



Specifica Tecnica

Gruppo SWEet BIT – Progetto SWEDesigner

Informazioni sul documento

Versione	1.0.0
Redazione	Santimaria Davide Massignan Fabio
Verifica	Massignan Fabio Bodian Malick
Approvazione	Pilò Salvatore
Uso	Esterno
Distribuzione	Prof. Tullio Vardanega Prof. Riccardo Cardin Gruppo SWEet BIT Zucchetti S.p.A.

Descrizione

Questo documento descrive la specifica tecnica e l'architettura del prodotto sviluppato dal gruppo SWEet BIT per la realizzazione del progetto SWEDesigner.

Versioni del documento

Versione	Data	Persone coinvolte	Descrizione
1.3.0	2017/??/??	Pilò Salvatore	Approvazione Documento
1.1.0	2017/??/??	Massignan Fabio	Verifica Documento
1.0.1	2017/05/02	NOME	Stesura sezione Descrizione architettura
1.0.1	2017/05/02	NOME	Stesura sezione Tecnologie utilizzate
1.0.1	2017/05/02	NOME	Stesura sezione Introduzione
1.0.0	2017/05/02	Santimaria Davide	Creazione struttura documento

Indice

1	Introduzione	5
1.1	Scopo del documento	5
1.2	Scopo del prodotto	5
1.3	Glossario	5
1.4	Riferimenti	5
1.4.1	Normativi	5
1.4.2	Informativi	6
2	Tecnologie utilizzate	7
2.1	Server	7
2.1.1	Node.js	7
2.2	Client	7
2.2.1	Express.js	7
2.2.2	MongoDB	8
2.2.3	Mongoose	8
2.2.4	mxGrafh	8
3	Descrizione architettura	9
3.1	Metodo e formalismo di specifica	9
3.2	Architettura generale	9
3.3	Interfaccia REST-like	9
3.4	Architettura del Server	9
3.5	Architettura del Client	9
4	Back-end	10
4.1	Interfaccia REST	10
4.2	Descrizione packages e classi	10
4.2.1	Back-end	10
4.2.1.1	Informazioni sul package	10
4.2.2	Back-end::Lib	10
4.2.2.1	Informazioni sul package	10
4.2.3	Back-end::Lib::AuthModel	10
4.2.3.1	Informazioni sul package	10
4.2.3.2	Classi	10
4.2.4	Back-end::Lib::Whatever	10
4.2.4.1	Informazioni sul package	10
4.2.4.2	Classi	10
4.3	Scenari	11
4.3.1	Gestione generale delle richieste	11
4.3.2	Fallimento vincolo "utente autenticato"	11
4.3.3	Fallimento vincolo "utente non autenticato"	11

4.3.4	Richiesta POST /login	11
4.3.5	Richiesta DELETE /logout	11
4.4	Descrizione librerie aggiuntive	11
5	Front-end	12
5.1	Descrizione packages e classi	12
5.1.1	Front-end	12
5.1.1.1	Informazioni sul package	12
5.1.2	Front-end::Controllers	12
5.1.2.1	Informazioni sul package	12
5.1.2.2	Classi	12
5.1.3	Front-end::Services	12
5.1.3.1	Informazioni sul package	12
5.1.3.2	Classi	12
5.1.4	Front-end::Model	12
5.1.4.1	Informazioni sul package	12
5.1.4.2	Classi	12
6	Diagrammi delle attività	13
6.1	Applicazione SWEDesigner	13
6.1.1	Attività principali	13
6.1.2	Registrazione	13
6.1.3	Recupero password	13
6.1.4	Login	13
6.1.5	Modifica profilo	13
6.1.6	Altro	13
7	Stime di fattibilità e di bisogno e di risorse	14
8	Design pattern	15
8.1	Design Pattern Architetture	15
8.1.1	MVVM	15
8.1.2	Dependency Injection	15
8.2	Design Pattern Creazionali	15
8.2.1	Factory ad esempio	15
8.3	Design Pattern Strutturali	15
8.3.1	Decorator	15
8.3.2	Facade	15
8.4	Design Pattern Comportamentali	15
8.4.1	Observer	15
8.4.2	Command	15
9	Tracciamento	16
9.1	Tracciamento componenti - requisiti	16

9.2	Tracciamento requisiti - componenti	16
10	Appendici	17
A	Descrizione Design Pattern	17
A.1	Design Pattern Architetture	17
A.1.1	MVVM	17
A.1.2	Dependency Injection	17
A.2	Design Pattern Creazionali	17
A.2.1	Factory ad esempio	17
A.3	Design Pattern Strutturali	17
A.3.1	Decorator	17
A.3.2	Facade	17
A.4	Design Pattern Comportamentali	17
A.4.1	Observer	17
A.4.2	Command	17

1 Introduzione

1.1 Scopo del documento

Questo documento ha come scopo quello di definire la *progettazione ad alto livello_G* per il prodotto. Verrà presentata la struttura generale secondo la quale saranno organizzate le varie componenti software e i *Design Pattern_G* utilizzati nella creazione del prodotto SWEDesigner. Verrà dettagliato il tracciamento tra le componenti software individuate ed i requisiti.

1.2 Scopo del prodotto

Lo scopo del progetto è la realizzazione di una *Web App_G* che fornisca all'*Utente_G* un *UML_G Designer_G* con il quale riuscire a disegnare correttamente *Diagrammi_G* delle *Classi_G* e descrivere il comportamento dei *Metodi_G* interni alle stesse attraverso l'utilizzo di *Diagrammi_G* delle attività. La *Web App_G* permetterà all'*Utente_G* di generare *Codice_G Java_G* dall'insieme dei *diagrammi classi_G* e dei rispettivi *metodi_G*.

1.3 Glossario

Con lo scopo di evitare ambiguità di linguaggio e di massimizzare la comprensione dei documenti, il gruppo ha steso un documento interno che è il *Glossario v1.2.0*. In esso saranno definiti, in modo chiaro e conciso i termini che possono causare ambiguità o incomprensione del testo.

1.4 Riferimenti

1.4.1 Normativi

- **Capitolato d'Appalto C6: SWEDesigner**
<http://www.math.unipd.it/~tullio/IS-1/2015/Progetto/C6p.pdf>
- **Norme di Progetto:** *Norme di Progetto v1.2.0*.
- **Analisi dei Requisiti:** *Analisi dei Requisiti v1.2.0*.

1.4.2 Informativi

- Slide dell'insegnamento Ingegneria del Software modulo A:
<http://www.math.unipd.it/~tullio/IS-1/2016/>.

2 Tecnologie utilizzate

L'architettura è stata progettata utilizzando lo stack di **MEAN_G** (<http://mean.io/>), il quale comprende 4 tecnologie, alcune delle quali espressamente richieste nel *capitolato_G* d'appalto. Vengono di seguito elencate e descritte le principali tecnologie impiegate comprese in **MEAN_G** e le motivazioni del loro utilizzo:

- **Node.js**: piattaforma per il *back-end_G*;
- **Express.js**: *framework_G* per la realizzazione dell'applicazione web in *Node.js_G* ;
- **MongoDB**: *database_G* di tipo *NoSQL_G* per la parte di recupero e salvataggio dei dati;
- **Mongoose**: *libreria_G* per interfacciarsi con il driver di **MongoDB**;
- **Angular.js**: *framework_G* *JavaScript_G* la realizzazione del *front-end_G* .

2.1 Server

2.1.1 Node.js

Node.js è una *piattaforma_G* software costruita sul motore *JavaScript_G* di *Chrome_G* che permette di realizzare facilmente applicazioni di rete scalabili e veloci. *Node.js_G* utilizza *JavaScript_G* come linguaggio di programmazione, e grazie al suo modello *event-driven_G* con chiamate di input/output non bloccanti risulta essere leggero e efficiente. I principali vantaggi dell'utilizzo di *Node.js_G* sono:

- **Approccio asincrono**: *Node.js_G* permette di accedere alle risorse del sistema operativo in modalità *event-driven_G* e non sfruttando il classico modello basato su processi concorrenti utilizzato dai classici web *server_G*. Ciò garantisce una maggiore efficienza in termini di prestazioni, poiché durante le attese il runtime può gestire qualcos'altro in maniera asincrona.
- **Architettura modulare**: Lavorando con *Node.js_G* è molto facile organizzare il lavoro in librerie, importare i *moduli_G* e combinarli fra loro. Questo è reso molto comodo attraverso il *node package manager_G* (**npm**) attraverso il quale lo sviluppatore può contribuire e accedere ai *package_G* messi a disposizione dalla community.

2.2 Client

2.2.1 Express.js

Express.js è un *framework_G* minimale per creare *Web App_G* con *Node.js_G*. Richiede *moduli_G* Node di terze parti per applicazioni che prevedono l'interazione con le *database_G*.

È stato utilizzato il *framework_G Express.js_G* per supportare lo sviluppo dell'applicazione *server_G* grazie alle utili e robuste caratteristiche da esso offerte, le quali sono pensate per non oscurare le funzionalità fornite da *Node.js_G* aprendo così le porte all'utilizzo di moduli per *Node.js_G* atti a supportare specifiche funzionalità.

2.2.2 MongoDB

MongoDB_G è un *database_G NoSQL_G open source_G* scalabile e altamente performante di tipo document-oriented, in cui i dati sono archiviati sotto forma di documenti in stile *JSON_G* con schemi dinamici, secondo una struttura semplice e potente.

I principali vantaggi derivati dal suo utilizzo sono:

- **Alte performance:** non ci sono join che possono rallentare le operazioni di lettura o scrittura. L'indicizzazione include gli indici di chiave anche sui documenti innestati e sugli array, permettendo una rapida interrogazione al *database_G*;
- **Affidabilità:** alto meccanismo di replicazione su server;
- **Schemaless:** non esiste nessuno *schema_G*, è più flessibile e può essere facilmente trasposto in un modello ad oggetti;
- Permette di definire query complesse utilizzando un linguaggio che non è *SQL_G*;
- Permette di processare parallelamente i dati (*Map-Reduce_G*);
- Tipi di dato più flessibili.

2.2.3 Mongoose

Mongoose è una *libreria_G* per interfacciarsi a *MongoDB_G* che permette di definire degli schemi per modellare i dati del *database_G*, imponendo una certa struttura per la creazione di nuovi Document. Inoltre fornisce molti strumenti utili per la validazione dei dati, per la definizione di query e per il cast dei tipi predefiniti. Per interfacciare l'applicazione *server_G* con *MongoDB_G* sono disponibili diversi progetti *open source_G*. Per questo progetto è stato scelto di utilizzare *Mongoose.js_G*, attualmente il più di uso.

2.2.4 mxGrafh

3 Descrizione architettura

3.1 Metodo e formalismo di specifica

Le scelte architetturali per lo sviluppo di SWEDesigner sono state fortemente influenzate dallo stack tecnologico utilizzato.

Nell'esposizione dell'architettura dell'applicazione si procederà con un approccio *top-down_G*, descrivendo l'architettura iniziando dal generale ed andando al particolare; si è partiti suddividendo il sistema in *front-end_G* e *back-end_G*, definendo l'interfaccia di comunicazione, scegliendo di seguire in ciascuno l'organizzazione suggeritaci dai framework (Express e Angular.js).

La descrizione dell'architettura di SWEDesigner è suddivisa in quattro sezioni:

- §3.2: che illustra gli aspetti generali dell'architettura del software;
- §3.3: che descrive il protocollo che lega le due interfacce tra *Client_G* e *Server_G*; che descrive l'architettura del front end dell'applicazione;
- §3.4: che descrive l'architettura del *back-end_G* dell'applicazione;
- §3.5: che descrive l'architettura del *front-end_G* dell'applicazione.

I vari tipi di diagrammi presentati di seguito utilizzano la specifica *UML_G* 2.0.

3.2 Architettura generale

3.3 Interfaccia REST-like

3.4 Architettura del Server

3.5 Architettura del Client

4 Back-end

4.1 Interfaccia REST

4.2 Descrizione packages e classi

4.2.1 Back-end

4.2.1.1 Informazioni sul package

4.2.2 Back-end::Lib

4.2.2.1 Informazioni sul package

4.2.3 Back-end::Lib::AuthModel

4.2.3.1 Informazioni sul package

4.2.3.2 Classi

4.2.4 Back-end::Lib::Whatever

4.2.4.1 Informazioni sul package

4.2.4.2 Classi

4.3 Scenari

4.3.1 Gestione generale delle richieste

4.3.2 Fallimento vincolo "utente autenticato"

4.3.3 Fallimento vincolo "utente non autenticato"

4.3.4 Richiesta POST /login

4.3.5 Richiesta DELETE /logout

4.4 Descrizione librerie aggiuntive

5 Front-end

5.1 Descrizione packages e classi

5.1.1 Front-end

5.1.1.1 Informazioni sul package

5.1.2 Front-end::Controllers

5.1.2.1 Informazioni sul package

5.1.2.2 Classi

5.1.3 Front-end::Services

5.1.3.1 Informazioni sul package

5.1.3.2 Classi

5.1.4 Front-end::Model

5.1.4.1 Informazioni sul package

5.1.4.2 Classi

6 Diagrammi delle attività

6.1 Applicazione SWEDwsigner

6.1.1 Attività principali

6.1.2 Registrazione

6.1.3 Recupero password

6.1.4 Login

6.1.5 Modifica profilo

6.1.6 Altro

7 Stime di fattibilità e di bisogno e di risorse

8 Design pattern

8.1 Design Pattern Architetture

8.1.1 MVVM

8.1.2 Dependency Injection

8.2 Design Pattern Creazionali

8.2.1 Factory ad esempio

8.3 Design Pattern Strutturali

8.3.1 Decorator

8.3.2 Facede

8.4 Design Pattern Comportamentali

8.4.1 Observer

8.4.2 Command

9 Tracciamento

9.1 Tracciamento componenti - requisiti

9.2 Tracciamento requisiti - componenti

10 Appendici

A Descrizione Design Pattern

A.1 Design Pattern Architetture

A.1.1 MVVM

A.1.2 Dependency Injection

A.2 Design Pattern Creazionali

A.2.1 Factory ad esempio

A.3 Design Pattern Strutturali

A.3.1 Decorator

A.3.2 Facede

A.4 Design Pattern Comportamentali

A.4.1 Observer

A.4.2 Command