

QUIZZIPEDIA



team404swe@gmail.com

Specifica Tecnica 1.0

Informazioni sul documento	
Nome Documento	Specifica Tecnica 1.0
Versione	1.0
Uso	Esterno
Data Creazione	22 aprile 2016
Data Ultima Modifica	12 maggio 2016
Redazione	M. Crivellaro - L. Alessio
Verifica	Alex Beccaro
Approvazione	Martin V. Mbouenda
Committente	Zucchetti SPA
Lista di distribuzione	Prof. Vardanega Tullio TEAM404

Registro delle modifiche

Versione	Autore	Data	Descrizione
0.0.7	Davide Bortot (Progettista)	29/04/2016	Stesura della sezione §4.3 "Package Presenter" e delle sue sottosezioni.
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

1	Introduzione	2
1.1	Scopo del documento	2
1.2	Scopo del prodotto	2
1.3	Glossario	2
1.4	Riferimenti	2
1.4.1	Normativi	2
1.4.2	Informativi	2
2	Specifiche del prodotto	3
2.1	Architettura generale del sistema	3
2.2	Descrizione del componente Model	4
2.3	Descrizione del componente View	4
2.4	Descrizione del componente Presenter	5
3	Tecnologie e strumenti utilizzati	6
3.1	HTML5	6
3.2	CSS3	6
3.3	Materialize	6
3.4	Javascript	6
3.5	Node.js	7
3.6	PostgreSQL	7
4	Diagrammi dei packages	7
4.1	Package della componente Model	7
4.2	Package della componente View	7
4.3	Package della componente Presenter	8
4.3.1	Presenter::UserInputManager	8
4.3.2	Presenter::ModelUpdater	8
4.3.3	Presenter::ViewUpdater	9
4.3.4	Presenter::Interpreter	9
5	Diagrammi delle classi	9
5.1	Diagrammi delle classi del Model	9
5.2	Diagrammi delle classi del View	9
5.3	Diagrammi delle classi del Presenter	9
6	Diagrammi di attività	9
7	Diagrammi di sequenza	9
8	Tracciamento	9
9	Analisi di fattibilità	9

Elenco delle figure

Elenco delle tabelle

Sommario

1 Introduzione

1.1 Scopo del documento

1.2 Scopo del prodotto

Il progetto **Quizzipedia** ha come obiettivo lo sviluppo di un sistema software basato su tecnologie Web (Javascript₆, Node.js₆, HTML5₆, CSS3₆) 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 QML₆. 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 HTML₆, 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_2.0.pdf*".

1.3 Glossario

Viene allegato un glossario nel file "*glossario_2.0.pdf*" nel quale viene data una definizione a tutti i termini che in questo documento appaiono con il simbolo '₆' 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_2.0.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/>

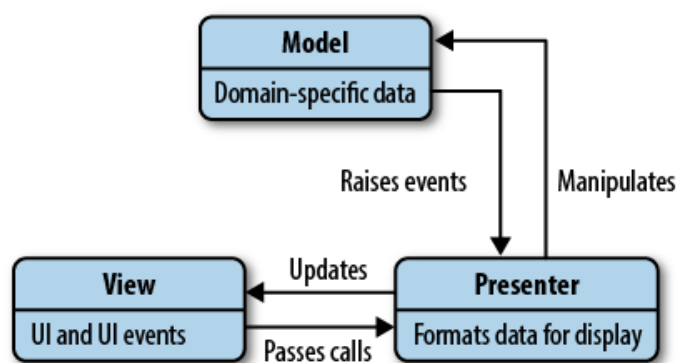
2 Specifiche del prodotto

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. «««« HEAD Nella realizzazione del sistema Quizzipedia verrà adottato il design pattern *Model View Presenter* nella sua variante *Passive View*.

MVP



- **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. Quest'interfaccia sarà l'unico punto di accesso disponibile al Presenter. Per accentrare le funzionalità d'accesso si farà uso del design pattern *Façade*.
- E' responsabile del calcolo ed aggiornamento delle statistiche relative a domande, questionari e utenti.

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 all'utente 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

Framework

- **Utilizzo:**
- **Vantaggi:**
- **Svantaggi:**

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 rivestirà un ruolo importante, verrà infatti utilizzato nella realizzazione del parser/interprete/boh andrea dimmi te.
- **Vantaggi:** eseguito lato Client (sul browser) non sovraccarica il Server per l'esecuzione di richieste, anche se lo script è complesso.

- **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

- **Utilizzo:**
- **Vantaggi:**
- **Svantaggi:**

3.6 PostgreSQL

DBMS ad oggetti open-source.

- **Utilizzo:** Base di dati con lo scopo di memorizzare domande e altri dati necessari al funzionamento del sistema.
- **Vantaggi:** Tecnologia più robusta, stabile e performante di MySQL (inizialmente preso in considerazione ma poi scartato in favore di PostgreSQL).
- **Svantaggi:** Complessità maggiore rispetto al classico MySQL.

4 Diagrammi dei packages

4.1 Package della componente Model

4.2 Package della componente View

QUESTA E' UNA BOZZA, CONTENUTO DA AMPLIARE, RIVEDERE E FORMATTARE
Descrizione: Package per il componente View del pattern architetturale MVP. Contiene le seguenti classi: Navigation UserAuthentication QuizExecution QuizManagement e i seguenti sottopackage: Sender -> singola classe UserEventSender (contiene tanti sottometodi ognuno per ogni situazione) Receiver -> ReceiveUpdate

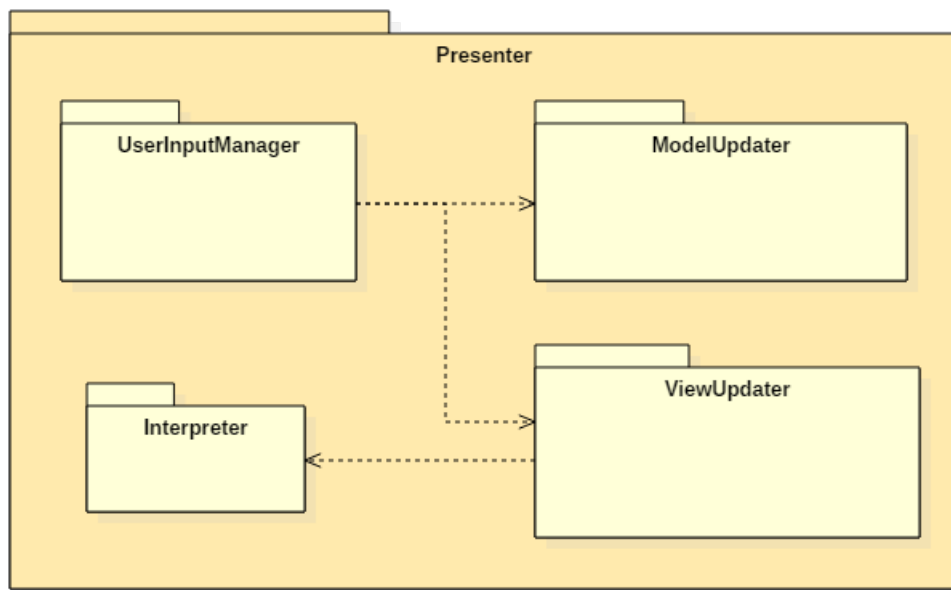
UserAuthentication: regola la registrazione, l'accesso e l'uscita di un utente - login - register - logout

Navigation: regola la navigazione nel sito e la visualizzazione a schermo degli elementi richiesti - showHome - viewCategory - orderBy - chooseCategory - viewQuizList - search - orderBy - chooseQuiz - viewQMLTutorial - viewSiteInfo

QuizExecution: gestisce lo svolgimento dei questionari - doQuiz - viewQuestion - doQuestion - submitAnswer - nextQuestion - prevQuestion - endQuiz - viewResult (- reDo)

QuizManagement: gestisce creazione, modifica ed eliminazione di questionari e singoli quesiti - createQuiz - modifyQuiz - deleteQuiz - submitQuestion - modifyQuestion - deleteQuestion

4.3 Package della componente Presenter



Il package del Presenter contiene i seguenti sub-package:

- *UserInputManager*
- *ModelUpdater*
- *ViewUpdater*
- *Interpreter*

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.

4.3.1 Presenter::*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 i package *ViewUpdater* e *ModelUpdater* a seconda di quale componente deve aggiornare. Il package contiene al suo interno le seguenti classi:

- *InputReceiver*
- *ModelUpdater*
- *ViewUpdater*

4.3.2 Presenter::*ModelUpdater*

Il package *ModelUpdater* ha il compito di riflettere le modifiche della View sul Model. Riceve dalla View i dati, se necessario li elabora, e li passa al Model. Il package contiene al suo interno le seguenti classi:

- *ViewDataReceiver*
- *ToModelSender*

4.3.3 Presenter::ViewUpdater

Il package *ViewUpdater* ha il compito di riflettere le modifiche del Model sulla View. Se i dati che riceve dal Model sono domande QML, queste vanno prima tradotte in linguaggio HTML per essere utilizzate nella View; per fare ciò si interfaccia con il package *Interpreter*. Il package contiene al suo interno le seguenti classi:

- ModelDataReceiver
- ToViewSender
- Translator

4.3.4 Presenter::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 "Interprete", le classi al suo interno sono organizzate seguendo il pattern *Abstract Factory*. Il package contiene al suo interno le seguenti classi:

- Interpreter («interface»)
- AbsInterpreterFactory
- QMLInterpreterFactory
- QMLInterpreter
- QML2HTMLInterpreter

5 Diagrammi delle classi

5.1 Diagrammi delle classi del Model

5.2 Diagrammi delle classi del View

5.3 Diagrammi delle classi del Presenter

6 Diagrammi di attività

7 Diagrammi di sequenza

8 Tracciamento

9 Analisi di fattibilità