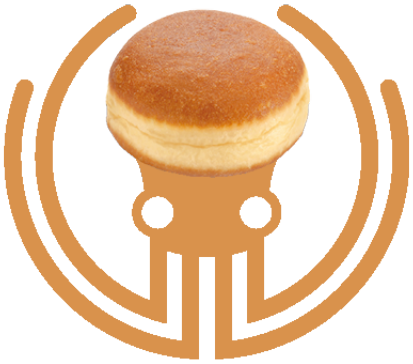

Manuale sviluppatore

Gruppo GitKraffen — ***Progetto IronWorks***
gitKraffen.swe16@gmail.com

Versione	1.0.0
Redazione	Elia Rebellato Daniel Rossi Federica Ramina
Verifica	Iris Balaj
Approvazione	Mattia Poloni
Uso	Esterno
Distribuzione	Prof. Tullio Vardanega Prof. Riccardo Cardin Gruppo GitKraffen

Descrizione

Questo documento ha lo scopo di fornire a coloro che eventualmente dovranno arricchire con nuove funzionalità il prodotto IronWorks, i mezzi e le linee guida con cui procedere.

Registro delle modifiche

Versione	Ruolo	Nominativo	Descrizione	Data
1.0.0	Responsabile	Mattia Poloni	Approvazione	02-06-2018
0.1.0	Verificatore	Iris Balaj	Verifica	02-06-2018
0.0.6	Programmatore	Daniel Rossi	Stesura sezioni Implementazione	30-05-2018
0.0.5	Programmatore	Elia Rebellato	Stesura sezione Architettura	29-05-2018
0.0.4	Programmatore	Elia Rebellato	Stesura sezione Ambiente di sviluppo	28-05-2018
0.0.3	Programmatore	Mattia Poloni	Stesura sezione Tecnologie	27-05-2018
0.0.2	Programmatore	Mattia Poloni	Stesura sezione Requisiti di sistema	26-05-2018
0.0.1	Programmatore	Daniel Rossi	Stesura sezione introduzione	26-05-2018
0.0.0	Programmatore	Daniel Rossi	Creazione template documento	26-05-2018

Indice

Elenco delle figure

Elenco delle tabelle

1 Introduzione

1.1 Scopo del documento

Il documento ha la finalità di fornire mezzi e linee guida a coloro volessero arricchire con nuove funzionalità il software *Ironworks*, oltre che fornire una indicazioni alla installazione dell'ambiente di sviluppo idoneo, i requisiti di accesso che dovranno essere soddisfatti per poter utilizzare l'applicazione, le librerie necessarie, l'achitettura del sistema e i *framework*$_G$ necessari allo sviluppo dell'applicazione in modo completo, questo con l'obiettivo di aiutare futuri sviluppatori nelle attività di estensione delle funzionalità del prodotto.

Questo documento è di tipo incrementale e rappresenta soltanto una bozza rispetto a ciò che mira a diventare, dato che il prodotto non è ancora completo il documento prevede alcune sezioni attualmente non presenti che verranno inserite successivamente. Le sezioni che verranno inserite successivamente sono le seguenti:

- **Avvio server**: liee guida sull'attivazione del server che ospiterà il software *Ironworks*
- **Estensioni**: linee guida con come eseguire una modifica e successivamente verificarne la correttezza.

  Verranno inserite anche le seguenti appendici:
- **Contributi**: linee guida su come contribuire al progetto *Ironworks*;
- **Licenza**: licenza con cui è verrà rilasciato il software *Ironworks*.

1.2 Scopo del prodotto

Lo scopo del *prodotto*$_G$ è creare un software, sotto forma di *applicazione web*$_G$ che permetta la creazione di diagrammi di robustezza, in grado di generare automaticamente il relativo codice *Java*$_G$, *SQL*$_G$ e *XML*$_G$.
L'utente, interagendo con il sistema, sarà in grado di:

- realizzare un diagramma di robustezza a proprio piacimento;
- generare codice *Java*, *SQL* e *XML* a partire dal diagramma di robustezza.

L'*editor*$_G$ sarà fruibile dall'utente attraverso un *browser*$_G$ idoneo all'utilizzo delle tecnologie *HTML5*$_G$, *CSS3*$_G$ e *Javascript*$_G$ versione 1.7.

1.3 Informazioni utili

Un requisito fondamentale è la conoscenza dei linguaggi di programmazione *HTML5*$_G$, *CSS3*$_G$ e *Javascript*$_G$ versione 1.7., oltre che le librerie *Javascript*:

- *BackBone.js*
- *Node.js*
- *Joint.js*

Per completezza, viene riportato in appendice alla fine del documento un glossario comprensivo di termini tecnici o riguardanti particolari funzionalità di *Ironworks*. Per identificare i termini presenti nel glossario, la loro prima occorrenza all'interno del documento è riportata in corsivo e marcata con una G al pedice.

1.4 Riferimenti informativi

- **Documentazione Backbone.js:**
  http://backbonejs.org/;

  Documentazione ufficiale della libreria *Backbone.js* utilizzata per definire l'achitettura software.

- **Documentazione Joint.js:**
  https://resources.jointjs.com/docs/jointjs/v2.1/joint.html;

  Documentazione ufficiale per la libreria *Joint.js*, utilizzata per il disegnatore diagrammi.

- **Documentazione Node.js:**
  https://nodejs.org/it/docs/.

  Documentazione ufficiale per la libreria *Node.js*, utilizzata per il lato server.

- **Documentazione HTML5, CSS, Javascript:**
  https://www.w3.org/.

  Documentazione ufficiale per i linguaggi *HTML5*, *CSS* e *Javascript*.

2 Requisiti di sistema

2.1 Browser supportati

Di seguito vengono fornite una lista dei browser che supporterrano l'applicazione *Ironworks* riportando anche la versione minima:

- *Google Chrome*: v_65.x;
- *Mozilla Firefox*: v_52.x;

Affinché l'esecuzione dell'applicazione avvenga correttamente è necessario che Javascript versione 1.7 sia attivato.

3 Tecnologie impiegate

Vengono di seguito riportate le tecnologie utilizzate per lo sviluppo e la verifica dell'applicazione.

3.1 Librerie utilizzate da Ironworks

3.1.1 Node.js

Libreria Javascript utilizzata per creare il lato server dell'applicazione.

3.1.2 Joint.js

Libreria Javascript utilizzata per creare e gestire il disegnatore di diagrammi.

3.1.3 Backbone.js

Libreria Javascript utilizzata come pattern architetturale poiché in rapporto privilegiato con Joint.js

3.1.4 Underscore.js

Libreria Javascript utilizzata per le sue funzionalità di manipolazione del DOM_G.

3.1.5

Libreria Javascript che racchiude funzionalità utili alla configurazione rapida di un server.

3.2 Interfaccia grafica

Per progettare l'interfaccia grafica si è ricorsi all'*IDE* Webstorm, la parte grafica è stata realizzata attraverso template *HTML5* con foglio di stile *CSS3*.

4 Ambiente di sviluppo

Vengono di seguito illustrate le modalità di installazione dell'*IDE*$_G$ consigliato e l'importazione del progetto nell'*IDE*.

4.1 Installazione e configurazione Ironworks

Il software *Ironworks* è reperibile su GitHub al seguente link: https://github.com/GitKraffen/ironworks-app

Una volta collegati al seguenti link seguire la seguente procedura:

1. cliccare sul pulsante verde in alto a destra *Clone or download*
2. premere il pulsante *Download Zip*

4.2 IDE suggerito

Il progetto *Ironworks* non è necessariamente legato ad alcun IDE, tuttavia si consiglia l'uso di *WebStorm*$_G$. Le librerie Javascript sono infatti utilizzate all'interno del prodotto per la realizzazione dell'interfaccia grafica, e l'adozione dei succitati IDE ne agevola notevolmente l'impiego nell'ottica di modifiche e/o estensioni alla stessa. Viene di seguito illustrata la procedura per l'installazione dell'IDE e l'importazione del progetto.

4.2.1 Installazione WebStorm

Il pacchetto d'installazione di WebStorm è reperibile gratuitamente per trenta giorni per progetti open source al seguente link: https://www.jetbrains.com/webstorm/download

Una volta eseguito l'installer, previa configurazione dei pacchetti d'installazione, l'IDE sarà installato sulla macchina desiderata. Ulteriori informazioni sull'installazione e procedure alternative sono reperibili al seguente link: https://www.jetbrains.com/help/webstorm/meet-webstorm.html

4.2.2 Importare il progetto

Per importare il progetto Ironworks all'interno dell'IDE, è necessario seguire la seguente procedura:

- Aprire l'IDE installato seguendo le indicazioni riportate nella precedente sezione;
- Dalla barra dei menù selezionare "File / Apri progetto" (ctrl + o);
- Selezionare la cartella IronWorks e verrà così aperto il progetto;

5 Architettura generale di Ironworks

5.1 Client-side

L'architettura generale del prodotto segue il pattern Model-View-Control (MVC). Questo pattern è basato su tre componenti principali:

- **Model**: un'implementazione del modello del dominio dei dati;
- **View**: la struttura, il layout e l'aspetto di ciò che l'utente visualizza a schermo;
- **Controller**: un'astrazione della view che espone proprietà pubbliche e comandi.

Esso permette tra le altre cose un totale disaccoppiamento tra logica di business e presentazione. L'architettura scelta, a livello di implementazione, risulta essere concettualmente diversa a causa dell'utilizzo della libreria *Backbone*. Infatti abbiamo la view che viene implementata con HTML ed CSS e manipolata con $template_G$ grazie alla libreria $Underscore_G$. La componente Controller invece verrà implementata dalle view di *Backbone*, esse infatti gestiranno la comunicazione tra le views ed i models. I models invece verranno gestiti dalle collection e non saranno persistenti, il loro ciclo di vita è limitato a quello dell'editor.
Un *design pattern* utilizzato in questa architettura è l'$observer_G$. La libreria *backbone* implementa una funzionalità chiamata *events* che svolge il ruolo di observer ed è integrata in ogni classe base di esso.
I seguenti diagrammi illustrano sinteticamente la struttura del software, con livello di dettaglio crescente.

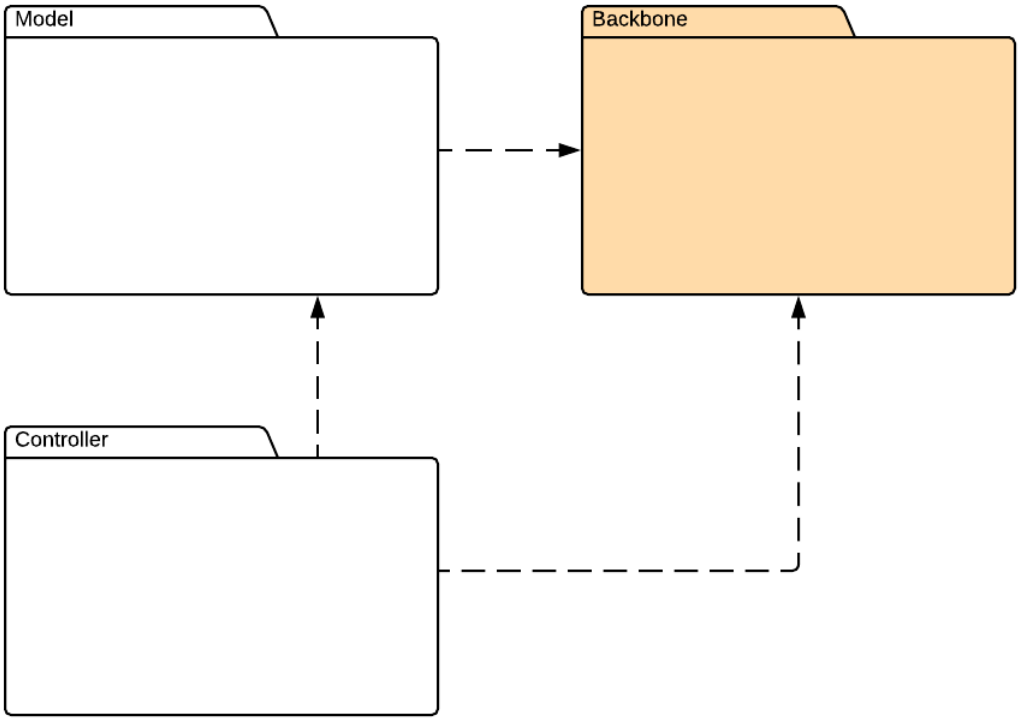

Figura 1: Diagramma dei package lato client

5.2 Server-side

L'architettura dal lato server invece si basa su due pattern:

5.2.1 Architettura a layer

Questo pattern architetturale prevede due componenti:

- *presentation*: layer che riceve le richieste dell'utente;
- *application*: layer che si occupa della logica funzionale.

Questo pattern architetturale è spesso implementato nella sua versione *three-tier*$_G$, in cui esiste anche una componente che si occupa della persistenza dei dati su un database. Non avendo dati da far persistere sul server abbiamo di conseguenza deciso di non implementarlo.

5.2.2 Template method

Questo design pattern comportamentale consente di definire la struttura di un algoritmo lasciando alle sottoclassi il compito di implementare alcuni passi in base alle esigenze, permettendo di evitare la duplicazione del codice. La sua struttura prevede:

- *AbstractClass*: definisce le operazioni primitive astratte e implementa il metodo contenente l'algoritmo base;
- *ConcreteClass*: implementa i metodi astratti ereditati, specificandone il comportamento secondo le necessità.

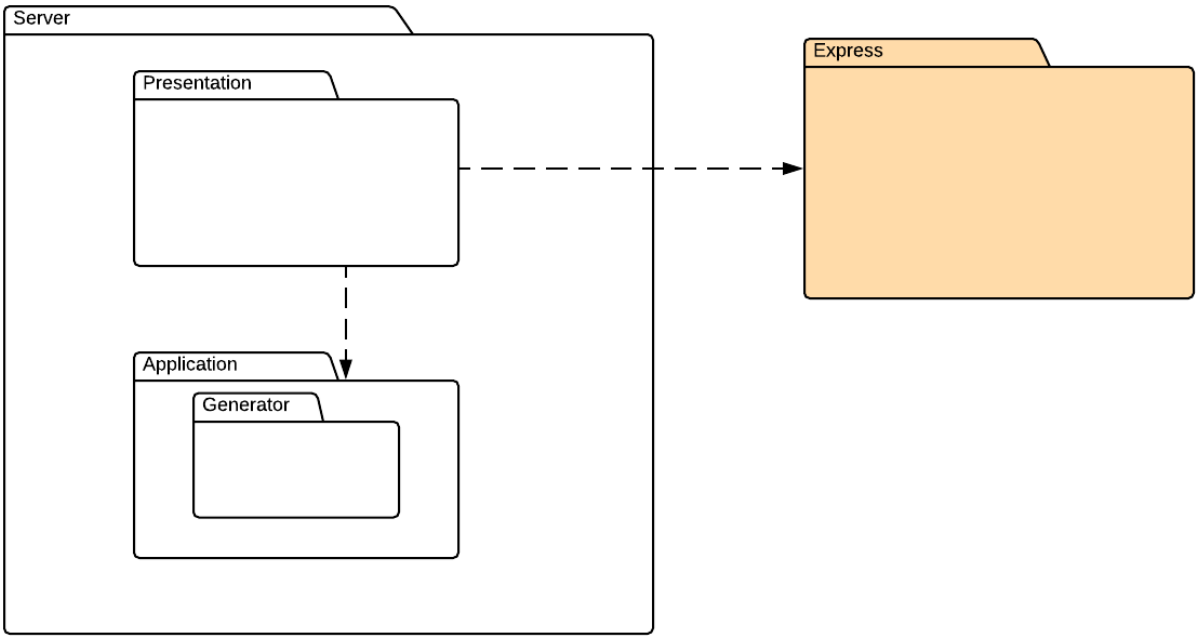

Figura 2: Diagramma dei package lato server

6 Implementazione MVC - Client side

Vengono di seguito illustrate le implementazioni per le componenti del pattern Model-View-Control in relazione all'architettura di IronWorks lato client. Per ogni componente, vengono illustrati:

- **Contestualizzazione**: spiegazione generale dell'architettura del componente all'interno del sistema;
- **Diagramma delle classi**: diagramma generale delle classi per il componente;
- **Dettaglio delle classi**: elencazione delle responsabilità di ogni singola classe;

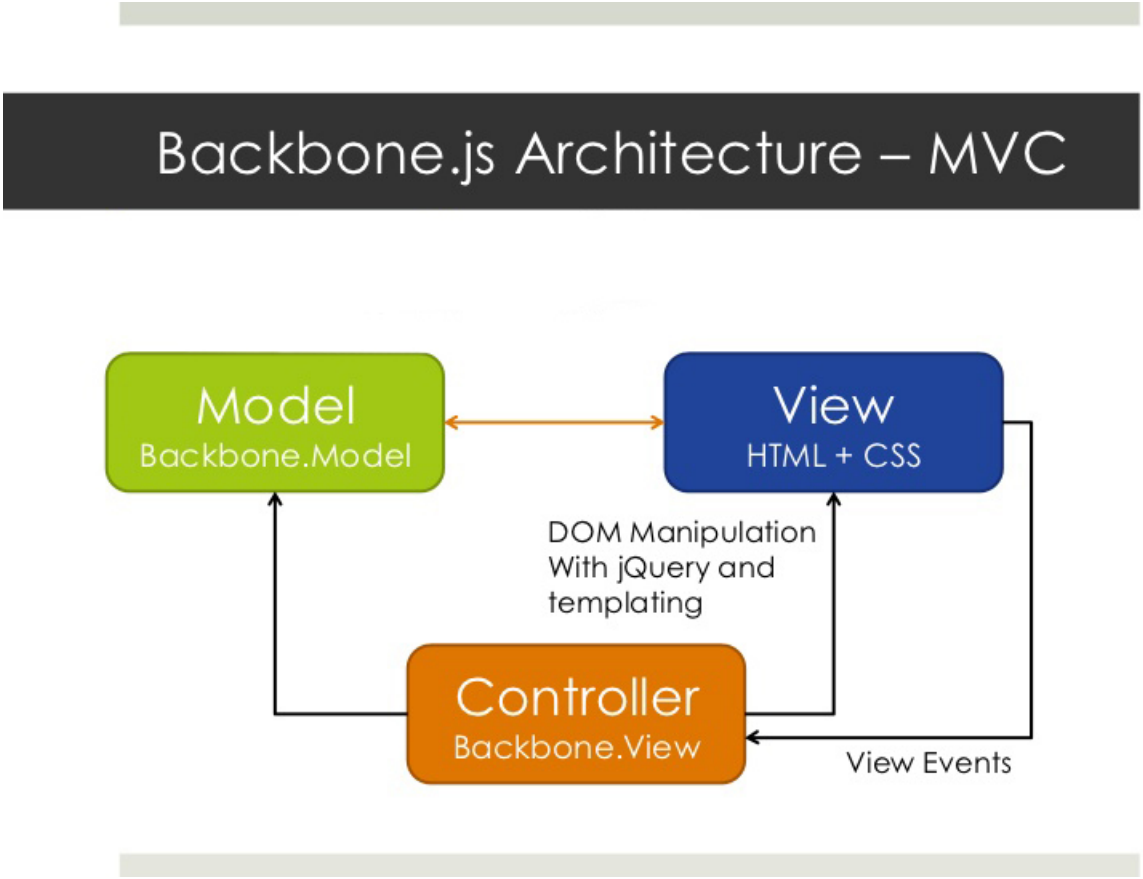

Figura 3: Struttura generale - Backbone MVC

6.1 View

La view, consiste in semplice HTML e CSS che verrà successivamente gestito dalla libreria *javascript underscore*. Verrà infatti convertito in un $template_G$ utilizzato poi dal relativo *controller*. In questo caso non è necessario definire nel dettaglio questa componente in quanto non ha una vera e propria architettura.

6.2 Model

6.2.1 Contestualizzazione

Durante la progettazione si è deciso di non rendere i dati persistenti in un database, in quanto non necessario per lo scopo del progetto. La componente model quindi non avrà alcuna necessità di sincronizzazione tramite le funzionalità *Backbone REST Sync*$_G$. Questa componente è divisa in due tipi di classi:

- *Collection*: una raccolta ordinata di *model*s, per facilitarne la gestione;
- *Model*: contenente i dati utili al sistema.

6.2.2 Diagramma delle classi

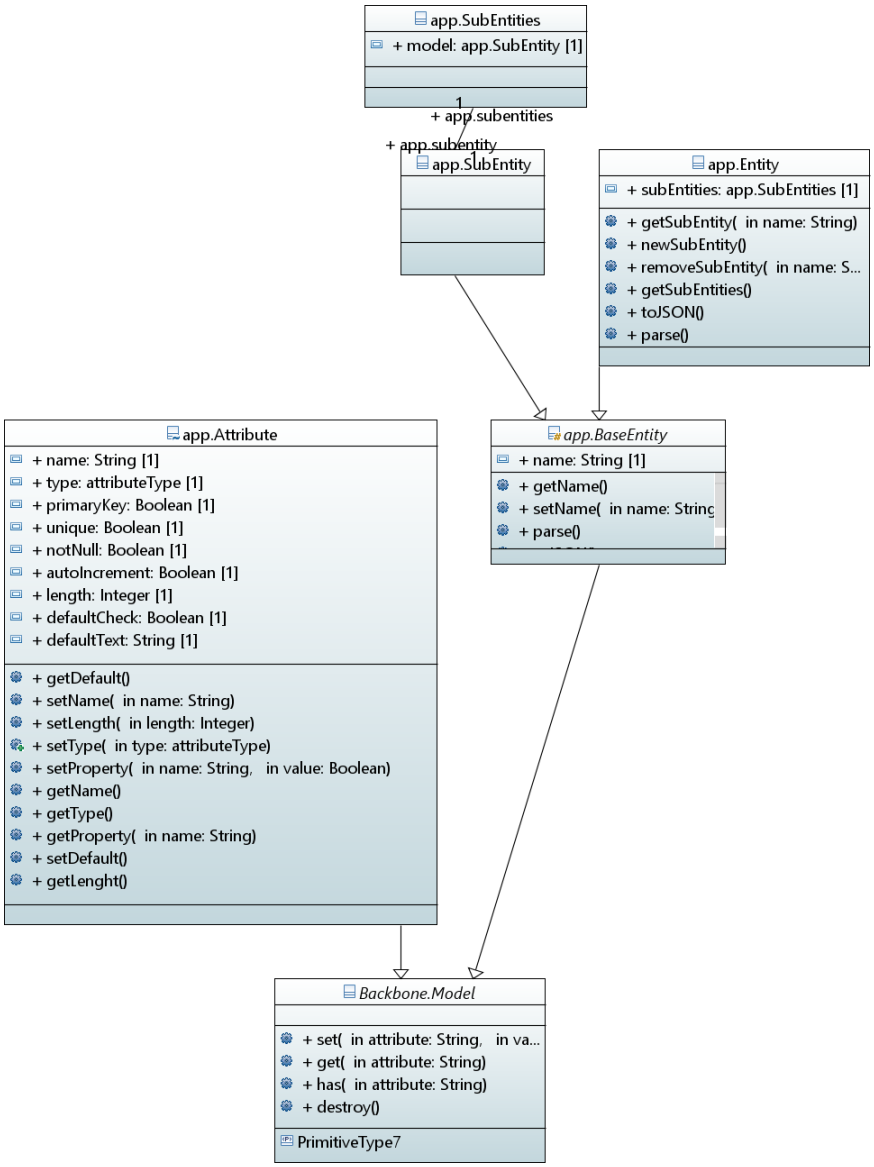

Figura 4: Diagrammi delle classi Model

6.2.3 Dettaglio delle classi

Di seguito verranno illustrate le classi appartenenti alla componente *model* divise nelle due tipologie.

- *Collection*
  - *Attributes*: contiene una lista di *attributo*, incapsulati dal *model Attribute*;
  - *Entities*: contiene una lista di *entità*, incapsulate dal *model Entity*;
  - *SubEntities*: contiene una lista di *sotto-entità*, incapsulate dal *model sub-Entity*.
- *Model*
  - *Attribute*: incapsula i dati relativi ad un *attributo* di una *entità*;
  - *Entities*: incapsula i dati relativi ad una *entità*;
  - *SubEntities*: incapsula i dati relativi ad una *sotto-entità*.

6.3 Controller

6.3.1 Contestualizzazione

Durante la progettazione si è studiato come, utilizzando la libreria *backbone*, si implementasse la classe *view* rendendola un controller. Le classi progettate fungeranno quindi da tramite tra i *model* ed i *controller*. Saranno inizializzati i *template* grafici utilizzando i dati contenuti nei *model* e gestiti gli eventi e le richieste dell'utente. Si è progettato un sistema modulare, permettendo quindi di distribuire i ruoli a sotto-controller avendo quindi una organizzazione più pulita.

6.3.2 Diagramma delle classi

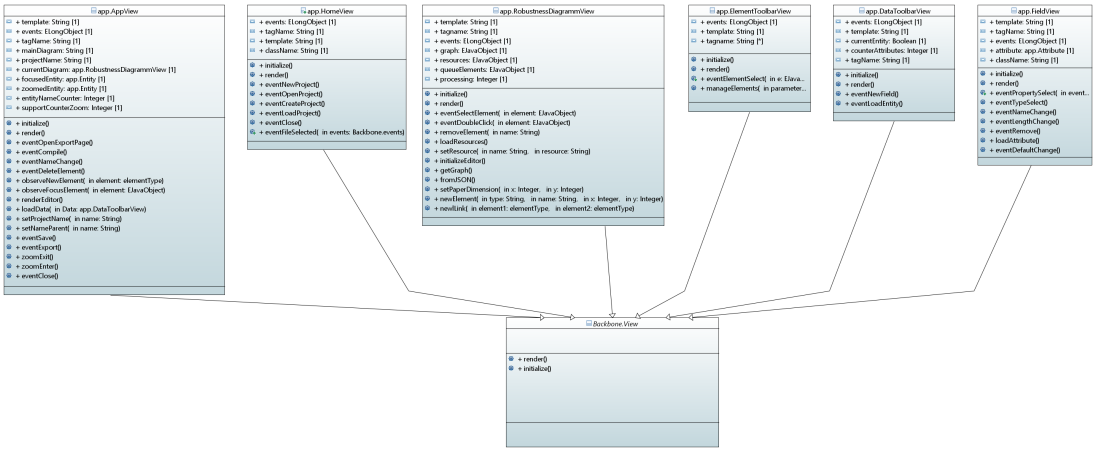

Figura 5: Diagrammi delle classi View

6.3.3 Dettaglio delle classi

Di seguito verranno illustrate le classi appartenenti alla componente *controller*.

- *AppView*: gestisce la schermata principale dell'editor e le sue componenti. In particolare si occupa di gestire i sotto-controller *ElementToolbarView*, *DataToolbarView* e *RobustnessDiagramView*, inizializzandoli e facendoli interagire tra di loro;
- *ElementToolbarView*: gestisce la vista della toolbar ove sono disponibili gli elementi da inserire, cattura le richieste di nuovo elemento e manda il segnale relativo;
- *DataToolbarView*: gestisce la *collection Attributes*, creando su richiesta i sotto-controller *FieldView*;
- *FieldView*: gestisce il *model Attribute*, attraverso la relativa vista, in base agli input dell'utente;
- *RobustnessDiagramView*: gestisce il diagramma di robustezza grazie alla libreria *JointJS*, coglie e gestisce gli input dell'utente;
- *HomeView*: Gestisce la schermata principale consentendo all'utente di creare o importare un progetto.

7 Implementazione architettura a layer - Server side

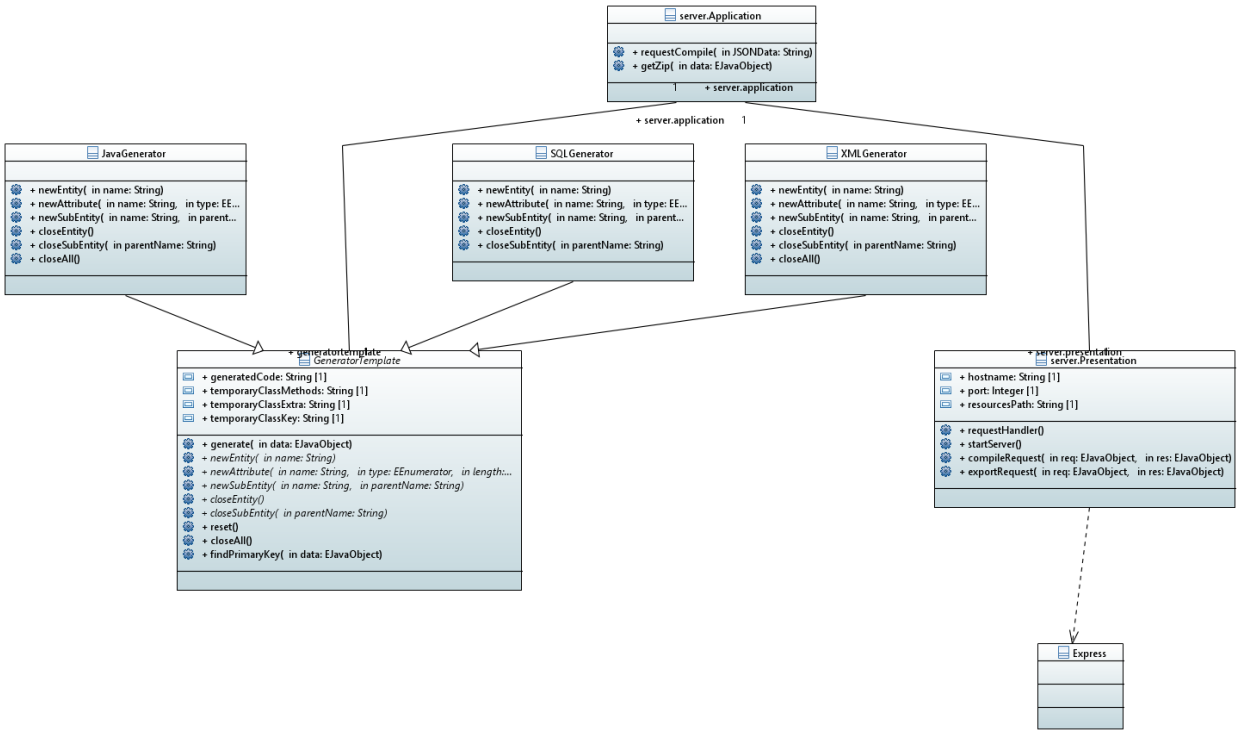

Figura 6: Diagramma delle classi lato server

Vengono di seguito illustrate le implementazioni per le componenti dell'architettura di IronWorks lato server. Per ogni componente, vengono illustrati:

- **Contestualizzazione**: spiegazione generale dell'architettura del componente all'interno del sistema;
- **Dettaglio delle classi**: elencazione delle responsabilità di ogni singola classe.

7.1 Presentation

7.1.1 Contestualizzazione

Questo layer gestirà le interazioni con l'utente. In particolare lavorerà usando il modulo *express* di *node.js* per gestire le richieste al server chiamando i metodi dell'application layer.

7.1.2 Dettaglio delle classi

- *Presentation*: inizializza express per la gestione delle richieste al server e avvia quest'ultimo.

7.2 Application

7.2.1 Contestualizzazione

Questo layer verrà usato per l'elaborazione di dati tramite l'utilizzo dei generator sucessivamente illustrati.

7.2.2 Dettaglio delle classi

- *Application*: in base alla richiesta si occuperà di chiamare i moduli necessari per la manipolazioni dei dati forniti;
- *GeneratorTemplate* e sottoclassi: si occupano di generare il codice per i file (Java, SQL e XML).

7.3 Generator

Questa classe è contenuta nel tier Application e si occupa di generare il codice Java, SQL e XML implementando il design pattern Template Method. Esiste una classe base astratta *GeneratorTemplate* che contiene la parte dell'algoritmo di generazione del codice comune a tutte le sottoclassi, e i metodi astratti che si occuperanno di creare le entità, attributi, sotto-entita... Questi metodi sono ridefiniti nelle sottoclassi *JavaGenerator*, *SQLGenerator* e *XMLGenerator*.

8 Glossario

- **Attributo**: Gli attributi SQL consentono di creare situazioni che controllano le informazioni dettagliate relative ad un attributo;
- **Collection**: oggetto raccolta utilizzato per immagazzinare informazioni;
- **Controller**: riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti;
- **CSS3**: Cascaded Style Sheet, è un linguaggio che permette di defnire tutte le proprietà di stile e formattazione di una pagina web in maniera modulare, tenendo questa parte della progettazione di un sito separata da quella relativa al contenuto.
- **Design pattern**: soluzione progettuale generale per la risoluzione di un problema ricorrente;
- **DOM**: forma di rappresentazione dei documenti strutturati come modello orientato agli oggetti.
- **Entità**: insieme di attributi raccolti sotto un concettto comune, come potrebbero essere gli attributi nome, cognome sotto il concetto comune persona;
- **HTML5**: Linguaggio di markup per la strutturazione delle pagine web, pubblicato come W3C Recommendation da ottobre 2014;
- **IDE**: ambiente di sviluppo integrato per la realizzazione di programmi per sistemi informatici;
- **Javascript**: Linguaggio di Scripting lato-client, che viene interpretato dal browser.
- **Model**: fornisce i metodi per accedere ai dati utili all'applicazione;
- **Observer**: design pattern utilizzato per gestire gli eventi;

- **Sotto-entità**: sotto insieme di attributi che formano una entità di livello più basso rispetto all'enitità principale;
- **Template**: in informatica, modello predefinito che consente di creare o inserire contenuti di diverso tipo in un documento o in una pagina web;
- **Three-tier**: l'espressione architettura three-tier ("a tre strati") indica una particolare architettura software di tipo multi-tier per l'esecuzione di un'applicazione web che prevede la suddivisione dell'applicazione in tre diversi moduli o strati dedicati rispettivamente alla interfaccia utente, alla logica funzionale (business logic) e alla gestione dei dati persistenti. Tale architettura va tipicamente a mappare a livello fisico-infrastrutturale quella del sistema informatico ospitante l'applicazione da eseguire;
- **View**: visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti;
- **WebStorm**: IDE consigliato per la realizzazione di applicazioni web;