

CENTRE DE FORMATION PROFESSIONNELLE TECHNIQUE

Soccer Pronostic

Rapport de stage

David Paulino (DP)

14 avril 2021

VERSION 1.0

Table des matières

| | | |
|----------|---|-----------|
| 1 | Introduction | 2 |
| 1.1 | Stage | 2 |
| 1.2 | Proof Of Concept | 2 |
| 1.3 | Environnement | 2 |
| 1.4 | Organisation | 3 |
| 1.5 | Livrable | 3 |
| 2 | Développement | 3 |
| 2.1 | Description de l'architecture | 3 |
| 2.1.1 | Architecture du projet | 3 |
| 2.1.2 | Base de données | 4 |
| 2.2 | Description des activités | 4 |
| 2.2.1 | Façade pour la communication avec l'API | 4 |
| 2.2.2 | Dotenv | 7 |
| 2.2.3 | Logging | 7 |
| 2.2.4 | Base de données | 7 |
| 2.2.5 | PredictionProvider | 8 |
| 3 | Bibliographie | 11 |
| 3.1 | Table des illustrations | 11 |
| 3.2 | Références | 11 |

Historique

| Version | Date | Auteur(s) | Modifications |
|---------|---------------|-----------|------------------|
| 1.0 | 13 avril 2021 | DP | Version initiale |

1 Introduction

Ce rapport a comme objectif de détailler toutes les étapes lors de mon stage effectué au CFPT. Comme les contrats de stages de techniciens ont été annulés par l'école suite à la pandémie du COVID-19, il nous a été demandé de faire un équivalent de stage pour pouvoir obtenir notre diplôme de technicien. Le travail du stage a pour but de produire un POC ("proof of concept") du travail de diplôme pour démontrer la faisabilité du projet et pour nous permettre d'avancer sur le projet qui nous permettra d'obtenir le titre de technicien ES.

1.1 Stage

Durant 6 jour de travail, soit 48 heures, l'objectif de mon stage est de réaliser un POC pour mon travail de diplôme. L'objectif de ce dernier est de réaliser des prédictions sur un match entre deux équipes de football sur une base de statistiques des matchs récents de ces deux équipes. Cela permet ensuite à des pronostiqueurs de se baser sur ce travail pour avoir une source supplémentaire pour choisir sur quelle équipe miser lors d'un paris sportif.

1.2 Proof Of Concept

Le POC a pour but de concevoir la classe qui me permet de récupérer les données de l'API que j'utilise pour mon travail de diplôme. J'y ajouterai évidemment de nouvelle chose à faire au fil des jours de stage, comme l'élaboration de l'architecture de l'application ou encore le logging des appels à l'API.

1.3 Environnement

- Un PC standard école avec Windows 10, 2 écrans
- Visual Studio Code
- Outil de versionnage de code (Git, avec dépôt distant sur Github / Bitbucket / GitLab)

- Navigateur web (Mozilla Firefox / Google Chrome)
- Outil bureautique à choix pour les documents
- Accès à une source de données pour avoir des statistiques sur des matchs de football¹

1.4 Organisation

Étudiant :

- David Paulino, **david.plnmr@eduge.ch**

Tuteur de stage :

- Antoine Schmid, **antoine.schmid@edu.ge.ch**

1.5 Livrable

Pour la fin du stage, le 15 avril 2021 :

Pour le tuteur de stage :

- Rapport de stage au format PDF
- Journal de bord au format PDF
- L'accès au répertoire distant pour pouvoir cloner le projet
- Une explication des pré-requis, pour pouvoir potentiellement exécuter le travail effectué par l'étudiant, dans un README à la racine du répertoire distant

2 Développement

2.1 Description de l'architecture

2.1.1 Architecture du projet

Après discussion avec mon tuteur de stage au sujet de l'architecture du projet, il était initialement prévu de faire une arborescence pour chaque fonctionnalité (sql, api, prediction, chacun dans un dossier différents). Cependant, après avoir commencé l'implémentation du provider pour la classe `Prediction`, je suis arrivé à un point bloquant qui m'empêchait d'importer les modules que j'avais créé. (soucis de pathing)

J'en ai donc parlé avec mon tuteur de stage et je lui ai proposé de changer

1. <https://apifootball.com>

l'architecture des dossiers du projet. Nous sommes donc arrivé au résultat suivant (voir fig. 1).

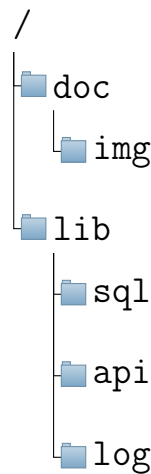


FIGURE 1 – Arborescence prévue pour le projet

2.1.2 Base de données

Au niveau du schéma de la base de données, j'avais initialement prévu de faire deux tables, une table qui contiendrait les matchs et une table qui contiendrait les prédictions en lien avec ces matchs. Cependant, comme il faut uniquement stocker les prédictions et pas forcément les matchs, après discussion avec M. Schmid, on en a conclu qu'une seule table suffisait. (voir fig. 2)²

2.2 Description des activités

2.2.1 Façade pour la communication avec l'API

Pour la communication avec l'API, il m'a été recommandé par mon tuteur de stage d'utiliser le design pattern **Façade**³.

Ce design pattern permet d'avoir une interface simple vers un système complexe. Par exemple, si nous utilisons une librairie pour faire du traitement d'image qui contient beaucoup de classes et que nous souhaitons convertir une image en SVG, nous allons créer une Façade qui va utiliser toutes les classes nécessaires pour faire cette conversion. (voir figure 3)

2. Le diagramme a été fait sur <https://dbdiagram.io/>

3. <https://refactoring.guru/fr/design-patterns/facade>

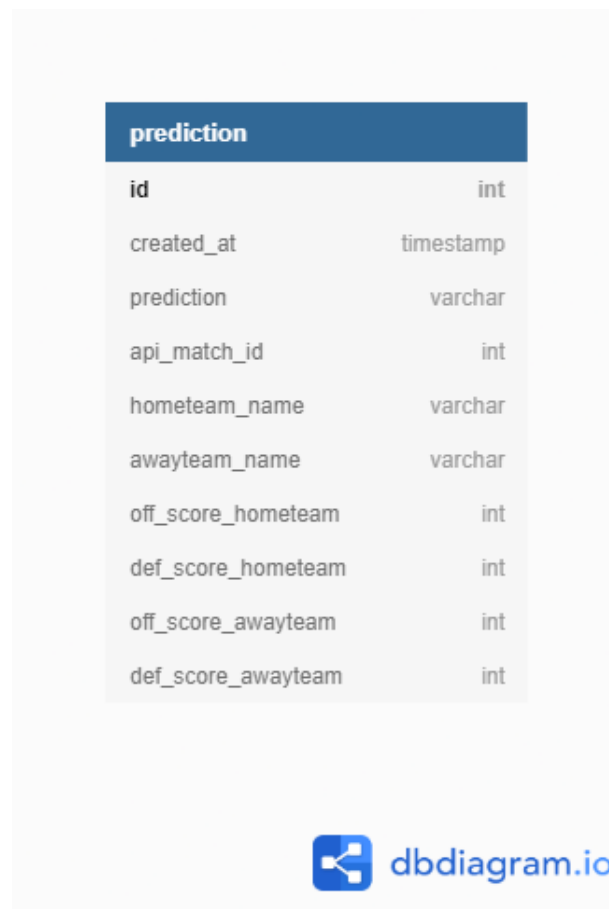


FIGURE 2 – MLD de la BDD

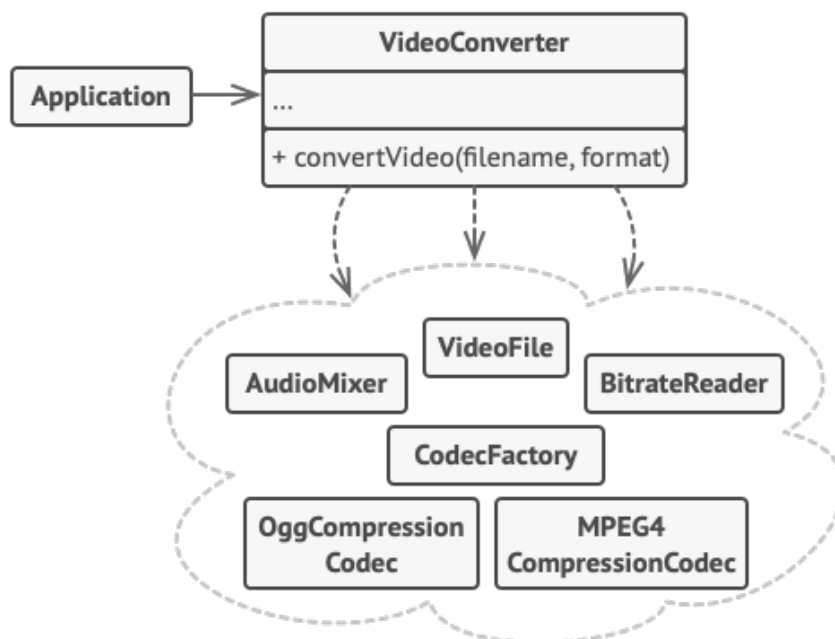


FIGURE 3 – Schéma pour montrer le fonctionnement de la façade

Dans notre cas, il permet de simplifier la communication entre l'API et l'application. Pour se faire, j'ai simplifier la manière de faire des requêtes à l'API en créant une méthode privée qui se nomme `getAction()`. Cette méthode est générale et elle prend en paramètres les paramètres nécessaire pour toutes les requêtes spécifiques faites à l'API.

Vous pouvez apercevoir sur le listing 1 que le code est très généraliste pour permettre d'être utilisé par n'importe quelle méthode. On peut aussi remarquer qu'à la ligne 5 du listing 1, on doit décoder le contenu récupéré de la requête car c'est un texte encodé en byte. (`b''`).

Le listing 2 quant à lui utilise `getAction()` en passant en paramètre les données spécifiques à `getCountries()`.

```
1 def __getAction(self, request_params):
2     """
3     Private method to call any endpoint of the API
4     """
5     response = requests.get(f'https://apiv2.apifootball.com/?
6                             APIkey={self.api_key}&{request_params}')
7     if response.status_code == 200: # Code 200 = OK. Healthy
8         connection
9         obj = json.loads(response.content.decode('utf-8'))
10        # Check if obj contains error so there is problem in the
11        API side
12        if 'error' in obj:
13            logging.error(f"Error {obj['error']} returned from the
14            API with message : {obj['message']}")
15            raise Exception(f"Error with the API : {obj['message']
16                            '}]")
17        else: # No error from the API we can return the result
18            logging.info(f"Request to the API with the params: {
19            request_params}")
20            return obj # decode to get the content in string
21        else:
22            # Error from the library
23            logging.error("Happened during the request to the API")
24            raise Exception("Could not connect to the API.")
```

Listing 1 – `getAction.py`

```
1 def getCountries(self):
2     """
3     Get countries available in the API
4     """
5     endpoint_action = "get_countries"
6     return self.__getAction(f'action={endpoint_action}')
```

Listing 2 – `getCountries.py`

Au niveau de l'API, j'ai découvert que lorsque l'on donne une mauvaise clé d'API, la librairie `requests` reçoit un code **200** mais l'API a envoyé un code erreur **404** avec comme message "Authentication failed!". De plus, le type de l'erreur est un dictionnaire. Alors qu'une réponse correcte

de l'API est un tableau.

Pour gérer ce "soucis", j'ai, après avoir reçu le code 200, vérifié si dans l'objet que me retourne l'API, l'élément `error` existe. (voir fig. 1)

2.2.2 Dotenv

La librairie Dotenv est très utile pour éviter de stocker des données sensibles, tels que des mots de passes de base de données ou des tokens d'API.

Je n'ai pas rencontré de problème dans l'ajout de la librairie au POC. J'ai donc créé un fichier nommé `.env` qui contient les variables d'environnement et lorsque j'ai besoin de ces variables-ci, je fais un `load_dotenv()` pour récupérer les variables stockés dans le fichier `.env`. Cela me permet de garder une sécurité sur ma clé d'API et sur les mots de passes utilisateurs de ma base de données.

2.2.3 Logging

Les logs sont une chose très important en programmation. Cela permet d'avoir une trace sur les actions faites et pour nous permettre de comprendre certaines erreurs qui peuvent apparaître, sans les afficher directement à l'utilisateur. Les logs sont stockés dans un fichier prévu à cet effet.

J'ai rencontré un soucis lors de l'implémentation des logs dans mon POC. En effet, les méthodes `info` et `debug` ne s'affichaient pas, mais les méthodes `warning` ou encore `error` s'affichaient. N'ayant pas trouvé de solution à ce soucis, j'en ai parlé brièvement à Monsieur Garcia qui m'a informé sur le fait que les librairies de logging utilise généralement un filtre sur l'affichage des logs. Après cette information, nous avons vérifié ensemble sur la documentation de la librairie et on pouvait spécifier le filtre que l'on souhaite appliquer dans la configuration de base des loggings. Cela m'a permis de régler ce point bloquant.

2.2.4 Base de données

Pour l'application, une base de données va nous permettre de pouvoir stocker les prédictions faites de manière hypothétiques pour avoir un historique sur ces dernières. La classe `DbManager` permet de gérer la base de données (`SELECT`, `INSERT`, `DELETE`). J'y ai ajouté des `SELECT` statements

pour récupérer une prédiction selon l'identifiant de l'API et selon les noms d'équipes.

Des logs sont évidemment écrit à chaque statement qui peut effectuer un changement sur la base de données.

```
1 def __query(self, your_query):
2     """
3     Make your own query in the specified table. Make sure to only
4     give a SELECT statement, otherwise your query won't work.
5     """
6     self.__cursor.execute(your_query)
7     return self.__cursor.fetchall()
```

Listing 3 – queryDb.py

Sur le listing 3, j'ai essayé de faire une méthode minimale qui sera appelé par tous les **SELECT** statements. Cette dernière est **private**.

```
1 def getAll(self):
2     """
3     Get all the rows in the table
4     """
5     return self.__query("SELECT * FROM prediction")
```

Listing 4 – getAll.py

Le code du listing 4 appelle uniquement le code du listing 3 avec la requête qu'il faut. Ensuite, il retourne le résultat de la requête sous forme de dictionnaire.

2.2.5 PredictionProvider

Évidemment, après avoir créé la Façade (point 2.2.1) et la classe qui me permet de gérer la base de données (point 2.2.4), je vais devoir les utiliser pour pouvoir élaborer des prédictions et les stocker dans la base de données. Cependant, les données transmises par l'API nécessite un traitement avant de pouvoir être utilisé correctement par ma classe **Prediction** (qui sera créé plus tard). C'est pourquoi, j'ai créé un **provider** qui va me permettre de faire des méthodes plus spécifiques pour la classe **Prediction** et de faire du sous-traitement au niveau des informations rendues par l'API. Ce provider sera aussi utilisé par le fichier principal de l'application pour pouvoir récupérer les prédictions stockées dans la base de données.

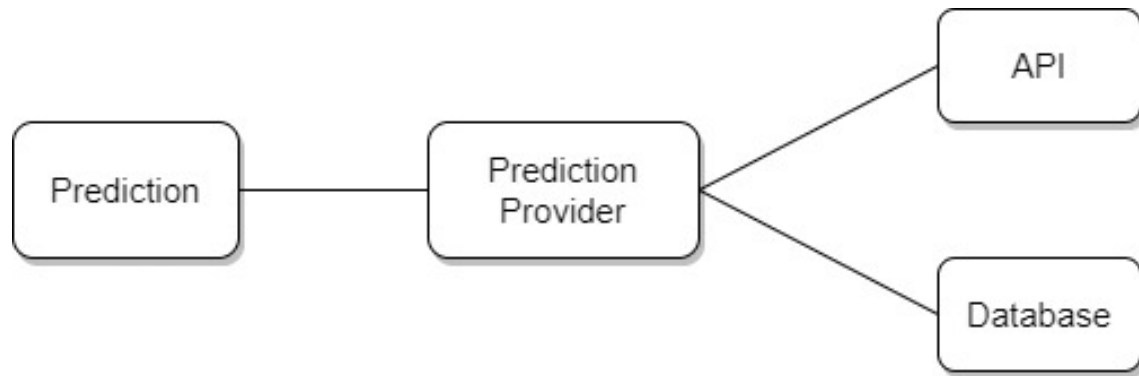


FIGURE 4 – Schéma du Provider pour les prédictions

Dans ce provider, j'ai développé la fonction `getAllStatsFromTeams()` qui a besoin de deux équipes en paramètres et qui récupèrent les statistiques de chaque match. En effet, car lorsque l'on utilise l'endpoint qui nous permet d'avoir les derniers résultats des matchs, nous n'avons pas accès aux statistiques de ces résultats. (voir listing 5)

J'ai découvert aussi lors de l'implémentation du provider que tous les matchs de l'API n'ont pas toutes les statistiques qui me sont nécessaires pour élaborer une prédiction. Par exemple, certains matchs n'ont pas la statistique **Dangerous Attacks**, j'ai donc créé une méthode pour être sûr que toutes les statistiques dont j'ai besoin sont disponibles pour chacun des matchs. Si un match n'a pas toutes les statistiques, nous ne le conserverons pas pour la prédiction. Cela peut tout de même être problématique si nous avons une quantité de matchs conséquentes que nous ne conservons pas. La liste des statistiques dont j'ai besoin est stockée dans le fichier `constants.py`

```
1  [
2    {
3      "firstTeam_VS_secondTeam": [
4        {
5          "match_id": "218349",
6          "country_id": "165",
7          "country_name": "Europe",
8          "league_id": "590",
9          "league_name": "Europa League",
10         "match_date": "2019-05-29",
11         "match_status": "Finished",
12         "match_time": "21:00",
13         "match_hometeam_id": "2616",
14         "match_hometeam_name": "Chelsea",
15         "match_hometeam_score": "4 ",
16         "match_awayteam_id": "2617",
17         "match_awayteam_name": "Arsenal",
18         "match_awayteam_score": " 1",
19         "match_hometeam_halftime_score": "0",
20         "match_awayteam_halftime_score": "0",
21         "match_live": "0"
22       },
23       .....
24     ],
25     "firstTeam_lastResults": [
26       {
27         "match_id": "218349",
28         "country_id": "165",
29         .....
30       },
31       .....
32     ],
33     "secondTeam_lastResults": [
34       {
35         "match_id": "218349",
36         "country_id": "165",
37         .....
38       },
39     ],
40     .....
41  ]
```

Listing 5 – Aperçu du JSON de l'endpoint H2H

3 Bibliographie

3.1 Table des illustrations

Table des figures

| | | |
|---|--|---|
| 1 | Arborescence prévue pour le projet | 4 |
| 2 | MLD de la BDD | 5 |
| 3 | Schéma pour montrer le fonctionnement de la façade | 5 |
| 4 | Schéma du Provider pour les prédictions | 9 |

Listings

| | | |
|---|--|----|
| 1 | getAction.py | 6 |
| 2 | getCountries.py | 6 |
| 3 | queryDb.py | 8 |
| 4 | getAll.py | 8 |
| 5 | Aperçu du JSON de l'endpoint H2H | 10 |

3.2 Références

- <https://refactoring.guru/fr/design-patterns/facade>
- <https://pypi.org/project/python-dotenv/>
- <https://docs.python.org/3/library/logging.html>
- <https://www.javatpoint.com/how-to-connect-database-in-python>
- <https://apifootball.com>