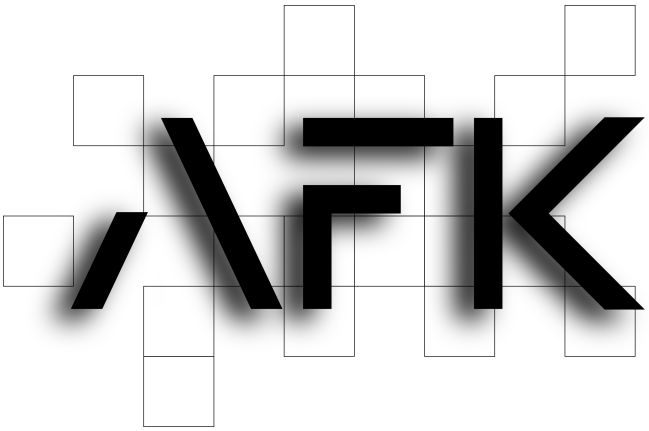

Developer Manual

TeamAFK - Project Predire in Grafana

gruppoafk15@gmail.com

Informations about the document

Version	1.0.0
Approval	
Drafting	Simone Federico Bergamin Olivier Utshudi
Check	
Use	External
Addressed to	Prof. Vardanega Tullio Prof. Cardin Riccardo TeamAFK

Description

Developer manual made by *TeamAFK* for the project *Predire in Grafana*.

Changelog

Version	Date	Description	Name	Role
1.0.0	2020-06-10	Document approved for RQ	Olivier Utshudi	*Manager*
0.4.0	2020-06-04	Writed and checked section §5	Olivier Utshudi	*Editor* *Verifier*
0.3.0	2020-06-04	Writed and checked section §4	Simone Federico Bergamin	*Editor* *Verifier*
0.2.0	2020-06-03	Writed and checked sections §2-§3	Olivier Utshudi	*Editor* *Verifier*
0.1.0	2020-06-03	Writed and checked section §1	Simone Federico Bergamin	*Editor* *Verifier*

Contents

List of Figures

List of Tables

1 Introduction

1.1 Document's purpose

Il manuale dello sviluppatore permette ad ogni sviluppatore che si approcci al prodotto software *"Predire in Grafana"* di assimilare le informazioni principali per poter manutenere e estendere tale prodotto. All'interno del documento verrà descritto il prodotto in modo dettagliato, consentendo allo sviluppatore di ottenere una spiegazione esaustiva necessaria per l'attività che andrà a svolgere.

The developer's manual allows each developer reader to absorb *"Predire in Grafana"*'s product key information to maintain and extend the product itself. This document describes the product in its totality, giving the developer an exaustive explanation required for his tasks.

1.2 Predire in Grafana's Purpose

"Predire in Grafana" è un plugin realizzato per la piattaforma open source$_G$ Grafana che permette di calcolare delle previsioni su un flusso dati. Viene utilizzato un algoritmo addestrato dall'utente, la cui definizione è contenuta in un file in formato JSON$_G$, per poter effettuare le previsioni. Viene fornito un applicativo esterno per l'addestramento degli algoritmi di previsione, attualmente sono implementati gli algoritmi di Support Vector Machine$_G$ e Regressione Lineare$_G$. Nello specifico il plugin monitora i dati in ingresso da un certo flusso, come per esempio percentuali di utilizzo della memoria o temperatura del processore, e produce delle previsioni che vengono successivamente mostrate attraverso l'interfaccia graficaG di Grafana. Il plugin rimane in esecuzione e riceve continuamente informazioni in ingresso da un flusso di dati. In questo modo gli operatori potranno monitorare eventuali cambiamenti sul flusso dati grazie alla previsione in real time$_G$ ed intervenire, se necessario, sull'origine del problema.

"Predire in Grafana" is a plugin made for Grafana which is an open source$_G$ platform commonly used to analyze data series. The plugin allows users to predict datas on a stream data. *"Predire in Grafana"* plugin uses a JSON file which contains a trained algorithm definition to get predictions. Users can use an external training tool, which use Machine Learning$_G$, to get these JSON' files. At the moment only Support Vector Machine and Linear Regression algorithm are implemented. In more detail input datas, like cpu's usage and cpu's temperature, are constantly monitored to get predictions on the aspect you want to examine. Predictions are shown throght Grafana GUI and continue to be updated after being calculated from datas coming from a database. Thanks to this operators can monitor each process and intervene at the root of the problem whenever neccessary.

1.3 Glossary

A fine documento viene riportato nell'appendice un glossario che racchiude al suo interno ciascun termine che necessita di una spiegazione dettagliata per risultare più comprensibile al lettore. Tali termini vengono contrassegnati nel documento con una G a pedice.

At the end of the document in the appendix is available a glossary where explanations for new or ambiguous terms can be found. These are marked with a subscript G.

1.4 References

1.4.1 Installation

- https://nodejs.org/en/ (https://nodejs.org/it/);
- https://git-scm.com/;
- https://www.gitkraken.com/;
- https://grafana.com/get;
- https://www.jetbrains.com/idea/.

1.4.2 Technologies

- https://reactjs.org/docs/getting-started.html;
- https://grafana.com/docs/grafana/latest/developers/plugins/;
- https://www.jetbrains.com/help/idea/eslint.html.

1.4.3 Legal

- https://www.apache.org/licenses/LICENSE-2.0.

1.4.4 Informative

- https://en.wikipedia.org/wiki/Linear_regression;
- https://en.wikipedia.org/wiki/Support_vector_machine.

2 System Requirements

Here the requirements for use of the product are listed.

2.1 Minimum Hardware Requirements

Here the requirements for use of the product are listed.

- 2GB of RAM;
- 5GB of space on a drive;
- Dual core processor.

2.2 Compatible Operating Systems

The software was developed and tested on the following:

- Windows 10;
- MacOS 10.15;
- Ubuntu 18, 20.

2.3 Compatible Browsers

Predire in Grafana can be accessed through the following browsers:

- Google Chrome version 58 or newer;
- Mozilla Firefox version 54 or newer;
- Apple Safari version 10 or newer;
- Microsoft Edge version 14 or newer;
- Opera version 55 or newer.

3 Installation

3.1 Instruments installation

3.1.1 Node.js installation

The installation of the runtime JavaScript Node.js can be done by visiting Node.js page at https://nodejs.org/en/download/. In this site the developer can download the most suitable version of Node.js for his operative system. For Linux user is also possible to use the package manager provided by the OS_G and exclusively for **Ubuntu/Debian** developers can run in terinal this code:
`apt-get install nodejs`

3.1.2 Git installation

For the installation of the version control system, the developer needs to reach Git site's at https://git-scm.com/downloads. Inside the 'Download' section are available the links to download the compatible version with his operative system. Also Linux base system can install Git from their package manager or running from **Ubuntu/Debian** terminal this line:
`apt-get install git`

3.2 Grafana installation

Developers can install Grafana by reaching its download page at https://grafana.com/grafana/download. There they can download compatible version for MacOs, Windows and Linux base system. Furthermore, the most common Unix base systems can install Grafana running terminal lines showed in the same page.

3.2.1 WEB Grafana execution

Depending from which OS the developer is working on,the execution of WEB Grafana can be done by:

- **Windows**: opening "bin" folder in Grafana installation folder (the path should be like `C:/Program Files/grafana-6.7.3/bin/`), and double-clicking on "grafana-server.exe";
- **Linux and MacOS**: running on terminal the following line:
  `systemclt start grafana-server`

After that, the developer needs to reach the address http://localhost:3000/ through a browser. For the first access, the developer needs to fill the fields username with "admin" and password with "admin", and once he/she is in, he/she will need to register himself/herself into the system for next accesses.

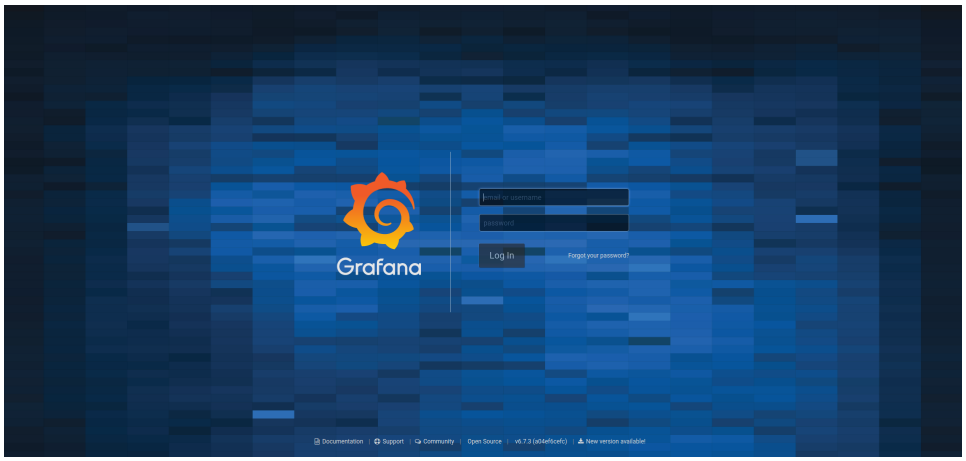

Figure 3.2.1: WEB Grafana page at http://localhost:3000

3.3 Plugin and Tool installation

The following sections will guide developers to install correctely both Training Tool and Prediction Tool.

3.3.1 GitHub repository clone

The developer to clone the GitHub repository needs to open the terminal and choose a folder inside the filesystem with command:
`cd /path/to/folder`
After that, in the same location, the developer needs to run this line:
`git clone https://github.com/teamafkSWE/PredireInGrafana-SW.git`
This line creates the folder that contains the source code of Training Tool and Prediction Plugin.

3.4 Dependencies

Dependencies are a list of packages needed to develop and run the project. For this reason, other developers will need to install all them to run correctely the tool and plugin. This operation can be done by moving to tool or plugin's folder and there run this line:
`npm install`
The following tables contain all the dependencies adopted to make both tool and plugin.

Table 3.4.1: Table of Training Tool dependency

Packacge	Version
@testing-library/react	9.5.0
@testing-library/user-event	7.2.1
bootstrap	4.4.1
chart.js	2.9.3

Table 3.4.1: (continua)

Package	Version
is-promise	2.2.2
libsvm-js	0.2.1
ml-svm	2.1.2
react	16.13.1
react-bootstrap	1.0.1
react-chartjs-2	2.9.0
react-csv-reader	3.5.0
react-dom	16.13.1
react-script	3.4.1
svm	0.1.1

Table 3.4.2: Table of Prediction Plugin dependency

Package	Version
@influxdata/influxdb-client	1.3.0
axios	0.19.2
react-datepicker	2.16.0
react-files	2.4.8

3.4.0.1 Developer dependency Developer will have to install dependencies to run correctely the project. All them are reported down below.

Table 3.4.3: Table of Prediction Tool developer dependency

Package	Version
@testing-library/jest-dom	4.2.4
csv-parser	2.3.2
eslint-plugin-react-hooks	4.0.4

Table 3.4.4: Table of Prediction Plugin developer dependency

Package	Version
@grafana/data	next
@grafana/toolkit	6.7.2
@grafana/ui	next
webpack	4.43.0

3.5 IDE Intellij IDEA installation

To install the integrated developing environment (IDE), it's necessary to reach the page https://www.jetbrains.com/idea/download/. This will address directely inside the section where Intellij IDEA can be downloaded for Windows, MacOS and Linux version. Ubuntu users can run one of the following lines to install the snap package:

- **Community version**:
  `sudo snap install intellij-idea-community -classic`
- **Ultimate version**:
  `sudo snap install intellij-idea-ultimate -classic`
- **Educatoinal version**:
  `sudo snap install intellij-idea-educational -classic`

4 System enviroment configuration

4.1 IntelliJ IDEA integrated development environment configuration

Per una corretta configurazione di IntelliJ IDEA è necessario configurare i path di sistema$_G$ e importare un progetto esistente.

A correct IntelliJ IDEA configuration needs to configure the path system$_G$ and after that to import an existing project.

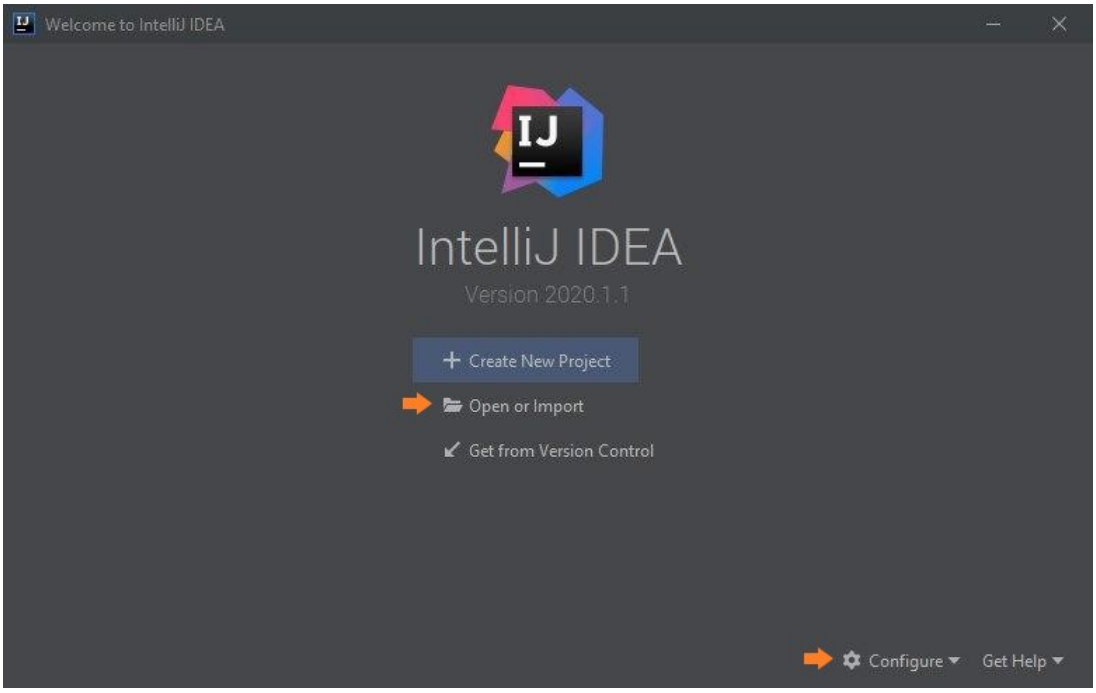

Figure 4.1.1: IntelliJ IDEA first execution

4.1.1 Path system configuration

Una volta avviato IntelliJ IDEA spostarsi su "Configure" | "Settings" in basso a destra. Dopo aver inserito "Node.js and NPM" nella barra di ricerca verificare che i campi "Node interpreter" e "Package manager" siano impostati correttamente, altrimenti aggiornare i percorsi selezionando la cartella corretta.

Once you run IntelliJ IDEA move to "Configure" then "Settings"in the lower-right corner. Write "Node.js and NPM" in the search bar and check for the correct settings of fields "Node interpreter" and "Package manager", oherwise update them with the correct paths.

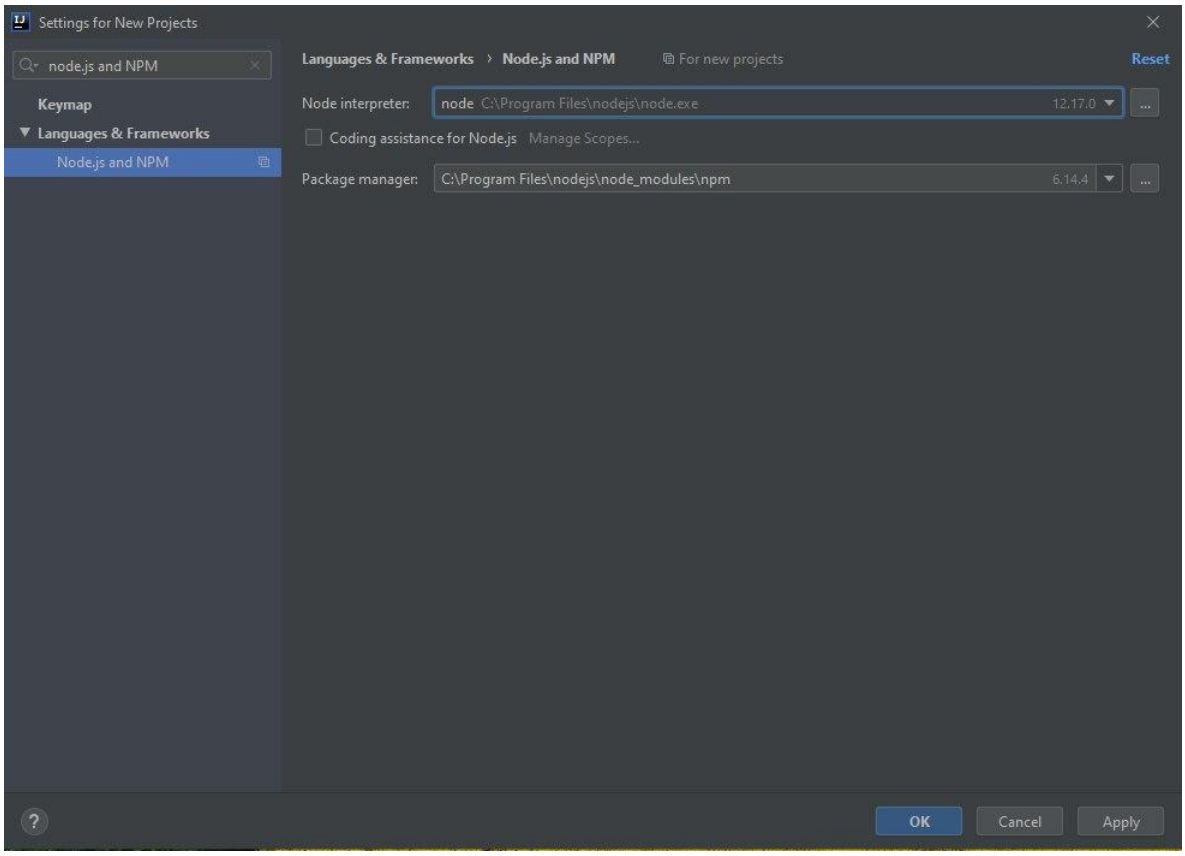

Figure 4.1.2: Node.j and NPM settings

4.2 Project import

Dalla schermata principale cliccare la voce "Open or Import" e selezionare la root directory del repository contenente il nostro progetto.

From IntelliJ IDEA main window click on "Open or Import" and select the root directory of our project repository folder.

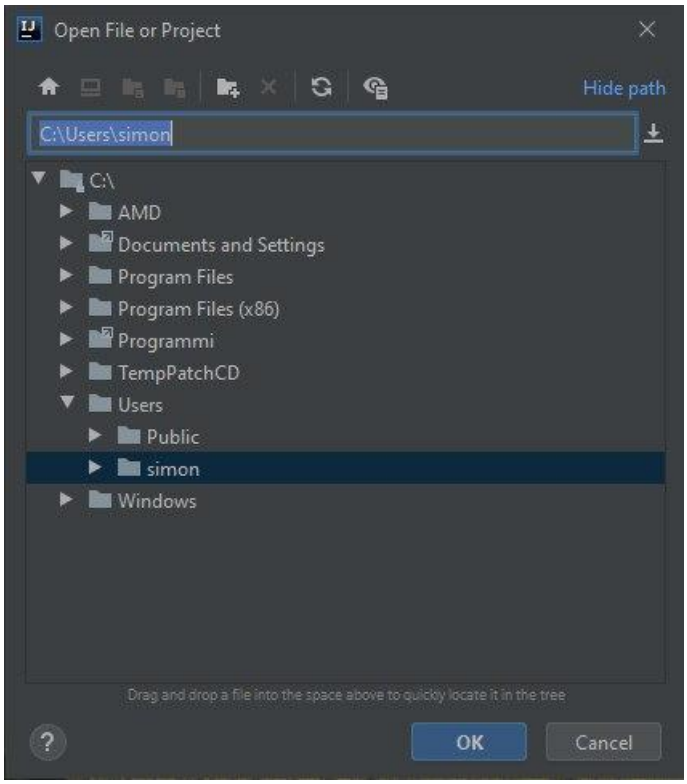

Figure 4.2.1: Project path on opening

4.3 ESLint configuration

4.3.1 Automatic configuration

Se ESLint è presente tra le dipendenze del progetto, viene configurato in automatico da IntelliJ IDEA. Verificare che sia attivo andando su "File" | "Settings" | "Languages and Frameworks" | "JavaScript" | "Code Quality Tools" | "ESLint" e controllare che sia selezionata la checkbox di configurazione automatica.

When ESLint is listed as a dependency in our project IntelliJ IDEA automatically configures it. Move to "File" | "Settings" | "Languages and Frameworks" | "JavaScript" | "Code Quality Tools" | "ESLint" and check that Automatic ESLint configuration option is enabled.

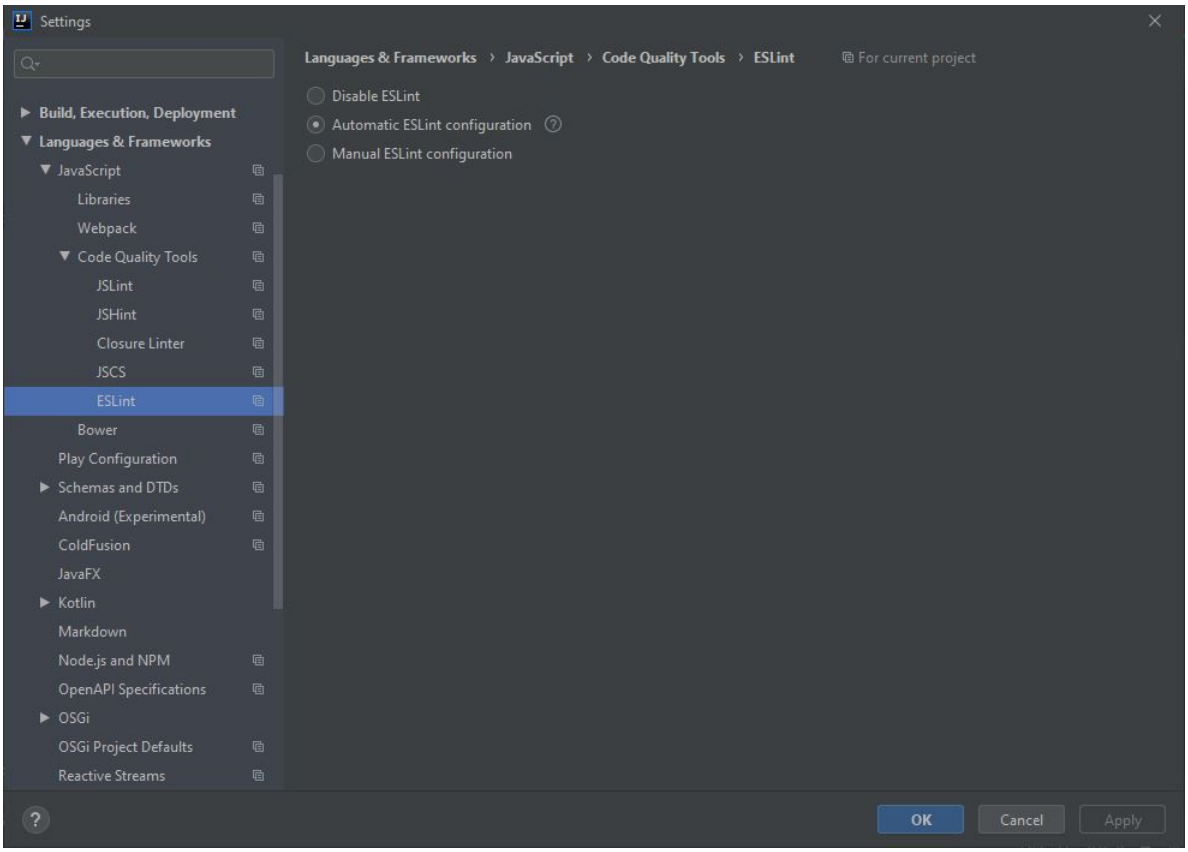

Figure 4.3.1: Automatic ESLint configuration

4.3.2 Manual configuration

Se lo sviluppatore lo desidera può configurare ESLint manualmente selezionando la checkbox di configurazione manuale e procedendo come segue:

1. inserire il path di Node.js nel campo "Node interpreter";
2. nel campo "ESLint package" specificare la posizione del package eslint o dello standard package;
3. scegliere il file di configurazione decidendo se affidare a IntelliJ IDEA la ricerca in modo automatico oppure selezionando un percorso a tale file.
4. opzionalmente compilare i campi "Additional Rules Directory" per specificare un ulteriore command-line per ESLint e "Extra ESLint Options" per specificare la posizione di file con ulteriori configurazioni e confermare.

You can also configure ESLint manually checking the Manual ESLint configuration option and complete fields as it follows:

1. in the "Node Interpreter" field, specify the path to Node.js;
2. in the "ESLint Package" field, specify the location of the eslint or standard package;

3. choose the configuration file to use checking "Automatic search" if you want to let IntelliJ IDEA do it for you or specify the file location in the path field;
4. optionally specify additional command-line options to run ESLint in "Extra ESLint Options" field and specify the location of the files with additional code verification rules in the "Additional Rules Directory" field then confirm the whole configuration.

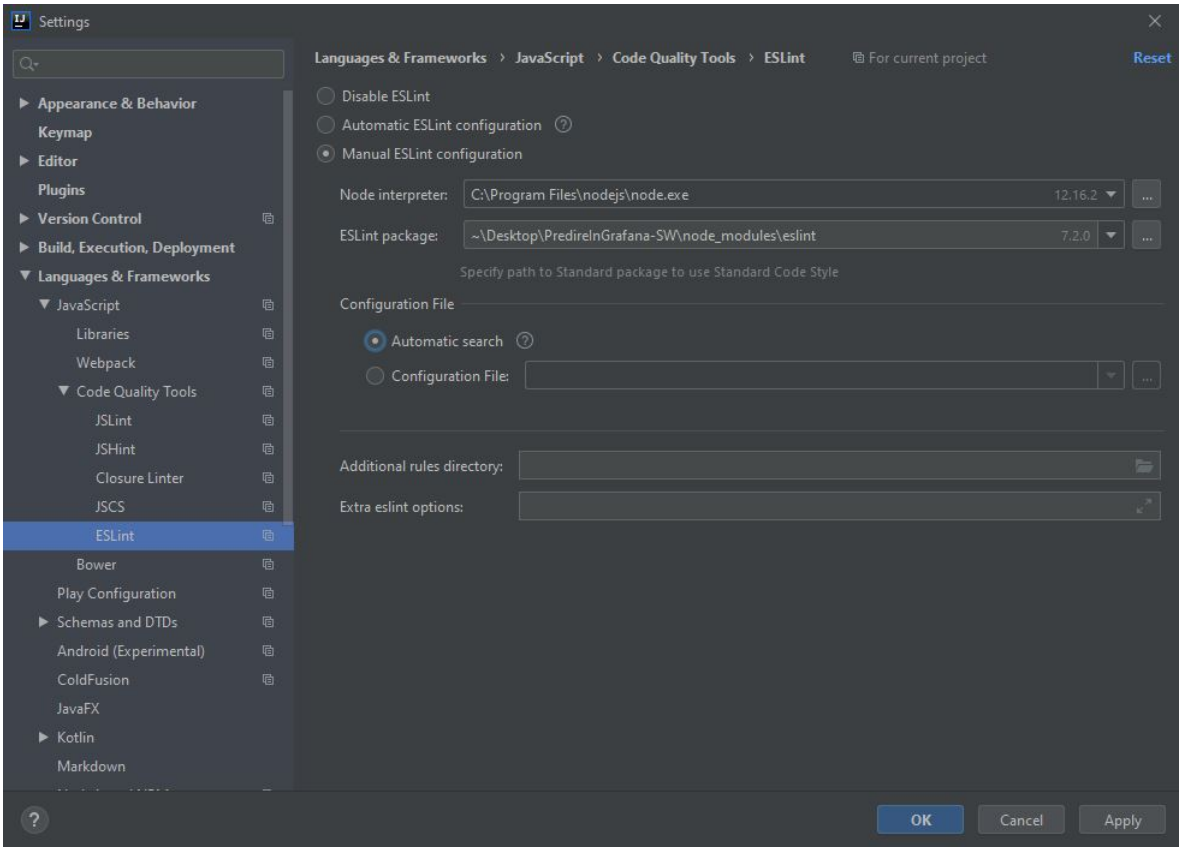

Figure 4.3.2: Manul ESLint configuration

4.4 Grafana plugin panel enviroment configuration

4.4.1 package.json content

Nel file package.json si trovano tutte le informazioni e i requisiti necessari al corretto funzionamento della nostra applicazione. Di seguito la lista degli attributi che rappresentano le informazioni più importanti:

- **scripts**: contiene una lista dei comandi usati dallo sviluppatore eseguibili da linea di comando:
  - build:
  - test:
  - dev:
  - watch:

- **dependencies**: contiene i seguenti pacchetti necessari al corretto funzionamento della nostra applicazione:
  - @influxdata/influxdb-client: il client javascript di riferimento per InfluxDB. Sono supportati gli ambienti node e browser;
  - axios: client HTTP per browser e Node.js;
  - react-files: componente React per la gestione di file di input (dropzone) usato nel caricamento dei file JSON all'interno del plugin.
- **devDependencies**: contiene i seguenti pacchetti necessari al corretto funzionamento della nostra applicazione durante lo sviluppo:
  - @grafana/data: libreria che contiene la maggior parte delle funzionalità di base e i tipi di dato utilizzati dalla piattaforma Grafana;
  - @grafana/toolkit: CLI che semplifica e aumenta l'efficienza dello sviluppo di un plugin in Grafana;
  - @grafana/ui: libreria che contiene i differenti componenti di design della piattaforma Grafana;
  - webpack: usato per compilare i moduli JavaScrip. Con webpack lo sviluppatore può interfacciarsi sia da riga di comando che da API.

In package.json file you can find all the app informations and requirements needed for its proper functioning. Attributes which represent important information are listed below:

- **scripts**: it contains all CLI command lines used from the developer:
  - build:
  - test:
  - dev:
  - watch:
- **dependecies**: it contains the following packages needed for our app proper functioning:
  - @influxdata/influxdb-client: the reference javascript client for InfluxDB. Both node and browser environments are supported;
  - axios: promise based HTTP client for the browser and node.js;
  - react-files: a file input (dropzone) management component for React we use when loading JSON files.
- **devDependencies**: it contains the following packages needed for our app proper functioning during development:
  - @grafana/data: a library containing most of the core functionality and data types used in Grafana.
  - @grafana-toolkit: a CLI_G that enables efficient development of Grafana plugins.

- @grafana/ui: a library containing the different design components of the Grafana ecosystem;
- webpack: used to compile JavaScript modules. Once installed, you can interface with webpack either from its CLI or API.

5 Test

A unity test is the operation that aims to check the correct execution of functions of a component. The run tests in this projetct, the developer needs to open the terminal and to move inside the folder of the product that needs to be tested (in this case `PredireInGrafanaSW/my-plugin` or `PredireInGrafanaSW/prediction-tool`). Inside the folder, the developer has to run the command `npm run test` inside the terminal.

5.1 Edit tests

Inside the project every components need to be tested. For this reason the developer has to name a test with **<component name>.test.ts**. Doing so a single component can be checked by a single file.

5.2 Code Coverage

Code coverage is the percentuage of the project's code that is analised by tests. This operation is made by Coverall which is an external tool linked to the code through the project's repository. So if the developer wants to chek coverage's value, he has to reach the READ.me file inside the GitHub repository at https://github.com/teamafkSWE/PredireInGrafana-SW. Inside that file there are two badges and the one with the label "coverage" shows the value of code coverage until the last build.

6 Product architecture

Our product consists of a plug-in developed for the Grafana platform and a Training Tool external to this platform. Therefore the analysis of the architecture is divided into these two components.

6.1 Training tool

The Training tool deals with training an SVM or LR algorithm using a dataset inserted by the user, to then generate a JSON file containing the necessary information to perform the prediction. This module has been developed following the *MVVM* behavioral design pattern.

6.1.1 Architectural design

We decided to implement this pattern because React$_G$ was used to build the component and we believed that this pattern coupled well with the structure of the latter. Moreover, it allows to divide the *Presentation Logic*$_G$ and the *Business Logic*$_G$ and it allows to reuse some components in other contexts, without having to change them.

As can be seen in the following figure, we have the View that exchanges information on user interactions with the ViewModel, which in turn transforms them into actions on the data performed by the Model. The data transition from the View to the Model occurs through the modification of a `props` data field entered by the ViewModel. Through these props, the ViewModel calls the correct functions when the user interacts with the View. The division between Business Logic and Presentation Logic is reinforced by this use of props. The Model provides functionality for the management of the algorithms through the `SVMtrain` and `RLtrain` classes that will be used by the ViewModel. Finally there is a communication between Model and View for the constant updating of the latter, thanks to a functionality provided by the Grafana platform.

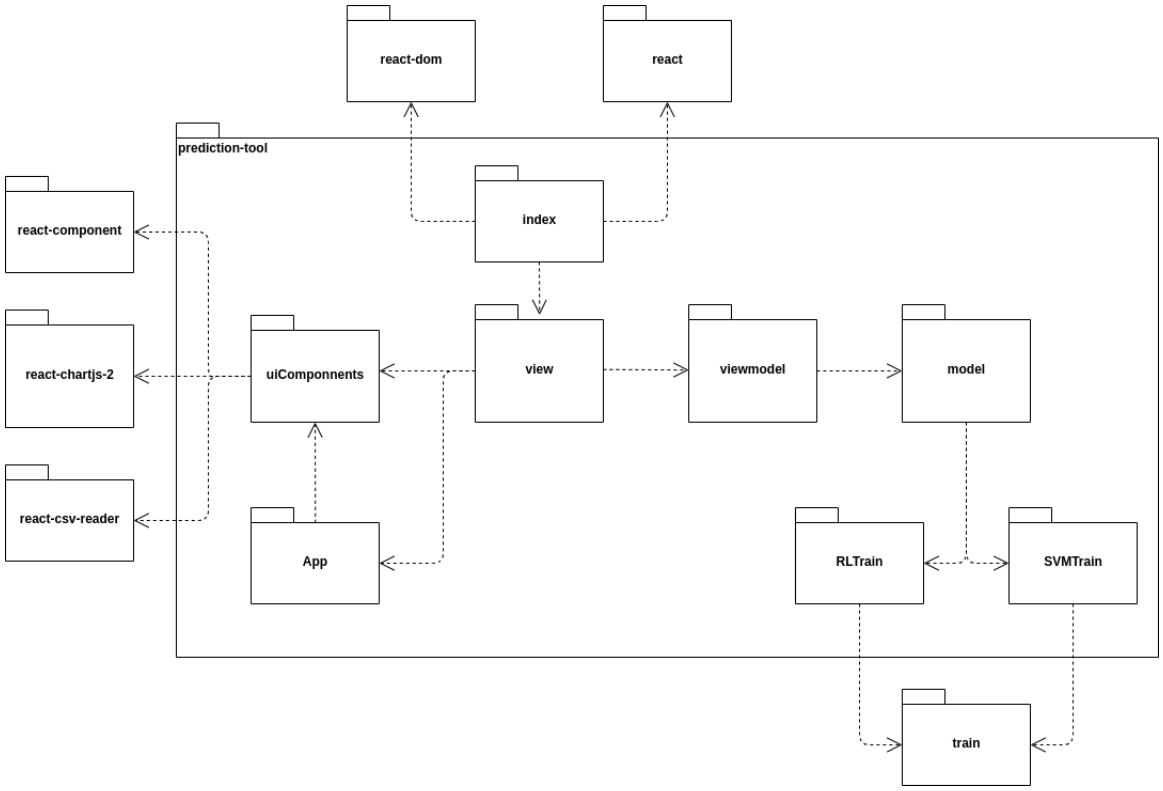

Figure 6.1.1: Training tool package diagram

Analyzing the specific components, our architecture is structured as follows:

- **Model**: it manages the *business logic*. It contains the data prediction algorithms that have been presently implemented and writing the result of the predictions to an Influx database;
- **View**: it manages the *presentation logic*. It allows the creation of a customized graphic panel within a Grafana dashboard. With this panel the user can select prediction algorithm settings, incoming data streams and the minimum and maximum thresholds;
- **ViewModel**: it manages the *application logic*$_G$. The ViewModel is an abstraction of the view exposing public properties and commands. MVVM has a binder, which automates communication between the View and its bound properties in the ViewModel.

6.1.1.1 Grafana's role

Grafana plays a fundamental role in the architecture as it allows continuous updating of the View to change the data in the Model. In particular, whenever there are updates on the data in the Model, Grafana makes them available to View through specific methods, together with the configuration of the queries with which this data was obtained. Moreover, through React, it manages the two-way data binding between HTML View and Javascript code, in addition to offering most of the necessary graphic components for the realization of the plug-in View.

6.1.2 Detail design

6.1.2.1 Model

The main component of the Model is the abstract class of the prediction algorithms called Train. Starting with this, we have implemented two algorithms: Support Vector Machine and Linear Regression. They are represented respectively by the concrete classes `SupportSVM` and `SupportRL`. We have found that, for families algorithms such as Support Vector Machine and Linear Regression algorithms, it is possible to trace them to a single abstract class as we have provided.

Train

The abstract class of the prediction algorithms is called `Train` and it represents the contract that all concrete classes must abide to be able to characterize a prediction algorithm. It contains only one parameter: `algorithm`. The `train()` method allows you to perform the prediction on a specific dataset. The `getJSON()` method writes the configuration parameters into the JSON file.

SupportSVM

To implement the Support Vector Machine algorithm we developed the concrete class `SupportSVM`. This class performs the prediction on the dataset. It also makes use of components specific to the SVM class imported from the SVM library to perform the real prediction. SupportSVM also contains two fields data:

- dataSVM: reference to the dataset;
- weights: the weights are the SVM coefficients that will be used to make the prediction.

Finally, there is the constructor that allows you to run the method binding to the class.

SupportRL

To implement the Linear Regression algorithm we developed the concrete class `SupportRL`. This class performs the prediction on the dataset. It also makes use of components specific to the RL class imported from the RL library to perform the real prediction. SupportRL also contains two fields data:

- dataRL: reference to the dataset;
- numOfX: the numbers of X founded on the CSV file;
- coefficients: the coefficients needed to make the prediction.

Finally, there is the constructor that allows you to run the method binding to the class.

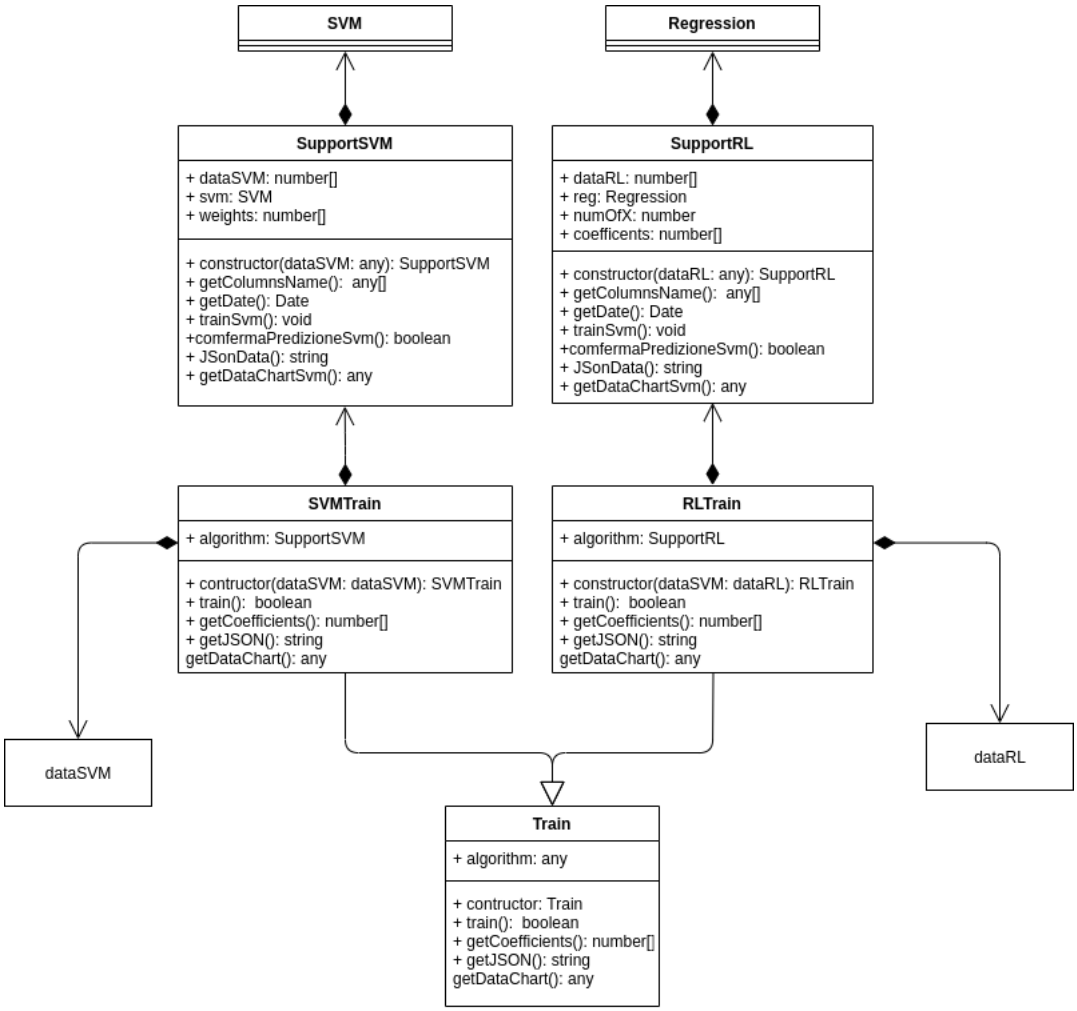

Figure 6.1.2: Training tool Model class diagram

6.1.2.2 View

Inside the View there are all the view components. The `App` class is the main component and it represents the tool's entry point. Each part of the View has been divided into components and rendered by the App. The communication between these components and the App takes place through the `props` (conceptually, the components are like JavaScript functions): it accepts arbitrary data (under the name of "props") and returns React elements that describe what should appear on the screen. Inside there is an instance of the ViewModel for data-binding.

App

The `App` class is the main component and it represents the tool's entry point. Each part of the View has been divided into components and rendered by the App. This class has just one parameter: `viewModel` in an instance of the ViewModel. The constructor contains the properties in common with the ViewModel. Moreover, it has the following methods:

- `changeAlgorithm(event)`: it manages the algorithm user's choice, setting the `algorithm` state value after the event was made;
- `resetAlgorithm(algorithm)`: it sets the `algorithm` state parameter to the default value;
- `setDataFromFile(data, fileInfo)`: it sets the `data` state parameter and the `fileName` state parameter with the inserted file's name;
- `handleTraining()`: it sends the data to the ViewModel and if the `performTraining()` method success, it calls the function to write the information into the JSON file, otherwise it resets the algorithm and sets the `jsonData` to null;
- `downloadJsonData()`: it creates the DownloadJson component;
- `render()`: it creates the InsertCsvButton, ComboBoxAlgorithm, TrainButton and Chart components and it calls the `downloadJsonData()` function.

combo_box_algorithm

It renders the combo box for the algorithm's choice.
The `render()` method renders the JSX_G element wanted.

download_json

It manages and render the "Download" button.
The `downloadJsonFile()` method allow to download the file, making the connection between the browser and the local pc. The `render()` method renders the "Download" button.

chart

It renders the chart. This component contains the following parameters:

- data: contains the data that must be shown;
- options: contains the various option that can be apply to the graph, such as colors, axes and the regression line.

It also implements the following methods:

- `RLChart`: it contains the methods to print the graphic of a rl-trained file on the screen;
- `SVMChart()`: it contains the methods to print the graphic of a svm-trained file on the screen;
- `formatData()`: it associates and manages the data to the graph;
- `render()`: it renders the chart graph.

insert_csv_button

It renders the insert button for the CSV file.
The `render()` method renders the "Inserisci file" button.

header

It renders the header.
The `render()` method renders the site's header.

train_button

It renders the train button, that will start the training.
The `render()` method renders the "Conferma" button.

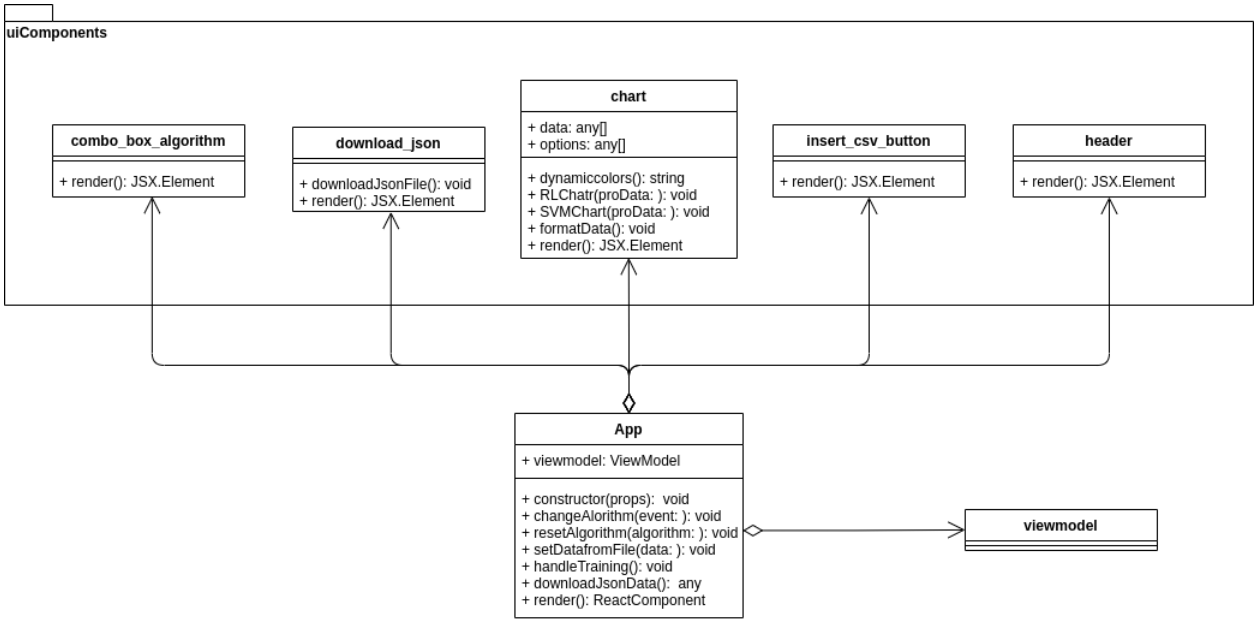

Figure 6.1.3: Training tool Model class diagram