

Configurations, l18n y Debug - Tutorial #3.

Juan Cabral - jbc.develop@gmail.com

Jan, 2018

settings.py.

Your settings can be found in settings.py. Here are explanations of a few oTree-specific settings. Full info on all Django's settings can be found:

<https://docs.djangoproject.com/en/1.8/ref/settings/>

settings.py.

SESSION_CONFIGS

- ▶ In settings.py, add an entry to `SESSION_CONFIGS` like this (assuming you have created apps named `my_app_1` and `my_app_2`):

```
{  
    'name': 'my_session_config',  
    'display_name': 'My Session Config',  
    'num_demo_participants': 2,  
    'app_sequence': ['my_app_1', 'my_app_2'],  
},
```

- ▶ Once you have defined a session config, you can run `otree resetdb`, then `otree runserver`, open your browser to the admin interface, and create a new session. You would select “My Session Config” as the configuration to use.

settings.py.

SESSION_CONFIG_DEFAULTS

- ▶ If you set a property in `SESSION_CONFIG_DEFAULTS`, it will be inherited by all configs in `SESSION_CONFIGS`, except those that explicitly override it.
- ▶ The session config can be accessed from methods in your apps as:

```
self.session.config['participation_fee']
```

`settings.py`.

DEBUG

- ▶ You can turn off debug mode by setting the **environment variable** `OTREE_PRODUCTION` to 1, or by directly modifying `DEBUG` in `settings.py`
- ▶ If you turn off `DEBUG` mode, you need to manually run `otree collectstatic` before starting your server-
- ▶ Also, you should set up Sentry to receive email notifications of errors.

settings.py.

REAL_WORLD_CURRENCY_CODE

- ▶ If you have a value that represents an amount of currency (either points or dollars, etc), you should mark it with `c()`, e.g.

```
c(1) + c(0.2) == c(1.2)
```

- ▶ The advantage is that when it's displayed to users, it will automatically formatted as **\$1.20** or **1,20 €**, etc., depending on your `REAL_WORLD_CURRENCY_CODE` and `LANGUAGE_CODE` settings.
- ▶ Money amounts are displayed with 2 decimal places by default; you can change this with the setting `REAL_WORLD_CURRENCY_DECIMAL_PLACES`. (If you change the number of decimal places, you must resetdb.)

settings.py.

USE_POINTS

- ▶ Sometimes it is preferable for players to play games for points or “experimental currency units”, which are converted to real money at the end of the session.
- ▶ You can set `USE_POINTS = True` in `settings.py`, and then in-game currency amounts will be expressed in points rather than dollars or euros, etc.

c(10) is displayed as 10 points.

- ▶ To change the exchange rate to real money, go to `settings.py` and set `real_world_currency_per_point` in the session config.
- ▶ For example, if you pay the user 2 cents per point, you would set

settings.py.

USE_POINTS

- ▶ Points are integers by default.
- ▶ You can change this by setting `POINTS_DECIMAL_PLACES = 2`, or whatever number of decimal places you desire.
- ▶ If you change the number of decimal places, you must resetdb.
- ▶ If you switch your language setting to one of oTree's supported languages, the name "points" is automatically translated, e.g. "puntos" in Spanish.
- ▶ To further customize the name "points" to something else like "tokens" or "credits", set `POINTS_CUSTOM_NAME`, e.g.

```
POINTS_CUSTOM_NAME = 'tokens'
```


`settings.py`.

SENTRY_DSN

- ▶ Sentry service which can log all errors on your server and send you email notifications.
- ▶ Sentry is necessary because many errors are not visible in the UI after you turn off debug mode.
- ▶ You will no longer see Django's yellow error pages; you or your users will just see generic:

Server Error (500)

There are several ways to find the cause of the issue:

- Set the `OTREE_PRODUCTION` environment variable to `0` and reload this page
- Look at your Sentry messages (see the docs on how to enable Sentry)
- Look at the server logs

- ▶ You need to check the sentry documentation to understand this setting.

settings.py.

AUTH_LEVEL

- ▶ It's somewhat preferable to set the environment variable `OTREE_AUTH_LEVEL` on your server, rather than setting `AUTH_LEVEL` directly in `settings.py`.
- ▶ When you first install oTree, The entire admin interface is accessible without a password.
- ▶ However, when you are ready to deploy to your audience, you should password protect the admin.
- ▶ If you are launching an experiment and want visitors to only be able to play your app if you provided them with a start link, set the environment variable `OTREE_AUTH_LEVEL` to `STUDY`.
- ▶ To put your site online in public demo mode where anybody can play a demo version of your game (but not access the full admin interface), set `OTREE_AUTH_LEVEL` to `DEMO`.
- ▶ If you don't want any password protection at all, leave this variable unset/blank.

`settings.py.`

ROOMS

- ▶ **DONE in Day 4**

settings.py.

ADMIN_USERNAME, ADMIN_PASSWORD

- ▶ For security reasons, it's recommended to put your admin password in an environment variable, then read it in settings.py like this:

```
ADMIN_PASSWORD = environ.get('OTREE_ADMIN_PASSWORD')
```

- ▶ If you change ADMIN_USERNAME or ADMIN_PASSWORD, you need to reset the database.

`settings.py`.

`DEMO_PAGE_TITLE`

- ▶ The title of the demo page

`DEMO_PAGE_INTRO_HTML`

- ▶ The HTML in the sidebar of the demo page

Localization

- ▶ oTree's participant interface has been translated to the following languages:
 - ▶ Chinese (simplified)
 - ▶ Dutch
 - ▶ French
 - ▶ German
 - ▶ Hungarian
 - ▶ Italian
 - ▶ Japanese
 - ▶ Korean
 - ▶ Norwegian
 - ▶ Russian
 - ▶ Spanish
- ▶ This means that all built-in text that gets displayed to participants is available in these languages.

Localization

- ▶ This localization includes things like:
 - ▶ Form validation messages
 - ▶ Wait page messages
 - ▶ Dates, times and numbers (e.g. “1.5” vs “1,5”)
- ▶ So, as long as you write your app’s text in one of these languages, all text that participants will see will be in that language.
- ▶ For more information, see the Django documentation on translation and format localization.
- ▶ However, oTree’s admin/experimenter interface is currently only available in English, and the existing sample games have not been translated to any other languages.

Localization

Changing the language setting

- ▶ Go to `settings.py`, change `LANGUAGE_CODE`, and restart the server. For example:

```
LANGUAGE_CODE = 'fr' # French  
LANGUAGE_CODE = 'zh-hans' # Chinese (simplified)
```


Localization

Writing your app in multiple languages

- ▶ You may want your own app to work in multiple languages.
- ▶ For example, let's say you want to run the same experiment with English, French, and Chinese participants.
- ▶ For this, you can use Django's translation system.

Localization

A quick summary:

- ▶ Go to settings.py, change LANGUAGE_CODE, and restart the server.
- ▶ Create a folder locale in each app you are translating, e.g. public_goods/locale.
- ▶ If you forget to create this folder, the translations will go into your root directory's locale folder. At the top of your templates, add

```
{% load i18n %}
```

- ▶ Then use `.`. There are some things you can't use inside a blocktrans, such as variables containing dots

```
{% blocktrans trimmed %}...{% endblocktrans %}
```

Localization

A quick summary:

- ▶ If you have localizable strings in your **Python code**, use `gettext`.
- ▶ Use `makemessages` to create the **.po** files in your app's locale directory:

```
$ django-admin makemessages -l fr  
$ django-admin makemessages -l zh_Hans
```

- ▶ Edit the `.po` file in Poedit

Localization

A quick summary:

- ▶ Run `compilemessages` to create `.mo` files next to your `.po` files.

```
$ django-admin compilemessages
```

- ▶ If you localize the files under `_templates/global`, you need to create a directory `locale` in the root of the project.

References

- ▶ <http://otree.readthedocs.io/en/latest/>