Monolith
by Red Babel.

Obelix Group
obelixswe@gmail.com

Specifica Tecnica

Versione	*v1_0_0*
Data creazione	2017-04-21
Redattori	
Verificatori	
Approvazione	
Stato	Approvato
Uso	esterno
Distribuzione	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Red Babel
	Gruppo Obelix

Sommario

Questo documento descrive l'architettura generale del prodotto Monolith che verrà sviluppato dal gruppo Obelix.

Diario delle revisioni

Modifica	Autore e Ruolo	Data	Versione
Approvazione documento	Nicolò Rigato Responsabile	2017-03-09	1.0.0
Verifica del documento	Silvio Meneguzzo Verificatore	2017-03-09	0.1.0
Stesura sezione Resoconto delle attività di verifica	Tomas Mali Verificatore	2017-03-05	0.0.6
Stesura sezione Gestione amministrativa della revisione	Riccardo Saggese Verificatore	2017-03-04	0.0.5
Stesura sezione Strategie di Verifica	Tomas Mali Verificatore	2017-02-28	0.0.4
Stesura sezione Definizione obiettivi di qualità	Riccardo Saggese Verificatore	2017-02-27	0.0.3
Stesura sezione Introduzione	Tomas Mali Verificatore	2017-02-26	0.0.2
Creazione template	Nicolò Rigato Responsabile	2017-02-25	0.0.1

Indice

Elenco delle tabelle

1 Introduzione

1.1 Scopo del documento

Questo documento ha come scopo quello di definire la progettazione ad alto livello per il prodotto Monolith. Verrà presentata l'architettura generale secondo la quale saranno organizzate le varie componenti software e i Design Pattern utilizzati nella creazione dell'SDK, delle bolle predefinite e della demo. Verrà inoltre dettagliato il tracciamento tra le componenti software individuate ed i requisiti.

1.2 Scopo del prodotto

Lo scopo del prodotto è quello di permettere la creazione di bolle interattive, che dovranno funzionare nell'ambiente Rocket.chat. Queste bolle permetteranno di aumentare l'interattività tra gli utenti della chat e aggiungeranno nuove funzionalità accessibili direttamente dalla conversazione senza il bisogno di ricorrere all'apertura di applicazioni diverse. Il sistema offrirà agli sviluppatori un set di $API_{|G|}$ per creare e rilasciare nuove bolle e agli utenti finali la possibilità di usufruire di un insieme di bolle predefinite.

1.3 Glossario

Al fine di evitare ogni ambiguità di linguaggio e massimizzare la comprensione dei documenti, i termini che necessitano di essere chiariti saranno scritti in corsivo e marcati con una |G| in pedice alla prima occorrenza e saranno riportati nel Glossario.

1.4 Riferimenti

1.4.1 Normativi

- **Norme di Progetto**:
  NormediProgetto_v1.1.0
- **Capitolato d'appalto C5**:
  `http://www.math.unipd.it/~tullio/IS-1/2016/Progetto/C5.pdf`
- **Analisi dei Requisiti**:
  AnalisideiRequisiti_v1.0.0

1.4.2 Informativi

- **Slide del corso di Ingegneria del Software**:
  `http://www.math.unipd.it/~tullio/IS-1/2016/`

2 Tecnologie Utilizzate

In questa sezione verranno descritte le tecnologie su cui si basa lo sviluppo del progetto. Per ognuna di esse, verranno indicati l'ambito di utilizzo della tecnologia, i vantaggi e gli svantaggi che ne derivano.Alcune delle tecnologie che saranno usate sono richieste come requisito dal capitolato scelto.

2.1 Javascript 6th edition (ECMA SCRIPT 6)

JavaScript è un linguaggio di scripting orientato agli oggetti e agli eventi. É comunemente utilizzato nella programmazione Web lato client per la creazione, in siti web e applicazioni web, di effetti dinamici interattivi tramite l'uso di funzioni di script invocate da eventi innescati in vari modi dall'utente sulla pagina web in uso.
Come richiesto dal capitolato, per la realizzazione di Monolith, deve essere utilizzato Javascript 6th edition (ECMA SCRIPT 6).

Licenza:
Non esiste una sola implementazione perché ECMAScript (o ES) è un linguaggio di programmazione standardizzato e mantenuto da Ecma International nell'ECMA-262 ed ISO/IEC 16262.

Vantaggi:

- Gestione degli eventi asincroni tramite le promises
- Possibilità di dichiarare classi
- Supporto per le costanti(*const*)
- Possibilità di isolare la definizione di variabili ad un blocco (*let*)
- Possibilità di isolare lo scope di una funzione usando blocchi delimitati da parentesi graffe() come ambienti isolati (vs closure)
- Uso di sintassi più espressiva per scrivere le funzioni anonime (*Arrow Functions*)

Svantaggi:

- Il supporto di ES6 da parte dei browser è ancora incompleto
- L'assenza di tipizzazione potrebbe ostacolare la valutazione della correttezza del codice

2.2 Meteor

Meteor è un framework web JavaScript libero e open source per lo sviluppo di applicazioni web e mobile. É una piattaforma basata su Node.js. Meteor utilizza, dunque, JavaScript sia lato client che lato server.

Licenza: MIT
La licenza MIT è una delle licenze più permissive nel panorama open source. In modo più esplicito dichiara i diritti dati all'utente finale, incluso il diritto di utilizzare , copiare, modificare, incorporare, pubblicare, distribuire, sottolicenziare, e/o vendere il software.

Vantaggi:

- Integrazione con diverse tecnologie utilizzate nello sviluppo web:
  - React
  - MongoDB

- Isomorfismo: il codice javascript scritto funziona in modo trasparente sul client (browser), sul server (Node.js) o in entrambi i mondi
- Ecosistema e modularità: la comunità di Meteor è molto attiva e molte funzionalità client o server potrebbero già essere pacchettizzate dal package manager ufficiale.

Svantaggi:

- Inizialmente sconosciuto ai membri del gruppo.

2.3 Mongo DB

MongoDB è un database NoSQL orientato ai documenti, basato sul formato JSON per la memorizzazione e la rappresentazione dei dati. É distribuito come software libero open source.

Licenza: GNU AGPL v3.0
É una licenza pubblicata da Free Software Foundation. É simile alla capostipite GNU GPL, una licenza fortemente copyleft per software libero.

Vantaggi:

- É più flessibile di un database SQL e facilita la rappresentazione su un modello ad oggetti
- Supporta ricerche per campi, intervalli e regular expression. Le query possono restituire campi specifici del documento e anche includere funzioni definite dall'utente in JavaScript.
- Qualunque campo in MongoDB può essere indicizzato

Svantaggi:

- Inizialmente sconosciuto ai membri del gruppo.

2.4 HTML5

HTML5 è un linguaggio di markup per la strutturazione delle pagine web.

Licenza:
Non esiste una sola implementazione perché HTML5 è un linguaggio di markup standardizzato e mantenuto da W3C.

Vantaggi:

- Codice più pulito e sintassi semplificata rispetto alle versioni precedenti
- Interattività senza l'ausilio di plugin esterni valida per diversi formati multimediali
- Semantica intuitiva grazie ai nuovi TAG di formattazione
- Introduzione della geolocalizzazione, dovuta ad una forte espansione di sistemi operativi mobili
- Sistema più efficiente alternativo ai normali cookie chiamato Web Storage

Svantaggi:

- Non tutti i browser supportano HTML5

2.5 SCSS

SCSS è una sintassi per i fogli di stile introdotta da Sass 3 (Syntactically Awesome StyleSheets). É un'estensione del CSS .

Licenza: MIT

La licenza MIT è una delle licenze più permissive nel panorama open source. In modo più esplicito dichiara i diritti dati all'utente finale, incluso il diritto di utilizzare , copiare, modificare, incorporare, pubblicare, distribuire, sotto-licenziare, e/o vendere il software.

Vantaggi:

- Possibilità di utilizzare variabili
- Possibilità di creare funzioni
- Possibilità di organizzare il foglio di stile in più file
- Compatibilità completa con la sintassi del CSS

Svantaggi:

- Sintassi più complessa.

2.6 React

React è una libreria Javascript open source che permette di costruire interfacce utente.

Licenza: BSD-3-Clause

Le licenze BSD sono una famiglia di licenze permissive, senza copyleft, per software. Le tre clausole della licenza BSD-3-Clause sono:

- Libertà di eseguire il programma per qualsiasi scopo
- Libertà di studiare il programma e modificarlo
- Libertà di ridistribuire copie del programma in modo da aiutare il prossimo

Vantaggi:

- Semplificazione della realizzazione di interfacce UI dinamiche che possono reagire ai cambiamenti di dati in maniera autonoma attraverso opportuni componenti
- Possibilità di utilizzare le viste per creare codice più facile da comprendere e su cui è più semplice effettuare il debugging.

Svantaggi:

- Implementa solo il livello view è quindi necessario utilizzare altre librerie per implementare altre parti dell'applicazione
- Curva di apprendimento ripida
- É una libreria relativamente nuova

2.7 Node.js

Node.js è una piattaforma event-driven per il motore JavaScript V8. Essa permette di realizzare applicazioni web utilizzando il linguaggio JavaScript, tipicamente client-side, per la scrittura anche della parte server-side.

Licenza: MIT
La licenza MIT è una delle licenze più permissive nel panorama open source. In modo più esplicito dichiara i diritti dati all'utente finale, incluso il diritto di utilizzare , copiare, modificare, incorporare, pubblicare, distribuire, sotto-licenziare, e/o vendere il software.

Vantaggi:

- Facile apprendimento
- Possibilità di realizzare applicazioni server-side senza dover imparare linguaggi di programmazione "tradizionali"

Svantaggi:

- Non supporta database relazionali

2.8 Rochet.chat

Rocket.chat è una Web chat server sviluppata in Javascript utilizzando il *Framework*$_{|G|}$ Meteor.

Licenza: MIT
La licenza MIT è una delle licenze più permissive nel panorama open source. In modo più esplicito dichiara i diritti dati all'utente finale, incluso il diritto di utilizzare , copiare, modificare, incorporare, pubblicare, distribuire, sotto-licenziare, e/o vendere il software.

Vantaggi:

- Codice open source
- Possibilità di creare chat di gruppo
- Possibilità di inviare audio, video e file
- Possibilità di effettuare video chiamate
- Community molto attiva

Svantaggi:

- Parzialmente documentata

2.9 Bootstrap

Bootstrap è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web. Essa contiene modelli di progettazione basati su HTML e CSS, sia per la tipografia, che per le varie componenti dell'interfaccia, come moduli, pulsanti e navigazione, così come alcune estensioni opzionali di JavaScript.

Licenza: MIT
La licenza MIT è una delle licenze più permissive nel panorama open source.

In modo più esplicito dichiara i diritti dati all'utente finale, incluso il diritto di utilizzare , copiare, modificare, incorporare, pubblicare, distribuire, sotto-licenziare, e/o vendere il software.

Vantaggi:

- Piattaforma ben standardizzata
- Non richiede l'appoggio né di un linguaggio di programmazione server side, né di un database
- Ottima documentazione
- Responsive Design
- É supportato dai browser moderni

Svantaggi:

- I plugin di jQuery sono limitati
- Le modifiche dovute al continuo sviluppo non sono sempre facili da integrare

2.10 polyglot.js

Polyglot.js è una libreria per la traduzione scritta in JavaScript, eseguita sia per il browser che per gli ambienti CommonJS(Node).

Licenza: MIT

La licenza MIT è una delle licenze più permissive nel panorama open source. In modo più esplicito dichiara i diritti dati all'utente finale, incluso il diritto di utilizzare , copiare, modificare, incorporare, pubblicare, distribuire, sotto-licenziare, e/o vendere il software.

Vantaggi:

- Non è rischiesta una iscrizione per l'utilizzo
- É una libreria non a pagamento
- Il Polyglot ha zero dipendenze
- Copre una traduzione di 30 lingue diverse

2.11 Money.js

Money.js è una libreria semplice con l'unico obiettivo di convertire un valore di denaro da qualsiasi valuta in qualsiasi altra valuta. Money.js utilizza una fusione algoritmica per calcolare un insieme di tassi costantemente preciso per 165 valute mondiali.

Licenza: MIT

La licenza MIT è una delle licenze più permissive nel panorama open source. In modo più esplicito dichiara i diritti dati all'utente finale, incluso il diritto di utilizzare , copiare, modificare, incorporare, pubblicare, distribuire, sotto-licenziare, e/o vendere il software.

Vantaggi:

- Non è rischiesta una iscrizione per l'utilizzo
- É una libreria non a pagamento
- É una libreria semplice da integrare nel codice JavaScript

2.12 weather.js

Weather.js è una libreria che recupera i dati da openweathermap.org e fa la ricerca di tutti i tipi di informazioni relative alle condizioni meteo.

Licenza: MIT

La licenza MIT è una delle licenze più permissive nel panorama open source. In modo più esplicito dichiara i diritti dati all'utente finale, incluso il diritto di utilizzare , copiare, modificare, incorporare, pubblicare, distribuire, sotto-licenziare, e/o vendere il software.

Vantaggi:

- Non è rischiesta una iscrizione per l'utilizzo
- É una libreria non a pagamento
- É una libreria semplice da integrare nel codice JavaScript

Svantaggi:

- Ha bisogno di 11 dipendenze

2.13 classNames

classNames è una semplice utility raccomandata per l'uso con React per l'unione condizionata di className

Licenza: MIT

La licenza MIT è una delle licenze più permissive nel panorama open source. In modo più esplicito dichiara i diritti dati all'utente finale, incluso il diritto di utilizzare , copiare, modificare, incorporare, pubblicare, distribuire, sotto-licenziare, e/o vendere il software.

Vantaggi:

- Semplifica la gestione dei className dinamici
- Non possiede ulteriori dipendenze

3 Descrizione Architettura

3.1 Metodo e formalismo di specifica

Nell'esposizione dell'architettura dell'applicazione si procederà con un approccio top-down, descrivendo l'architettura iniziando dal generale ed andando al particolare. Si procederà quindi alla descrizione dei package, per poi descrivere nel dettaglio le singole classi, specificando per ognuna il tipo, l'obiettivo, la funzione e le relazioni in ingresso ed in uscita. Successivamente si illustreranno degli esempi di uso dei Design Pattern nell'architettura del sistema, rimandando la spiegazione generale alla sezione dedicata. L'architettura dell' SDK e della

demo sono state progettate separatamente. Per i diagrammi delle componenti di classe e di attività, si utilizza il formalismo UML 2.0. Le classi e componenti presenti in librerie o framework esterni vengono contraddistinte da colori diversi. I framework esterni verranno rappresentati con un colore viola, mentre le classi e componenti proprie invece, saranno rappresentate con un colore giallo. L'intera applicazione è progettata utilizzando il framework $Meteor_{|G|}$ che permette di utilizzare il linguaggio JavaScript sia per il lato client che per quello server(tramite NodeJS). I diagrammi delle classi che permettono di mostrare l'architettura generale del sistema vengono affiancati anche dai diagrammi di sequenza e attività, che permettono di definire le interazioni tra le componenti, senza preoccuparsi della loro classificazione.

3.2 Architettura generale

4 Standard di Progetto

4.1 Standard di progettazione architetturale

4.2 Standard di documentazione del codice

4.3 Standard di denominazione di entità e relazioni

4.4 Standard di programmazione

4.5 Strumenti di lavoro

5 Diagrammi di Attività

Il diagramma delle attività è un diagramma definito all'interno dello Unified Modeling Language (UML) che definisce le attività da svolgere per realizzare una data funzionalità. Può essere utilizzato durante la progettazione del software per dettagliare un determinato algoritmo. Più in dettaglio, un activity diagram definisce una serie di attività o flusso, anche in termini di relazioni tra le attività, i responsabili per le singole attività e i punti di decisione. L'activity diagram è spesso usato come modello complementare allo Use Case Diagram, per descrivere le dinamiche con cui si sviluppano i diversi use case.

5.1 Configurazione sondaggio

Questo diagramma rappresenta la configurazione e l'invio della bolla sondaggio. L'utente inserisce il titolo del sondaggio, dopodichè avviene l'inserimento delle opzioni. L'utente ha la possibilità di inserire più opzioni (rappresentato nel diagramma con la freccia che parte dal nodo decisione e raggiunge il nodo azione chiamato inserisci opzioni). Una volta compiuto il processo di configurazione l'utente decide di inviare la bolla (rappresentato dal nodo invia). Se il numero delle opzioni inserite è minore di due allora il flusso ritorna al nodo azione chiamato "inserisci opzione" per poter permettere all'utente di incrementare il numero delle opzioni del sondaggio. Altrimenti (il numero di opzioni è maggiore di uno) il flusso arriva al nodo di fine dell'attività.

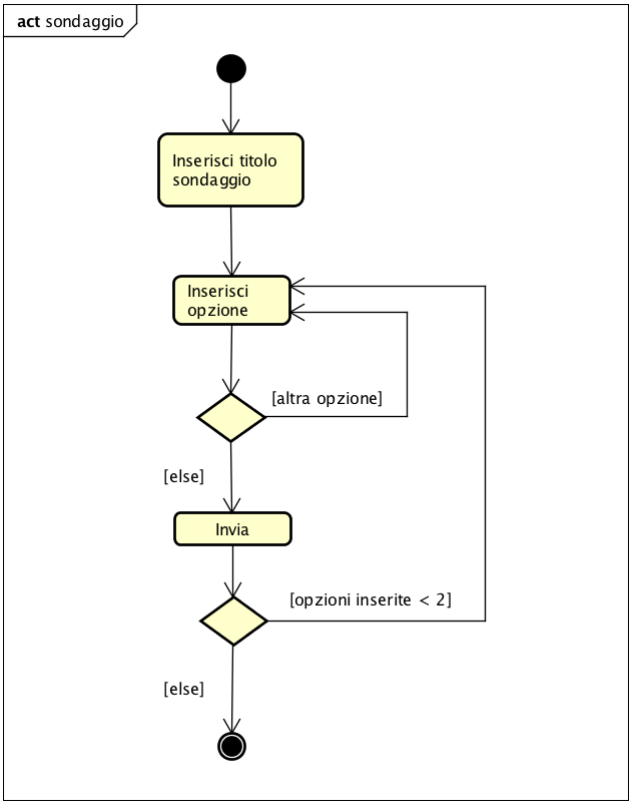

Figura 1: Diagramma di attività per la bolla sondaggio

5.2 configurazione ListBubble

Questo diagramma rappresenta la configurazione della bolla ListBubble. Il flusso inizia con l'inserimento del titolo della lista. Dopo questo processo il flusso raggiunge il nodo decisione dove l'utente sceglie il modo di inserimento degli elementi. L'inserimento degli elementi avviene o manualmente (inserisci elemento manualmente) oppure con la selezione nella lista degli elementi predefiniti (inserisci elemento da checklist). Nel caso l'utente scegliesse di inserire un elemento dalla checklist predefinita, il flusso raggiunge il nodo decisione. Da questo punto l'utente può scegliere di inviare la bolla già configurata e il flusso raggiunge il nodo terminale, oppure scegliere di ritornare nel nodo decisione raggiunto precedentemente per poter inserire un altro elemento. Anche nel caso in cui l'utente scegliesse di inserire un elemento manualmente il flusso segue lo stesso percorso, ovvero l'utente inserisce l'elemento e decide di inviare la bolla con il raggiungendo il nodo terminale oppure inserire un altro elemento.

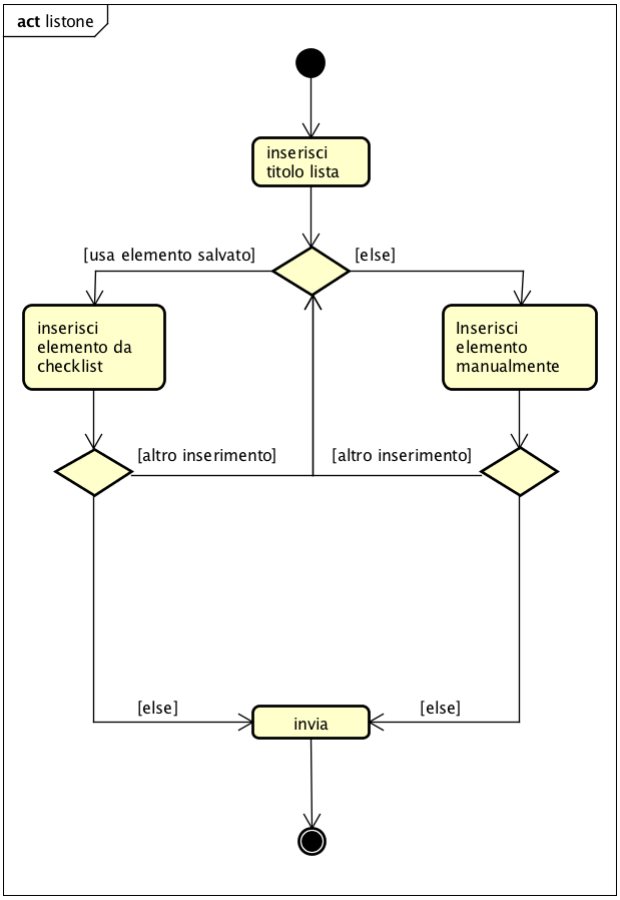

Figura 2: Diagramma di attività per la bolla ListBubble

6 Diagramma di Sequenza

7 Design Pattern

8 Tracciamento

A Descrizione Design Pattern

Un design pattern è una soluzione progettuale elegante e generale ad un problema ricorrente. In particolare si tratta di una descrizione o modello logico da applicare per la risoluzione di un problema che può presentarsi in diverse situazioni durante la fase di progettazione e sviluppo del software, ancora prima della definizione dell'algoritmo ricorsivo della parte computazionale. Essi si suddividono in quattro categorie:

- **Architetturali :** esprimono schemi di base per impostare l'organizzazione strutturale di un sistema software;

- **Creazionali :** forniscono un'astrazione del processo di istanziazione degli oggetti;
- **Strutturali :** si occupano delle modalità di composizione di classi e oggetti per formare strutture complesse;
- **Comportamentali :** si occupano di algoritmi e dell'assegnamento di responsabilità tra oggetti collaboranti.

A.1 Design Pattern Utilizzati

A.1.1 Factory Method

Rappresenta uno dei pattern creazionali adottati dal gruppo Obelix, esso indirizza il problema della creazione di oggetti senza specificarne l'esatta classe. Questo pattern raggiunge il suo scopo fornendo un'interfaccia per creare un oggetto, ma lascia che le sottoclassi decidano quale oggetto istanziare.

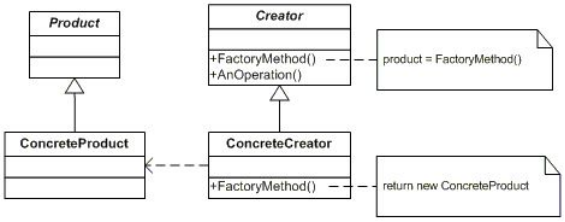

Figura 3: factory method

Product definisce l'interfaccia implementata da ConcreteProduct, creator definisce il factory method che restituisce una interfaccia di tipo product, ConcreteCreator definisce il metodo factory effettivo per la creazione di un'istanza particolare di tipo Product.

I motivi che portano alla scelta del suo utilizzo sono:

- La creazione di un oggetto preclude il suo riuso senza una significativa duplicazione di codice
- La creazione di un oggetto richiede l'accesso ad informazioni o risorse che non dovrebbero essere contenute nella classe di composizione
- La gestione del ciclo di vita degli oggetti gestiti deve essere centralizzata in modo da assicurare un comportamento coerente all'interno dell'applicazione

polyglot.js money.js weather.js