Engenharia e Dados e Conhecimento

Wiki Company - TP2

Final Report

December 20, 2018

By:
Davide Cruz 71776,
João Aniceto 72255 ,
Javier Borrallo Fernández 92092

Table of contents

Introduction

The objective of this project was to expose and manage data in a local triplestore GraphDB database, and also to extract data from a external database, in our case Wikidata[1].

For this purpose we chose a theme which we call Wiki Company. We created a simple wiki of publicly traded companies and CEOs of publicly traded companies.

By having specific and standardized data in our wiki we managed to make connections between different companies, different CEOs, and companies and their CEOs.

One of the main features of this project was being able to add informations from an outside source in a relatively easy manner.

FInally we allow the users to favourite companies and CEOs, giving them an easier access to those same entities through a personalized profile page.

Since the main focus of this project was to relate data using a triple store type database we found this theme an appropriate choice.

Data and Sources

The project contains 2 databases:

1. *SQLite* database which is used only for registering, authentication and user management purposes (deleting and granting/removing permissions). This is the Django default database. This way we don't have a major concern regarding the security of the authentication system (storing passwords properly) and we can use all the features it provides.
2. *Triplestore GraphDB* database used for almost everything else in the website, companies information, ceos information, coins information, and also all the user profile informations (name, favorite companies and favorite CEOs).

   The data that is currently stored by default is based in information from Wikidata.

Data format

Our data is stored in N3 format. We chose this format due to its simplicity and readability.

Our database is constituted by 6 different type nodes:

- Company

[1] https://www.wikidata.org/wiki/Wikidata:Main_Page

- CEO
- User
- Coin
- Revenue
- RevenueValues

❖ **Company, revenue and revenueValues nodes**

The company node contains all the company information, that information contains 13 predicates, 11 are literal values, one points to a ceo node, and another to the revenue node.

If there isn't a CEO node for the respective CEO the predicate will point to a literal value containing only for the CEO name. When a node for a CEO with that name is added to the database the company pointer will be updated.

The revenue node serves only as a intermediary between the company node and one or more revenueValues nodes. The revenueValues node contains only two predicates *Year* and *Value*.

```
company:tsla    pred:name "Tesla";
                pred:symbol "TSLA";
                pred:website "https://www.tesla.com/";
                pred:wikidataRef "Q478214";
                pred:revenue   [
                    pred:revenueValues  [
                                            pred:Year "2015";
                                            pred:Value "4046024000"
                                        ],
                                        [
                                            pred:Year "2014";
                                            pred:Value "3198356000 "
                                        ],
                                        [
                                            pred:Year "2016";
                                            pred:Value "7000132000"
                                        ],
                                        [
                                            pred:Year "2017";
                                            pred:Value "11758751000"
                                        ]
                ];
                pred:industry "automotive industry";
                pred:ceo ceo:elon_musk;
                pred:foundedBy "Elon Musk";
                pred:foundingYear "2003";
                pred:country "United States of America";
                pred:description "American automotive, energy storage and
solar power company";
                pred:logo "tsla.png";
                a "company".
```

- **CEO node**

The CEO node contains all the CEO information, that information contains at least 9 predicates and all of the associated objects are literal values.

A CEO can contain several nationalities triples and be the CEO of more than one company (although that is extremely rare).

If the CEO doesn't have a worth propriety in Wikidata we will add the literal 'Unknown'.

```
ceo:elon_musk   pred:name "Elon Musk";
                pred:photo "elon_musk.jpg";
                pred:description "South African-born American
entrepreneur";
                pred:sex "male";
                pred:nationality "South Africa";
                pred:nationality "Canada";
                pred:nationality "United States of America";
                pred:birth "28 June 1971";
                pred:worth "14200000000";
                pred:wikidataRef "Q317521";
                a "ceo".
```

- **Coin node**

The coin node contains only three predicates (*symbol, code, type*), all pointing to literal values.

Its value is extracted dynamically from an api[2] that returns its conversion rate in relation to US dollar.

```
coin:usd    pred:coinCode "usd";
            pred:coinSymbol "$";
```

[2] https://www.freeforexapi.com/api/live?pairs=EURUSD,USDEUR

```
             a "coin".
```

❖ **User node**

The user node contains the user information, that information contains at least 3 mandatory predicates, all of witch are literal values. The optional predicates are favCompany and favCeo and are pointers to companies and CEO nodes.

```
user:1  pred:idUser "1";
        pred:name "Antonio MoneyBags";
        pred:favCompany company:tsla;
        pred:favCompany company:amd;
        pred:favCeo ceo:elon_musk;
        a "user".
```

Operations over the data (SPARQL)

Throughout the project we used 17+1+2+11+6+7 queries, which of whom XX are SPARQL Update.

The '{}' string will appear in most queries, that string is replaced by necessary attributes or clauses before the query is made and if not obvious the context will be explained.

❖ **Query used in *home* view found in *views/bolsaViews.py* :**

In the home page we present the user two tables, onde containing the companies with the biggest revenue and the other with the smallest revenue, both in 2017. To extract this information we use a simple query that returns all the companies that have revenue in 2017.

We also show the ceo name in these tables, and because the ceo can be a node we had to use a optional triple.

```
PREFIX pred: <http://wikicompany.pt/pred/>
SELECT ?companyName ?value ?ceoName ?company ?ceo ?symbol
WHERE {
   ?company     pred:revenue         ?revnode.
   ?company     pred:symbol          ?symbol.
   ?revnode     pred:revenueValues   ?revSubNode.
   ?revSubNode pred:Year             "2017";
                pred:Value           ?value.
   ?company     pred:ceo             ?ceo.
   OPTIONAL{
   ?ceo         pred:name            ?ceoName.}
   ?company     pred:name            ?companyName.
}
```

❖ **Query used in *companies* view found in *views/bolsaViews.py* :**

In the /companies/ page we show a table with all the companies,their name, their symbol and their industry. If the user doesn't use any advanced search, by name or other parameter the query used is the following:

```
PREFIX pred: <http://wikicompany.pt/pred/>
SELECT ?company ?companyName ?companySymbol ?companyIndustry
WHERE {{
   ?company a "company".
   ?company pred:name ?companyName.
   ?company pred:symbol ?companySymbol.
   ?company pred:industry ?companyIndustry.
   {}
}}
```

However because we implement a search option, we have the necessity of adding triples to the query, depending of the number of parameters the user uses for their search the one or several options will be added to the previous query in place of the '{}' symbol.

For the name search we will filter the companies whose name or symbol contain the words placed in the search bar, ignoring their case:

```
filter (contains(lcase( ?companyName ),lcase("{arg}")) ||
contains(lcase(?companySymbol),lcase("{arg}"))).
```

If the user searches by industry the following lines will be added:

```
?company pred:industry ?companyIndustry.
filter contains(lcase(?companyIndustry),lcase("{}")).
```

Note also that we ignore the word industry in the industry form, due to bad categorizing in wikidata, however we do this at python level and not in the query.

To search by ceo we add the following lines (note the similar optional in /home/ query):

```
?company pred:ceo ?ceo.
optional{{
?ceo a "ceo".
?ceo pred:name ?ceoName.}}
filter( contains(lcase(?ceoName),lcase("{arg}")) ||
contains(lcase(?ceo),lcase("{arg}")) ).
```

If search by founder the following lines will be added:

```
?company pred:foundedBy ?founder.
filter contains(lcase(?founder),lcase("{}")).
```

If search by founding year the following lines will be added:

```
?company pred:foundingYear ?foundingYear.
filter (?foundingYear = "{}").
```

If searched by country the following lines will be added:

```
?company pred:country ?country.
filter contains(lcase(?country),lcase("{}")).
```

- **Query used in *companiesInfo* view found in *views/bolsaViews.py* :**

If we click on a company name or their symbol in the /companies/ page we will be redirected to their info page (we can also get here by going directly through the url, which has the following format “ <localhost:port>/companies/<companySymbol> “).

In this page we show all the company info in the our database, to be able to do this we do the following query (note: ‘{s}’ stands for company symbol):

```
PREFIX pred: <http://wikicompany.pt/pred/>
SELECT ?companyCeo ?companyName ?companySymbol ?companyIndustry
?companyWebsite ?companyWikiRef ?companyFounder ?companyFoundingYear
?companyCountry ?companyDescription ?companyLogo ?ceoName ?revenue
?revenueYear ?revenueValue
```

```
WHERE {{
   ?company a "company".
   ?company pred:symbol '{s}'.
   ?company pred:name ?companyName.
   ?company pred:industry ?companyIndustry.
   ?company pred:symbol ?companySymbol.
   ?company pred:website ?companyWebsite.
   ?company pred:wikidataRef ?companyWikiRef.
   ?company pred:foundedBy ?companyFounder.
   ?company pred:foundingYear ?companyFoundingYear.
   ?company pred:country ?companyCountry.
   ?company pred:description ?companyDescription.
   ?company pred:logo ?companyLogo.
   ?company pred:ceo ?companyCeo.
   OPTIONAL{{?companyCeo pred:name ?ceoName.}}
}}
```

A company can have several revenues, one for each year, all of hose can be seen in a graph in the company info page. We decided that a good way to extract this information would be through a separate query, the reason being that the revenues belong to another node.

```
PREFIX pred: <http://wikicompany.pt/pred/>
SELECT ?revenueYear ?revenueValue
WHERE {{
   ?company a "company".
   ?company pred:symbol '{s}'.
   ?company pred:revenue ?rev.
   ?rev pred:revenueValues ?revenueValues.
   ?revenueValues pred:Year ?revenueYear.
   ?revenueValues pred:Value ?revenueValue.

}}
```

❖ **Query used in *valuesHome* view found in *views/bolsaViews.py* :**

In the page /values/ we show a bar chart of all the companies revenues, a search option similar to the /companies/ was implemented, however the man query like it can be seen next is different.

In the query from the previous point (companiesInfo), we separated the query for company info and the query for revenues, in this however we took a different path and implemented both in the same query. One of the drawbacks of this implementation is that although we will have a unique result for each revenue the rest of the company information will be repeat, e.g, if a company has 5 revenues there will be 5 rows with mostly the same information. We corrected this results posteriorly in python.

```
PREFIX pred: <http://wikicompany.pt/pred/>
SELECT ?companyName ?revenueYear ?revenueValue
WHERE {{
   ?company a "company".
   ?company pred:name ?companyName.
   ?company pred:symbol ?companySymbol.
   {}
   ?company pred:revenue ?rev.
   ?rev pred:revenueValues ?revenue.
   ?revenue pred:Year ?revenueYear.
   ?revenue pred:Value ?revenueValue.
}}
```

Here is one option that can be added to the query in place of the '{}' symbols, as we can see this is identical to the industry search in /companies/ page and view, the other search parameters are also identical.

```
?company pred:industry ?companyIndustry.
filter contains(lcase(?companyIndustry),lcase(\"{}\")).
```

- **Query used in *ceos* view found in *views/bolsaViews.py* :**

In the /ceos/ page we show a table with all the CEOs, their name, their sex and their birth date. If the user doesn't use any advanced search, by name or other parameter the query used is the following:

```
PREFIX pred: <http://wikicompany.pt/pred/>
SELECT ?ceo ?ceoName ?ceoSex ?ceoBirth
WHERE {{
   ?ceo a "ceo".
   ?ceo pred:name  ?ceoName.
   ?ceo pred:sex   ?ceoSex.
   ?ceo pred:birth ?ceoBirth.
   {}
}}
```

However like in the /companies/ page and /values/ page we give the option to do a advanced search of all the CEOs, in this case aside for the name we only offer to extra parameters in advanced search: by sex or by nationality. Like in the other searches the user can use as many parameters as he/she wants ate the same time to do a search.

If we decide to filter using the name form the following line will be added to the query:

```
filter (contains(lcase(?ceoName),lcase("{arg}"))).
```

If we decide to filter using the sex form the following line will be added to the query:

```
?ceo pred:sex ?ceoSex.
filter contains(lcase(?ceoSex),lcase(\"{}\")).
```

If we decide to filter using the nationality form the following line will be added to the query:

```
?ceo pred:nationality ?ceoNacionality.
filter contains(lcase(?ceoNacionality),lcase(\"{}\")).
```

- **Query used in *ceosInfo* view found in *views/bolsaViews.py* :**

Like any company in our database the CEOs have their own page with their wikidata information(only the data we found relevant).

To show this information to the user we make the following query.

Note: a ceo can have several nationalities, and companies, and for that reason we find ourselves in a similar position if we look back on how we dealt with a company info and their revenue throughout the years. One difference is that the nationalities and companies that the CEO runs aren't on their own node.

Anyhow since there can be several triples with the same predicate we 'divided' the query in three phases and used a UNION to join the queries. This results in a 'weird' table with some rows being almost empty. To make the workable dictionaries out of the resulting info we made some operations in python.

```
PREFIX pred: <http://wikicompany.pt/pred/>
SELECT ?ceoName ?ceoPhoto ?ceoSex ?ceoBirth ?ceoWorth ?ceoDescription
?ceoNationality ?companyName ?companySymbol ?ceoWiki
WHERE {{
       {{
               ?ceo a "ceo".
               ?ceo pred:name '{x}'.
               ?ceo pred:name ?ceoName.
               ?ceo pred:photo ?ceoPhoto.
               ?ceo pred:sex ?ceoSex.
               ?ceo pred:birth ?ceoBirth.
               ?ceo pred:worth ?ceoWorth.
               ?ceo pred:description ?ceoDescription.
               ?ceo pred:wikidataRef ?ceoWiki.
```

```
            }}
        UNION
        {{
            ?ceo a "ceo".
                ?ceo pred:name '{x}'.
                 ?ceo pred:nationality ?ceoNationality.
            }}
        UNION{{
            ?ceo a "ceo".
                ?ceo pred:name '{x}'.
                ?company a "company".
                ?company pred:name ?companyName.
                ?company pred:symbol ?companySymbol.
                ?company pred:ceo ?ceo

            }}
    }}
```

❖ **Query used in *profile* view found in *views/bolsaViews.py* :**

If a user is registered and logged in he will have a profile page. This page will show tables with the users favorite companies and CEOs, as well as tables with suggestions to companies and CEOs for the user to follow. The user will also have the option to directly favorite those suggestions or delete their favorites directed on the page.

The information on this page will be the result of 5 different queries, all will use the user id given by django on the moment a user registers. That id will be shared through both databases.

The first query is very simple, we read from the database the users name:

```
PREFIX pred: <http://wikicompany.pt/pred/>
SELECT ?userName
WHERE {{
   ?user a "user".
   ?user pred:idUser "{}".
   ?user pred:name ?userName.
}}
```

In the second query we read, if existing, the information from the users favorite companies, and since only some companies have data for their 2017 revenue we encapsulate the query for that information in a OPTIONAL parameter. We only show up to 3 companies however that selection, in this case, is made in the front end.

```
PREFIX pred: <http://wikicompany.pt/pred/>
```

```
SELECT distinct ?companyName ?companySymbol ?revenueValue
WHERE {{
       ?user a "user".
       ?user pred:idUser "{}".
       ?user pred:favCompany ?company.
       ?company pred:name ?companyName.
       ?company pred:symbol ?companySymbol.
       ?company pred:revenue ?rev.
       optional{{
        ?rev pred:revenueValues ?revenueValues.
           ?revenueValues pred:Year '2017'.
           ?revenueValues pred:Value ?revenueValue.
       }}
   }}
```

In the third query we read, if existing, the information from the users favorite CEOs. Like before we only show up to 3 CEOs however that selection is also made in the front end.

```
PREFIX pred: <http://wikicompany.pt/pred/>
SELECT ?ceoName ?ceoWorth  ?ceoBirth
WHERE {{
       ?user a "user".
       ?user pred:idUser "{}".
       ?user pred:favCeo ?ceo.
       ?ceo  pred:name ?ceoName.
       ?ceo  pred:worth ?ceoWorth.
       ?ceo  pred:birth ?ceoBirth
}}
```

In the fourth query we read, if existing, CEOs suggestions based on favorite companies. Since we don't want to suggest CEOs that the user is already following we apply a MINUS parameter to remove that data. We also limit the showed results to 3 in this table however in this case and the next we decided to apply this limitation in the query itself (see limit 3).

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX user: <http://wikicompany.pt/user/>
select ?ceoName ?ceoWorth
where{{
   ?user pred:idUser "{}".
   ?user pred:favCompany ?company.
   ?company a "company".
   ?company pred:ceo ?ceo.
   ?ceo a "ceo".
   ?ceo pred:name ?ceoName.
   ?ceo pred:worth ?ceoWorth.
   minus{{
        ?user pred:favCeo ?ceo.
    }}
}}
limit 3
```

In the fifth query we read, if existing, companies suggestions based on favorite CEOs. Since we don't want to suggest companies that the user is already following we apply a MINUS parameter to remove that data. We also limit the showed results to 3.

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX user: <http://wikicompany.pt/user/>
select ?companyName ?companySymbol
where{{
   ?company a "company".
   ?company pred:name ?companyName.
   ?company pred:symbol ?companySymbol.
   ?company pred:ceo ?ceo.
   ?user pred:idUser "{}".
   ?user pred:favCeo ?ceo.
   ?ceo a "ceo".
   minus{{
   ?user pred:favCompany ?company.
   }}
}}
limit 3
```

- **Query used in *seeFull* view found in *views/bolsaViews.py* :**

In the /profile/ page we limit all the tables to 3 rows, that was a aesthetic decision, however the user can follow as many companies and CEOs as there are in the database, to see the information in its interelly we have several 'show more' buttons, those buttons redirect to a different page, but since the information is the 'same' that of the profile page we do the exact same queries only remove the limits in the frontend or in the query.

- **Query used in *adminAddCompany* view found in *views/adminViews.py* :**

To add a company to the database, after we query Wikidata for information (those queries will be explored in a later point in the report), we use two queries.

First we need to check if the companies ceo is a node in the database, if it isn't the predicate will point to a literal value, if it is the predicate will point to the CEO node.

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX user: <http://wikicompany.pt/user/>
SELECT ?ceo
WHERE {{
   ?ceo a "ceo".
   ?ceo pred:name "{}".
}}
```

Then we finally insert the queried data from Wikitada to our local database:

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX company: <http://wikicompany.pt/companies/>
DELETE {{
   company:{symbol} ?a ?b
}}
WHERE {{
   company:{symbol} ?a ?b
}};
INSERT DATA{{
   company:{symbol} pred:name "{name}".
   company:{symbol} pred:symbol "{symbolUpper}".
   company:{symbol} pred:website "{website}".
   company:{symbol} pred:wikidataRef "{wikidataRef}".
   company:{symbol} pred:revenue  [{revenue}].
   company:{symbol} pred:industry "{industry}".
   company:{symbol} pred:ceo {ceo}.
   company:{symbol} pred:foundedBy "{founder}".
   company:{symbol} pred:foundingYear "{fYear}".
   company:{symbol} pred:country "{country}".
   company:{symbol} pred:description "{description}".
   company:{symbol} pred:logo "{logo}".
   company:{symbol} a "company".
}}
```

❖ **Query used in *adminEditCompany* view found in *views/adminViews.py* :**

This view is used to edit the information in the edit page and also the add page, therefore the code in both is the same, since the previous query can also be used to edit information we reused it in this query.

❖ **Query used in *adminDeleteCompany* view found in *views/adminViews.py* :**

When we delete a company we have to delete not only that company from the database but also their connection to every user that favorited that company.

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX company: <http://wikicompany.pt/companies/>
DELETE {{
   ?user pred:favCompany ?company.
}}
WHERE {{
   ?company a "company".
   ?company pred:symbol "{code}".
   ?user pred:favCompany ?company.
}};
DELETE {{
   ?company ?a ?b.
}}
WHERE {{
   ?company a "company".
   ?company pred:symbol "{code}".
   ?company ?a ?b.
}};
```

- **Query used in *adminAddCeo* view found in *views/adminViews.py* :**

To add a CEO to the database, after we query Wikidata for information (those queries will be explored in a later point in the report), we use two queries.

First we add the ceo node in the database.

Note: {nat}n symbol represented the lines to add the nationality of the ceo, since the nr of nationalities changes from ceo to ceo we have to choose the number of lines to add.

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX ceo: <http://wikicompany.pt/ceo/>
DELETE {{
   ceo:{code} ?a ?b.
}}
WHERE {{
   ceo:{code} ?a ?b.
}};
INSERT DATA{{
   ceo:{code} pred:name "{name}".
   ceo:{code} pred:photo "{logo}".
   ceo:{code} pred:description "{description}".
   ceo:{code} pred:sex "{sex}".
   {nat}
   ceo:{code} pred:birth "{birth}".
   ceo:{code} pred:worth "{worth}".
   ceo:{code} pred:wikidataRef "{wiki}".
   ceo:{code} a "ceo".
}}
```

After adding the ceo and all their information we check if the CEO works for any company in our database, if it does we change the literal value in the company node that points to the ceo, to a pointer to the ceo node.

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX company: <http://wikicompany.pt/companies/>
INSERT{{
 ?company pred:ceo ?ceo.
}}
WHERE {{
   ?company a "company".
   ?company pred:ceo "{name}".
   ?ceo a "ceo".
   ?ceo pred:name "{name}".
}};
DELETE{{
 ?company pred:ceo "{name}".
}}
WHERE {{
   ?company a "company".}};
```

- **Query used in *adminEditCeo* view found in *views/adminViews.py* :**

Like in the case adminEditCompanies we reuse the queries in the add view, only in this case we reuse the adminAddCeo queries.

- **Query used in *adminDeleteCeo* view found in *views/adminViews.py* :**

When we delete a CEO we have to delete not only that CEO from the database but also their connection to every user that favorited that CEO.

When we delete a CEO also we have to delete the connection of that node to a company, if it exists, and change it to a literal value .

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX company: <http://wikicompany.pt/companies/>
INSERT{{
 ?company pred:ceo "{name}".
}}
WHERE {{
   ?company a "company".
   ?company pred:ceo ?ceo.
   ?ceo a "ceo".
   ?ceo pred:name "{name}"
}};
DELETE{{
 ?company pred:ceo ?ceo.
}}
WHERE {{
   ?company a "company".
   ?company pred:ceo ?ceo.
   ?ceo a "ceo".
   ?ceo pred:name "{name}".
}};
```

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX company: <http://wikicompany.pt/companies/>
DELETE {{
   ?user pred:favCeo ?ceo.
}}
WHERE {{
   ?ceo a "ceo".
   ?ceo pred:name "{}".
   ?user pred:favCeo ?ceo.
}};
DELETE {{
   ?ceo ?a ?b.
}}
WHERE {{
   ?ceo a "ceo".
   ?ceo pred:name "{}".
   ?ceo ?a ?b.
}};
```

❖ **Query used in *adminAddCoin* view found in *views/adminViews.py* :**

If the user happens to be a superuser he/she can add a coin to the database as long as that currency is in the wikidata database.

After making a query to wikidata to extract the code and symbol of a coin we insert a new coin in the database using the following query:

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX coin: <http://wikicompany.pt/coin/>
INSERT DATA{{
   coin:{code} pred:coinCode "{code}".
   coin:{code} pred:coinSymbol "{symbol}".
   coin:{code} a "coin".
}}
```

- **Query used in *adminDeleteCoin* view found in *views/adminViews.py* :**

If the user happens to be a superuser he/she can delete a coin from the database as long as that currency isn't the US dollar or the current coin selected.

We remove that data from the database through the following query:

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX coin: <http://wikicompany.pt/coin/>
DELETE {{
   coin:{code} ?s ?o.
}}
WHERE {{
   coin:{code} ?s ?o.
}}
```

- **Query used in *register* view found in *views/authenticationViews.py* :**

When the user registers we have to insert their name and id in the GraphDB database, we accomplish that with the following query:

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX user: <http://wikicompany.pt/user/>
INSERT DATA {{
   user:{id} a "user".
   user:{id} pred:idUser "{id}".
   user:{id} pred:name "{name}".
}}
```

- **Query used in *favoriteCompany* view found in *views/favoriteViews.py* :**

As mentioned before we give the option for a user to favorite a company, if he/she chooses to use this feature we have to make a connection from the user node to the company node. We do this with the following query:

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX user: <http://wikicompany.pt/user/>
INSERT {{
   ?user pred:favCompany ?company.
}}
WHERE {{
   ?user pred:idUser "{id}".
   ?company a "company".
   ?company pred:symbol "{symbol}".
}}
```

❖ **Query used in *unfavoriteCompany* view found in *views/favoriteViews.py* :**

If the user unfavorites a company we only need to remove the connection between the two nodes, of course we have to check if the company is a favorite of the user:

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX user: <http://wikicompany.pt/user/>
DELETE {{
   ?user pred:favCompany ?company.
}}
WHERE {{
   ?user pred:idUser "{id}".
   ?company a "company".
   ?company pred:symbol "{symbol}".
}}
```

❖ **Query used in *favoriteCeo* view found in *views/favoriteViews.py* :**

We give the option for a user to favorite a CEO, if he/she chooses to use this feature we have to make a connection from the user node to the CEO node. We do this with the following query:

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX user: <http://wikicompany.pt/user/>
INSERT {{
   ?user pred:favCeo ?ceo.
}}
WHERE {{
   ?user pred:idUser "{id}".
   ?ceo a "ceo".
   ?ceo pred:name "{name}".
}}
```

❖ **Query used in *unfavoriteCeo* view found in *views/favoriteViews.py* :**

If the user unfavorites a CEO we need to remove the connection between the two nodes, of course we have to check if the CEO is a favorite of the user:

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX user: <http://wikicompany.pt/user/>
DELETE {{
   ?user pred:favCeo ?ceo.
}}
WHERE {{
   ?user pred:idUser "{id}".
   ?ceo a "ceo".
   ?ceo pred:name "{name}".
}}
```

❖ **Query used in *coinSymbol* function found in *utils.py* :**

Throughout the project we need to be able to show/insert a coin symbol, for this purpose we use the following simple query:

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX coin: <http://wikicompany.pt/coin/>
SELECT ?symbol
WHERE {{
   ?coin pred:coinCode '{}'.
   ?coin pred:coinSymbol ?symbol.
}}
```

❖ **Query used in *coins* function found in *utils.py* :**

The following query allow us to read all the coins and their symbols from the database:

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX coin: <http://wikicompany.pt/coin/>
SELECT ?code ?symbol
WHERE {
   ?coin pred:coinCode ?code.
   ?coin pred:coinSymbol ?symbol.
}
```

❖ **Query used in *getCompanyFromDB* function found in *utils.py* :**

In this function we use two queries identical to the companiesInfo view.

❖ **Query used in *getCEOfromDB* function found in *utils.py* :**

In this function we extract a CEO information from our database, all the fields but the name are optional in case any of that fields are empty.

```
   PREFIX pred: <http://wikicompany.pt/pred/>
   SELECT ?image ?sexLabel ?birth ?worth ?description ?wikidataRef
   WHERE {{
     ?ceo a "ceo".
     ?ceo pred:name "{}".
     OPTIONAL {{?ceo pred:photo ?image. }}
     OPTIONAL {{?ceo pred:sex ?sexLabel.}}
     OPTIONAL {{?ceo pred:birth ?birth.}}
     OPTIONAL {{?ceo pred:worth ?worth.}}
     OPTIONAL {{?ceo pred:description ?description.}}
     OPTIONAL {{?ceo pred:wikidataRef ?wikidataRef.}}
   }}
```

We also need to extract all the CEO nationalities:

```
PREFIX pred: <http://wikicompany.pt/pred/>
SELECT ?nationalityLabel
WHERE {{
 ?ceo a "ceo".
 ?ceo pred:name "{}".
 OPTIONAL {{?ceo pred:nationality ?nationalityLabel. }}
}}
```

❖ **Query used in *isFavouriteCompany* template tag found in *favTags.py* :**

This is simple query using the *ASK* in SPARQL to find if a certain user has favorited a certain company. It filter through user and company symbol.

This result of this query is then passed to a function in **utils.py** that returns True or False given the result.

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX user: <http://wikicompany.pt/user/>
ASK {{
  ?user pred:favCompany ?company.
  ?company a "company".
  ?user pred:idUser "{}".
  ?company pred:symbol "{}".
}}
```

❖ **Query used in *isFavouriteCEO* template tag found in *favTags.py* :**

This is simple query using the *ASK* in SPARQL to find if a certain user has favorited a certain CEO. It filter through user and ceo name.

This result of this query is then passed to a function in **utils.py** that returns True or False given the result.

```
PREFIX pred: <http://wikicompany.pt/pred/>
PREFIX user: <http://wikicompany.pt/user/>
ASK {{
      ?user pred:favCeo ?ceo.
      ?ceo a "ceo".
      ?user pred:idUser "{}".
      ?ceo pred:name "{}".
}}
```

Publication of semantic data through RDFa

Throughout all pages we added RDFa to allow machines to read data we have. We'll present some examples throughout the app divided by section. RDFa implementation is dynamic as it must be presented by item whether there are 1 or 20 in the page.

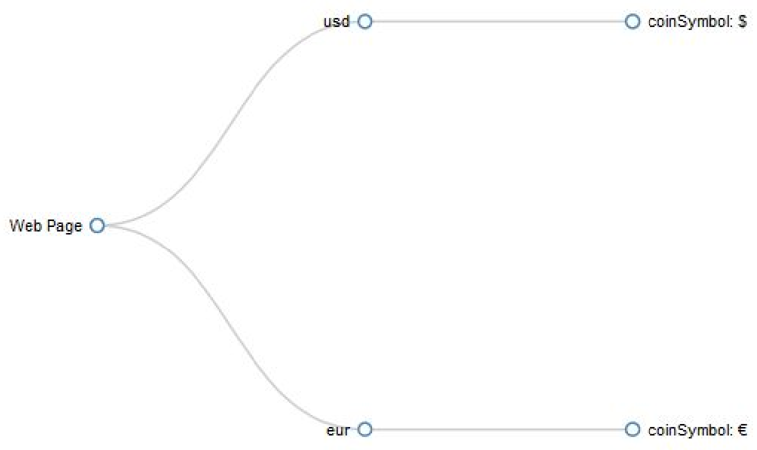
We make use of a validator of RDFa[3] that presents the results in a diagram to assure our triples can be understood by another machine.

Coin Dropdown

Given that coins are stored in our database, we RDFa to represent the coins that appear in navbar. This element appears in all pages throughout the app as it is part of the main layout. We make use of the attribute *about* to refer the subject URI, *typeof* for the type, and obviously *property* to make a reference to the predicate.

```
<ul id="dropdown1" class="dropdown-content"
xmlns:user="http://wikicompany.pt/user/"
xmlns:pred="http://wikicompany.pt/pred/">
      <li about="coin:usd" typeof="coin" property="pred:coinSymbol"><a
href="/selectCoin/usd/?last=/">$</a></li>
      <li about="coin:eur" typeof="coin" property="pred:coinSymbol"><a
href="/selectCoin/eur/?last=/">€</a></li>
</ul>
```

Running this html through the validator it produces the following diagram:



Which represents the following RDF triples as given by the validator as well:

[3] https://rdfa.info/play/

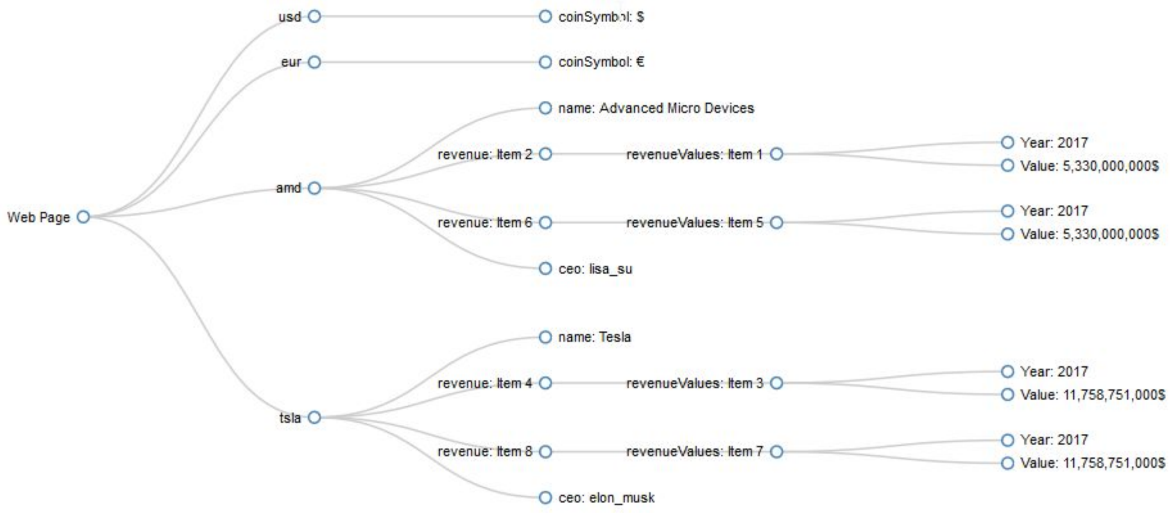
```
<http://wikicompany.pt/coin/usd>
   <http://wikicompany.pt/pred/coinSymbol> "$" .
<http://wikicompany.pt/coin/eur>
   <http://wikicompany.pt/pred/coinSymbol> "€" .
```

Home Page

In the page we show the companies with the biggest and lowest revenue in 2017. For each entry in the tables we declare the company URI and type and for each element we defined the property (predicate). To simplify the predicate expression we make use of prefixes declared with *xmlns* attribute.

```
<tbody xmlns:pred="http://wikicompany.pt/pred/">
<tr about="http://wikicompany.pt/companies/tsla" typeof="company">
    <td property="pred:name"><a href="/companies/TSLA">Tesla</a></td>
    <td rel="pred:revenue">
       <span rel="pred:revenueValues">
             <div property="pred:Year" hidden>2017</div>
             <div property="pred:Value">11,758,751,000$</div>
       </span>
    </td>
    <td property="pred:ceo" href="http://wikicompany.pt/ceo/elon_musk"> <a
href="/ceos/Elon Musk">Elon Musk</a></td>
</tr>
</tbody>
```

This is read by the validator as:



In this diagram we can see the coins are also represented in the page as they are part of the main layout present in every page.

Unfortunately when the same company appears in the two tables in the home two blank nodes will be created for each entry. This is due to a limitation in RDFa by not checking if the node is equal to a one already created (optimization). Unfortunately a blank node should have an id given automatically, it is impossible to create a decent solution, with no workarounds. However we don't consider this a problem as:

- Only happens when there are less than 8 companies with revenue values for 2017, which will be unlikely
- And the data is equal but correct. So the next person/machine to retrieve it will most likely reach the conclusion that the nodes are equal and optimize it (probably automatically). Even if that doesn't happen, only when interpreting data, shouldn't be a problem.

Companies Page

Similarly to the home for each row in the table we declare the subject and type.

```
<tbody xmlns:pred="http://wikicompany.pt/pred/">
<tr about="http://wikicompany.pt/companies/tsla" typeof="company">
    <td><span property="pred:name"><a
href="/companies/TSLA">Tesla</a></span></td>
    <td ><span property="pred:symbol"><a
href="/companies/TSLA">TSLA</a></span></td>
    <td property="pred:industry"> Automotive Industry</td>
</tr></tbody>
```

That will output to a set of triple for each company present in the table.

```
<http://wikicompany.pt/companies/tsla>
   <http://wikicompany.pt/pred/name> "Tesla"@en;
   <http://wikicompany.pt/pred/symbol> "TSLA"@en;
   <http://wikicompany.pt/pred/industry> " Automotive Industry"@en .
```

Company Info Page

On the almost top element of the page we declare the *about* and the *prefixes* used in the page.

```
<div class="w3-row-padding" ; style="margin-top:70px"
xmlns:pred="http://wikicompany.pt/pred/"
xmlns:company="http://wikicompany.pt/company/"  about="company:tsla"
typeof="company">
```

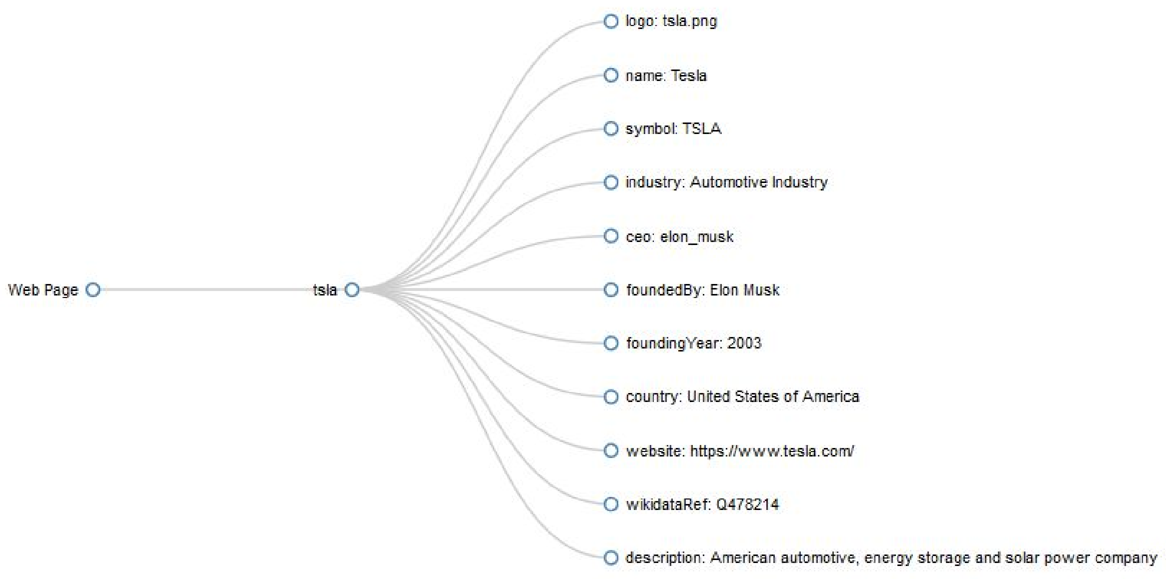
Then in each field associated to the company we set the predicate (as a example:

```
<!-- Industry -->
<h4>Industry : </h4>
```

```
<span property="pred:industry"><a href="/companies?industry=Automotive
Industry">Automotive Industry</a></span>

<!-- CEO -->
<h4>CEO : </h4>
<ul>
    <li><a href="/ceos/Elon Musk">Elon Musk</a></li>
    <span property="pred:ceo" href="http://wikicompany.pt/ceo/elon_musk"
hidden></span>
</ul>
```

This results as a simple diagram with a single nodes with all the fields associated to the company:


Web Page
tsla
logo: tsla.png
name: Tesla
symbol: TSLA
industry: Automotive Industry
ceo: elon_musk
foundedBy: Elon Musk
foundingYear: 2003
country: United States of America
website: https://www.tesla.com/
wikidataRef: Q478214
description: American automotive, energy storage and solar power company

CEO's Page

This follows the same structure of Companies page but this time is for CEO's. In each table row a CEO is presented and its fields tagged appropriately.

```
<tbody xmlns:pred="http://wikicompany.pt/pred/">
<tr about="http://wikicompany.pt/ceo/elon_musk" typeof="ceo">
    <td><span property="pred:name"><a href="/ceos/Elon Musk">Elon
Musk</a></span></td>
    <td property="pred:birth">28 June 1971</td>
    <td property="pred:sex"> Male</td>
</tr>
<tr about="http://wikicompany.pt/ceo/lisa_su" typeof="ceo">
    <td><span property="pred:name"><a href="/ceos/Lisa Su">Lisa
Su</a></span></td>
    <td property="pred:birth">November 1969</td>
    <td property="pred:sex"> Female</td>
</tr></tbody>
```

This results, by using the validator, in a simple set of tuple for each CEO:

```
<http://wikicompany.pt/ceo/elon_musk>
   <http://wikicompany.pt/pred/name> "Elon Musk"@en;
   <http://wikicompany.pt/pred/birth> "28 June 1971"@en;
   <http://wikicompany.pt/pred/sex> " Male"@en .
<http://wikicompany.pt/ceo/lisa_su>
   <http://wikicompany.pt/pred/name> "Lisa Su"@en;
   <http://wikicompany.pt/pred/birth> "November 1969"@en;
   <http://wikicompany.pt/pred/sex> " Female"@en .
```

CEO's Info Page

Following the same structure as the companies info page we first declare what the page is about and the prefixes used.

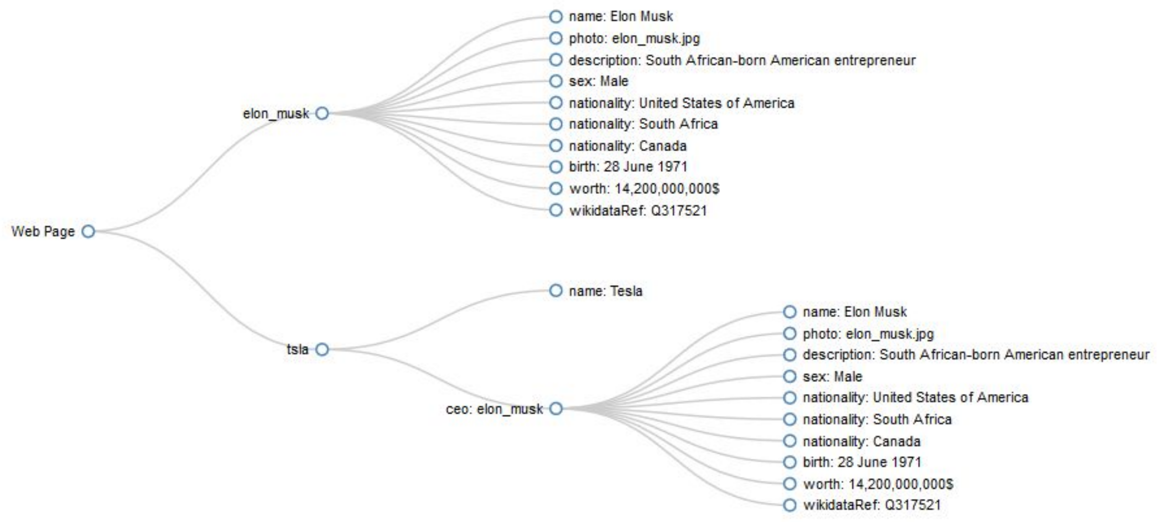
```
<div class="w3-row" xmlns:pred="http://wikicompany.pt/pred/"
xmlns:ceo="http://wikicompany.pt/ceo/"
xmlns:company="http://wikicompany.pt/company/"
about="http://wikicompany.pt/ceo/elon_musk" typeof="ceo">
```

Then for each of the fields that represent a CEO there is the property associated with it (example):

```
<!-- Wikidata Ref -->
<li><h4>Wikidata Reference</h4> <a property="pred:wikidataRef"
href="https://www.wikidata.org/wiki/Q317521">Q317521</a></li>

<!-- Companies -->
<li ><h4>CEO of:</h4>
<span about="company:tsla" property="pred:name"><a
href="/companies/TSLA">Tesla<br></a></span>
<span rev="pred:ceo" href="http://wikicompany.pt/company/tsla"
hidden>company:tsla</span>
</li>
```

We make use of the *rev* attribute to link the company to the CEO in case they are both in the database. With this in mind we get this following diagram:



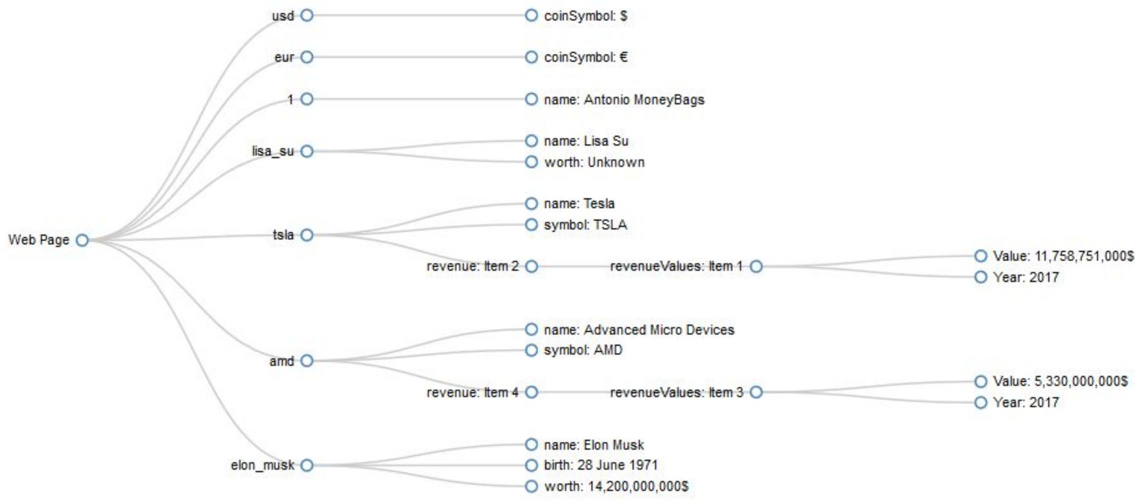
The information about the CEO inside the company node is generated automatically with the information from above as they have the same URI.

Profile Page

At the top of the html page we load the prefixes that we will use, the same way as last pages show.

We then for each table (2 for CEO's of which one is suggestions to favorite and 2 for Companies of which one is suggestions to favorite) do similar to pages above to tag nodes and their triples appropriately.

The result is very complete even though the user only favorited 1 company and 1 CEO:



Integration of Wikidata data

To simplify the queries to Wikidata we created a function that given a query returns the result to the query in Wikidata formatted. It is named *queryWikidata*() and makes uses of the requests python package to query the SPARQL endpoint with a POST (to get by the limit of url length)[4]. By choosing the correct header to get the contents in *json* we can make use of the function *transformResults* (also in **utils.py**) already created for GraphDB to format the result, as the output of a SPARQL endpoint is standard.[5]

Most queries make use of label service provided by Wikidata.

The queries present may present "{" escaped as we make used of the format python function[6].

- **Query used in *adminAddCoin* view found in *views/adminViews.py* :**

```
SELECT ?symbol
WHERE {{
 ?coin wdt:P498 "{}".
 ?coin wdt:P5061 ?symbol.
 SERVICE wikibase:label {{ bd:serviceParam wikibase:language
"[AUTO_LANGUAGE],en". }}
}}
```

This query is used to get the symbol of a coin given a code. Makes use of the property ISO 4217 code and unit symbol.

- **Query used in *getWikidataCompany* function found in *views/utils.py* :**

Inside this function we make 3 queries to WIkidata. The first one verifies if the company with the symbol given exists in Wikidata. This company must be in a stock market exchange. It makes use of the property stock exchange and its sub property ticker symbol. It extracts the label and description using the label service of Wikidata.

```
SELECT DISTINCT ?id ?idLabel ?idDescription
WHERE {{
?id wdt:P414 ?exchange .
?id p:P414 ?exchangeSub .
?exchangeSub pq:P249 '{}' .
SERVICE wikibase:label {{ bd:serviceParam wikibase:language
"[AUTO_LANGUAGE],en". }}
}}
```

[4] https://www.mediawiki.org/wiki/Wikidata_Query_Service/User_Manual#SPARQL_endpoint
[5] https://www.w3.org/TR/sparql11-results-json/
[6] https://docs.python.org/3.4/library/functions.html#format

The second, extracts all the information we need from Wikidata about that company. We make use of the optional function to return results even if one field is missing.

```
SELECT ?industryLabel ?website ?ceoLabel ?founderLabel ?foundingYear
?countryLabel ?logo
WHERE{{
  OPTIONAL {{ wd:{id} wdt:P452 ?industry.}}
  OPTIONAL {{ wd:{id} wdt:P856 ?website.}}
  OPTIONAL {{ wd:{id} wdt:P169 ?ceo.}}
  OPTIONAL {{ wd:{id} wdt:P112 ?founder.}}
  OPTIONAL {{ wd:{id} wdt:P571 ?foundingYear.}}
  OPTIONAL {{ wd:{id} wdt:P17 ?country.}}
  OPTIONAL {{ wd:{id} wdt:P154 ?logo.}}
  SERVICE wikibase:label {{ bd:serviceParam wikibase:language
"[AUTO_LANGUAGE],en". }}
}}
LIMIT 1
```

The last, gets all the revenue values and respective years. This was the most difficult query to make as we didn't find a clear explanation on how to get all the info from a property. Example wikidata queries use most time this qualifier *wdt:PXXX,* but this only returns the most relevant result and not all items. The answer however can be found here[7]. By making use of the *p, ps* and *pq* qualifier we can get all the items and its properties.

```
SELECT ?revValue ?revYear
WHERE{{
   wd:{} p:P2139 ?stat.
   ?stat ps:P2139 ?revValue;
   pq:P585 ?revYear.
}}
```

- **Query used in *getWikidataCEO* function found in *views/utils.py* :**

Given that Wikidata has multiple properties for a given name we decided to take the label to search for the CEO's as we don't have to deal with strange origin language names and Lisa **Su** for example, where Su would be characterized as a chinese character. Also these CEO's are most known for their label in Wikidata.

[7] https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Qualifiers

But another problem from Wikidata, appeared. Sometimes data is not bidirectional. Searching for the CEO of AMD returns Lisa Su but searching for Lisa Su and get her position as CEO returns nothing. So we make use of *union* to join the results of persons that held a position of CEO in a stock market company and the CEO's of the stock market companies. We then make use of the distinct option to filter the results that are equal, as there is so bidirectionality in Wikidata after all.

```
SELECT DISTINCT ?id ?idLabel ?idDescription
WHERE {{
 ?company wdt:P414 ?exchange .
 ?company wdt:P169 ?id.
 ?id wdt:P735 ?firstNameItem.
 ?id wdt:P734 ?lastNameItem.
 SERVICE wikibase:label { bd:serviceParam wikibase:language
"[AUTO_LANGUAGE],en". }
}
 UNION {
    ?id wdt:P31 wd:Q5.
    ?id p:P39 ?subItem.
    ?subItem ps:p39 wd:Q484876.
    ?subItem pq:P642 ?company.
    ?company wdt:P414 ?exchange .
  }}
```

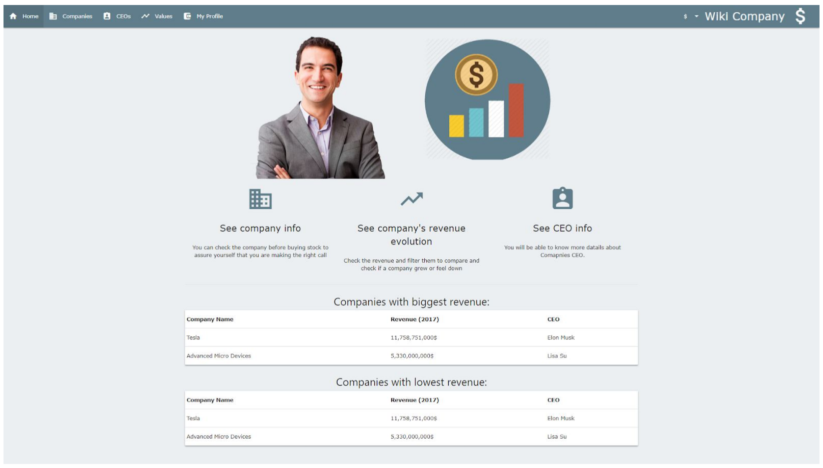
Now that we have the list of CEO's of stock market companies which is only around 200. We cycle through them to find a name match in Python and return it's Wikidata reference.

```
SELECT ?image ?sexLabel ?birth ?worth
WHERE {{
 OPTIONAL {{wd:{id} wdt:P18 ?image. }}
 OPTIONAL {{wd:{id} wdt:P21 ?sex.}}
 OPTIONAL {{wd:{id} wdt:P569 ?birth.}}
 OPTIONAL {{wd:{id} wdt:P2218 ?worth.}}
 SERVICE wikibase:label {{ bd:serviceParam wikibase:language
"[AUTO_LANGUAGE],en". }}
}}
```

We then query Wikidata again to get the CEO's nationality.

```
SELECT ?nationalityLabel
WHERE {{
 OPTIONAL {{wd:{} wdt:P27 ?nationality. }}
 SERVICE wikibase:label {{ bd:serviceParam wikibase:language
"[AUTO_LANGUAGE],en". }}}}
```

UI Functionality



See company info

You can check the company before buying stock to assure yourself that you are making the right call

See company's revenue evolution

Check the revenue and filter them to compare and check if a company grew or feel down

See CEO info

You will be able to know more datails about Comapnies CEO.

Companies with biggest revenue:

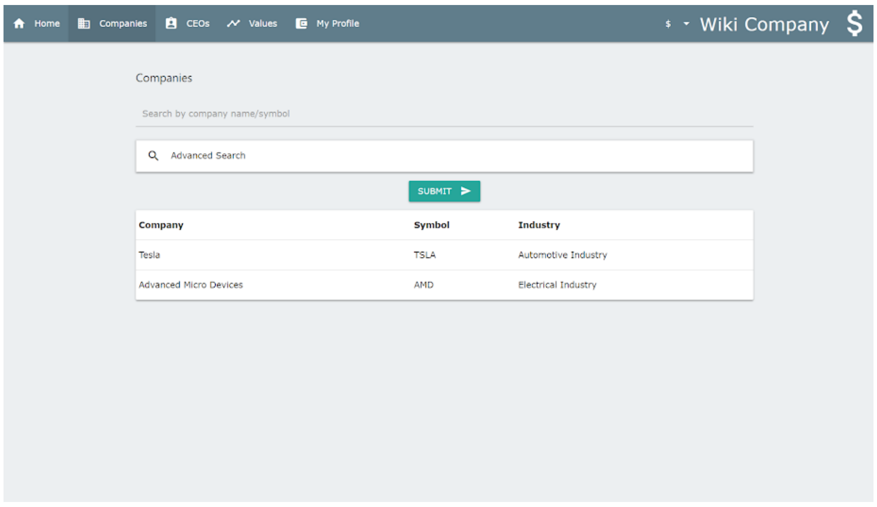
Company Name	Revenue (2017)	CEO
Tesla	11,758,751,000$	Elon Musk
Advanced Micro Devices	5,330,000,000$	Lisa Su

Companies with lowest revenue:

Company Name	Revenue (2017)	CEO
Tesla	11,758,751,000$	Elon Musk
Advanced Micro Devices	5,330,000,000$	Lisa Su

Main Page

In the main page, we can see in the upper part all the different pages that can be accessed through the use of a navbar. We also have two tables where we can find information about the companies that have gotten the most revenue in 2017 and which had the lowest. These tables can be ordered alphabetically by name, by revenue or by CEO name (sortable).



Companies

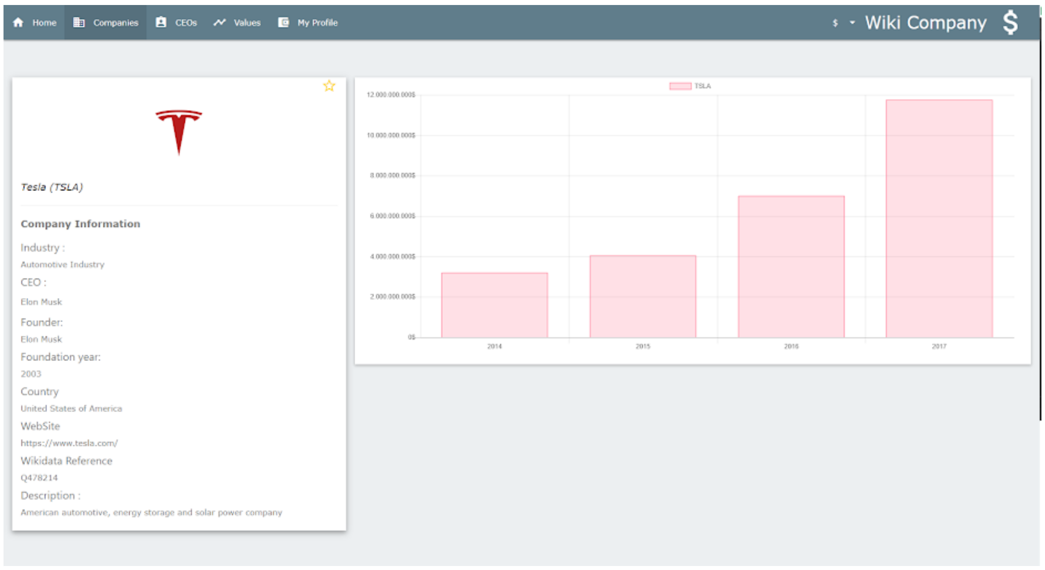
Search by company name/symbol

Advanced Search

Company	Symbol	Industry
Tesla	TSLA	Automotive Industry
Advanced Micro Devices	AMD	Electrical Industry

Companies Page

In this page we find a table with all the companies, their symbol, and the industry they belong to. It can be ordered by company name, symbol or industry. We can also use the search function in order to look for a certain company, filtering by name, symbol, industry, founder, CEO name, founding year and country. If we click in the company name or symbol we will be redirected to company page.



Tesla (TSLA)

Company Information

Industry :
Automotive Industry
CEO :
Elon Musk
Founder:
Elon Musk
Foundation year:
2003
Country
United States of America
WebSite
https://www.tesla.com/
Wikidata Reference
Q478214
Description :
American automotive, energy storage and solar power company

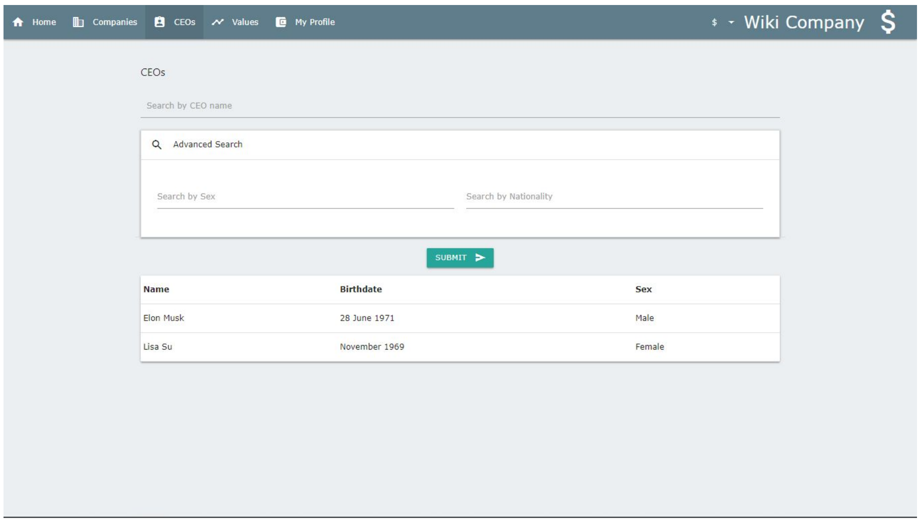
A Company page

Here we get the company info obtained from the database using the company symbol. There is also a chart where we can see the revenue over the years.

If we are logged in, there is a option to choose a company as a favourite, if we click on the star and we aren't logged we will be redirected to the login page, else it will refresh and the company will be added to the users favourite companies list.

By clicking in the CEO name the user will be redirected to the CEO info page, unless the ceo isn't in the database in that case there will be no link. If the user clicks on the industry, founding year, founder or country him will be redirected to the companies search page with that filter applied (the user still has to confirm the search).

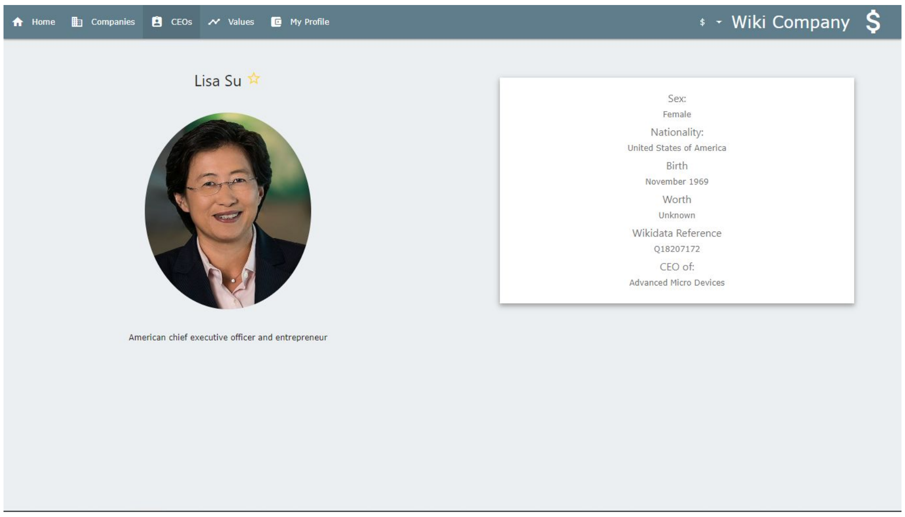
If we click on the wikidata ref we will be redirected to the company wikidata page, else if we click on the company website link we will be redirected to the company website.



Name	Birthdate	Sex
Elon Musk	28 June 1971	Male
Lisa Su	November 1969	Female

CEOs Page

In this page we find a table with all the company's CEOs, their birth date, and their sex. The table is sortable. There is a search function in order to look for a certain CEO, and able to filter by name, sex or nationality. If we click in the CEO's name we will be redirected to the CEO's page.



Lisa Su ☆

American chief executive officer and entrepreneur

Sex:
Female

Nationality:
United States of America

Birth
November 1969

Worth
Unknown

Wikidata Reference
Q18207172

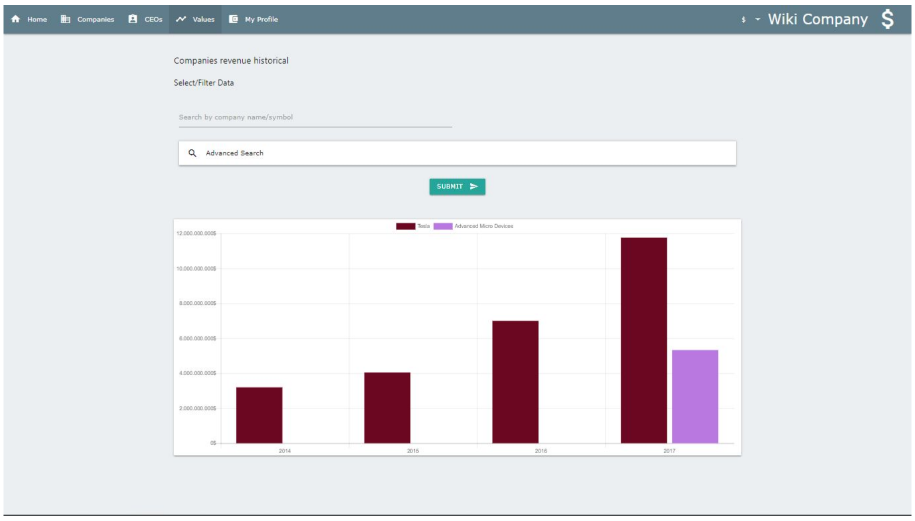
CEO of:
Advanced Micro Devices

Certain CEO page

Here we get the CEO info obtained from the database using CEO's name. If we are logged in we can choose a CEO as favourite, else, redirected to login page.

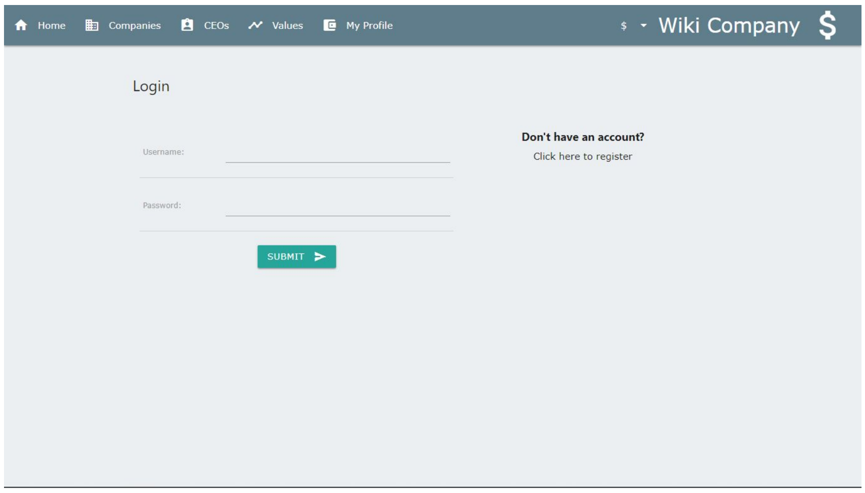
By clicking on the sex or nationality we will be redirected to the CEOs search page with that filters applied. The wikidata ref is a working link to the ceo wikidata page.

If we click on the company name after the CEO of header we will be redirected to the company info page.



Values Page

In this page we find a chart with all the companies revenue history. There is a search function in order to look for a certain company, filtering by name, symbol, industry, founder, CEO name, founding year and country.



Login

Username:

Password:

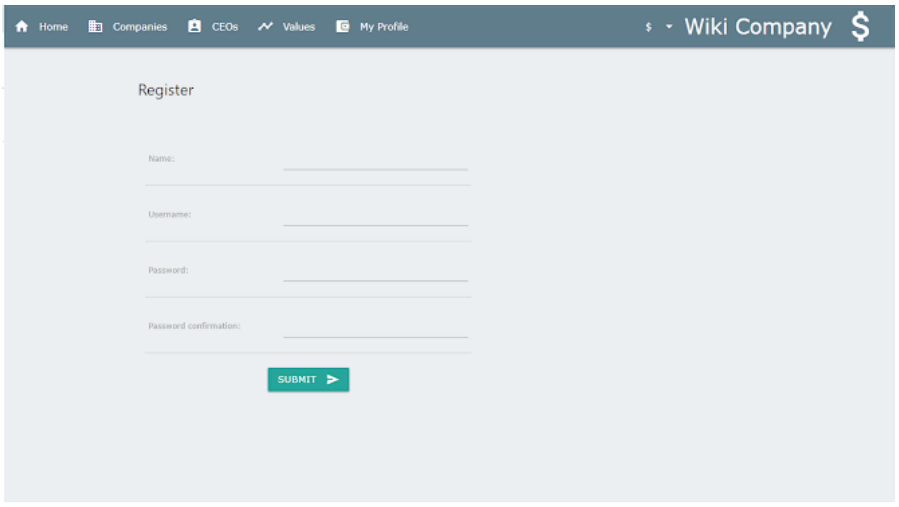
SUBMIT

Don't have an account?

Click here to register

Login Page

If no one is logged in, login page will appear when choosing *“My profile”*.



Register

Name:

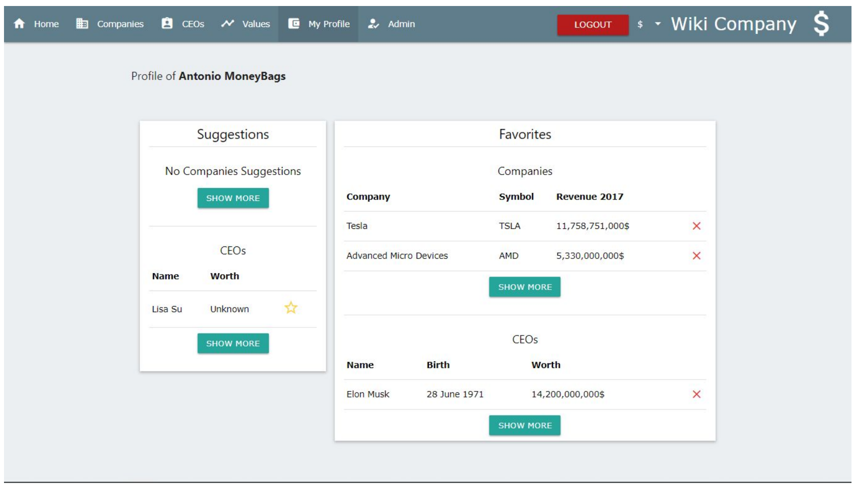
Username:

Password:

Password confirmation:

SUBMIT

Register Page



Profile of **Antonio MoneyBags**

Suggestions

No Companies Suggestions

CEOs

Name	Worth	
Lisa Su	Unknown	☆

Favorites

Companies

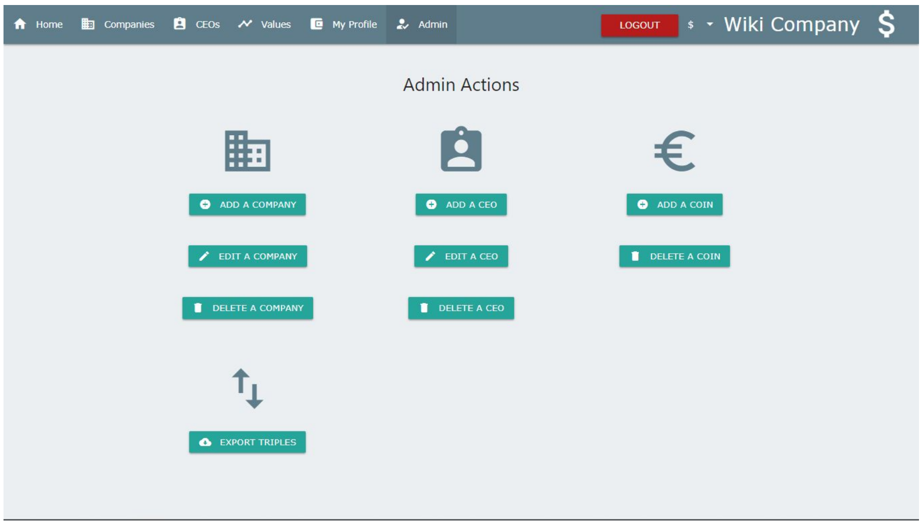
Company	Symbol	Revenue 2017	
Tesla	TSLA	11,758,751,000$	×
Advanced Micro Devices	AMD	5,330,000,000$	×

CEOs

Name	Birth	Worth	
Elon Musk	28 June 1971	14,200,000,000$	×

Profile Page

In case the user is logged in, the user page will be shown in *“My Profile”* tab. This page shows user information, as username, and tables with the chosen favourite Companies and CEOs. There are suggestions for the user to favorite. These suggestions are based on what companies and CEOs the users has already favorited.



Admin Page

If the user is super-user, an admin tab will appear. This tab allows to administrate the page. There are several functions supported.

Add a company: add a company, retrieving company data from wikidata with just the company symbol

Edit a company: modify company info in our database.

Delete a company: delete a company from our database.

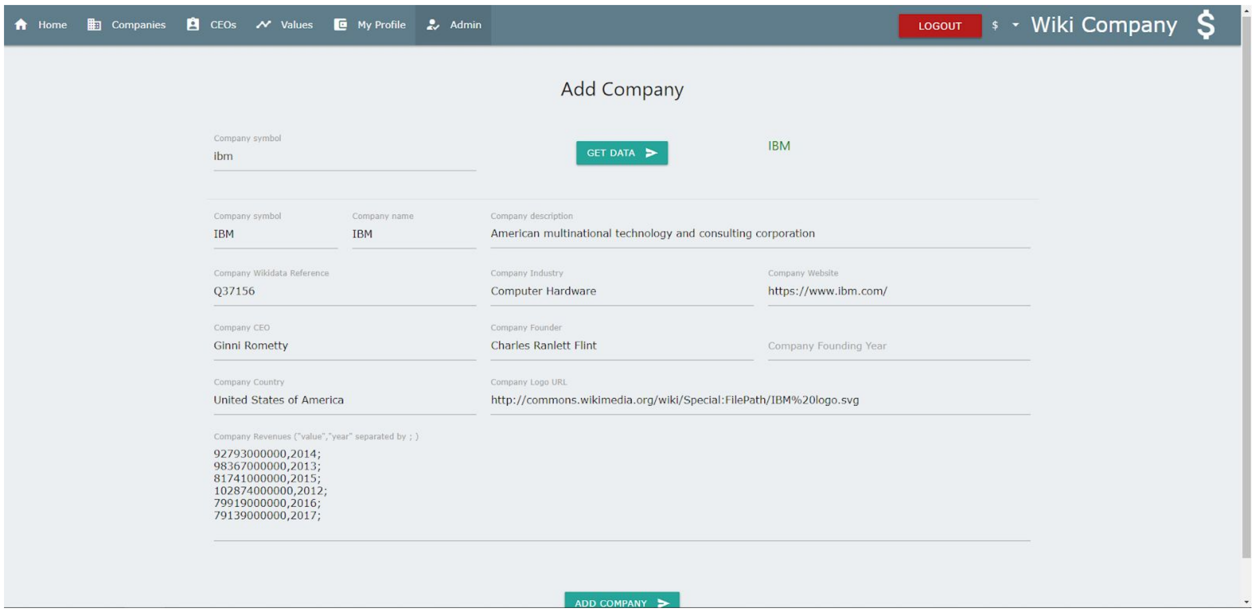
<u>Add CEO</u>: add a CEO retrieving its data from wikidata just typing CEO's name

<u>Edit CEO:</u> modify CEO info in our database.

<u>Delete CEO:</u> delete a CEO from our database.

<u>Add/Edit Company/CEO Page</u>

This page asks the user to input the company symbol or the CEO name. If exists in the Wikidata or, in case, if editing, in the database, it fills the form with all the data automatically.



Add Company

Company symbol: ibm

IBM

Company symbol: IBM	Company name: IBM	Company description: American multinational technology and consulting corporation
Company Wikidata Reference: Q37156	Company Industry: Computer Hardware	Company Website: https://www.ibm.com/
Company CEO: Ginni Rometty	Company Founder: Charles Ranlett Flint	Company Founding Year
Company Country: United States of America	Company Logo URL: http://commons.wikimedia.org/wiki/Special:FilePath/IBM%20logo.svg	

Company Revenues ("value","year" separated by ;)

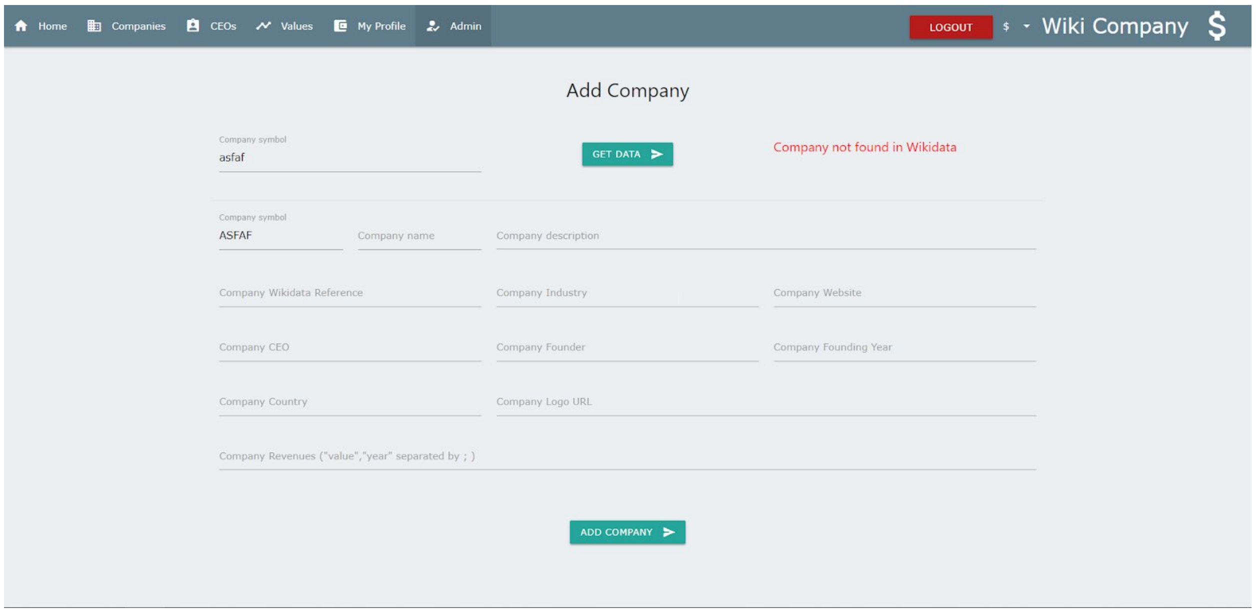
```
92793000000,2014;
98367000000,2013;
81741000000,2015;
102874000000,2012;
79919000000,2016;
79139000000,2017;
```

<u>Add company - Wikidata result</u>

The user simply needs to click add, or if editing change the values and click to submit the form.

If adding, if the symbol/name is not in wikidata it will say so but will still let the user add the company manually.

If editing, if the symbol/name is not in the database, it will display a message and will not show the form.



Add Company

Company symbol: asfaf

GET DATA

Company not found in Wikidata

Company symbol: ASFAF

Company name

Company description

Company Wikidata Reference

Company Industry

Company Website

Company CEO

Company Founder

Company Founding Year

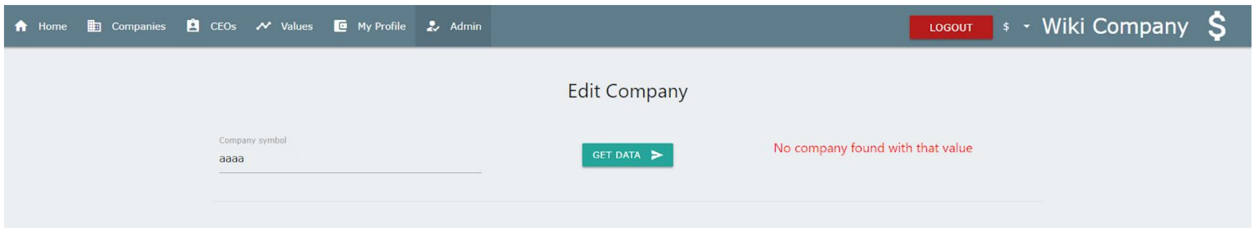
Company Country

Company Logo URL

Company Revenues ("value","year" separated by ;)

ADD COMPANY

Add company - Not found on Wikidata



Edit Company

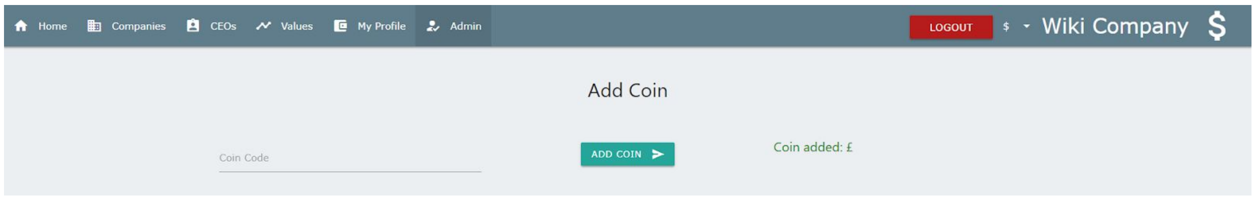
Company symbol: aaaa

GET DATA

No company found with that value

Edit company - Not found on database

Add a coin: by typing the coin code, the symbol is fetched from wikidata and added to our database. It also check if the coin exists in our forex exchange API to convert from dollar (usd)



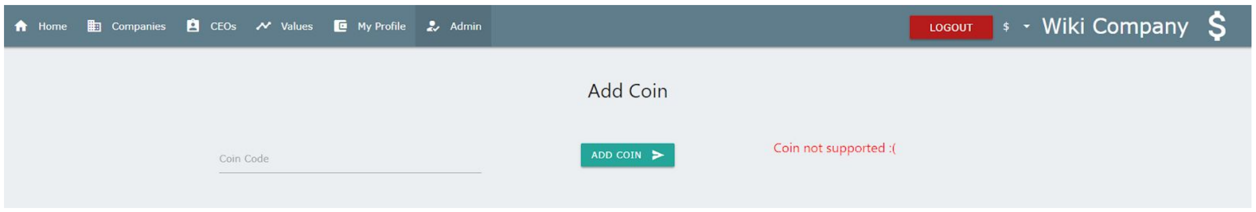
Add Coin

Coin Code

ADD COIN

Coin added: £

Add coin - Successfully added



Add Coin

Coin Code

ADD COIN

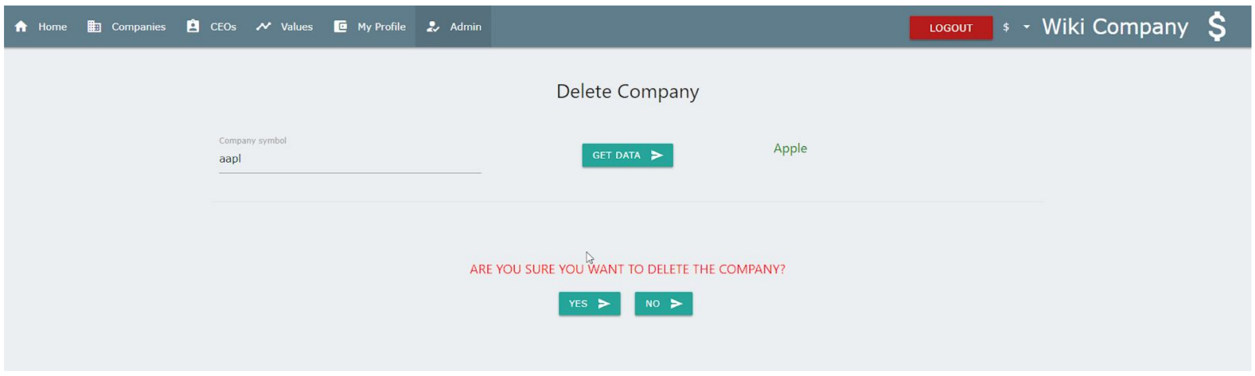
Coin not supported :(

Add coin - Invalid coin

Delete coin: delete a coin from our database.

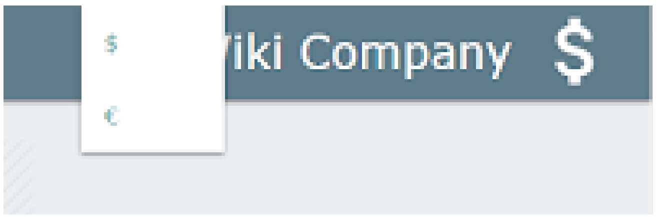
Delete Company/CEO/Coin page

The page asks for the symbol/name/code to delete from the database. If it finds the item it will ask if the user really wants to delete. If the user clicks yes, the item will be irreversible deleted.



Delete company - Confirm deletion

Export triples: download in n3 format the current database. Makes use of the REST API of GraphDB to get all the statements.



Coin selector.

In the upper right part of the page we find a coin selector. That allow us to choose the coin we would like to see in the values.

Additional Considerations

Layout System

Given that TP1 layout system was well designed we decided to maintain it as the system makes changes after the initial design easy, making the whole frontend flexible. If we want to add a item in the navbar, or simply load another script/CSS style we simply need to add it to the base layout and all the pages are updated/changed.

The front-end got a bit of a makeover as the project focus is now different.

Custom template tags[8]

Template tags are something we can customize to our needs effectively running Python code in the template system.

We created functions to format bigger numbers (add commas), generate random integers for the charts colours, to check if a user has a company/CEO favorited right on the template and to get a list of coins and coin symbol to feed our selector in the navbar.

We divided these tags into three files, named accordingly to their function. They can be found inside ***bolsa/templatetags/.***

Django Authentication System[9]

As last project, we implemented authentication by using Django own authentication system.

The solution was to use the user id from Django, that is unique, as part of the URI and a object given the predicate "*idUser*" and effectively associate the two databases that way.

With this we have the best of both worlds: our information inside GraphDB as requested and the security already implemented in Django.

Division of views and urls

As with last project, dividing the views into different files at the beginning proved very effective for organization. As a view file gets bigger the more urls that are added dividing them according to function/part of the app is crucial to debug.

There are 5 views files inside ***bolsa/views***.

Coin Selector

Given the similarity of the theme from TP1, the functionality was implemented. However, this time, as the functionality is already designed from the beginning we added a couple of adjustments as the previous implementation was far from ideal:

- Calculation is now made in the view function and not on the template system

[8] https://docs.djangoproject.com/en/2.1/howto/custom-template-tags/
[9] https://docs.djangoproject.com/en/2.1/topics/auth/

- Creation of a function in ***bolsa/utils.py*** named *forexValuesExchange* which will calculate the value in a list to the coin currently selected by the user. This allows for more streamlined development and clear repetitive code.
- As the values this project works with are much bigger we needed to formatted with commas to be readable for the user. This is done automatically by the function above n or with a template tag as the charts require a callback function in to format the values[10]
- The conversion value is not stored in the database anymore. Conversion is done on-the-fly with real-time values from a API[11]
- Ability to add or delete coins. When adding the symbol is fetched from Wikidata and the forex API is queried to check if it supports the new coin

Utils Functions

As we learned from the previous work, standardizing the processes in the code is very helpful to improve readability and quick comprehension among team members.

As a result, we created a file which includes a lot of functions to streamline processes, and take code away from the views files. The functions deal with:

- Querying the database and Wikidata
- Transforming the results into a workable format in python
- Converting values to a certain coin
- Getting a coin value (does a *GET* request)
- Transforming data for the charts
- Check the result of a ask query
- Getting a company/CEO from database/Wikidata
- Transform inputs from user, CEO nationalities and revenue values for a company into part of a query

Creating the database dynamically

As with last work we tried to make running the project for the first time as easy as possible. One of these aspects is the creation of the repository of the database and the upload of data automatic as Django starts. This proved more difficult than last time.

At first glance, the Python package provided *s4api*[12] seemed more than adequate as it includes functions for creating/deleting repositories and uploading data files to it.

However we run into a few problems. Creating repositories was not working and uploading files also. Finding the problem was hard as the error outputs were a bit generic. Seems

[10] https://www.chartjs.org/docs/latest/axes/labelling.html#creating-custom-tick-formats
[11]https://www.freeforexapi.com/api/live?pairs=EURUSD,USDEUR,USDKRW
[12] https://pypi.org/project/s4api/

that this python package is not well maintained as the documentation is non existent at the moment[13].

We sorted to the REST API provided by GraphDB. After a bit of time using *requests* python package[14], we were successfully creating and deleting repositories. However uploading our *.n3* proved to be a challenge.

After looking at the *s4api* code at Github[15] we realized this package was simply a wrapper using *requests* package for the REST API of GraphDB and why loading our file was not working as the method ***upload_data_file*** accepts a file in RDF/XML.

A simple check at the non-standard content type for *.n3* files[16] proved and by interacting directly with the REST API, uploading a data file was correctly working.

With all of this applied to our work, the user doesn't need to configure a repository as it's done automatically. Only thing the user needs to do is open GraphDB. If it isn't open our Django app will spit a warning but will still run.

Frameworks used

We used a couple of frameworks that we need to reference/thank:

- Materialize - front-end styling and components
- W3.CSS - front-end styling
- Jquery - *JS* scripting
- Chart.js - included graphs
- Material Icons - icons on the app
- Sortable.js - sortable tables

Conclusions

Thanks to this project, we have acquired a better knowledge of the theoretical area of the course and applied in a deeper way, what we had learned in the practical one. It has also allowed us to form a multicultural group in which we have seen the strengths and weaknesses of each component of the group, in order to take advantage of those strengths and try to improve each other weaknesses, in order to get to the final result which is a functional page where you can do everything that was planned at the starting point.

Configuration to execute the application

[13] http://docs.s4.ontotext.com/display/S4docs/Python+SDK and https://s4api.readthedocs.io/
[14] http://docs.python-requests.org/en/master/
[15] https://github.com/Ontotext-AD/S4/blob/master/S4-Clients/Python-client/s4api/graphdb_api.py
[16] https://www.w3.org/2008/01/rdf-media-types

We tried to make the app run almost automatically without much user configuration. GraphDB repository is created dynamically given that GraphDB (with its REST API operational) is running, and its default data imported. There is also a file which lists all the requirements used in the project so that the user can easily install python dependencies by calling *pip -r requirements.txt*.

Base requirements:

- Python >= 3.7 (because of a datetime function used[17])
- GraphDB
- Pycharm (optional)
- Tested in Windows and Ubuntu (as development was done in both simultaneously)

Python packages requirements:

- Django==2.1
- s4api==1.1.0
- requests==2.20.1

To run the project:

1. Check if you have all the requirements:
    a. Run "pip install -r requirements.txt"

        OR

    b. Install manually each package
2. Start GraphDB
3. Run Django Project
    a. Import to a project in Pycharm proj1 folder which contains all django project

        OR

    b. Run "python manage.py runserver"

Considerations:

GraphDB repository is created dynamically on each Django start.

bolsa/repmanager.py contains all this code create the repository.

User Accounts (admin):

There is one user account already created which is the admin. This account was created to test and see the functionalities of the system fully without the need to tamper around.

[17] https://docs.python.org/3/library/datetime.html#datetime.datetime.fromisoformat

Details of this account is included in a file at the root of the project.

References

https://www.w3schools.com/

https://docs.djangoproject.com/en/2.1/

https://tutorial.djangogirls.org/en/

https://simpleisbetterthancomplex.com/

http://docs.python-requests.org/en/master/

https://www.w3.org/TR/sparql11-query/

https://docs.python.org

https://wikidata.org