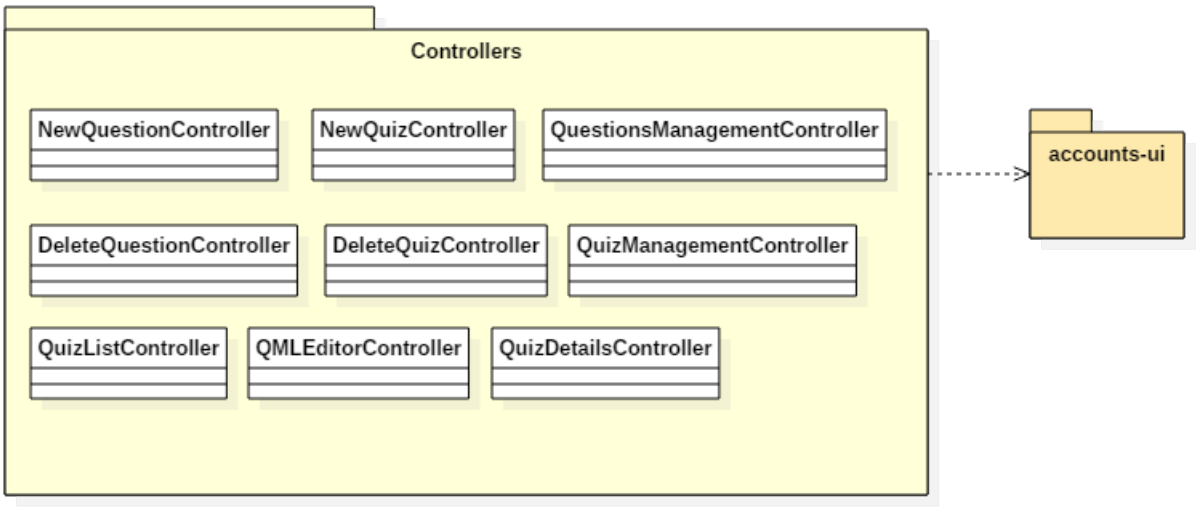


Figura 4: Diagramma delle classi del package Controllers

8.2.1.1 Classe NewQuestionController

- **Funzione del componente**: la classe permette di gestire la creazione di una nuova domanda;
- **Relazione d'uso di altre componenti**: il controller è collegato al template QuestionForm.
- **Attività svolte e dati trattati**: Fornisce le funzionalità per la creazione di una domanda, quali la scelta della categoria e l'inserimento del testo, e espone la funzione di salvataggio dei dati inseriti che compongono la nuova domanda.

8.2.1.2 Classe NewQuizController

- **Funzione del componente**: la classe permette di gestire la creazione di un nuovo quiz (questionario);
- **Relazione d'uso di altre componenti**: il controller è collegato al template QuizCreationForm;
- **Attività svolte e dati trattati**: fornisce le funzionalità per la creazione del nuovo quiz, tramite la scelta di più domande e la scelta della categoria. Espone la funzione di salvataggio del nuovo quiz.

8.2.1.3 Classe QuizListController

- **Funzione del componente**: la classe permette il caricamento e la visualizzazione della lista di quiz disponibili nel sistema;

- **Relazione d'uso di altre componenti**: il controller è collegato al template QuizList;
- **Attività svolte e dati trattati**: la classe si occupa di caricare la lista aggiornata dei quiz creati dagli utenti, e fornisce funzionalità di ordinamento della stessa e di selezione dei singoli quiz per la visualizzazione dei dettagli.

8.2.1.4 Classe QuizDetailsController

- **Funzione del componente**: la classe permette di visualizzare le informazioni generali di un quiz, una volta selezionato dalla lista dei quiz;
- **Relazione d'uso di altre componenti**: il controller è collegato al template Quiz. Si relaziona inoltre con il modulo QuizList.
- **Attività svolte e dati trattati**: la classe si occupa di recuperare e rendere visualizzabili le informazioni relative ad un singolo quiz.

8.2.1.5 Classe DeleteQuestionController

- **Funzione del componente**: la classe fornisce le funzionalità necessarie alla cancellazione di una domanda precedentemente creata;
- **Relazione d'uso di altre componenti**: il controller è collegato al template QuestionList. Si relaziona inoltre con il package Methods.
- **Attività svolte e dati trattati**: la classe si occupa di richiedere la cancellazione di una domanda, precedentemente creata dall'utente, dal sistema.

8.2.1.6 Classe DeleteQuizController

- **Funzione del componente**: la classe fornisce le funzionalità necessarie alla cancellazione di un quiz precedentemente creato;
- **Relazione d'uso di altre componenti**: il controller è collegato al template QuizList. Si relaziona inoltre con il package Methods.
- **Attività svolte e dati trattati**: la classe si occupa di richiedere la cancellazione di un quiz, precedentemente creato dall'utente, dal sistema.

8.2.1.7 Classe QuizManagementController

- **Funzione del componente**: la classe permette di gestire la somministrazione di un questionario;
- **Relazione d'uso di altre componenti**: il controller è collegato al template QuestionCompilation;
- **Attività svolte e dati trattati**: la classe si occupa di gestire la risoluzione di un questionario da parte di un utente fornendo la possibilità di navigare tra le domande, salvando le risposte dell'utente e gestendo il tempo limite. Espone la funzione di consegna del quiz compilato.

8.2.1.8 Classe QuestionsManagementController

- **Funzione del componente**: la classe permette di gestire la somministrazione di una singola domanda all'interno di un questionario;
- **Relazione d'uso di altre componenti**: il controller è collegato al template Question.
- **Attività svolte e dati trattati**: la classe si occupa di recuperare i dati associati ad una singola domanda di un questionario e di presentarli all'utente. Tali operazioni si svolgono durante la somministrazione all'utente di un questionario.

8.2.1.9 Classe QMLEditorController

- **Funzione del componente**: fornisce le funzionalità per creare e modificare una domanda tramite editor QML;
- **Relazione d'uso di altre componenti**: il controller è collegato al template QuizCreationForm.
- **Attività svolte e dati trattati**: la classe definisce la logica dello strumento principale di creazione delle domande, l'editor QML, gestisce il testo che viene inserito e espone la funzionalità di checking sintattico del testo inserito.

8.2.2 Package Subscribers

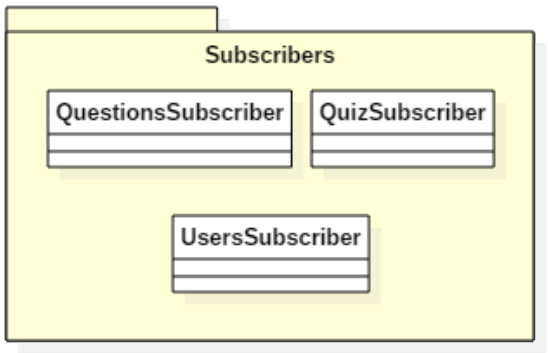


Figura 5: Diagramma delle classi del package Subscribers

8.2.2.1 Classe QuestionsSubscriber

- **Funzione del componente**: la classe è necessaria ad effettuare il *subscribe* relativo alla collezione di domande del sistema;
- **Relazione d'uso di altre componenti**: si relazione con la componente QuestionsManagementController;
- **Attività svolte e dati trattati**: la classe espone le funzionalità necessarie ad effettuare il subscribing di una componente del client alla collezione di domande del sistema, se questa è stata precedentemente pubblicata dal server.

8.2.2.2 Classe QuizSubscriber

- **Funzione del componente**: la classe è necessaria ad effettuare il *subscribe* relativo alla collezione di quiz del sistema;
- **Relazione d'uso di altre componenti**: si relazione con le componenti QuizListController e QuizManagementController;
- **Attività svolte e dati trattati**: la classe espone le funzionalità necessarie ad effettuare il subscribing di una componente del client alla collezione di quiz del sistema, se questa è stata precedentemente pubblicata dal server.

8.2.2.3 Classe UsersSubscriber

- **Funzione del componente**: la classe è necessaria ad effettuare il *subscribe* relativo alla collezione di utenti del sistema;
- **Relazione d'uso di altre componenti**: si relaziona con il package esterno 'accounts-ui';
- **Attività svolte e dati trattati**: la classe espone le funzionalità necessarie ad effettuare il subscribing di una componente del client alla collezione di utenti del sistema, se questa è stata precedentemente pubblicata dal server.

8.2.3 Package Methods

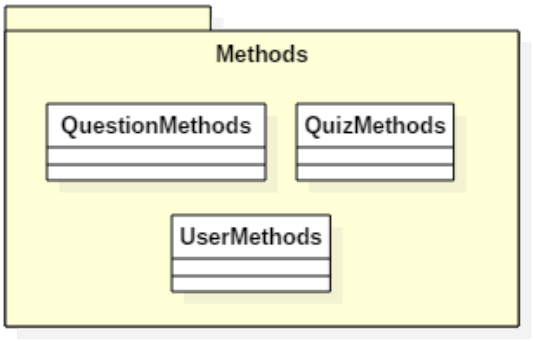


Figura 6: Diagramma delle classi del package Methods

8.2.3.1 Classe QuestionMethods

- **Funzione del componente**: permette al client di richiedere la modifica della collezione di domande del server;
- **Relazione d'uso di altre componenti**: si relaziona con le classi NewQuestionController e DeleteQuestionController;
- **Attività svolte e dati trattati**: espone le funzionalità di salvataggio e cancellazione di domande dal sistema.

8.2.3.2 Classe QuizMethods

- **Funzione del componente**: permette al client di richiedere la modifica della collezione di quiz del server;
- **Relazione d'uso di altre componenti**: si relaziona con le classi NewQuizController e DeleteQuizController;
- **Attività svolte e dati trattati**: espone le funzionalità di salvataggio e cancellazione di quiz dal sistema.

8.2.3.3 Classe UserMethods

- **Funzione del componente**: permette al client di richiedere la modifica della collezione di utenti del server;
- **Relazione d'uso di altre componenti**: si relaziona con il package esterno 'accounts-ui';
- **Attività svolte e dati trattati**: espone le funzionalità di salvataggio e cancellazione di utenti dal sistema.

8.2.4 Package Interpreter

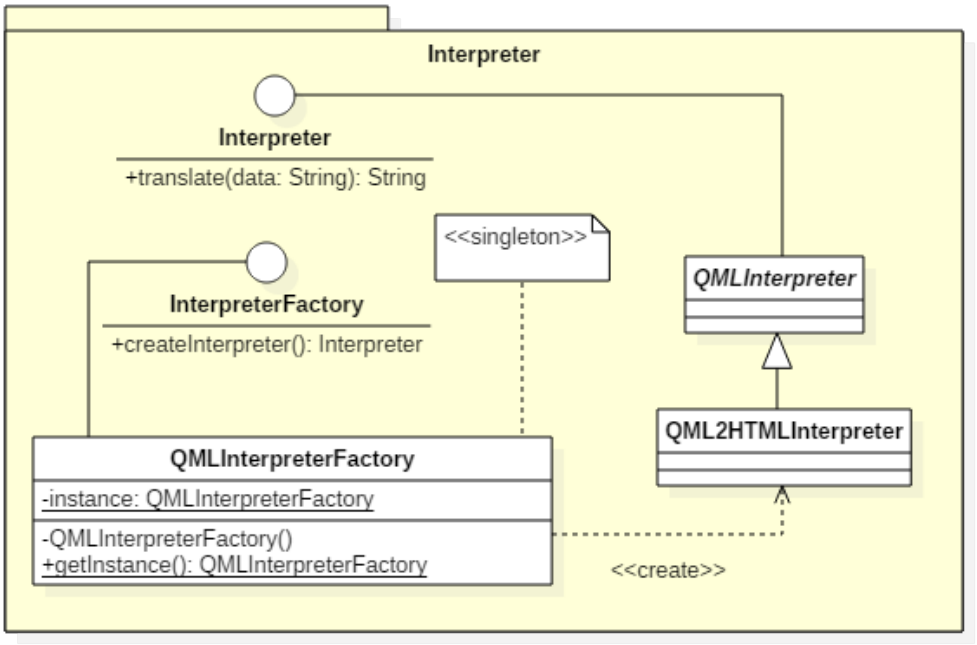


Figura 7: Diagramma delle classi del package Interpreter

8.2.4.1 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.
- **Attività svolte e dati trattati**: definisce il contratto degli Interpreter, cioè le operazioni di traduzione che saranno definite in ogni concretizzazione.

8.2.4.2 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.
- **Attività svolte e dati trattati**: definisce il contratto delle factory, cioè le operazioni di costruzione di Interpreter che saranno definite in ogni concretizzazione.

8.2.4.3 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.

8.2.4.4 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.

8.2.4.5 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.

8.2.5 Package Router

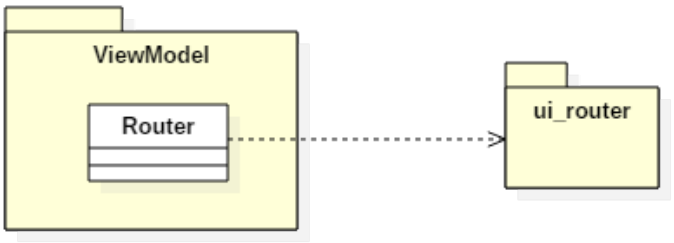


Figura 8: Diagramma del package Router

8.2.5.1 Classe Router

- **Funzione del componente**: implementa il routing dinamico dell'applicazione. Permette di dividere la parte statica dell'applicazione dalla parte che va caricata dinamicamente;
- **Relazione d'uso di altre componenti**: Utilizza il package esterno ui-router. Conosce i Template dell'applicazione.
- **Attività svolte e dati trattati**: il Router collega elementi di navigazione associando Url univoci al caricamento dinamico di specifici template.

9 Diagrammi di attività

9.1 Creazione Questionario

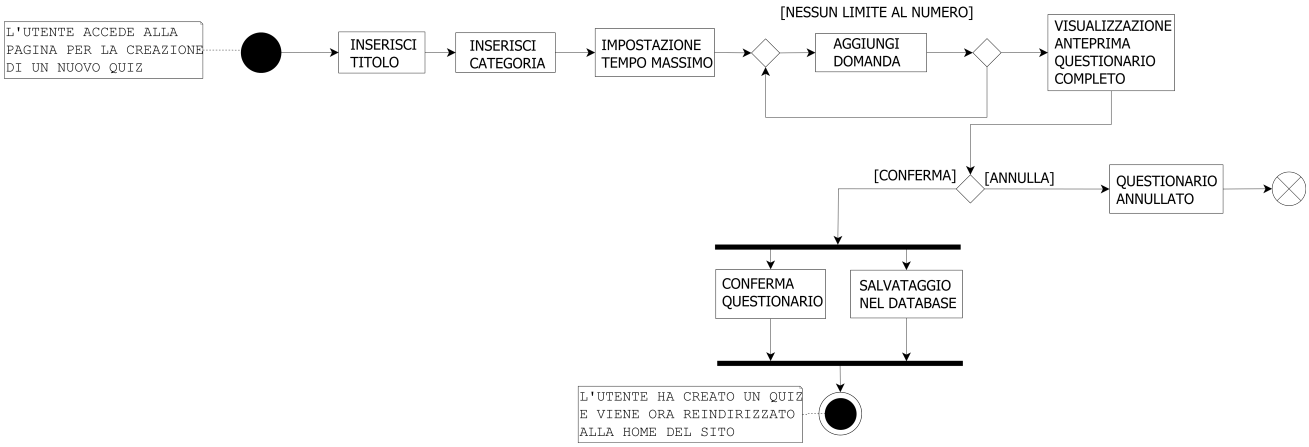


Figura 9: Diagramma di attività che descrive la creazione di un questionario

- **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.

9.2 Creazione Domanda

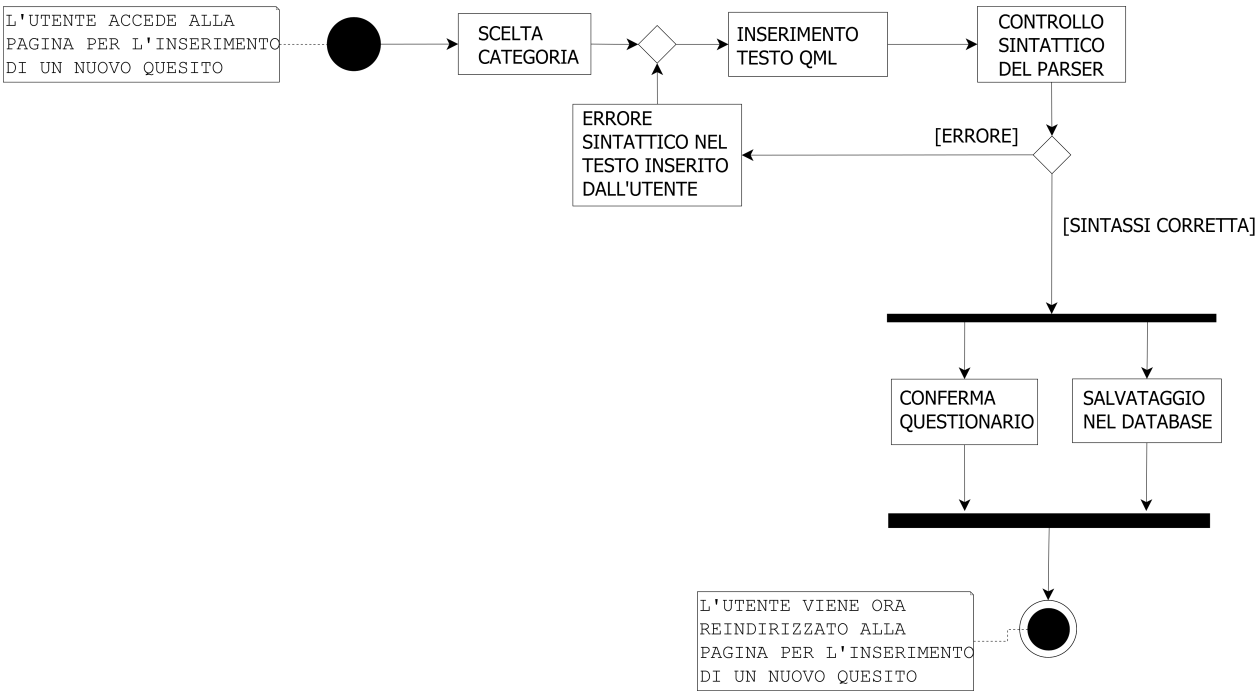


Figura 10: Diagramma di attività sulla creazione di una domanda

- **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).

9.3 Compilazione Questionario

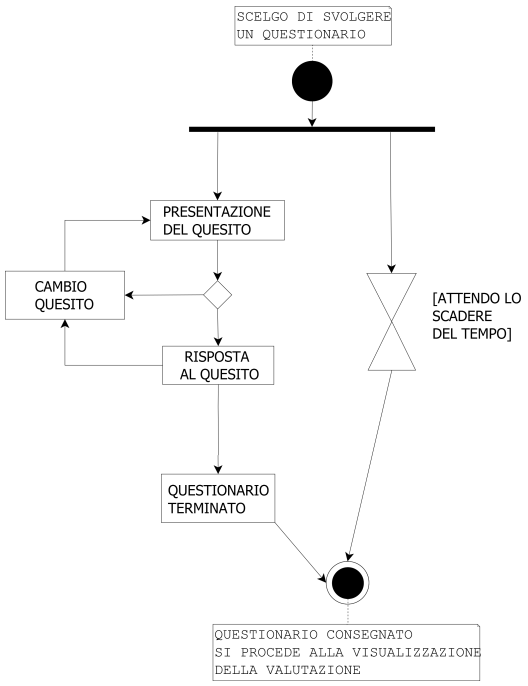


Figura 11: Diagramma di attività sulla compilazione di un questionario

- **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.

9.4 Scelta Questionario

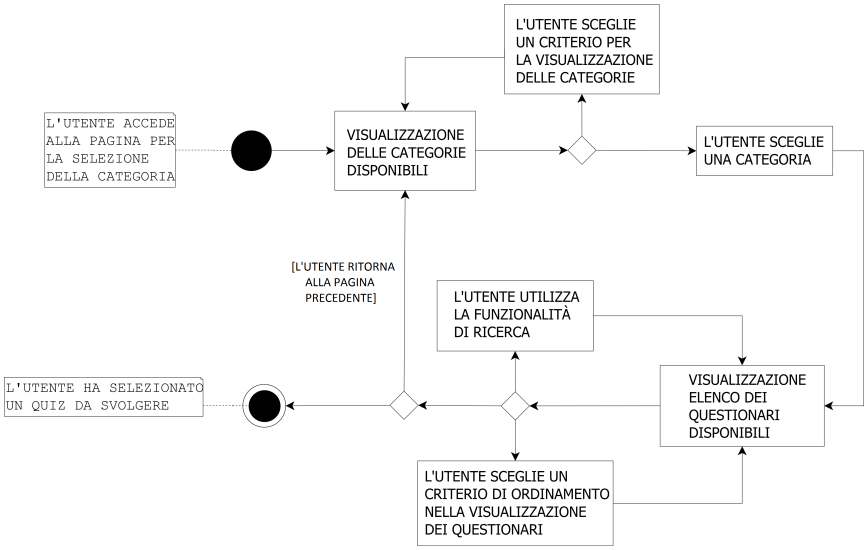


Figura 12: Diagramma di attività sulla scelta di un questionario

- **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.

10 Diagrammi di sequenza

10.1 Salvataggio di una Domanda

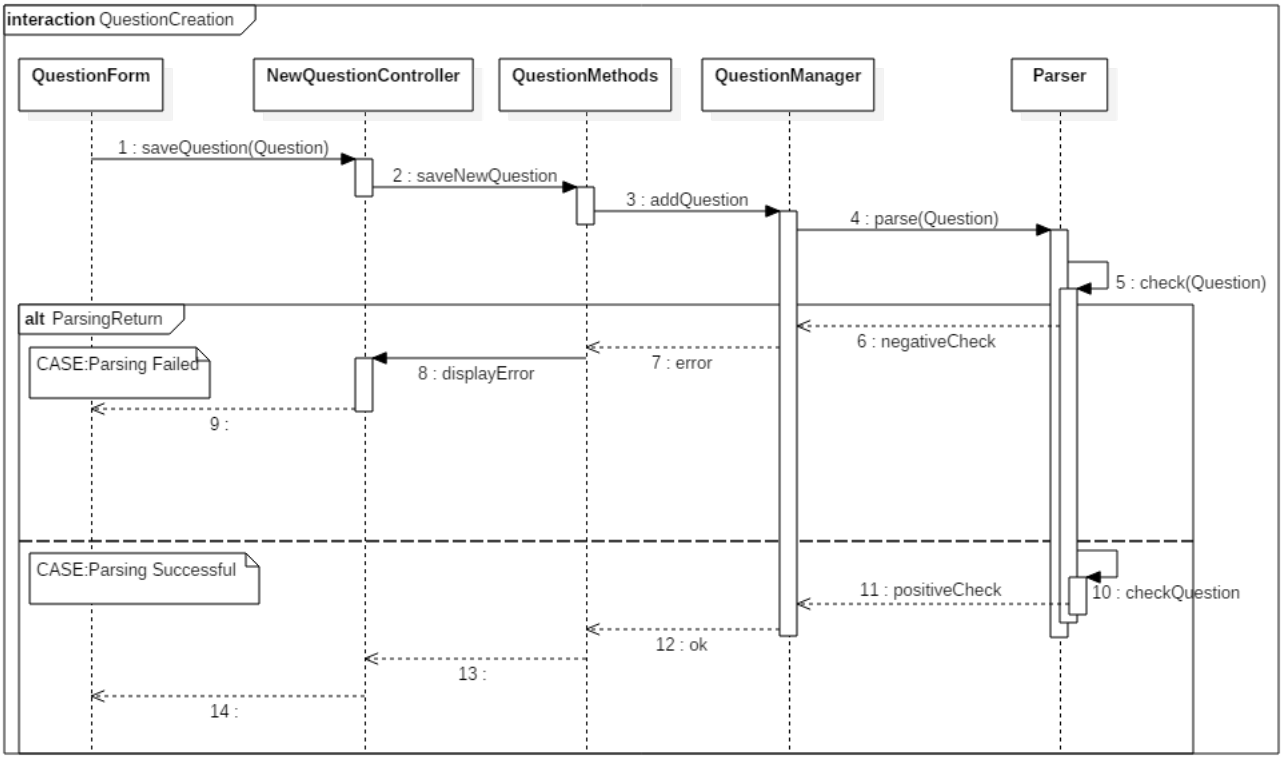


Figura 13: Diagramma di sequenza del salvataggio di una domanda

- **Precondizioni:** l'utente è autenticato nel sistema Quizzipedia e accede all'area del sito dedicata alla creazione di un singolo quesito.

- **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 alla ViewModel che chiama il Model passandogli l'input di testo QML in modo che sia validato dal Parser. 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 alla ViewModel 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).

10.2 Visualizzazione e scelta di un Questionario

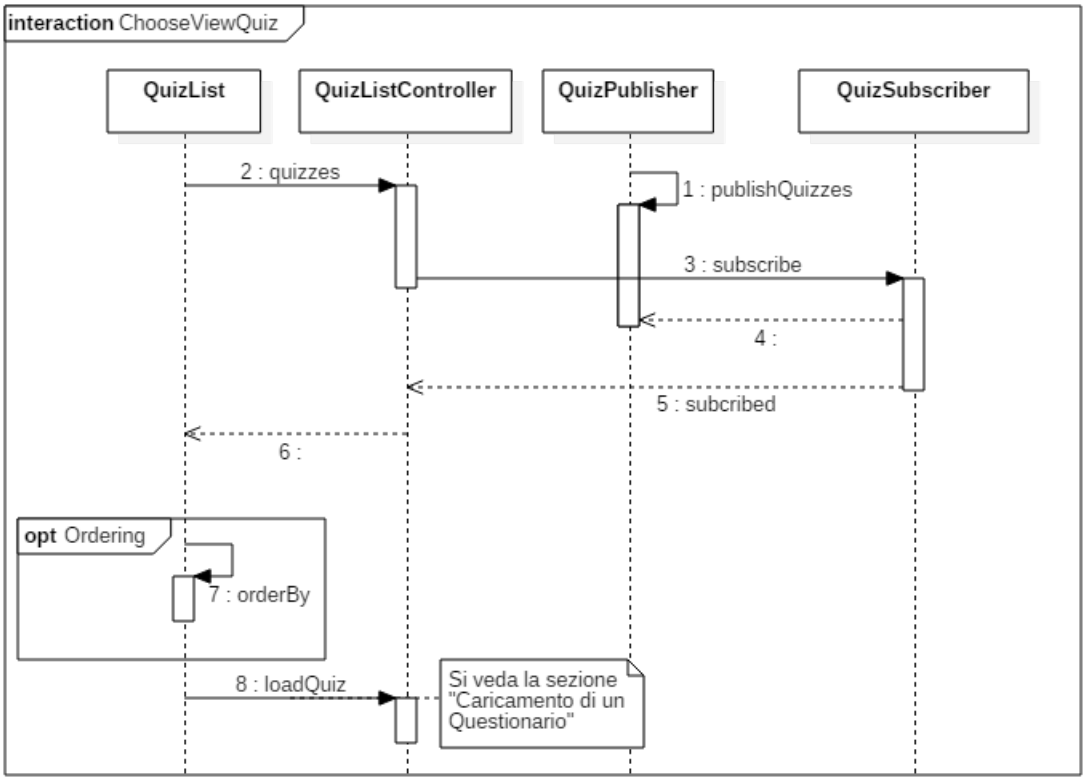


Figura 14: Diagramma di sequenza sulla visualizzazione e scelta di un questionario

- **Precondizioni:** L'utente, intenzionato a svolgere un questionario, accede alla pagina che espone la lista di questionari disponibili.

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

- **Descrizione:** Al caricamento del template QuizList il controller si preoccupa di caricare l' elenco di questionari da visualizzare. Tale collezione di dati viene resa disponibile a QuizListController effettuando il 'subscribe' alla collezione, pubblicata all'avvio del sistema. 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".

10.3 Caricamento di un Questionario

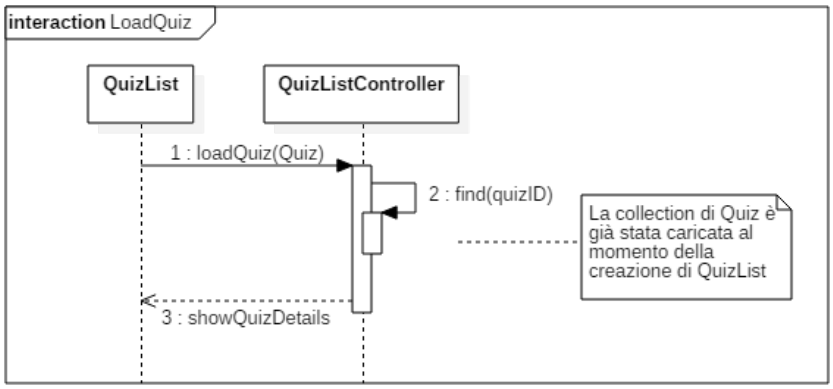


Figura 15: Diagramma di sequenza sul caricamento di un questionario

- **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 alla ViewModel contiene l' identificativo del questionario scelto dall'utente. Il controller di QuizList a questo punto ha già disponibile la collezione di quiz, di cui ha effettuato il subscribe al momento del caricamento del template; per recuperare il quiz scelto quindi è sufficiente una ricerca mirata sulla collezione. Una volta caricato il questionario è pronto per essere somministrato all'utente (il cui diagramma di sequenza è riportato nella sezione "Svolgimento di un Questionario").

10.4 Traduzione di una domanda

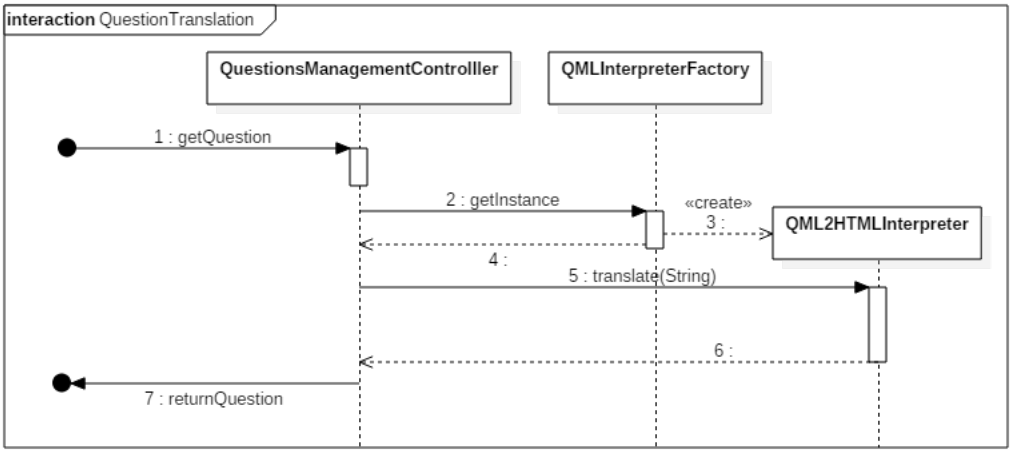


Figura 16: Diagramma di sequenza sulla traduzione di una domanda QML

- **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 il controller delegato s'interfaccia col package Interpreter, richiedendo l'unica istanza di QMLInterpreterFactory, la quale crea un oggetto QML2HTMLInterpreter. A questo punto QuestionsManagementController può richiedere la traduzione del codice QML all'Interpreter ottenuto, e ritornare il risultato al richiedente, che riceverà la domanda tradotta in HTML.

10.5 Svolgimento di un Questionario

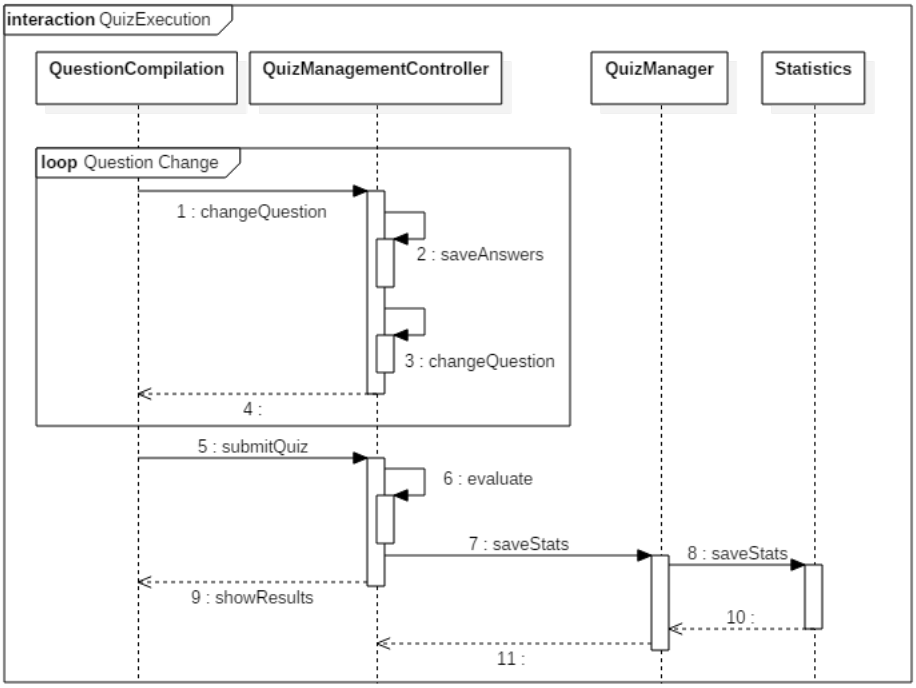


Figura 17: Diagramma di sequenza sullo svolgimento di un questionario

- **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 che preferisce. 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.

11 Design pattern utilizzati

11.1 Abstract Factory

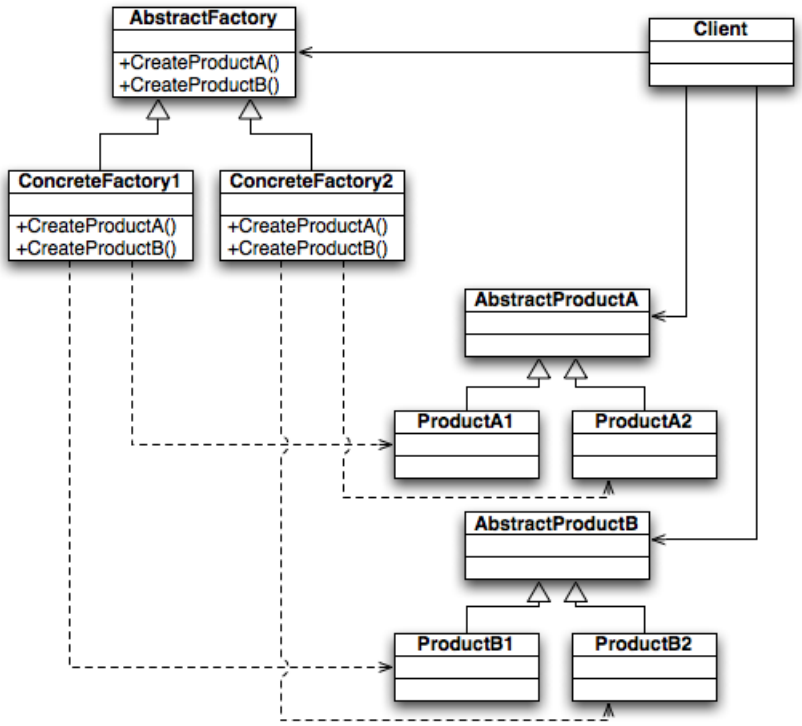


Figura 18: Design Pattern Abstract Factory

- **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 nel package Interpreter. Il package si avvale del pattern per rendere il sistema indipendente dalla creazione delle classi concrete ed essere aperto all'estensione tramite la definizione di nuovi tipi "Interpreter".

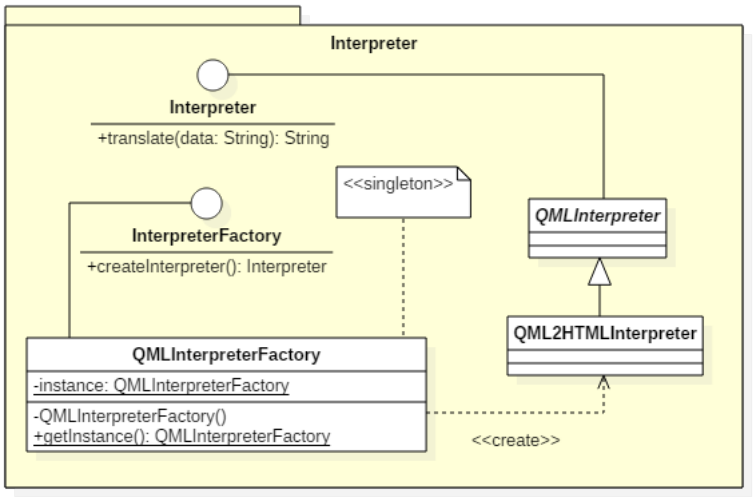


Figura 19: Abstract Factory in Interpreter

11.2 Singleton

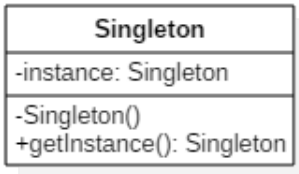


Figura 20: Design Pattern Singleton

- **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.

11.3 Publish – Subscribe

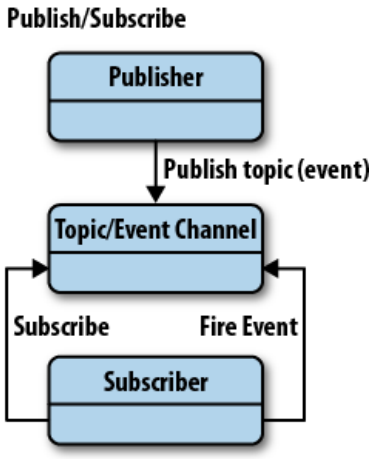


Figura 21: Design Pattern Publish-Subscribe

- **Scopo**: in questo schema, mittenti e destinatari di messaggi dialogano attraverso un tramite, che può essere detto dispatcher o broker. Il mittente di un messaggio (detto publisher) non deve essere consapevole dell'identità dei destinatari (detti subscriber); esso si limita a "pubblicare" (in inglese to publish) il proprio messaggio al dispatcher. I destinatari si rivolgono a loro volta al dispatcher "abbonandosi" (in inglese to subscribe) alla ricezione di messaggi. Il dispatcher quindi inoltra ogni messaggio inviato da un publisher a tutti i subscriber interessati a quel messaggio.

- **Applicabilità**:
  - implementare la comunicazione asincrona tra diverse componenti
  - rendere disponibili al Client solo una parte ristretta dei dati d'interesse.

- **Conseguenze**:
  - disaccoppiamento tra publisher e subscribers
  - alta scalabilità
  - scarsa flessibilità nella struttura dei dati pubblicati.
- **Utilizzo**: nel sistema Quizzipedia il design pattern è stato utilizzato per le classi Publishers e Subscribers, il cui funzionamento è del tutto analogo a quanto descritto sopra. Va notato che nel framework Meteor la componente 'dispatcher' è completamente trasparente al programmatore. E' sufficiente 'pubblicare' i dati lato server e 'abbonarsi' ai dati lato client, Meteor si preoccuperà della comunicazione client-server.

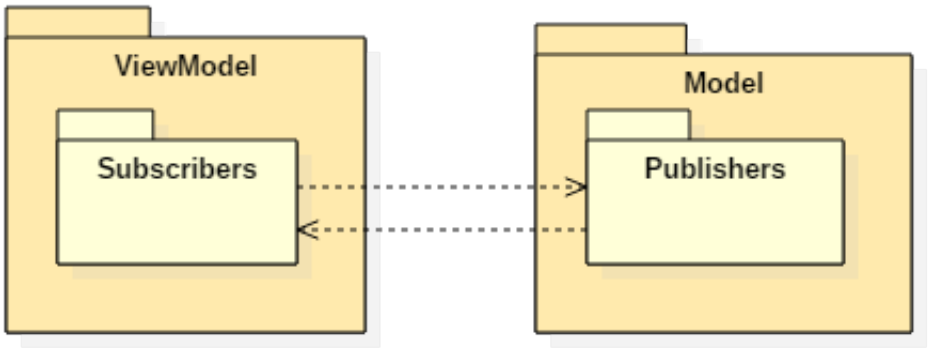


Figura 22: Publish-Subscribe in Quizzipedia

12 Tracciamento

12.1 Mappatura componenti–requisiti

Tabella 2: Mappatura dei componenti sui requisiti

Nome componente	Codice requisito
Model::Database	F 1.2.2 F 3 F 3.1.1 F 3.2 F 6.2.2
Model::Database::QuestionManager::AddQuestion	F 5 F 5.1 F 5.2
Model::Database::QuestionManager::ModifyQuestion	F 5.3
Model::Database::QuestionManager::RemoveQuestion	F 5.4
Model::Database::QuizManager::AddQuiz	F 6 F 6.1 F 6.2
Model::Database::QuizManager::RemoveQuiz	Non tracciabile
Model::Database::UserManager::LogIn	F 2 F 2.2
Model::Database::UserManager::AddUser	F 1 F 2.3.2
Model::Database::UserManager::RemoveUser	Non tracciabile
Model::Parser::Parser	F 5.2.1 F 8
Model::Statistics::Statistics	4.5.2 4.5.2.1 4.5.2.1.1 4.5.2.1.2 4.5.2.1.3
Model::Publishers::QuizPublishers	Non tracciabile
Model::Publishers::QuestionPublishers	Non tracciabile
View::Pages::RegistrationPage	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

Tabella 2: Mappatura dei componenti sui requisiti

Nome componente	Codice requisito
	F 1.2.3
View::Pages::LoginPage	F 2 F 2.1 F 2.1.1 F 2.1.2 F 2.3 F 2.3.1 F 2.3.1.1 F 2.3.2
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 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 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 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

Tabella 2: Mappatura dei componenti sui requisiti

Nome componente	Codice requisito
View::Pages::PasswordRecoveryPage	F 2.2 F 2.2.1 F 2.2.2 F 2.2.2.1 F 2.2.2.2 F 2.2.3
View::Pages::QuizResultPage	4.5.2 4.5.2.1 4.5.2.1.1 4.5.2.1.2 4.5.2.1.3
View::Pages::QuizManagementPage	F 6
View::Pages::ViewTutorialPage	F 8.3
View::Pages::QuizCreationPage	F 6
View::Pages::QuestionCreationPage	F 5
View::Pages::QuestionUpdatePage	F 5.3
View::Templates::QuestionList	F 4.1.1
View::Templates::Question	F 4.2
View::Templates:QuizList:	F 3
View::Templates::Quiz	F 4
View::Templates::QuestionForm	F 5
View::Templates::QuizCreationForm	F 6
View::Templates::QuestionCompilation	F 4
View::Templates::QuizResults	F 4.5.2
View::Templates::RegistrationForm	F 1
View::Templates::LoginForm	F 2
View::Templates::PasswordRecoveryForm	F 2.2
View::Templates::SearchForm	F 3.3
ViewModel::Controllers::NewQuestionController	F 5
ViewModel::Controllers::NewQuizController	F 6
ViewModel::Controllers::QuestionManagementController	F 5.3
ViewModel::Controllers::DeleteQuestionController	F 5.4
ViewModel::Controllers::DeleteQuizController	Non tracciabile
ViewModel::Controllers::QuizManagementController	F 6
ViewModel::Controllers::QuizListController	Non tracciabile
ViewModel::Controllers::QMLEditorController	F 5.2.1 F 8

Tabella 2: Mappatura dei componenti sui requisiti

Nome componente	Codice requisito
ViewModel::Controllers::QuizDetailsController	Non tracciabile
ViewModel::Subscribers::QuestionSubscriber	Non tracciabile
ViewModel::Subscribers::QuizSubscriber	Non tracciabile
ViewModel::Subscribers::UserSubscriber	Non tracciabile
ViewModel::Methods::QuestionMethods	Non tracciabile
ViewModel::Methods::QuizMethods	Non tracciabile
ViewModel::Methods::UserMethods	Non tracciabile
ViewModel::Interpreter::QMLInterpreterFactory	F 8
ViewModel::Interpreter::QMLInterpreter	F 8.1 F 8.2
ViewModel::Interpreter::QML2HTMLInterpreter	F 8 F 8.1 F 8.2
ViewModel::Router	non tracciabile

Tabella 2: Mappatura dei componenti sui requisiti

12.2 Mappatura requisiti – componenti

Tabella 3: Mappatura dei requisiti sui componenti

Codice Requisito	Nome Componente
F 1	View::Pages::RegistrationPage Model::Database::UserManager::AddUser View::Templates::RegistrationForm
F 2	Model::Database::UserManager::LogIn View::Pages::LoginPage View::Pages::PasswordRecoveryPage View::Templates::LoginForm View::Templates::PasswordRecoveryForm
F 3	Model::Database View::Pages::CategoryListPage View::Pages::QuizListPage View::Templates:QuizList: View::Templates::SearchForm
F 4	View::Pages::QuizExecutionPage View::Pages::QuizResultPage Model::Statistics::Statistics View::Templates::QuestionList View::Templates::Question View::Templates::Quiz View::Templates::QuestionCompilation View::Templates::QuizResults
F 5	Model::Database::QuestionManager::AddQuestion Model::Database::QuestionManager::ModifyQuestion Model::Database::QuestionManager::RemoveQuestion Model::Parser::Parser View::Pages::QuestionCreationPage View::Pages::QuestionUpdatePage View::Templates::QuestionForm ViewModel::Controllers::NewQuestionController ViewModel::Controllers::QuestionManagementController ViewModel::Controllers::DeleteQuestionController ViewModel::Controllers::QMLEditorController
F 6	Model::Database::QuizManager::AddQuiz View::Pages::QuizManagementPage View::Pages::QuizCreationPage View::Templates::QuizCreationForm ViewModel::Controllers::NewQuizController ViewModel::Controllers::QuizManagementController
F 8	View::Pages::ViewTutorialPage ViewModel::Controllers::QMLEditorController ViewModel::Interpreter::QMLInterpreterFactory

Tabella 3: Mappatura dei requisiti sui componenti

Codice Requisito	Nome Componente
	ViewModel::Interpreter::QMLInterpreter
	ViewModel::Interpreter::QML2HTMLInterpreter

Tabella 3: Mappatura dei requisiti sui componenti