Website Oscar: Computer architecture documentation

Table of contents

1. Introduction

This document presents the technical architecture of the Oscar website. It is strictly destined to the students and members of UCL involved in the course project [LINGI2255] - «Software engineering project» 2017-2018. The following informations concern the version 0.5 (September 2017). The described points in this document assume that you master the vocabulary linked to the framework Django (app, model, …).

2. Technological choices

Oscar is currently developed with the following technological stack:

- python 2.7
- Django 1.11
- PostgreSQL 9.4
- Haml as a HTML pre-processor
- Bootstrap v3.3.5 as a CSS framework
- jQuery v1.11 as well as Angular v1.2.6 on some pages
- git as a version control tool
- mathquill v0.10.0 and jquery-keyboard v1.26.13 are used for the mathematical keyboards
- jsxgraph is used for the answer of graph type (landmark with grid allowing to put points and draw lines, circles, etc.)

- sphinx to generate the code documentation

3. Code conventions et license(s) choice(s)

The **pep8** is mostly respected and followed, except the lines length.

It is highly recommended to use **Opensource** addons (packages, librairies, plug-in, …) with **permissive** licenses.

A permissive license can be distributed under any license of non-copyleft type (gives the rights to use, study, modify and broadcast a source code), but it ensures also an easy and simple distribution. Below a comparative array of the main licenses of copyleft types and non-copyleft types.

Name	Type	Advantages	Constraints	Recommendation
GNU General Public License (GPL)	Copyleft	It is the most popular and known license by the developers	License that contaminates: any software with GNU GPL code becomes GNU GPL	Not recommended
MIT	Non-copyleft	High permissive license	It is particularly applied on small programms	Recommended
BSD	Non-copyleft	Permissive license, with a minimum of restrictions	Do not use the BSD original license that contained an advertising clause today removed	Recommended
Apache	Non-copyleft	Elle est permissive Initially destined to the Apache packages, it is nowadays very popular. It is permissive	It is not compatible with GNU GPL 2	Recommended

In case of need, your responsible can contact Mr. Adam Sébastien, TIC valuer expert at UCL. Email: sebastien.adam@uclouvain.be, Phone: 010 47 24 43.

4. General style of development

Oscar is a young project, that need flexibility, scalability to be able to react quickly to professors feedbacks. We have then to avoid too rigid choices that would block needs that can come promptly. Given the development context. The following choices have been made:

- Promote as much as possible simple code;
- Use build-in abstractions in Django (class-based generic views) when it is relevent as well as the crispy-forms;
- The pages that need a lot of interactivity are created with jQuery and/or Angular;
- django-debug-toolbar and django-extensions are used to ease the development;
- virtualenv/pip are used for the release as well as the development.

5. Code and content organization

5.1 Code structure

- **authentification/**: Django app, contains the code linked to the authentication, uses django.contrib.auth
- **documentation/**: Code documentation (models, views, ...) handled by sphinx
  - *Makefile*: allows to generate the documentation with the target *html*
  - *doc_example.py*: contains the nomenclature to respect in order to document the code, as well as all the possibilities that sphinx offers for the formatting of the different elements
  - **source/**: contains the .rst configuration files to recover the files architecture of the Django project. It contains also the *conf.py* file, that configure the documentation generation
  - **builds/**: will contain the HTML files which contain the generated documentation once it is generated
- **end_test_poll/**: Django app: code linked to the poll proposed to the student at the end of their first test
- **examinations/**: Django app, contains the code linked to the tests and the exercises
  - *models.py*: When a new exercise is created, a *Context* is created. (when we speak about *Context*, exercise or exercise, it is that model that is designated). A C*ontext* is a container of one or several *Questions*

A *Question* contains a description, an answer, a source and an indication (whose the content is visible only to the professors). The links between the *Contexts* and their *Question(s)* are defined in the model *List_question*

The answers stored in the *Questions* are the corrections, not to be mixed up with the answers of the *Answer* model, that contains a field raw_answer, and that indicates a set of answers corresponding to a Context for a particular student

- **jsxgraph/**: *jsxgraph* configuration for the exercises of graphical type
- **oscar/**: folder of the Django project for Oscar, that contains *settings.py*
- **planification/**: empty app, not used yet
- **promotions/**: Django app that contains the majority of the code relative to the professor interface.
- **resources/**: Django app that concerns the pedagogical resources (documents, videos, links, texts, …) linked to the *Skills* and to the *CodeR*. Careful not to mix up the code resources (*CodeR*) and the pedagogical resources. To avoid this ambiguity, we will not use the word «resource» to designate a *CodeR*
- **skills/**: contains all the models linked to the skills
- **stats/**: Django app that contains some views to display statistics about the website utilisation (number of classes, students, exercises, ...)
- **student/**: Django app that contains the majority of the code linked to the student interface.
- **templates/**: contains all the project templates
- **test_from_class/**: Django app that contains the code linked to the free evaluations («évaluations libres»), allowing the professor to encode online the state of the skills evaluated in class
- **test_online/**: Django app that contains the code relative to the online tests
- **users/**: Django app for the website users (*Professor* and *Student*)
- **README.md**: contains the installation instructions for the project
- **requirements-oscar2.txt**: project dependences

5.2 Static files

The static files (CSS, javascript, images, ...) are stored in the concerning apps. Ideally, we need to store all of them in a folder at the root as the templates:

- **oscar/static/**: contains all the static files (images, libraries javascript) destined to be displayed on all the website.
- *application*/**static/**: contains the javascript, CSS, images, … relative to the interface used by the *application*. Examples: *examinations/***static**, *promotions/***static**…

5.3 Templates and URLs organization

With some exceptions, all the templates files as well as the Urls try to keep a correspondence between the location on the system file and the URL.

For example: the template used at the adress */professor/lesson/2/test/from-class/4/* will be */professor/lesson/test/from-class/detail.haml*

Following the *CRUD* principles and *class-based generic views* of Django, the following correspondence is respected between the template names and the URLs:

- / → list.*haml*
- */<id>* → *detail.haml*
- */add* → *add.haml*
- */<id>/update* → *update.haml*

In the other cases, the template name is generally close or identical at the end of the URL.

The function names in the views (*views.py*) as well as the URL names (Django name of the URLs) generally respect this convention.

6. Database

Here are some notes about the database:

The part relative to the tests is the most complicated part of the schema:

- *BaseTest* is the model from all the tests are based;
- *Test* concerns the online tests (sometimes called «hybrid tests» when in a test some skills are evaluated online and the remaining are evaluated on paper);
- *TestSkillFromClass* concerns the free evaluations.

When a new online test is created:

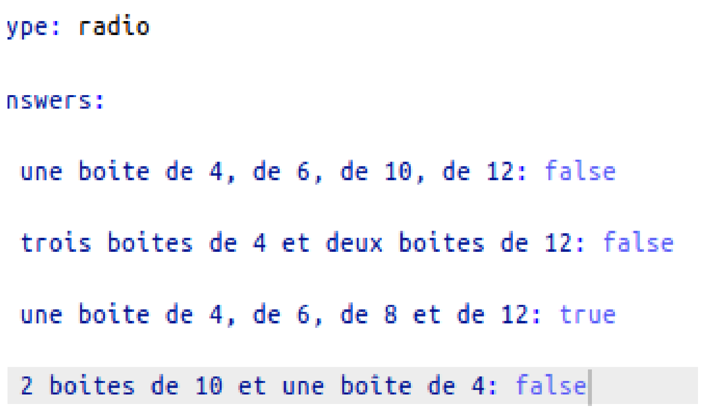
- a *TestStudent* is created for each student, it is linked to the *Test*;
- for each tested skill by *Test*, a *TestExercice* is created and makes the link between the *Test* and the exercise (*Context*) when it is available;

- *Answer* stores the student’s response and is linked with a *TestExercice* and a *TestStudent*. Theoretically, it is possible to have several answers to a *TestExercice*, but this functionality is not used.

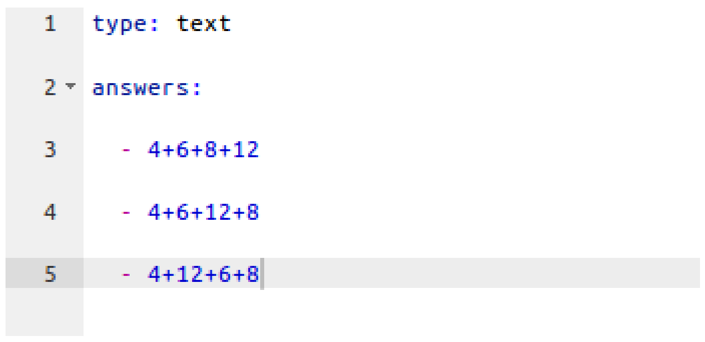
7. Exercises structure

For historical reasons, the answers (corrections) of the exercises are encoded in YAML format.

Answers examples:



```
ype: radio

nswers:

 une boite de 4, de 6, de 10, de 12: false

 trois boites de 4 et deux boites de 12: false

 une boite de 4, de 6, de 8 et de 12: true

 2 boites de 10 et une boîte de 4: false
```



```
type: text

answers:

  - 4+6+8+12

  - 4+6+12+8

  - 4+12+6+8
```

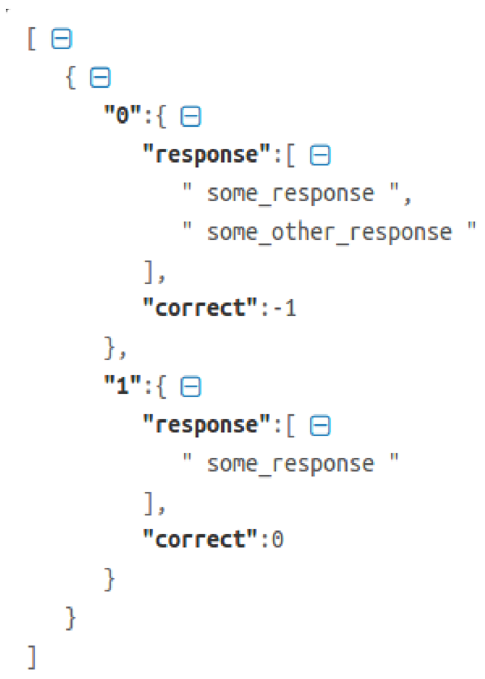
A *Context* must have at least one question. Every *Question* possesses a type (stored in the answer field). The answers must respect a format linked to this type: list, dictionary (*hashmap)* or a more complex structure if it necessary (especially for the *graph* type).

In September 2017, here is the list of the *Question* types available:

- **text**: a free field (*<input type="text">*) in which the student must provide an answer (which will be compared with the correct answers)
- ***math-simple* et *math-advanced***: A *mathquill* field (generation of symbols LaTeX for the web) for answers of mathematic type with a simple keyboard for the mathematical basic operations or advanced if more complex operations are needed
- ***radio***: One correct answer among a list (*<input type="radio">*)
- ***checkbox***: Multiple choice question (MCQ) (*<input type="checkbox">*)

- ***graph***: answers of graphical type, nowadays only has the possibility to define a series of points on a graph, uses *jsxgraph*. This type is declined in sub-types, that encompasses for the time being only the *point*.
- ***professor***: questions that will be assed by a professor. The question are thus not corrected online, and so contain no correct/incorrect pre-recorded answer in the field answer of *Question*.

Here below the structure (JSON) of an answer (in reality: one or several answer(s), because it corresponds to a *Context*) provided by a student.



```
[ ⊟
   { ⊟
      "0":{ ⊟
         "response":[ ⊟
            " some_response ",
            " some_other_response "
         ],
         "correct":-1
      },
      "1":{ ⊟
         "response":[ ⊟
            " some_response "
         ],
         "correct":0
      }
   }
]
```

The answers are ordered in a dictionary whose the keys are numbers starting from 0, and corresponding to the order of the *Questions* of the *Context* to which the student answered. This dictionary is itself contained in a list (not exploited yet, but that allows to extend the structure if needed).

Each answer is a dictionary containing a «response» and a correction («correct»). The «response» is a list, and its structure varies according to the *Question* type to which it responds (even if the list contains only one element, the «response» will always be stored under the form of a list). The correction can take three values: -1 if not assessed (useful when the question is not corrected yet by the professor), 0 if it is false, 1 if it is correct.

8. References

- Oscar https://oscar.education/ (project presentation website)

- Django https://docs.djangoproject.com/en/1.11/
- django-bootstrap3 https://django-bootstrap3.readthedocs.io
- Haml http://haml.info/, django-hamlpy https://github.com/nyaruka/django-hamlpy
- Bootstrap https://getbootstrap.com/
- YAML http://yaml.org/ et https://en.wikipedia.org/wiki/YAML and PyYAML http://pyyaml.org/wiki/PyYAML
- angular v1 https://angularjs.org/
- crispy-forms https://django-crispy-forms.readthedocs.io/
- git https://git-scm.com/
- jsxgraph https://jsxgraph.uni-bayreuth.de/
- mathquill http://mathquill.com/
- jquery-keyboard https://mottie.github.io/Keyboard/
- sphinx http://www.sphinx-doc.org/en/stable/