

# ProApes

*proapes11@gmail.com*

## Manuale Sviluppatore

<b>Versione</b>	1.0.0-1.10
<b>Data approvazione</b>	2020-05-15
<b>Responsabile</b>	Federico Carboni
<b>Redattori</b>	Federico Carboni Francesco Bari Alessandro Discalzi
<b>Verificatori</b>	Valentina Signor Fiammetta Cannavò
<b>Stato</b>	Approvato
<b>Lista distribuzione</b>	<i>ProApes</i> <i>Prof. Tullio Vardanega</i> <i>Prof. Riccardo Cardin</i>
<b>Uso</b>	Esterno

## Sommario

Il presente documento ha lo scopo di presentare le tecnologie e l'architettura del sistema agli sviluppatori interessati alla estensione e al mantenimento del software

*Predire in Grafana.*

# Diario delle Modifiche

Versione	Data	Modifica	Autore	Ruolo
v1.0.0-1.10	2020-05-15	<i>Approvazione del documento per RA</i>	Federico Carboni	<i>Responsabile di Progetto</i>
v0.6.0-1.9	2020-05-12	<i>Revisione e correzioni di coerenza e coesione (Verificatore: Fiammetta Cannavò)</i>	Francesco Bari	<i>Programmatore</i>
v0.5.2-1.9	2020-05-11	<i>Aggiornate immagini §5.1 e descritto meglio §6 (Verificatore: Fiammetta Cannavò)</i>	Federico Carboni	<i>Programmatore</i>
v0.5.1-1.9	2020-05-10	<i>Aggiunte descrizioni architettura con funzionalità facoltative §5.2 (Verificatore: Valentina Signor)</i>	Federico Carboni	<i>Programmatore</i>
v0.5.0-1.9	2020-05-09	<i>Revisione e correzioni di coerenza e coesione (Verificatore: Valentina Signor)</i>	Alessandro Discalzi	<i>Programmatore</i>
v0.4.3-1.9	2020-05-08	<i>Aggiornate immagini §5.1 (Verificatore: Fiammetta Cannavò)</i>	Francesco Bari	<i>Programmatore</i>
v0.4.2-1.9	2020-05-07	<i>Aggiunte descrizioni architettura con funzionalità facoltative §5.1 (Verificatore: Fiammetta Cannavò)</i>	Alessandro Discalzi	<i>Programmatore</i>
v0.4.1-1.9	2020-05-05	<i>Aggiornate descrizioni funzionalità facoltative in §2 e §3 (Verificatore: Fiammetta Cannavò)</i>	Federico Carboni	<i>Programmatore</i>

<b>Versione</b>	<b>Data</b>	<b>Modifica</b>	<b>Autore</b>	<b>Ruolo</b>
v0.4.0-1.8	2020-05-04	<i>Revisione e correzioni di coerenza e coesione (Verificatore: Valentina Signor)</i>	Alessandro Discalzi	<i>Programmatore</i>
v0.3.3-1.8	2020-05-03	<i>Aggiornati riferimenti tecnologie §1 e spostati alcuni più generici in §A (Verificatore: Valentina Signor)</i>	Francesco Bari	<i>Programmatore</i>
v0.3.2-1.8	2020-05-02	<i>Tolti riferimenti interni §1 (Verificatore: Valentina Signor)</i>	Francesco Bari	<i>Programmatore</i>
v0.3.1-1.8	2020-05-02	<i>Aggiornamento registro delle modifiche (Verificatore: Fiammetta Cannavò)</i>	Federico Carboni	<i>Programmatore</i>
v0.3.0-1.5	2020-04-13	<i>Approvazione del documento per RQ</i>	Giacomo Piran	<i>Responsabile di Progetto</i>
v0.2.0-1.5	2020-04-11	<i>Revisione e correzioni di coerenza e coesione (Verificatore: Valentina Signor)</i>	Federico Carboni	<i>Programmatore</i>
v0.1.3-0.4	2020-04-09	<i>Stesura sezione §6 e appendice §A (Verificatore: Fiammetta Cannavò)</i>	Alessandro Discalzi	<i>Programmatore</i>
v0.1.2-0.4	2020-04-07	<i>Stesura sezioni §4 e §5 (Verificatore: Fiammetta Cannavò)</i>	Federico Carboni	<i>Programmatore</i>
v0.1.1-0.4	2020-04-04	<i>Stesura §3 (Verificatore: Fiammetta Cannavò)</i>	Federico Carboni	<i>Programmatore</i>
v0.1.0-0.3	2020-04-01	<i>Revisione e correzioni di coerenza e coesione (Verificatore: Valentina Signor)</i>	Alessandro Discalzi	<i>Programmatore</i>

Versione	Data	Modifica	Autore	Ruolo
v0.0.3-0.3	2020-03-28	<i>Stesura §2 (Verificatore: Fiammetta Cannavò)</i>	Francesco Bari	<i>Programmatore</i>
v0.0.2-0.2	2020-03-25	<i>Stesura §1 (Verificatore: Valentina Signor)</i>	Federico Carboni	<i>Programmatore</i>
v0.0.1-0.2	2020-03-23	<i>Creazione documento LATEX</i>	Giacomo Piran	<i>Responsabile di Progetto</i>

# Indice

<b>1</b>	<b>Introduzione</b>	<b>7</b>
1.1	Scopo del documento . . . . .	7
1.2	Scopo del prodotto . . . . .	7
1.3	Prerequisiti per la comprensione . . . . .	7
1.4	Glossario . . . . .	7
1.5	Note sulle versioni . . . . .	7
1.6	Riferimenti . . . . .	8
1.6.1	Normativi . . . . .	8
1.6.2	Informativi . . . . .	8
1.6.3	Legali . . . . .	9
<b>2</b>	<b>Tecnologie e librerie coinvolte</b>	<b>10</b>
2.1	Tecnologie . . . . .	10
2.1.1	<i>Grafana</i> . . . . .	10
2.1.2	<i>InfluxDB</i> . . . . .	10
2.1.3	<i>Telegraf</i> . . . . .	10
2.1.4	<i>React</i> . . . . .	10
2.1.5	<i>TypeScript</i> . . . . .	11
2.1.6	<i>JSON</i> . . . . .	11
2.1.7	<i>CSV</i> . . . . .	11
2.1.8	<i>HTML5</i> . . . . .	11
2.1.9	<i>CSS3</i> . . . . .	11
2.1.10	<i>NodeJS</i> . . . . .	11
2.1.11	<i>Yarn</i> . . . . .	12
2.1.12	<i>SonarCloud</i> . . . . .	12
2.2	Librerie di terze parti . . . . .	12
2.2.1	<i>MobX</i> . . . . .	12
2.2.2	<i>ESLint</i> . . . . .	12
2.2.3	<i>Jest</i> . . . . .	12
2.2.4	<i>Regression-js</i> . . . . .	13
2.2.5	<i>Svmjs</i> . . . . .	13
<b>3</b>	<b>Setup</b>	<b>14</b>
3.1	Requisiti minimi di sistema . . . . .	14
3.1.1	Prerequisiti . . . . .	14
3.1.2	Requisiti Hardware . . . . .	14
3.1.3	Browser . . . . .	14
3.2	Installazione . . . . .	14
3.2.1	Installazione Training Module . . . . .	14
3.2.2	Installazione Prediction Module . . . . .	15
3.3	Ambiente di lavoro . . . . .	16
3.3.1	<i>Grafana</i> . . . . .	16
3.3.2	<i>InfluxDB</i> . . . . .	16
3.3.3	<i>NodeJS</i> . . . . .	16
3.3.4	<i>React</i> . . . . .	16
3.3.5	Librerie necessarie . . . . .	16
<b>4</b>	<b>Test</b>	<b>17</b>
4.1	<i>Jest</i> . . . . .	17

4.2	<i>ESLint</i>	17
4.3	<i>SonarCloud</i>	17
4.4	<i>GitHub Actions</i>	17
<b>5</b>	<b>Architettura del prodotto</b>	<b>18</b>
5.1	Architettura Training Module	18
5.1.1	Pattern architettonico: MVVM	18
5.1.2	<i>MobX</i>	19
5.1.3	Architettura di dettaglio: Strategy	20
5.1.4	<code>train()</code>	21
5.1.5	<code>loadData()</code>	22
5.1.6	Diagramma delle classi	23
5.2	Architettura Prediction Module	24
5.2.1	Pattern architettonico: MVVM	24
5.2.2	Architettura di dettaglio: Strategy	25
5.2.3	Architettura di dettaglio: Iterator	27
5.2.4	<code>updatePrediction()</code>	28
5.2.5	Diagramma delle classi	29
<b>6</b>	<b>Estendere <i>Predire in Grafana</i></b>	<b>30</b>
6.1	Training Module	30
6.2	Prediction Module	32
<b>A</b>	<b>Glossario</b>	<b>34</b>

## Elenco delle figure

1	Diagramma dei package del Training Module . . . . .	18
2	Implementazione pattern MVVM nel Training Module . . . . .	19
3	Strategy per la scelta dell'algoritmo . . . . .	20
4	Viste in base all'algoritmo scelto. . . . .	20
5	Diagramma di sequenza per la funzione <code>train()</code> . . . . .	21
6	Diagramma di sequenza per la funzione <code>loadData()</code> . . . . .	22
7	Diagramma delle classi del Training Module . . . . .	23
8	Diagramma dei package del Prediction Module . . . . .	24
9	Implementazione pattern MVVM Prediction Module . . . . .	25
10	Strategy per la ricostruzione della funzione di predizione . . . . .	25
11	Viste in base al predittore importato . . . . .	26
12	Iterator per l'accesso sequenziale alle strutture dati . . . . .	27
13	Diagramma di sequenza per la funzione <code>updatePrediction()</code> . . . . .	28
14	Diagramma delle classi del Prediction Module . . . . .	29
15	Aggiunta di una nuova Strategy <code>NewConcreteStrategy</code> . . . . .	30
16	Aggiunta di una nuova vista specializzata <code>NewView</code> . . . . .	30
17	Aggiunta di una nuova classe di Options <code>NewOptionAlgorithm</code> . . . . .	31
18	Aggiunta di una nuova classe Data <code>NewDataAlgorithm</code> . . . . .	31
19	Aggiunta di un nuovo algoritmo <code>NewConcreteStrategy</code> . . . . .	32
20	Aggiunta della configurazione di un nuovo componente <code>NewConfig</code> . . . . .	32
21	Aggiunta di una nuova classe di Options <code>NewOptionAlgorithm</code> . . . . .	33

# 1 Introduzione

## 1.1 Scopo del documento

Lo scopo del *Manuale Sviluppatore* è presentare l'architettura del prodotto *Predire in Grafana<sub>G</sub>*, l'organizzazione del codice sorgente e in particolare fornire informazioni utili al mantenimento e all'estensione del progetto. Questo documento ha inoltre il fine di illustrare le procedure di installazione e di sviluppo in locale, citare i framework e le librerie di terze parti coinvolte e di presentare la struttura del progetto a livelli progressivi di dettaglio, grazie all'utilizzo di diagrammi dei package, di classe e di sequenza.

## 1.2 Scopo del prodotto

Il capitolo C4 - *Predire in Grafana* nasce dall'esigenza, a seguito dell'applicazione di una politica di tipo DevOps<sub>G</sub> durante il ciclo di vita<sub>G</sub> del software, di effettuare un monitoraggio costante delle applicazioni e delle informazioni ivi contenute. A tal fine il gruppo *ProApes* si propone di sviluppare per l'azienda *Zucchetti S.p.A.* un plug-in<sub>G</sub> da affiancare allo strumento di monitoraggio *Grafana<sub>G</sub>* che applichi le tecniche di *SVM<sub>G</sub>* e *Regressione Lineare<sub>G</sub>* sul flusso dei dati ricevuti per allarmi o segnalazioni tra gli operatori del servizio Cloud e la linea di produzione del software.

## 1.3 Prerequisiti per la comprensione

Per poter comprendere al meglio il contenuto del documento, in particolare i diagrammi presentati nelle sezioni successive, è opportuno che il lettore abbia almeno un'infarinatura generale del linguaggio *UML2.0<sub>G</sub>*, degli algoritmi di Machine Learning *Regressione Lineare*, *Esponenziale*, *Logaritmica* e *Support Vector Machine*, della piattaforma *Grafana* e del framework *React<sub>G</sub>*.

## 1.4 Glossario

All'interno del documento sono presenti termini che possono presentare significati ambigui o incongruenti a seconda del contesto. Al fine quindi di evitare l'insorgere di incompreseioni viene fornito un glossario individuabile nell'appendice §A, posta alla fine di questo documento, contenente i suddetti termini e la loro spiegazione. Nella seguente documentazione per favorire maggiore chiarezza ed evitare inutili ridondanze tali parole vengono indicate mettendo una "G" a pedice di ogni prima occorrenza del termine che si incontri ad ogni inizio di sezione.

## 1.5 Note sulle versioni

Il gruppo *ProApes* garantisce il funzionamento del prodotto *Predire in Grafana* solo nel caso in cui vengano usate le versioni esatte delle librerie, dei framework e di *Grafana* che verranno elencate nel resto del documento. Nonostante sia probabile che le successive versioni degli strumenti utilizzati mantengano la retro-compatibilità, ciò non può essere garantito. Il gruppo non si assume quindi nessuna responsabilità nel caso in cui il prodotto risenta, del tutto o in parte, di problematiche riconducibili ad un utilizzo di software di terze parti con versioni diverse da quelle specificate.

## 1.6 Riferimenti

### 1.6.1 Normativi

- **Capitolato d'appalto C4:**  
<https://www.math.unipd.it/~tullio/IS-1/2019/Progetto/C4.pdf>

### 1.6.2 Informativi

- **Model-View Patterns - Materiale didattico del corso di Ingegneria del Software:**  
<https://www.math.unipd.it/~rcardin/sweb/2020/L02.pdf>
  - Struttura MV patterns, slide 6 - 14;
  - MVVM, slide 29 - 50.
- **SOLID Principles - Materiale didattico del corso di Ingegneria del Software:**  
<https://www.math.unipd.it/~rcardin/sweb/2020/L04.pdf>
  - Single Responsibility Principle, slide 5 - 10;
  - Open-Close Principle, slide 11 - 17;
  - Liskov Substitution Principle, slide 18 - 23;
  - Interface Segregation Principle, slide 26 - 30;
  - Dependency Inversion Principle, slide 32 - 36.
- **Diagrammi delle classi - Materiale didattico del corso di Ingegneria del Software:**  
<https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E01b.pdf>
  - Proprietà e Operazioni, slide 11 - 34;
  - Caratteristiche, slide 35 - 38.
- **Diagrammi dei package - Materiale didattico del corso di Ingegneria del Software:**  
<https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E01c.pdf>
  - Package e Dipendenze, slide 12 - 15.
- **Diagrammi di sequenza- Materiale didattico del corso di Ingegneria del Software:**  
<https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E02a.pdf>
  - Partecipanti e Segnali, slide 8 - 16;
  - Modellazione, slide 18 -21.
- **Design Pattern Comportamentali - Materiale didattico del corso di Ingegneria del Software:**  
<https://www.math.unipd.it/~tullio/IS-1/2019/Dispense/E08.pdf>
  - Strategy, slide 32 - 40.
- **Documentazione *Grafana*:**
  - Indicazioni di sviluppo plug-in  
<https://grafana.com/docs/grafana/latest/plugins/developing/development/>;
  - Stile di codifica per i plug-in in *Grafana*  
<https://grafana.com/docs/grafana/latest/plugins/developing/code-styleguide/>;
- **Libreria esterna consigliata per l'addestramento tramite *RL*:**  
<https://github.com/Tom-Alexander/regression-js>

- **Libreria esterna consigliata per l'addestramento tramite *SVM*:**  
<https://github.com/karpathy/svmjs>
- ***InfluxDB<sub>G</sub>* API<sub>G</sub>:**  
<https://docs.influxdata.com/influxdb/v1.7/tools/api/>

### 1.6.3 Legali

- **Licenza *MIT<sub>G</sub>*:**  
<https://opensource.org/licenses/MIT>

## 2 Tecnologie e librerie coinvolte

Di seguito vengono descritte brevemente le tecnologie e le librerie di terze parti utilizzate per il progetto *Predire in Grafana*.

### 2.1 Tecnologie

#### 2.1.1 *Grafana*

*Grafana* è un software di analisi *open-source* multipiattaforma di visualizzazione interattiva. Fornisce tavole, grafici e alerts per interfaccia web, collegati a sorgenti dati supportate e permette di essere espanso attraverso vari plug-in. *Grafana* è un software popolare per il monitoraggio di dati, per questo spesso è utilizzato in combinazione con *time series databases* quali *Prometheus*, *Graphite* e *InfluxDB*. Nel progetto *Predire in Grafana* questo software ricopre il ruolo di host per il plug-in, il quale estenderà le sue funzionalità con quelle fornite dal plug-in stesso.

- **Versione utilizzata:** 6.7.x;
- **Link per il download:** <https://grafana.com/grafana/download>

#### 2.1.2 *InfluxDB*

*InfluxDB* è un *time series database open-source* sviluppato da *InfluxData*. Assicura alte prestazioni ed è ottimizzato nelle operazioni di scrittura dei dati e nel parallelizzare e gestire la concorrenza di molteplici sorgenti di scrittura in contemporanea (sensori, metriche hardware...). Viene utilizzato nel progetto *Predire in Grafana* come principale *data source* per la lettura e scrittura dei dati da monitorare.

- **Versione utilizzata:** 1.7.x;
- **Link per il download:** <https://portal.influxdata.com/downloads/>

#### 2.1.3 *Telegraf*

*Telegraf* è un *server agent* *open-source* che permette di collezionare dati e metriche da stacks, sensori e sistemi di qualsiasi tipologia. Viene collegato ad *InfluxDB* per la raccolta di metriche hardware (CPU, RAM, Disk I/O...) da utilizzare come fonte di dati esterna per *Grafana* durante la fase di sviluppo.

- **Versione utilizzata:** 1.13.x;
- **Link per il download:** <https://portal.influxdata.com/downloads/>

#### 2.1.4 *React*

*React* è una libreria *JavaScript* per la creazione di interfacce utente. Può essere utilizzato come base nello sviluppo di applicazioni a pagina singola o mobile. Tuttavia, *React* si occupa solo del rendering dei dati sul *DOM*, pertanto la creazione di applicazioni *React* richiede generalmente l'uso di librerie aggiuntive per lo *state management* e il *routing*. Nell'ambito del progetto *Predire in Grafana*, *React* è stato utilizzato per la creazione dell'interfaccia utente sia del modulo interno (monitoraggio) che di quello esterno (addestramento).

- **Versione utilizzata:** 16.13.x;
- **Link Repository *NPM*:** <https://www.npmjs.com/package/react>

### 2.1.5 *TypeScript*

*TypeScript*<sub>G</sub> è un linguaggio di programmazione *open-source*. Si tratta di un super-set di *JavaScript*<sub>G</sub> che basa le sue caratteristiche su *ECMAScript 6*. Il linguaggio estende la sintassi di *JavaScript* in modo che qualunque programma scritto in *JavaScript* sia anche in grado di funzionare con *TypeScript* senza nessuna modifica. È progettato per lo sviluppo di grandi applicazioni ed è destinato a essere compilato in *JavaScript* per poter essere interpretato da qualunque *web browser* o *app*. *TypeScript* è stato scelto come linguaggio principale per la codifica del plug-in *Predire in Grafana*.

- **Versione utilizzata:** 3.7.5 o superiore;

### 2.1.6 *JSON*

*JSON* è il formato di serializzazione testuale richiesto per l'esportazione dei file predittore dal modulo esterno a quello interno per poter effettuare il monitoraggio dei dati. Esso è un formato estremamente diffuso per lo scambio dei dati in applicazioni *client-server*<sub>G</sub>. Si basa su oggetti, ovvero coppie chiave/valore, e supporta i tipi booleano, stringa, numero, e lista. È semplice e leggibile ad occhio umano, inoltre non necessita di alcun processo di compilazione particolare per essere modificato.

### 2.1.7 *CSV*

Il *comma-separated values (CSV)*<sub>G</sub> è un formato di file basato su file di testo, utilizzato per l'importazione ed esportazione (fogli elettronici o database) di una tabella di dati. Non esiste uno standard formale che lo definisca, ma solo alcune prassi più o meno consolidate. È il formato che viene richiesto per fornire i dati di addestramento al modulo esterno per produrre il file del predittore.

### 2.1.8 *HTML5*

*HTML5* è un linguaggio di *markup* per la strutturazione di pagine web. Viene utilizzato in *Predire in Grafana* assieme a *React* per definire gli elementi strutturali del plug-in.

### 2.1.9 *CSS3*

Il *CSS* è un linguaggio usato per definire la formattazione di documenti *HTML5*, *XHTML* e *XML*, come ad esempio siti web e le relative pagine web. Le regole per comporre il *CSS* sono contenute in un insieme di direttive. L'uso del *CSS* permette la separazione dei contenuti delle pagine *HTML* dal loro layout e permette una programmazione più chiara e facile da utilizzare, sia per gli autori delle pagine stesse sia per gli utenti, garantendo contemporaneamente anche il riutilizzo di codice ed una sua più facile manutenzione. Essendo il plug-in, *Predire in Grafana*, composto di elementi *HTML* viene utilizzato il *CSS* per la loro stilizzazione.

### 2.1.10 *NodeJS*

*Node.js* è una runtime di *JavaScript* *open-source* multipiattaforma, *event-driven*<sub>G</sub>, per l'esecuzione di codice *JavaScript*. Molti dei suoi moduli base sono scritti in *JavaScript*. È parte integrante della fase di codifica del plug-in *Predire in Grafana*, proprio per la sua funzionalità primaria.

- **Versione utilizzata:** 13.7.x;
- **Link per il download:** <https://nodejs.org/it/download/>

### 2.1.11 *Yarn*

*Yarn* è un gestore di pacchetti per il linguaggio di programmazione *JavaScript*. È il gestore di pacchetti rilasciato da *Facebook* con l'obiettivo di risolvere alcuni problemi e difetti rispetto a *NPM*. *Yarn* è inoltre più performante in termini di velocità e implementa un sistema di cache per installare *packages* precedentemente scaricati anche in assenza di connessione ad internet. Viene utilizzato dal gruppo per effettuare le operazioni di build del codice.

- **Versione utilizzata:** 1.22.x;
- **Link per il download:** <https://classic.yarnpkg.com/en/docs/install>

### 2.1.12 *SonarCloud*

*SonarCloud* è uno strumento per la *Continuous Code Quality*, basato su *SonarQube*. *SonarQube* è una piattaforma *open-source*, ampiamente utilizzata, per ispezionare periodicamente la qualità del codice sorgente, eventuali vulnerabilità e possibili situazioni problematiche presenti nell'architettura generale del programma. Supporta più di venti linguaggi differenti. È possibile collegarlo ad un *repository* pubblico con lo scopo di massimizzare il throughput e migliorare le releases del prodotto.

- **Link per l'utilizzo:** <https://sonarcloud.io/>

## 2.2 Librerie di terze parti

### 2.2.1 *MobX*

*MobX* è una libreria appositamente creata per *React* che permette la gestione dello *state* dei componenti in maniera semplice e scalabile. Permette di implementare il design pattern Observer che non è nativamente implementato in *React*.

- **Versione utilizzata:** 6.2.2;
- **Link Repository NPM:** <https://www.npmjs.com/package/mobx-react>

### 2.2.2 *ESLint*

*ESLint* è uno strumento di analisi statica per il codice *JavaScript*. È utilizzato per identificare gli errori presenti nel codice senza doverne fare la build, e determina anche la qualità del codice scritto. In particolare, ciò che viene segnalato sono errori di sintassi, import che non vengono mai usati all'interno del progetto, variabili dichiarate e non utilizzate e molto altro. Permette inoltre di essere configurato dallo sviluppatore con regole di analisi personalizzate.

- **Versione utilizzata:** 7.x.x;
- **Link Repository NPM:** <https://www.npmjs.com/package/eslint>

### 2.2.3 *Jest*

Framework di test *open-source* sviluppato da *Facebook* e viene utilizzato per testare codice scritto in *JavaScript*. Inizialmente usato per il testing su *React*, ora sta diventando sempre di più il punto di riferimento per i test anche per altri progetti *Babel*, *Angular*, *NodeJS*, *TypeScript*.

- **Versione utilizzata:** 25.1.0 o superiore;
- **Link Repository NPM:** <https://www.npmjs.com/package/jest>

#### 2.2.4 *Regression-js*

*Regression-js* è la libreria *JavaScript* utilizzata per implementare gli algoritmi di *ML Regressione Lineare*, *Regressione Esponenziale* e *Regressione Logaritmica* nel modulo esterno. Permette di effettuare il training e la produzione del predittore dato un file *CSV* contenente i dati per l'addestramento.

- **Link Repository GitHub:** <https://github.com/Tom-Alexander/regression-js>

#### 2.2.5 *Svmjs*

*Svmjs* è la libreria *JavaScript*, scritta da Andrej Karpathy, utilizzata per implementare l'algoritmo di *ML Support Vector Machine* nel modulo esterno. Permette di effettuare il training e la produzione del predittore dato un file *CSV* contenente i dati per l'addestramento.

- **Link Repository GitHub:** <https://github.com/karpathy/svmjs>

## 3 Setup

*Predire in Grafana*, consiste di due moduli separati. Uno esterno di addestramento e uno interno al sistema *Grafana*, di monitoraggio. Essendo entrambi funzionanti su browser, non verranno elencati i sistemi operativi che supporteranno i plug-in. Verrà, invece, sfruttato l'ambiente *UNIX\_G* per le dimostrazioni in quanto *OS* utilizzato per lo sviluppo.

### 3.1 Requisiti minimi di sistema

Vengono elencati qui i requisiti necessari al corretto utilizzo del prodotto *Predire in Grafana*.

#### 3.1.1 Prerequisiti

- *Grafana* v6.7.x;
- *NodeJS* v13.7.x;
- *Yarn* v1.22.x.

#### 3.1.2 Requisiti Hardware

- 2GB di memoria RAM;
- CPU dual-core.

Per maggiori informazioni è possibile leggere la documentazione fornita da *Grafana*:

<https://grafana.com/docs/grafana/latest/installation/requirements/>

#### 3.1.3 Browser

- *Google Chrome* v58 o superiore;
- *Microsoft Edge* v14 o superiore;
- *Mozilla Firefox* v54 o superiore;
- *Apple Safari* v10 o superiore.

## 3.2 Installazione

Questa sezione mostra tutti i passi necessari all'installazione del modulo di addestramento esterno e del modulo di monitoraggio, interno al sistema *Grafana*.

### 3.2.1 Installazione Training Module

Per installare ed avviare il modulo di addestramento è necessario:

- scaricare la cartella Training Module, in formato .zip, dalla seguente repository:

[https://github.com/Kero2375/proapes-predire-in-grafana/](https://github.com/Kero2375/proapes-predire-in-grafana;)

- estrarre il contenuto del file in una qualsiasi *directory*;
- aprire il prompt dei comandi (*Windows*) o il terminale (*Linux/Mac*);
- posizionarsi tramite il comando

```
cd /percorso
```

nella cartella in cui sono stati estratti i file;

- sarà necessario installare in primis tutte le dipendenze tramite il comando

```
yarn install
```

seguito poi dal comando

```
yarn start
```

e aprire su browser il percorso visibile sul terminale (Es. <http://localhost:3000>).

### 3.2.2 Installazione Prediction Module

Per scaricare ed avviare il modulo interno è necessario:

- scaricare la cartella Prediction Module dalla *repository GitHub*

```
https://github.com/Kero2375/proapes-predire-in-grafana
```

Se si decide di eseguire l'operazione da linea di comando scaricare la cartella tramite *GitG* con il comando

```
git clone https://github.com/Kero2375/proapes-predire-in-grafana.git
```

- estrarre i file nella cartella `../grafana/plugins` presente dentro la root di *Grafana*;
- aprire il terminale e raggiungere la cartella del plug-in per lanciare il comando

```
yarn install
```

con cui *Yarn* provvederà a scaricare tutte le dipendenze necessarie per buildare il plug-in correttamente.

Per rendere operativo il plug-in dopo il processo di installazione, eseguire sempre nel terminale della cartella dove è stato installato il plug-in il seguente comando:

```
yarn dev
```

oppure

```
yarn watch
```

Successivamente sarà necessario far partire il server di *Grafana* il cui launcher si trova nella cartella `/bin` dentro la root di *Grafana* oppure da linea di comando tramite

```
sudo systemctl start grafana-server
```

A questo punto, aprire il *browser* e collegarsi all'indirizzo del server dove è installato *Grafana* seguito dall'eventuale numero di porta utilizzata.

**Nota:** se il server *Grafana* è installato sulla stessa macchina dalla quale si vuole accedere e la porta utilizzata è quella di default, si può accedere tramite l'indirizzo

```
localhost:3000
```

Il plug-in sarà disponibile nella barra laterale della pagina di benvenuto di *Grafana* non appena verrà abilitato nella pagina di ricerca dei plug-in.

### 3.3 Ambiente di lavoro

Vengono esposte brevemente le procedure d'installazione degli altri strumenti necessari allo sviluppo del plug-in.

#### 3.3.1 *Grafana*

*Grafana* è la piattaforma su cui il plug-in *Predire in Grafana* lavora. Si tratta di una tecnologia essenziale per lo sviluppo ma non creata dal gruppo, quindi le istruzioni per il suo setup vengono rimandate alla documentazione ufficiale:

<https://grafana.com/docs/grafana/latest/installation/>

#### 3.3.2 *InfluxDB*

*InfluxDB* è la data source utilizzata per lo sviluppo del plug-in che si interfaccia con *Grafana*. Anche questa è una tecnologia fondamentale per lo sviluppo del prodotto ma non è creata dal gruppo quindi le istruzioni per il suo setup vengono rimandate alla documentazione ufficiale:

<https://docs.influxdata.com/influxdb/v1.7/introduction/installation/>

Potrebbe essere necessario modificare il file di configurazione di *InfluxDB* reperibile nella directory

/etc/influxdb/influxdb.conf

#### 3.3.3 *NodeJS*

*NodeJS* è la runtime *JavaScript* orientata agli eventi necessaria alla compilazione del codice. Si tratta di una tecnologia di supporto essenziale per lo sviluppo dei plug-in di *Grafana* i quali sono scritti proprio in linguaggio *JavaScript* o *TypeScript*. Le istruzioni per il suo setup vengono rimandate alla documentazione ufficiale:

<https://nodejs.org/it/download/>

#### 3.3.4 *React*

*React* è un requisito fondamentale poiché utilizzato dall'applicazione che il plug-in andrà ad estendere. Non è necessario eseguire nessuna chiamata esplicita o utilizzare librerie grafiche di *React* ma è necessario solamente utilizzare la variabile

props

per collegare il front-end al controller in *JavaScript/TypeScript*.

#### 3.3.5 Librerie necessarie

*Regression-js* e *Svmjs* sono le librerie necessarie all'implementazione degli algoritmi di *ML Regressione Lineare*, *Regressione Esponenziale*, *Regressione Logaritmica* e *Support Vector Machine* nel modulo d'addestramento. Sono state fornite dall'azienda proponente ed è possibile reperirle alle seguenti *repository GitHub* e in cui sono contenute anche le istruzioni per la loro importazione nel progetto:

- **Regression-js:** <https://github.com/Tom-Alexander/regression-js.git>
- **Svmjs:** <https://github.com/karpathy/svmjs.git>

## 4 Test

Questo capitolo ha lo scopo di indicare agli sviluppatori come controllare in che modo opera il codice e la sua sintassi. Vengono di seguito esposti gli strumenti utilizzati, nello sviluppo del plug-in *Predire in Grafana*, per effettuare i test di analisi statica e dinamica. Essendo il prodotto suddiviso in due moduli è bene specificare che tutte le indicazioni sull'utilizzo dei seguenti strumenti di *testing* saranno le medesime sia per il modulo esterno di addestramento sia per quello interno di monitoraggio.

### 4.1 Jest

Il framework di testing *Jest* viene utilizzato per effettuare i test di unità. La configurazione del framework è presente nella directory principale del progetto con il nome

```
jest.config.js
```

Per eseguire i test sarà sufficiente eseguire il comando

```
yarn test
```

*Yarn* si occuperà automaticamente di effettuare tutti i test specificati nel file di configurazione.

### 4.2 ESLint

I test di analisi statica del codice, ovvero i controlli sulla qualità del codice scritto, vengono effettuati dallo strumento *ESLint*. Per effettuare i test sul codice è necessario eseguire il comando

```
eslint
```

seguito dal percorso che conduce alla directory contenente il file di configurazione. Ad esempio

```
eslint ../grafana/plugins/react-app/config_file.js
```

In presenza di errori il sistema provvederà a segnarne il tipo e la locazione. In caso contrario, invece, non verrà notificato nulla all'utilizzatore.

### 4.3 SonarCloud

Per svolgere l'attività di *code coverage* è stato scelto di utilizzare *SonarCloud*. *SonarCloud* è una piattaforma *cloud* basata su *SonarQube* per il controllo continuo della qualità del codice. Esegue revisioni automatiche del codice facendo analisi statica, rilevazione di bug, code smells e vulnerabilità sulla sicurezza. Non vi sono comandi da utilizzare, una volta configurato sul repository eseguirà la sua attività ogni qualvolta verranno eseguiti i test.

### 4.4 GitHub Actions

Il servizio di *Continuous Integration* che è stato deciso di utilizzare è *GitHub Actions*, fornito appunto da *GitHub*. Questo permette di creare dei workflow personalizzati, ovvero dei processi automatici creati sulla base delle proprie esigenze. Ciò ha l'obiettivo di automatizzare il ciclo di vita di sviluppo del software grazie ad un'ampia gamma di strumenti e servizi.

## 5 Architettura del prodotto

L'architettura generale del progetto *Predire in Grafana* è modellata in due moduli distinti, entrambi aderenti al design pattern architettonicale *Model-View-ViewModel (MVVM)*. Il primo modulo, esterno al sistema *Grafana* e chiamato **Training Module**, effettuerà l'addestramento su un dataset importato dall'utente, producendo un file *JSON* contenente la definizione del predittore.

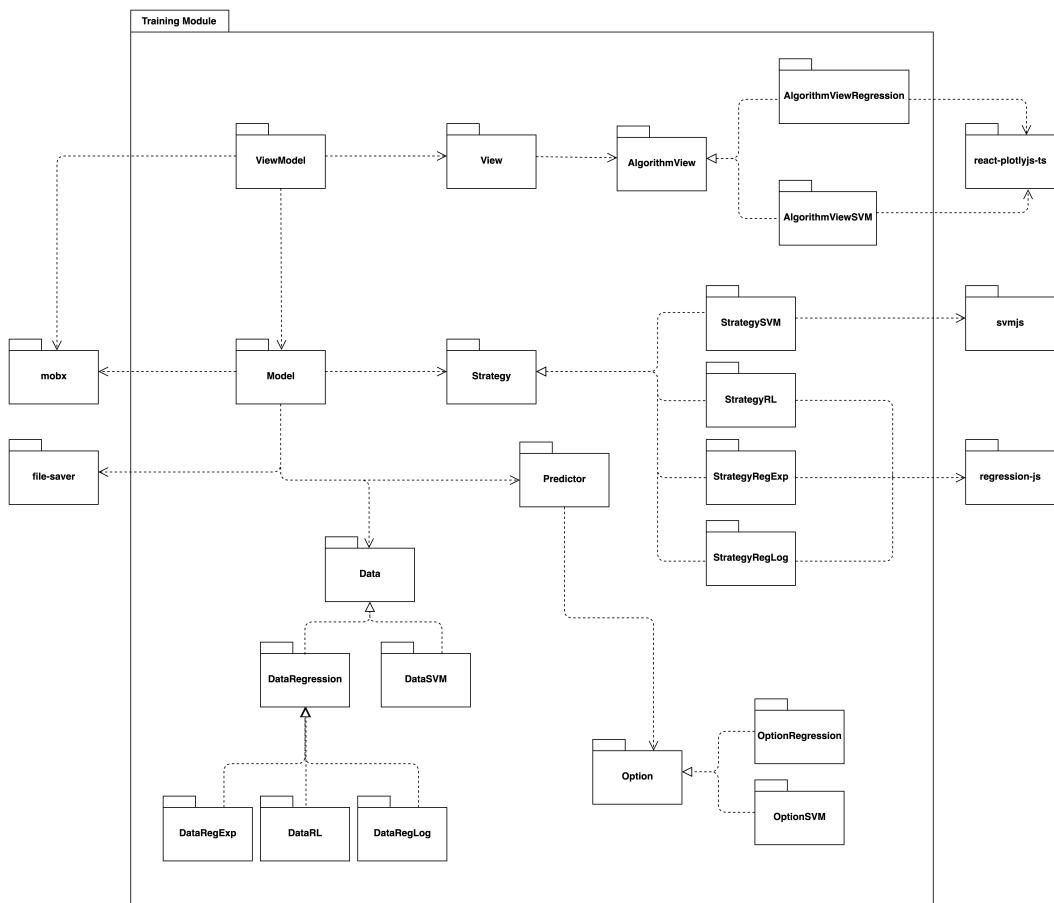
Il secondo modulo, questa volta interno a *Grafana* e chiamato **Prediction Module**, associerà il predittore prodotto dal modulo esterno al flusso dati monitorato appunto in *Grafana*. Grazie alle ultimissime aggiunte apportate al prodotto sarà inoltre possibile effettuare l'addestramento, previsto nativamente nel Training Module, anche all'interno di Prediction Module stesso.

Le principali motivazioni della scelta del pattern architettonicale *MVVM* sono le seguenti:

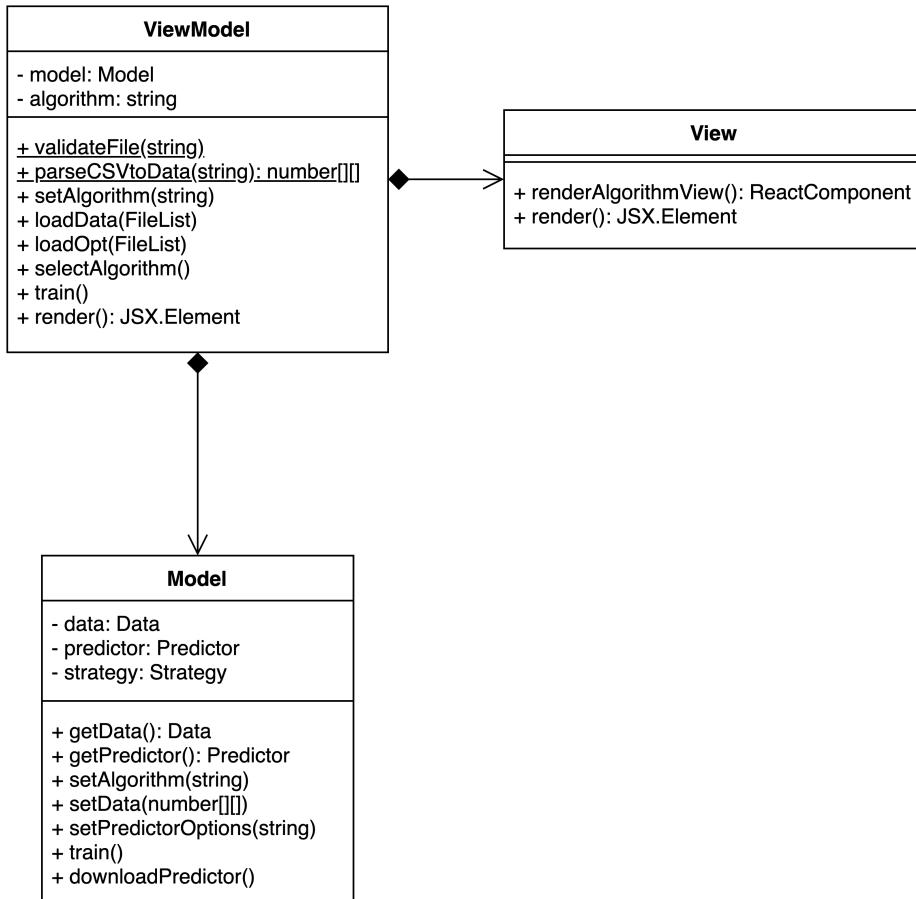
- entrambi i moduli sono scritti utilizzando il framework *React* ed è stato ritenuto il pattern maggiormente integrabile per questo framework;
- forte riutilizzo di componenti (modello e vista) in altri contesti, senza la necessità di doverli modificare;
- forte disaccoppiamento tra *business logic* e *presentation logic*.

### 5.1 Architettura Training Module

#### 5.1.1 Pattern architettonicale: MVVM



**Figura 1:** Diagramma dei package del Training Module



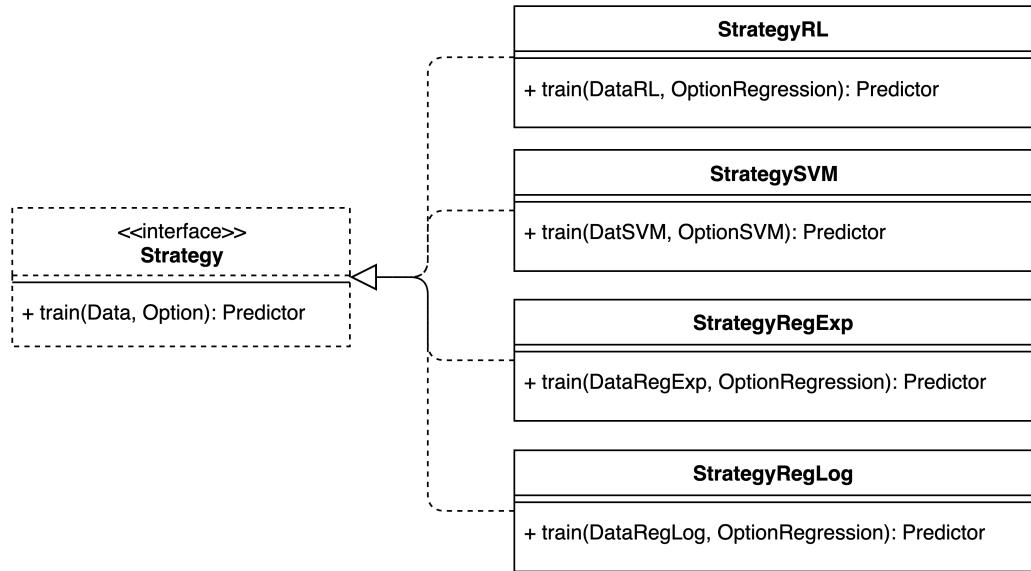
**Figura 2:** Implementazione pattern MVVM nel Training Module

Come specificato in precedenza, per questo modulo è stato utilizzato il pattern architettonico *MVVM*. Il passaggio di dati dal *Model* alle *View* avviene tramite la modifica di una variabile `props` iniettata dal *Controller (ModelView)*. Il *Controller* passa tramite queste `props` i giusti metodi da chiamare quando l'utente interagisce con la vista. Questo utilizzo delle `props` favorisce la separazione tra la *presentation logic* e la *business logic*. Inoltre il *ViewModel* possiede anche un riferimento al *Model* tramite un campo dati di tipo *Model* con il quale comunica col modello chiamandone i metodi pubblici.

### 5.1.2 MobX

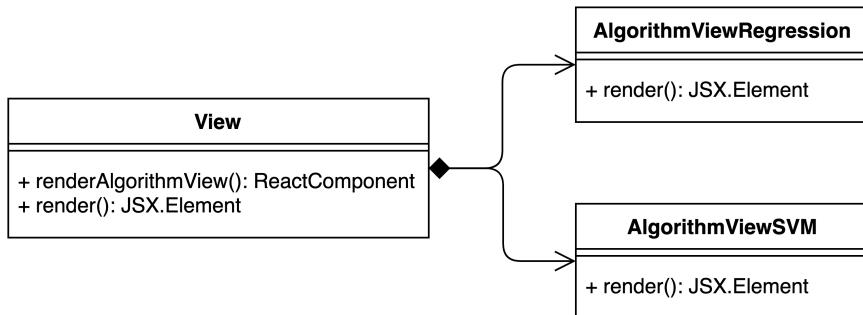
Per l'aggiornamento dei dati riferiti al *Model* e la notifica del cambiamento di questi dati al *Controller* abbiamo utilizzato la libreria *MobX*. Essa ci ha permesso di implementare il meccanismo degli *Observer* non nativamente supportato in *React*. Nel dettaglio, essa viene utilizzata per marcare come *Observable* i campi dati del *Model* e come *Observer* il *ViewModel*. In questo modo ogni qualvolta lo stato del *Model* viene aggiornato a seguito di qualche metodo, la sua modifica viene recepita istantaneamente anche dal *ViewModel* che potrà quindi aggiornare la *View*.

### 5.1.3 Architettura di dettaglio: Strategy



**Figura 3:** Strategy per la scelta dell'algoritmo

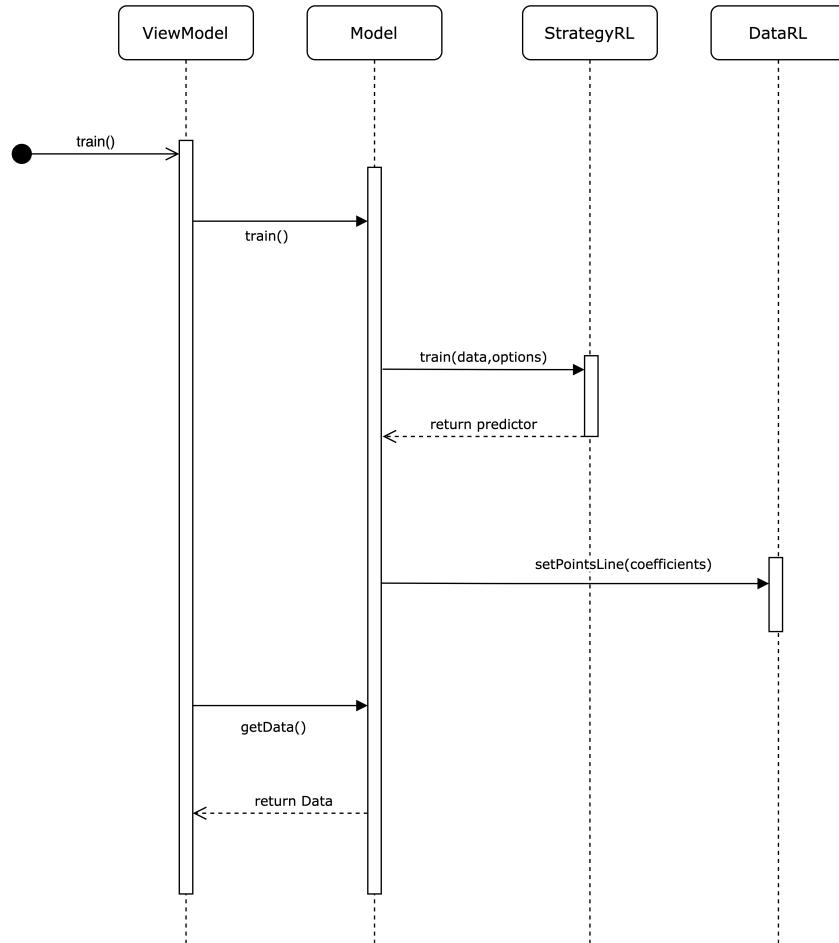
Il pattern Strategy si compone di un metodo fondamentale `train()` per l'esecuzione dell'addestramento. Esso è implementato nella maniera più opportuna in base alle necessità di ogni algoritmo di *ML*. Come ci si aspetta, la Strategy è istanziata solamente una volta che l'utente ha scelto l'algoritmo, in maniera dinamica e attraverso un array associativo. Questo permette di associare la stringa selezionata dall'utente con la strategia concreta più adatta alla scelta.



**Figura 4:** Viste in base all'algoritmo scelto.

In questo caso invece, grazie allo Strategy è possibile scegliere quale vista specializzata renderizzare. Questo avviene grazie ad un array associativo il quale permette di associare la stringa selezionata dall'utente con la strategia concreta più adatta alla scelta. Nel dettaglio avviene grazie a `renderAlgorithmView()` il quale inizialmente non visualizza nulla, ma una volta selezionato l'algoritmo renderizza la vista in modo specifico. Il metodo `render()`, invece, renderizza tutti gli altri elementi *HTML* della vista che non dipendono dalla scelta dell'algoritmo.

#### 5.1.4 train()

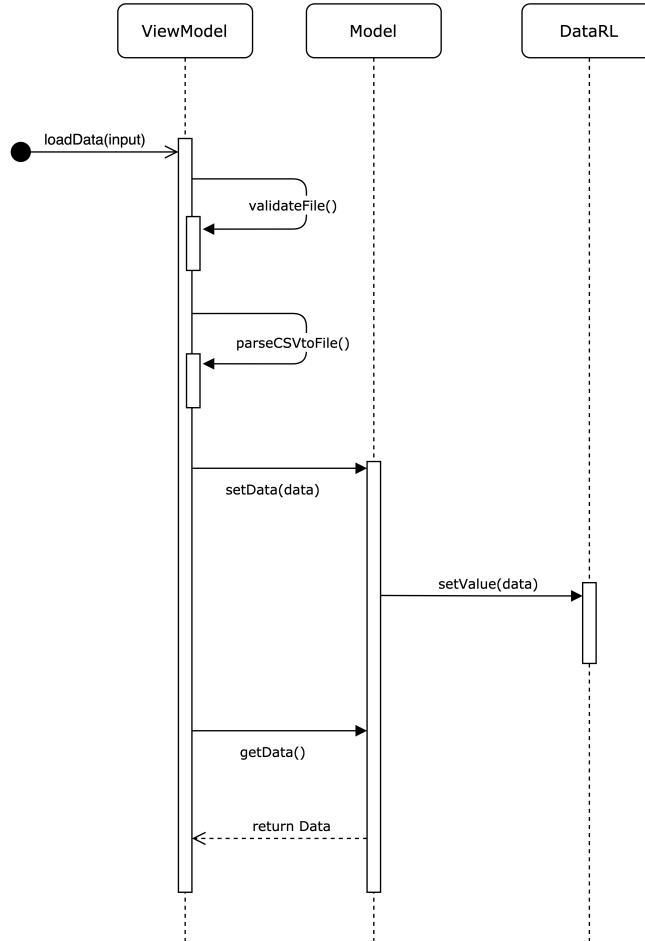


**Figura 5:** Diagramma di sequenza per la funzione `train()`

La funzione `train()` è una delle due funzioni chiave del modulo esterno ed è adibita all’addestramento tramite l’algoritmo di *Machine Learning* scelto dall’utente. Essa viene invocata nel `ViewModel` al click dell’utente nella UI, e da qui viene chiamata nel `Model`. In base all’algoritmo scelto precedentemente nella UI, verrà utilizzata la relativa `Strategy` e verranno passati i dati (memorizzati nell’oggetto `Data`) e le opzioni di configurazione (memorizzate nell’oggetto `Options`). Verrà ritornato quindi il predittore rappresentato dai coefficienti dell’equazione di predizione dell’algoritmo. A questo punto, il metodo `setPointsLine()` calcolerà i punti per disegnare la retta all’interno del grafico, che potranno poi essere eventualmente prelevati tramite il metodo `getData()`.

Nel diagramma di sequenza, per semplicità, è stato preso di esempio solo l’algoritmo di *Regressione Lineare*. Naturalmente anche gli altri algoritmi forniti dal plug-in aderiranno allo stesso schema di funzionamento.

### 5.1.5 loadData()



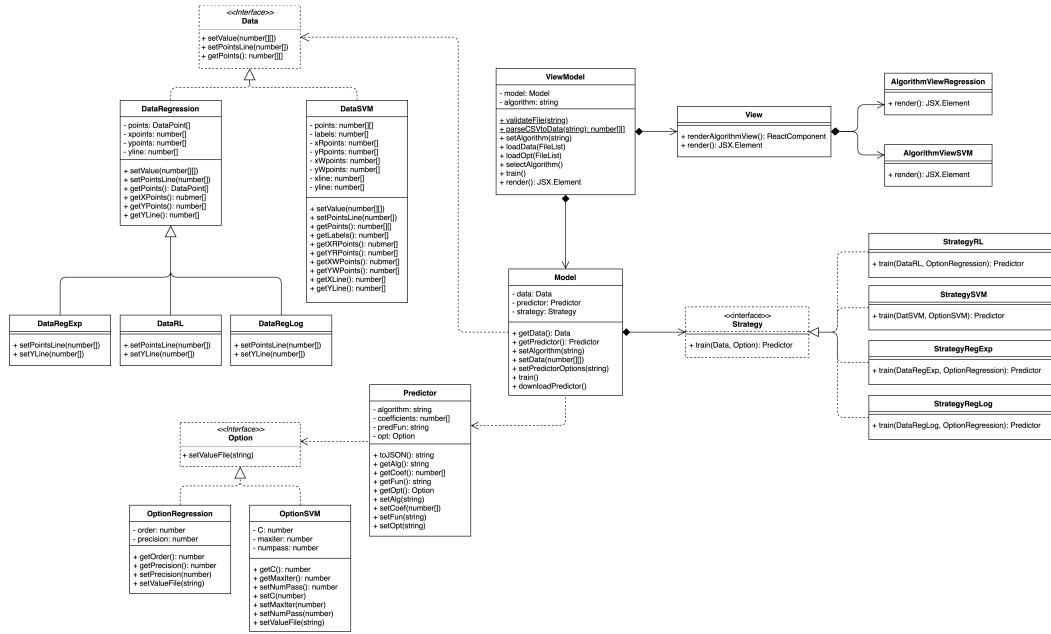
**Figura 6:** Diagramma di sequenza per la funzione `loadData()`

La seconda funzione degna di nota per quanto concerne il modulo esterno è la funzione `loadData()`. Adibita al caricamento dei dati di addestramento nel modulo, viene invocata all'inserimento del file *CSV* nella UI. Dopo un iniziale verifica sulla correttezza del formato del file inserito (`validateFile()`), e un successivo parsing (`parseCSVtoFile()`), per la trasformazione del contenuto in dati utilizzabili dal plug-in, verrà invocato il metodo `setData()` per il salvataggio dell'oggetto *Data* dell'algoritmo di *Machine Learning* scelto dall'utente. I dati salvati potranno quindi essere prelevati tramite il metodo `getData()`.

Nel diagramma di sequenza, per semplicità, è stato preso di esempio solo l'algoritmo di *Regressione Lineare*. Naturalmente anche gli altri algoritmi forniti dal plug-in aderiranno allo stesso schema di funzionamento.

### 5.1.6 Diagramma delle classi

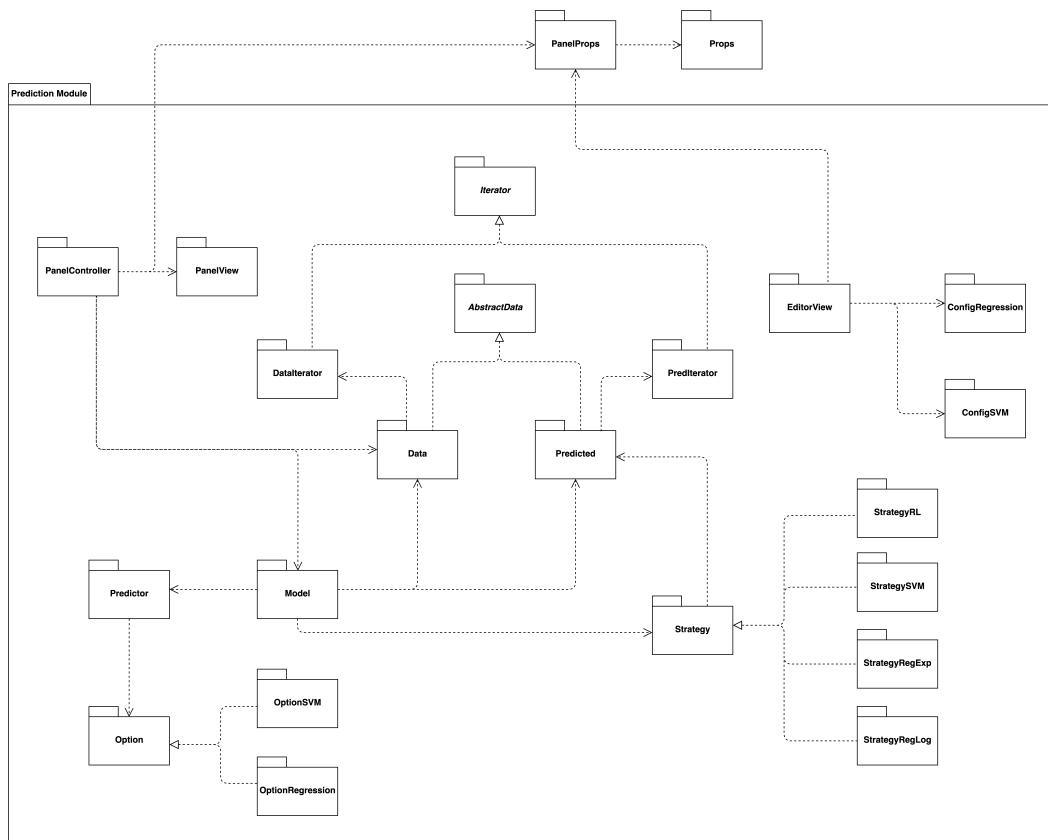
Di seguito vengono riportate le classi nel dettaglio con il relativo diagramma



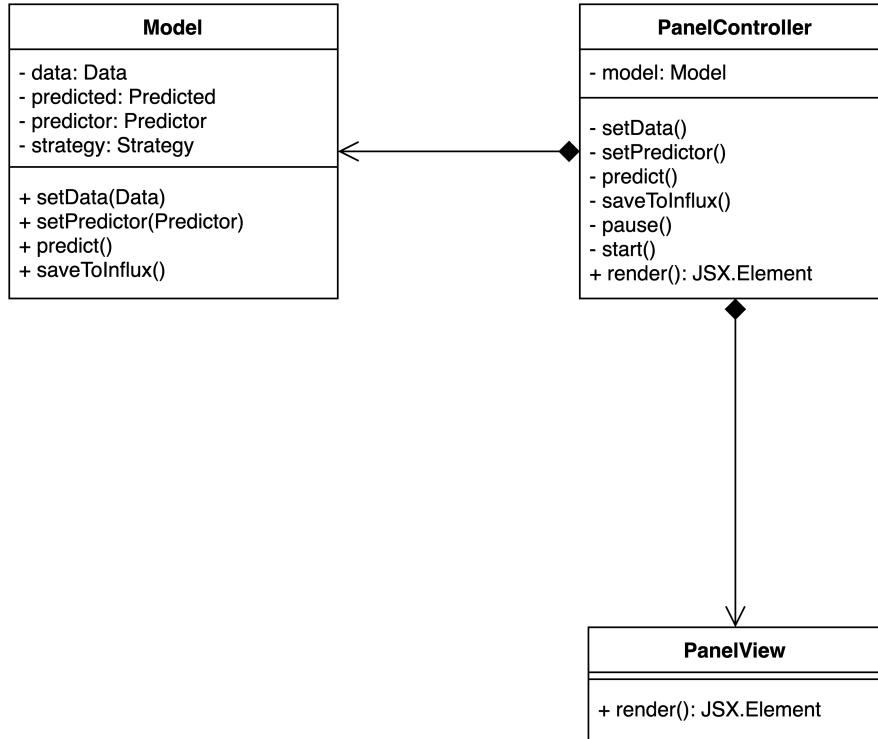
**Figura 7:** Diagramma delle classi del Training Module

## 5.2 Architettura Prediction Module

### 5.2.1 Pattern architetturale: MVVM



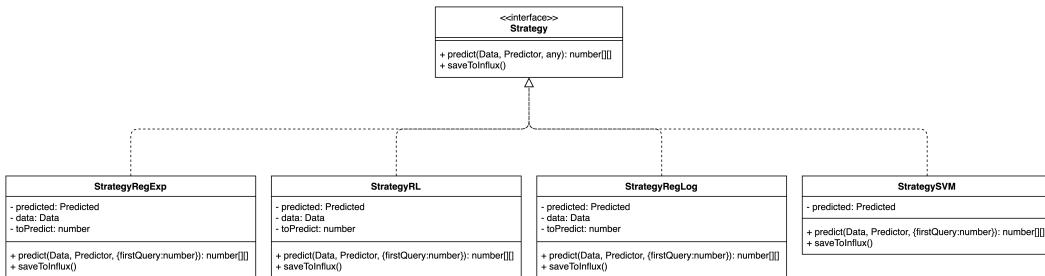
**Figura 8:** Diagramma dei package del Prediction Module



**Figura 9:** Implementazione pattern MVVM Prediction Module

In modo analogo al Training Module, anche per il Prediction Module è stato utilizzato il pattern *MVVM* per quanto riguarda il *Panel*. Esso è composto dal modello *Model*, controller *PanelController* e dalla vista *PanelView*. Il *PanelController* comunica con la parte di *Editor* tramite la variabile *props* che viene istanziata automaticamente da *Grafana* e viene utilizzata per memorizzare i risultati delle query effettuate sul datasource<sub>G</sub>. Per quanto riguarda l'*EditorView* invece, è stato deciso di non utilizzare un pattern architetturale in quanto ha il solo compito di modificare e aggiornare la variabile *props*. Sarà poi compito del pannello principale prendersi carico dei vari controlli, prelevare le query (*setData()*), effettuare la predizione (*predict()*), e memorizzare nuovamente i dati nel database (*saveToInflux()*). Inoltre, sarà proprio il *Panel* che verrà aggiornato continuamente ad ogni intervallo di tempo specificato dall'utente tramite un apposito selettore già implementato dalla piattaforma *Grafana*.

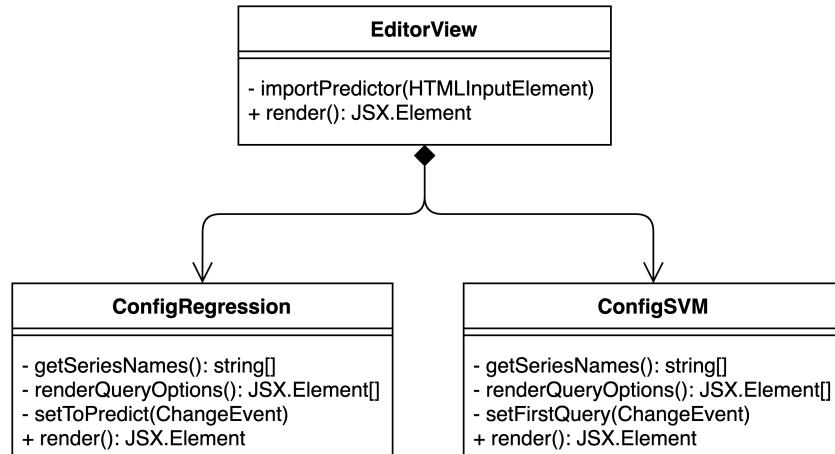
### 5.2.2 Architettura di dettaglio: Strategy



**Figura 10:** Strategy per la ricostruzione della funzione di predizione

Anche per il Prediction Module è stato utilizzato il design pattern *Strategy*. Lo scopo è differenziare gli algoritmi di *Regressione Lineare*<sub>G</sub>, *Regressione Esponenziale*<sub>G</sub>, *Regressione*

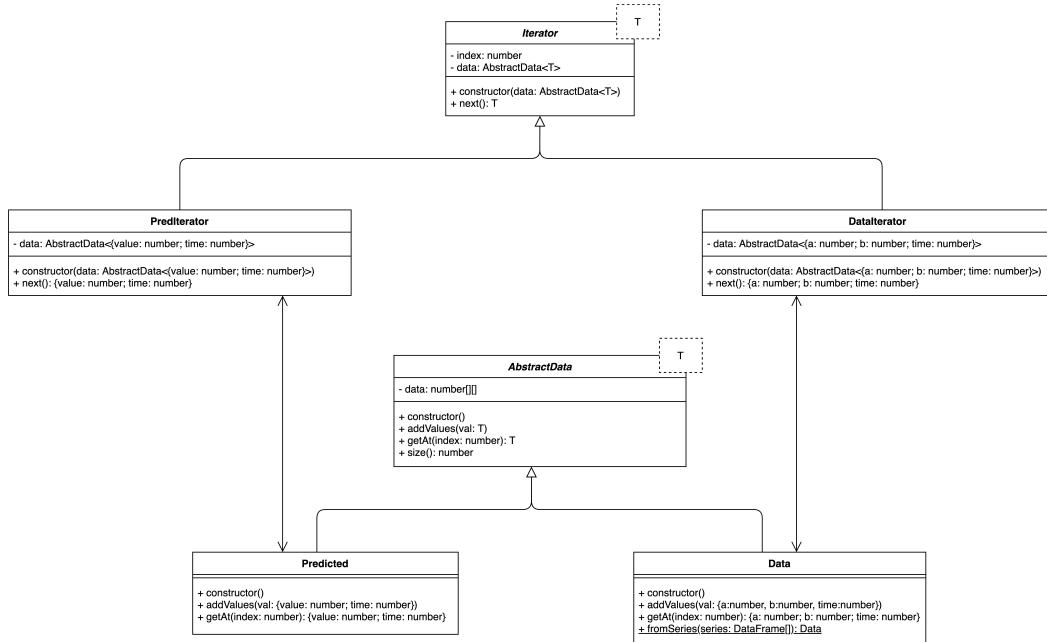
*Logaritmica* e *Support Vector Machine* in modo da poter ricreare la funzione di predizione calcolata tramite i coefficienti passati dal file *JSON*. Nello specifico, questa scelta è fatta a runtime utilizzando un array associativo. Si compone di un solo metodo `predict()` che assolve questo scopo. Inoltre viene implementato da ogni Strategy specifica il metodo `saveToInflux()` per salvare i valori della predizione effettuata, all'interno del datasource impostato.



**Figura 11:** Viste in base al predittore importato

Grazie allo Strategy è possibile scegliere quale vista specializzata dell'*EditorView* renderizzare. *ConfigRegression* e *ConfigSVM* vengono qui istanziate solo se prima è stato importato il predittore tramite il metodo `importPredictor()`, altrimenti viene renderizzato un componente vuoto con il metodo `render()`. Esse hanno il compito di visualizzare la funzione in formato stringa e possono andare a modificate le `opt` del predittore, le quali contengono i metodi `toPredict()` oppure `firstQuery()`. *ConfigRegression* e *ConfigSVM* andranno quindi a renderizzare queste configurazioni e, al loro cambiamento, riporteranno le modifiche alle `props`. *ConfigRegression* è utilizzata per tutti i tipi di regressione (*Lineare*, *Esponenziale* e *Logaritmica*) in quanto interscambiabili per quanto concerne la visualizzazione.

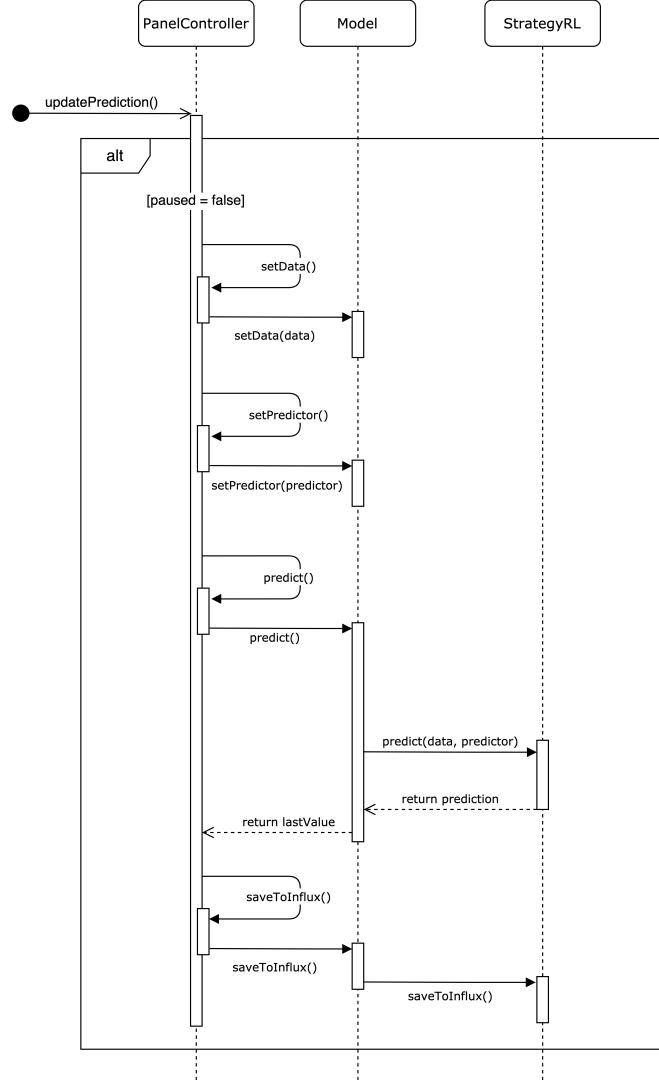
### 5.2.3 Architettura di dettaglio: Iterator



**Figura 12:** Iterator per l'accesso sequenziale alle strutture dati

All'interno del Prediction Module è stato utilizzato il pattern *Iterator* per quanto riguarda le strutture dati *Data* e *Predicted*, utilizzate rispettivamente per contenere i dati provenienti dal datasource per e per immagazzinare i dati provenienti dalle previsioni effettuate dal plug-in. Le previsioni di *Predicted* sono effettuate a partire dai dati contenuti in *Data* stesso, quindi la loro struttura sarà pressoché identica. Per questa ragione derivano entrambe dalla stessa classe astratta *AbstractData*. Il pattern *Iterator* viene quindi implementato in maniera classica, fornendo un metodo di accesso sequenziale agli elementi che formano un oggetto composito, sia che sia di tipo *Data* (*DataIterator*), sia che sia di tipo *Predicted* (*PredIterator*).

### 5.2.4 updatePrediction()



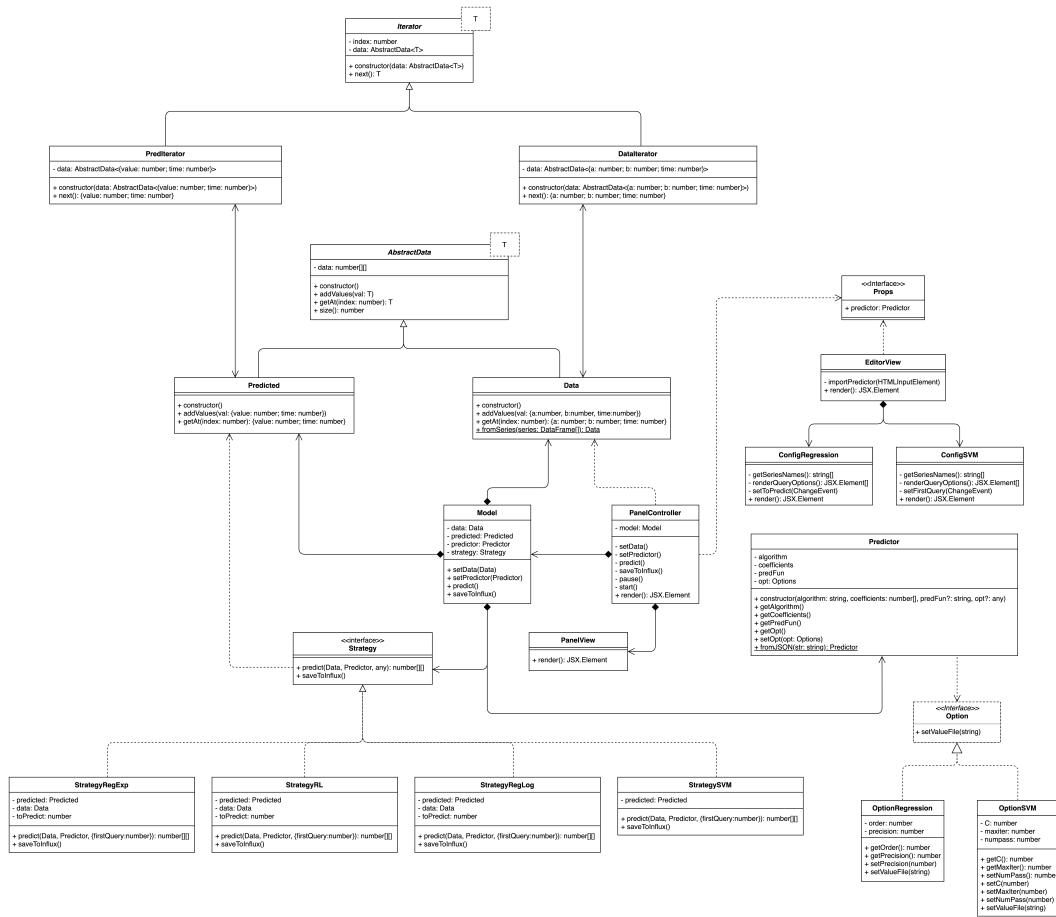
**Figura 13:** Diagramma di sequenza per la funzione `updatePrediction()`

Nel diagramma di sequenza viene descritta la funzione `updatePrediction()`. Essa è adibita all'aggiornamento in tempo reale della predizione in fase di monitoraggio attivo (`paused=false`). Una volta prelevati i dati da monitorare dal datasource (`setData()`) e settato il predittore precedentemente calcolato nel modulo di addestramento (`setPredictor()`) verrà invocato il metodo `predict()` che calcolerà i valori predetti da visualizzare successivamente nel grafico di *Grafana*. Questo metodo avrà in input i dati da monitorare e il predittore. Quest'ultimo conterrà al suo interno anche l'informazione relativa all'algoritmo di *Machine Learning* utilizzato precedentemente nella fase di addestramento. L'output risultante dall'esecuzione di `predict()` sarà quindi l'ultimo valore predetto calcolato nell'intervallo di tempo scelto dall'utente nella dashboard di *Grafana*. I valori verranno costantemente salvati nel database (`saveToInflux()`) per poi essere mostrati nel relativo grafico di visualizzazione.

Nel diagramma di sequenza, per semplicità, è stato preso di esempio solo l'algoritmo di *Regressione Lineare*. Naturalmente anche gli altri algoritmi forniti dal plug-in aderiranno allo stesso schema di funzionamento.

### 5.2.5 Diagramma delle classi

Di seguito vengono riportate le classi nel dettaglio con il relativo diagramma



**Figura 14:** Diagramma delle classi del Prediction Module

**Nota:** all'interno del Prediction Module è stata implementata in ultima istanza la possibilità di eseguire l'addestramento direttamente all'interno del software *Grafana*. L'architettura che permette questa operazione è analoga a quella presentata nella sezione relativa al Training Module §5.1, perciò non ne viene fornita ulteriore documentazione. Vengono comunque sfruttate le stesse strutture dati (`Predictor` e `Options`) per immagazzinare rispettivamente le informazioni relative al predittore e alle opzioni di configurazione dell'algoritmo di *ML*. Il resto può essere inteso come un modulo completamente distaccato del quale si è fatto il *porting*.

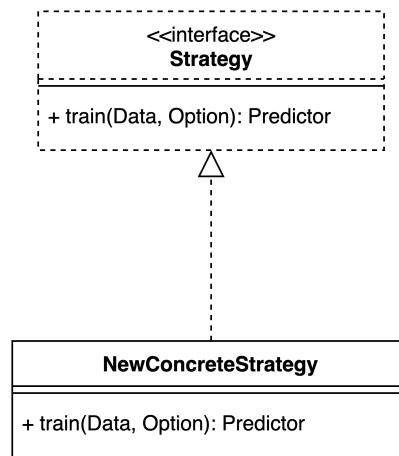
## 6 Estendere *Predire in Grafana*

Vengono esposte in questa sezione le modalità con le quali è possibile estendere il prodotto *Predire in Grafana*. Le aggiunte in questo caso, sono da intendersi come aggiunte relative agli algoritmi di *Machine Learning* che il prodotto nativamente supporta.

### 6.1 Training Module

Per aggiungere una nuova tipologia di algoritmo di *Machine Learning* nel Training Module bisognerà:

- all'interno del file `strategies.ts`, aggiungere una nuova Strategy (*NewConcreteStrategy*) relativa al nuovo algoritmo. Bisognerà quindi aggiornare l'array associativo contenente l'identificativo degli algoritmi con l'id relativo al nuovo algoritmo;



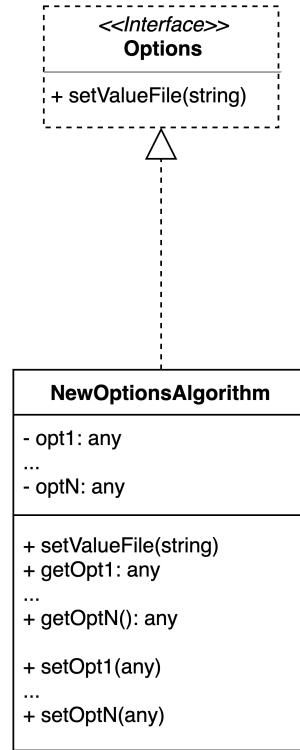
**Figura 15:** Aggiunta di una nuova Strategy `NewConcreteStrategy`

- aggiungere una nuova View (*NewView*) per la vista specializzata, aggiornando il metodo `renderAlgorithmView()` per renderizzare la nuova vista appena aggiunta;



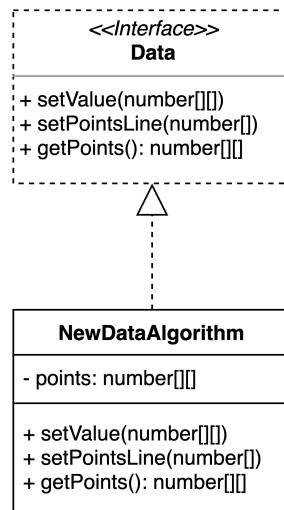
**Figura 16:** Aggiunta di una nuova vista specializzata `NewView`

- aggiungere una nuova classe di Options relative al nuovo algoritmo (*NewOptionAlgorithm*). Queste opzioni di configurazione saranno specifiche per ogni nuovo algoritmo di addestramento e dipenderanno dall'eventuale libreria di *Machine Learning* utilizzata. Per generalizzare l'aggiunta, nella figura vengono indicate come una lista 1..N di nuove opzioni;



**Figura 17:** Aggiunta di una nuova classe di Options `NewOptionAlgorithm`

- una nuova classe Data riferita ai dati dell'algoritmo da rappresentare nel grafico (*NewDataAlgorithm*). La tipologia dei dati dipenderà dall'algoritmo implementato e da quali informazioni sarà necessario rappresentare. Andranno implementati i metodi `setValue(number[][])`, `setPointsLine(number[])` e `getPoints()` per ottenere le informazioni dei dati, impostarli e rappresentarli come punti nel grafico.

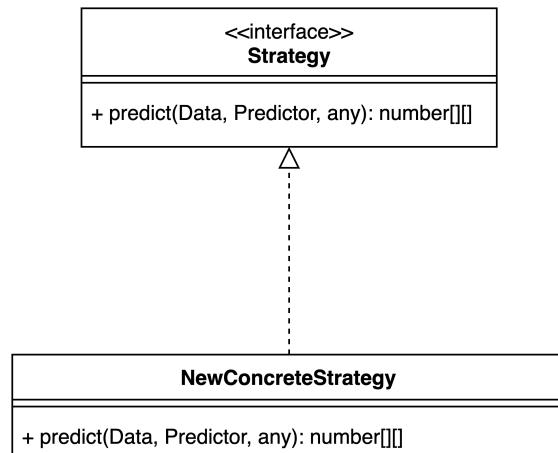


**Figura 18:** Aggiunta di una nuova classe Data `NewDataAlgorithm`

## 6.2 Prediction Module

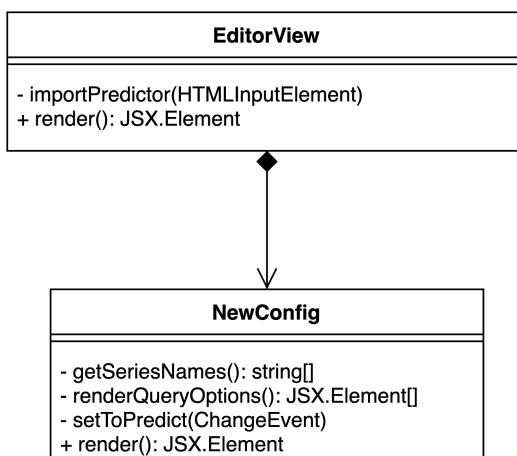
Per aggiungere una nuova tipologia di algoritmo di *Machine Learning* all'interno del Prediction Module bisognerà:

- aggiungere una nuova Strategy (*NewConcreteStrategy*) relativa al nuovo algoritmo. Bisognerà quindi aggiornare l'array associativo contenente l'identificativo degli algoritmi con l'id relativo al nuovo algoritmo;



**Figura 19:** Aggiunta di un nuovo algoritmo *NewConcreteStrategy*

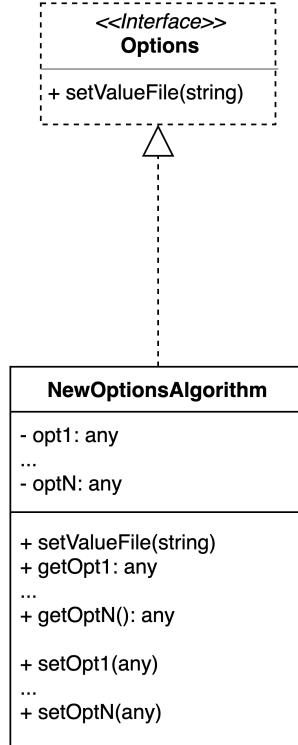
- aggiungere il file di configurazione relativo al nuovo componente *React* (*NewConfig*). Questo componente permetterà di editare la configurazione dell'algoritmo (come per esempio quali *query* predire) e dipenderà dall'algoritmo stesso che si vuole implementare.



**Figura 20:** Aggiunta della configurazione di un nuovo componente *NewConfig*

- aggiungere una nuova classe di Options relative al nuovo algoritmo (*NewOptionAlgorithm*). Queste opzioni di configurazione saranno specifiche per ogni nuovo algoritmo.

Per generalizzare l'aggiunta, nella figura vengono indicate come una lista 1..N di nuove opzioni;



**Figura 21:** Aggiunta di una nuova classe di Options `NewOptionAlgorithm`

**Nota:** all'interno del Prediction Module è stata implementata in ultima istanza la possibilità di eseguire l'addestramento direttamente all'interno del software *Grafana*. L'architettura che permette questa operazione è analoga a quella presentata nella sezione relativa al Training Module §5.1, perciò non ne viene fornita ulteriore documentazione. Vengono comunque sfruttate le stesse strutture dati (*Predictor* e *Options*) per immagazzinare rispettivamente le informazioni relative al predittore e alle opzioni di configurazione dell'algoritmo di *ML*. Il resto può essere inteso come un modulo completamente distaccato del quale si è fatto il *porting*. Nel caso si volessero quindi estendere gli algoritmi disponibili per l'addestramento interno al Prediction Module, si potrà fare riferimento a quanto descritto in §6.2 in quanto completamente analogo.

## A Glossario

### A

**API:** Acronimo per *Application Programming Interface*. Indica un insieme di procedure (in genere raggruppate per strumenti specifici) atte all'espletamento di un dato compito. Spesso tale termine designa le librerie software di un linguaggio di programmazione.

### C

**Client-Server:** Architettura di rete nella quale genericamente un computer *client* o terminale si connette ad un *server* per la fruizione di un certo servizio.

**Continuous Code Quality:** Aspetto della *Continuous Integration* **G** che include il run di test automatici e l'analisi statico-dinamica del codice ad ogni build. È un modo per assicurare la qualità del codice.

**Code Coverage:** Nel testing è la misura di quante linee/blocchi/archi del codice vengono eseguiti mentre i test automatici sono in esecuzione.

**Continuous Integration:** Pratica nella quale lo sviluppo avviene attraverso un sistema di versionamento; consiste nell'allineare frequentemente gli ambienti di lavoro degli sviluppatori verso l'ambiente condiviso. Acronimo usato: CI.

**CSV:** *Comma-separated values* è un formato di file basato su file di testo utilizzato per l'importazione ed esportazione di una tabella di dati.

### D

**Datasource:** È un nome assegnato alla connessione impostata su un database da un server.

**DevOps:** Metodo di sviluppo del software che punta alla comunicazione, collaborazione e integrazione tra sviluppatori e addetti alle operations della *information technology (IT)*. DevOps vuole rispondere all'interdipendenza tra sviluppo software e *IT operations*, puntando ad aiutare un'organizzazione a sviluppare in modo più rapido ed efficiente prodotti e servizi software. Molta importanza viene data inoltre alla qualità.

**DOM:** *Document Object Model*, letteralmente modello a oggetti del documento, è una forma di rappresentazione dei documenti strutturati come modello orientato agli oggetti.

### E

**EsLint:** Strumento di analisi del codice statico per identificare i modelli problematici trovati nel codice *JavaScript* **G**.

- <https://eslint.org/>.

## G

**Git:** Sistema di controllo della versione distribuito per tenere traccia delle modifiche al codice sorgente durante lo sviluppo del software. Progettato per coordinare il lavoro tra i programmati, ma utilizzabile per tenere traccia delle modifiche in qualsiasi set di file. I suoi obiettivi sono velocità, integrità dei dati e supporto per flussi di lavoro distribuiti e non lineari.

**GitHub:** Servizio di hosting per progetti software. Implementazione dello strumento di controllo versione distribuito *Git*<sub>G</sub>.

**Grafana:** Sistema impiegato dall’azienda *Zucchetti S.p.A.* per il monitoraggio delle applicazioni. Tale funzionalità diviene di particolare rilevanza all’interno di uno scenario di tipo *DevOps*<sub>G</sub> possedendo due compiti:

- controllo della salute del sistema e successiva verifica che le performance rientrino all’interno di parametri precisi;
- identificazione dei punti deboli da correggere da parte del team di sviluppo;
- fornitura di elementi per definire la scala delle priorità nelle migliorie e nelle nuove implementazioni.

L’ambito di monitoraggio è abbastanza vasto; nel caso si verifichino rilevazioni di situazioni estreme scattano avvisi (e-mail) destinati ai responsabili.

- Indicazioni di sviluppo plug-in  
<https://grafana.com/docs/grafana/latest/plugins/developing/development/>;
- Stile di codifica per i plug-in in *Grafana*  
<https://grafana.com/docs/grafana/latest/plugins/developing/code-styleguide/>;

## I

**InfluxDB:** è un *open-source time series database (TSDB)* sviluppato da InfluxData. È scritto in *Go* e ottimizzato per l’high-availability storage di serie temporali riferite ad operazioni di monitoraggio dati.

- <https://docs.influxdata.com/>

## J

**JavaScript:** Linguaggio di scripting orientato agli oggetti e agli eventi, utilizzato nella programmazione Web sia lato client che lato server.

**Jest:** Framework di test *open-source* sviluppato da *Facebook* utilizzato per testare codice scritto in *JavaScript*.

- <https://github.com/facebook/jest>

## M

**MIT License:** Licenza per il software libero, nata al *Massachusetts Institute of Technology (MIT)* a fine anni ’80. Pone restrizioni in merito al riuso del codice protetto da tale licenza.

## N

**NodeJS:** *Node.js* è una runtime di *JavaScript<sub>G</sub>* *open-source<sub>G</sub>* multipiattaforma orientata agli eventi per l'esecuzione di codice *JavaScript<sub>G</sub>*.

- <https://nodejs.org/>

## O

**Open-Source:** Termine impiegato in ambito informatico per riferirsi ad un software in cui gli attori rendono pubblico il codice sorgente. Tutto questo viene regolamentato tramite l'applicazione delle licenze d'uso. I software open-source permettono ai programmatori distanti di coordinarsi e di lavorare tutti allo stesso progetto.

## R

**React:** Libreria *JavaScript<sub>G</sub>* creata da *Facebook* che permette lo sviluppo di interfacce web.

- <https://github.com/facebook/react>

**Regressione Esponenziale:** Variante della *Regressione Lineare<sub>G</sub>* in cui i dati forniti non permettendo la rappresentazione di una retta vengono trasformati con delle operazioni matematiche per essere disposti in modo da rappresentare una funzione di tipo esponenziale. La libreria esterna consigliata per l'addestramento tramite *Regressione Esponenziale* è:

- <https://github.com/Tom-Alexander/regression-js>

**Regressione Lineare:** Abbreviata con l'acronimo *RL* è una tecnica di previsione di valori numerici utilizzando il metodo dei "minimi quadrati". La *Regressione Lineare*, come dice il nome, immagina che la legge sottostante i dati osservati sia esprimibile con una retta. Ogni punto osservato viene posto in un sistema per determinare i coefficienti della retta. Poiché sarebbe un sistema sovrastimato appena si hanno più di due punti (e quindi non risolvibile), viene considerata la somma del quadrato di tutte le differenze tra i valori trovati e i valori stimati. Minimizzando questa somma si trova la retta migliore per approssimare i dati. La libreria esterna consigliata per l'addestramento tramite *RL* è:

- <https://github.com/Tom-Alexander/regression-js>

**Regressione Logaritmica:** Variante della *Regressione Lineare<sub>G</sub>* in cui i dati forniti non permettendo la rappresentazione di una retta vengono trasformati con delle operazioni matematiche per essere disposti in modo da rappresentare una funzione di tipo logaritmica. La libreria esterna consigliata per l'addestramento tramite *Regressione Logaritmica* è:

- <https://github.com/Tom-Alexander/regression-js>

**Repository:** Abbreviato anche con "repo", è un ambiente di un sistema informativo in cui vengono conservati e gestiti file, documenti e metadati relativi ad un'attività di progetto.

## S

**SonarCloud:** Servizio cloud di analisi della qualità del codice, gratuito per repository  $G$  open-source. Sviluppato dalla stessa azienda di *SonarQube*, offre la possibilità di importare repository da servizi di hosting di codice come *GitHub*  $G$ .

- <https://sonarcloud.io/>

**Support Vector Machine:** Algoritmi impiegati per classificare allo scopo di risolvere la "maledizione della dimensionalità" che insorge quando si ha a che fare con grandi quantità di dati. In questo contesto viene infatti osservato un diradamento dei dati man mano che vengono aggiunte dimensioni attraverso la considerazione di più variabili.

Le *SVM* cercano l'iperpiano che divide meglio i dati osservati in due classi. In questo modo si riesce a resistere bene anche all'aggiunta di dimensioni ed al diradarsi dei punti nello spazio corrispondente. La libreria esterna consigliata per l'addestramento tramite *SVM* è:

- <https://github.com/karpathy/svmjs>

## T

**Telegraf:** È un server agent specializzato a ricevere ed inviare informazioni ed eventi da/a database, sistemi, e sensori *IoT*.

- <https://github.com/influxdata/telegraf>

**Time Series Database:** Spesso usato come acronimo (*TSDB*), è un sistema software ottimizzato per immagazzinare e rendere fruibili serie storiche di oggetti, ossia oggetti che associano valori temporali a valori numerici svoltosi in quell'istante di tempo.

**Throughput:** capacità effettiva di un canale di telecomunicazione.

**TypeScript:** Linguaggio di programmazione libero ed *open-source*  $G$  sviluppato da *Microsoft*. Si tratta di un super set tipizzato di *JavaScript*  $G$ .

## U

**UML:** *United Modeling Language*. Linguaggio di modellazione e specifica basato sul paradigma orientato agli oggetti, contraddistinto da una notazione di tipo visuale, semi-grafica e semi-formale. Fornisce una serie di diagrammi, elementi testuali formali ed elementi di testo libero, che compongono un linguaggio unico, allo scopo di semplificare la progettazione e la programmazione orientata agli oggetti.

**UNIX:** È un sistema operativo portabile per computer inizialmente sviluppato da un gruppo di ricerca dei laboratori *AT&T* e *Bell Laboratories*, nel quale figurarono sulle prime anche Ken Thompson e Dennis Ritchie. Storicamente è stato il sistema operativo maggiormente utilizzato su sistemi mainframe a partire dagli anni settanta.

## Y

**Yarn:** Package manager per *NodeJS*, compatibile con i package registry *npm*.

- <https://github.com/yarnpkg/yarn>