Definizione di Prodotto

Gruppo SWEet BIT — ***Progetto SWEDesigner***

Informazioni sul documento	
Versione	1.0.0
Redazione	
Verifica	
Approvazione	
Uso	Esterno
Distribuzione	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Zucchetti S.p.A.

Descrizione

Questo documento descrive la struttura e le relazioni tra le parti del prodotto SWEDesigner del gruppo SWEet BIT.

Registro delle modifiche

Versione	Data	Persone coinvolte	Descrizione

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

1.1 Scopo del documento

Il presente documento ha lo scopo di definire in dettaglio la struttura e il funzionamento delle componenti del prodotto SWEDesigner. Questo documento servirà come guida per i componenti del gruppo fornendo direttive e vincoli per la realizzazione del progetto.

1.2 Scopo del prodotto

Lo scopo del progetto è la realizzazone 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 v*2.0.0. 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-1/2016/Progetto/C6p.pdf;
- **Norme di Progetto:** *Norme di Progetto v2.0.0.*
- **Analisi dei Requisiti:** *Analisi dei Requisiti v2.0.0.*

1.4.2 Informativi

- **Slide dell'insegnamento Ingegneria del Software modulo A:** http://www.math.unipd.it/~tullio/IS-1/2016/.
  - **Slides del corso di Ingegneria del Software mod. A: *Diagrammi delle classi*$_G$**: http://www.math.unipd.it/~tullio/IS-1/2015/Dispense/E03.pdf;
  - **Slides del corso di Ingegneria del Software mod. A: Diagrammi dei package**: http://www.math.unipd.it/~tullio/IS-1/2015/Dispense/E04.pdf;
  - **Slides del corso di Ingegneria del Software mod. A: Diagrammi di sequenza**: http://www.math.unipd.it/~tullio/IS-1/2015/Dispense/E05.pdf;
  - **Slides del corso di Ingegneria del Software mod. A: Diagrammi di attività**: http://www.math.unipd.it/~tullio/IS-1/2015/Dispense/E06.pdf;
  - **Slides del corso di Ingegneria del Software mod. A: *Design pattern*$_G$ strutturali: Decorator, Proxy, Facade, Adapter**:http://www.math.unipd.it/~tullio/IS-1/2015/Dispense/E07.pdf;
  - **Slides del corso di Ingegneria del Software mod. A: *Design pattern*$_G$ creazionali: Singleton, Builder, Abstract Factory**: http://www.math.unipd.it/~tullio/IS-1/2015/Dispense/E08.pdf;
  - **Slides del corso di Ingegneria del Software mod. A: *Design pattern*$_G$ comportamentali: Observer, Template Method, Command, Strategy, Iterator**: http://www.math.unipd.it/~tullio/IS-1/2015/Dispense/E09.pdf;
- **Design Patterns - E. Gamma, R. Helm, R. Johnson, J. Vlissides (Pearson Education, Addison-Wesley, 1995;**;
- ***Node.js*$_G$**: https://nodejs.org/dist/latest-v6.x/docs/api/;
- **MongoDB**: https://docs.mongodb.org/manual/;
- **HTML5**: http://www.w3schools.com/html/html5_intro.asp;
- **CSS3**: http://www.w3schools.com/css/css3_intro.asp;
- **ExpressJS**: http://expressjs.com/en/4x/api.html.
- **Mustache**: http://mustache.github.io/.

2 Standard di progetto

2.1 Standard di progettazione architetturale

Gli standard di progettazione sono definiti *Specifica Tecnica v* 1.0.0 .

2.2 Standard di documentazione del codice

Gli standard per la scrittura della documentazione del codice sono definiti nelle Norme di Progetto 2.0.0.

2.3 Standard di denominazione di entità e relazioni

Tutti gli elementi definiti come package, classi, metodi o attributi, devono avere denominazioni chiare ed esplicative. Il nome deve avere una lunghezza tale da non pregiudicarne la leggibilità e chiarezza. È preferibile utilizzare dei sostantivi per le entità e dei verbi per le relazioni. Le abbreviazioni sono ammesse se:

- immediatamente comprensibili;
- non ambigue;
- sufficientemente contestualizzate.

Le regole tipografiche relative ai nomi delle entità sono definite nelle *Norme di Progetto v*2.0.0.

2.4 Standard di programmazione

Gli standard di programmazione sono definiti e descritti nelle *Norme di Progetto v*2.0.0.

2.5 Strumenti di lavoro

Per gli strumenti di lavoro da utilizzare durante la codifica e le procedure per il loro corretto funzionamento e coordinamento si rimanda al documento *Norme di Progetto v*2.0.0.

3 Specifica Front-End

3.1 SWEDesigner::Client

3.1.1 Informazioni generali

3.1.2 Classi

3.2 SWEDesigner::Client::Components

3.2.1 Informazioni generali

3.2.2 Classi

3.3 SWEDesigner::Client::Components::Activity-Frame

3.3.1 Informazioni generali

3.3.2 Classi

3.4 SWEDesigner::Client::Components::Editor

3.4.1 Informazioni generali

3.4.2 Classi

3.4.2.1 SWEDesigner::Client::Components::Editor::Class-Menu

- **Descrizione:**
- **Utilizzo:**
- **Metodi:**

3.4.2.2 SWEDesigner::Client::Components::Editor::Toolbar

- **Descrizione:**
  La classe si occupa di fornire una toolbar per l'inserimento degli elementi del diagramma delle attività o del diagramma delle classi.

- **Utilizzo:**
  Ogni volta che viene selezionato un elemento esso viene inserito sul grafico. Nel caso dei connettori occorre selezionare, successivamente al connettore, i due elementi da collegare.
- **Metodi:**
  - *+addClasse(): void*
    Il metodo aggiunge una classe di nome "Classe" nell'area di disegno;
  - *+addAstratta(): void*
    Il metodo aggiunge una classe astratta di nome "ClasseAstratta" nell'area di disegno;
  - *+addInterfaccia(): void*
    Il metodo aggiunge un interfaccia di nome "Interfaccia" nell'area di disegno;
  - *+addGeneralizzazione(): void*
    Il metodo seleziona il tipo di connettore "Generalizzazione";
  - *+addImplementazione(): void*
    Il metodo seleziona il tipo di connettore "Implementazione";
  - *+addCommento(): void*
    Il metodo aggiunge un elemento di tipo "Commento" nell'area di disegno;
  - *+addAssociazione(): void*
    Il metodo seleziona il tipo di connettore "Associazione";
  - *+addConnettore(cellView: any): void*
    Il metodo serve, in caso venga selezionato un connettore, a selezionare i due elementi da collegare con il connettore selezionato da uno dei metodi precedenti.
  - **Parametri:**
    * *cellView: any*
      Elemento da selezionare per essere collegato con il connettore selezionato

3.5 SWEDesigner::Client::Components::Menu

3.5.1 Informazioni generali

3.5.2 Classi

3.5.2.1 SWEDesigner::Client::Components::Menu::File

- **Descrizione:**
- **Utilizzo:**
- **Metodi:**

3.5.2.2 SWEDesigner::Client::Components::Menu::Layer

- **Descrizione:**
- **Utilizzo:**
- **Metodi:**

3.5.2.3 SWEDesigner::Client::Components::Menu::Modifica

- **Descrizione:**
- **Utilizzo:**
- **Metodi:**

3.5.2.4 SWEDesigner::Client::Components::Menu::Profilo

- **Descrizione:**
- **Utilizzo:**
- **Metodi:**

3.5.2.5 SWEDesigner::Client::Components::Menu::Progetto

- **Descrizione:**

- **Utilizzo:**
- **Metodi:**

3.5.2.6 SWEDesigner::Client::Components::Menu::Template

- **Descrizione:**
- **Utilizzo:**
- **Metodi:**

4 Specifica Back-End

4.1 SWEDesigner::Server

4.1.1 Informazioni generali

- **Descrizione:**
  Questo package contiente tutte le componenti del server scritte in JavaScript.
- **Padre:** SWEDesigner
- **Package contenuti:**
  - Controller
    Questo package contiene al suo interno tutti i controller che implementano il pattern MVVM fornito da *Angular.js*$_G$. In particolare sono contenuti i Middleware e tutti i Servizi da essi utilizzati.
  - Model
    Questo package contiene tutte le classi utili per la creazione del database, la connessione ad esso e le relative interrogazioni.

4.1.2 Classi

4.1.2.1 SWEDesigner::Server::serverLoader

- **Descrizione:**
  Classe che consente il caricamento di tutte le componenti e gli elementi utili al primo avvio dell'applicazione
- **Utilizzo:**
  La classe viene utilizzata per il caricamento del server e di tutti i suoi elementi.
- **Metodi:**
  - *+ load(db: string, mR: string, mu: string, encr: string, cb: function): void*
    Si tratta della funzione principale che si occupa di chiamare i metodi load contenuti in tutte le altre classi.
  - **Parametri:**
    * *db: string*
      Il path del modulo che gestisce la connessione al database.

  - *mR: string*
    Il path del modulo che gestisce le query.
  - *mu: string*
    Il path del modulo che gestisce il servizio di parsing.
  - *encr: string*
    Il path del modulo che gestisce il servizio di encrypt.
  - *cb: function*taliano Callback che gestisce le rischieste asicnrone al database.

- **- loadCryptParam(db: string, cb: function): void**
  Si tratta della funzione utilizzata da load per la richiesta dei parametri crittografici al database.
- **Parametri:**

  - *db: string*
    Il path del modulo che gestisce la connessione al database.
  - *cb: function* Callback che gestisce le rischieste asicnrone al database.

4.2 SWEDesigner::Server::Model

4.2.1 Informazioni generali

- **Descrizione:**
  Questo package contiene tutte le classi e le funzionalità legate al database.
- **Padre:** SWEDesigner::Server

4.2.2 Classi

4.2.2.1 SWEDesigner::Server::Model::mongooseConnection

- **Descrizione:**
  Classe che si occupa della connessione al database e degli errori che ne possono derivare
- **Utilizzo:**
  La classe viene utilizzata per effettuare la connessione al database all'avvio dell'applicazione.
- **Metodi:**

– *+ conn() : void*
Si tratta della funzione che effettua la connessione al database e ne gestisce gli eventuali errori derivanti.

4.2.2.2 SWEDesiger::Server::Model::mongooseRequest

Tutte le query riguardanti l'aggiornamento e la cancellazione di dati dal database verranno trattate nella successiva versione di questo documento.

- **Descrizione:**
  Classe che si occupa di gestire tutte le query da e vero il database.
- **Utilizzo:**
  La classe viene utilizzata per tutte le richieste, inserimento e fetch, di dati dal e nel database.
- **Metodi:**

  – *+`ins_usr`(usr: Object, cb: function) : void*
  Si tratta della funzione che si occupa di inserire un utente all'interno del database.

  – **Parametri:**

    * *usr: Object*
      L'utente, in formato JSON, da inserire all'interno dello schema.
    * *cb: function*
      Callback che gestisce le richieste asincrone al database.

  – *+`ins_proj`(proj: Object, cb: function) : void*
  Si tratta della funzione che si occupa di inserire un progetto all'interno del database.

  – **Parametri:**

    * *proj: Object*
      Il progetto, in formato JSON, da inserire all'interno dello schema.
    * *cd: function*
      Callback che gestisce le richieste asincrone al database.

  – *+`ins_crypt_param`(k: string, i: string, cb: function) : void*
  Si tratta della funzione che si occupa di inserire una chiave crittografica al-

l'interno del database.

– **Parametri:**

  * *k: string*
    La chiave crittografica.
  * *i: string*
    Valore iv per la crittografia.
  * *cb: function*
    Callback che gestisce le richieste asincrone al database.

– *+load_all_proj(username: string, cb: function) : void*
Si tratta della funzione che si occupa di richiedere tutti i progetti di un dato utente.

– **Parametri:**

  * *username: string*
    Nome dell'utente di cui sono richiesti i progetti.
  * *cd: function*
    Callback che gestisce le richieste asincrone al database.

– *+load_key_crypt(cb: function) : void*
Si tratta della funzione che si occupa di richiedere l'unica chiave crittografica salvata nel database.

– **Parametri:**

  * *cb: function*
    Callback che gestisce le richieste asincrone al database.

– *+load_proj(projectName: string, cb: function) : void*
Si tratta della funzione che si occupa di cercare e ritornare un dato progetto.

– **Parametri:**

  * *projectName: string*
    Nome del progetto richiesto

    * *cb: function*
      Callback che gestisce le richieste asincrone al database.

- *+login(username: string, password: string, cb: function) : void*
  Si tratta della funzione che verifica che l'utente che cerca di loggare esiste all'interno del database.

- **Parametri:**

    * *username: string*
      L'username dell'utente che cerca di loggare.

    * *password: string*
      La password dell'utente che cerca di loggare.

    * *cb: function*
      Callback che gestisce le richieste asincrone al database.

- *+`forgot_mail`(username: string, cd: function)*
  Si tratta della funzione che restituisce la mail dell'utente dato.

- **Parametri:**

    * *username: string*
      Nome dell'utente

    * *cb: function*
      Callback che gestisce le richieste asincrone al database.

- *+`drop_schema`() : void*// Si tratta della funzione che elimina il database.

4.3 SWEDesigner::Server::Controller::Middleware

4.3.1 Informazioni generali

In questa versione del documento sono omesse le classi errHandler e Profile pociché verranno definite in seguito.

- **Descrizione:**
  In questo package sono definite tutte le componenti middleware del server scritte in JavaScript.
- **Padre:** SWEDesigner::Server::Controller

4.3.2 Classi

4.3.2.1 SWEDesigner::Server::Controller::Middleware::midLoader

- **Descrizione:**
  La classe contenente i metodi di caricamento dei servizi utilizzati dalle componenti middleware
- **Utilizzo:**
  La classe viene utilizzata all'avvio dell'applicazione per caricare tutto cià che serve per il funzionamento del middleware.
- **Metodi:**
  - *+load() : void*
    La funziona carica il servizio di parsing

4.3.2.2 SWEDesigner::Server::Controller::Middleware::Parse

- **Descrizione:**
  La classe si occupa di gestire il caricamento del template e di richiamare il servizio di parsing
- **Utilizzo:**
  La classe viene utilizzata sia per il caricamento del template all'avvio dell'applicazione, sia per richiamare il servizio di parsing quando il client lo richiede.
- **Metodi:**
  - *+load() : void*
    La funzione si occupa di ripulire la cache, compilare il template e caricarlo in cache.
  - *+parse(template: Object, myMu: Object, cb: function) : void*
    La funzione si occupa di richiamare la funzione di parsing del relativo servizio
  - **Parametri:**
    - *template: Object*
      Il template precompilato da Moustache.
    - *myMu: Object*
      L'oggetto JSON di cui è necessario il parsing.
    - *cb: function*
      Callback che gestisce la chiamata asincrona al modulo di Moustahce.

- *+getTemplate() : string*
  La funzione ritorna il percorso in cui è contenuto il template, compilato o meno.

4.3.2.3 SWEDesigner::Server::Controller:Middleware::Encrypt

- **Descrizione:**
  La classe si occupa di gestire le funzionalità del servizio di encrypt.
- **Utilizzo:**
  La classe viene utilizzata per chiamare le funzioni di encrypt del relativo servizio.
- **Metodi:**

  - *+encrypt(myFile: Object, key: string, iv: string) : Object*
    La funzione si occupa di richiamare la funzione di encrypt del relativo servizio e ritorna il file crittato correttamente.
  - **Parametri:**

    * *myFile: Object*
      Oggetto JSON da crittare
    * *key: string*
      Chiave crittografica
    * *iv: string*
      IV necessario per la crittografia in AES
  - *+decrypt(myFile: Object, key: string, iv: string) : Object*
    La funzione si occupa di richiamare la funzione di decrypt del relativo servizio e ritorna il JSON decriptato.
  - **Parametri:**

    * *myFile: Object*
      Oggetto JSON da crittare
    * *key: string*
      Chiave crittografica
    * *iv: string*
      IV necessario per la crittografia in AES
  - *+getKey() : void*
    La funzione si occupa di richiamare la funzione di generazione della chiave crittografica del relativo servizio.

- *+getI() : void*
  La funzione si occupa di richiamare la funzione di generazione del valore iv per la crittografia del relativo servizio.

4.4 SWEDesigner::Server::Controller::Services

4.4.1 Informazioni generali

In tale versione del documento non sarà trattato il servizio di Profile poiché verrà trattato nelle versioni successive.

- **Descrizione:**
  Questo package contiene tutti i servizi utilizzati dal middleware del server scritti in JavaScript.
- **Padre:** SWEDesigner::Server::Controller

4.4.2 Classi

4.4.2.1 SWEDesigner::Server::Controller::Services::parseService

- **Descrizione:**
  La classe si occupa di renderizzare il template pre-compilato e generare, così, un file scritto in Java.
- **Utilizzo:**
  La classe viene utilizzata ogni volta che il client richiede la generazione di codice Java a partire dai diagrammi UML disegnati.
- **Metodi:**

  - *+parsing(template: string, myMu: Object, cb: function) : void*
    La funzione renderizza il template pre-compilato in fase di avvio dell'applicazione generando, a fronte dell'oggetto JSON inviato, un file in Java.
  - **Parametri:**

    * *template: string*
      Il percorso del template precompilato da Moustache.
    * *myMu: Object*
      L'oggetto JSON di cui è necessario il parsing.
    * *cb: function*
      Callback che gestisce la chiamata asincrona al modulo di Moustahce.

4.4.2.2 SWEDesigner::Server::Controller::Services::encryptService

- **Descrizione:**
  La classe si occupa di tutti i servizi òegati alla crittografia.
- **Utilizzo:**
  La classe viene utilizzata per generare le chiavi crittografiche da salvare nel database al primo avvio, qualora queste non esistessero, e di realizzare tutti i servizi legati alla crittografia, quindi encrypt e decrypt.
- **Metodi:**
  - *+encrypt(myFile: Object, key: string, iv: string) : Object*
    La funzione si occupa di criptare il file in arrivo mediante codifica AES utilizzando gli algoritmi di Forge.
  - **Parametri:**
    - *myFile: Object*
      Oggetto JSON da crittare
    - *key: string*
      Chiave crittografica
    - *iv: string*
      IV necessario per la crittografia in AES
  - *+decrypt(myFile: Object, key: string, iv: string) : Object*
    La funzione si occupa di decriptare il file in arrivo mediante gi algritmi di Forge.
  - **Parametri:**
    - *myFile: Object*
      Oggetto JSON da crittare
    - *key: string*
      Chiave crittografica
    - *iv: string*
      IV necessario per la crittografia in AES
  - *+getKey() : string*
    La funzione genera, tramite Forge, una chiave crittografica e la ritorna.
  - *+getIv() : string*
    La funzione genera, tramite Forge, un gruppo di iv e lo ritorna.

5 Diagrammi di sequenza

6 Tracciamento

6.1 Tracciamento Classi-Requisiti

6.2 Tracciamento Requisiti-Classi

6.3 Tracciamento Componenti-Requisiti

6.4 Tracciamento Requisiti-Componenti