IronWorks

Utility per la Costruzione di Software Robusto

swearoncode@gmail.com

Manuale Sviluppatore

Versione	2.0.0
Redattori	Anna Poletti, Antonio Moz, Sharon Della Libera
Verificatori	Antonio Moz
Responsabili	Mirko Gibin
Uso	Esterno
Distribuzione	Gruppo Swear on Code
	Prof. Tullio Vardanega
	Prof. Riccardo Cardin
	Gregorio Piccoli, Zucchetti S.p.A.

Descrizione

Questo documento contiene il Manuale Sviluppatore per il progetto **IronWorks** del gruppo Swear on Code.

Registro delle modifiche

Versione	Data	Autori	Ruolo	Descrizione
2.0.0	2018/08/22	Mirko Gibin	Responsabile	Approvazione
1.1.0	2018/08/10	Antonio Moz	Verificatore	Verifica
1.0.7	2018/08/09	Sharon Della Libera	Programmatore	Aggiornata figura 1
1.0.6	2018/08/08	Anna Poletti	Programmatore	Aggiunti termini in §A
1.0.5	2018/08/08	Anna Poletti	Programmatore	Aggiunta libreria Spectrum in §1.4.1, §2, §5.1.3
1.0.4	2018/08/06	Sharon Della Libera	Programmatore	Modificate §8.4, §8.5, §9.4, §9.5
1.0.3	2018/08/04	Sharon Della Libera	Programmatore	Aggiunta §10
1.0.2	2018/08/03	Anna Poletti	Programmatore	Migliorata §6.1 ed Ampliata §6.2.2
1.0.1	2018/08/02	Sharon Della Libera	Programmatore	Ampliate §7.1.1, §7.2
1.0.0	2018/07/12	Anna Poletti	Responsabile	Approvazione
0.1.0	2018/07/12	Stefano Nordio	Verificatore	Verifica
0.0.9	2018/07/11	Sharon Della Libera	Programmatore	Aggiunti termini in §A
0.0.8	2018/07/11	Antonio Moz	Programmatore	Stesura §A
0.0.7	2018/07/10	Antonio Moz	Programmatore	Stesura §8.1, §8.2, §8.3, §9.1, §9.2, §9.3
0.0.6	2018/07/10	Sharon Della Libera	Programmatore	Stesura §8.4, §8.5, §9.4, §9.5
0.0.5	2018/07/09	Antonio Moz	Programmatore	Stesura §5, §6
0.0.4	2018/07/09	Sharon Della Libera	Programmatore	Stesura §7
0.0.3	2018/07/09	Sharon Della Libera	Programmatore	Modifica §2.1, §2.2 e Stesura §3, §4
0.0.2	2018/07/06	Antonio Moz	Programmatore	Stesura §1, §2.3, §2.4, §2.5, §2.6, §2.7

0.0.1	2018/07/05	Antonio Moz	Programmatore	Creazione del documento

Tabella 1: Storico versioni del documento

Indice

1 Introduzione **1**
1.1 Scopo del Documento . . . 1
1.2 Scopo del Prodotto . . . 1
1.3 Ambiguità . . . 1
1.4 Riferimenti . . . 1
1.4.1 Riferimenti Informativi . . . 1

2 Tecnologie Utilizzate **3**
2.1 Node.js . . . 3
2.2 Express.js . . . 3
2.3 HTML5 . . . 3
2.4 CSS3 . . . 3
2.5 ECMAScript . . . 3
2.6 jQuery . . . 4
2.7 JointJS . . . 4
2.8 Spectrum . . . 4

3 Requisiti di Sistema **5**
3.1 Dispositivi Supportati . . . 5
3.2 Browser Supportati . . . 5
3.3 Requisiti Hardware . . . 5

4 Configurazione Ambiente di Lavoro **6**
4.1 Accesso all'Applicazione . . . 6
4.1.1 Utilizzo in Locale . . . 6
4.1.2 Utilizzo Tramite la Rete . . . 6
4.2 Installazione in Locale dell'Applicazione . . . 6
4.2.1 Pre-Requisiti . . . 6
4.2.2 Installare le Dipendenze . . . 6
4.2.3 Avvio di IronWorks . . . 6

5 Architettura **7**
5.1 Architettura ad Alto Livello . . . 7
5.1.1 Descrizione . . . 8
5.1.2 Package Contenuti . . . 8
5.1.3 Librerie e Framework Esterni . . . 9

6 `Front-End` **10**
6.1 `Front-End::View` . . . 10
6.2 `Front-End::Controller` . . . 10
6.2.1 `Front-End::Controller::FirstPagesController` . . . 11
6.2.2 `Front-End::Controller::EditorController` . . . 11
6.3 `Front-End::Model` . . . 13

6.3.1 Front-End::Model::FirstPages 13
6.3.1.1 Front-End::Model::FirstPages::HomePage 14
6.3.1.2 Front-End::Model::FirstPages::NewProject 14
6.3.2 Front-End::Model::Editor 15
6.3.2.1 Front-End::Model::Editor::Graph 16
6.3.2.2 Front-End::Model::Editor::Element 17

7 Back-End **19**
7.1 Back-End::PresentationTier 19
7.1.1 Back-End::PresentationTier::Middleware 19
7.1.2 Back-End::PresentationTier::PresentationController 20
7.2 Back-End::ApplicationTier 21
7.2.1 Back-End::ApplicationTier::ApplicationController 21
7.2.2 Back-End::ApplicationTier::Components 22
7.2.2.1 Back-End::ApplicationTier::Components::Factory . 23
7.2.2.2 Back-End::ApplicationTier::Components::Parser . . 25

8 Estensione Funzionalità **27**
8.1 Aggiunta di un Foglio per un Nuovo Diagramma 27
8.2 Aggiunta di Altri Elementi per il Diagramma 27
8.3 Aggiunta di Attributi ad un Elemento 27
8.4 Creazione del Codice in Altri Linguaggi 27
8.5 Generazione Codice da un Oggetto JSON Diverso 28

9 Estensione Codice **29**
9.1 Aggiunta di un Foglio per un Nuovo Diagramma 29
9.2 Aggiunta di Altri Elementi per il Diagramma 29
9.3 Aggiunta di Attributi ad un Elemento 29
9.4 Creazione del Codice in Altri Linguaggi 30
9.5 Generazione Codice da un Oggetto JSON Diverso 30

10 Utilizzo del Codice Generato **32**
10.1 Contenuto del File Zip Scaricato 32
10.2 Esecuzione dello Script Sql 32
10.3 Gestione del Database Tramite Driver JDBC 32
10.4 Gestione del Database Tramite Framework Hibernate ORM 33
10.5 Compilazione ed Esecuzione 33
10.6 Contenuto Tabelle SQL 34
10.7 Contenuto Classi Java 35
10.8 Contenuto Classi DAO 35
10.9 Ulteriori Informazioni 36

A Glossario **37**

Elenco delle figure

1 Diagramma dei package `Front-End` 7
2 Diagramma dei package `Back-End` 8
3 Diagramma delle Classi - `Graph` 16
4 Diagramma delle Classi - `Element` 17
5 Diagramma delle Classi - `Middleware` 19
6 Diagramma delle Classi - `Factory` 23
7 Diagramma delle Classi - `Parser` 25

1 Introduzione

1.1 Scopo del Documento

Questo documento ha lo scopo di fornire informazioni utili agli sviluppatori che vorranno manutenere ed estendere il prodotto IronWorks.
In particolare vengono specificati i requisiti necessari per installare l'applicazione e avviarla in modo corretto.
Si fornisce inoltre il dettaglio dell'architettura del software per aiutare il lettore a comprendere com'è strutturato e per guidarlo nelle eventuali modifiche che vorrà apportare.

1.2 Scopo del Prodotto

Lo scopo del prodotto è quello di fornire un editor di *diagrammi di robustezza*$_G$ che permetta all'utente di progettare l'architettura del proprio software secondo lo standard *UML*$_G$.
A partire dal diagramma disegnato è possibile generare automaticamente il corrispondente codice *Java*$_G$ delle *entità*$_G$ persistenti.
Per ogni *entità*$_G$ viene inoltre generato il codice *SQL*$_G$ per permettere la creazione delle tabelle nel database relazionale mentre nei *file*$_G$ *DAO*$_G$ vengono implementate tutte le operazioni *CRUD*$_G$ per gestire la persistenza dei dati.
È prevista anche la generazione del codice per implementare l'interazione con il database attraverso il middleware *Hibernate*$_G$.

1.3 Ambiguità

Al fine di dipanare qualsiasi dubbio o ambiguità relativa al linguaggio impiegato viene fornito il Glossario in appendice A, sezione contenente la definizione di tutti i termini scritti in corsivo e marcati con una 'G' pedice.

1.4 Riferimenti

1.4.1 Riferimenti Informativi

- *Slides del corso di Ingegneria del Software*:
  - *Diagrammi delle classi*:
    http://www.math.unipd.it/~tullio/IS-1/2017/Dispense/E03.pdf;
  - *Diagrammi dei package*:
    http://www.math.unipd.it/~tullio/IS-1/2017/Dispense/E04.pdf.
- *Documentazione della piattaforma Node.js*$_G$:
  https://nodejs.org/docs/latest-v8.x/api/index.html;
- *Documentazione del framework*$_G$ *Express*$_G$*.js*:
  http://expressjs.com/it/4x/api.html;
- *Documentazione della libreria Spectrum*$_G$:
  https://bgrins.github.io/spectrum/;

- *Documentazione della libreria JointJS$_G$*:
  - *Documentazione ufficiale*: http://resources.jointjs.com/docs/jointjs/v2.1/joint.html;
  - *Tutorial di vario livello*: https://resources.jointjs.com/tutorial.
- *Utilizzo e tutorial di HTML$_G$ e HTML5$_G$*:
  - *Documentazione ufficiale*: https://www.w3.org/TR/html5/;
  - *Utilizzo generale*: https://developer.mozilla.org/it/docs/Web/HTML/HTML5;
  - *Tutorial di vario livello*: https://www.w3schools.com/html/.
- *Utilizzo e tutorial di CSS$_G$ e CSS3$_G$*:
  - *Documentazione ufficiale*: https://www.w3.org/Style/CSS/;
  - *Utilizzo generale*: https://developer.mozilla.org/it/docs/Web/CSS;
  - *Tutorial di vario livello*: https://www.w3schools.com/css/.
- *Utilizzo e tutorial di JavaScript$_G$ ed ECMAScript$_G$ 2017*:
  - *Documentazione ufficiale JavaScript$_G$*: https://www.w3.org/standards/webdesign/script;
  - *Documentazione ufficiale ECMAScript$_G$ 2017*: http://ecma-international.org/ecma-402/;
  - *Utilizzo generale*: http://exploringjs.com/es2016-es2017/;
  - *Tutorial di vario livello*: https://www.w3schools.com/js/.
- *Utilizzo e tutorial di jQuery$_G$*:
  - *Documentazione ufficiale*: https://api.jquery.com/;
  - *Tutorial di vario livello*: https://www.w3schools.com/jquery/.

2 Tecnologie Utilizzate

2.1 Node.js

Node.js$_G$ è una piattaforma *open source*$_G$ che permette l'esecuzione di codice *JavaScript*$_G$ lato *server*$_G$, basato sul motore *JavaScript*$_G$ V8 di Chrome.
Utilizza un modello di programmazione ad eventi che permette l'introduzione di operazioni I/O asincrone. È inoltre capace di creare un proprio *server*$_G$ web che gestisce le richieste *HTTP*$_G$ in modo asincrono.
Grazie a queste caratteristiche risulta molto utile per lo sviluppo di *applicazioni web*$_G$ come IronWorks. *Node.js*$_G$ utilizza un vasto ecosistema di pacchetti, definito *npm*$_G$, che permette l'utilizzo di librerie *open source*$_G$ importanti per lo sviluppo del codice.

2.2 Express.js

Express$_G$.js è un *framework*$_G$ che semplifica la gestione del *server*$_G$ web in *Node.js*$_G$.
Offre funzionalità che aiutano lo sviluppatore a implementare il sistema di *routing*$_G$, facilitando il meccanismo di richiesta e risposta di un'applicazione *REST*$_G$.

2.3 HTML5

HTML5$_G$ è un linguaggio di *markup*$_G$ per la strutturazione delle pagine web, pubblicato come W3C Recommendation da ottobre 2014.
Il suo utilizzo permette di realizzare la struttura delle pagine web, ed infatti si pone come una tecnologia essenziale per lo sviluppo di un'*applicazione web*$_G$.
Nello specifico si è deciso per l'utilizzo di *HTML5*$_G$ con sintassi *XHTML*$_G$ per mantenere comunque un codice più pulito, manutenibile e per migliorarne la retrocompatibilità.

2.4 CSS3

CSS$_G$ è un linguaggio di stile usato per descrivere la presentazione di un documento scritto in un linguaggio di *markup*$_G$ come *HTML*$_G$, risultando così una tecnologia essenziale per lo sviluppo di un'*applicazione web*$_G$.
La versione utilizzata è *CSS3*$_G$, la quale offre maggiori potenzialità agli sviluppatori rispetto alle versioni precedenti rendendo così più gradevole e responsive il layout delle pagine web.

2.5 ECMAScript

ECMAScript$_G$ è un linguaggio di programmazione standardizzato e mantenuto da Ecma International.
Tra le implementazioni più conosciute di questo linguaggio (definiti come "dialetti") vi è *JavaScript*$_G$ usato come linguaggio lato *client*$_G$ per lo sviluppo di *applicazioni web*$_G$.
La versione utlizzata è *ECMAScript*$_G$ 2017, in modo da garantire retrocompatibilità ma allo

stesso tempo avere a disposizione la maggior parte delle funzionalità offerte da tale linguaggio.

2.6 jQuery

jQuery$_G$ è una libreria *JavaScript*$_G$ per lo sviluppo di *applicazioni web*$_G$ nata con l'obiettivo di semplificare la selezione, la manipolazione, la gestione degli eventi e l'animazione di elementi *HTML*$_G$ nelle pagine web.
Le sue caratteristiche permettono agli sviluppatori *JavaScript*$_G$ di astrarre le interazioni a basso livello tra interazione e animazione dei contenuti delle pagine. L'approccio di tipo modulare di *jQuery*$_G$ consente la creazione semplificata di *applicazioni web*$_G$ e versatili contenuti dinamici.

2.7 JointJS

JointJS$_G$ è una libreria *JavaScript*$_G$ che permette la realizzazione di diagrammi completamente interattivi.
Si basa su un'architettura *MVC*$_G$ mantenendo una netta separazione tra il grafico del diagramma, il foglio sul quale il diagramma è rappresentato, ed i rispettivi elementi che compongono il diagramma.
Le *API*$_G$ esposte da tale libreria permettono una facile integrazione di essa con la parte *back-end*$_G$ di un'*applicazione web*$_G$.
JointJS$_G$ è realizzata grazie alla libreria *Backbone.js*$_G$, e presenta altre due dipendenze per un suo corretto funzionamento: *jQuery*$_G$ e *Lodash*$_G$.

2.8 Spectrum

Spectrum$_G$ è una libreria *JavaScript*$_G$ che permette la realizzazione di color picker per la scelta di colori.
L'uso di questa libreria permette di appianare qualsiasi divergenza tra i browser, sia per quanto concerne l'utilizzo effettivo di un color picker, sia la sua presentazione mostrata all'utente dal browser che resta sempre modificabile tramite i fogli di stile *CSS*$_G$.

3 Requisiti di Sistema

IronWorks è un'*applicazione web*$_G$, pertanto necessita di una connessione ad internet per poter funzionare correttamente.
L'abilitazione di *JavaScript*$_G$ su qualsiasi browser si desideri utilizzare è una condizione necessaria per un corretto e completo utilizzo dell'applicazione.

3.1 Dispositivi Supportati

L'applicazione è stata testata e risulta compatibile con i seguenti sistemi operativi desktop:

- *Microsoft Windows*$_G$ 10;
- *Apple OSX*$_G$ High Sierra;
- *Ubuntu*$_G$ 16.04 *LTS*$_G$ (64 bit).

IronWorks è un'*applicazione web*$_G$ per la realizzazione di *diagrammi di robustezza*$_G$ tramite un editor, vista la difficoltà di raggiungere un tale scopo tramite un dispositivo mobile non è presente alcuna versione dell'applicazione per questa tipologia di dispositivi.

3.2 Browser Supportati

L'applicazione è compatibile con i seguenti browser, dove la versione indicata è la versione minima dalla quale il funzionamento di IronWorks è garantito:

- *Google Chrome*$_G$ versione 57;
- *Mozilla Firefox*$_G$ versione 52;
- *Safari*$_G$ versione 10.1;
- *Microsoft Edge*$_G$ versione 40;
- *Opera*$_G$ versione 44.

3.3 Requisiti Hardware

Non vi sono particolari requisiti hardware per il funzionamento del prodotto. Gli unici requisiti da segnalare sono quelli relativi al browser utilizzato.

4 Configurazione Ambiente di Lavoro

4.1 Accesso all'Applicazione

L'applicazione può essere utilizzata sia in locale sia da remoto tramite la rete.

4.1.1 Utilizzo in Locale

Per poter accedere in locale all'applicazione IronWorks è necessario:

- Aver installato precedentemente nella propria macchina IronWorks (le istruzioni sono presenti nella sezione 4.2);
- Collegarsi all'indirizzo `http://localhost:3000/` tramite uno dei browser supportati.

4.1.2 Utilizzo Tramite la Rete

Per poter accedere tramite la rete all'applicazione IronWorks è necessario recarsi all'indirizzo `http://46.101.244.120` utilizzando uno dei browser supportati.

4.2 Installazione in Locale dell'Applicazione

4.2.1 Pre-Requisiti

L'applicazione necessita del runtime *JavaScript*$_G$ "*Node.js*$_G$" ed il package manager "*npm*$_G$" installati nel proprio sistema. La versione minima d'installazione è 8.11.1 *LTS*$_G$ di *Node.js*$_G$.
La versione per il proprio sistema operativo è scaricabile all'indirizzo:
`https://nodejs.org/it/download`.

4.2.2 Installare le Dipendenze

Posizionandosi all'interno della *directory*$_G$ "BackEnd", aprire qui un terminale e lanciare il comando *npm*$_G$ `install`.
Questo comando si occuperà di installare tutte le dipendenze necessarie al funzionamento dell'applicazione.

4.2.3 Avvio di IronWorks

Per avviare l'applicazione occorre aprire nuovamente un terminale all'interno della *directory*$_G$ "BackEnd" ed eseguire il comando *npm*$_G$ `start`.
Una volta eseguito il comando, per accedere all'applicazione, lasciare in esecuzione il terminale ed aprire uno dei browser supportati.
Fatto questo, recarsi all'indirizzo `http://localhost:3000` .

Nel caso in cui l'utente decida di aprire il progetto con un *IDE*$_G$ deve configurare manualmente la "run" per l'esecuzione. Affinchè tutto funzioni regolarmente è necessario impostare la *directory*$_G$ "BackEnd" come "working *directory*$_G$" e "bin/www.js" per il *file*$_G$ di esecuzione.

5 Architettura

L'architettura del software IronWorks viene descritta utilizzando un approccio top-down, ovvero presentando ad alto livello la struttura utilizzando i package secondo la notazione UML_G (versione 2) e scendendo nel dettaglio fino alle classi che compongono il prodotto.
Ogni package viene rappresentato da una figura e da una descrizione testuale che ne specifica la funzione. Vengono inoltre indicati gli eventuali package contenuti e le classi di cui è composto e specificate le interazione con i package a cui è connesso.

5.1 Architettura ad Alto Livello

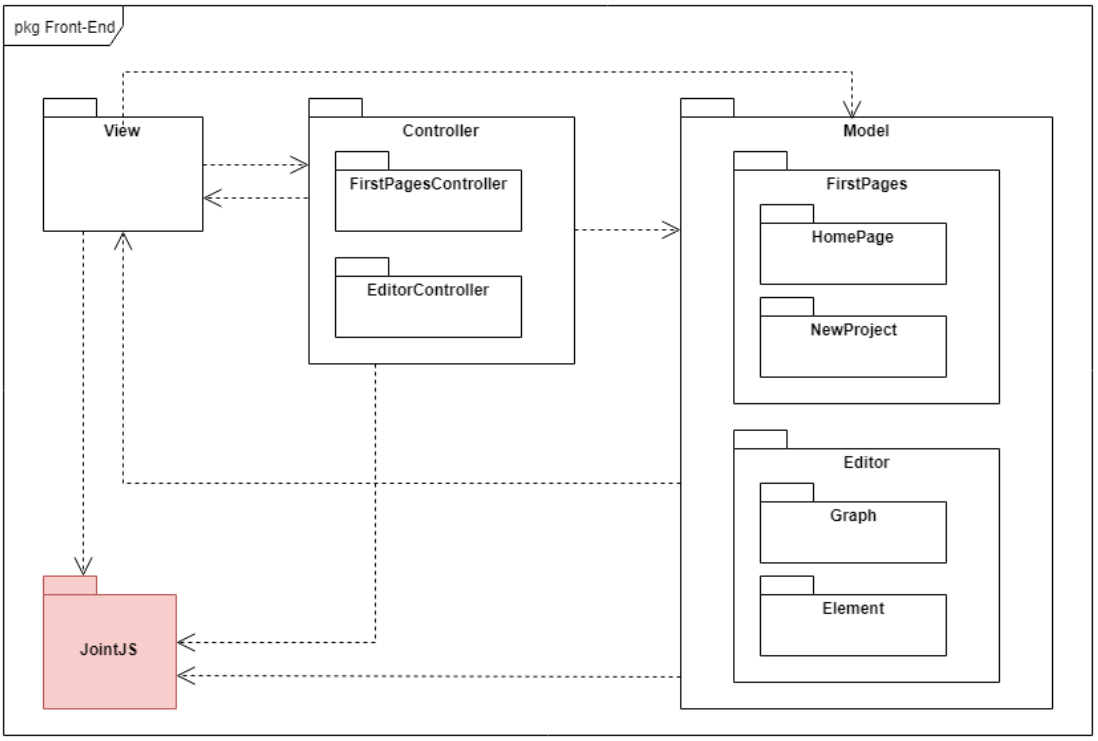

Figura 1: Diagramma dei package `Front-End`

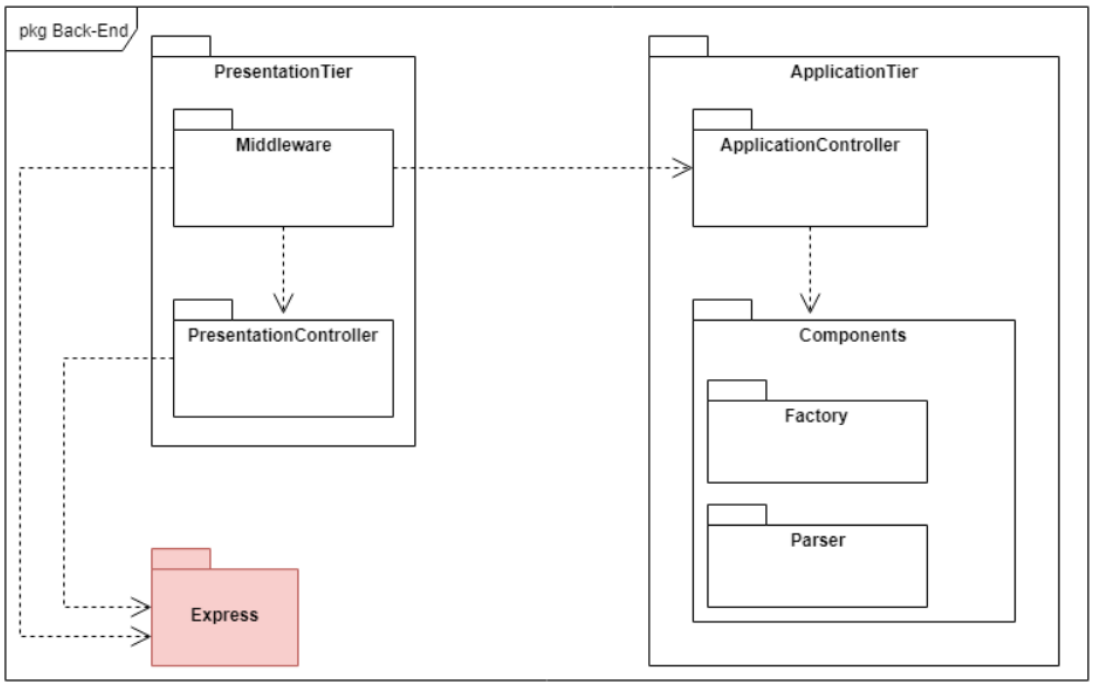

Figura 2: Diagramma dei package `Back-End`

5.1.1 Descrizione

L'applicazione è suddivisa nelle due macro-componenti `Front-End` e `Back-End`.
Il `Front-End` si occupa di interfacciarsi con l'utente per la creazione del *diagramma di robustezza*$_G$, permettendo quindi l'inserimento degli elementi e l'estensione delle informazioni concernenti le *entità*$_G$. Offre inoltre la possibilità di salvare il progetto e generare il codice *Java*$_G$ e *SQL*$_G$ del diagramma realizzato.
È quindi costituito dai *file*$_G$ *HTML*$_G$, *CSS*$_G$ e *JavaScript*$_G$ ed è strutturato secondo il pattern architetturale *MVC*$_G$.
Il `Back-End` si occupa di generare il codice a partire dal diagramma, restituendo un archivio ZIP contenente i *file*$_G$ *Java*$_G$ e *SQL*$_G$. È totalmente scritto in linguaggio *JavaScript*$_G$ ed è strutturato secondo un'architettura *Two-Tier*$_G$ (ovvero *Client*$_G$-*Server*$_G$).
La comunicazione tra le due componenti avviene tramite chiamate *HTTP*$_G$ *REST*$_G$.
In particolare, attraverso una chiamata *POST*$_G$, il `Front-End` invia la struttura del diagramma realizzato al `Back-End` in formato *JSON*$_G$, il quale lo elabora e risponde creando l'archivio ZIP da scaricare.

5.1.2 Package Contenuti

- `Front-End`: contenente i package necessari per implementare la parte *client*$_G$;
- `Back-End`: contenente i package necessari per implementare la parte *server*$_G$.

5.1.3 Librerie e Framework Esterni

- *JointJS*$_G$: libreria che offre le funzionalità per realizzare i *diagrammi di robustezza*$_G$ e renderli interattivi;
- *Spectrum*$_G$: libreria *JavaScript*$_G$ che permette la realizzazione di color picker per la scelta di colori;
- *Express*$_G$: *framework*$_G$ di supporto alla parte *server*$_G$ per la gestione delle chiamate *REST*$_G$.

6 Front-End

6.1 Front-End::View

- **Descrizione**
  Il package Front-End::View permette l'avvio dell'applicazione IronWorks richiedendone nell'immediato il caricamento dell'homepage.
  Inoltre Front-End::View è l'unico package che entra in diretto contatto con l'utente utilizzatore, assumendo così il ruolo di interfaccia con quest'ultimo.
  Questo è il primo package analizzato che rappresenta uno dei tre componenti principali dell'architettura *MVC*$_G$ sulla quale è progettato il Front-End.

- **Package Padre**
  Front-End

- **Interazioni**
  Il package in analisi Front-End::View si occupa dell'interazione con il package Front-End::Controller che di gestisce tutte le interazioni richieste dall'utente.
  Questo package inoltre utilizza la libreria *JointJS*$_G$ per reagire a determinati eventi che si verificano nell'editor, come lo spostamento di un determinato elemento da una posizione iniziale ad un'altra.

- **Altri *File*$_G$**

  - index.html: unico *file*$_G$ prettamente appartenente a Front-End::View, con il fondamentale scopo di avviare una pagina iniziale che avvisa l'utente dell'inutilizzo dell'*applicazione web*$_G$ se *JavaScript*$_G$ non è attivo, ed in caso contrario avvia il processo di caricamento.

6.2 Front-End::Controller

- **Descrizione**
  Il package Front-End::Controller si occupa dell'esecuzione di tutte le richieste derivanti dall'utente utilizzatore dell'applicazione, modellando in questo modo uno dei tre componenti principali dell'architettura *MVC*$_G$ sulla quale è progettato il Front-End.

- **Package Padre**
  Front-End

- **Interazioni**
  Front-End::Controller funge da intermediario tra il package Front-End::View e il package Front-End::Model ricevendo le richieste da Front-End::View ed eseguendole grazie a Front-End::Model.
  Tale package necessita dell'uso della libreria *JointJS*$_G$ per la gestione di diverse funzionalità tutte legate all'editor, tra queste vi sono:

  - *Drag and Drop*$_G$ per la creazione degli elementi;
  - Aggiunta delle linee di associazione per collegare due elementi;

- Gestione degli errori se si tenta di collegare due elementi tra loro incompatibili;
- Salvataggio in locale del diagramma;
- Caricamento di un diagramma salvato in locale.

- **Package Contenuti**
  - `FirstPagesController`: contiene i *file*$_G$ pertinenti le funzionalità relative alla gestione delle pagine iniziali dell'applicazione;
  - `EditorController`: contiene i *file*$_G$ necessari all'esecuzione delle funzioni della pagina dell'editor interattivo per la realizzazione di *diagrammi di robustezza*$_G$.

6.2.1 Front-End::Controller::FirstPagesController

- **Descrizione**
  Il package `Front-End::Controller::FirstPagesController` si occupa del caricamento dell'homepage dell'applicazione e della completa gestione delle altre funzionalità richieste dalle prime pagine di IronWorks.

- **Package Padre**
  `Front-End::Controller`

- **Interazioni**
  All'avvio dell'applicazione, se l'uso di *JavaScript*$_G$ è abilitato, il package in analisi viene chiamato dal package `Front-End::View` per il caricamento dell'homepage.
  Qualsiasi altra richiesta effettuata dell'utente prima di accedere all'editor è sempre gestita da `Front-End::Controller::FirstPagesController`, il quale interagisce con il package `Front-End::Model` per la richiesta delle pagine *HTML*$_G$ ed i fogli di stile *CSS*$_G$ necessari per soddisfare le domande dell'utente utilizzatore.

- **Altri *File*$_G$**
  - `firstpages.js`: questo *file*$_G$ è una collezione di funzioni *JavaScript*$_G$ che si occupano di reindirizzare alla pagina corretta l'utente, oltre a gestire il salvataggio in *Local Storage*$_G$ del nome del progetto e dell'eventuale progetto caricato da locale dall'utente.

6.2.2 Front-End::Controller::EditorController

- **Descrizione**
  Il package `Front-End::Controller::EditorController` è il package più corposo dell'intero package `Front-End::Controller`, e si occupa della completa gestione di tutte le funzionalità esposte all'utente che sono presenti nella pagine dell'editor di IronWorks.

- **Package Padre**
  `Front-End::Controller`

- **Interazioni**
  Questo package si occupa dell'inizializzazione dell'editor al caricamento della pagina ad esso relativa interagendo con il package `Front-End::Model` e dei suoi relativi sotto-package.
  Gestisce anche la richiesta di tornare all'homepage di IronWorks interagendo così con il package `Front-End::Controller::FirstPagesController`.
  Inoltre genera l'oggetto *JSON*$_G$ contenente il diagramma se l'utente richiede la generazione del codice, emettendo la richiesta *POST*$_G$ con il *JSON*$_G$ collegato che verrà ascoltata dal package `Back-End::ApplicationTier`.

- **Altri *File*$_G$**

  - `functionsCalled`: *directory*$_G$ contenente i *file*$_G$ che implementano le funzioni messe a disposizione nell'editor ed invocate al click di uno specifico bottone da parte dell'utente:
    * `addline.js`: *file*$_G$ contenente la funzione che permette l'aggiunta di una linea di associazione per collegare due elementi del diagramma;
    * `editmenu.js`: *file*$_G$ contenente le funzioni che permettono di mostrare e nascondere il menu di sinistra degli elementi su richiesta dell'utente;
    * `savediagram.js`: *file*$_G$ contenente la funzione che permette il salvataggio in locale del diagramma attualmente realizzato dall'utente;
    * `senddata.js`: *file*$_G$ contenente la funzione che genera l'oggetto *JSON*$_G$ del diagramma corrente ed emette la richiesta *POST*$_G$ per il *server*$_G$.
  - `setEvents`: *directory*$_G$ contenente i *file*$_G$ che implementano le funzionalità a disposizione nell'editor ed invocate al verificarsi di un determinato evento generato dall'utente:
    * `draganddrop.js`: *file*$_G$ contenente le funzioni che permettono all'utente di effettuare un *drag and drop*$_G$ per l'aggiunta di un nuovo elemento *Actor*$_G$, *Boundary*$_G$, *Control*$_G$ e/o *Entity*$_G$ al diagramma;
    * `editelement.js`: *file*$_G$ contenente le funzioni che permettono la modifica dei campi di un determinato elemento quando viene effettuato un doppio click su di esso, l'aggiunta, la rimozione, e la modifica di attributi per gli elementi *Entity*$_G$, e la rimozione di un qualsiasi tipo di elemento del diagramma;
    * `editlink.js`: *file*$_G$ contenente la funzione che si occupa di garantire il rispetto delle regole previste dal *Diagramma di Robustezza*$_G$ quando l'utente cerca di modificare l'elemento di partenza e/o di arrivo di una linea di associazione già presente nel diagramma;
    * `zoom.js`: *file*$_G$ contenente la funzione che viene attivata ad ogni movimento della "rotella del mouse" per permettere lo zoom in avanti od indietro del foglio rappresentante l'editor.
  - `editor.js`: *file*$_G$ che si occupa di collegare le relative funzioni agli eventi su determinati elementi della pagina *HTML*$_G$ dell'editor, di inizializzare l'editor stesso in base alle richieste effettuate dall'utente dall'avvio dell'applicazione, di gestire

il refresh della pagina dell'editor e di rimandare l'utente all'homepage dell'applicazione quando lo richiede.

6.3 Front-End::Model

- **Descrizione**
  Il package Front-End::Model contiene tutti i package e le classi che permettono la creazione delle pagine iniziali dell'applicazione e l'istaziazione degli oggetti necessari alla realizzazione di un *diagramma di robustezza*$_G$ grazie ad un editor interattivo.
  Questo package è anch'esso uno dei tre componenti principali dell'architettura *MVC*$_G$ sulla quale è progettato il Front-End.

- **Package Padre**
  Front-End

- **Interazioni**
  Il Front-End::Model viene chiamato dal package Front-End::Controller al fine di creare qualsiasi nuova pagina da mostrare all'utente fornendo le pagine *HTML*$_G$ richieste con i relativi fogli di stile *CSS*$_G$.
  All'avvio della pagina dell'editor il Front-End::Model fornisce al Front-End::Controller anche i *file*$_G$ necessari all'istanziazione dell'editor stesso.
  Il package in analisi inoltre necessita dell'uso della libreria *JointJS*$_G$ per la creazione iniziale dei grafici e dei fogli attinenti all'editor nonché degli elementi inseribili nel diagramma:

  - *Actor*$_G$;
  - *Boundary*$_G$;
  - *Control*$_G$;
  - *Entity*$_G$.

  La libreria *JointJS*$_G$ espone anche il foglio di stile *CSS*$_G$ utile alla presentazione dei componenti sopra elencati e realizzabili proprio grazie all'uso di tale libreria.

- **Package Contenuti**

  - FirstPages: contiene i *file*$_G$ relativi alla creazione delle pagine iniziali dell'applicazione IronWorks;
  - Editor: contiene i *file*$_G$ relativi alla creazione della pagina dell'editor dell'applicazione ed i *file*$_G$ *JavaScript*$_G$ per l'istanziazione dei grafici, fogli ed elementi per la realizzazione di un diagramma.

6.3.1 Front-End::Model::FirstPages

- **Descrizione**
  Il package Front-End::Model::FirstPages contiene i package incaricati della gestione delle pagine iniziali dell'applicazione, quindi non espone funzionalità ma si occupa di mantenere diviso il codice necessario ai rispettivi moduli di IronWorks.

- **Package Padre**
  Front-End::Model

- **Interazioni**
  Front-End::Model::FirstPages viene chiamato dal package Front-End::Controller::FirstPagesController al caricamento di ogni nuova pagina iniziale dell'applicazione per fornire le pagine *HTML*$_G$ richieste con i relativi fogli di stile *CSS*$_G$ al fine di permettere all'utente la fruizione dei contenuti da lui richiesti.

- **Package Contenuti**

  - HomePage: contiene i *file*$_G$ strettamente correlati alla creazione dell'homepage di IronWorks;
  - NewProject: contiene i *file*$_G$ necessari alla creazione della pagina emessa quando l'utente desidera realizzare un nuovo progetto.

6.3.1.1 Front-End::Model::FirstPages::HomePage

- **Descrizione**
  Il package Front-End::Model::FirstPages::HomePage contiene tutti i *file*$_G$ attinenti la creazione della pagina di homepage di IronWorks, separando così in modo modulare tra diversi package le pagine iniziali dell'applicazione.

- **Package Padre**
  Front-End::Model::FirstPages

- **Interazioni**
  Il package in analisi viene chiamato dal package Front-End::Controller::FirstPagesController all'avvio dell'applicazione per fornire la pagina *HTML*$_G$ con il relativo foglio di stile *CSS*$_G$ per la creazione della pagina di homepage di IronWorks.

- **Altri *File*$_G$**

  - homepage.html: *file*$_G$ contenente il codice *HTML*$_G$ della struttura dell'homepage;
  - homepage.css: *file*$_G$ contenente le istruzioni *CSS*$_G$ per la presentazione degli elementi presenti nella struttura della pagina di homepage.

6.3.1.2 Front-End::Model::FirstPages::NewProject

- **Descrizione**
  Il package Front-End::Model::FirstPages::NewProject contiene tutti i *file*$_G$ necessari alla creazione della pagina di inserimento del nome di un nuovo progetto, separando così in moduli distinti i diversi package delle pagine iniziali dell'applicazione IronWorks.

- **Package Padre**
  Front-End::Model::FirstPages

- **Interazioni**
  Il package in analisi viene chiamato dal package
  `Front-End::Controller::FirstPagesController` alla scelta dell'utente di realizzare un nuovo progetto, fornendo a
  `Front-End::Controller::FirstPagesController` la pagina *HTML*$_G$ con il relativo foglio di stile *CSS*$_G$ per la creazione della pagina relativa al nuovo progetto.

- **Altri *File*$_G$**

  - `newproject.html`: *file*$_G$ contenente il codice *HTML*$_G$ per la struttura della pagina concernente l'inserimento di un nuovo progetto;
  - `newproject.css`: *file*$_G$ contenente le istruzioni *CSS*$_G$ per la presentazione degli elementi presenti nella struttura della pagina relativa all'inserimento di un nuovo progetto.

6.3.2 Front-End::Model::Editor

- **Descrizione**
  Il package `Front-End::Model::Editor` contiene i package incaricati della gestione della pagina dell'editor dell'applicazione, ed in particolare contengono i *file*$_G$ *JavaScript*$_G$ necessari all'istanziazione dei relativi elementi che compongono l'editor interattivo di IronWorks.

- **Package Padre**
  `Front-End::Model`

- **Interazioni**
  `Front-End::Model::Editor` viene chiamato dal package
  `Front-End::Controller::EditorController` al caricamento della pagina relativa all'editor dell'applicazione per fornire la pagina *HTML*$_G$, il relativo foglio di stile *CSS*$_G$ e gli script *JavaScript*$_G$ al fine di permettere all'utente la fruizione dei contenuti e delle funzionalità che IronWorks mette a disposizione all'utilizzatore.

- **Package Contenuti**

  - `Graph`: contiene le classi necessarie all'istanziazione degli oggetti `Graph` e `Paper` per la creazione dell'editor interattivo;
  - `Element`: contiene le classi necessarie all'istanziazione degli elementi relativi ad un *diagramma di robustezza*$_G$.

- **Altri *File*$_G$**

  - `editor.html`: *file*$_G$ contenente il codice *HTML*$_G$ per la struttura della pagina dell'editor dell'applicazione;
  - `editor.css`: *file*$_G$ contenente le istruzioni *CSS*$_G$ per la presentazione degli elementi presenti nella struttura della pagina dell'editor.

6.3.2.1 Front-End::Model::Editor::Graph

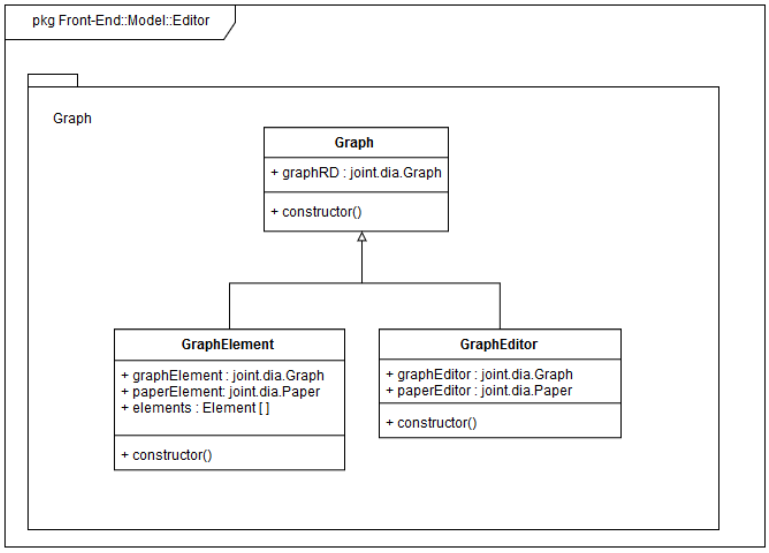

Figura 3: Diagramma delle Classi - Graph

- **Descrizione**
  Il package Front-End::Model::Editor::Graph contiene le classi che si occupano di istanziare gli oggetti Graph e Paper personalizzati per gli scopi di IronWorks e rendono così possibile la creazione di un editor interattivo.

- **Package Padre**
  Front-End::Model::Editor

- **Interazioni**
  Le classi contenute nel package in analisi vengono istanziate dal package Front-End::Controller::EditorController al caricamento della pagina dell'editor.
  Inoltre tali classi istanziano a loro volta oggetti joint.dia.Graph e joint.dia.Paper nella loro forma di default grazie all'uso della libreria esterna $JointJS_G$.

- **Classi Contenute**

  - Graph: classe base per la creazione dei Graph maggiormente specializzati, istanziando un oggetto joint.dia.Graph grazie alle API_G esposte da $JointJS_G$;
  - GraphEditor: classe derivata da Graph che istanzia un oggetto joint.dia.Paper grazie alle API_G esposte da $JointJS_G$, ed imposta i campi dati di tali oggetti in modo peculiare alle esigenze dell'editor in cui l'utente può realizzare i diagrammi;
  - GraphElement: classe derivata da Graph che istanzia un oggetto joint.dia.Paper grazie alle API_G esposte da $JointJS_G$, imposta i campi dati di tali oggetti in modo peculiare alle esigenze dell'editor in cui l'utente visualizza

gli elementi inseribili nell'editor vero e proprio grazie al *drag and drop*$_G$. Inoltre nel momento del caricamento della pagina dell'editor istanzia gli oggetti per la realizzazione di un *diagramma di robustezza*$_G$:

* *Actor*$_G$;
* *Boundary*$_G$;
* *Control*$_G$;
* *Entity*$_G$.

6.3.2.2 Front-End::Model::Editor::Element

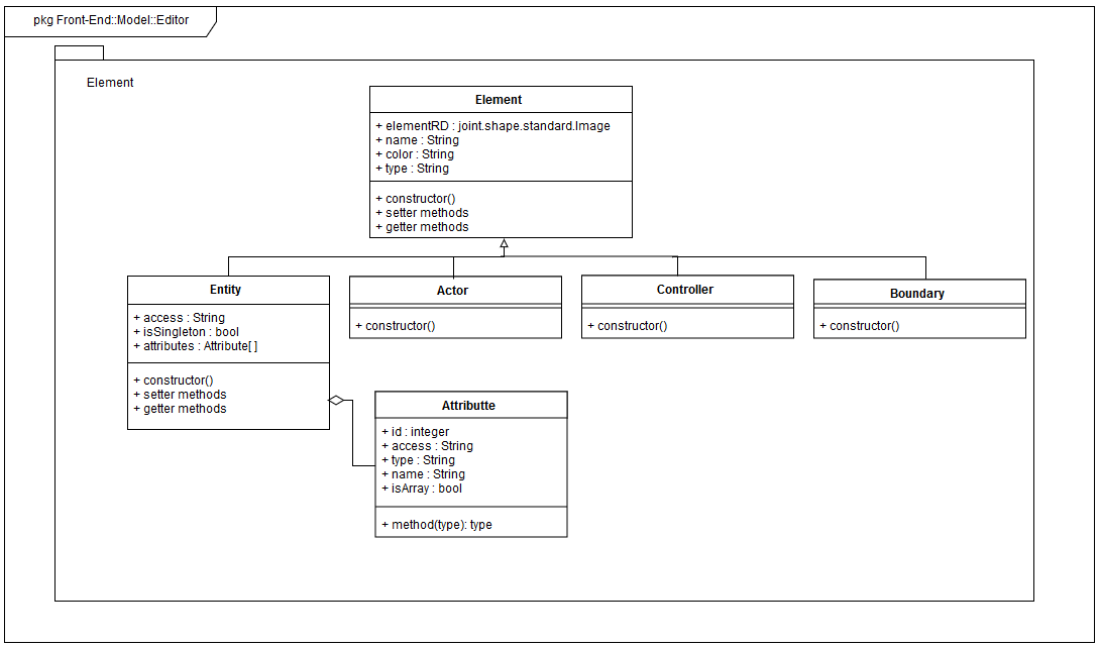

Figura 4: Diagramma delle Classi - Element

- **Descrizione**
  Il package Front-End::Model::Editor::Element contiene le classi che si occupano di istanziare e gestire gli oggetti che identificano gli elementi peculiari di un *diagramma di robustezza*$_G$.

- **Package Padre**
  Front-End::Model::Editor

- **Interazioni**
  Le classi contenute nel package in analisi vengono istanziate dalla classe Front-End::Model::Editor::Graph::GraphElement al caricamento della pagina dell'editor.
  In particolare la classe Element istanzia un oggetto joint.shapes.standard.Image nella sua forma di default grazie all'uso della libreria esterna *JointJS*$_G$.

- **Classi Contenute**
  - `Element`: classe base per la creazione dei singoli elementi maggiormente specializzati appartenenti al *diagramma di robustezza*$_G$ secondo lo standard *UML*$_G$;
  - *Actor*$_G$: classe derivata da `Element` che viene specializzata con i suoi peculiari campi dati relativi ad un "*Actor*$_G$";
  - *Boundary*$_G$: classe derivata da `Element` che viene specializzata con i suoi peculiari campi dati relativi ad un "*Boundary*$_G$";
  - *Control*$_G$: classe derivata da `Element` che viene specializzata con i suoi peculiari campi dati relativi ad un "*Control*$_G$";
  - *Entity*$_G$: classe derivata da `Element` che viene specializzata con i suoi peculiari campi dati relativi ad una "*Entity*$_G$". Tale classe rispetto alle altre classi che rappresentano gli elementi possiede ulteriori campi dati:
    * `access`: visibilità della classe *Java*$_G$ corrispondente;
    * `isSingleton`: flag per specificare se la classe *Java*$_G$ corrispondente deve essere o meno un *Singleton*$_G$;
    * `attributes`: array di oggetti di tipo `Attribute` che descrivono i campi dati della classe *Java*$_G$ corrispondente.

7 Back-End

7.1 Back-End::PresentationTier

- **Descrizione**
  Il package Back-End::PresentationTier si occupa della presentazione dell'applicazione. Non offre quindi funzionalità, ma si occupa di comunicare con il Front-End, di inviare le risorse richieste corrette e di caricare la pagina iniziale di IronWorks.
- **Package Padre**
  Back-End
- **Interazioni**
  Questo package comunica con il *framework*$_G$ *Express*$_G$ per gestire l'attività di *routing*$_G$ e con il BackEnd::ApplicationTier per eseguire le funzionalità richieste.
- **Package Contenuti**
  - BackEnd::PresentationTier::Middleware: contiene le classi per la gestione della comunicazione *client*$_G$-*server*$_G$ e dei relativi errori.
  - BackEnd::PresentationTier::PresentationController: contiene la classe che carica la pagina iniziale nel browser.

7.1.1 Back-End::PresentationTier::Middleware

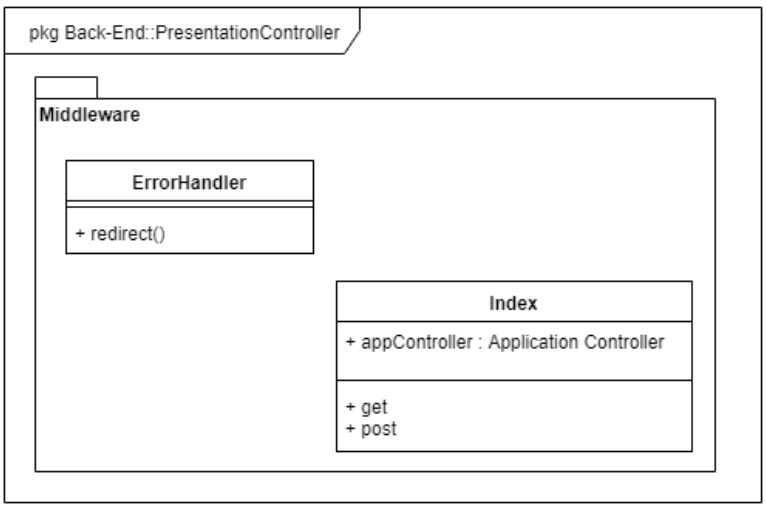

Figura 5: Diagramma delle Classi - Middleware

- **Descrizione**
  Il package Back-End::PresentationTier::Middleware funge da intermediario tra la parte *server*$_G$ e la parte *client*$_G$. Ogni funzione di questo package gestisce le richieste *HTTP*$_G$ del metodo corrispondente chiamando il Controller corretto a cui affidare l'esecuzione delle funzionalità.

- **Package Padre**
  Back-End::PresentationTier
- **Interazioni**
  Questo package utilizza il *framework*$_G$ *Express*$_G$.
  Comunica inoltre con il Back-End::PresentationTier::PresentationController se viene richiesto il caricamento della pagina iniziale e con Back-End::ApplicationTier::ApplicationController se viene richiesta la generazione del codice.
- **Classi Contenute**
  - Index: si occupa delle richieste *REST*$_G$ in base all'*URI*$_G$ ricevuto e prepara la risposta. In particolare:
    * *GET*$_G$ '/': è la prima chiamata al *server*$_G$ per richiedere la pagina principale dell'applicazione. Il *server*$_G$ risponde invocando il Back-End::PresentationTier::PresentationController e inviando il *file*$_G$ ".html" corrispondente alla prima pagina;
    * *POST*$_G$ '/code': è la chiamata per la generazione del codice che ha come parametro un oggetto *JSON*$_G$. Il *server*$_G$ risponde invocando la generazione del codice e l'elaborazione dell'oggetto *JSON*$_G$ in Back-End::ApplicationTier::ApplicationController. La cartella contenente i *file*$_G$ da scaricare viene inserita in un archivio ZIP dalla classe Index per procedere con il download.
  - ErrorHandler: si occupa di gestire gli errori per richieste *REST*$_G$ non definite rispondendo con un redirect alla pagina precedente.

7.1.2 Back-End::PresentationTier::PresentationController

- **Descrizione**
  Il package Back-End::PresentationTier::PresentationController si occupa di ritornare la pagina iniziale di IronWorks per permettere all'utente di accedere all'applicazione.
- **Package Padre**
  Back-End::PresentationTier
- **Interazioni**
  Questo package utilizza il *framework*$_G$ *Express*$_G$ e la classe al suo interno viene invocata dal Back-End::PresentationTier::Middleware in seguito ad una chiamata *GET*$_G$ sulla porta 3000.
- **Classi Contenute**
  - PresentationController: contiene un solo metodo che si occupa di rispondere alla richiesta inviando la pagina *HTML*$_G$ che rappresenta la pagina iniziale dell'applicazione.

7.2 Back-End::ApplicationTier

- **Descrizione**
  Il package Back-End::ApplicationTier contiene tutte le funzionalità che il *server*$_G$ implementa per generare il codice *Java*$_G$, *SQL*$_G$ e gli altri *file*$_G$ necessari per una corretta interazione col database per gestire la persistenza dei dati da un oggetto *JSON*$_G$ ricevuto che contiene le specifiche di un diagramma realizzato nel Front-End.

- **Package Padre**
  Back-End

- **Interazioni**
  Questo package comunica con il Back-End::PresentationTier::Middleware in seguito ad una richiesta *POST*$_G$.

- **Package Contenuti**

  - Back-End::ApplicationTier::ApplicationController: contiene una classe ApplicationController che istanzia le factory per la creazione dei *file*$_G$. Si occupa inoltre di elaborare l'oggetto *JSON*$_G$ e di svuotare ad ogni chiamata la cartella temporanea contente i *file*$_G$ per la creazione dell'archivio ZIP.
  - Back-End::ApplicationTier::Components: contiene i package e le classi che implementano le funzionalità del *server*$_G$.

7.2.1 Back-End::ApplicationTier::ApplicationController

- **Descrizione**
  Il package Back-End::ApplicationTier::ApplicationController gestisce tutte le funzioni necessarie per portare a termine l'azione richiesta dal Back-End::PresentationTier::Middleware.
  L'unica funzionalità richiesta in questo caso è la generazione del codice, di cui si occupa la classe ApplicationController.

- **Package Padre**
  Back-End::ApplicationTier

- **Interazioni**
  Questo package è invocato dal Back-End::PresentationTier::Middleware in seguito ad una chiamata *POST*$_G$. Questa richiesta ha inoltre un parametro che consiste nell'oggetto *JSON*$_G$ creato dal *client*$_G$.

- **Classi Contenute**

  - ApplicationController: si occupa di effettuare il "parsing" dell'oggetto *JSON*$_G$ utilizzando la classe JsonParser in Back-End::ApplicationTier::Components::JsonParser, di istanziare le Factory per la generazione del codice e di svuotare la cartella temporanea nella quale vengono inseriti i *file*$_G$ da scaricare.

Le factory istanziate utilizzando il package
Back-End::ApplicationTier::Components::Factory sono:

* JavaFactory: factory per la generazione del codice in linguaggio *Java*$_G$ per la rappresentazione delle *entità*$_G$. Vengono rappresentate sia come classi pure, sia come classi contenenti le direttive per *Hibernate*$_G$;
* SqlFactory: factory per la generazione del codice in linguaggio *MySQL*$_G$ per la rappresentazione delle *entità*$_G$ in un database;
* DaoFactory: factory per la generazione del codice in linguaggio *Java*$_G$ per l'interazione col database e la persistenza delle *entità*$_G$;
* HibernateFactory: factory per la generazione del codice in linguaggio *Java*$_G$ che crea i metodi di interazione con il database appoggiandosi all'*ORM*$_G$ *Hibernate*$_G$.

7.2.2 Back-End::ApplicationTier::Components

- **Descrizione**
  Il package Back-End::ApplicationTier::Components contiene tutti i package e le classi che permettono all'applicazione di generare i *file*$_G$ ".java" e ".sql" relativi al *diagramma di robustezza*$_G$ realizzato nel Front-End. Rappresenta quindi le funzionalità elaborate nel *server*$_G$.

- **Package Padre**
  Back-End::ApplicationTier

- **Interazioni**
  Questo package viene utilizzato dal
  BackEnd::ApplicationTier::ApplicationController per generare i *file*$_G$ necessari.

- **Package Contenuti**

  - Back-End::ApplicationTier::Components::Factory: contiene le classi per generare il codice nei *file*$_G$ ".java" e ".sql".
    Ogni *file*$_G$ ".java" prodotto sarà utilizzato dall'utente per interagire col database, pertanto verranno anche creati i *file*$_G$ per gestire questa funzione, in particolare per mantenere la persistenza dei dati tra il codice e il database.
    Per questa funzionalità si propone sia la soluzione semplice con la creazione dei metodi nella DaoFactory sia la soluzione che interagisce con il middleware *Hibernate*$_G$ per il mantenimento automatico della persisitenza dei dati.
  - Back-End::ApplicationTier::Components::Parser: contiene le classi che si occupano di organizzare l'oggetto *JSON*$_G$ inviato dal Front-End in una struttura facilmente accessibile per manipolare in modo più semplice i dati ottenuti.

7.2.2.1 Back-End::ApplicationTier::Components::Factory

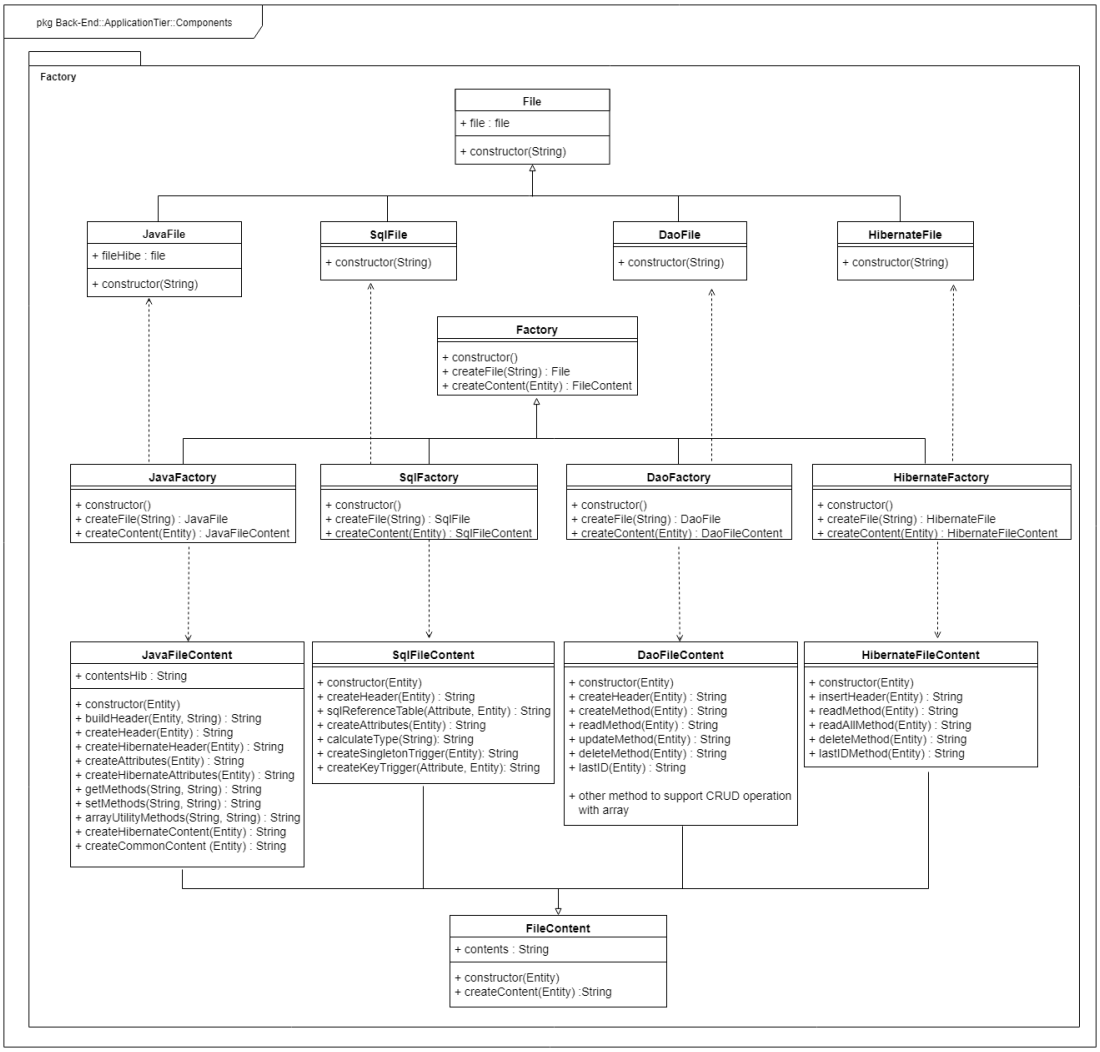

Figura 6: Diagramma delle Classi - Factory

- **Descrizione**
  Il package Back-End::ApplicationTier::Components::Factory contiene le classi che si occupano di implementare, creare e popolare correttamente i *file*$_G$ ".java", ".sql" e i relativi *file*$_G$ per la gestione della persistenza dei dati. Per svolgere questo compito, tali classi sono state strutturate secondo il *design pattern*$_G$ *Abstract Factory*$_G$ per creare strutturalmente *file*$_G$ equivalenti ma con diversa implementazione.

- **Package Padre**
  Back-End::ApplicationTier::Components

- **Interazioni**
  Le Factory all'interno di questo package vengono istanziate dal

Back-End::ApplicationTier::ApplicationController dopo aver ricevuto la richiesta di generazione del codice.

- **Classi Contenute**
  - Factory: classe base per la creazione delle Factory. Dichiara l'intestazione delle funzioni per creare i *file*$_G$ e i contenuti dei *file*$_G$ da implementare nelle sottoclassi;
  - JavaFactory: classe derivata da Factory che crea i *file*$_G$ ".java" ed i contenuti da inserire. Questa factory genera:
    * I *file*$_G$ con le classi *Java*$_G$ rappresentanti le *entità*$_G$ per la gestione manuale della persistenza dei dati;
    * I *file*$_G$ con le classi *Java*$_G$ rappresentanti le *entità*$_G$ per la gestione automatica della persistenza dei dati attraverso *Hibernate*$_G$.
  - SqlFactory: classe derivata da Factory che crea il *file*$_G$ ".sql" ed i contenuti da inserire per la rappresentazione delle *entità*$_G$ nelle tabelle del database;
  - DaoFactory: classe derivata da Factory che crea il *file*$_G$ ".java" ed i contenuti da inserire. I *file*$_G$ generati contengono i metodi di interazione con il database, quindi le operazioni *CRUD*$_G$, con altre funzionalità di supporto;
  - HibernateFactory: classe derivata da Factory che crea il *file*$_G$ ".java" ed i contenuti da inserire. I *file*$_G$ generati contengono i metodi di interazione con il database che sfruttano il middleware *Hibernate*$_G$ per l'implementazione;
  - *File*$_G$: classe base per la creazione di un *file*$_G$;
  - JavaFile: classe derivata da *File*$_G$ per la creazione di un *file*$_G$ ".java";
  - SqlFile: classe derivata da *File*$_G$ per la creazione del *file*$_G$ ".sql";
  - DaoFile: classe derivata da *File*$_G$ per la creazione di un *file*$_G$ ".java" per la gestione della persistenza;
  - HibernateFile: classe derivata da *File*$_G$ per la creazione di un *file*$_G$ ".java" per la gestione della persistenza con *Hibernate*$_G$;
  - FileContent: classe base per la creazione dei contenuti dei *file*$_G$;
  - JavaFileContent: classe derivata da FileContent che crea gli attributi, i metodi "get" e "set" e alcuni altri metodi per aiutare l'utente nel gestire gli array rispettando l'*entità*$_G$ generata nel *client*$_G$;
  - SqlFileContent: classe derivata da FileContent che produce le tabelle per creare la struttura del database rispettando l'*entità*$_G$ generata nel *client*$_G$;
  - DaoFileContent: classe derivata da FileContent che produce le operazioni *CRUD*$_G$ relative alle *entità*$_G$ generate nel *client*$_G$ per gestire la persistenza;
  - HibernateFileContent: classe derivata da FileContent che gestisce le operazioni *CRUD*$_G$ con il middleware *Hibernate*$_G$.

  Per semplificare la scrittura del codice riguardante la generazione dei contenuti dei *file*$_G$ sono stati creati dei *file*$_G$ *template*$_G$ in una cartella fileTemplates nella *directory*$_G$

BackEnd. Questi *file*$_G$ *template*$_G$ vengono letti al momento della creazione dei contenuti e in base all'*entità*$_G$ in esame vengono sostituiti i parametri di questo *file*$_G$ con le informazioni corrette. Questo perchè i *file*$_G$ generati da una factory sono strutturalmente equivalenti.

7.2.2.2 Back-End::ApplicationTier::Components::Parser

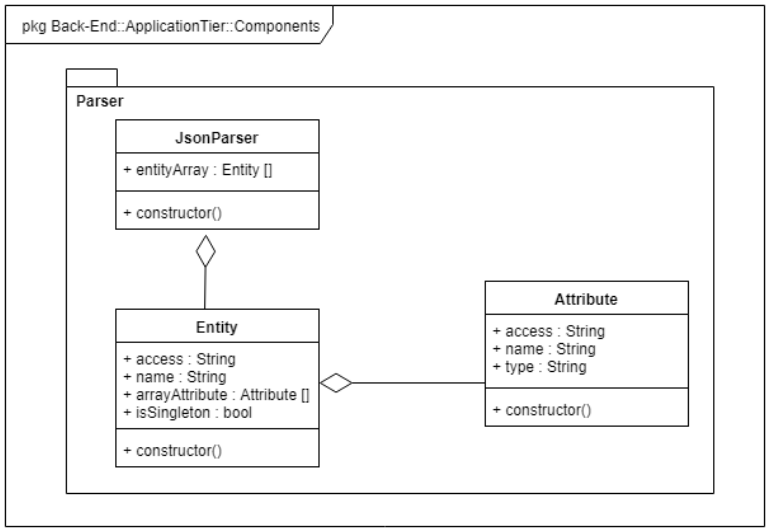

Figura 7: Diagramma delle Classi - Parser

- **Descrizione**
  Il package Back-End::ApplicationTier::Components::Parser si occupa di creare oggetti *Entity*$_G$ a partire dall'oggetto *JSON*$_G$ che rappresenta il diagramma realizzato dall'utente nel Front-End. L'oggetto Parser istanziato contenente l'array di oggetti *Entity*$_G$ è accessibile in qualunque punto dell'applicazione in modo da poter ottenere i dati contenuti in esso in modo celere e semplice.

- **Package Padre**
  Back-End::ApplicationTier::Components

- **Interazioni**
  Questo package viene utilizzato dal
  Back-End::ApplicationTier::ApplicationController che riceve l'oggetto *JSON*$_G$ e ne esegue il "parsing" attraverso le classi implementate nel Parser.

- **Classi Contenute**
  - JsonParser: si occupa di creare un array di *Entity*$_G$ leggendole dall'oggetto *JSON*$_G$ ricevuto;
  - *Entity*$_G$: si occupa di istanziare un oggetto *Entity*$_G$ settando il nome, la visibilità e la caratteristica *singleton*$_G$ di questa. Crea inoltre un array di Attribute per tenere traccia degli attributi presenti nell'*entità*$_G$;

- `Attribute`: si occupa di istanziare un oggetto `Attribute` costituito dal nome, dal tipo e dall'accesso dell'attributo.

8 Estensione Funzionalità

8.1 Aggiunta di un Foglio per un Nuovo Diagramma

IronWorks è un'*applicazione web*$_G$ per la realizzazione di *diagrammi di robustezza*$_G$ ma è possibile voler aggiungere un ulteriore foglio interattivo per voler aumentarne le potenzialità. L'aggiunta di un nuovo foglio interattivo prevede la definizione di una nuova classe che possegga le proprietà specifiche del nuovo foglio che si desidera realizzare derivandola dalla classe base `Graph`.
Inoltre sarà necessario anche aggiungere un elemento alla struttura della pagina *HTML*$_G$ dell'editor presente nel package `Front-End::Model::Editor`.

8.2 Aggiunta di Altri Elementi per il Diagramma

Per come è strutturato il *front-end*$_G$ di IronWorks l'aggiunta di un nuovo elemento che possa essere inserito all'interno del diagramma è piuttosto semplice.
Questa operazione necessita di avere l'*SVG*$_G$ del nuovo elemento che si desidera aggiungere, ed estendere in modo opportuno il package `Front-End::Model::Editor::Element` aggiungendo un'ulteriore classe che descriva i campi dati del nuovo elemento derivandola dalla classe base `Element`.
Dovrà inoltre essere manipolata la struttura della pagina *HTML*$_G$ dell'editor presente nel package `Front-End::Model::Editor`.
Questo procedimento è di facile implementazione, ma si vuole far notare che lo standard *UML*$_G$ per quanto concerne il *diagramma di robustezza*$_G$ non prevede altri elementi al di fuori di quelli già presenti, per cui se si vuole rimanere ferrei allo standard tale operazione è fortemente sconsigliata.

8.3 Aggiunta di Attributi ad un Elemento

Un oggetto della classe *Entity*$_G$ possiede delle proprietà in più rispetto agli altri elementi che compongono il *diagramma di robustezza*$_G$.
Per poter fornire l'aggiunta di proprietà o attributi ad un qualsiasi elemento del diagramma è necessario aggiungere tali nuovi campi dati alla definizione della classe derivata dalla classe base `Element`, entrambe presenti nel package `Front-End::Model::Editor::Element`. La struttura della pagina *HTML*$_G$ dell'editor ed il relativo foglio di stile *CSS*$_G$ dovranno essere correttamente aggiornati ed adattati ai nuovi campi dati inseriti.

8.4 Creazione del Codice in Altri Linguaggi

La struttura del codice permette facilmente di inserire la generazione del codice del diagramma in un altro linguaggio di programmazione.
Estendendo opportunamente il pattern *Abstract Factory*$_G$ nel package `Back-End::ApplicationTier::Components::Factory` e aggiungendo l'istanziazione della nuova `Factory` in `Back-End::ApplicationTier::ApplicationController` si possono ottenere i *file*$_G$ contenenti il codice espresso in un altro linguaggio di programmazione.

8.5 Generazione Codice da un Oggetto JSON Diverso

Grazie all'implementazione delle classi all'interno del package `Back-End::ApplicationTier::Components::Parser` è possibile ricevere e leggere un oggetto $JSON_G$ con una struttura diversa modificando poche righe di codice. Una volta adattato il codice, le altre classi continueranno a funzionare correttamente in quanto utilizzano tutte lo stesso oggetto istanziato dalla classe `JsonParser`.

9 Estensione Codice

9.1 Aggiunta di un Foglio per un Nuovo Diagramma

Per aggiungere un nuovo foglio interattivo è necessario procedere nel seguente modo per quanto concerne il package `Front-End::Model::Editor`:

- Creare una classe `GraphNuovoFoglio` che eredita dalla classe base `Graph`;
- Aggiungere alla struttura della pagina *HTML*$_G$ presente nel *file*$_G$ `editor.html` del package in analisi un nuovo blocco `<div>` con un codice identificativo univoco associato `id="idNuovoFoglio"`;
- Modificare il *file*$_G$ `editor.css` per adattare le regole del foglio di stile *CSS*$_G$ in modo da gestire la presentazione del nuovo foglio interattivo inserito.

Nel package `Front-End::Controller::EditorController` bisogna istanziare un nuovo oggetto della classe `GraphNuovoFoglio` ed in base alle funzionalità che si vogliono esporre riguardanti questo nuovo foglio è necessario recarsi nelle rispettive funzioni presenti nel package `Front-End::Controller::EditorController`.

9.2 Aggiunta di Altri Elementi per il Diagramma

Per aggiungere un nuovo elemento da inserire all'interno del diagramma è necessario procedere nel seguente modo per quanto concerne il package `Front-End::Model::Editor`:

- Creare una classe `NuovoElemento` che eredita dalla classe base `Element`;
- Aggiungere l'istanziazione di tale elemento al caricamento dell'editor nella classe `GraphElement`;
- Aggiungere alla struttura della pagina *HTML*$_G$ presente nel *file*$_G$ `editor.html` del package in analisi un nuovo blocco `<div>` con la classe associata `class="divElement"` per mantenere le stesse regole del foglio di stile *CSS*$_G$ applicate agli altri elementi già presenti.

Dopo aver inserito la classe relativa al nuovo elemento ed averlo aggiunta alla struttura della pagina è necessario aggiungere la funzionalità del *drag and drop*$_G$ anche al nuovo elemento creato aggiungendo tale funzionalità anche nella funzione *JavaScript*$_G$ `dragAndDrop()` nel *file*$_G$ omonimo presente nel package `Front-End::Controller::EditorController`.

9.3 Aggiunta di Attributi ad un Elemento

L'aggiunta di proprietà o attributi ad un qualsiasi elemento del diagramma prevede il seguire determinate operazioni:

- Aggiungere alla classe dell'elemento che si vuole accrescere, la quale è situata nel package `Front-End::Model::Editor::Element` e deriva dalla classe base `Element`;
- Adattare la sezione relativa all'elemento in analisi della pagina *HTML*$_G$ dell'editor;

- Scrivere le regole del foglio di stile *CSS*$_G$ per i nuovi elementi *HTML*$_G$ aggiunti nel *file*$_G$ `editor.css` presente nel package `Front-End::Model::Editor`.

Per completare l'aggiunta è anche richiesta la modifica della funzione `editElement()` e delle funzioni ad essa correlate presenti nel *file*$_G$ omonimo del package `Front-End::Controller::EditorController`.

9.4 Creazione del Codice in Altri Linguaggi

Per aumentare le funzionalità del prodotto aggiungendo la generazione di codice in un nuovo linguaggio di programmazione è necessario modificare le classi all'interno del package `Back-End::ApplicationTier::Components::Factory` nel seguente modo:

- Creare una classe `NuovoLinguaggioFactory` che eredita dalla classe base `Factory`;
- Creare una classe `NuovoLinguaggioFile` che eredita dalla classe base *File*$_G$ e crea il *file*$_G$ con l'estensione desiderata;
- Creare una classe `NuovoLinguaggioFileContent` che eredita dalla classe base `FileContent` e costruisce i contenuti correttamente.

Affinchè venga istanziata la nuova `Factory` è necessario accedere alla classe `ApplicationController` nel package `Back-End::ApplicationTier::ApplicationController` e creare l'oggetto `NuovoLinguaggioFactory`.
I progettisti ed i programmatori hanno scelto di non accedere all'oggetto *File*$_G$ e `FileContent` nella classe `ApplicationController`, ma di delegare al costruttore della `factory` la costruzione dei "prodotti".
Pertanto il costruttore del `NuovoLinguaggioFactory` si deve occupare di salvare i *file*$_G$ generati (o la *directory*$_G$ contente i *file*$_G$) con i relativi contenuti all'interno della cartella "cartellatemp". Questa cartella viene generata automaticamente dal *server*$_G$ e verrà poi utilizzata per creare l'archivio ZIP con i *file*$_G$ da scaricare.

9.5 Generazione Codice da un Oggetto JSON Diverso

Nel caso vengano aggiunte nuove funzionalità all'editor e venga quindi aggiornato l'oggetto *JSON*$_G$ relativo, è necessario che anche le funzionalità lato *server*$_G$ vengano aggiornate.
A tale scopo è necessario accedere alle classi del package `Back-End::ApplicationTier::Components::Parser` e seguire le seguenti istruzioni:

- Per aggiungere informazioni al *server*$_G$ riguardo a nuove specifiche degli elementi *entità*$_G$ dell'editor occorre aggiungere un campo dati alla classe *Entity*$_G$ e costruirlo seguendo la specifica dell'oggetto *JSON*$_G$;
- Per aggiungere informazioni al *server*$_G$ riguardo a nuove specifiche degli attributi degli elementi *entità*$_G$ procedere come sopra descritto nella classe `Attribute`;

- Nel caso si volessero utilizzare nuove informazioni dal diagramma ora non previste, come identificare altri elementi con alcune specifiche, consigliamo di creare una classe apposita che ricostruisca tale oggetto nel *server*$_G$ seguendo il modello del *file*$_G$ *Entity*$_G$. Va poi aggiornata la classe `JsonParser` introducendo come campo dati l'array contenente questi nuovi oggetti.

10 Utilizzo del Codice Generato

10.1 Contenuto del File Zip Scaricato

La zip scaricata al suo interno contiene:

- **scriptTables.sql**: script *SQL*$_G$ che genera le tabelle e i trigger necessari per la gestione del *database*$_G$;
- **config.properties**: *file*$_G$ di configurazione contenente le informazioni necessarie all'applicazione *Java*$_G$ per connettersi al *database*$_G$ utilizzando unicamente il driver *JDBC*$_G$;
- **hibernate.cfg.xml**: *file*$_G$ di configurazione contenente le informazioni necessarie all'applicazione *Java*$_G$ per connettersi al *database*$_G$ attraverso il *framework*$_G$ *Hibernate*$_G$ *ORM*$_G$;
- **app.java**: *file*$_G$ contenente unicamente una classe `App` con il metodo `main`, modificabile per eseguire operazioni;
- **cartella hibernate**: contiene tutti i *file*$_G$ ".java" necessari a gestire il *database*$_G$ attraverso *Hibernate*$_G$ *ORM*$_G$;
- **cartella jdbc**: contiene tutti i *file*$_G$ ".java" necessari a gestire il *database*$_G$ attraverso il solo utilizzo del driver *JDBC*$_G$.

10.2 Esecuzione dello Script Sql

Lo script *SQL*$_G$ è stato scritto seguendo la sintassi *MySQL*$_G$ e testato per funzionare in un *database*$_G$ *MySQL*$_G$ con storage engine *InnoDB*$_G$.
Una volta creato il *database*$_G$ che si desidera popolare, è sufficiente eseguire lo script per creare tabelle e trigger: da linea di comando *MySQL*$_G$ digitare `source path-to/scriptTables.sql`.
Le tabelle sono relative alle *entità*$_G$ create nel diagramma, mentre i triggers permettono di mantenere e gestire eventuali singleton e gli indici delle tabelle relative agli array.

10.3 Gestione del Database Tramite Driver JDBC

I *file*$_G$ ".java" relativi alla gestione del *database*$_G$ tramite driver *JDBC*$_G$ sono organizzati nei seguenti package:

- **jdbc.entities**: contiene semplici classi *Java*$_G$ (anche dette *POJO*$_G$) con attributi e relativi metodi `getter` e `setter`;
- **jdbc.dao**: contiene le classi *Java*$_G$ che interagiscono con il *database*$_G$ (nello specifico, classi *DAO*$_G$);
- **jdbc.helpers**: contiene le classi necessarie a creare la connessione al *database*$_G$.

È innanzitutto necessario impostare url, username e password del *database*$_G$ che si vuole gestire, modificando gli opportuni campo del *file*$_G$ "config.properties".
Per funzionare, l'applicazione necessita della libreria "MySQL connector" reperibile al link

https://dev.mysql.com/downloads/connector/j/.

Attenzione: in caso di segnalazione di errori relativi al Time-Zone durante la connessione al *database*$_G$, potrebbe essere necessario aggiungere in fondo all'url la seguente stringa:

```
?autoReconnect=true&useSSL=false&
useLegacyDatetimeCode=false&serverTimezone=UTC
```

10.4 Gestione del Database Tramite Framework Hibernate ORM

I *file*$_G$ ".java" relativi alla gestione del *database*$_G$ tramite *framework*$_G$ sono organizzati nei seguenti package:

- **hibernate.entitites**: contiene le classi *Java*$_G$ con attributi e relativi metodi `get`. Sugli attributi troviamo le annotazioni *Hibernate*$_G$ che permettono il mapping tra questi e le tabelle *SQL*$_G$;
- **hibernate.dao**: contiene le classi *Java*$_G$ che interagiscono con il *database*$_G$.

È innanzitutto necessario impostare url, username e password del *database*$_G$ che si vuole gestire, modificando gli opportuni campi del *file*$_G$ "hibernate.cfg.xml".

Per funzionare, l'applicazione necessita delle seguenti librerie:

- **libreria MySQL connector**: reperibile al link https://dev.mysql.com/downloads/connector/j/;
- **Framework Hibernate ORM**: reperibile al link http://hibernate.org/orm/releases/5.3/. Le librerie necessarie sono tutte all'interno della cartella "required".

Attenzione: in caso di segnalazione di errori relativi al Time-Zone durante la connessione al *database*$_G$, potrebbe essere necessario aggiungere in fondo all'url la seguente stringa:

```
?autoReconnect=true&amp;useSSL=false&amp;
useLegacyDatetimeCode=false&amp;serverTimezone=UTC
```

10.5 Compilazione ed Esecuzione

Il codice *Java*$_G$ generato è utilizzabile inserendo nel *file*$_G$ App.java, contenente il metodo main, le istruzioni che si vogliono eseguire.
È necessario modificare gli import in base all'uso di solo *JDBC*$_G$ o di *Hibernate*$_G$. Gli import sono già presenti nel *file*$_G$, è sufficiente commentare quelli non necessari.

Di seguito si riportano le istruzioni per compilare ed eseguire l'applicazione da terminale.
Assumendo che la cartella scaricata dall'editor si chiami Project:

- Da dentro la cartella Project, aprire il terminale e digitare:
  - Linux o Mac: `find -name "*.java" > sources.txt`

  - Windows: `dir /s /B *.java > sources.tx`

  In questo modo viene creato il *file*$_G$ sources.txt che contiene tutti i path delle classi che devono essere compilate;

- creare dentro Project una cartella rinominata ad esempio out;
- creare dentro Project una cartella rinominata ad esempio javalib;
- copiare dentro javalib tutte le librerie necessarie sopra indicate, sia per *JDBC*$_G$ che per *Hibernate*$_G$;
- da terminale, dentro Project, eseguire: textttjavac -cp "javalib/*" -d ./out/ @sources.txt In questo modo vengono compilate le classi *Java*$_G$, e il bytecode sarà disponibile all'interno della cartella out;
- per eseguire l'applicazione da terminale, sempre dentro Project:
  - in *Java*$_G$ 8:
    * Linux o Mac: `java -cp ../javalib/*: App`
    * Windows: `java -cp ../javalib/*; App`
  - in *Java*$_G$ 9 o superiore:
    * Linux o Mac: `java -cp ../javalib/*: -add-modules java.xml.bind App`
    * Windows: `java -cp ../javalib/*; -add-modules java.xml.bind App`

10.6 Contenuto Tabelle SQL

Le tabelle *SQL*$_G$ relative alle *entità*$_G$ hanno la seguente struttura:

- Nome uguale alla variabile che contiene l'istanza;
- Campo `db_id` generato automaticamente, impostato come Primary Key con politica auto-increment;
- Lista di attributi aventi nomi uguali a quello inserito nel diagramma.

Le tabelle *SQL*$_G$ relative agli array hanno la seguente struttura:

- Nome uguale al nome dell'array;
- Campo `db_id`, gestito da trigger o dai metodi *Java*$_G$ per mantenere la posizione di un elemento all'interno dell'array;
- Campo chiave esterna avente nome uguale alla variabile che contiene l'istanza dell'*entità*$_G$, seguito da "id";
- Campo `element`, contenente i valori dell'array.

La chiave primaria è composta dai campi `db_id` e dalla chiave esterna.

I trigger hanno lo scopo di mantenere le classi marcate come singleton e gestire gli indici degli array in modo da mantenere correlazione tra istanza, array e posizione degli elementi all'interno dell'array.

10.7 Contenuto Classi Java

Le classi *POJO*$_G$ mettono a disposizione il costruttore di default senza parametri e i metodi `getter` e `setter` per gestire gli attributi dell'*entità*$_G$. Assumendo che il nome dell'attributo sia "Prova" i metodi sono invocabili con:

- `getProva()`, restituisce il valore di "Prova";
- `setProva(value)`, imposta il valore di "Prova" con l'elemento passato come parametro.

Viene automaticamente inserito un campo dati `Db_id` che ha lo scopo di mantenere la correlazione tra *entità*$_G$ *Java*$_G$ e i record nel *database*$_G$. Sono disponibili i relativi metodi `getter` e `setter` qualora lo si volesse manipolare.
Modifiche al campo `Db_id` non comporteranno modifiche al relativo campo nella tabella per non perdere la correlazione tra *entità*$_G$ e oggetto.
Vengono forniti dei metodi di utilità di gestione degli array qualora questi fossero presenti. Assumendo che il nome dell'array sia "Example":

- `popFromExample()`, rimuove l'elemento in coda all'array e lo restituisce;
- `removeFromExample(index)`, rimuove l'elemento nella posizione indicata e lo restituisce;
- `pushToNomeExample(value)`, aggiunge in coda l'elemento passato come parametro e restituisce un `Boolean`;
- `editExampleElement(index, value)`, permette di sostituire l'elemento nella posizione indicata con l'elemento passato come parametro e restituisce un `Boolean`.

10.8 Contenuto Classi DAO

Le classi *Dao*$_G$ offrono le seguenti funzionalità:

- `insert(Object)`, permette di inserire (o aggiornare qualora la chiave primaria del *database*$_G$ fosse la stessa) l'oggetto `Object` nel *database*$_G$ restituendo l'oggetto aggiunto;
- `read(id)`, fornito l'`id` dell'oggetto, il metodo restituisce un'istanza creata con i parametri prelevati dal databas. In caso di errori viene restituito il valore `null`;
- `readAll()`, restituisce un `ArrayList` contenente tutti gli elementi letti dal *database*$_G$;
- `delete(Object)`, rimuove dal *database*$_G$ l'elemento passato come parametro. Elimina anche tutti gli elementi dagli array e dalle relative tabelle in una politica CASCADE;
- `lastDb_id()`, restituisce l'indice più alto all'interno del *database*$_G$. Viene utilizzato per inizializzare il campo dati `Db_id` delle classi *POJO*$_G$ durante l'inserimento nel *database*$_G$.

In aggiunta, le classi *Dao*$_G$ che utilizzano unicamente il driver *JDBC*$_G$ mettono a disposizione anche i seguenti metodi privati, utilizzati dai metodi appena descritti per mantenere la consistenza con gli array.
Assumendo il nome dell'array "Example" troviamo:

- `deleteFromExample(id, i, connection)`, rimuove dalla tabella tutti gli elementi collegati all'*entità*$_G$ di indice `id` e con il loro `id >= i`. Viene utilizzato per eliminare gli elementi in eccesso dalla tabella qualora l'array venisse ridotto di dimensione;
- `deleteFromExample(id, connection)`, invoca il metodo `deleteFromExample(id, i, connection)`, passando come indice `i` il valore '0'. Lo scopo è di eliminare tutti gli elementi della tabella relativa all'array dell'*entità*$_G$ di indice `id`. Viene invocato generalmente in seguito alla `delete` dell'*entità*$_G$, eliminando prima gli elementi della tabella relativa agli array in una politica CASCADE;
- `sqlAddExample(Object, connection)`, inserisce o aggiorna la tabella relativa agli array dell'elemento `Object`, sfruttando la connessione creata dal metodo chiamante;
- `tableExampleLength(id, connection)`, restituisce il numero di elementi all'interno della tabella. Viene generalmente utilizzato dai chiamanti per confrontare il numero di elementi nella tabella con quello negli array per scegliere l'operazione da effettuare per gestire la consistenza;
- `readExample(id, connection)`, restituisce un `ArrayList` contenente gli elementi collegati ad una *entità*$_G$, ricevendo il suo `id` e la connessione creata dal chiamante.

È messa a disposizione una classe `Database` che legge il *file*$_G$ "config.properties" e che restituisce la connessione invocando il metodo `getConnection()`.

10.9 Ulteriori Informazioni

È possibile in ogni momento passare dalla gestione del *database*$_G$ con il codice che utilizza il solo driver *JDBC*$_G$ al codice che utilizza il *framework*$_G$ *Hibernate*$_G$ e viceversa.

Infatti, grazie all'utilizzo del costrutto *try-with-resources*$_G$, è possibile gestire le eccezioni e chiudere in modo automatico le connessioni, indipendentemente dalla loro implementazione.

I metodi che ricevono la connessione come parametro utilizzano se necessario il costrutto *try-with-resources*$_G$, senza utilizzo del catch, lasciando la gestione dell'eccezione al chiamante.
Molto importante è la gestione dell'atomicità delle operazioni ottenuta grazie ai `commit` in caso di successo e ai `rollback` in caso di fallimento.

Attenzione: si ricorda che in *SQL*$_G$ gli indici partono da 1, mentre da *Java*$_G$ da 0. Prestare attenzione nella gestione degli array (e delle rispettive tabelle *SQL*$_G$).

A Glossario

A

Abstract Factory

Design pattern$_G$ creazionale che fornisce un'interfaccia per creare famiglie di oggetti connessi o dipendenti tra loro.

Actor

Elemento del *Diagramma di Robustezza*$_G$ che rappresenta l'utente che interagisce con gli altri elementi del sistema.

API

Acronimo di "Application Programming Interface", indica un insieme di procedure disponibili al programmatore, solitamente raggruppate in un set di strumenti specifici per il raggiungimento di un determinato compito all'interno di un certo programma.

Apple OSX

Sistema operativo sviluppato da Apple Inc. per i computer Macintosh.

Applicazione web

Indica genericamente un'applicazione distribuita, basata su tecnologie per il web e accessibile via web per mezzo di un network.

B

Back-end

Denota la parte di un sistema software che permette l'effettivo funzionamento delle interazioni effettuate dall'utente sull'interfaccia grafica e comunicate al sistema dal *Front-end*$_G$.

Backbone.js

Libreria *JavaScript*$_G$ che fornisce gli strumenti di base per dare una struttura alle *applicazioni web*$_G$, e basa la sua architettura su modelli *MVC*$_G$ con un componente Controller non tradizionale.

Boundary

Chiamata anche interfaccia, rappresenta elementi software come schermate, report, pagine *HTML*$_G$ o interfacce di sistema con cui gli *actor*$_G$ interagiscono.

C

Client

Una qualunque componente che accede ai servizi o alle risorse messe a disposizione da un'altra componente detta *server*$_G$.

Control

Chiamato anche "Controller", implementa la logica necessaria per gestire i vari elementi e le loro interazioni fungendo da collante tra oggetti *boundary*$_G$ ed *entity*$_G$.

CRUD

Acronimo di "Create Read Update Delete", indica le quattro funzioni di base di un sistema informatico che associa utente e risorse.

CSS

Acronimo di "Cascading Style Sheets", è un linguaggio usato per definire la formattazione e lo stile di documenti *HTML*$_G$ come i siti e le pagine web.

CSS3

Versione 3 del linguaggio *CSS*$_G$ che rispetto alla versione precedente presenta migliorie e funzionalità aggiornate.

D

Dao

Acronimo di Data Access Object, indica una classe che possiede i metodi per rappresentare un'*entità*$_G$ in un database relazionale, creando un maggiore livello di astrazione ed una più facile manutenibilità.

Database

Insieme organizzato di dati, strutturati e collegati tra loro secondo un determinato modello logico. Possono adottare un modello relazionale ove i dati sono salvati su tabelle interconnesse tra loro, oppure un modello non relazionale che salva i dati su documenti strutturati (ad esempio su *file*$_G$ *JSON*$_G$).

Design pattern

Soluzione progettuale ricorrente per la risoluzione di problemi durante la fase di progettazione e sviluppo software.

Diagramma di robustezza

Diagramma *UML*$_G$ semplificato che utilizza dei simboli grafici per rappresentare 5 concetti principali: *Actor*$_G$, *Boundary*$_G$, *Control*$_G$, *Entity*$_G$, Linea di associazione.

Directory

Contenitore di *file*$_G$ con le stesse funzionalità, per permettere una migliore organizzazione di questi e quindi una maggior usabilità da parte di utenti e programmi.

Drag and Drop

Questa nomenclatura in un'interfaccia di un computer indica una successione di tre azioni, consistenti nel cliccare su un oggetto virtuale (quale una finestra o un'icona) per trascinarlo in un'altra posizione, dove infine viene rilasciato.

E

Entity

Insieme di elementi che hanno proprietà comuni ed esistenza autonoma ai fini dell'applicazione di interesse.

ECMAScript

Linguaggio di programmazione standardizzato e mantenuto dalla Ecma International. Fra le sue implementazioni più note è doveroso citare *JavaScript*$_G$.

ECMAScript 2017

L'ottava versione rilasciata da *ECMAScript*$_G$ nel 2017 (anche chiamata "ES8") che introduce svariate funzionalità per la concorrenza, la gestione delle classi e molto altro.

Express

Framework$_G$ *open source*$_G$ per *Node.js*$_G$ utilizzato per lo sviluppo di *applicazioni web*$_G$.

F

File

Viene utilizzato per riferirsi a un contenitore di informazioni/dati in formato digitale, tipicamente presenti su un supporto digitale di memorizzazione opportunamente formattato in un determinato *file*$_G$-system.

Framework

Architettura logica di supporto (spesso un'implementazione logica di un particolare *design pattern*$_G$) composta da componenti creati per essere sfruttati da altre applicazioni per adempiere ad una serie di compiti che il programmatore non dovrà così preoccuparsi di implementare.

Front-end

Parte di un sistema software che gestisce l'interazione con l'utente o con sistemi esterni che producono flusso di dati in ingresso.

G

GET

Metodo $HTTP_G$ utilizzato per richiedere dati da una specifica risorsa.

Google Chrome

Browser web sviluppato da Google, oggigiorno ricopre una delle prime posizioni tra i browser web più utilizzati a livello mondiale.

H

Hibernate

Piattaforma middleware *open source*$_G$ per lo sviluppo di applicazioni *Java*$_G$ che fornisce un servizio di *ORM*$_G$, ovvero gestisce la persistenza dei dati sul database attraverso la rappresentazione e il mantenimento su database relazionale di un sistema di oggetti *Java*$_G$.

HTML

Acronimo di "HyperText *Markup*$_G$ Language", è un linguaggio di *markup*$_G$ utilizzato principalmente per la creazione della struttura di documenti destinati al web.

HTML5

Versione 5 del linguaggio *HTML*$_G$ che rispetto alla versione precedente presenta migliorie e funzionalità aggiornate.

HTTP

Acronimo di "HyperText Transfer Protocol", è il principale protocollo per lo scambio delle informazioni su internet tra *client*$_G$ e *server*$_G$.

I

IDE

Acronimo di "Integrated Development Environment", è un software che assiste il programmatore nello sviluppo del proprio software.

InnoDB

Motore per il salvataggio di dati per *MySQL*$_G$.

J

Java

Linguaggio di programmazione ad alto livello, orientato agli oggetti e a tipizzazione statica, specificatamente progettato per essere il più possibile indipendente dalla piattaforma di esecuzione.

JavaScript

Linguaggio orientato agli oggetti e agli eventi utilizzato nella programmazione Web.

JDBC

Acronimo di "*Java*$_G$ DataBase Connectivity", è un driver per database che consente l'accesso e la gestione della persistenza dei dati da qualsiasi programma in linguaggio *Java*$_G$, indipendentemente dal tipo di database utilizzato.

JointJS

Libreria *JavaScript*$_G$ utilizzata per creare diagrammi.

jQuery

Libreria *JavaScript*$_G$ che semplifica la selezione, la manipolazione e la gestione degli eventi di elementi *HTML*$_G$.

JSON

Acronimo di "*JavaScript*$_G$ Object Notation", è un formato per lo scambio dei dati tra applicazioni.

L

Local Storage

Area di memoria dove vengono memorizzati dati salvati da un'*applicazione web*$_G$ anche dopo la chiusura del browser. Tale memoria è accessibile e condivisa da qualsiasi script eseguito all'interno della stessa combinazione di protocollo, host, e numero di porta.

Lodash

Libreria *JavaScript*$_G$ che aiuta i programmatori a scrivere codice più facile e conciso. Viene utilizzata da *JointJS*$_G$ come dipendenza.

LTS

Acronimo di "Long-Term Support", è una tipologia speciale di versione per un prodotto software e serve ad indicare quando è previsto un supporto per un periodo di tempo maggiore del normale.

M

Markup

Regole che descrivono i meccanismi di rappresentazione di un testo che, seguendo convenzioni standardizzate, sono utilizzabili su più supporti.

Microsoft Edge

Browser web sviluppato da Microsoft e incluso in Windows 10, sostituendo Internet Explorer come browser predefinito di Windows.

Microsoft Windows

Sistema operativo sviluppato dalla Microsoft Corporation.

Mozilla Firefox

Browser web libero e multipiattaforma, mantenuto dalla Mozilla Foundation.

MVC

Acronimo di "Model-View-Controller", è un pattern architetturale, usato nello sviluppo di sistemi software, in grado di separare la logica di presentazione dei dati dalla logica applicativa.

MySQL

Sistema di gestione di database relazionale composto da un *client*$_G$ a riga di comando e un *server*$_G$.

N

Node.js

Web *server*$_G$ *open source*$_G$, fornisce un *framework*$_G$ basato su *JavaScript*$_G$ utilizzato per la gestione degli eventi.

npm

Insieme di pacchetti che mettono a disposizione di *Node.js*$_G$ le librerie più importanti.

O

Open source

Software di cui gli autori rendono pubblico il codice sorgente, permettendo ai programmatori di apportarvi modifiche ed estensioni.

Opera

Browser web freeware e multipiattaforma, prodotto da Opera Software.

ORM

Acronimo di "Object-Relational Mapping", è un tecnica di programmazione che favorisce l'integrazione tra software che utilizzano linguaggi orientati ad oggetti e database relazionali, riducendo la complessità del codice.

P

POJO

Acronimo di "Plain Old *Java*$_G$ Object", indica un oggetto semplice costruito in linguaggio *Java*$_G$, la cui classe non utilizza estensioni o implementazioni di interfacce o annotazioni particolari.

POST

Metodo *HTTP*$_G$ utilizzato per inviare dati al *server*$_G$ per creare o aggiornare risorse.

R

Redirect

Azione che permette all'applicazione di tornare alla pagina precedente, generalmente usato in caso di URL sbagliati o di risorse non trovate.

REST

Acronimo di "Representational State Transfer", è un'architettura software per sistemi distribuiti basato su *HTTP*$_G$.

Routing

Metodo per gestire le chiamate e le risposte al *server*$_G$ attraverso la definizione dell'*URI*$_G$. Deriva quindi da uno dei metodi di *HTTP*$_G$.

S

Safari

Browser web sviluppato dalla Apple Inc. per i sistemi operativi iOS e macOS.

Server

Insieme di componenti per l'elaborazione e la gestione del traffico di informazioni attraverso una rete di computer o un sistema informatico.

Singleton

Design pattern$_G$ creazionale che ha lo scopo di garantire che di una determinata classe venga creata una ed una sola istanza e di fornire un punto di accesso globale a tale istanza.

Spectrum

Libreria *JavaScript*$_G$ che permette la realizzazione di color picker per la scelta di colori funzionante su svariati browser uniformandone il comportamento.

SQL

Acronimo di "Structured Query Language", linguaggio di programmazione per la gestione e l'amministrazione di database relazionali.

SVG

Acronimo di "Scalable Vector Graphics", tecnologia in grado di visualizzare oggetti di grafica vettoriale e, pertanto, di gestire immagini scalabili dimensionalmente.

T

Template

Inteso come *file*$_G$, è appunto un *file*$_G$ che contiene dati e parametri, utilizzato quando si devono generare *file*$_G$ con gli stessi dati a cui vengono però sostituiti i parametri con delle informazioni specifiche.

Try-with-resource

Costrutto particolare di tipo try che dichiara le risorse aperte e utilizzate e si assicura che esse vengano correttamente chiuse alla fine dell'esecuzione.

Two-Tier

Design pattern$_G$ architetturale nel quale la parte di presentazione e quindi l'interfaccia lavorano sul *client*$_G$ mentre lo strato dei dati viene elaborato dal *server*$_G$, definendo quindi una struttura *client*$_G$ *server*$_G$. Viene utilizzato per separare le funzionalità di un sistema.

U

Ubuntu

Sistema operativo distribuito liberamente con licenza GNU GPL proveniente da Debian GNU/Linux.

UML

Acronimo di "Unified Modeling Language", è un linguaggio di modellazione e specifica orientato agli oggetti basato su una notazione semi-grafica e semi-formale.

URI

Sequenza di caratteri che identifica univocamente una risorsa generica rendendola disponibile tramite dei protocolli di comunicazione, quale *HTTP*$_G$.

X

XHTML

Linguaggio di *markup*$_G$ che unisce le proprietà del linguaggio *HTML*$_G$ e *XML*$_G$. Deriva dalla versione 4.1 di *HTML*$_G$.

XML

Acronimo di "eXtensible *Markup*$_G$ Language", è un metalinguaggio per la definizione di linguaggi di *markup*$_G$ basato su un meccanismo sintattico che consente di definire e controllare il significato degli elementi contenuti in un documento.