Preflib 2.0

Simon Rey

August 26, 2021

Contents

Chapter 1

Developing PrefLib

In this chapter, we detail the inner structure of PrefLib. We will first focus on the folder structure, explaining the role of each file. The second part of this chapter is devoted to the structure of the database.

The aim of this chapter is to provide all the necessary information to someone who would like to develop further the website. If you are only interested in maintaining the website and do some small changes, the second chapter might be more interesting to you.

1.1 Folder Structure

The folder structure of the project follows that of a typical Django project with one application. The overall Django project is called `preflib` and the main, and only, django application is called `preflibApp`. The overall folder structure can be found in Figure 1.1. In what follows, we give an explanation for each folder and files.

We start by detailing the `preflib` folder.

preflib

The `preflib` folder contains the files that have an impact on the entire project. This is the highest folder in the Django hierarchy. Files in there are mainly used to set up the global parameters of the project.

settings.py This is the main files for the global settings. Among other things, you will find there the settings for the database, the location of the static files, the debugging mode, the installed applications, ... Not that this file is not on the git for security reasons.

urls.py Use this file to set up the global rules for urls. Whenever a request passed over to Django, this file is used to decide where to send the request next. Handlers for the errors (404, 500, etc...) are also defined there.

wsgi.py This is only used to set up the connection between Django and whatever WSGI tool is used (uWSGI, Gunicorn, Passenger). If you did not understand the previous sentence, you will most likely never have to deal with this file.

Let us now move to the `preflibApp` folder.

- preflib
  - settings.py
  - urls.py
  - wsgi.py
- preflibApp
  - management
    - commands
      - adddataset.py
      - generatezip.py
      - initializedb.py
      - updatemetadata.py
      - updatepapers.py
  - preflibtools
    - drawing.py
    - graph.py
    - instance.py
    - properties.py
  - static
    - adminLogo
      - images for the admin menu
    - css
      - nouislider.css
      - style.css
    - data
      - CD
      - ED
      - MD
      - types
    - datatoadd
    - js
      - nouislider.js
    - types
      - dataTypesImages.tex
      - images to present the types
    - crisner.png
    - gitHubLogo.png
    - papers.bib
    - preflibADTCite.txt
    - preflibADTPaper.pdf
    - preflibTrendsComSoCChapter.pdf
    - preflibTrendsComSoCCite.txt
  - templates
    - includes
      - HTML files that can be included
    - preflib
      - main HTML files
    - 400.html
    - 403.html
    - 404.html
    - 500.html
  - templatetags
    - extratags.py
  - admin.py
  - apps.py
  - choices.py
  - forms.py
  - models.py
  - scripts.py
  - urls.py
  - views.py

Figure 1.1: Folder Structure of the Project, some less relevant files have been omitted.

preflibApp

This folder contains all the files related to the `preflibApp` Django application. Because this is the only application we have, it thus contains the entire website. Let us first go through the Python files you can find there, sub-folders will be examined afterwards.

urls.py This file sets the url patterns for the application. It is in there that the link between an URL and the corresponding view is made. It is also in there that the set of all acceptable URLs (that will not return a 404) is defined. When a request is passed over to Django, the URL is first filtered by the `urls.py` file in the `preflib` folder which then calls the url pattern defined in this file to know what view to call for.

views.py In Django, a view is a function that takes an input a request and that return a rendered template (an HTML file). This files gathered all the views. This file is probably the most important one as it includes all the code that is run once a request arrives: updating the database, computing so stuff, filter the entries of the database, etc.

forms.py Django offers a Python class to deal with forms that makes it easier to check whether they have been properly filled in etc... The forms are defined in this file. It, for instance, includes the search form, the login form, etc.

models.py A model in Django is the corresponding Python class to a table in the database. In this files all the models are described in Python. Django then reads through this file and create the database accordingly. The entire database is thus defined in there.

choices.py Choices are lists of constant values that are not meant to change often (they would be in the database otherwise). An example is the set of data category for instance. This files gathered them all.

admin.py In this file, one can register the models so that they appear in the Django admin website.

scripts.py This file defines few useful functions for management purposes.

apps.py This is a file that is used by Django to know the application exists. Do not modify it.

For the rest of this section, we will explore the sub-folders present in the `preflibApp` folder.

preflibApp/templates

In Django, a template is an HTML that can incorporate some Django tags in it. They are the main files containing the HTML code of the website. They all are gathered in the `templates` folder. It contains all the template to render the errors: `400.html`, `404.html`, etc. It also contains several sub-folders used to separate the templates based on their use.

includes This folder gathers all the templates that can be included in some other templates using the {% `include 'template'` %} Django tag. These files are:

- footer.html: the footer displayed at the bottom of a page;
- header.html: the header displayed at the top of a page;
- htmlHeaderContent.html: the content of the HTML header that is shared with all pages;
- metadataCategorySearch.html: the code used to render a search widget in the search page;
- paginator.html: the code used to render a paginator in a page.

preflib This folder gathers the templates used to render all the pages of the website. We list them below.

- about.html: the about page
- admin.html: the admin menu
- adminadddataset.html: the admin page for adding datasets
- adminlog.html: the admin page for viewing logs
- adminpaper.html: the admin page for adding a paper
- adminzip.html: the amdin page for zipping the files
- data.html: the data page describing the structure of the data
- dataformat.html: the page explaining the format we use
- datametadata.html: the page describing the metadata
- datapatch.html: the template for rendering a datapatch
- datasearch.html: the search page
- dataset.html: the template for rendering a dataset
- datasetall.html: the template for rendering a category of data
- datatypes.html: the page describing the types
- index.html: the main page of the website
- toolscris.html: the Crisner tool page
- toolsivs.html: the Iterative Voting Simulator page
- toolsskdg.html: the Kidney Dataset Generator page
- userlogin.html: the login page
- usernewaccount.html: the page to create new account
- userprofile.html: the page displaying the profile of a user

preflibApp/templatetags

The `templatetags` folder is used to describe user-defined tags that can then be used in the templates. It contains a single file—`extratags.py`—where the extra tags are defined.

preflibApp/static

The `static` folder contains all the static files that are used (unsurprisingly). This folder is the one considered by the `collecstatic` management command from Django.

adminLogo This folder gathers all the images of the logo used for the admin menu. Note that it might disappear in some later versions.

css This folder gathers all the CSS files. There are currently two of them: `style.css` which is the main CSS file for the style of the website and `nouislider.css` that is used to render the sliders in the search page.

js This folder gathers all the JavaScript files. The only file in there is used for the sliders in the search page.

types This folder gathers the images used to present the types together with the TeX file generating some of them.

crisner.png This image illustrates the Crisner tool.

gitHubLogo.png This image is the GitHub logo displayed in the footer.

papers.bib This bib file is the one read by the management command `updatepapers`. It contains all the bib entries that are then put in the database as "papers using PrefLib".

preflib citations For the ADT paper and the Trends in ComSoC chapter, the pdf and a txt file with the bib information are provided.

preflibApp/management/commands

The `management/commands` folder includes the user-defined commands that one can access using the Django's management tool. Each file in this folder contains a class `Command` which should contain some specific methods. Importantly, once defined and put in this folder, the command can be accessed as any other management command, for instance from the command line using: `python3 manage.py adddataset`.

adddataset.py This command is used to add datasets to the database. Only zip files located in the folder `static/datatoadd` can be retrieved by this command. Two arguments can be passed to the command, either `--file zipfilename` to add only a specific zip file, or `--all` to add all the zip files in the `static/datatoadd` folder to the database.

When adding a dataset, the command unzip the archive in a temporary folder. It then goes through the files to find the info file used to get all the information for the dataset. Then, each file is added to the database, together with its datapatch (if needed). Note that the metadata are not computed when adding a dataset.

generatezip.py This command is used to generate all the zip files served by the website. It first generates the zip files for the dataset, creating the info file with the entry in the database. In a second time, it generates the zip files per type of data.

initializedb.py This command is only run once, at the very beginning, to populate the database with the entries that are needed. It mainly sets up the metadata.

updatemetada.py This command computes the metadata for the data files. Two options can be passed to the command. When used with `--dataset datasetAbbreviation`, only the metadata for the given dataset will be computed. One can also use `--noDrawing` to avoid generating the images for the data file (which takes a lot of time).

For each relevant data file, the command will go through all the *active* metadata (see later what active mean here). Whenever an active metadata applies to the data type of the data file, the corresponding function is called to compute its value. If drawing is allowed, the relevant drawing method is also called.

updatepapers.py This command updates the list of papers that are using Preflib. It reads the bib file `static/papers.bib` and update the database accordingly.

preflibApp/preflibtools

This folder contains the tools that were developed around PrefLib to work with the data. Note that this version is far from what you can find in the PrefLibTools repository on GitHub. These tools are mainly used to analyze the data when computing the relevant metadata.

drawing.py This file contains all the functions that are used to draw the images representing the data.

graph.py This file contains the definition of the `Graph` class that we use to represent graphs and access several useful methods for them.

instance.py This file contains the class used to represent PrefLib instances. The methods for parsing the data file are defined there.

properties.py This file contains a collection of functions used to check whether some properties hold for a given instance. This is the main file used to compute the metadata of the instances.

1.2 Database Structure

In the following, we provide more details about the structure of the database behind Preflib.

Let us first go through all the tables present in the database.

DATAFILE The most fundamental entity for Preflib is the datafile. The DATAFILE table contains a reference to all the datafiles that are in Preflib. The table does not contain the data in itself—it is stored in a file and not in the database—but all the relevant information about it: some basic details and datapatch in which the file is.

DATAPATCH The datapatch is the first level of classification of the datafile. It contains several datafile of different datatype. All the datafiles are based on the same preferences but the representation, the datatype, is different.

DATASET A dataset is a collection of datapatches. The datapatches will typically represent different years of the same election.

DATAPROPERTY A dataproperty is an additional information about a given datafile. It can have to do about the general properties of the data (number of candidates...) or about some more specific structure of the data (single-peakedness...).

METADATA The METADATA table stores all the different metadata available in the system. Their values are in the DATAPROPERTY table.

PAPER This table stores the information about the papers which are using Preflib.

USERPROFILE All the informations about the users that are not in the Django User class are present in this table.

LOG Logs of what is happening in the inside are gathers in this table.

The following figure summarizes the links between the different tables and presents all the elements present in the tables.

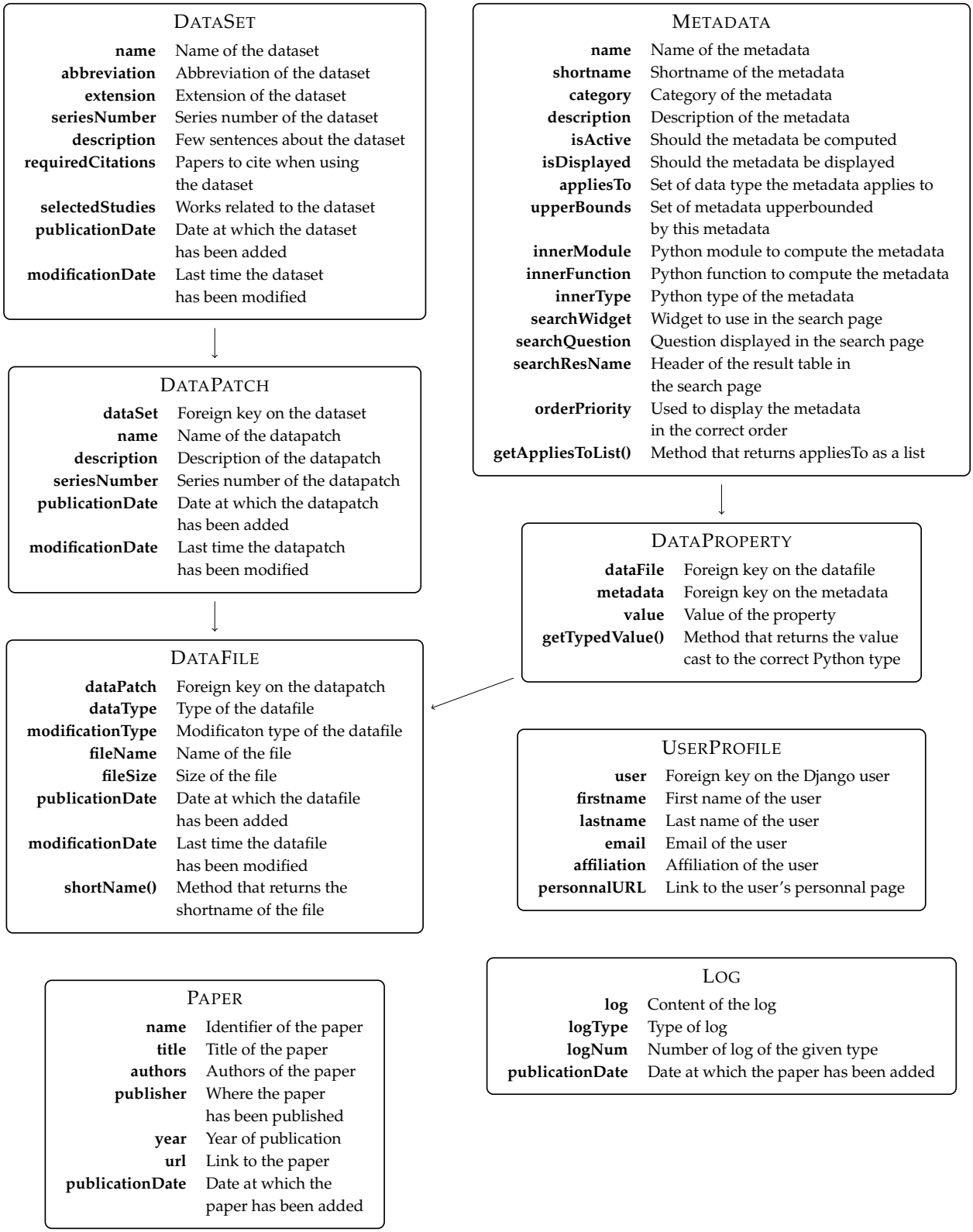

Figure 1.2: Structure of the database

Chapter 2

Maintaining PrefLib