Agents of S.W.E.

A SOFTWARE COMPANY

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

Manuale dello Sviluppatore

Versione	1.1.0
Approvazione	Luca Violato
Redazione	Marco Chilese Luca Violato Diego Mazzalovo
Verifica	Marco Favaro
Stato	Approvato
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
1.1.0	2019-05-15	Verificatore	Luca Violato	Verifica documento
1.0.7	2019-05-14	Programmatore	Diego Mazzalovo	Stesura sezione §7.2
1.0.6	2019-05-14	Verificatore	Bogdan Stanciu	Aggiunta termini in glossario
1.0.5	2019-05-13	Verificatore	Bogdan Stanciu	Stesura sezione §5.4.2
1.0.4	2019-05-13	Verificatore	Favaro Marco	Inserimento sezione §B
1.0.3	2019-05-13	Verificatore	Bogdan Stanciu	Sistemazione paragrafo §5.2, suddivisione metodi per ogni classe panello
1.0.2	2019-05-10	Programmatore	Diego Mazzalovo	Stesura §5.3.2
1.0.1	2019-04-24	Analista	Marco Chilese	Ristrutturazione §4, approfondimento §3.1 e §4.2.1, aggiunta sezione §7
1.0.0	2019-04-12	Responsabile	Luca Violato	Approvazione documento per rilascio RQ
0.2.0	2019-04-11	Verificatore	Diego Mazzalovo	Verifica documento
0.1.1	2019-04-10	Progettista	Bogdan Stanciu	Raffinamento capitolo §5.2, §3.1, §3.3
0.1.0	2019-04-10	Progettista	Marco Favaro	Verifica documento

Versione	Data	Ruolo	Autore	Descrizione
0.0.3	2019-04-09	Programmatore	Marco Chilese	Stesura di §1, §1.3, §2, §3.1, §3, §3.4, §4.2.3, §4.2.6, §5.3, §5.4, §A
0.0.2	2019-04-06	Progettista	Luca Violato	Aggiunte sezioni §2 e §5
0.0.1	2019-04-04	Verificatore	Marco Chilese	Strutturazione del Documento. Prima stesura di §4.1, §6.1, §4.2.2, §4.2.4, §6.2, §6.3, §3.4

Tabella 1: Registro delle Modifiche

Indice

1 Introduzione **6**
1.1 Scopo del Documento . . . 6
1.2 Scopo del Prodotto . . . 6
1.3 Riferimenti . . . 6
1.3.1 Referimenti per l'Installazione . . . 6
1.3.2 Referimenti Legali . . . 6
1.3.3 Referimenti Informativi . . . 6

2 Principali Tecnologie Utilizzate **8**
2.1 Scopo del Capitolo . . . 8
2.2 Tecnologie Lato Client . . . 8
2.3 Tecnologie Lato Server . . . 8
2.4 Tecnologie per il Testing . . . 8

3 Installazione **9**
3.1 Requisiti . . . 9
3.2 Installazione del Plug-in . . . 10
3.3 Installazione del Server . . . 10
3.3.1 Installazione *NodeJS* . . . 10
3.3.2 Installazione *Pm2* . . . 10
3.4 Esecuzione . . . 11
3.4.1 Esecuzione del Plug-in . . . 11
3.4.2 Esecuzione del Server . . . 11

4 Impostare l'Ambiente di Lavoro **12**
4.1 Scopo del Capitolo . . . 12
4.2 Tecnologie dell'Ambiente di Lavoro . . . 12
4.2.1 Requisiti . . . 12
4.2.2 *WebStorm* . . . 13
4.2.3 *NPM* . . . 13
4.2.4 *ESLint* . . . 14
4.2.5 *Jest* . . . 14
4.2.6 *Webpack* . . . 14

5 Architettura **15**
5.1 Scopo del Capitolo . . . 15
5.2 Visione Generale . . . 15

5.2.1 Architettura MVC . . . 15
5.2.2 UML . . . 16
5.3 Pannello . . . 18
5.3.1 UML . . . 18
5.3.2 Descrizione delle Classi e dei Metodi del Pannello . . . 19
5.3.2.1 GBCtrl . . . 19
5.3.2.2 TemporalPolicyCtrl . . . 22
5.3.2.3 TresholdsCtrl . . . 22
5.3.2.4 Parser . . . 23
5.3.2.5 ConnectServer . . . 24
5.3.2.6 ModalCreator . . . 24
5.3.2.7 GetApiGrafana . . . 25
5.4 Server . . . 26
5.4.1 UML . . . 26
5.4.2 Descrizione delle Classi e dei Metodi del Server . . . 27
5.4.2.1 Server . . . 27
5.4.2.2 Network . . . 29
5.4.2.3 InfluxDB . . . 29

6 Test **31**
6.1 Scopo del Capitolo . . . 31
6.2 Test sul Codice *JavaScript* . . . 31
6.3 Code Coverage . . . 31

7 Estensione delle Funzionalità **32**
7.1 Backend . . . 32
7.1.1 Aggiunta di una route nel server . . . 32
7.1.2 Aggiunta database . . . 33
7.1.3 Estensione funzionalità Network . . . 33
7.2 Pannello . . . 34
7.2.1 Modifica conseguente ad estensioni lato Server . . . 34
7.2.2 Estensione funzionalità indipendenti dal Server . . . 34

A Licenza **35**

B Glossario **36**

Elenco delle tabelle

1 Registro delle Modifiche 2

Elenco delle figure

1 Architettura dell'applicativo 15
2 Diagramma di package dell'applicativo 17
3 Diagramma delle classi del Pannello 18
4 Architettura dell'applicativo 26

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*$_{\text{G}}$: principale linguaggio con cui il plug-in è strutturato, in particolar modo viene utilizzato per lo sviluppo del pannello;
- *AngularJS*$_{\text{G}}$: framework adottato da *Grafana* per l'interazione con l'utente;
- *HTML*$_{G}$ *& CSS*$_{G}$: rispettivametne nelle versioni 5 e 3, utilizzati per modellare l'interfaccia del pannello;
- *Jsbayes Viz*$_{\text{G}}$: libreria in javascript utilizzata per visualizzare la rete bayesiana.

2.3 Tecnologie Lato Server

Lato server vengono adottate le seguenti tecnologie:

- *NodeJS*$_{\text{G}}$: Tecnologia runtime javascript costruito sul motore V8$_{\text{G}}$ di *Chrome*$_{\text{G}}$ con la quale viene implementato il server
- *pm2*: Gestore dei processi avanzato per *NodeJS*. Fornisce inoltre supporto ai processi DevOps$_{\text{G}}$, di riavvio automatico al crash del processo e al log efficiente dei processi;
- *Grafana*: Ecosistema di monitoraggio e visualizzazione dati open-source;
- *Jsbayes*$_{\text{G}}$: libreria in javascript utilizzata per il ricalcolo delle probabilità.

2.4 Tecnologie per il Testing

La tecnologia utilizzata per testare il codice è *Jest*$_{\text{G}}$, 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. Di seguito vengono riportati i requisiti minimi richiesti dalle seguenti tecnologie:

- *InfluxDB*:
  - CPU: 2-4 core;
  - RAM: 2-4GB;
  - IOPS: 500.
- *Grafana*:
  - CPU: 1 core;
  - RAM: 250MB.
- *NodeJS*:
  - Architettura: 64bit.
- Browser web:
  - *Google Chrome* versione 58 o superiore, per il supporto al linguaggio *EcmaScript6*;
  - *Microsoft Edge* versione 14 o superiore, per il supporto al linguaggio *EcmaScript6*;
  - *Mozilla Firefox* versione 54 o superiore, per il supporto al linguaggio *EcmaScript6*;
  - *Apple Safari* versione 10 o superiore, per il supporto al linguaggio *EcmaScript6*.

3.2 Installazione del Plug-in

Viene richiesta come pre-condizione l'installazione della piattaforma *Grafana*, a seconda del sistema operativo utilizzato, seguendo la documentazione fornita da quest'ultima.

Per l'installazione del Plug-in seguire i seguenti passaggi:

1. Arrestare l'esecuzione di *Grafana*, qualora fosse in esecuzione;
2. Copiare la directory del plug-in 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*$_{\text{G}}$. 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 v2.0.0*.

3.3 Installazione del Server

3.3.1 Installazione *NodeJS*

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

3.3.2 Installazione *Pm2*

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

```
npm install pm2 -g
```

3.4 Esecuzione

3.4.1 Esecuzione del 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.

3.4.2 Esecuzione del Server

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

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

  dove `:porta` è 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.

4.2 Tecnologie dell'Ambiente di Lavoro

Le tecnologie utilizzate dal gruppo `Agents of S.W.E.` sono le seguenti:

- *WebStorm*;
- *NPM*;
- *ESLint*;
- *Jest*;
- *WebPack*.

4.2.1 Requisiti

Per i requisiti minimi di funzionamento dell'ambiente di lavoro consigliato, si veda di seguito:

- *WebStorm*:
  - OS:
    - Apple macOS 10.8.3 o superiore;
    - Microsoft Windows 10/8/7/Vista/2003/XP (incl.64-bit);
      - CPU: Intel Pentium III/800 MHz o superiore, o compatibile.
    - Linux con GNOME o KDE desktop.
  - RAM: 2GB, raccomandati 4GB;
  - Risoluzione schermo: 1024x768.

4.2.2 *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.

4.2.3 *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 }
```

4.2.4 *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.

4.2.5 *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",.
```

4.2.6 *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",.
```

5 Architettura

5.1 Scopo del Capitolo

Il seguente capitolato ha lo scopo di fornire una visione del prodotto necessaria allo sviluppatore per potersi interfacciare con il prodotto, in modo tale da rendere più agevole l'ampliamento e la modifica.

5.2 Visione Generale

Il prodotto si basa su 3 componenti chiave:

- Il client, 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 scambiano messaggi e informazioni una con le altre seguendo un pattern meglio conosciuto come MVC_{G}

5.2.1 Architettura MVC

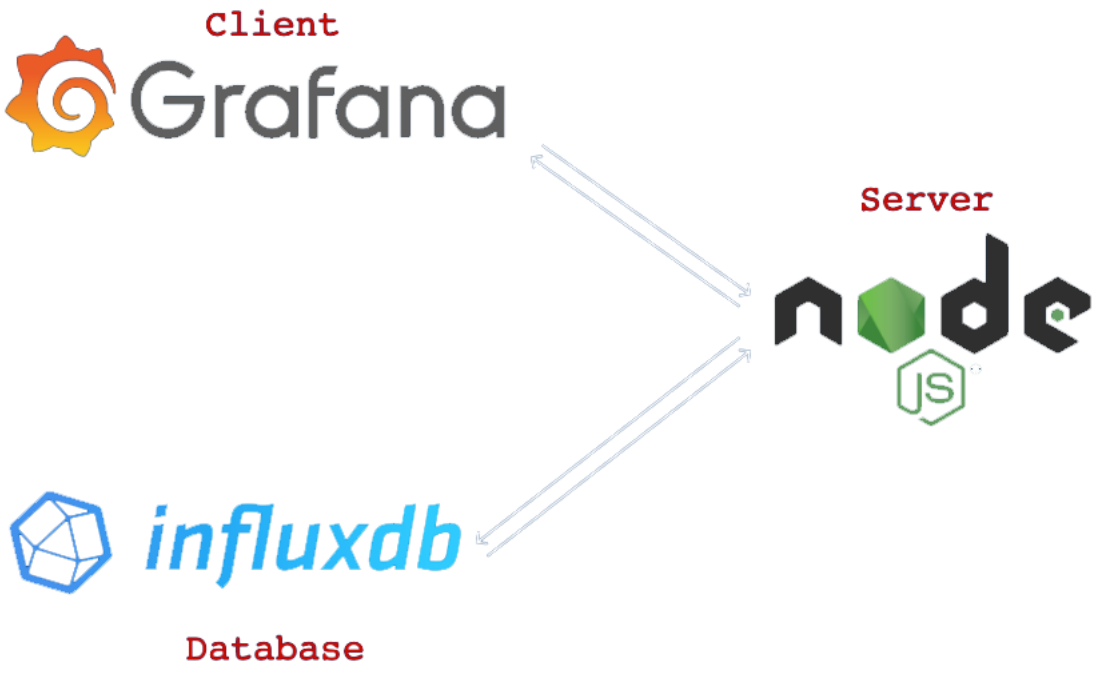

Figura 1: Architettura dell'applicativo

Il client comunica con il server attraverso il plug-in fornito dal sistema. Quest'ultimo, una volta connesso al server NodeJS il quale agirà da controller, ha a disposizione un set di funzionalità fornitoli tramite API_G . Le principali mansioni del client sono le seguenti:

- Verifica dei dati: il plug-in offre all'utente la possibilità di scegliere una ristretta selezione dei dati tramite dei menù tendina, il plug-in stesso eseguirà una prima verifica dei dati inseriti;
- Visualizzazione dei dati: il plug-in fornisce la funzionalità incorporata di visualizzazione della rete in monitoraggio, mostrando a pannello una rappresentazione grafica della rete selezionata attualmente in monitoraggio sul server;
- Gestione rete bayesiana: il plug-in prende in carico la rete bayesiana, in formato *.json*, caricata dall'utente, effettuando un'iniziale verifica statica dei campi. Successivamente viene incaricato dell'aggiunta di informazioni inserite e o modificate dall'utente;
- Eliminazione di reti: il plug-in richiede al server l'eliminazione di una rete scelta dall'utente.

La parte del controller e demandata al server, il quale ha il compito di filtrare le richieste ricevute dai vari plug-in connessi ad esso, salvare le reti bayesiane caricate verificandone la sintassi, calcolare le probabilità delle reti a seconda delle politiche temporali inserite e scrivere sul modello i dati calcolati. Fornisce delle api con le quali i vari plug-in possono interfacciarsi al server e modella la connessione al modello nel quale salvare i dati.
L'ultimo componente é il modello rappresentato dal database. Di default il prodotto arriva con la possibilità di interfacciarsi a un database di tipo InfluxDB. L'estensibilità o l'aggiunta del modello viene rimandata al paragrafo §7.1

5.2.2 UML

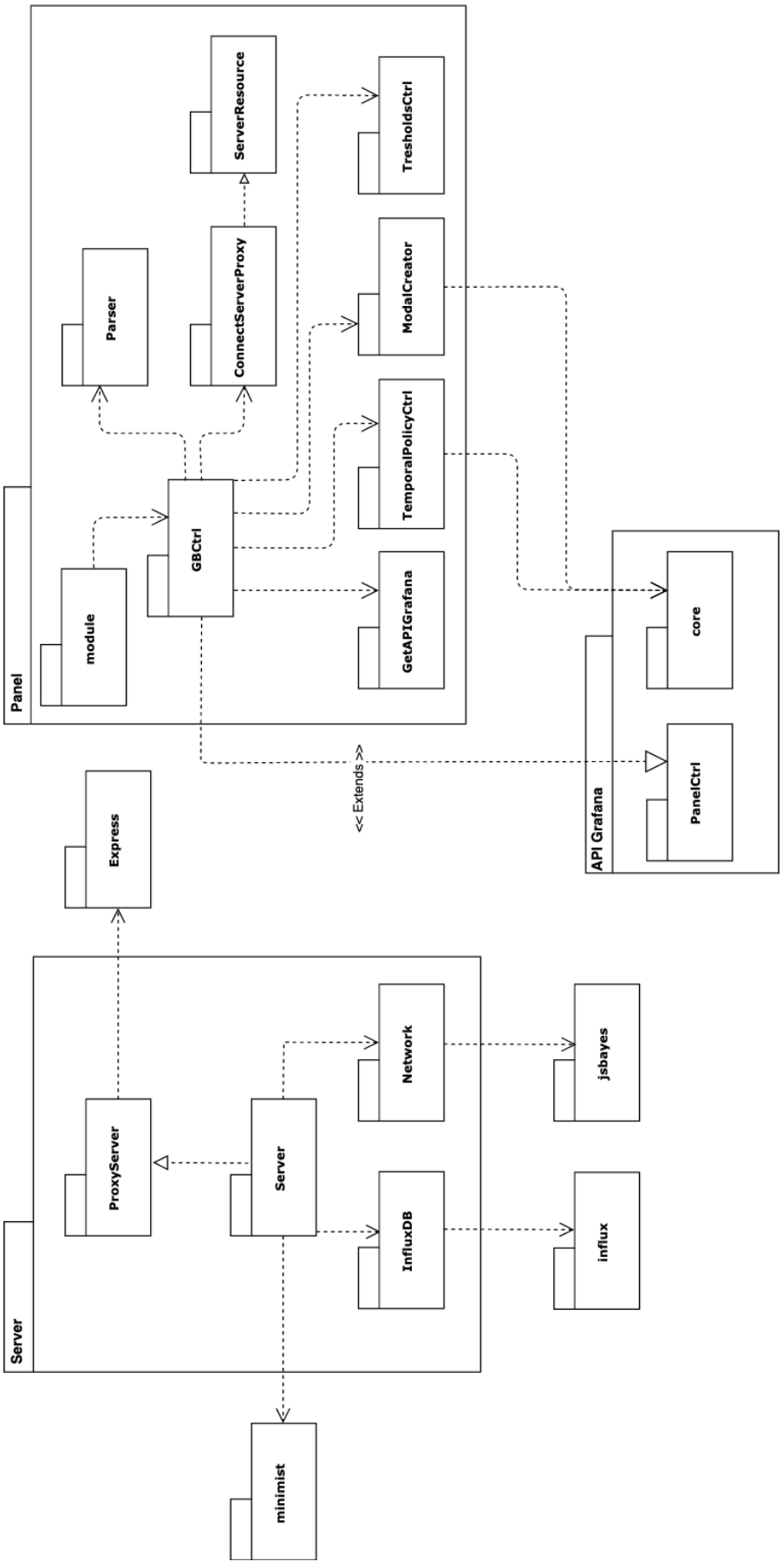

Figura 2: Diagramma di package dell'applicativo

5.3 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.

5.3.1 UML

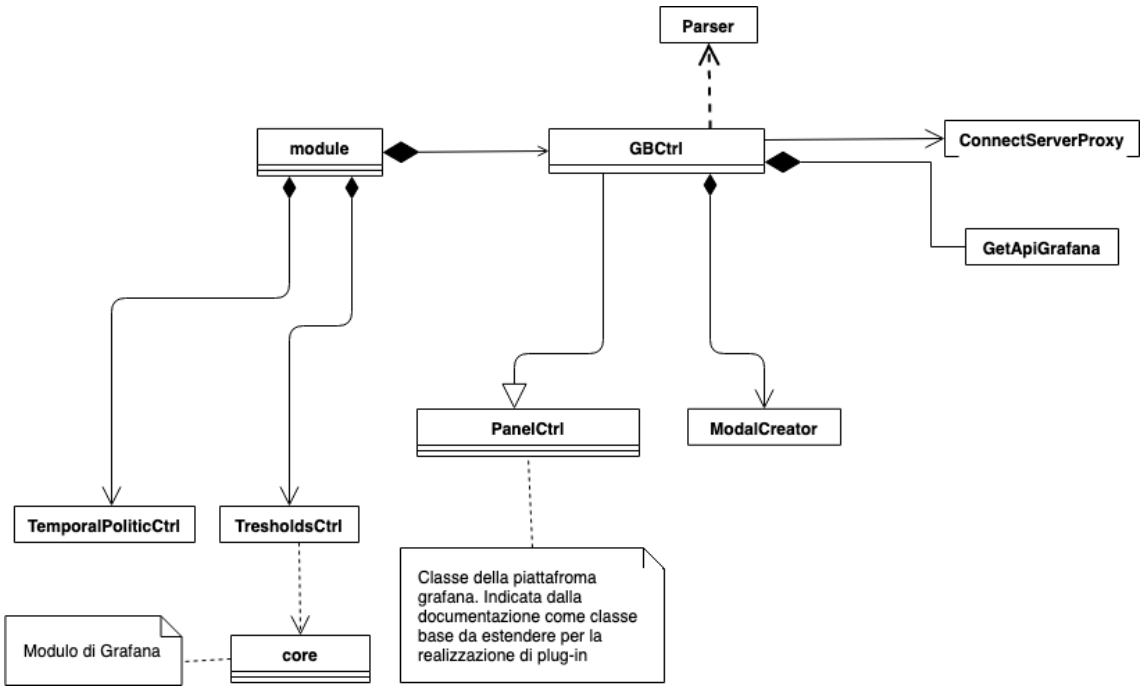

Figura 3: Diagramma delle classi del Pannello

- **GBCtrl**: è il modulo principale del plug-in, considerato come *"main"* dall'ecosistema grafana, viene utilizzato per inizializzare l'intero panello;
- **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*;
- **ConnectServer**: è il modulo che si occupa di inoltrare le richieste al Server;

- **ModalCreator**: è il modulo che si occupa di visualizzare le finestre che permettono l'interazione con l'utente;
- **ModalCreator**: è il modulo che si occupa della creazione e visualizzazione dinamica di finestre, le quali 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.

5.3.2 Descrizione delle Classi e dei Metodi del Pannello

5.3.2.1 GBCtrl

- **resetData()** : metodo usato per riportare lo stato del pannello a prima che venga caricata una rete. Viene usato quando si deve caricare una nuova rete per eliminare le precedenti impostazioni;
- **resetTresholds()** : metodo usato per preparare il pannello a salvare le soglie di una nuova rete. Elimina le soglie precedenti e imposta il dizionario coi nodi della nuova rete;
- **checkIfAtLeastOneTresholdDefined()** : controlla se l'utente ha definito almeno una soglia, viene invocato al momento dell'inizio del monitoraggio;
- **deleteLinkToFlush(node)** : cancella il collegamento di un nodo ad un flusso dati e ne cancella le soglie. Viene richiamato quando l'utente clicca sul pulsante "Scollega Nodo";
- **freeAllFlushes()** : elimina tutti i collegamenti ai flussi e riaggiunge i flussi a quelli disponibili da scegliere. Viene invocato dal metodo *resetData()*;
- **getDatabases()** : metodo invocato al momento della creazione del pannello per ottenere da *Grafana* i database disponibili;
- **getConnectionToDb()** : metodo che crea l'oggetto per ottenere i dati sui flussi dal database. Viene invocato al momento del collegamento al database;
- **checkIfConnectableToDB()** : metodo che verifica se ci si possa connettere al database selezionato, verificando: che la rete non sia sotto monitoraggio, sia stato selezionato un database e che il database selezionato sia di tipo influxdb;
- **getFlushes()** : metodo usato per ottenere i flussi dal database selezionato. Viene invocato quando ci si collega al database o si carica una rete salvata nel Server;

- **connectToDB()** : metodo che effettua il collegamento al database. Viene invocato quando l'utente preme il pulsante "Conferma" per selezionare il database;
- **showTresholdModal(node)** : metodo invocato quando l'utente preme sul nome di un nodo. Fa apparire la modal per impostare le soglie;
- **selectTemporalPolicy()** : metodo invocato quando l'utente preme sul pulsante "Imposta" che fa apparire la modal per impostare la politica temporale;
- **visualizeMonitoring()** : metodo invocato quando l'utente preme il pulsante "Visualizza Monitoraggi Attivi", che fa cambiare la schermata per visualizzare i monitoraggi;
- **visualizeSettings()** : metodo invocato quando l'utente preme il pulsante "Visualizza Impostazioni", per tornare alla schermata delle impostazioni della rete;
- **splitMonitoringNetworks()** : metodo invocato quando vengono richieste le reti al Server. Si preoccupa di capire quali sono sotto monitoraggio e di suddividerle correttamente;
- **requestNetworks()** : metodo invocato ogni volta che viene salvata una rete sul Server e quando ci si collega ad esso. Richiede la lista delle reti al Server;
- **tryConnectServer()** : metodo per provare a vedere se il Server è online su porta e ip specificati. Viene invocato quando l'utente preme sul tasto "Connetti" da Server Settings;
- **checkIfNetworkIsDeletable(net)** : metodo che viene invocato quando l'utente preme sul tasto "Elimina" per eliminare una rete. Richiede al Server che venga eliminata la rete selezionata;
- **calculateSeconds(temp)** : metodo per calcolare i secondi totali della politica temporale. Viene usato dal metodo changeNetworkToVisualizeMonitoring() per impostare la politica di refresh delle probabilità;
- **deleteProbRefresh()** : metodo usato per eliminare l'intervallo di refresh delle probabilità. Viene invocato quando l'utente ritorna alla schermata delle impostazioni delle reti;

- **updateProbs()** : metodo invocato per aggiornare il disegno delle probabilità della rete. Viene invocato ad intervalli regolari stabiliti dalla politica temporale della rete;
- **changeNetworkToVisualizeMonitoring()** : metodo invocato quando l'utente seleziona una nuova rete di cui visualizzare le probabilità. Imposta il timer di refresh delle probabilità e aggiorna il disegno della rete;
- **buildDataToSend()** : metodo invocato quando bisogna salvare una rete sul Server. Costruisce il pacchetto contenente tutti i dati necessari da salvare;
- **loadNetworkToServer(net)** : metodo usato per salvare una rete sul Server;
- **saveActualChanges()** : metodo invocato quando viene caricata una rete. Verifica se era già presente una rete e se essa abbia bisogno di essere salvata sul Server. In caso positivo la salva prima di caricare quella nuova;
- **requestNetworkToServer(net)** : metodo invocato quando l'utente preme il pulsante "Apri" per caricare una rete salvata nel Server. Si occupa di caricare nel pannello le informazioni della rete e di collegarsi al database corretto;
- **loadNetworkFromSaved(net)** : metodo usato quando l'utente carica una rete da quelle salvate. Si occupa di assegnare le variabili;
- **loadNetwork(data)** : metodo invocato quando l'utente clicca sul pulsante per confermare la rete da caricare dalla memoria locale. Si preoccupa verificare che la rete sia corretta e di inizializzarla;
- **panelPath()** : metodo per ottenere il percorso del pannello;
- **checkIfCanStartComputation()** : metodo che verifica se l'utente ha correttamente impostato i parametri per far partire la computazione;
- **startComputation()** : metodo invocato quando l'utente preme sul pulsante "Avvia Monitoraggio". Esso fa partire la computazione e salva la rete nel Server;
- **closeComputation()** : metodo invocato quando l'utente preme sul pulsante "Interrompi Monitoraggio" per interrompere il monitoraggio.

5.3.2.2 TemporalPolicyCtrl

- **checkCorrectData()** : metodo che verifica la correttezza dei dati inseriti per la politica temporale;
- **setConfirmationToTrue()** : metodo invocato quando l'utente preme sul tasto "Conferma" dalla modal della politica temporale. Verifica che sia stata impostata correttamente e lo notifica al pannello, facendo anche sparire la modal;
- **setConfirmationToFalse()** : metodo invocato quando l'utente modifica la politica temporale per notificare al pannello che al momento non è valida.

5.3.2.3 TresholdsCtrl

- **checkIfTherIsAtLeastOneTreshold(node)** : verifica che l'utente abbia impostato almeno una soglia. Viene invocato dal metodo *confirmTresholdsChanges(node)* per verificare che l'utente abbia definito almeno una soglia;
- **checkNotRepeatedTresholds(node)** : verifica che l'utente non abbia definito soglie ripetute. Viene invocato dal metodo *confirmTresholdsChanges(node)*;
- **splitForSign(node)** : divide le soglie per segno. Viene invocato per verificare che non ci siano conflitti tra le soglie;
- **setError(value1, value2, sign1, sign2)** : metodo invocato da tutti gli altri metodo qual ora vi sia un errore. fa apparire una modal con l'errore;
- **checkConflictMin(min, maj, maje)** : verifica che non ci siano conflitti tra le soglie "<" con le soglie ">" e ">=". Viene in vocato dal metodo*checkConflicts(data)*;
- **checkConflictMine(mine, maj, maje)** : verifica che non ci siano conflitti tra le soglie "<=" con le soglie ">" e ">=". Viene in vocato dal metodo*checkConflicts(data)*;
- **checkConflictSameSign(arr1, arr2, sign1, sign2)** : verifica che non ci siano conflitti tra le soglie con lo stesso segno. Viene in vocato dal metodo*checkConflicts(data)*;
- **checkConflicts(data)** : si occupa di invocare tutti i metodi per verificare la presenza di conflitti tra le soglie. Viene invocato dal metodo *confirmTresholdsChanges(node)*;

- **associate(node)** : crea l'associazione tra il nodo e il flusso attualmente selezionato. Viene invocato dal metodo *confirmTresholdsChanges(node)*.
- **confirmTresholdsChanges(node)** : metodo invocato quando l'utente preme sul tasto "Conferma" dalla modal della politica temporale. Invoca tutti i metodi per verificare la correttezza delle soglie e crea le associazioni qual ora necessario;
- **addTreshold(node, state)** : metodo invocato quando l'utente preme un tasto per aggiungere una soglia al nodo. Aggiunge la soglia relativa allo stato a cui si riferisce;
- **deleteTreshold(node, name)** : metodo invocato quando l'utente preme su un tasto "Remove" dalla modal delle soglie. Rimuove la soglia corrispondente;
- **setNotLinked(node)** : rimuove il collegamento tra il nodo e il flusso. Viene invocato quando l'utente esegue una qualsiasi azione diversa dal confermare le soglie dalla modal delle soglie;
- **setLinked(node)** : imposta il nodo come collegato. Viene invocato dal metodo *confirmTresholdsChanges(node)*;

5.3.2.4 Parser

- **validateNet()** : invoca tutti i metodi per verificare la correttezza della rete;
- **checkMinimumFields()** : verifica che il file *.JSON* abbia il numero corretto di campi, i quali devono avere anche il nome corretto;
- **checkNamedNodes(data, field)** : verifica che il campo data abbia il corretto numero di linee e che esse abbiano il nome giusto;
- **checkDuplicates(values)** : verifica che nell'array values non ci siano valori duplicati. Viene invocato dal metodo *checkParents()* per verificare che non ci siano padri ripetuti;
- **checkStates()** : verifica che gli stati dei nodi siano correttamente definiti;
- **checkParents()** : verifica che i padri dei nodi siano correttamente definiti;
- **checkProbabilities()** : verifica che le probabilità dei nodi siano correttamente definiti;

- **countNumberOfValue(node)** : metodo usato da *checkProbabilities()* per verificare che ogni nodo abbia il numero corretto di probabilità. Calcola quante probabilità il nodo deve avere sulla base degli stati e dei padri.

5.3.2.5 ConnectServer

- **alive()** : verifica che il Server sia online su porta e ip specificati. Viene invocato quando dal metodo *tryConnectServer()* di *GBCtrl*;
- **networks()** : richiede al Server le reti salvate. Viene invocato da *GBCtrl* in molteplici punti;
- **uploadnetwork(net)** : effettua la richiesta al Server per salvare la rete net passata come parametro. Viene invocato da *GBCtrl*;
- **deletenetwork(net)** : effettua la richiesta al Server per eliminare la rete net passata come parametro. Viene invocato da *GBCtrl* nel metodo *requestNetworkDelete(net)*; *getnetworkprob(net)* : effettua la richiesta al Server per ottenere il graph della libreria *JSBayes* relativo alla rete net. Viene invocato dal metodo *updateProbs()* di *GBCtrl* per rinnovare le probabilità della rete sotto monitoraggio;
- **getnetwork(net)** : effettua la richiesta al Server per ottenere i dati della rete di nome specificato nel parametro net. Viene invocato in vari punti da *GBCtrl*.

5.3.2.6 ModalCreator

- **checkMonitoring(message)** : verifica se la rete attuale è sotto monitoraggio e in caso positivo fa apparire il messaggio di errore. Viene invocato da *showTresholdModal(node) e selectTemporalPolicy()*;
- **checkDB(message)** : verifica che il pannello sia collegato ad un database e in caso negativo fa apparire un messaggio di errore. Viene invocato da *showTresholdModal(node) e selectTemporalPolicy()*;
- **showMessageModal(message, title)** : fa apparire una modal con il messaggio di errore message e titolo title. Viene invocato dagli altri metodi;
- **showTresholdModal(node)** : metodo invocato da *GBCtrl* per far apparire la modal per definire le soglie del nodo node;
- **selectTemporalPolicy()** : metodo invocato dal metodo *selectTemporalPolicy()* di *GBCtrl* per far apparire la modal per la definizione della politica temporale.

5.3.2.7 GetApiGrafana

- **queryAPI()** : metodo invocato dal metodo *getData()* per ottenere i database disponibili da *Grafana*;
- **getData()** : metodo invocato dal metodo *getDatabases()* di *GBCtrl* per ottenere la lista dei databse ben strutturata;
- **getDatasources()** : ritorna i flussi disponibili dal database selezionato ben strutturati. Viene invocato dal metodo *getFlushes()* di *GBCtrl*;
- **getTables()** : richiede a *Grafana* le tabelle del database selezionato. Viene invocato dal metodo *getDatasources()*;
- **getFields(tables)** : richiede a *Grafana* i campi delle tabelle specificate nel parametro tables. Viene invocato dal metodo *getDatasources()*;
- **divideFields(elements)** : metodo invocato da *getDatasources()* per formattare correttamente i dati sui campi delle tabelle ottenuti da *Grafana*;
- **initialize()** : metodo invocato da *getDatasources()* per interpretare correttamente il link del database.

5.4 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.

5.4.1 UML

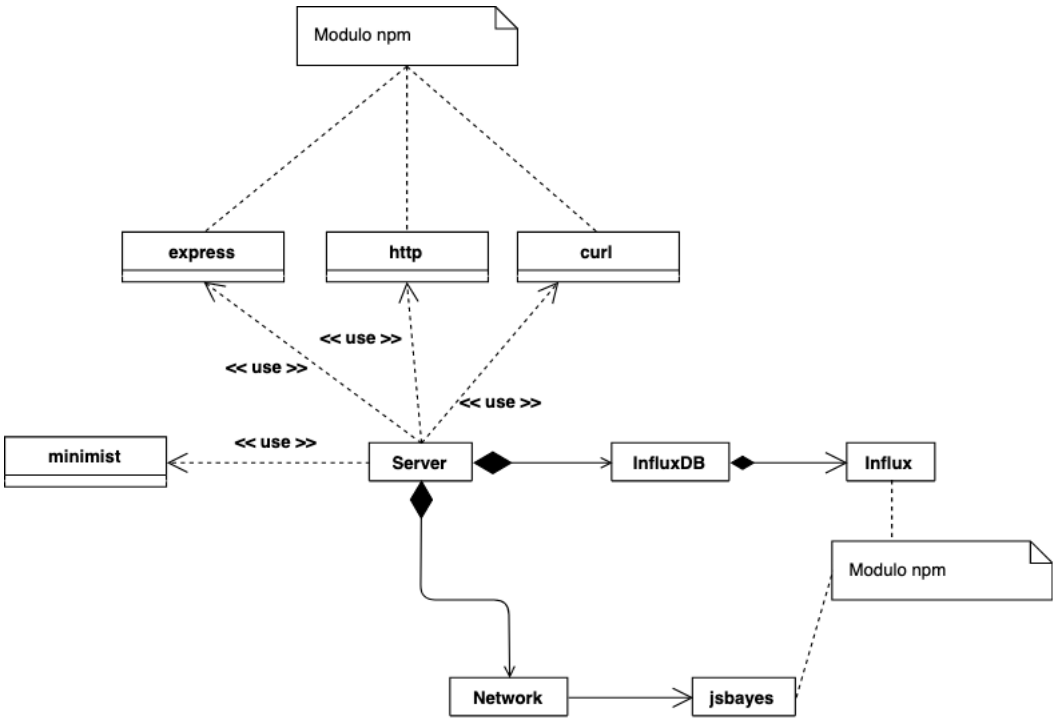

Figura 4: Architettura dell'applicativo

- **Server**: è il modulo principale che si occupa della comunicazione dei vari moduli che compongono il server, e della gestione delle richieste provenienti dai vari panelli;
- **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;

5.4.2 Descrizione delle Classi e dei Metodi del Server

5.4.2.1 Server

- **constructor()**: costruttore di classe. Inizializza l'istanza di express$_G$, l'array di connessioni, reti e di pool, inoltre esegue una verifica sintattica sui parametri di configurazioni passati tramite *conf.json*;
- **configExpressApp()**: metodo utilizzato per l'inizializzazione delle varie route$_G$ messe a disposizione dal server;
- **parserNetworkNameURL(name)**: metodo utilizzato per parsare il nome di una rete proveniente da indirizzo URL. Ritorna false in caso di errore o stringe non valide, altrimenti ritorna il nome parsato;
- **saveNetworkToFile(data)**: metodo utilizzato per il salvataggio su filesystem delle reti bayesiane caricate dall'utente. Riceve come input un json rappresentate la rete bayesiana. Ritorna un booleano a true se il salvataggio è stato eseguito o false altrimenti. Lancia eccezioni in caso di errori di scrittura su filesystem;
- **startServer()**: metodo utilizzato per l'effettivo avvio del server. Mette in ascolto il server sulla porta configurata a costruttore ed inizializza le reti salvate su filesystem;
- **shutDown()**: metodo utilizzato per la chiusura del server;
- **initSavedNetworks()**: metodo utilizzato per la lettura da file system e il successivo caricamento in memoria delle reti bayesiane salvate in filesystem. Per ogni rete letta, viene chiamato il metodo *initBayesianNetwork(net)*;
- **initDatabaseConnection(connection)**: metodo utilizzato per l'inizializzazione di una connessione al database. Riceve come parametro un json di definizione della connessione da creare. Può lanciare un'eccezione in caso di fallimento;
- **initBayesianNetwork(net)**: metodo utilizzato per aggiungere un'istanza di **Network** all'array di reti monitorate. Riceve in input la definizione della rete in json, e ne crea l'oggetto che la rappresenta; Può lanciare eccezioni. Ritorna true se la creazione è andata a buon fine;
- **countNetworks()**: ritorna in numero di reti nell'array di reti monitorate;

- **observeNetworks(net, data)**: metodo utilizzato per risolvere le dependency richiesta dall'oggetto *Network* per eseguire il calcolo delle probabilità. Riceve in input la rete da osservare e i dati richiesti da quest'ultima. Successivamente esegue una chiamata al database chiedendo i dati necessari al calcolo per poi eseguire i calcoli con quest'ultimi chiamando il metodo apposito. Inoltre il metodo è incaricato della gestione delle soglie critiche modificando qualora fosse necessario la politica temporale di ricalcolo. Fine richiama il metodo *writeMesure(net)*;

- **configParser()**: metodo utilizzato per il controllo della sintassi nel file di configurazione. Ritorna true se la sintassi è corretta, false altrimenti;

- **checkParam(argv)**: metodo utilizzato per il controllo dei parametri passati da terminale all'avvio del server. Ritorna true se i parametri sono validi, false altrimenti;

- **getNetworks()**: metodo utilizzato per la creazione di un'array contenente il nome e il campo *monitoring* di tutte le reti salvate in filesystem. Ritorna un'array di tuple stringa, boolean;

- **getMilliseconds(temporal)**: metodo utilizzato per la conversione della politica temporale in ore, minuti, secondi in millisecondi. Riceve in input un json con ore, minuti e secondi definiti come interi, e ritorna un intero, il quale rappresenta la conversione in millisecondi;

- **addToPool(net)**: metodo utilizzato per aggiungere al monitoraggio una rete bayesiana già caricata in memoria. Riceve come parametro il nome della rete, con quest'ultimo viene verificato se è gia presente nel all'array di pool. Se non è presente viene aggiunto al pool seguendo le politiche temporali riportate dall'utente. Ritorna true nel caso in cui la rete sia stata aggiunta con successo, false altrimenti;

- **deleteFromPool(net)**: metodo utilizzato per l'eliminazione di una rete in monitoraggio. Riceve come parametro il nome della rete. Se la rete è in monitoraggio viene fermata ed eliminata dal pool. Ritorna true se la rete è stata eliminata dal pool di monitoraggio, false altrimenti;

- **writeMesure(net)**: metodo utilizzato per la scrittura delle probabilità calcolate dalla varie reti in monitoraggio sui rispetti database di salvataggio. Riceve

in input il nome di una rete, dopodiché preleva le probabilità calcolate dall'oggetto rappresentate la rete e le scrive sul database. Ritorna true se i dati sono stati scritti con successo, false altrimenti;

5.4.2.2 Network

- **constructor(net)**: costruttore di classe a un parametro. Riceve la definizione di una rete bayesiana in formato JSON e ne costruisce l'oggetto rappresentate;
- **observe(node, state)**: metodo utilizzato da wrapper_G per l'utilizzo del metodo *observe* della classe *jsbayes*. Riceve in input due parametri: net di tipo stringa e state di tipo stringa;
- **sample(n)**: wrapper del metodo *sample(n)*, della classe **jsbayes**. Esegue il calcolo delle probabilità dei vari nodi con il metodo $\textit{Monte Carlo}_\text{G}$. Riceve in input un numero intero il quale indica il numero di iterazioni da eseguire;
- **unobserveAll()**: metodo utilizzato per togliere dall'osservazione i vari nodi precedentemente collegati. Per ogni nodo viene richiamato il metodo *unobserve(node)* della classe jsbayes;
- **orderTresholds()**: metodo utilizzato per il riordinamento delle soglie, in ordine crescente e decrescente, definite nella rete;
- **observeData(dati)**: metodo utilizzato per l'osservazione dei dati. Riceve in input un'array associativo dei dati necessari da monitorare tramite i quali verranno effettuate, a seconda delle soglie, i confronti tra dati - soglie, per poi successivamente fare scattare la probabilità con il metodo *observe*. In fine una volta eseguiti tutti i confronti verrà richiamato il metodo *sample* per calcolare le probabilità. Ritorna true se una soglia critica è stata superata, false altrimenti;
- **getProbabilities()**: metodo utilizzato per estrarre le probabilità di ogni stato, per ogni singolo nodo. Ritorna un'array associativo con il nome del nodo e un array contenete le probabilità. Quest'ultimo a sua volta è un'array associativo formato da tuple: nome stato e probabilità calcolata;

5.4.2.3 InfluxDB

- **constructor(host, port, database, dbwrite, user, pwd)**: costruttore di classe. Riceve in input host e porta sulla quale il database è in ascolto, il

nome del database, il database sul quale salvare i dati, username e password. Il metodo costruisce un oggetto *Influx* il quale rappresenta la connessione al database. Avviene un controllo dei dati passati input con opportuno lancio di eccezioni nel caso in cui alcuni dei dati non fossero conformi alla sintassi o errati;

- **getDatasources()**: metodo utilizzato per estrapolare le tabelle disponibili per la lettura. Ritorna una $Promise_G$ da risolvere successivamente. Può lanciare un'eccezione nel caso in cui la $query_G$ al database fallisca o sia interrotta;
- **queryDB(query)**: wrapper del metodo *query* della classe *InfluxDB*, il quale viene utilizzato per l'interrogazione del database tramite query;
- **getDatasourcesFields(source)**: metodo utilizzato per estrarre le colonne di una determinata tabella. Riceve in input una source da ispezionare. Ritorna una *Promise* da risolvere successivamente;
- **getLastValue(source, field)**: metodo utilizzato per leggere l'ultimo dato in serie temporale dal database. Riceve in input la source dalla quale estrarre il field interessato. Ritorna una *Promise* da risolvere successivamente. Può lanciare un'eccezione nel caso in cui l'interrogazione del database fallisca;
- **getListData(fields)**: metodo utilizzato per generare un'array di ultimi dati, prelevati in serie temporali dal database. Riceve in input un'array di json, nel quale per ognuno di essi viene definito un campo table ed un campo flush. Per ogni valore da prelevare viene richiamato il metodo *getLastValue* per ricevere l'ultimo dato in ordine temporale. Ritorna un'array di tuple: nome campo e valore campo;
- **writeOnDB(net, probs)**: metodo utilizzato per la scrittura delle probabilità di una rete sul database di scrittura definito in fase di costruzione. Riceve in input il nome della rete e le sue probabilità. Per ogni nodo della rete viene inserito un campo contenente il nome del nodo e le varie probabilità calcolate;
- **checkStatus(timeout)**: metodo utilizzato per verificare la connessione al database. Riceve in input un timeout utilizzato per effettuare un $ping_G$ al database ogni *timeout* secondi. Ritorna un *Promise* da risolvere successivamente la quale contiene i dati di risposta dal database;

6 Test

6.1 Scopo del Capitolo

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

6.2 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.

6.3 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 §6.2 il quale provvederà a generare i report necessari.

7 Estensione delle Funzionalità

Questa sezione ha l'obbiettivo di fornire tutte le indicazioni necessarie all'evoluzione del prodotto. Essa si suddivide a sua volta in due sezioni principali: *Backend* e *Pannello*

7.1 Backend

7.1.1 Aggiunta di una route nel server

Per aggiungere una route al server bisogna dichiararla nel metodo `configExpressApp()`. A seconda che la chiamata sia una *POST* o una *GET*, esistono due diverse sintassi per scrivere la route.

- *GET*: le chiamate get trasmettono dati attraverso l'url e possono ritornare dati in return in formato html o come semplice text.

```
1 this.app.get('/route', (req, res) => {
2  // do something
3  res.send(response);
4 });
```

  Una chiamata get parametrizzata attraverso url viene definita usando un placeholder con la sintassi `:placeholder` nella definizione della route. Quest'ultima è accessibile al metodo tramite: `req.params.param`

```
1 this.app.get('/route/:param', (req, res) => {
2  // do something
3  console.log(req.params.param);
4  res.send(response);
5 });
```

- *POST*: le chiamate post al server vengono utilizzate per l'invio di dati da elaborare al server. Esse come le chiamate get hanno bisogno di un url che le identifica. Possono inoltre ritornare un json di risposta o un semplice text.

```
1 this.app.post('/route', (req, res) => {
2  // do something
3  res.send(response);
4 });
```

  Inoltre è possibile definire delle route in post parametrizzate con la stessa sintassi definita per le chiamate get.

```
1 this.app.post('/route/:param', (req, res) => {
2  // do something
3  console.log(req.params.param);
4  res.send(response);
5 });
```

7.1.2 Aggiunta database

Per modellare l'utilizzo di un database diverso da quello inizialmente fornito dal prodotto, bisognerà creare una nuova classe che metta a disposizione i seguenti metodi:

- `queryDB(query)`: preleva i dati dal database. In input riceva la query da eseguire sul database; Ritorna una *Promise* che verrà risolta in seguito;
- `getLastValue(source, field)`: preleva l'ultimo dato dalla sorgente desiderata in ordine temporale dal database. In input riceve la *source* da cui prelevare il *field* interessato. Ritorna una *Promise* che verrà risolta in seguito;
- `getListData(field)`: riceve in input un'array associativo `["source" => "field"]` da prelevare. Ritorna un'array di tutti i field richiesti;
- `writeOnDB(net, probs)`: scrive nel database una entry con il nome della rete e le varie probabilità calcolate passate come parametri al metodo.

Infine bisogna rendere la classe esportabile definendo l'esportazione a fine definizione di quest'ultima come segue:

```
module.exports = nomeClasse;
```

Definendo i seguenti metodi si fornisce la possibilità di prelevare dati gestiti in serie temporali, i quali verranno forniti come dipendenze alla rete bayesiana per il calcolo delle probabilità.

7.1.3 Estensione funzionalità Network

La gestione delle reti bayesiane viene demandata alla classe `Network`, con il supporto dalla libreria *jsbayes*. L'estensibilità della classe é rimandata in gran parte alla libreria utilizzata, cosa per cui ogni possibile estensione o modifica risulta difficile. Ogni altra estensione o modifica della classe riguarderebbe l'utilizzo di un'ulteriore libreria per la gestione delle reti bayesiane e il calcolo delle probabilità, la quale porterebbe alla riscrittura totale della classe.

7.2 Pannello

7.2.1 Modifica conseguente ad estensioni lato Server

La parte del pannello deve essere modificata in conseguenza ad un'eventuale modifica del Server, per adeguarsi ai relativi cambiamenti.

Aggiunta nuovi Database:
Nel momento in cui si desideri aggiungere il supporto a nuovi database (sezione §7.1.2), bisogna modificare :

- Il metodo *connectToDB()*, in modo che distingua tra database di tipo *inluxdb* e nuovi;
- Modificare il metodo *checkIfConnectableToDB()* in modo che gestisca i nuovi database;
- Aggiungere metodi alla classe *GetApiGrafana*, in modo che riesca a gestire i flussi derivanti da tali database.

Aggiunta route nel server:
Qualora nel Server vengano aggiunte nuove route (Sezione §7.1.1), bisogna aggiungere nuovi metodi alla classe *ConnectServer*, in modo che gestiscano le nuove chiamate.

7.2.2 Estensione funzionalità indipendenti dal Server

Visualizzazione dei dati di monitoraggio
Una sezione del pannello estendibile indipendentemente dal Server può essere la visualizzazione delle probabilità. Per far ciò bisogna aggiungere un metodo alla classe *GBCtrl* ed aggiungere una parte alla sezione di visualizzazione delle reti, nella quale l'utente possa scegliere il metodo di visualizzazione.

Modifica della struttura delle Reti Bayesiane accettate
Per modificare la struttura della rete bayesiana accettata, bisogna modificare la classe Parser, aggiungendo nuovi metodi che facciano i controlli desiderati ed estendere il metodo *validateNet()* in modo che richiami anche tali metodi.

A Licenza

B Glossario

AngularJS
AngularJS è un framework per applicazioni web, sviluppato con lo scopo di affrontare le difficoltà incontrate con lo sviluppo di applicazioni su singola pagina di tipo client side.

API
Application Programming Interface (API) si indica un insieme di procedure atte all'espletamento di un dato compito.

CSS
Acronimo di Cascading Style Sheets (fogli di stile a cascata), è un linguaggio usato per definire la formattazione di documenti HTML, XHTML e XML ad esempio i siti web e relative pagine web. Le regole per comporre il CSS sono contenute in un insieme di direttive emanate a partire dal 1996 dal W3C.

DevOps
Acronimo di Development e Operations, DevOps è un approccio allo sviluppo e all'implementazione di applicazioni in azienda, che enfatizza la collaborazione tra il team di sviluppo vero e proprio e quello delle operations, ossia che gestirà le applicazioni dopo il loro rilascio.

EcmaScript 6
Sesta edizione di ECMAScript definita nel 2015, implementa significativi cambiamenti sintattici per scrivere applicazioni più complesse. Tuttavia il supporto per ES6, da parte dei browser, è ancora incompleto.

Express
Framework per applicazioni web NodeJS flessibile e leggero che fornisce una serie di funzioni avanzate per le applicazioni web e per dispositivi mobili.

Grafana
Piattaforma open source che permette il monitoraggio e l'analisi di dati che vengono visualizzati in dashboard operative.

HTML
Acronimo di HyperText Markup Language, è un linguaggio di markup nato per la

formattazione e impaginazione di documenti ipertestuali disponibili nel web

Jest
È un framework di testing di JavaScript. È compatibile con Babel, TypeScript, Node, React, Angular, Vue. Offre la possibilità di eseguire test in parallelo in maniera affidabile, ed eseguire il code coverage di interi progetti.

Liveliness
Dall'Inglese: "vivacità". Tutta via si vuole intendere lo stato di vita di un sistema.

Metodo Monte Carlo
Il metodo Monte Carlo è un'ampia classe di metodi ed algoritmi computazionali basati sul campionamento casuale per ottenere risultati numerici. Può essere utile per superare i problemi computazionali legati ai test esatti (ad esempio i metodi basati sul calcolo combinatorio, che per grandi campioni generano un numero di permutazioni eccessivo).
L'algoritmo Monte Carlo è dunque un metodo numerico che viene utilizzato per trovare le soluzioni di problemi matematici che possono avere molte variabili e che non possono essere risolti facilmente.

MVC
È un pattern architetturale molto diffuso nella programmazione orientata agli oggetti in grado di separare la logica di presentazione dalla logica di business. È composto da tre componenti: il "model" che fornisce i metodi per accedere ai dati all'applicazione, la "View" visualizza i dati contenuti nel model e si occupa dell'iterazione con gli utenti, e infine il "Controller" che riceve i comandi dall'utente attraverso la view e va a modificare lo stato degli altri due componenti (model e view).

NodeJS
Piattaforma open source per scrivere applicazione in JavaScript Server-side.

Open source
Termine utilizzato per indicare programmi software non protetti da copyright. Essenziale per favorire il libero studio e permettere ai programmatori indipendenti di apportarvi modifiche.

Plug-in

Componente aggiuntivo che interagisce con un altro programma per ampliarne le funzioni.

Promise Rappresenta un proxy per un valore non necessariamente noto quando la promise è stata creata. Consente di associare degli handlers con il successo o il fallimento di un'azione asincrona (e il "valore" in caso di successo). Questo in pratica consente di utilizzare dei metodi asincroni di fatto come se fossero sincroni.

REST
Representational State Transfer (REST) è uno stile architetturale (di architettura software) per i sistemi distribuiti e rappresenta un sistema di trasmissione di dati su HTTP senza ulteriori livelli.

Rete bayesiana
Rappresentazione grafica delle relazioni di dipendenza tra le variabili di un sistema. In statistica la rete bayesiana è utilizzata per individuare più agevolmente le relazioni di dipendenza assoluta e condizionale tra le variabili, al fine di ridurre il numero delle combinazioni delle variabili da analizzare.

Route
Per route si intende determinare come un'applicazione risponde a una richiesta client a un endpoint particolare, il quale è un URI (o percorso) e un metodo di richiesta HTTP specifico (GET, POST, e cosi via).

wrapper
Metodo che avvolge un altro metodo per adattarlo alle esigenze del di una determinata classe.