Sciences Po
Centre d'Études Européennes

May 2015

EUROPOLIX

TECHNICAL MANUAL

Romain LALANDE
romain.lalande@gmail.com

Table of Contents

What is Europolix?....4
Overview of the website....5
Welcome page....5
Login page....5
Administration page....6
Import page....6
Act ids validation page....8
Ministers' attendance validation page....9
Act data validation page....11
Database management page....12
Export page....13
History page....14
Fields to retrieve....15
Database....16
Naming conventions....16
Import tables or tables used for the web application....17
Tables containing act data and used for the statistical analysis....19
Tools used....21
Django files....22
How is organized the website....22
The applications....22
europolix....22
login....23
import_app....23
act_ids....24
attendance....25
act....25
db_mgmt....29
export....29
history....29
Other folders....30
Common module....30
Templatetags....31
Commands....32
Explanation....32
import_app....33
export....33
Statistics module....35
Types of analyses....35
Structure of the module....35
queries.py....35
init.py....36
get.py....37
write.py....38
common.py....39
queries.csv....39
check_queries.sql....40

query folder....40
result folder....41
Parameters....41
factor....41
res....41
count....42
total and res_total....42
query....42
Model....42
variable....42
variable_2....43
res_2....43
exclude_values....43
filter_vars_acts....43
filter_vars_acts_ids....43
exclude_vars_acts....44
check_vars_acts....44
check_vars_acts_ids....44
adopt_var....44
periods....45
question....45
percent....45
Examples....45
Nombre d'actes....45
Nombre de mots moyen....48
Pourcentage d'actes avec un vote public....49
Parmi les votes AdoptCSAbs=Y, pourcentage de chaque Etat membre....50
Pourcentage d'actes avec au moins un EM sans statut 'M' (et au moins un 'CS' ou 'CS_PR') 51
Pourcentage d'actes provenant de la Commission et adoptés par procédure écrite....52
Nombre moyen de EPVotesFor1-2....54
Pourcentage d'actes adoptés en 1ère lecture parmi les actes de codécision....57
Durée de la procédure (= Moyenne DureeTotaleDepuisTransCons ET DureeProcedureDepuisTransCons)....58
Call a query....60
Run a statistical analysis....60
Update the eurlex or oeil url....61
Add a new variable....62
Update the model....62
Update the form (optional)....62
Retrieve the variable....62
Update the view....62
Display the proper name....63
Update the export functionality....63
Run the project on a server....64
Install the project....64
Administration access....64
TO DO....65

What is Europolix?

This section explains what is the project and what it is for.

Europolix is a project started by Sciences Po and Centre d'Études Européennes (CEE) in October 2012. Its aim is to build a database gathering European acts from *eurlex* (council) and *oeil* (parliament) - and *prelex* until the beginning of 2015 (commission).
This project can be seen as a mini website where information is automatically retrieved from the websites above and where experts validate it, act after act.
On top of the creation of a common database, a first goal is to cross validate data between the different European institutions and with an expert to highlight errors. In the end, we wish to have a clean database with no error.
A second goal is to do a statistical analysis of the cleaned acts. To this end, the website allows to extract all the acts of the database into a file readable in a spreadsheet tool.

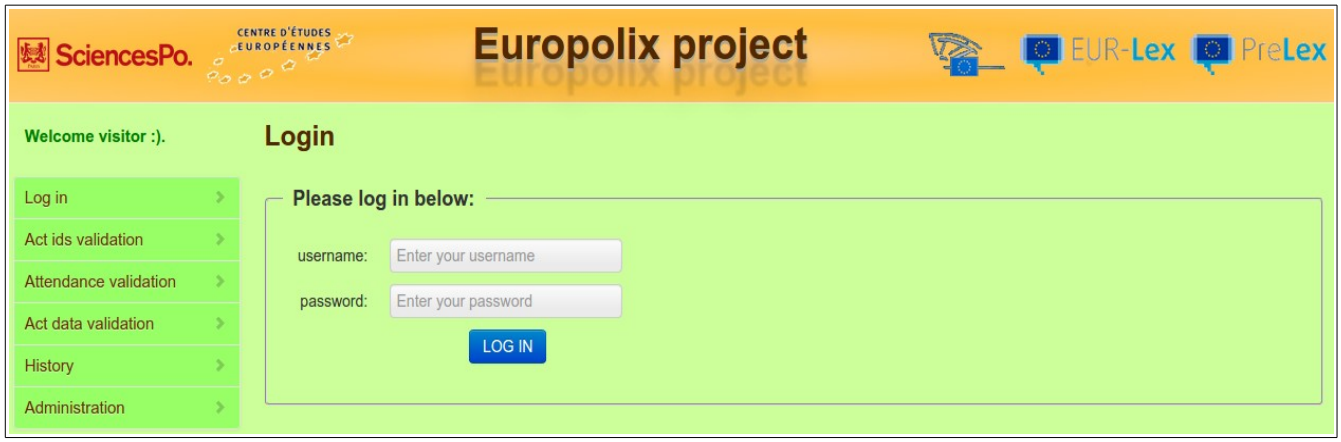
Overview of the website

This section shows how works the website with screenshots, practical and technical information.

Welcome page

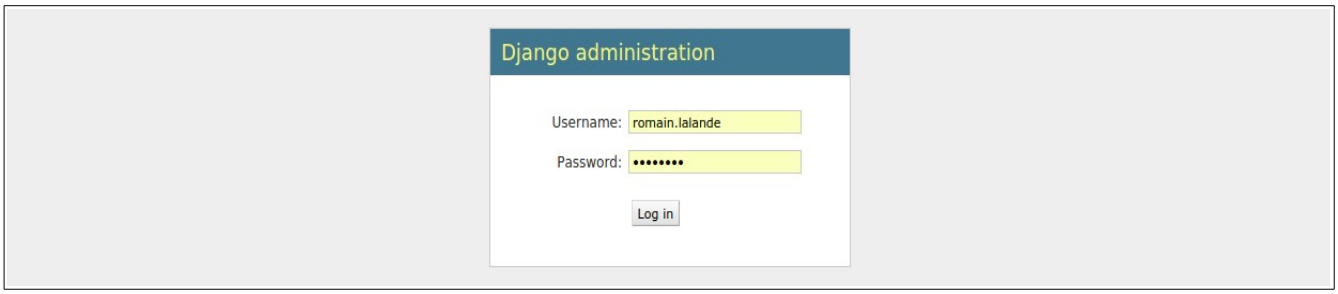
This is the homepage of the website. From the left menu, you can log in and access the administration. You can also import, export and validate acts.

Login page



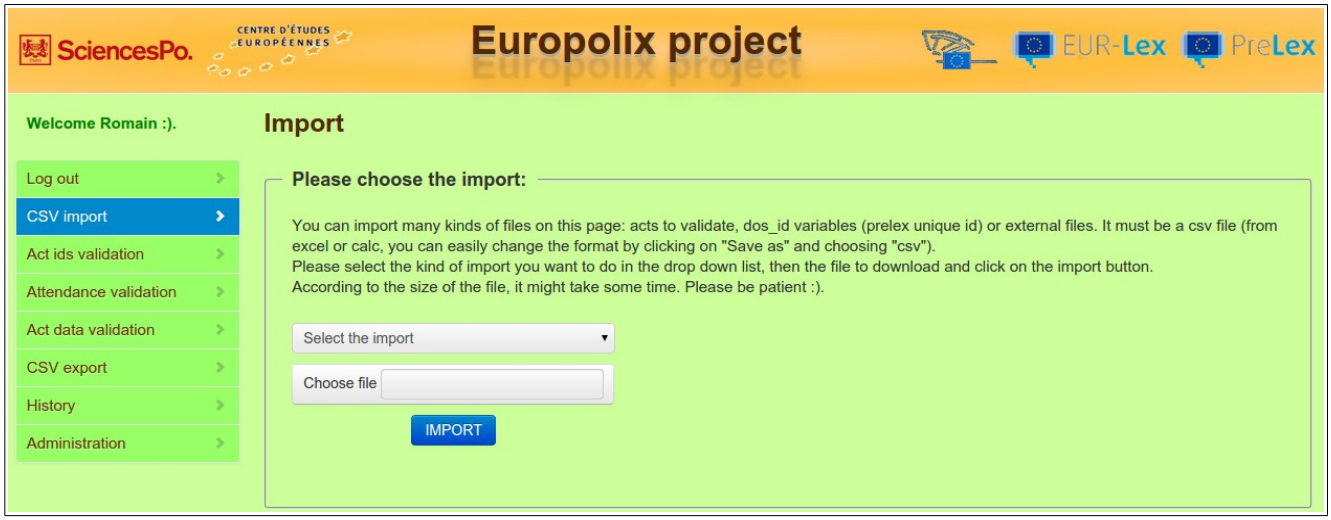
This is the login page (click on “Log in” from the menu). You must log in to get access the other functionalities of the website. If you have no id, you must ask the administrator to use the website. Enter your username and password. Then click on “Log in”.

Administration page



This is the administration tool (click on “Administration” from the menu). Only an administrator or staff with special privileges can access it.
Fill in your username and password and click on “Log in”.
The *Django* administration comes with a bunch of functionalities and can be customized. For more information, please have a look at https://docs.Djangoproject.com/en/dev/ref/contrib/admin/.

Import page



This is the import page. You can import different csv files that are first saved on the server and then processed.

- Import acts to validate: Import a file containing information on acts (information found in monthly summaries and then gathered in a spreadsheet document). Two steps:
  - 1/ the content of the file is saved under the tables *Act (*main data of the acts), *ActsIds* (ids of the acts), *adopt*_cs_contre (a*dopt_cs_contre* variable) and *adopt_cs_abs* (a*dopt_cs_abs* variable).
  - for each act newly inserted, we retrieve corresponding data (ids only) from the *eurlex* or *oeil* websites. Finally we update the database with this new information. A message is displayed if errors are encountered during the import.
- Import DosId: Import a file containing *prelex* unique id (called *dos_id)* and the *no_celex variable* to do the mapping (*no_celex* is contained either in the *VAL_CHP_SUP_01* or *VAL_CHP_SUP_03* column. The content of the file is saved in the database under a table called *ImportDosId.*

- Import CodeAgenda*: This file is used to do the mapping between the *code_sect_** variable of the *Act* model and the *code_sect* variable of the *CodeSect* model; it also associates a *code_agenda* variable (*CodeAgenda* model) to each *code_sect* variable (*CodeSect* model).
- Import RespPropos* and relative data: This file fills the data about each *resp_propos variable* (*eurlex*). Each *resp_propos* has a name, nationality, national party and party family. Each *resp_propos_** variable of the *Act* model is linked to the *Person* model (*resp_propos* for *eurlex* and *rapp* for *oeil*). In the *Person* model, the *country* variable is linked to the *Country* model while the *party* variable is linked to the *Party* model. The *party_family* variable is present in the *party_family* association that links one country to one party.
- Import DGProposition* and SiglesDG*: This file associates each *dg_** variable of the *Act* model with a *dg name* and a *sigle (*and a *dg* with a number for old *dg* names, if any*)*. The *DG* model does the mapping with the *name* of the *dg*, while the *DGSigle* model associates each *dg* variable to a *sigle*. The *DGNb* model and the *dg_nb* association are used to do the mapping for dg variables with a number in their name.
- Import ConfigCons: This file is used to do the mapping between *config_cons* variables (*ConfigCons* model) and *code_sect* variables (*CodeSect* model).
- Import AdoptPCAbs and AdoptPCContre: This table links an act (*Act* model) to its *adopt_pc_contre* (a*dopt_pc_contre* association) and *adopt_pc_abs* variables (a*dopt_pc_abs* model)
- Import NationGvtPoliticalComposition: This file contains 3 columns: a start date, and end date and a “gvt compo” column. It represents the main political parties of each European nation with their respective dates of appearance and disappearance. The “gvt compo” column contains the *country* and its list of *parties* and *party families* for a given period of time (defined by the *start_date* and *end_date* variables). These dates variables are stored in the *GvtCompo* model which is linked to the *Party* model via an association and to the C*ountry* model via another association. It is also linked to the *Act* model to show the governments in power at the date of adoption of each act.
- Import opal file (NP variables): This file links an act (*Act* model) to its *np* variables (*NP* model) through its *no_celex* variable. There is a *case_number*, an *act_type* and an *act_type* variable. Each *NP* model is linked to a *country* through the *Country* model.
- Import new Attendance of ministers: This file describes the ministers that attended council meetings. For each council meeting, there is the *country* of each minister, along with their *verbatim* and *status*. Each council meeting is linked to the *Act* model through the act ids (*releve ids*) and / or *no_celex*. Each *verbatim* is stored in the *Verbatim* model. The *min_attend* association links an *act*, a *verbatim* and a *country*. s*tatus* variables link a *verbatim* and a *country* through the *status* association.
- Import party families of Rapporteurs: link each *rapp* variable (*Act* model) with a *party_family* (*party_family* association). The file contains matchings between *party* and *party_family* variables.
- Update Attendance of ministers: Same thing than Import new Attendance of ministers but is used to update already existing ministers' attendances (because they contained wrong data when they were saved). In this case, the link with the *Act* model is made with

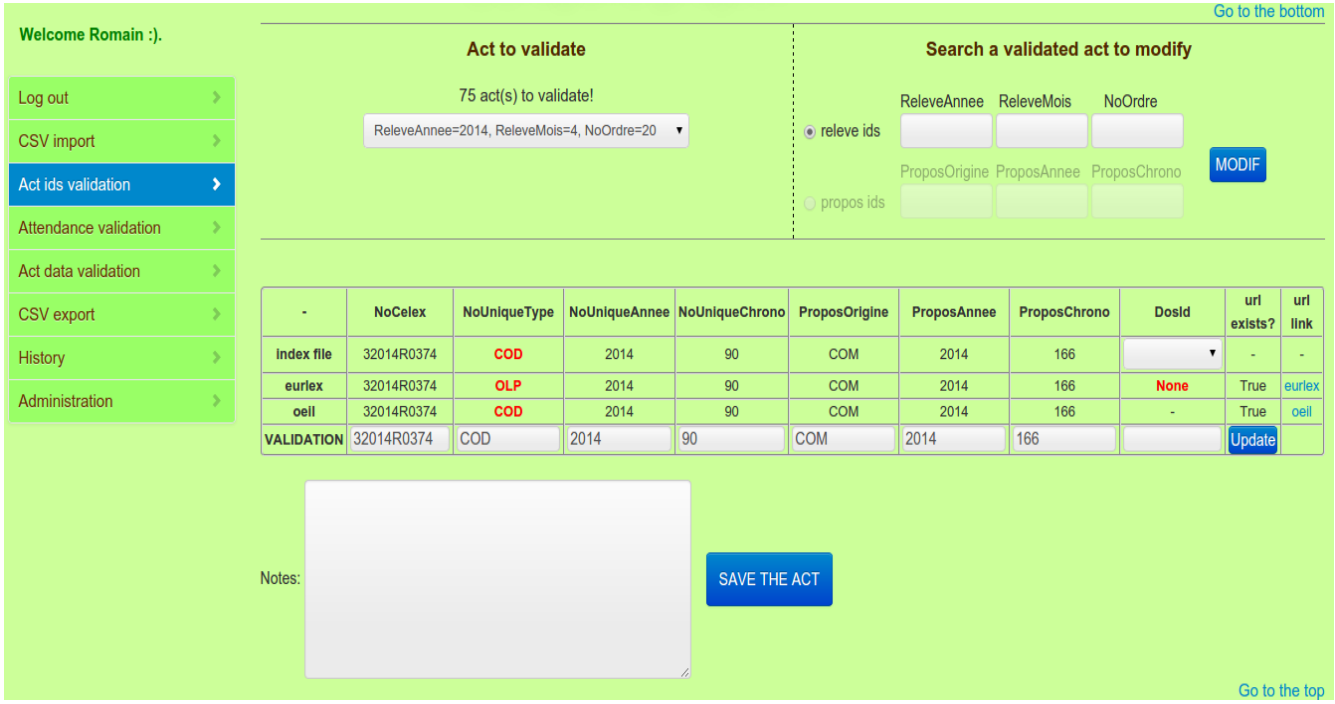
the *no_celex* only.

The web form indicates the variables and their order for each file to import. At the end of the import process, a message is displayed if errors were encountered (in case of already existing record, incorrect data, etc.).

First of all, choose the import to perform in the drop down list. Then click on “Choose file” and select the file to import (it must be a csv file). Finally click on “Import” to launch the import. It can take a while...

After the step Import acts to validate, you can validate the ids of the acts you just imported (see below).

Act ids validation page



You can perform two actions on this page:

- validate ids of an act that has already been imported (see Import page): select an act in the drop down list to validate it.
- modify ids of an act that has already been validated: select *releve ids* to find an act with its *releve_annee*, *releve_mois* and *no_ordre* variables or select *propos ids* to find an act with its *propos_origine*, *propos_annee* and *propos_chrono* variables. Fill the corresponding text boxes and click on the MODIF button to validate the act.

Information related to the selected act is displayed from the database (*ActsIds* table) to an html table. All the fields are displayed in column regarding its source in row. The first source (row) corresponds to the csv file (which can be imported – see previous paragraph), the second row corresponds to the information retrieved on the *eurlex* page and the third one on the *oeil* page. The

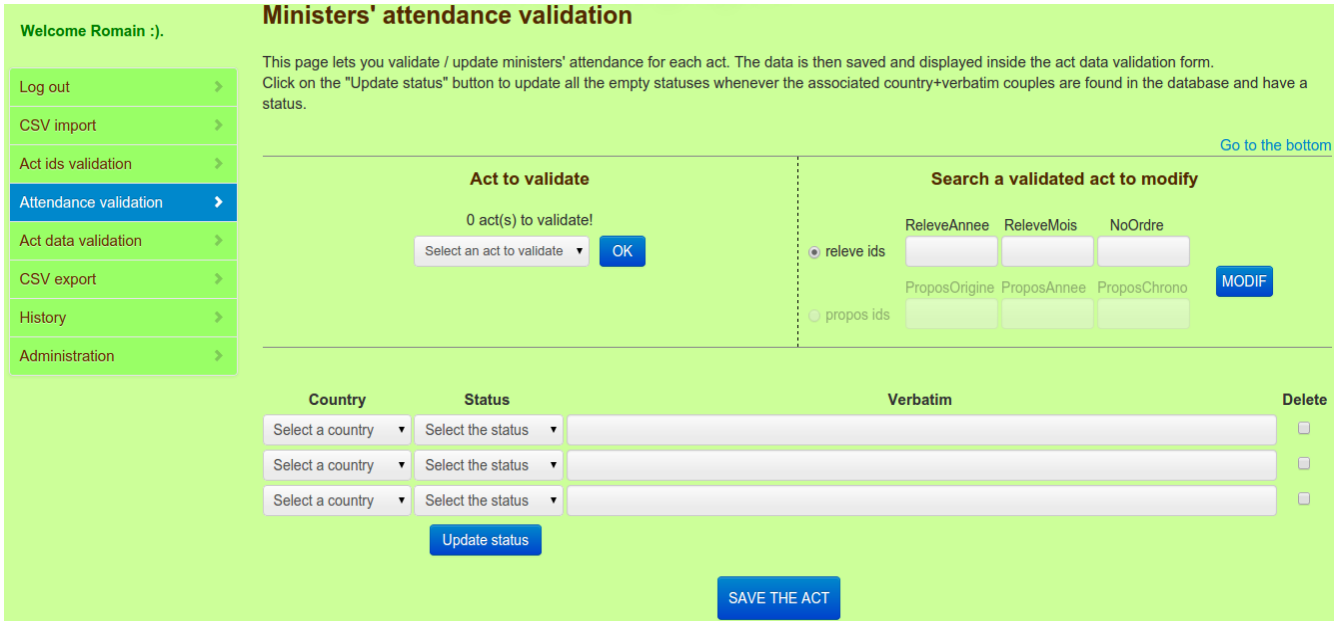
last raw called “Validation” is used to validate and save the true data of the act. By default, it takes the values of the csv file source.
The *dos_id* variable does not exist in the csv file. The drop down list in the DosId column displays all the values that match the *no_celex* of the selected act, from the imported file containing *prelex* unique ids (see Import page). The *dos_id* displayed is the one used by default in the validation row.
When information do not match (between sources), data is displayed in red, which means that there is an error and an expert is required to validate the act. If there is an error, the expert can change the value of the field. When everything is checked, the act can be saved (*ActsIds* table).

You can also update the ids of an act if you think they have been updated on *eurlex or oeil* or if one of these sources was not available at the import of the act or if you made a mistake when you first validated it. Click on the Update button at the bottom right corner of the table and wait for the update of the ids of the act.

Select the act to validate in the drop down list (or to modify by entering the corresponding *releve_ids* in the text boxes) and check the corresponding information in the table below. Modify fields if needed in the row “Validation”. You can check or / and add notes regarding the act. Finally, click on “Save” to validate and save the act.

After this step, you can validate the ministers' attendance of any act you have already imported (see below).

Ministers' attendance validation page



This page displays a list of countries and minister's status and verbatim that attended the council meeting concerning an act to validate. This is possible thanks to the extraction of a pdf whose url is included in csv files that were imported previously (see Error: Reference source not found to learn how to retrieve the data of those pdf files).

You can perform two actions on this page:

- validate the ministers' attendance of an act that has already been imported (see Import page): select an act in the drop down list and click on the OK button to validate it.
- modify the ministers' attendance of an act whose ministers' attendance have already been validated: select *releve ids* to find an act with its *releve_annee*, *releve_mois* and *no_ordre* variables or select *propos ids* to find an act with its *propos_origine*, *propos_annee* and *propos_chrono* variables. Fill the corresponding text boxes and click on the MODIF button to validate the act.

modify ids of an act that has already been validated:

Information related to the selected act is displayed in 3 columns: country, status and verbatim. For each minister attending the council meeting, we have retrieved his country, verbatim (position title) and status (code created to facilitate the statistical use of the verbatim – see codebook).

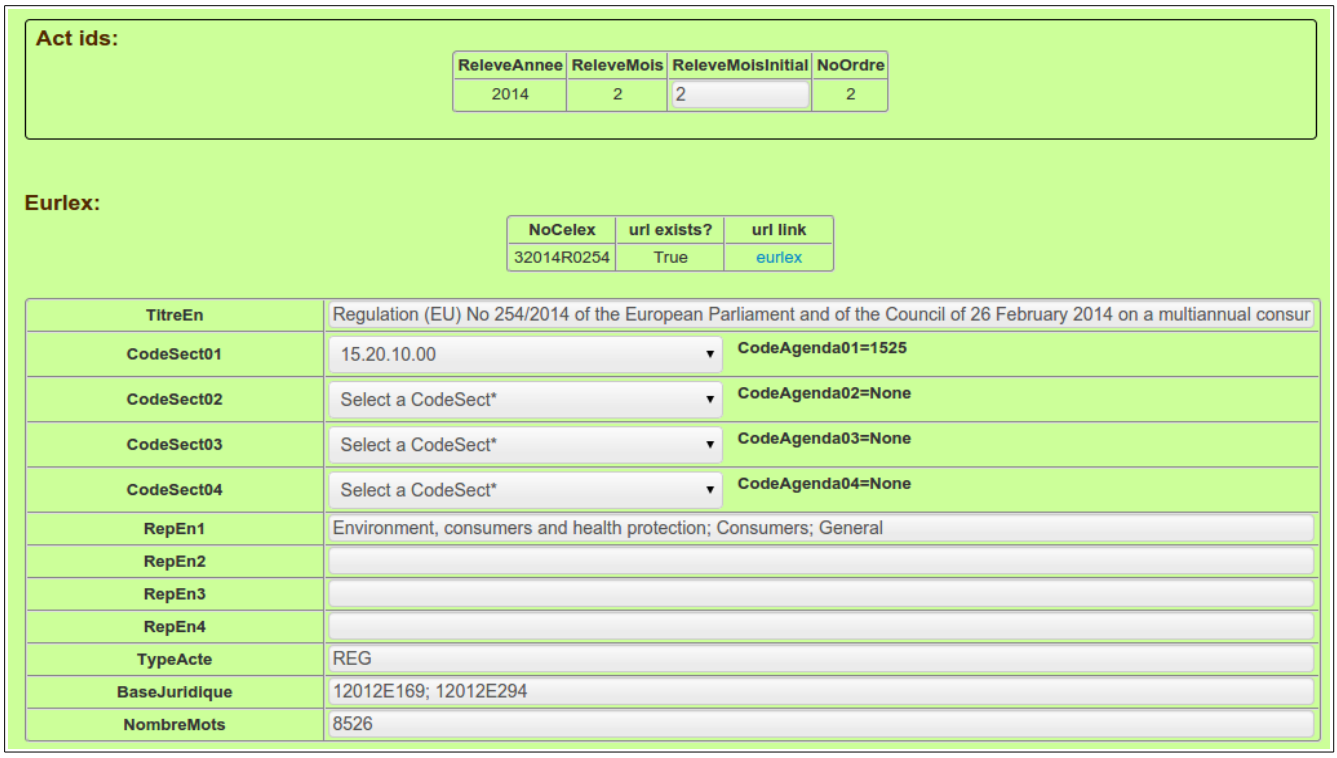
You can also update the status by clicking on the Update status button. This will search matches of couples (verbatim+country) and will try to associate the corresponding status to all the same couples (verbatim+country) with no status.

There is the possibility to delete one row by ticking the checkbox Delete in the last column (and clicking on the SAVE button) and adding extra rows using the 3 blank lines at the bottom.

Select the act to validate in the drop down list (or to modify by entering the corresponding *releve ids or propos ids* in the textboxes) and check the corresponding information in the table below. Modify fields if needed (choose another country or status in the corresponding drop down lists or change the verbatim). Finally, click on “Save” to validate and save the act.

After this step, you can retrieve information of the acts you just validated for the statistical analysis (see below).

Act data validation page



You can perform two actions on this page:

- validate data of an act that has already been imported (see Import page): select an act in the drop down list to validate it.
- modify data of an act that has already been validated: select *releve ids* to find an act with its *releve_annee*, *releve_mois* and *no_ordre* variables or select *propos ids* to find an act with its *propos_origine*, *propos_annee* and *propos_chrono* variables. Fill the corresponding text boxes and click on the MODIF button to validate the act.

Acts data can be retrieved and then validated on this page. You can select acts for which ids have been validated only. Data related to one act is displayed from the database (*Act* table) to html tables. For each source (*eurlex and oeil*), a first table recalls the ids and the url while a second table displays the corresponding retrieved data. The *NationGvtPoliticalComposition* variable, at the end of the *eurlex* table, shows the composition of each national government when the act was adopted. The two other tables show the *ministers' attendance* data (country and status only) and the *opal* data with one row per country.

All the retrieved fields are displayed in row. The first column corresponds to labels and the second one corresponds to retrieved data. If there is an error, the expert can change the value of any field. When everything is checked, the act is ready to be saved (*Act* table).

Select the act to validate in the drop down list (or to modify by entering the corresponding *releve ids or propos ids* in the textboxes) and check the corresponding information in the tables below. Modify fields if needed in the second column of the *eurlex or oeil* table. *NationGvtPoliticalComposition*, *ministers' attendance* and *opal* data cannot be modified, but *ministers' attendance* data can actually be modified in another form (see Ministers' attendance validation page). You can check or / and add notes regarding the act. Finally, click on “Save” to

validate and save the act.

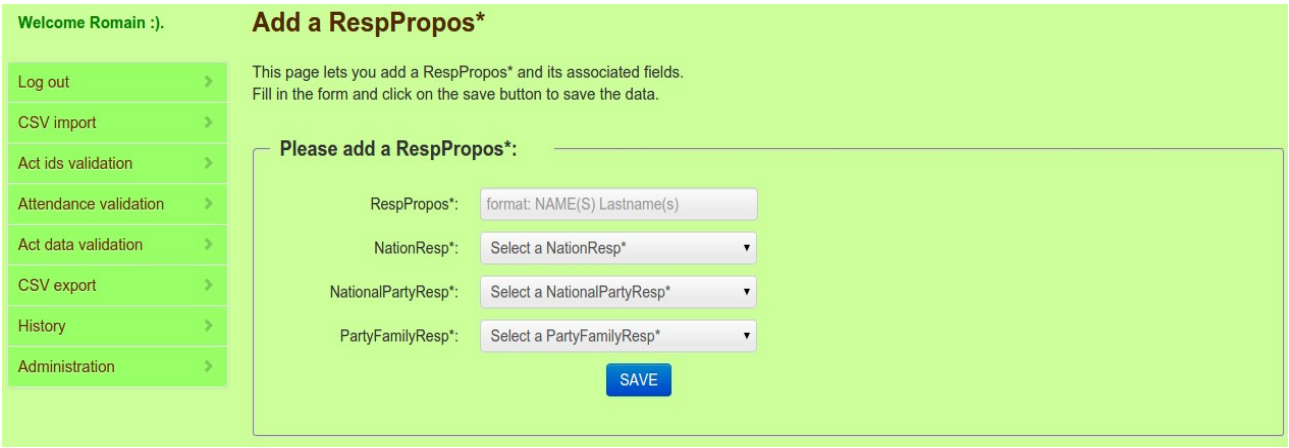
In fact there is one more action you can perform on this page...

Database management page

On the Act data validation page, after selecting an act, you can Add a new *rapp*, *dg*, or *resp* variable:

- in front of each *rapp* variable, you can click on “Add a new Rapporteur*” to add a new *rapp* variable (*name*, *country* and *party*) .
- in front of each *dg* variable, you can click on “Add a new DGProposition*” to add a new *dg* variable (*dg* and *dg_sigle*) .
- in front of each *resp* variable, you can click on “Add a new RespPropos*” to add a new *resp* variable (*name*, *country*, *party* and *party_family*) .

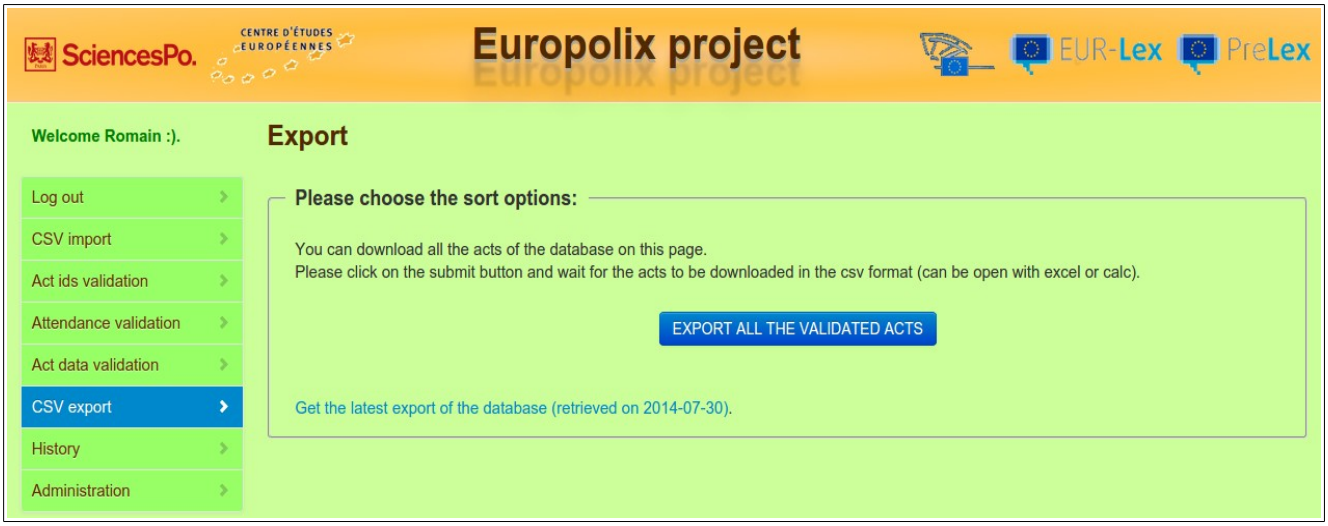
The screenshot above can be seen when JavaScript is activated. If JavaScript is not activated, it's a completely new page that you see:



This interface is called database management because a few functionalities could be added if needed, such as the possibility to update or delete an existing record as well as the possibility to manage other models.

After this step, you can export the acts you already imported and validated in a csv file (see below).

Export page



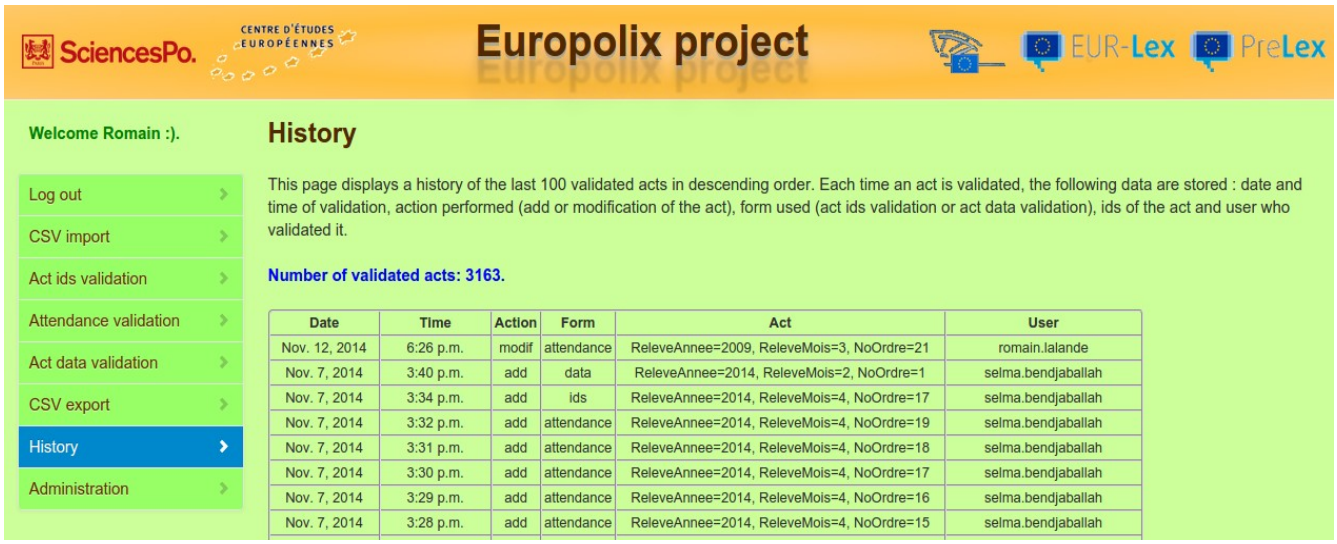
This is the export page. Once acts have been imported, ids validated and data retrieved, you can choose this tool to export them under the csv format.
This action exports acts saved and validated only, from the database, in two steps: a csv file is created on the server and contains the data of the database, then it is sent and downloaded by the web browser.

To download the validated file, click on the Export button to start the download.

You can also get the latest export of the database by clicking on the link at the bottom of the form. This file is exported on the server automatically, every Sunday. Choose this option if you have problems exporting the acts.

After this step, acts are ready for the statistical analysis :).

History page



This page shows a history of the last 100 actions performed, useful for validators to help them remember their most recent tasks.

There are 6 columns:

- the *date* of the performed action
- the *time* of the performed action
- the *action* itself: *add* when an act is added (selection of an act from the drop down list) or *modif* when an act is modified (after entering its *releve ids* or *propos ids* in the textboxes and clicking on the Modif button)
- the *form* used: *ids* for the Act ids validation form, *data* for the Act data validation form and *attendance* for the Ministers' attendance validation form
- the *act* column displays the *releve ids* of the act
- the *user* column displays the user who has performed the action

Fields to retrieve

This section gives some information about the retrieved fields from the index files, *eurlex url*, *oeil url* and the external tables.

These fields can be found in the *models.py* files of the *act_ids*, *act*, *attendance* and *import_app* applications. Some technical explanations are given in the source code, but the explanation of each field, in natural language, is given in the **codebook**.

In the future, fields should be renamed in English. To this end, a file for ids and a file for all the other variables have been created to do the match between variable names used by the program and variable names that are displayed / exported. These files are located at acts_ids/var_name_ids.py for variables concerning ids of the acts and act/var_name_data.py for variables concerning data of the acts. They use dictionaries, with names used by the program as keys and names for the display as values. See the source code for more details. These files should be used each time you want to display or export a variable name.

Let us see now how these fields are stored in the database.

Database

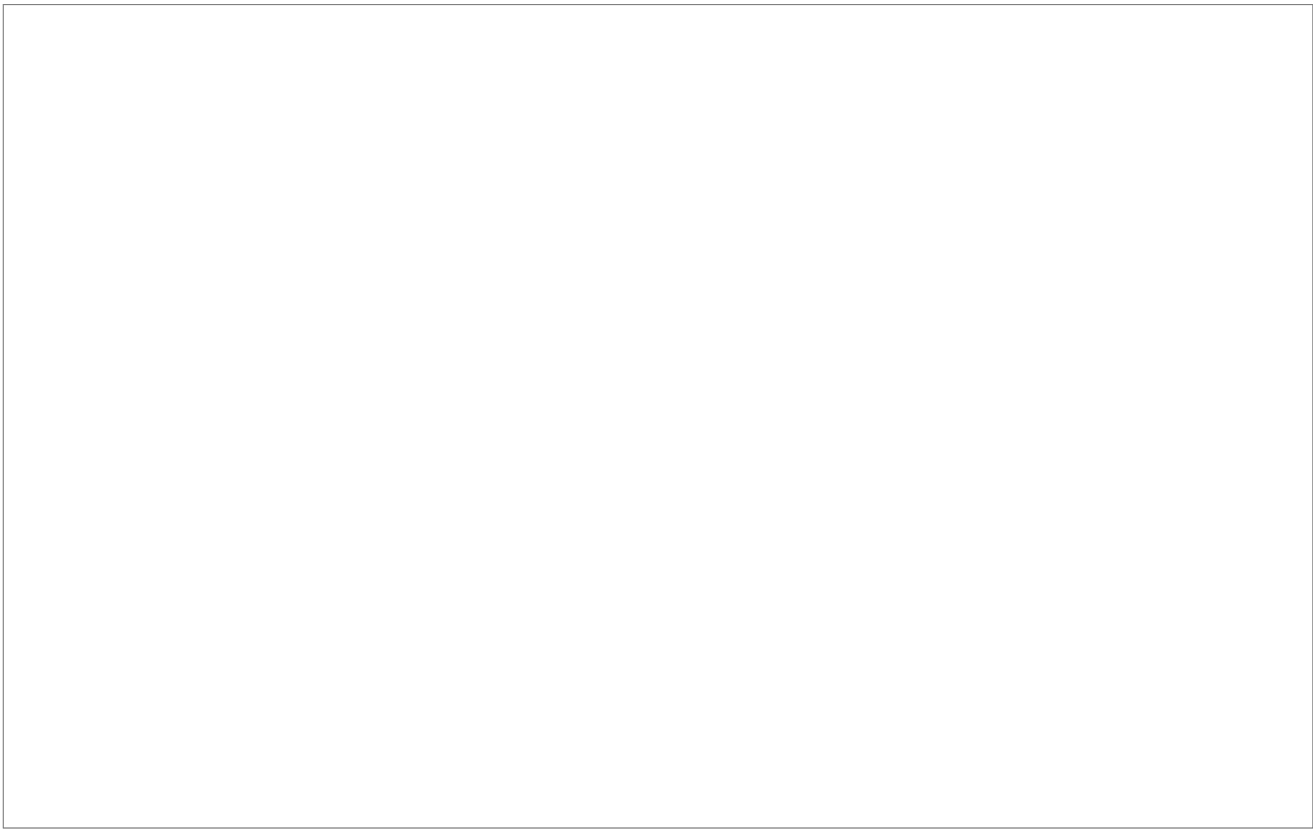
This section describes the database and relations between *tables* (*models* in *Django* vocabulary).

Naming conventions

To create a name for a model, *Django* concatenates the application name, an underscore "_" and the model name. Everything is lowercased. For example, the *DG* model of the *act* application is named *act_dg*.
For an association that links two tables:

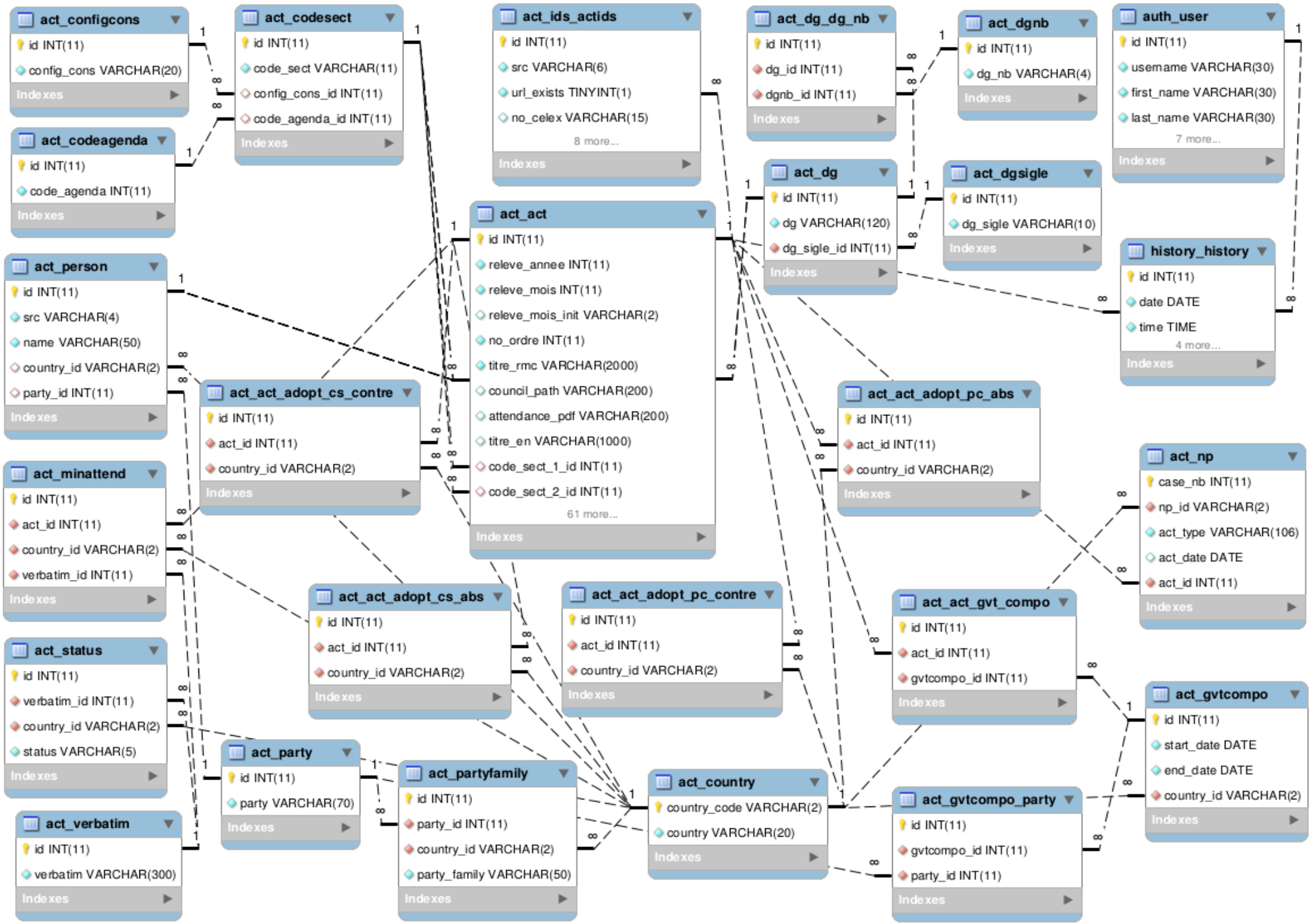
- the association is manually created : the model name follows the convention above. For exemple, the *PartyFamily* association of the *act* application is named *act_partyfamily*.
- The association is created by *Django*: in such a case, there is a *manytomany* variable created in one particular *model*. We then use the name of the model and the name of the variable, joined by an underscore, to generate the name of the association. Let's take the example of the *dg_nb* variable of the *DG* model of the *act* application (used to join the *DG* and *DGNb* models). The name of the association is then: *act_dg_dg_nb*.

Import tables or tables used for the web application



These tables all come from the *import_app* application. They are only used for the import of external tables (see Import page) and for populating all the tables that will serve the statistical analysis. They are temporary tables and therefore have no physical relation with any other table.

- *import_app_importdosid:* This model is used to populate the *dos_id* variable of the *ActIds* model. The link is made with the *no_celex* variable.
- *import_app_importrapppartyfamily:* This is used to populate the *party_family* variable of the *PartyFamily* model so as to link every *party* of rapporteurs (*Party* model) to a *party family*.
- *import_app_importnp:* This model concerns the *opal* project. The variables are used to populate the variables of the *NP* model; the link is made with the *no_celex* variable.
- *import_app_importminattend:* This model is used to populate the *MinAttend, Verbatim and Status* tables. The link is made via the *no_celex* variable.
- *import_app_importadoptpc:* This model is used to populate the *act_act_adopt_pc_abs association* (with the *adopt_pc_abs* variable) and *act_act_adopt_pc_contre* association (with the *adopt_pc_contre* variable). The link is made via the *no_celex* variable.
- *import_app_csvupload:* This model is used for the web application only. It shows the path of each imported file and facilitates import of files with *Django*.


act_configcons
id INT(11)
config_cons VARCHAR(20)
Indexes
act_codesect
id INT(11)
code_sect VARCHAR(11)
config_cons_id INT(11)
code_agenda_id INT(11)
Indexes
act_ids_actids
id INT(11)
src VARCHAR(6)
url_exists TINYINT(1)
no_celex VARCHAR(15)
8 more...
Indexes
act_dg_dg_nb
id INT(11)
dg_id INT(11)
dgnb_id INT(11)
Indexes
act_dgnb
id INT(11)
dg_nb VARCHAR(4)
Indexes
auth_user
id INT(11)
username VARCHAR(30)
first_name VARCHAR(30)
last_name VARCHAR(30)
7 more...
Indexes
act_codeagenda
id INT(11)
code_agenda INT(11)
Indexes
act_act
id INT(11)
releve_annee INT(11)
releve_mois INT(11)
releve_mois_init VARCHAR(2)
no_ordre INT(11)
titre_rmc VARCHAR(2000)
council_path VARCHAR(200)
attendance_pdf VARCHAR(200)
titre_en VARCHAR(1000)
code_sect_1_id INT(11)
code_sect_2_id INT(11)
61 more...
Indexes
act_dg
id INT(11)
dg VARCHAR(120)
dg_sigle_id INT(11)
Indexes
act_dgsigle
id INT(11)
dg_sigle VARCHAR(10)
Indexes
history_history
id INT(11)
date DATE
time TIME
4 more...
Indexes
act_person
id INT(11)
src VARCHAR(4)
name VARCHAR(50)
country_id VARCHAR(2)
party_id INT(11)
Indexes
act_act_adopt_cs_contre
id INT(11)
act_id INT(11)
country_id VARCHAR(2)
Indexes
act_act_adopt_pc_abs
id INT(11)
act_id INT(11)
country_id VARCHAR(2)
Indexes
act_np
case_nb INT(11)
np_id VARCHAR(2)
act_type VARCHAR(106)
act_date DATE
act_id INT(11)
Indexes
act_minattend
id INT(11)
act_id INT(11)
country_id VARCHAR(2)
verbatim_id INT(11)
Indexes
act_act_adopt_cs_abs
id INT(11)
act_id INT(11)
country_id VARCHAR(2)
Indexes
act_act_adopt_pc_contre
id INT(11)
act_id INT(11)
country_id VARCHAR(2)
Indexes
act_act_gvt_compo
id INT(11)
act_id INT(11)
gvtcompo_id INT(11)
Indexes
act_status
id INT(11)
verbatim_id INT(11)
country_id VARCHAR(2)
status VARCHAR(5)
Indexes
act_party
id INT(11)
party VARCHAR(70)
Indexes
act_partyfamily
id INT(11)
party_id INT(11)
country_id VARCHAR(2)
party_family VARCHAR(50)
Indexes
act_country
country_code VARCHAR(2)
country VARCHAR(20)
Indexes
act_gvtcompo_party
id INT(11)
gvtcompo_id INT(11)
party_id INT(11)
Indexes
act_gvtcompo
id INT(11)
start_date DATE
end_date DATE
country_id VARCHAR(2)
Indexes
act_verbatim
id INT(11)
verbatim VARCHAR(300)
Indexes

Tables containing act data and used for the statistical analysis

The main table is *act_act*. All the other tables are directly or indirectly linked to it. It contains data for each *act*. Each *act* is identified by its *releve ids* (*releve_annee*, *releve_mois* and *no_ordre* variables). The table also contains variables taken from the monthly council summary, from the *eurlex* and *oeil* and websites and from external tables (see Import page). You can validate / see all the variables of this table in the Act data validation form (see Act data validation page).

The second main table is *act_ids_actids*. It contains the ids of each act retrieved on every source (identified by the *src* variable): the ids for *eurlex* (*no_celex*) or the ones for *oeil* (*no_unique_type*, *no_unique_annee*, et *no_unique_chrono*) or the ones for *prelex* (*propos_origine*, *propos_annee* and *propos_chrono* or *dos_id*) or the validated ids (true ids), coming from the *index* files, that were filled or validated in the Act Ids validation form (see Act ids validation page). Each act_act instance has exactly 4 *act_ids_actids* instances, one for each source. Once the ids of an act are validated, only its ids from the *index* source are used by the application (true ids).

The *act_codesect, act_configcons* and *act_codeagenda* models are used to populate the *code_sect_1_id*, *code_sect_2_id*, *code_sect_3_id* and *code_sect_4_id* variables of the *act_act* model. They are also used to display the 4 different *code_sect* with their associated *code_agenda* variables as well as the *config_cons* variable, *for a given act*, in the Act data validation form (see Act data validation page). Each *code_sect_*_id* of the *act_act* model has one *act_codesect* instance and each *act_codesect* instance has one *act_configcons* and one *act_codeagenda* instance.

The *act_dg, act_dgsigle, act_dgnb* and *act_dg_dg_nb* models are used to populate the dg*_1_id*, dg_2_*id* and *dg_3_id* variables of the *act_act* model. They are used to display the two *dg* with their *sigle, for a given act,* in the Act data validation form (see Act data validation page). Each dg_*_id of the *act_act* model has one *act_dg* instance and each *act_dg* instance has one *act_dgsigle* instance. Sometimes dg appear with numbers instead of names. They are listed in the table *act_dgnb*. To get their real names, there is an association *act_dg_dg_nb* that maps *dg* with names (*act_dg*) and *dg* with numbers (*act_dgnb*). One *act_dg* instance can have many *act_dg_dg_nb* instances and one *act_dg_dg_nb* instance can have many *act_dg* instances.

The *act_country* model gives a list of *country codes* and *countries* of the European Union and is linked to many models.
The *act_party* model gives a list of *political parties* of the European Union and is linked to many models.

The *act_partyfamily* model gives a list of *political party families* of the European Union and links a *party* to a *country*. One *party* and one *country* identify a unique *party family*.

The *act_person* model is used to populate the *rapp_1_id*, *rapp_2_id*, *rapp_3_id*, *rapp_4_id*, *rapp_5_id*, *resp_1_id*, *resp_2_id* and *resp_3_id* variables of the *act_act* model. A *person* is either a *rapporteur* (*oeil*) or a *responsible* (*eurlex*), which is identified by the *src* variable. The model aims to display the 5 different *rapporteurs* / 3 different *responsibles* along with their *country*, *party* and *party family, for a given act,* in the Act data validation form (see Act data validation page). Each person has a name, a country, a party and a party family.

The *act_np* model is used to display the opal variables of an act (*act_act* model) in the Act data validation form (see Act data validation page). Each *act* can be associated to many *act_np* instances and each *act_np* object is associated to one *act* and one *country*.

The *act_act_adopt_pc_abs*, *act_act_adopt_pc_contre*, *act_act_adopt_cs_abs* and *act_act_adopt_cs_contre* models are used to populate the *adopt_pc_abs*, *adopt_pc_contre*, *adopt_cs_abs* and *adopt_cs_contre* variables of the *act_act* model. They are displayed in the Act data validation form (see Act data validation page). Each of these variables links an *act* with a *country*. One *act* and one *country* can have many *adopt_pc_abs*, *adopt_pc_contre*, *adopt_cs_abs* or *adopt_cs_contre* variables.

The *act_gvtcompo* model indicates the composition of the European governments for a given period defined by a *start date* and an *end date*. Each period is associated to a *country* and a list of *parties* and *party families*. One *period* is associated to one or many *acts* (act_act model) and one *act* is associated to one or many *periods*. Those variables are displayed in the Act data validation form (see Act data validation page).

The *act_minattend* model indicates the *attendance of ministers* for every European council. Each minister who attends a council has a *country*, a position identified by a *verbatim* (*act_verbatim* model) and a *status* (*act_status* model), that is, a code for the verbatim. Every *act* (*act_act* model) has many ministers' attendances and so many associated *verbatims*, *status* and *countries*. A *country* and a *verbatim* identify a unique *status*. For a given *act*, minister's attendances can be checked and validated, in the Attendance validation form (see Ministers' attendance validation page); the same data are also displayed in the Act data validation form (see Act data validation page).

The *history_history* model shows, for each validated form, the *user* who validated it (*auth_user* model), the *date* and *time* of validation, the *action* performed (add or modification) and the *form* used (act data, act ids or ministers' attendance). Each *act* and each *user* can be seen in many *history_history* instances but each *history_history* instance is associated to one *act* and one *user* only. A history of the last 100 actions performed can be seen in the History page.

So far we have seen what has been created for the project. Let us now see which tools were used.

Tools used

This section describes the tools used for this project.

- operating system: *linux* (lubuntu 14.04)

- programming language: *python* (version 2.7.6). This is the language used for the project. Easy to learn, short to write (many built-in functions) and quite fast.

- framework: D*jango* (version 1.4.2) [English documentation: https://docs.Djangoproject.com/en/1.4/ - French documentation: http://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-Django]. This python framework is used to quickly develop applications on the Internet. It tends to differentiate design and content. Its goal is to facilitate a developer's life and maintenance. Each *Django* application is composed of 4 layers:
  - Model: it is the database layer. It creates a connection with the database. No need to write sql queries, D*jango* takes care of that! In a model, precise the tables and fields you want to use (for a select, insert, update or delete operation).
  - Form: it is the web form layer. Declare the fields to appear in your web form and describe a custom validation if needed.
  - View: it is the content layer. You describe / create what you want to display and use the model and form previously created.
  - Template: it is the display layer. A template is mainly html code with *Django* variables and expresses how the html page is rendered.
  - An other kind of file is essential for *Django* applications: the url file. It tells *Django* what view and parameters need to be called for a given url.

- back-end database: *mysql*

- *MySQL-python* version 1.2.5 is used to connect the python program to a *MySQL* database, which is crucial to the project as it is using a *MySQL* backend database.

- html extraction: *BeautifulSoup* (version 4.3.2) [English documentation: http://www.crummy.com/software/BeautifulSoup/bs4/doc/], regex and python string manipulation tools. These tools are used to extract information on acts from the *eurlex and oeil* websites.

- *pdfminer*, version 2011-05-15, used to extract some pdfs for the ministers' attendance

- dbms: *mysql Workbench* (version 5.2) [optional]

Django files

This section gives a general explanation and purpose of all the important folders / files of the project. For more details, please consult the files directly: they are all commented.

How is organized the website

Django is organized in applications, each of them containing a specific functionality of the website. At the root of the server, *act*, *act_ids*, *attendance*, db_mgmt, *europolix*, history, *export*, *import_app and* login applications can be found. They are described in the next section.

At the root of the server, you can also find other folders:

- *common*: common files used by several applications (see Common module).
- *media*: contains csv files either imported (*import* subfolder) or exported (*export* subfolder).
- *static*: contains static files such as *css* and *js* files:
  - *admin*: *css*, *js* and *images* files for the administration tool.
  - *bootstrap*: *bootstrap* files used by the *css* and *js* files of the project.
  - *europolix*: *css*, *js* and images files of the project.
  - *jquery_gentleSelect*: jquery plugin used to display the *adopt_cs* and *adpot_pc* dropdown lists (Act data validation form).
  - *jquery.min.js*: *jQuery* file, used by js files of the project.
  - *jQueryRotate.min.js*: *jQuery* file to rotate images, used on the image in front of the *adopt_conseil* variable (Act data validation form).
- *templates*: this folder contains the html rendering of each page. At the root, *index.html* is the template of the homepage while *base.html* contains common elements to each page and *menu.html* contains the left menu. For details about application pages, see next section.

The root of the server also contains files:

- *europolix.log[.*]*: log files of the applications, used to debug the application on the remote server.
- *manage.py*: this very important file in any *Django* application allows to run commands, to start the local server, to synchronize the database, to run some tests, etc.
- *README.md*: this file explains how to deploy and run the application on a server.
- *requirements.txt*: this file contains a list of packages to install to have this project working.

The applications

Let us see now each application in details. They are physically organized as folders located at the server root.

europolix

europolix is the first and main folder. It essentially contains the homepage and allows to activate the administration part. Here is a short explanation of each important file of the folder:

- ***settings.py*: this file is very important since it contains all the specificities of your project. It contains the database access (including the password), the applications to load, path for media and static files, path for templates and more...**
- *urls.py*: contains the url to each application of the website. It also activates the *Django* administration tool (automatically created by *Django*!).
- *views.py*: contains the *reload_menu* view called to reload the left menu when a user logs in or logs out.
- the file *index.html* within the *templates* folder is used to render the homepage and is called from the *urls.py* file.

login

This is the *login* folder, linked to the Login page. It allows the user to authenticate and access all the functionalities of the website. Here is a short explanation of each important file of the folder:

- *views.py*: contains the *login_view* view called to render the connection page. Allows the user to enter his ids and checks them.
- *The index.html* file within the folder *templates/login* is used to render the login page and is called by the *login_view* view.

import_app

This is the *import_app* folder, linked to the Import page. It allows the user to import various csv files containing acts to validate and external tables linked to these acts. Here is a short explanation of each important file of the folder:

- *urls.py*: contains all the possible urls of the application.
- *models.py*: this file contains the database model of the application. It contains the *ImportDosId* model (imported *dos_id*), *ImportNP* (imported opal variables), *ImportAdoptPC* (imported *adopt_pc_contre* and *adopt_pc_abs* variables), *ImportMinAttend* (imported ministers' attendance variables: *verbatim*, *country*, *status*), *ImportRappPartyFamily* (makes the correspondance between *party* and *party_family* variables for a *rapporteur*) and the *CSVUpload* model which contains the name of the imported files. This last model is not really useful but is the easiest way to import files with *Django*.
- *forms.py*: contains the *CSVUploadForm* form which uses the *CSVUpload* model. It defines the form used by the view and displayed by the template.
- *get_ids_eurlex.py and get_ids_oeil.py* are used to retrieve ids from *eurlex* and *oeil* respectively. Collected information is stored in a dictionary.
- *get_ids.py* is the main file called by the view. It builds the urls, checks their availability and call the files above to retrieve ids from the *eurlex and oeil* websites.
- *views.py*: contains the *import_view* view called to render the import page and calls the following functions:
  - *get_data_act* and *get_save_act_ids* to “Import acts to validate”

- *get_data_dos_id* to “Import DosId”
- *get_data_adopt_pc and save_adopt_cs_pc* to “Import AdoptPCAbs and AdoptPCContre”
- *get_data_gvt_compo* to “Import NationGvtPoliticalComposition”
- *get_data_np* to “Import opal file (NP variables)”
- *get_data_min_attend_insert* to “Import new Attendance of ministers”
- *get_data_rapp_party_family* to “Import party families of Rapporteurs”
- *get_data_min_attend_update* to “Update Attendance of ministers”
- *import_2_tables to “Import CodeAgenda*”, “Import ConfigCons”,* “Import RespPropos* and relative data”, *“Import DGProposition* and SiglesDG*”*

All the imports that do not use the *import_2_tables* function (generic function to import data in two different tables of the database, that is, tables linked by a 1-n relation) use the *import_table* function instead (generic function to import data in one table only).

- the *index.html* file *within* the folder *templates/import* is used to render the import page and is called by the *import_view* view.

act_ids

This is the *acts_ids* folder, linked to the Act ids validation page. It allows an expert to validate ids on an act saved in the database but not checked yet. Experts can modify any field if needed and then save the act. Here is a short explanation of each important file of the folder:

- *urls.py*: contains all the possible urls of the application.
- *admin.py*: the *ActAdmin* class gives the Administration tool access to this application (see Administration page).
- *models.py*: the *ActIds* model is the first important table of the project and contains ids of the imported acts for 3 sources: *index* (true validated ids), *eurlex* (ids retrieved on the *eurlex* page) and *oeil* (ids retrieved on the *oeil* page).
- *forms.py*: contains the *ActIdsForm* form which uses the *ActIds* model. It defines a customized validation of the fields and describes the form used by the view and displayed by the template. It also contains the *Add* form to display a drop down list of acts to validate and the *Modif* form (that inherits the *AbstractModif* form, see Common module) to modify an act already validated.
- *views.py*: render the Act ids validation page and calls the following functions:
  - *act_ids*: this is the view that actually renders the Act ids validation page. It processes the *add* or *modification* of an act (*Add*/*Modif* forms), *update* of an act (click on *update button*) and *save* an act (click on *save button*). When the ids of an act are to be displayed, it checks that the data between sources (index/csv file, *eurlex and oeil*) match. If not, the data must be highlighted (in red) so experts can easily see the problem.
  - *check_equality_fields*: this function is used by the view to check whether all the sources have the same value for one particular variable. It is used to highlight in red variables with different values.

- *reset_ids_form*: this view is called with Ajax to empty the *ActIdsForm* form (when an act has been saved or when the *Add* / *Modif* form is not valid).

- *var_name_ids.py*: does the match between variables name used for the application and variables names displayed or exported. It contains ids variables only.
- the *index.html* file *within* the folder *templates/act_ids* is used to render the Act ids validation page and is called by the *act_ids* view. This file calls the *form.html* file that contains the form of the Act ids. Also, the *add_modif.html* file is used to display the Add and Modif forms while the *notes_save.html* file is used to display the note field and to save the form.

attendance

This is the *attendance* folder, linked to the Ministers' attendance validation page. It allows an expert to validate minister's attendances on an act saved in the database but not checked yet. Experts can modify any field if needed and then save the act. Here is a short explanation of each important file of the folder:

- *urls.py*: contains all the possible urls of the application.
- *forms.py*: contains the *ImportMinAttendForm* form which uses the *ImportMinAttend* model (from the *import_app* application). It defines a customized validation of the fields and describes the form used by the view and displayed by the template. It also contains the *Add* form to display a drop down list of acts to validate and the *Modif* form to modify an act already validated.
- *views.py*: render the Ministers' attendance validation page and calls the following functions:
  - *MinAttendUpdate*: this is the view that actually renders the Ministers' attendance validation page. It processes the *add* or *modification* of an act (*Add*/*Modif* forms), *update* all the empty statuses (click on *update button*) and *save* an act (click on *save button*). It contains 3 main functions: *post* to deal with post data, *form_valid* when all the forms are valid and ready to be saved and *form_invalid* when at least one value from one form is not valid and need to be changed in order to save the attendances.
  - *reset_form_attendance*: this view is called with Ajax to empty all the *ImportMinAttendForm* forms (when an act has been saved or when the *Add* / *Modif* form is not valid).
- the *index.html* file *within* the folder *templates/attendance* is used to render the Ministers' attendance validation page and is called by the *MinAttendUpdate* view. This file calls the *form.html* file that contains the forms of the Ministers' attendance. Also, the *add_modif.html* file is used to display the Add and Modif forms while the *notes_save.html* file is used to display the note field and to save the form.

act

This is the *act* folder, linked to the Act data validation page. It allows an expert to validate information of an act for which ids have already been validated. Experts can modify any field if needed and then save the act. Here is a short explanation of each important file of the folder:

- *urls.py*: contains all the possible urls of the application.
- *admin.py*: the ActAdmin, DGSigleAdmin, DGAdmin and *GvtCompoAdmin* classes give the

Administration tool access to this application (see Administration page).

- *models.py*: this file contains the main models of the project:
  - *Act* is the main table of the project and contains or is linked to all the data of an act. It stores data of the *index file* as well as data from *eurlex and oeil*. All the other tables are directly or indirectly linked to the Act table (except temporary ones).
  - *Country*, *Party*, *PartyFamily* and *Person* are used, among others, to get the rapp_1-*5* and the resp_1-*3* and their associated variables *country*, *party*, *party_family*.
  - *GvtCompo* gives the *gvt_compo* data (*country*, *party* and *party_family*).
  - *DGSigle*, *DGNb* and *DG* are used for the *dg_1-2* and their associated *sigle*.
  - *ConfigCons*, *CodeAgenda* and *CodeSect* give the code_sect_1-4 and their associated variables *code_agenda* and *config_cons*.
  - *NP* is used for the *opal* variables.
  - *Verbatim*, *Status* and *MinAttend* store the Ministers' attendance data (*country*, *verbatim* and *status* variables).
- *forms.py*: contains the *ActForm* form which uses the *Act* model. It defines a customized validation of the fields and describes the form used by the view and displayed by the template. It also contains the *Add* form to display a drop down list of acts to validate and the *Modif* form (that inherits the *AbstractModif* form, see Common module) to modify an act already validated.
- *get_data_eurlex.py* is used to retrieve information from *eurlex*. Collected information is stored in a dictionary. Here are the main functions:
  - the main function is called *get_data_eurlex* and is used to get all the variables from *eurlex*. It calls all the other functions.
  - *get_titre_en* to get the *titre_en* variable.
  - *get_code_sect* for the *code_sect_1-4* variables.
  - *save_code_agenda* to save the *code_agenda* variables knowing the *code_sect* variables.
  - *get_rep_en* to get the *rep_en_1-4* variables.
  - *get_type_acte* for the *type_acte* variable.
  - *get_base_j* to get the *base_j* variable.
  - *get_date_doc* for the *date_doc* variable.
  - *get_nb_mots* to get the *nb_mots* variable.
  - *get_adopt_propos_origine* to get the *adopt_propos_origine* variable.
  - *get_com_proc* for the *com_proc* variable.
  - *get_dg_1* to get the *dg_1* variable.
  - *get_dg_2_3 for* the *dg_2 and dg_3* variables.
  - *get_resp_1* to get the *resp_1* variable.

- *get_resp_2_3 for* the resp*_2 and resp_3* variables.
- *get_transm_council* for the *transm_council* variable.
- *get_nb_point_b* to get the *nb_point_b* variable.
- *get_date_cons_b* for the *date_cons_b* variable.
- *get_cons_b* to get *cons_b* variable.
- *get_split_propos* to get the *split_propos* variable.
- *get_adopt_conseil* for the *adopt_conseil* variable.
- *get_nb_point_a* to get the *nb_point_a* variable.
- *get_date_cons_a* for the *date_cons_a* variable.
- *get_cons_a* for the *council_a* variable.
- *save_config_cons* to get the *config_cons* variable.
- *get_chgt_base_j* for the *chgt_base_j* variable.
- *get_duration_fields* for the *duree_adopt_trans*, *duree_proc_depuis_prop_com*, *duree_proc_depuis_trans_cons*, *duree_tot_depuis_prop_com* and d*uree_tot_depuis_trans_cons* variables.
- *get_vote_public* to get the *vote_public* variable.
- *save_get_adopt_pc* for the *adopt_pc_contre* and *adopt_pc_abs* variables.

➢ *get_data_oeil.py* is used to retrieve information from *oeil* Collected information is stored in a dictionary. Here are the main functions:

- the main function is called *get_data_oeil* and is used to get all the variables from *oeil*. It calls all the other functions.
- *get_nb_lectures* to get the *nb_lectures* variable.
- *get_commission* for the *commission* variable.
- *get_com_amdt_tabled* to get the *com_amdt_tabled* variable.
- *get_com_amdt_adopt* to get the *com_amdt_adopt* variable.
- *get_amdt_tabled* to get the *amdt_tabled* variable.
- *get_amdt_adopt* to get the *amdt_adopt* variable.
- *get_vote_tables* and *get_vote* for the *votes_for_1*, *votes_agst_1*, *votes_abs_1*, *votes_for_2*, *votes_agst_2* or *votes_abs_2* variables.
- *get_rapps_html*, *get_rapps*, *get_country_instance*, *get_country*, *get_party*, *save_party_family* and *get_rapp* to get the *rapp_1-5* and their associated variables (*country*, *party*, *party_family*).
- *get_modif_propos* for the *modif_propos* variable.
- *get_sign_pecs* to get the *sign_pecs* variable.

- *get_dg_names* for the *dg_1-2* variables.
- *get_resp_names* to get the *resp_1-3* variables.

➢ *get_data_others* is used to retrieve information from external tables. Collected information is stored in a dictionary. Here are the main functions:

- the main function is called *get_data_others* and is used to get all the variables from external tables. It calls all the other functions.
- *save_gvt_compo* and *get_gvt_compo* to get the *gvt_compo* variables (*country*, *party*, *party_family*).
- *save_get_min_attend* for the *min_attend* variables (*country*, *status*, *verbatim*).
- *save_get_opal* to get the *opal* variables (*case_nb*, *country*, *act_type*, *act_date*).

➢ *views.py*: render the Act data validation page and calls the following functions:

- *ActUpdate* view: this is the view that actually renders the Act data validation page. It processes the *add* or *modification* of an act (*Add*/*Modif* forms), *update* code_agenda/rapp or resp or dg related variables (click on *update button*) and *save* an act (click on *save button*). It contains 3 main functions: *post* to deal with post data (calls all the functions above to retrieve all the data of the act), *form_valid* when the form is valid and ready to be saved and *form_invalid* when at least one value from the form is not valid and need to be changed in order to save the act data.
- *get_data_all* and *get_data* are the main functions called by the *post* method of the view to get all the variables of the act from *eurlex and oeil* and external sources. Other functions help to get and format some specific variables: *get_party_family*, *check_multiple_dgs*, *store_dg_resp*, *get_adopt_variables and get_cons_vars*.
- *reset_form*: this view is called with Ajax to empty the *ActForm* form (when an *act* has been saved or when the *Add* / *Modif* form is not valid).
- *update_code_sect*, *update_person* and *update_dg* are views for ajax only used to update *code_agenda_1-4*, *rapp_1-5* or *resp_1-3* related variables and *dg_sigle* variables, respectively. The view is called when one selects another value in one of the associated drop down lists.
- *update_durations* is a view for ajax only used to update the *duree_adopt_trans*, *duree_proc_depuis_prop_com*, *duree_proc_depuis_trans_cons*, *duree_tot_depuis_prop_com* and d*uree_tot_depuis_trans_cons* variables. The view is called on the click of the blue circle (with two white arrows inside) in front of the *duree_adopt_trans* variable.

➢ var_name_data.py: does the match between variables name used for the application and variables names displayed or exported. It contains data variables only.

➢ the *index.html* file *within* the folder *templates/act* is used to render the Act data validation page and is called by the *ActUpdate* view. This file calls the *form.html* file that contains the form of the Act data. Also, the *add_modif.html* file is used to display the Add and Modif forms while the *notes_save.html* file is used to display the note field and to save the form.

db_mgmt

This is the *db_mgmt* folder, linked to the Act data validation page (with JavaScript activated) and to the Database management page (with JavaScript deactivated). It allows an expert to add a *rapp*, *dg* or *resp* variable in the database. Here is a short explanation of each important file of the folder:

- *urls.py*: contains all the possible urls of the application.
- *forms.py*: contains the *AddDG* form to add a new DG, *AddRapp* form to add a new *rapp* and *AddResp* form to add a new *resp* into the database. It defines a customized validation of the fields and describes the form used by the view and displayed in the template.
- *views.py*: render the modal displayed inside the Act data validation page (if javascript activated) or the Database management page (if javascript deactivated). It is composed of:
  - *init_response*: called by the view to pass parameters to the template (through the *response* dictionary)
  - *add*: this is the view that actually renders the forms and save them when valid.
  - *form_add*: this view is called with Ajax to empty / reset the *AddDG*, *AddRapp* or *AddResp* form (when an *object* has been saved or when the corresponding form is not valid).
- the add.*html* file *within* the folder *templates/db_mgmt* is used to render the modal or the Database Management page and is called by the *add* view (the modal is included in the *modal.html* file at the root of the templates folder). This file calls the *form_add.html* file that contains the form to add a *dg* or a *rapp* or a *resp*.

export

This is the *export* folder, linked to the Export page. It allows the user to export all the acts stored in the database and already validated in the csv format. Here is a short explanation of each important file of the folder:

- urls.py: contains all the possible urls of the application.
- views.py: contains the *export* view called to render the export page. It creates a csv file containing all the validated acts on the server and then download it onto the client's computer through his/her web browser; you can also access the last automatic export (created every Sunday). The view calls the following functions:
  - get_headers: get the name of the fields to export
  - get_save_acts: fetch and save all the validated acts in a csv file
  - send_file: send the export file from the server to the client's web browser
- the *index.html* file *within* the folder *templates/export* is used to render the Export page and is called by the *export* view.

history

This is the *history* folder, linked to the History page This application stores every saved acts and display the 100 latest. Here is a short explanation of each important file of the folder:

- urls.py: contains all the possible urls of the application.
- *models.py*: contains the *History* model, that records every saved acts: date and time, action

performed (“add” or modif”), form used (“ids”, “data” or “attendance”) as well as the user who performed the save and the ids of the act.

- views.py: contains the *HistoryListView* view called to render the History page. It returns the history of the latest 100 saved acts as well as the number of validated acts.
- the *index.html* file *within* the folder *templates/history* is used to render the History page and is called by the *HistoryListView* view.

Other folders

The project contains files or folders that are not part of the applications above or not yet cited above. Let us see them now in details.

Common module

This folder contains common files used by at least two applications and that can't be stored in just one application folder. They are mainly composed of functions called from within the applications.

- ***config_file.py***: **this file is very important** and contains all the constants of the project. It gives the **url pattern of eurlex and oeil** and thus **must be updated each time one of these urls is modified.** It also gives, for each act, the maximum number of *cs* (code sectoriel), *rep_en*, *dg*, *rapp* and *resp* variables.
- *db.py* contains all the common functions that interact with the database:
  - *get_act_ids*: get the act ids of an act, from the *ActIds* model, for each source ("index", "eurlex" or "oeil").
  - *save_fk_code_sect*: find and link the matching *code_agenda* or *config_cons* to the *code_sect* instance in parameter.
  - *save_get_object*: return the instance of a variable in parameter (if the instance does not exist, create it and then return it).
  - *save_get_field_and_fk*: save a field and its foreign keys in the corresponding models (e.g.: save a *person* instance and its related attributes: *country, party, party_family*).
  - *save_get_resp_eurlex*: save and return the *resp* instance from a *name* in parameter.
  - *is_member*: check if a user belongs to a group given in parameter (used to check the authorization to access the export page for example).
  - *user_context*: add two extra variables to each request dictionary passed to a view or a template: the groups the current user belongs to and his/her name (see TEMPLATE_CONTEXT_PROCESSORS in europolix/settings.py).
- *forms.py*: contains the *AbstractModif* form which is a generic form for the modification forms (on the Act ids, Act data and attendance pages).
- *functions.py*: contains all the common functions that do not interact with the database:
  - *split_fr_date*: split a French date ('DD-MM-YYYY' or 'DD/MM/YYYY') into year,

month and day.

- *date_split_to_iso*: transform a split date to the iso format ('YYYY-MM-DD').
- *date_string_to_iso*: transform a string date (American format with "/" or French format) to the iso format ('YYYY-MM-DD').
- *list_reverse_enum*: reverse a list or string and return it along with the original position of each element.
- *remove_nonspacing_marks*: remove non-spacing marks (accents) of a string.
- *format_dg_name*: format all the *dg* names in the db so they respect the following format: "DG ..."

➢ *view.py*: contains common functions used in views:

- *get_ajax_errors*: return the form errors in a *json* format so they can be used by a *javascript* program (for *ajax*).
- *get_ordered_queryset:* order the *attendances* of a specific *act*, to be displayed on the attendance page.
- *check_add_modif_forms*: this function is used by the act ids, act and attendance views. Three goals: 1/ detect if the user is adding or modifying an act 2/ check whether the corresponding form (*Add* or *Modif* form) is valid or not 3/ get the data of the act if the *Add* or *Modif* form is valid.

Templatetags

Temple tags can be found within any application. They are usually stored within a *templatetags* subfolder and are specific to the application. They are used to give more control to the template: python code (*functions*) can be executed directly from the template, with arguments in parameter and return arguments. Currently, there are two applications using templatags in the project:

➢ *import_app*: contains the function *selected_label* that displays a help text after selection of an import in the drop down list.

➢ *act*: contains three functions:

- *get_value*: get the value associated to a key in a dictionary.
- *get_related_field*: get the related field of a selected value in a drop down list (e.g.: *code_agenda* for *code_sect*).
- *get_party_family*: get the *party_family* variable from the *pers id* in parameter.
- numeric_loop: loop over a range of numbers in a template.

We now know the use of every file / folder needed to have the project working. Yet, there are other files that are useful to process tasks from outside the website and to perform statistics upon the database...

Commands

Explanation

Commands are tools used for administrative tasks. Commands are django programs developed inside an application. They are directly called from a terminal, and not from the website linked to the project. It's like a *python* program that gives you access to any component of the project such as its global settings, its database, its functions, etc.

To write a command in the application *my_app*: create a folder *management* inside *my_app*, then create a folder *commands* inside *management*. Each folder must contain an empty *__init__.py* file to be recognized as a *python* program. In the commands folder, create *my_command.py* and write your command inside the file. Here is the structure of the project:

```
root/
    manage.py
    my_app/
        __init__.py
        management/
            __init__.py
            commands/
                __init__.py
                my_command.py
```

To execute your command, go to the root of the project in a terminal and execute the following command (don't write the file extension of the command):

python manage.py my_command

For a command with no argument, here is the structure of the file :

```
from django.core.management.base import NoArgsCommand

class Command(NoArgsCommand):

  def handle(self, **options):
      #write the code of the command
      ...
```

Currently, two applications use commands. Let's see them in details...

import_app

The *import_app* contains many commands. Let's see the most important ones:

- *attendance_html*.py* and *attendance_pdf*.py* files are commands to retrieve and import into the database ministers' attendances for the years 1996, 1997, 1998, 2002, 2013 and 2014. The other years have been manually imported already. The difficulty of this import is that each year uses different file formats. Before 2000, *html* files were used. In more recent years, *pdf* files are used. Even with the same extensions, some files need different tools to be correctly read. Until 2013, the *poppler-utils* package is required for the import. From 2014, the *pdftotext* package is required. For future imports (2015 and later), the *pdftotext* package is likely to be used again.
- *get_save_act_data.py*: for a set of acts, this command automatically gets data from *eurlex*, and *œil* and saves them in the database. It is an automatic validation and can be used for test validation.
- *import_index_file_again.py*: this commands is used when there is a problem with an index file. We import the ids of the concerned acts again and devalidate them if there is at least one different field compared to the first import.
- *update_act_ids.py*: update the act ids of a set of acts from a csv file uploaded in the */media/import/* folder. The columns of the csv file must be the following: *releve_annee*, *releve_mois*, *no_ordre*. This command mimics a click on the *Update* button in the Act ids validation form (see Act ids validation page).
- *update_adopt_cs.py*: if the *adopt_cs_abs* or *adopt_cs_contre* variables are not present in the database for some acts, you can use this command to add them afterwards, from a csv file uploaded in the */media/import/* folder. The columns of the csv file must be the following: *releve_annee*, *releve_mois*, *no_ordre*, *adopt_cs_abs*, *adopt_cs_contre*. The *adopt_cs_abs* and *adopt_cs_contre* columns must contain country codes separated by commas or semi-columns.
- *update_attendance_url.py:* when an *act* has no *attendance_pdf* field in the database, this command adds it afterwards, from a csv file uploaded in the */media/import/* folder. The csv file is an index file (RMC) with a extra column added for the field to import.
- *update_dg.py*: update the format of the *dg* names to the following: “DG ...”.
- *update_min_attend.py*: this command updates *minister's attendance* variables for every *act* and also checks for errors.
- *update_nb_mots.py*: update all the acts for which the *nb_mots* field is not filled yet.
- *update_party_family_rapp.py*: With the help of the *ImportRappPartyFamily* table, update the *party_family* variable of each *rapp* variable if not imported yet.

export

The export application contains only one command called *export_db.py* to export all the acts from the database. It exports every variable of every act in a csv file, stored in the */media/export/* folder.

The *export_db_cron* file is a cron task that launches the *export_db* command every Sunday at 11pm.

You can access the latest exported file in the Export page, at the bottom of the form.

If automatic exports do not work anymore, you need to change the log files permissions on the may server:

```
ssh -X [username]@may.cdsp.sciences-po.fr
cd /data/www/europolix/
sudo chmod 777 europolix.log*
```

There is another application that uses commands and that we haven't seen yet. In fact this application is not really part of the project, but it is stored there for convenience. Let us see it in the next section.

Statistics module

The statistics module is used to make a statistical analysis of any data stored in the database. The statistical analysis is part of the project hosting the website even though it's not directly related and could be stored elsewhere. It is so for a question of ease. Let's now see how it works.

Types of analyses

There are currently 5 main kinds of queries, determined by the factors to analyze:

- factor *all*
- factor *periods*
- factor *country*
- factor *cs*
- factor *year*
- factor *csyear*

For example if you want to analyze the **number of adopted acts**, you can count the number of adopted acts:

- since 1996 (factor *all)*
- during specific periods of time, usually 4 periods (factor *periods)*
- for each *country*, where a country is the EU state of the first rapporteur of the act
- for each “code sectoriel”, since 1996 (factor *cs)*
- each year (factor *year)*
- for each “code sectoriel” and each year (factor *csyear)*

Structure of the module

This part describes the organization of he module.

queries.py

The *queries.py* file is the **main file** of the application and aims to call the queries to perform (they are stored in other files, see query folder). One query is in fact a function called “q” with the number of the query as suffix. For example, if you're calling the query number 2, you have to call it this way:

```
q2()
```

The number has no meaning, it's just a convention to avoid long names for queries and to get a

chronological order. Thus, if *q2* is the last created query and you want to create a new one, the new query will be *q3*.

A query function always calls the 3 following functions: *init*, *get*, and *write*.

init.py

init.py is the first file used for a query and contains the *init* function. It initializes the data structure that will store the results of the query. According to the factor (see Types of analyses), it can be a simple variable, a list or a dictionary. Let's call this data structure *res*.

Let's see examples with a few queries:

- **Number of adopted acts since 1996**. *res* needs to store only one number:

  res=0 → zero initializes the number of adopted acts since 1996. It corresponds to the *all* factor.

- **Number of adopted acts before and after 2005**. *res* needs to store two numbers, we can use a list:

  res=[0, 0] → the first zero initializes the number of adopted acts before 2005 and the second zero initializes the number of adopted acts after 2005. It corresponds to the *periods* factor.

- **Percentage of adopted acts by country**. for every country, we need to compute the percentage of adopted acts. *res* needs to be a dictionary where each country is a key and where the value is a list of two numbers: the first one is the total number of adopted acts for the specific country and the second one is the total number of adopted acts.

  res={"FR": [0, 0], "DE": [0, 0], …, "DK": [0, 0]} → For France ("FR"), the first zero initializes the number of acts of the country whereas the second zero initializes the total number of acts. It corresponds to the co*untry* factor.

- **Number of adopted acts by cs**. *res* needs to be a dictionary where each cs (code sectoriel) is a key and the number of acts for a specific cs a value.

  res={"01": 0, "02": 0, …, "20": 0} → the first zero initializes the number of acts for the sector "01", the second zero initializes the number of acts for the sector "02" and the last zero initializes the number of acts for the sector "20". It corresponds to the *periods* factor.

- **Average number of words each year**: for every year, we need to compute the average number of words per act. *res* needs to be a dictionary where each year is a key and where the value is a list of two numbers: the first one is the total number of words for the specific year and the second one is the number of acts for the specific year.

  res={1996: [0, 0], 1997: [0, 0], …, 2014: [0, 0]} →*For the year* 1996, the first zero initializes the total number of words of all the acts of this specific year while the second zero initializes the number of acts for this specific year. It corresponds to the *year* factor.

- **Number of adopted acts by cs and by year**. *res* needs to be a dictionary of dictionary where each cs and each year is a key and the number of acts for a specific cs and year a value.

  res={"01": {1996: 0, 2014: 0}, "20": {1996: 0, 2014: 0}} → For the cs "01" and the year 1996, the zero initializes the number of adopted acts for this specific cs and specific year. It

corresponds to the *csyear* factor.

The *init* function calls different functions according to the factor:

- *init_all* for the *all* factor and *periods* factor
- *init_country* for the *country* factor
- *init_cs* for the *cs* factor
- *init_year* for the *year* factor
- *init_csyear* for the *csyear* factor

get.py

get.py is the second file used for a query and contains the *get* function. It updates the *res* data structure (see init.py) that stores the results of the query: at the end of the call, *res* contains the final results.

Let's take the same examples we used for the *init* function and see how is updated res at the end of the call of the *get* function (with dummy numbers as examples):

- **Number of adopted acts since 1996**:

  res=3000 → there has been 3000 adopted acts since 1996.

- "Number of adopted acts before and after 2005" :

  res=[1200, 1800] → there were 1200 adopted acts before 2005 and 1800 adopted acts after 2005.

- **Number of adopted acts by cs**:

  res={"01": 200, "02": 100, …, "20": 250} → there were 200 adopted acts for the cs "01", 100 for the sector "02" and 250 adopted acts for the cs "20".

- **Percentage of adopted acts by country**:

  res={"FR": [200, 1000], "DE": [100, 1000], …, "DK": [250, 1000]} → For France ("FR"), there were 200 adopted acts out of a total of 1000 acts.

- **Average number of words each year**:

  res={1996: [1000, 10], 1997: [5000, 20], …, 2014: [4000, 20]} → *There were 10 adopted acts in 1996 and their total number of words is 1000, whereas there were 20 adopted acts in 2014 and their total number of words is 4000 (see write.py for the average computation).*

- **Number of adopted acts by cs and by year:**

  res={"01": {1996: 10, 2014: 20}, "20": {1996: 15, 2014: 35}} → *There were 10 adopted acts in 1996 for the sector "01" whereas there were 35 adopted acts in 2014 for the cs "20".*

The *get* function calls different functions according to the factor:

- *get_all_year_periods* for the *all* factor, the *year* factor and *periods* factor

- *get_countries* for the *country* factor
- *get_cs_csyear* for the *csyear* factor

The *compute* function makes the necessary computations to update the *res* structure until it stores the final result (after all the acts have been processed).

write.py

write.py is the third and last file used for a query and contains the *write* function. From the *res* data structure containing the final results (see get.py), it writes them down in a csv file, but first computes the average or percentage if required by the query.

Let's take the same examples we used for the *get* function and see what is done in the *write* function: (with still dummy numbers as examples):

- **Number of adopted acts since 1996**. The output in the csv file is:

  3000 → no modification compared to the *get* result.

- **Number of adopted acts before and after 2005**. The output is:

  1200, 1800 → no modification compared to the *get* result.

- **Percentage of adopted acts by country**:

  20%, 10%, …, 25% → *There were 200 adopted acts for the country France and the total number of adopted acts is 1000: the percentage is 200*100/1000=20% and corresponds to the first output number;there were 250 adopted acts for the Denmark and the total number of acts is 1000: the average is 250*100/1000=25% and corresponds to the last output number;*

- **Number of adopted acts by cs**:

  200, 100, …, 250 →no modification compared to the *get* result.

- **Average number of words each year**:

  100, 250, …, 200 → *There were 10 adopted acts in 1996 and their total number of words is 1000: the average is 1000/10=100 and corresponds to the first output number;there were 20 adopted acts in 2014 and their total number of words is 4000: the average is 4000/20=200 and corresponds to the last output number;*

- **Number of adopted acts by cs and by year**.

  10, 20, 15, 35 → no modification compared to the *get* result.

The *write* function calls different functions according to the factor:

- *write_all* for the *all* factor
- *write_cs_year_country_periods* for the *cs, year, country and periods* factors
- *write_csyear* for the *csyear* factor

The *compute* function makes the average or percentage computations if needed.

common.py

This file is composed of *functions* and *constants* used by the *init.py*, *get.py* and *write.py* files. It is used to:

- get a list of *subfactors* (for example all the different *years* if the factor is *year*):
  - *prepare_query* is the main function, called in every query. It calls the functions below to get the factors of the query, their associated question title (see below) and get the periods of the query (in the French format for display purpose and in American format for Django queries).
  - *get_factors function* and *factors constant* to get the list of factors by default
  - *get_factors_dic, get_factors_question* to get the text of the question corresponding to each factor (*example*: we want to determine the *number of acts per year*. In the query, we initialize the question: “Number of acts”, regardless of the factor. The two functions above will add the text “per year” to the question title whenever the *year* factor is called).
  - *get_periods function* and *periods constant* to get the default periods (*periods* factor)
  - *get_countries function* and *countries_list* function for the *country* subfactors
  - *get_css function* and *css* constant for the *cs* subfactors
  - *get_years function* and *years_list constant* for the year subfactors
- manipulate dates with *str_to_date* and *fr_to_us_date*
- update the *filter* dictionary of the query (used in a *django filter* query):
  - *get_validated_acts* to get all the validated acts to process
  - *get_validated_acts_periods* to get all the validated acts to process for a given period
- other constants are used for every query:
  - *min_year:* oldest year used for validation (1996 at the time of writing)
  - *max_year:* most recent year used for validation (2014 at the time of writing)
  - *nb_css:* maximum number of *code sectoriel* per act (4 at the time of writing)
  - *nb_dgs:* maximum number of *directeur general* per act (3 at the time of writing)
  - *nb_rapps:* maximum number of *rapporteur* per act (5 at the time of writing)
  - *nb_resps:* maximum number of *responsable* per act (3 at the time of writing)

queries.csv

The *queries.csv* file is the output file where is written the result of the queries. The path of the output file can be changed in the *write.py* file, variable *path*, line 25.

check_queries.sql

This file is composed of *sql* queries to check and validate the results given with *Django*. For each query there is a corresponding *sql* query, with the number of the query (the queries are in chronological order).

query folder

The *query* folder gather all the queries called in the *queries.py* command file (see queries.py). Each query uses the three functions, *init*, *get* and *write* seen previously plus the *common.py* file. It is composed of many files that distinguish the different kind of queries:

- *acts.py*: all the queries that start with :
  - "Number of acts with": **Nombre d'actes adoptés sans point B** (*q103*)
  - "Percentage of acts with": **Pourcentage d'actes avec un vote public** (*q107*)
- *adopt_cs.py*: all the queries that deal with the *adopt_cs_contre* and *adopt_cs_abs* variables:
  - **Pourcentage de AdoptCSContre=Y, parmi les actes AdoptCSRegleVote=V** (*q97*)
- *country.py*: all the queries that deal with the distribution of EU countries (for *rapporteurs*, *responsibles* or *adopt_cs* variables)
  - **Parmi les votes AdoptCSAbs=Y, pourcentage de votes de chaque Etat membre** (*q126***)**
- *duree.py*: all the queries that deal with the average duration of the acts for duree_proc_depuis_prop_com, duree_proc_depuis_trans_cons, duree_tot_depuis_prop_com and / or duree_tot_depuis_trans_cons
  - **Durée moyenne de la procédure** (*q128*)
- *ep_amdt_vote.py:* all the queries that deal with amdt variables (*com_amdt_tabled*, *com_amdt_adopt*, *amdt_tabled*, *amdt_adopt*) or vote variables (*votes_for_1*, *votes_for_2*, *votes_agst_1*, *votes_agst_2*, *votes_abs_1*, *votes_abs_2*):
  - **Nombre total d'EPComAmdtAdopt** (*q99*)
  - **Nombre moyen de EPVotesFor1-2** (*q127*)
- *min_attend.py: all the queries that deal with the* percentage of attendance of ministers:
  - **Pourcentage de M présents parmi les personnes de statut différent de NA ou AB** (*q117*)
- *modif_propos.py:* all the queries that deal with the percentage of acts modified by commission:
  - **Pourcentage des propositions modifiées par la Commission** (*q27)*
- *nb_mots.py:* all the queries that deal with the number of words:
  - **Total nombre de mots * nombre d'actes** (*q83*)
  - **Nombre de mots moyen** (*q54*)
- *party_family.py:* all the queries that deal with the political families:

- **Pourcentage des familles politiques des RespPropos** (*q93*)
- **Pourcentage de discordance de PartyFamilyRappPE* et PartyFamilyRespPropos*** (*q84*)

➢ *point_b.py:* all the queries that deal with the number of "points B"

- **Nombre total de points B** (*q65*)
- **Nombre moyen de points B** (*q73*)
- **Pourcentage de points B par rapport aux points A** (*q95*)

➢ *type_acte.py:* all the queries about the type_acte variable:

- **Nombre de d'actes de type CS DVE+DVE** (*q85*)

➢ *vote.py:* all the queries about the number of votes:

- **Nombre de votes** (*q17*)

The Statistics module has been rewritten a few times. Some queries do not work anymore because they refer to functions that do not exist anymore → those queries must be rewritten!

If you want to call the query *q103* (for example) of the *acts.py* file, you must call it in the *queries.py* file as below:

```
acts.q103()
```

result folder

The *result* folder contains all the results of the previous sets of executed queries. Each time the *queries.py* file is ran, the output is saved in *queries.csv*. Before running another set of queries, a good practice is to rename the file (using the date and the kind of queries inside) and to move it to the *result* folder.

Parameters

The three main functions init (see init.py), get (see get.py) and write (see write.py) use many arguments or parameters that depend on the query (these arguments can be used in only one function or two or all of them). Let's see them in details.

factor

This is the factor of the analysis. For example, if we want to analyze the **number of acts every year**, then *factor="year"*.

res

res is the data structure that stores the result of all the queries.

count

- *count* is true: we are doing a *percentage* or *average* computation.
  - query **percentage of acts with a public vote** for a percentage computation (*q107*)
  - query **average number of words per act** for an average computation (*q54*)
- *count* is false: we are simply counting a number of occurrences
  - query **number of acts** (*q2*)

total and res_total

total is used for *percentage* or average *computation.* Let's take again the example of the **Percentage of adopted acts by country** (*q126*, see get.py). The final *res* data structure is:

res={"FR": [200, 1000], "DE": [100, 1000], …, "DK": [250, 1000]}

The second number of the list of any country corresponds to the total number of acts. This number is the same no matter the country because it is not related to any country.

Instead of storing the same total number for each country, we can use a *res_total* variable:

res={"FR": 200, "DE": 100, …, "DK": 250}

res_total=1000

In this situation *count = false* because we are "simply" counting a number of occurrences in *res (see count).*

query

When there is a particular query, we use the *query* parameter to tell the program to do a specific computation (which depends on the query itself).

e.g.: the query **Pourcentage d'actes avec au moins un EM sans statut 'M'** (*q122*) requires a specific computation (see *compute* in *get.py*), which is indicated with the *query* parameter (*query="no_minister_percent"*).

Model

Most of the time, we only need the *Act* model to perform the query. It is the default model. But sometimes, we need to used the *MinAttend* model (for ministers' attendances queries) or *ActIds* (when we want to filter by an act id field). In such a situation, we use the *Model* parameter to indicate which model to use.

e.g.: for the query **Pourcentage d'actes adoptés en 1ère lecture parmi les actes de codécision** (*q74*), *Model=ActIds*.

variable

When the query is not about counting a number of occurrences but adding the values taken by a variable, we must use the *variable* parameter.

e.g.: the query **Nombre de mots moyen** (*q54*) uses this parameter to indicate that we want to sum the number of words for each act (*variable="nb_mots"*).

variable_2

We use the *variable_2* parameter when we want to compute the average of two variables.

e.g.: the query **Nombre moyen de EPVotesFor1-2** (*q127*) uses the *variable* and *variable_2* parameters: *variable=votes_for_1* and *variable_2= votes_for_2*.

This query means:

1) for each act, compute the average between *votes_for_1* and *votes_for_2*.
2) *add the average value of each act and compute the average for all the acts.*

For some acts, only one variable has a value. Let's take the example of an act where there is no value for *votes_for_2*. In such a situation, we don't compute the average in 1), we just take the value of *votes_for_1*.

res_2

We use the *res_2* parameter when we want to compute the average of two variables.

e.g.: **Durée moyenne de la procédure (= Moyenne DureeTotaleDepuisTransCons + DureeProcedureDepuisTransCons)** (*q128*)

res_1 is used to get the sum of *DureeTotaleDepuisTransCons* while *res_2* is used to get the sum of *DureeProcedureDepuisTransCons.*

Unlike the variable_2 parameter, the 2 variables are computed over the same set of acts : if DureeTotaleDepuisTransCons is computed over a set of 10 acts, DureeProcedureDepuisTransCons is computed over the same set of 10 acts, which means if one of the two variables is empty, the act is not taken into account.

exclude_values

This parameter appears in the *get* function only and is always called with its default values: “” and *None*. When there is a *variable* parameter, it excludes acts for which the *variable* value is *empty* or *None*.

filter_vars_acts

filter_vars_acts updates the *django filter dictionary* to indicates the acts to loop through for the analysis. The variables to filter come from the *Act* model. In a **percentage** computation, it is associated to the **denominator**.

e.g.: **Nombre de mots moyen** (*q54*):

filter_vars_acts={nb_mots__isnull: False}

It indicates that we don't take into account acts with a *None* or *empty nb_mots* variable.

filter_vars_acts_ids

filter_vars_acts_ids updates the *django filter dictionary* to indicates the acts to loop through for the analysis. The variables to filter come from the *ActIds* model. In a **percentage** computation, it is associated to the **denominator**.

e.g.: **Pourcentage d'actes adoptés en 1ère lecture parmi les actes de codécision** (*q74*):

filter_vars_acts_ids={"no_unique_type": "COD"}

It indicates that we take into account only acts with a *no_unique_type=”COD”*.

exclude_vars_acts

exclude_vars_acts updates the *django exclude dictionary* to indicates the acts **not** to loop through for the analysis. The variables to exclude come from the *Act* model.

e.g.: **Nombre moyen de EPVotesFor1-2** (*q127*):

exclude_vars_acts={votes_for_1: 0, votes_for_2: 0}

It indicates that we don't take into account acts for which both *votes_for_1* and *votes_for_2* are equal to zero.

check_vars_acts

check_vars_acts is used for *percentage* computation. It is used to compute the numerator of the percentage. The variables come from the *Act* model. In a **percentage** computation, it is associated to the **numerator**.

e.g.: **Pourcentage d'actes adoptés en 1ère lecture parmi les actes de codécision** (*q74*):

check_vars_acts={"nb_lectures": 1}

The numerator of the percentage corresponds to the number of acts for which *nb_lectures=1* (among the acts with *no_unique_type=”COD”*).

check_vars_acts_ids

check_vars_acts_ids is used for *percentage* computation. It is used to compute the numerator of the percentage. The variables come from the *ActIds* model. In a **percentage** computation, it is associated to the **numerator**.

e.g.: **Pourcentage d'actes provenant de la Commission et adoptés par procédure écrite** (*q71*):

check_vars_acts_ids={"propos_origine": "COM"}

The numerator of the percentage corresponds to the number of acts for which *propos_origine=”COM”*.

adopt_var

adopt_var is used to check the existence of an *adopt_cs_contre* or *adopt_cs_abs* variable for a given act. It cannot be checked simply with *act.adopt_cs_contre==True* because *adopt_cs_contre* is a *many to many relationship*. We then use another test to check it and use the *adopt_var* variable to indicate the program to run such a test.

e.g.: **Parmi les votes AdoptCSAbs=Y, pourcentage de votes de chaque Etat membre** (*q126*):

adopt_var="adopt_cs_abs"

We are looking for acts with *adopt_cs_abs = True*.

periods

When the factor is *periods*, we can indicate specific periods of the query in parameters.

e.g.: **Nombre d'actes** (*q2*)

In this question, the periods used are:

```
periods=(
        ("1996-01-01", "1999-10-31"),
        ("1999-11-01", "2004-10-31"),
        ("2004-11-01", "2009-10-31"),
        ("2009-11-01", "2013-12-31")
    )
```

This means we are looking for the number of acts between *1996-01-01* and *1999-10-31*, between *1999-11-01* and *2004-10-31*, between *2004-11-01* and *2009-10-31* and finally between *2009-11-01* and *2013-12-31*.

question

This argument indicates the text of the query to write in the csv file.

e.g.: In *q2*, *question=" Nombre d'actes"*.

percent

percent indicated the multiplicator of the numerator of a query:

- *percent=100* for *percentage computation*

  e.g.: **Pourcentage d'actes avec un vote public** (*q107*)
- percent=1 in all the other queries

We have seen all the different parameters used to run a query, let's see now how to use them with examples of queries.

Examples

This part shows query examples to better understand how this module work and to be able to create your own queries afterwards.

The examples below were run and tested on *February 20, 2015* and can be found in *result/2015-02-20_TESTS.csv*.

Nombre d'actes

This is the query *q2()*. Let's run it on 5 different factors (see Types of analyses): *all*, *periods*, *cs*,

year and *csyear*.

- output

Nombre d'actes, pour la période 1996-2013
3154

Nombre d'actes, par année

1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
226	205	218	199	184	185	195	199	233	135	216	161	246	237	61	84	78	92

Nombre d'actes, par secteur

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
182	247	780	344	166	140	264	37	245	97	164	100	509	75	438	121	62	11	352	13

Nombre d'actes, par secteur et par année

	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
1	4	0	12	6	14	1	3	9	19	8	25	12	24	15	5	5	7	13
2	36	17	14	23	24	14	8	7	9	13	15	22	16	13	3	4	4	5
3	94	91	83	69	55	53	32	53	25	21	50	34	39	51	7	6	9	8
4	39	42	45	21	17	14	13	12	29	20	19	20	21	16	5	3	4	4
5	6	6	11	10	7	9	11	13	15	10	15	8	14	11	5	3	5	7
6	6	13	10	1	6	12	11	3	6	7	6	7	17	13	8	5	2	7
7	18	8	18	11	10	12	17	14	25	12	20	11	20	46	10	5	2	5
8	2	4	3	4	1	1	8	3	3	0	3	1	1	3	0	0	0	0
9	3	27	8	18	16	18	21	15	31	13	16	13	17	11	6	6	1	5
10	1	4	8	4	5	6	3	2	5	6	4	7	9	12	6	4	5	6
11	7	4	17	8	12	9	6	8	22	7	9	3	10	8	3	13	12	6
12	17	4	11	5	5	1	6	7	7	4	7	4	5	11	2	1	2	1
13	38	23	32	34	22	24	42	28	37	27	33	24	47	52	6	11	14	15
14	2	0	1	8	2	2	2	4	2	4	9	2	5	14	5	4	2	7
15	15	22	22	29	28	25	35	35	30	21	32	23	36	36	5	20	10	14
16	8	6	10	19	5	2	7	6	11	5	8	5	17	6	0	1	2	3
17	2	0	4	1	1	6	1	9	5	2	6	3	3	8	0	4	7	0
18	0	0	0	0	2	0	1	1	0	0	0	1	2	2	0	1	1	0
19	0	1	1	1	11	31	33	42	36	22	28	26	59	22	8	14	6	11
20	1	0	0	0	1	0	0	0	3	0	2	0	1	1	0	1	2	1

Nombre d'actes, par période

Période 1	Période 2	Période 3	Période 4
795	961	740	658

- query

```
def q2(factors=factors, periods=None):
    #Nombre d'actes
    init_question="Nombre d'actes"

   #get the factors specific to the question and update the periods (fr to us format)
    factors_question, periods=prepare_query(factors, periods)

    #for each factor
    for factor, question in factors_question.iteritems():
        question=init_question+question
        res=init(factor, count=False)
        res=get(factor, res, periods=periods, count=False)
        write(factor, question, res, periods=periods, count=False)
```

- explanation

  - In the definition of the functions, there are always 2 parameters (see Call a query): *factors* indicates the factors to analyze and *periods* indicates the periods to analyze, if any (see Parameters).

- factors is initialized this way:

  ```
  factors=["all", "year", "cs", "csyear", "periods"]
  ```

- periods is initialized this way:

  ```
  periods=(
          ("1996-01-01", "1999-10-31"),
          ("1999-11-01", "2004-10-31"),
          ("2004-11-01", "2009-10-31"),
          ("2009-11-01", "2013-12-31")
  )
  ```

- *get_factors_question* returns an ordered dictionary with the factors as *keys* and suffixes of the questions as *values*.

  For example, here the question without suffix is *init_question="Nombre d'actes"*. With the function mentionned above, we loop through each factor this way:

  ```
  for factor, question in factors_question.iteritems():
  ```

  And we update the question according to the factor this way:

  ```
  question=init_question+question
  ```

  Let's consider the *year* factor. The suffix of this factor is ", par année", then the question becomes *"Nombre d'actes*, par année".

- We now have to call the three main functions *init* (see init.py), *get* (see get.py) and *write* (see write.py) with the appropriate parameters (see Parameters).

  - we always use the factor parameter (first position)

  - *count* is *false* because we are just counting a number of acts, we don't want to do an *average* or *percentage* computations

  - *init* initializes and returns the result data structure *res*.

  - in *get*, *res* is always in second position, it transmits the initialized result data structure to the *get* function. *periods* is always sent to the function in case of a *periods* analysis. *res* receives the final results computed in the *get* function.

  - in *write*, *question* is always the second parameter to output the text of the question of the query in the csv file. *res* is always the third parameter to output the final results.

➢ sql

- For each query executed with *django*, it is good to check the results by running a *sql* query of one or two results per *factor* (in *MySQL workbench* for instance).

- Let's check the result of the year 2000 (*year* factor):

  ```
  select count(*)
  from europolix.act_act
  where validated=2 and releve_annee=2000;
  ```

- **Result: 184**, which is the same result given with the *django* program.

- The *sql* query can be found in the *check_queries.sql* file (see check_queries.sql).

Nombre de mots moyen

This is the query q54*()*. Let's run it on the *year* factor.

- output

Nombre de mots moyen, par année

1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
3675.0	3165.9	2960.8	3467.3	2792.7	3721.8	4004.1	4263.4	4827.1	4990.4	8373.7	6186.5	7056.0	8855.2	8164.6	6856.4	8933.4	16342.6

- query

```
def q54(factors=factors, periods=None):
    init_question="Nombre de mots moyen"
    variable="nb_mots"
    filter_vars_acts={variable+"__isnull": False}

    #get the factors specific to the question and update the periods (fr to us format)
    factors_question, periods=prepare_query(factors, periods)

    #for each factor
    for factor, question in factors_question.iteritems():
        question=init_question+question
        res=init(factor)
        res=get(factor, res, variable=variable, filter_vars_acts=filter_vars_acts,
periods=periods)
        write(factor, question, res, percent=1, periods=periods)
```

- explanation
  - we don't want to add a number of occurrences fir this query but add the values of a variable (sum the number of words for each act). That's why we declare:

    ```
    variable="nb_mots"
    ```
  - We don't want to take into account acts that do not have the *nb_mots* variable filled:

    ```
    filter_vars_acts={variable+"__isnull": False}
    ```
  - we pass the *variable* and *filter_vars_acts* parameters to the get function
  - we want an average, not a percentage: we pass *percent=1* to the *write* function.

- *sql* check for the year 2005:

```
select avg(nb_mots)
from europolix.act_act
where validated=2 and releve_annee = 2005;
```

Pourcentage d'actes avec un vote public

This is the query q107*()*. Let's run it on the *year* factor.

- output

Pourcentage d'actes avec VotePublic=Y, par année

1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
15.0	22.4	32.6	14.6	11.4	17.3	21.5	19.1	14.2	14.8	17.1	23.0	11.4	18.1	19.7	26.2	30.8	33.7

- query

```
def q107(factors=factors, periods=None):
    #Pourcentage d'actes avec VotePublic=Y
    init_question="Pourcentage d'actes avec VotePublic=Y"
    check_vars_acts={"vote_public": True}
    #get the factors specific to the question and update the periods (fr to us format)
    factors_question, periods=prepare_query(factors, periods)

    #for each factor
    for factor, question in factors_question.iteritems():
        question=init_question+question
        res=init(factor)
        res=get(factor, res, check_vars_acts=check_vars_acts, periods=periods)
        write(factor, question, res, periods=periods)
```

- explanation

  - in the percentage computation, the numerator corresponds to the number of acts with a public vote (and the denominator to the total number of acts). For the numerator computation, we use:

    ```
    check_vars_acts={"vote_public": True}
    ```

    It indicates that the numerator should count acts with a public vote only. We then pass it to the *get* function.

- *sql* check for the year 2010:

```
select
(
    select count(*)*100 from europolix.act_act
    where validated=2 and vote_public=True and releve_annee=2010
)
/
(
```

```
        select count(*) from europolix.act_act
        where validated=2 and releve_annee=2010
    ) as final_res;
```

Parmi les votes AdoptCSAbs=Y, pourcentage de chaque Etat membre

This is the query q126*()*. It uses the *country* factor.

- output

Parmi les votes AdoptCSAbs=Y, pourcentage de votes de chaque Etat membre, par état membre

AT	BE	BG	HR	CY	CZ	DK	EE	FI	FR	DE	EL	HU	IS	IE	IT	LV	...
5.5	6.1	1.2	0.0	1.0	2.4	5.1	2.2	2.4	5.5	10.8	2.9	2.0	0.0	2.9	5.1	1.6	...

- query

```
def q126(factors=factors, periods=None):
    #Parmi les votes AdoptCSAbs=Y, pourcentage de chaque Etat membre
    init_question="Parmi les votes AdoptCSAbs=Y, pourcentage de votes de chaque Etat membre"
    #get the factors specific to the question and update the periods (fr to us format)
    factors_question, periods=prepare_query(factors, periods)
    variable="adopt_cs_abs"
    exclude_vars_acts={variable: None}

    #for each factor
    for factor, question in factors_question.iteritems():
        question=init_question+question
        res, res_total=init(factor, count=False, total=True)
        res, res_total=get(factor, res, exclude_vars_acts=exclude_vars_acts, count=False, res_total_init=res_total, adopt_var=variable)
        write(factor, question, res, count=False, res_total=res_total)
```

- explanation

  - we are looking for a percentage of appearance of each countries of the *adopt_cs_abs* variable: we use the *country* factor.
  - As explained before (see total and res_total), we use *res_total* to store the denominator of the *percentage* and use *count=False* and *total=True*.
  - we don't want to loop through all the acts but only the act with *adopt_cs_abs=True*, so we exclude the acts with no *adopt_cs_abs*:

    ```
    exclude_vars_acts={variable: None}
    ```

    We pass the parameter to the *get* function.

- We pass the *adopt_cs_abs* variable name to the *adopt_var* parameter of the get function to retrieve all the countries of *adopt_cs_abs* for each act.

➢ *sql* check for the year 2010:

```
select
(
    select count(*)*100
    from europolix.act_act act, europolix.act_act_adopt_cs_abs adopt
    where validated=2 and releve_annee<2014 and  adopt.act_id=act.id and
country_id="DK"
)
/
(
    select count(*)
    from europolix.act_act act, europolix.act_act_adopt_cs_abs adopt
    where validated=2 and releve_annee<2014 and  adopt.act_id=act.id
) as final_res;
```

Pourcentage d'actes avec au moins un EM sans statut 'M' (et au moins un 'CS' ou 'CS_PR')

This is the query q122*()*. Let's run it on the cs factor.

➢ output

Pourcentage d'actes avec au moins un EM sans statut 'M' (et au moins un 'CS' ou 'CS_PR'), par secteur

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17 ...
71.2	77.7	57.4	82.8	76.1	84.0	80.6	88.9	72.7	70.1	76.4	72.6	71.6	85.5	71.5	69.5	80.3 ...

➢ query

```
def q122(factors=factors, periods=None):
   #Pourcentage d'actes avec au moins un EM sans statut 'M'
   #get the factors specific to the question and update the periods (fr to us format)
   factors_question, periods=prepare_query(factors, periods)
   init_question="Pourcentage d'actes avec au moins un EM sans statut 'M' (et au moins un
'CS' ou 'CS_PR')"

   for factor, question in factors_question.iteritems():
      question=init_question+question
      res=init(factor)
      res=get(factor, res, query="no_minister_percent", periods=periods)
      write(factor, question, res, periods=periods)
```

- explanation
  - all queries about attendance of ministers require special computations (see the *compute* function of the *get.py* file for more details). Special computation because 3 tables are used to get the status of the minister: *Act*, *MinAttend* and *Status*; besides these tables are linked by *m2m* relationships. We pass the *query* keyword to the *get* function to indicate what is the query and what is the computation to perform.

- *sql*: this query is impossible to check with simple *sql*. It could be checked with *pl/sql* though.

Pourcentage d'actes provenant de la Commission et adoptés par procédure écrite

This is the query q71*()*. Let's run it on the cs factor.

- output

Pourcentage d'actes provenant de la Commission et adoptés par procédure écrite, par secteur

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	...
58.8	75.7	57.7	71.2	62.0	62.9	64.4	21.6	55.5	43.3	76.8	35.0	65.4	53.3	66.4	59.5	59.7	...

- query

```
def q71(factors=factors, periods=None):
    #actes pour lesquels ProposOrigine="COM" et ComProc="Written procedure"
    init_question="Pourcentage d'actes provenant de la Commission et adoptés par procédure écrite"
    check_vars_acts={"com_proc": "Written procedure"}
    check_vars_acts_ids={"propos_origine": "COM"}
   #get the factors specific to the question and update the periods (fr to us format)
    factors_question, periods=prepare_query(factors, periods)

    #for each factor
    for factor, question in factors_question.iteritems():
        question=init_question+question
        res=init(factor)
        res=get(factor, res, Model=ActIds, check_vars_acts=check_vars_acts, check_vars_acts_ids=check_vars_acts_ids, periods=periods)
        write(factor, question, res, periods=periods)
```

- explanation
  - the numerator of the percentage corresponds to the number of acts adopted by written procedure. *com_proc* is a field of the *Act* model, we then use *check_vars_acts*:

```
check_vars_acts={"com_proc": "Written procedure"}
```

- the numerator of the percentage also corresponds to the number of acts coming from the commission. *propos_origine* is a field of the *ActIds* model, we then use *check_vars_acts_ids*:

```
check_vars_acts_ids={"propos_origine": "COM"}
```

- The query uses at least one field from the *ActIds* model (here *propos_origine*), we need to pass the *ActIds* model to the *get* function via the keyword *Model*.

➢ *sql* check for the cs “10” (very complex because each act has up to 4 “code sectoriel”, we must use *union all*):

```
select
(
    select sum(res)*100 from
    (
        select count(*)  as res
        from europolix.act_act
        where validated=2 and com_proc="Written procedure" and id in
        (select act_id from europolix.act_ids_actids where src="index" and
propos_origine="COM")
        and code_sect_1_id in (select id from europolix.act_codesect where code_sect like
'10%')
        union all
        select count(*) as res
        from europolix.act_act
        where validated=2 and com_proc="Written procedure" and id in
        (select act_id from europolix.act_ids_actids where src="index" and
propos_origine="COM")
        and code_sect_2_id in (select id from europolix.act_codesect where code_sect like
'10%')
        union all
        select count(*) as res
        from europolix.act_act
        where validated=2 and com_proc="Written procedure" and id in
        (select act_id from europolix.act_ids_actids where src="index" and
propos_origine="COM")
        and code_sect_3_id in (select id from europolix.act_codesect where code_sect like
'10%')
        union all
        select count(*) as res
        from europolix.act_act
        where validated=2 and com_proc="Written procedure" and id in
        (select act_id from europolix.act_ids_actids where src="index" and
```

```
propos_origine="COM")
            and code_sect_4_id in (select id from europolix.act_codesect where code_sect like
'10%')
      ) a1
)
/
(
      select sum(res) from
      (
            select count(*) as res
            from europolix.act_act
            where validated=2
            and code_sect_1_id in (select id from europolix.act_codesect where code_sect like
'10%')
            union all
            select count(*) as res
            from europolix.act_act
            where validated=2
            and code_sect_2_id in (select id from europolix.act_codesect where code_sect like
'10%')
            union all
            select count(*) as res
            from europolix.act_act
            where validated=2
            and code_sect_3_id in (select id from europolix.act_codesect where code_sect like
'10%')
            union all
            select count(*) as res
            from europolix.act_act
            where validated=2
            and code_sect_4_id in (select id from europolix.act_codesect where code_sect like
'10%')
      ) a2
) as final_res;
```

Nombre moyen de EPVotesFor1-2

This is the query q127*()*. Let's run it on the *csyear* factor.

- output

Nombre moyen de EPVotesFor1-2, par secteur et par année

	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013
1	0.0	0.0	0.0	0.0	486.5	0.0	467.0	528.0	560.4	603.9	580.5	454.6	579.6	558.4
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	568.0	631.5	634.7	570.0	545.5	539.1
3	0.0	0.0	0.0	0.0	583.3	415.2	478.2	460.5	550.6	528.5	562.9	621.2	573.6	548.4
4	0.0	0.0	0.0	0.0	0.0	518.4	476.3	522.3	571.8	631.4	615.5	631.0	642.3	598.3
5	0.0	0.0	570.0	0.0	0.0	619.0	446.2	0.0	590.4	565.6	550.8	521.3	546.8	589.9
6	0.0	0.0	0.0	0.0	0.0	619.0	0.0	567.0	554.7	544.7	582.4	610.0	618.5	590.1
7	0.0	0.0	0.0	0.0	0.0	619.0	527.9	535.8	599.2	613.2	591.4	544.8	0.0	594.0
8	0.0	0.0	0.0	0.0	0.0	0.0	534.0	0.0	0.0	601.7	0.0	0.0	0.0	0.0
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	582.0	567.9	555.0	548.7	584.0	584.0	573.8
10	0.0	0.0	0.0	0.0	0.0	315.5	607.0	597.5	562.0	538.3	590.8	475.8	577.2	540.0
11	0.0	0.0	0.0	0.0	0.0	479.3	543.0	0.0	574.3	636.0	590.7	579.8	516.8	589.9
12	0.0	0.0	0.0	0.0	0.0	0.0	457.0	0.0	644.3	559.2	591.5	616.0	500.5	0.0
13	0.0	0.0	0.0	0.0	559.0	0.0	519.9	583.4	601.0	596.5	606.8	532.0	611.0	575.9
14	0.0	0.0	0.0	0.0	603.0	581.3	551.4	0.0	618.0	571.3	579.3	589.3	591.0	585.5
15	0.0	0.0	0.0	0.0	562.0	0.0	512.3	557.9	648.3	581.7	545.0	538.7	607.7	560.9
16	0.0	0.0	0.0	0.0	0.0	0.0	555.6	579.5	561.2	586.8	0.0	597.0	575.0	641.0
17	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	598.7	0.0	553.5	527.9	0.0
18	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	567.0	624.0	0.0
19	0.0	0.0	0.0	0.0	572.0	455.5	502.6	511.4	533.4	495.5	570.3	561.7	607.5	517.9
20	0.0	0.0	0.0	0.0	471.0	0.0	474.0	0.0	613.0	594.0	0.0	628.0	518.0	430.0

- query

```
def q127(factors=factors, periods=None):
    #Nombre moyen de EPVotesFor1-2
    init_question="Nombre moyen de EPVotesFor1-2"
    variable="votes_for_"
    #get the factors specific to the question and update the periods (fr to us format)
    factors_question, periods=prepare_query(factors, periods)
    #the two variables cannot be null or equal to zero at the same time
    exclude_vars_acts={variable+"1": 0, variable+"2": 0}

    #for each factor
    for factor, question in factors_question.iteritems():
        question=init_question+question
        res=init(factor)
        res=get(factor, res, variable=variable+"1", variable_2=variable+"2",
exclude_vars_acts=exclude_vars_acts, periods=periods)
        write(factor, question, res, percent=1, periods=periods)
```

- explanation
  - we use *exclude_vars_acts* to exclude all the acts with both *votes_for_1* and *votes_for_2* empty.
  - We want to compute the average for *votes_for_1 +votes_for_2* **:** we use the *variable* and *variable_2* parameters to compute this average. For each act, if one of the two values is empty, we don't compute the average but simply take the non empty value.
  - We don't want to compute a *percentage*, then we pass *percent=1* to the write function.

- *sql* check for the cs “05” and the year 2012:

```
select
(
	select sum(res) from
	(
	select sum(IF(votes_for_1 is null or votes_for_1=0, votes_for_2, if(votes_for_2 is null
or votes_for_2=0, votes_for_1, (votes_for_1+votes_for_2)/2))) as res
	from europolix.act_act
	where validated=2 and releve_annee=2012 and (votes_for_1>0 or votes_for_2>0)
	and code_sect_1_id in (select id from europolix.act_codesect where code_sect like
'05%')
	union all
	select sum(IF(votes_for_1 is null or votes_for_1=0, votes_for_2, if(votes_for_2 is null
or votes_for_2=0, votes_for_1, (votes_for_1+votes_for_2)/2))) as res
	from europolix.act_act
	where validated=2 and releve_annee=2012 and (votes_for_1>0 or votes_for_2>0)
	and code_sect_2_id in (select id from europolix.act_codesect where code_sect like
'05%')
	union all
	select sum(IF(votes_for_1 is null or votes_for_1=0, votes_for_2, if(votes_for_2 is null
or votes_for_2=0, votes_for_1, (votes_for_1+votes_for_2)/2))) as res
	from europolix.act_act
	where validated=2 and releve_annee=2012 and (votes_for_1>0 or votes_for_2>0)
	and code_sect_3_id in (select id from europolix.act_codesect where code_sect like
'05%')
	union all
	select sum(IF(votes_for_1 is null or votes_for_1=0, votes_for_2, if(votes_for_2 is null
or votes_for_2=0, votes_for_1, (votes_for_1+votes_for_2)/2))) as res
	from europolix.act_act
	where validated=2 and releve_annee=2012 and (votes_for_1>0 or votes_for_2>0)
	and code_sect_4_id in (select id from europolix.act_codesect where code_sect like
'05%')
	) a1
)
/
(
	select sum(res) from
	(
		select count(*) as res
		from europolix.act_act
		where validated=2 and releve_annee=2012 and (votes_for_1>0 or votes_for_2>0)
		and code_sect_1_id in (select id from europolix.act_codesect where code_sect like
'05%')
		union all
```

```
            select count(*) as res
            from europolix.act_act
            where validated=2 and releve_annee=2012 and (votes_for_1>0 or votes_for_2>0)
            and code_sect_2_id in (select id from europolix.act_codesect where code_sect like
'05%')
            union all
            select count(*) as res
            from europolix.act_act
            where validated=2 and releve_annee=2012 and (votes_for_1>0 or votes_for_2>0)
            and code_sect_3_id in (select id from europolix.act_codesect where code_sect like
'05%')
            union all
            select count(*) as res
            from europolix.act_act
            where validated=2 and releve_annee=2012 and (votes_for_1>0 or votes_for_2>0)
            and code_sect_4_id in (select id from europolix.act_codesect where code_sect like
'05%')
        ) a2
) as final_res;
```

Pourcentage d'actes adoptés en 1ère lecture parmi les actes de codécision

This is the query q74*()*. Let's run it on the *periods* factor.

- output

```
Pourcentage d'actes adoptés en 1ère lecture parmi les actes de codécision, par période
Période 1   Période 2   Période 3   Période 4
    2.344      29.679      67.931      83.925
```

- query

```
def q74(factors=factors, periods=None):
    init_question="Pourcentage d'actes adoptés en 1ère lecture parmi les actes de codécision"
    filter_vars_acts_ids={"no_unique_type": "COD"}
    check_vars_acts={"nb_lectures": 1}
    #get the factors specific to the question and update the periods (fr to us format)
    factors_question, periods=prepare_query(factors, periods)

    #for each factor
    for factor, question in factors_question.iteritems():
        question=init_question+question
        res=init(factor)
        res=get(factor, res, Model=ActIds, filter_vars_acts_ids=filter_vars_acts_ids,
```

```
check_vars_acts=check_vars_acts, periods=periods)
    write(factor, question, res, periods=periods)
```

- explanation
  - we use *filter_vars_acts_ids* to filter the codecision acts only. *no_unique_type* comes from the *ActIds* model:

    ```
    filter_vars_acts_ids={"no_unique_type": "COD"}
    ```
  - Among the codecision acts, we want the percentage of acts adopted in first lecture. We use *check_vars_acts; nb_lectures* comes from the *Act* model:

    ```
    check_vars_acts={"nb_lectures": 1}
    ```

- *sql* check for the period number 4:

```
select
(
    select count(*)*100
    from europolix.act_act
    where validated=2 and releve_annee<=2013 and adopt_conseil between "2008-9-15"
and "2013-12-31" and id in
    (select act_id from europolix.act_ids_actids where src="index" and
no_unique_type="COD") and nb_lectures=1
)
/
(
    select count(*)
    from europolix.act_act
    where validated=2 and releve_annee<=2013 and adopt_conseil between "2008-9-15"
and "2013-12-31" and id in
    (select act_id from europolix.act_ids_actids where src="index" and
no_unique_type="COD")
) as final_res;
```

Durée de la procédure (= Moyenne DureeTotaleDepuisTransCons ET DureeProcedureDepuisTransCons)

This is the query q128*()*. Let's run it on the *periods* factor.

- output

```
Durée moyenne de la procédure (= Moyenne DureeTotaleDepuisTransCons + DureeProcedureDepuisTransCons), par période
Période 1 Période 2 Période 3 Période 4
   375.55    445.329    400.157    498.324
```

- query

```
def q128(factors=factors, periods=None):
    #Durée de la procédure (= Moyenne DureeTotaleDepuisTransCons ET DureeProcedureDepuisTransCons)
    init_question="Durée moyenne de la procédure (= Moyenne DureeTotaleDepuisTransCons + DureeProcedureDepuisTransCons)"
    variables=("duree_tot_depuis_trans_cons", "duree_proc_depuis_trans_cons")
    filter_vars_acts={variables[0]+"__gt": 1, variables[1]+"__gt": 1}
    #get the factors specific to the question and update the periods (fr to us format)
    factors_question, periods=prepare_query(factors, periods)

    #for each factor
    for factor, question in factors_question.iteritems():
        question=init_question+question
        res_1=init(factor)
        res_2=init(factor)
        res_1=get(factor, res_1, variable=variables[0], filter_vars_acts=filter_vars_acts, periods=periods)
        res_2=get(factor, res_2, variable=variables[1], filter_vars_acts=filter_vars_acts, periods=periods)
        write(factor, question, res_1, res_2=res_2, percent=1, periods=periods, query="1+2")
```

- explanation

  - we use *res_1* to compute the average of *duree_tot_depuis_trans_cons* and *res_2* to compute the average of *duree_proc_depuis_trans_cons*. In this situation, we don't take into account acts where at least one of the two durations is empty. If we wanted to take into account acts where at least one of the two durations is not empty, we would use only *res* and then the *variable* and *variable_2* parameters (see Nombre moyen de EPVotesFor1-2).
  - *query="1+2"* in the *write* function tells the program to add the two values to compute the *average* (normal average computation)

- *sql* check for the period number 1:

```
select avg((duree_tot_depuis_trans_cons+duree_proc_depuis_trans_cons)/2)
from europolix.act_act
where validated=2 and adopt_conseil between "1996-01-01" and "1999-09-15"
and duree_tot_depuis_trans_cons>1 and duree_proc_depuis_trans_cons>1;
```

Now we know how to create a query. Let's see how to call a query and execute it.

Call a query

The queries are called from the *queries.py* file (see queries.py). As seen previously, to call the query number 2, you just write:

```
q2()
```

Since queries are stored in other files (see query folder), you must indicate the name first. The query number 2 is located in the *acts.py* file and thus is called that way:

```
acts.q2()
```

As a good practice, we always pass two parameters to the query: *factors* and *periods* (see Parameters). The query is then called:

```
factors=[...] #fill the factors to analyze, at least one
periods=(...) #fill the periods to analyze to analyze the periods factor
acts.q2(factors=factors, periods=periods)
```

Run a statistical analysis

To use the module and run a query, there is only one command file to execute: it's the *queries.py* file. As any django command, you run the file as below:

```
python manage.py queries
```

Now we know the use of each file, let us see which ones are required to add a new variable to retrieve.

Update the eurlex or oeil url

The *eurlex* and *oeil* urls used for the project are written in the *common/config_file.py* file (see Common module).

Each time the *eurlex* or *oeil* url change, you must update their urls in the *config_file.py* file:

- change the *url_eurlex* variable to update the *eurlex* url for the acts data; change *url_text_act_html* and *url_text_act_pdf* to update the *eurlex* url for the text of the acts.
- change the *url_oeil* variable to update the *oeil* url for the acts data.

Add a new variable

This section explains how to add a new variable to retrieve from a specific source. This can happen when experts want to analyze a new variable from *eurlex or oeil*.

In this part, we will take the example of a string variable called ***my_var*** to extract from the ***eurlex*** website.

Update the model

In the act/models.py file, add a line in the *Act* model:
my_var=models.CharField(max_length=50, blank=True, null=True, default=None)
Then you need to synchronize the database with the **python manage.py syncdb** command.
If the table is already created and if you have Django 1.6 or earlier, the table will not be updated. If the table contains no data, delete it and then run the command again. If it contains data, add a column manually with an *sql* query through your dbms.

Update the form (optional)

If the field requires a special validation, such as the use of a regex, you can add a line in the *ActForm* class of the act/forms.py file:

Retrieve the variable

In the act/get_data_eurlex.py file, create a function to fetch the *my_var* variable from the *eurlex* url.

```
def get_my_var(soup):
    my_var=...
    ...
    return my_var
```

Use the *soup* variable and the *Beautifulsoup* library to extract what you need. You can also use *regex* and *python* string tools.
Then, still in the same file, call the function you just created. In the function get_data_eurlex*(soup)*, add these two lines:

```
data['my_var']=get_my_var(soup)
print "my_var:", data['my_var']
```

Update the view

In the *view.py* file of the act application, update the *init_context* function: add "*my_var*" at the end of the list of the *context["vars_eurlex"]* variable (this function lists all the variables to display in the template, grouping them by source).

Display the proper name

There can be a difference between the variable name used by the program and the one displayed or exported. To this end, you need to add the variable in the act/var_name_data.py file. Add this line: var_name["my_var"]="NameToDisplay"

Update the export functionality

- If the variable belongs to the Act or ActIds model, there is nothing to change.
- If it is linked to the Act model with a *one-to-many* or *many-to-many* relationship, you must update the *export view*:
  - update the *get_headers* function to create a column for the variable in the export file.
  - update the get_save_acts function to fill the column with the variable value for each act. You must follow the ordering you chose in *get_headers* for the variable.

That's all! The variable has now been added and can be displayed anytime you go on the the Act data validation page.

Now we know (almost) everything concerning the project, let's see if we can make it work for real :).

Run the project on a server

This section explains how to run the project on a server.

Install the project

The source code of the project is hosted on github at the following address: https://github.com/EPX/epx2013.

Click on the *README.md* file to open it. This file contains the instructions to deploy the project on another machine.

Follow those instructions to set up the project on your machine.

You might need to change the permissions of the files as well as their ownership to make it work.

Administration access

There are three groups (stored in the database) which grant different levels of access:

- the first group for standard staff gives access to the Act ids, Attendance and Act data validation page as well as the History page. The users belonging to this group cannot access the administration tool. Behind the scene, no group was created for those users, so they do not belong to any group.
- The second group is the advanced staff. They have access to everything the standard staff can access + the Import page (some functionalities only) + the Export page. Internally, they belong to the *import_export* group.
- The last group is the administration group. They have access to everything that the advanced staff can access + all the functionalities of the Import page + Administration page (to perform administrative tasks). In the database, they are part of the *admin* group.

If you want to create a superuser of the project, run the command ***python manage.py createsuperuser***.

To finish with this manual, let us see now the tasks that are still to perform.

TO DO

This sections deals with several tasks not yet realized.

What could be done to improve the website is :

- **In October 2015 Django 1.4 LTS will no longer be supported. It is strongly advised to install the latest LTS version, Django 1.8 and to use the last version of python 3.x (python 2.7 Is currently used). Some parts of the source code will have to be rewritten (the 2to3 package can be used to easily port the application to python 3.x).**
- The project currently uses the version 2.3.3 of the css framework *Bootstrap*. This version is no longer supported and therefore should be upgraded to the newest one (currently 3.3.2).
- About the ministers' attendance validation form, it currently uses the *ImportMinAttend* model of the *import_app* application. Or this model is a temporary model and should be used to import a miisters' attendance file only (from the import functionality). Instead, the form should be based on the *MinAttend* model of the *Act* application (but not used cause this model uses foreign keys, quite hard to handle in the form).
- Still about the Ministers' attendance validation form, *ajax* should be used to have a smoother and faster browsing (partially developed but not used because the step to handle errors in a formset with *ajax* is missing).
- Act fields should be renamed in English (see the *var_name_ids* and *var_name_data* files of the *act_ids* and act applications).
- Explain all the *js* / *jQuery* files of the project (used for a smoother experience when *javascript* is activated).
- Obviously, the source code and its documentation can always be improved.