Agents of S.W.E.

A SOFTWARE COMPANY

Agents of S.W.E. - Progetto "G&B"

Manuale dello Sviluppatore

Versione	0.1.1
Approvazione	??
Redazione	Marco Chilese Luca Violato
Verifica	Favaro Marco
Stato	Work in Progress
Uso	Esterno
Destinato a	Agents of S.W.E. Prof. Tullio Vardanega Prof. Riccardo Cardin Zucchetti S.p.A.

agentsofswe@gmail.com

Registro delle Modifiche

Versione	Data	Ruolo	Autore	Descrizione
0.1.1	2019-04-10	Progettista	Bogdan Stanciu	Raffinamento capitolo §6.2, §3.1, §3.1.2
0.1.0	2019-04-10	Progettista	Marco Favaro	Verifica documento
0.0.3	2019-04-09	Programmatore	Marco Chilese	Stesura di §1, §1.3, §2, §3.1, §3, §3.2, §5.2, §5.5, §6.3, §6.4, §8
0.0.2	2019-04-06	Progettista	Luca Violato	Aggiunte sezioni §2 e §6
0.0.1	2019-04-04	Verificatore	Marco Chilese	Strutturazione del Documento. Prima stesura di §4.1, §7.1, §5.1, §5.3, §7.2, §7.3, §3.2

Tabella 1: Registro delle Modifiche

Indice

Elenco delle tabelle

Elenco delle figure

1 Introduzione

1.1 Scopo del Documento

Lo scopo del documento è di fornire tutte le informazioni necessarie agli sviluppatori che intenderanno estendere o migliorare il plug-in `G&B`.
Verranno fornite informazioni relative anche ad un possibile ambiente di sviluppo il più completo possibile da cui sarà possibile partire. In particolare sarà illustrato l'ambiente di sviluppo utilizzato dal team `Agents of S.W.E.` per lo sviluppo del prodotto in oggetto.
La seguente guida può essere utilizzata indifferentemente sia da utenti Microsoft Windows, Linux e Apple MacOs.

1.2 Scopo del Prodotto

Lo scopo del prodotto è la creazione di un plug-in per la piattaforma open source$_G$ di visualizzazione e gestione dati, denominata *Grafana*$_G$, con l'obiettivo di creare un sistema di alert dinamico per monitorare la "liveliness"$_G$ del sistema a supporto dei processi DevOps$_G$ e per consigliare interventi nel sistema di produzione del software. In particolare, il plug-in$_G$ utilizzerà dati in input forniti ad intervalli regolari o con continuità, ad una rete bayesiana$_G$ per stimare la probabilità di alcuni eventi, segnalandone quindi il rischio in modo dinamico, prevenendo situazioni di stallo.

1.3 Riferimenti

1.3.1 Referimenti per l'Installazione

- *https://nodejs.org/it/*;
- *https://www.npmjs.com/*;
- *https://grafana.com/docs/installation/*.

1.3.2 Referimenti Legali

- *https://grafana.com/docs/contribute/cla/*.

1.3.3 Referimenti Informativi

- *https://grafana.com/*;
- *https://www.influxdata.com/time-series-platform/telegraf/*;

- *https://www.influxdata.com/time-series-platform/influxdb/*;
- *https://jestjs.io/*;
- *https://www.jetbrains.com/webstorm/*;
- *https://webpack.js.org/*.
- *https://angularjs.org/*;
- *http://codecov.io*;
- *https://jquery.com/*;
- *http://pm2.keymetrics.io/*;

2 Principali Tecnologie Utilizzate

2.1 Scopo del Capitolo

In questo capitolo verranno esposte e contestualizzate le tecnologie impiegate all'interno del progetto, in modo tale da offrire agli sviluppatori un quadro più completo del progetto.

2.2 Tecnologie Lato Client

Lato client vengono adottate le seguenti tecnologie:

- *EcmaScript 6*: principale linguaggio con cui il plug-in è strutturato, in particolar modo viene utilizzato per lo sviluppo del pannello;
- *AngularJS*: framework adottato da *Grafana* per l'interazione con l'utente;
- *HTML & CSS*: rispettivametne nelle versioni 5 e 3, utilizzati per modellare l'interfaccia del pannello.

2.3 Tecnologie Lato Server

Lato server vengono adottate le seguenti tecnologie:

- *NodeJS*: Tecnologia runtime javascript costruito sul motore V8$_G$ di Chrome$_G$ con la quale viene implementato il server
- *pm2*: Gestore dei processi avanzato per NodeJS. Fornisce inoltre supporto ai processo DevOps$_G$, di riavvio automatico al crash del processo e al log efficiente dei processi;
- *Grafana*: Ecosistema di monitoraggio & visualizzazione dati open-source.

2.4 Tecnologie per il Testing

La tecnologia utilizzata per testare il codice è *Jest*, framework di test per codice *JavaScript*.

3 Installazione

3.1 Requisiti

I requisiti minimi richiesti per il funzionamento del plug-in non sono dovuti al prodotto in sè, ma sono dovuti alle tecnologie che vengono utilizzate. Pertanto si rimanda ai requisiti minimi delle seguenti tecnologie:

- *InfluxDB*;
- *Grafana*;
- *NodeJS*;
- *pm2*.

3.1.1 Installazione plug-in

Viene richiesta come pre-condizione l'installazione di grafana a seconda del sistema operativo utilizzato seguendo la documentazione fornita da quest'ultima.

1. Arrestare l'esecuzione di *Grafana*, qualora fosse in esecuzione;
2. Copiare la directory del progetto all'interno della cartella predestinata da *Grafana* per ospitare i plug-in aggiuntivi da installare.
3. Per poter utilizzare il prodotto all'interno di *Grafana*, e poiché quest'ultimo sia riconosciuto come plug-in, è necessario per prima cosa eseguire la build del prodotto. La build del prodotto avviene attraverso due fasi: la prima che va a risolvere le dipendenze, la seconda che esegue la build.
   Il processo di build avviene attraverso l'intervento del module bundler *WebPack*. È necessario quindi eseguire i seguenti comandi dalla directory che ospita il plug-in:

   ```
   npm install
   npm run build
   ```

4. Avviare *Grafana* e procedere all'aggiunta del plug-in. Per informazioni relative al funzionamento del plug-in si rimanda al *Manuale Utente v1.0.0*.

3.1.2 Installazione server

3.1.2.1 Installazione NodeJS Per quanto riguarda l'installazione di *NodeJS* si rimanda al sito del produttore.

3.1.2.2 Installazione pm2 Per quanto riguarda l'installazione di *pm2* sarà necessario eseguire da terminale il seguente comando:

```
npm install pm2 -g
```

3.2 Esecuzione

3.2.1 Esecuzione plug-in

- Aprire il browser consigliato;
- Collegarsi all'indirizzo ip di grafana sulla porta 3000 (le configurazione di default suggeriscono `http://localhost:3000`);
- Accedere al sistema con le credenziali pre-impostate;
- Nella dashboard selezionata, aggiungere il pannello *"GrafanaAndBayes"*;
- Accedere alle impostazioni del plug-in inserendo l'indirizzo ip e la porta del server in ascolto.

3.2.2 Esecuzione server

- Aprire un terminale e spostarsi nella directory contenente il server;
- Da terminale, eseguire il seguente commando il quale avvia il server attraverso il manager dei processi pm2

  ```
  pm2 start ./src/index.js -- -p :porta
  ```

  dove `:porta` e il numero della porta su cui impostare il server in ascolto;

4 Impostare l'Ambiente di Lavoro

4.1 Scopo del Capitolo

In questa sezione viene riportata una guida per la corretta configurazione dell'ambiente di sviluppo, in modo che sia la stessa utilizzata dal gruppo `Agents of S.W.E.`.
Per contribuire al progetto non è strettamente necessario seguire questa sezione, tuttavia è consigliato al fine di ottenere un ambiente di lavoro pronto e correttamente impostato per lo sviluppo.

5 Requisiti

Per i requisiti minimi di funzionamento dell'ambiente di lavoro consigliato, si rimanda al sito del produttore di ogni singola tecnologia.

5.1 *WebStorm*

WebStorm è l'ambiente di sviluppo integrato (IDE) di *JetBrains* utilizzato dal team per lo sviluppo del progetto. Esso può essere ottenuto mediante download dal sito ufficiale nella formula di prova gratuita se non si dispone di licenza, che può essere ottenuta attraverso l'indirizzo email universitario.
Tale software è disponibile per i sistemi operativi Microsoft Windows, Linux e Apple MacOS.
Per ulteriori informazioni si rimanda al sito ufficiale.

5.2 *NPM*

NPM è il gestore di pachetti utilizzato per gestire il progetto dal team `Agents of S.W.E.`. Per la relativa installazione si rimanda al sito del produttore.
Per l'effettivo utilizzo è necessario avere all'interno della directory del progetto un file denominato "package.json". Esso permette di esprimere le dipendenze e le caratteristiche del prodotto, oltre che alla definizione dei vari comandi da eseguire con radice "`npm run ...`".
Di seguito è riportato, nei punti salienti, il file "package.json" utilizzato durante lo sviluppo del progetto:

```
1 {
2   "name": "GrafanaAndBayes",
```

```
3   "version": "2.0.0",
4   "description": "Plug-in␣per␣Grafana",
5   "main": "src/module.js",
6   "scripts": {
7     "test": "jest",
8     "build": "webpack␣--config␣build/webpack.prod.conf.js",
9     "dev": "webpack␣--mode␣development",
10    "eslint": "eslint␣./src",
11     "codecov": "codecov"
12  },
13  "jest": {
14    "verbose": true,
15    "collectCoverage": true,
16    "coverageDirectory": "./coverage/"
17  },
18  "author": "Agents␣Of␣S.W.E.",
19  "license": "ISC",
20  "devDependencies": {
21    ...
22  },
23  "dependencies": {
24    ...
25  }
26 }
```

5.3 *ESLint*

ESLint verrà installato automaticamente attraverso il comando `npm install`.
Per abilitarlo all'interno di *WebStorm* è necessario lanciare l'IDE e recarsi in: File > Settings > ESLint e scegliere "Enable". All'interno del campo "Node Interpreter" è necessario inserire il percorso alla directory in cui si trovano i file eseguibili di Node. Se non rilevato in automatico, specificare la posizione del file di configurazione ".eslintrc" all'interno della directory del progetto.

5.4 *Jest*

Jest è il framework di test utilizzato per testare il codice *JavaScript*. È inoltre il responsabile della generazione dei report di coverage utilizzati in seguito per il calcolo del code coverage.
L'installazione di tale componente consta nell'aggiunta del relativo pacchetto *npm*. In particolare si dovranno eseguire i seguenti comandi:

`npm install -save-dev jest`.

A ciò seguirà poi l'aggiunta del relativo comando di script all'interno di "package.json" in modo tale da consentire l'esecuzione dei test all'invocazione del comando `npm run test`. La modifica da apportare al file sopracitato, e sopra riportato, è visibile alla riga 7:

```
"test":  "jest",.
```

5.5 *Webpack*

Webpack è il model bundler utilizzato per eseguire la build del progetto.
L'installazione di tale componente avviene tramite *NPM*. È necessario eseguire il seguente comando:

`npm install -save-dev webpack`.

In seguito è necessario aggiungere una riga al file "package.json", come riportato in riga 8:

```
"build":  "webpack -config build/webpack.prod.conf.js",.
```

6 Architettura

6.1 Scopo del Capitolo

Il seguente capitolo ha lo scopo di fornire le informazioni necessarie allo sviluppatore per potersi interfacciare con il prodotto, in modo tale da rendere più agevole l'ampliamento e la modifica.

6.2 Visione Generale

Il prodotto si basa su 3 componenti chiave:

- Il cliente, ovvero il plug-in per Grafana;
- Il server, il quale agisce da controller;
- Il database che fornisce il model.

Queste tre componenti unite formano il prodotto finale. Esse cambiano messaggi e informazioni una con le altre seguendo un pattern meglio conosciuto come MVC_G

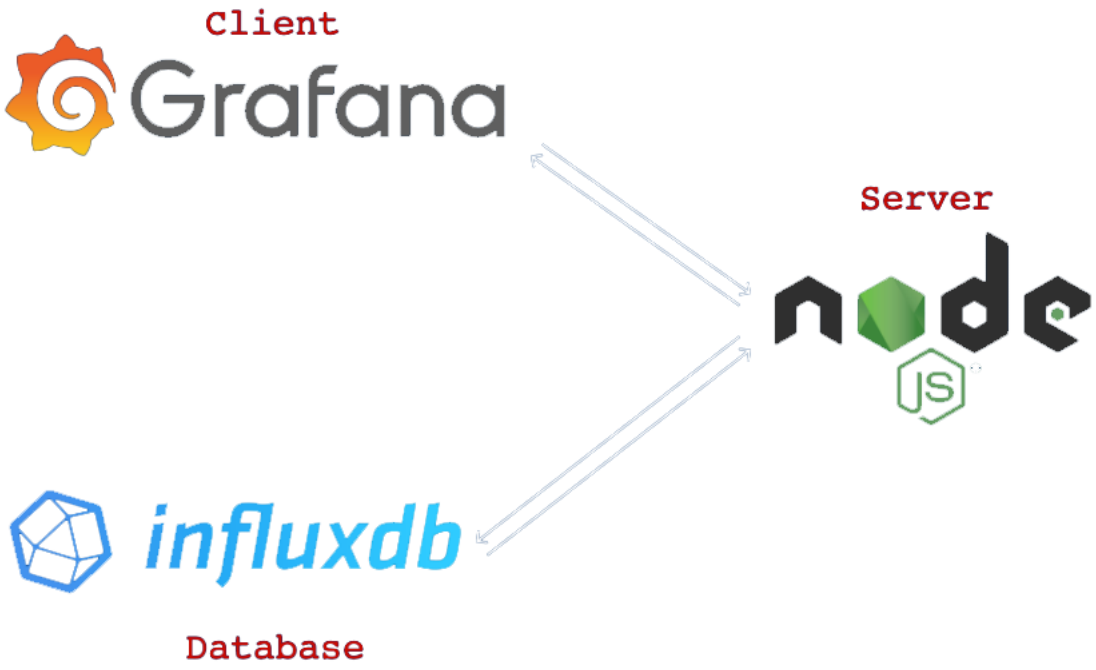

Figura 1: Architettura dell'applicativo

6.3 Pannello

Per rendere lo sviluppo più semplice, e garantire la manutenibilità del codice il team ha optato per un approccio modulare. In questo modo, avendo moduli separati con compiti distinti, sarà più semplice modificarne o estenderne il comportamento senza dover necessariamente modificare la base comune.
In particolare i moduli separati sono:

- **GBCtrl**: è il modulo principale che utilizza gli altri e svolge le operazioni fondamentali;
- **TemporalPolicyCtrl**: è il modulo che si occupa di gestire ed impostare le politiche temporali all'interno del pannello;
- **ThresholdsCtrl**: è il modulo che si occupa di gestire ed impostare le soglie per il monitoraggio;
- **Parser**: è il modulo che si occupa di controllare e interpretare la rete bayesiana in input, in formato *JSON*;
- **ConnectServerProxy**: è il modulo che si occupa di inoltrare le richieste al server preoccupandosi che i dati siano strutturati nel modo corretto e ricevere i dati;
- **ModalCreator**: è il modulo che si occupa di visualizzare le finestre che permettono l'interazione con l'utente;
- **GetAPIGrafana**: è il modulo che si occupa di interagire con le API di *Grafana* per ottenere i dati relativi alle sorgenti dati.

6.3.1 UML - Diagramma dei Package

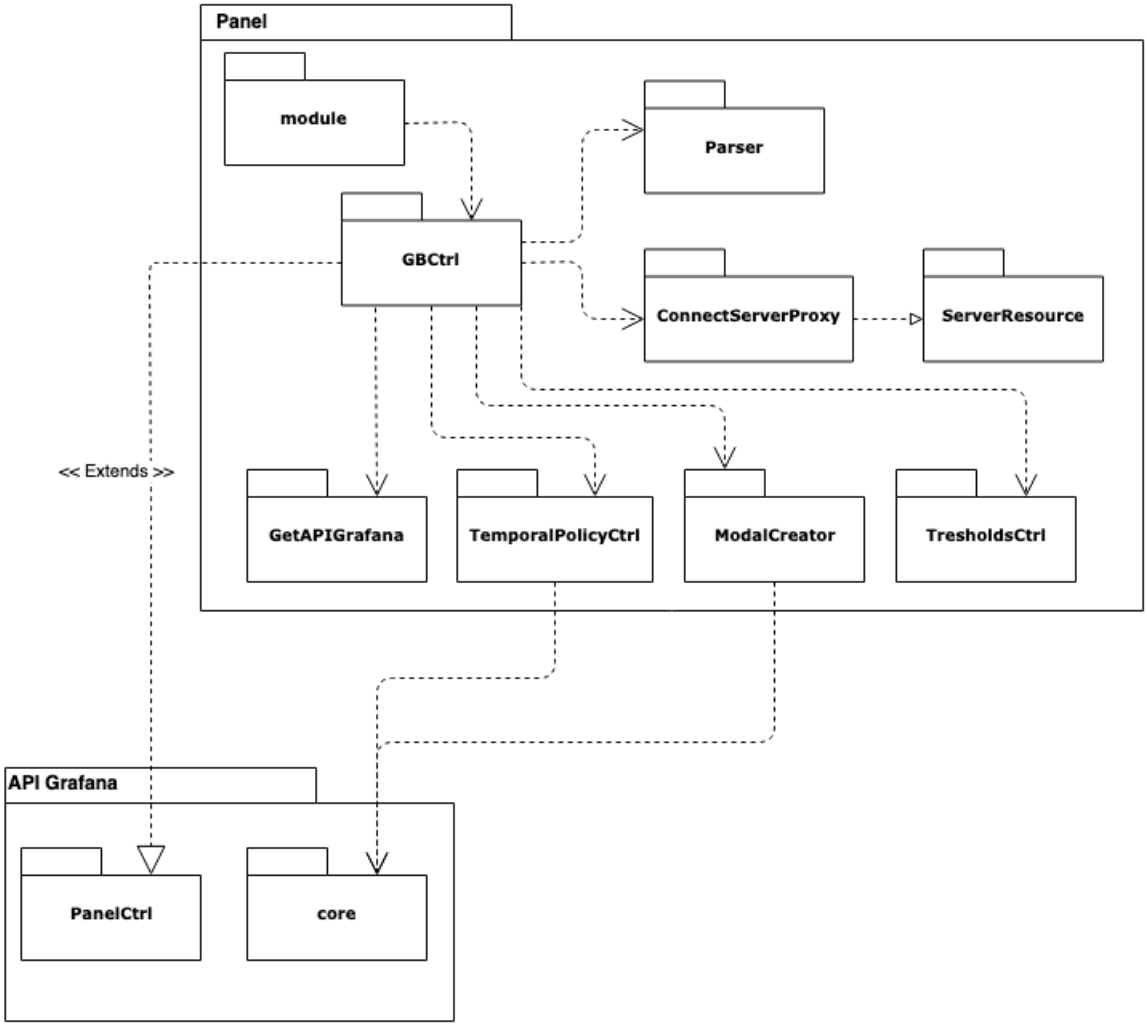

Figura 2: Diagramma dei Package del Pannello

6.3.2 UML - Diagramma delle Classi

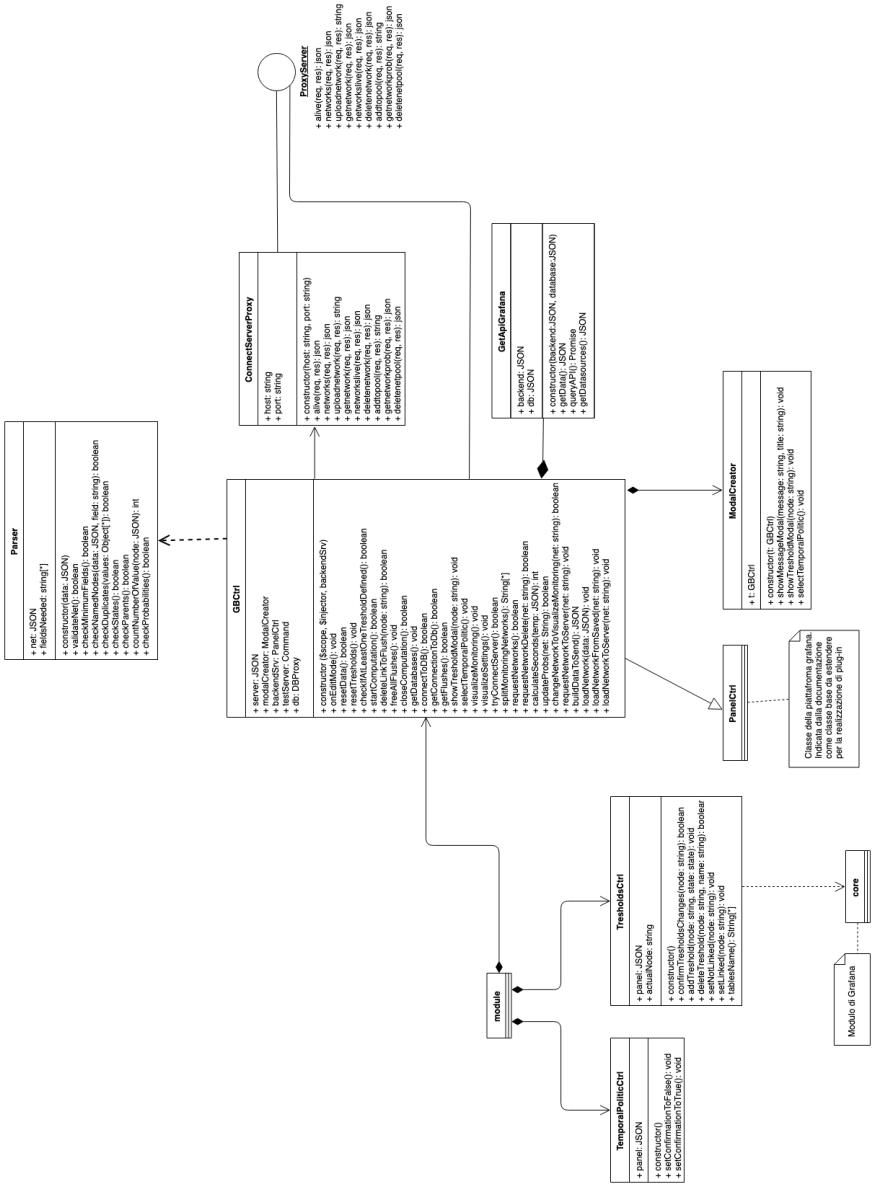

Figura 3: Diagramma delle Classi del Pannello

6.4 Server

Per rendere lo sviluppo più semplice, e garantire la manutenibilità del codice il team ha optato per un approccio modulare. In questo modo, avendo moduli separati con compiti distinti, sarà più semplice modificarne o estenderne il comportamento senza dover necessariamente modificare la base comune.
In particolare i moduli presenti per quanto riguarda lo sviluppo del server sono:

- **Server**: è il modulo principale che si occupa di comunicare con gli altri, ed esegue le operazioni fondamentali;
- **InfluxDB**: è il modulo che si occupa di adattare la libreria *Influx* e ci fornisce un set di metodi specializzati;
- **Network**: è il modulo che si occupa di adattare la libreria *jsbayes* e costruisce la rete bayesiana da utlizzare;
- **ProxyServer**: è il modulo che si occupa di filtrare, autenticare le richieste verso il server e protegge lo stesso.

6.4.1 UML - Diagramma dei Package

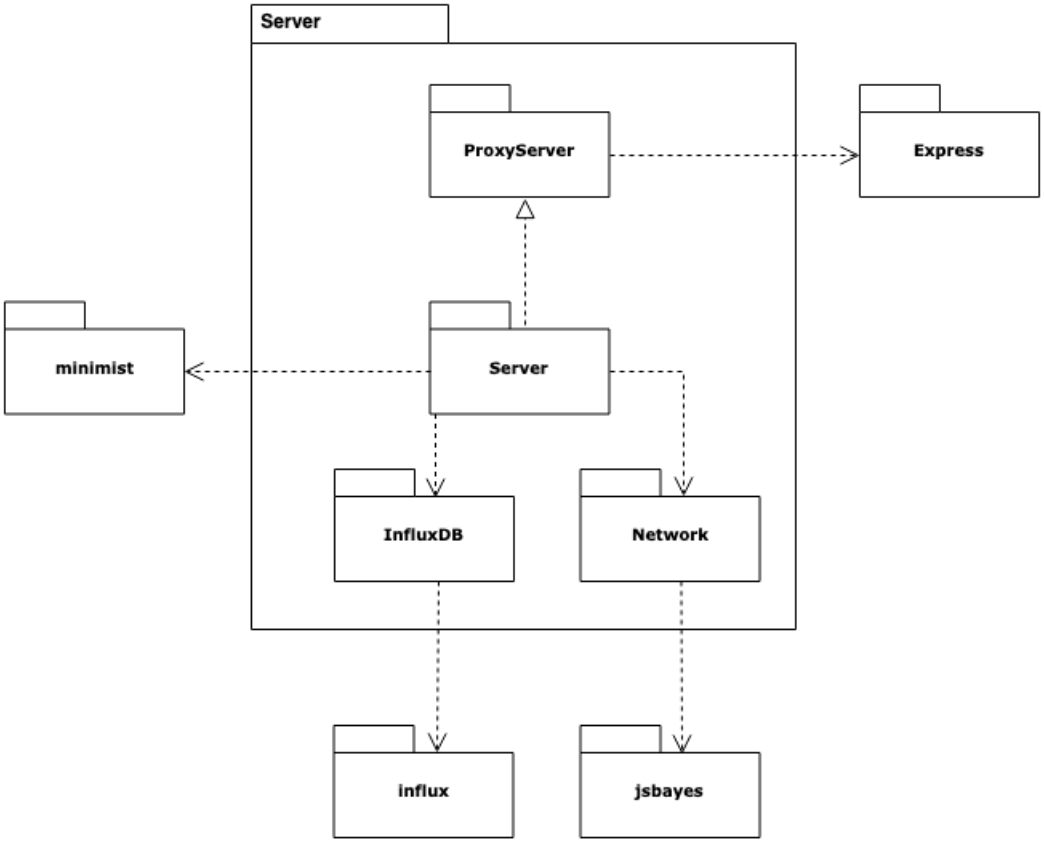

Figura 4: Diagramma dei Package del Server

6.4.2 UML - Diagramma delle Classi

Figura 5: Diagramma delle Classi del Server

7 Test

7.1 Scopo del Capitolo

Questo capitolo ha lo scopo di indicare agli sviluppatori come controllare in modo automatico il proprio codice e la sintassi.

7.2 Test sul Codice *JavaScript*

Per eseguire i test sul codice è necessario eseguire i seguenti comandi:

`npm run test`

o

`jest`.

Per verificare che il codice sia coerente con le linee guida adottate è necessario eseguire:

`npm run eslint`.

Per correggere in modo automatico alcuni dei potenziali problemi, eseguire:

`npm run eslint-fix`.

Questi script eseguono un controllo del codice all'interno di `"./src"`, la directory dov'è contenuto il codice.
Se si desidera eseguire *ESLint* su un unico file, si rimanda il lettore alla documentazione ufficiale.

7.3 Code Coverage

Per eseguire il controllo di copertura del codice *JavaScript*, è necessario eseguire il seguente comando:

`npm run codecov`.

Nel caso tale comando fallisca la principale motivazione è data dall'assenza di report su cui generare il code coverage. Per rimediare a ciò basterà eseguire il primo comando nella sezione §7.2 il quale provvederà a generare i report necessari.

8 Licenza