



D07 - Python-Django Training

Django - Advanced

Summary: Today, we will discover some of Django's advanced functionalities.

Contents

I	Preamble	2
II	Ocaml piscine, general rules	5
III	Today's specific rules	7
IV	Exercise 00	8
V	Exercise 01	10
VI	Exercise 02	12
VII	Exercise 03	14
VIII	Exercise 04	16
IX	Exercise 05	17
X	Exercise 06	18

Chapter I

Preamble

Frameworks' heaven

I wanted to build a little shelf to keep my condiments handy.

Being familiar to carpentry, I had an idea of the supplies I needed: some wood, and a basic set of tools. A rule, a saw, a level and a hammer.

I actually would need those tools if I wanted to build a whole house. So I went to a hardware store and asked the seller where I could find a hammer.

“- A hammer?”, he answered. “Nobody buys hammers nowadays, you know. They’re old school, see?”

Shocked, I asked him why.

– “Well, hammers, you know, there are so many different types. American ones, European ones. What would happen if you were to buy a hammer that didn’t suit your future needs and have to buy another one later? Fact is people today, they want a tool of all trades they can use everyday of their lives.”

– “Sounds legit. Can you show me a universal hammer by any chance?”

– “We don’t sell those anymore. They’re obsolete.”

– “Really? I thought you just told me the universal hammer was the future...”

– “As it is, if you make just one multitask hammer, it won’t work well for any specific task. Hit a nail with a sledgehammer is far from effective. And if you want to kill your girlfriend, nothing beats the American hammer.”

– “Obviously! So... If no one is to buy universal hammers, and you don’t sell specialized hammers anymore... What kind of hammer do you sell?”

– “Well, we don’t.”

- “So...”
- “Our studies show people really don’t need any universal hammer. It’s always better to have the right hammer for the job. So we started selling hammer factories that can build the hammer you need when you need it. You just have to fill it with workers, run the machines, buy the basic supplies, pay your taxes and you get exactly the hammer you need in a wink.”
- “But I don’t want to buy a hammer factory...”
- “Good. We don’t sell those anymore.”
- “But you’ve just told me...”
- “We found out most people, they don’t need the whole shebang. For instance, some will never need an American hammer (Maybe they don’t have any ex-girlfriend. Or maybe they killed them with an ice-pick). So we have no reason to sell a hammer factory that can craft any kind of hammer to anyone.”
- “Sure”
- “So, instead, we started selling hammer blue prints and instructions so our customers can craft their own hammers from scratch. That way they could craft the perfect tool for the perfect job.”
- “Lemme guess... You don’t sell them anymore...”
- “Of course not! People don’t want to craft a whole machine just to craft a couple of hammers. There are professional builders for these kinds of machines! That’s what I keep telling ’em!!”
- “And you’re so right...”
- “I am. So we stopped selling these instructions and started selling machines that build the machines. Professionally crafted so you don’t have to worry about this part. You have your own machine, crafting your hammer crafting machine, following your own designs.”
- “That doesn’t seem too...”
- “I know what you’re about to say!! We don’t sell them anymore actually. As it appeared, very few people bought the hammer crafting machine crafting machine so we came up with a new solution.”
- “Hum.”
- “We took the necessary time to assess our technical infrastructure and found out

that people got frustrated at the idea of operating a hammer crafting machine crafting machine. It became redundant if you got to use a rule crafting machine crafting machine and a level crafting machine crafting machine. And let's not talk about the loss of wood this whole thing ended producing. It sounded way too complicated for someone who just aimed at crafting a simple shelf"

– “No shit...”

– “So this week, we start selling a tool crafting machine crafting machine crafting machine for all types of tools so that your different tool crafting machines crafting machines can be crafted with one unified machine. The machine crafting machine will only craft the crafting machine machine you really need and your machine crafting machines will only craft one crafting machine based on your personalized tools specifications. You will get the hammer you need and the perfect rule for the job by a simple click (you still may have a couple of configuration files for everything to work as expected).

– “So you don't have any hammer? At all?”

– “No. If you're looking for a shelf for your condiments, a real good shelf, high quality, industrial standard and all, you really need something more sophisticated than a simple hammer from your local hardware store.”

– “Well... It takes what it takes. If it's the current trend, I've got to learn to live with my time”

– “Excellent!!”

– “It comes with instructions, right?”

Source : Loosely based on Joel Spolsky's "Why I hate Frameworks"

Chapter II

Ocaml piscine, general rules

- Every output goes to the standard output, and will be ended by a newline, unless specified otherwise.
- The imposed filenames must be followed to the letter, as well as class names, function names and method names, etc.
- Unless otherwise explicitly stated, the keywords `open`, `for` and `while` are forbidden. Their use will be flagged as cheating, no questions asked.
- Turn-in directories are `ex00/`, `ex01/`, ..., `exn/`.
- You must read the examples thoroughly. They can contain requirements that are not obvious in the exercise's description.
- Since you are allowed to use the OCaml syntaxes you learned about since the beginning of the piscine, you are not allowed to use any additional syntaxes, modules and libraries unless explicitly stated otherwise.
- The exercises must be done in order. The graduation will stop at the first failed exercise. Yes, the old school way.
- Read each exercise FULLY before starting it! Really, do it.
- The compiler to use is `ocamlopt`. When you are required to turn in a function, you must also include anything necessary to compile a full executable. That executable should display some tests that prove that you've done the exercise correctly.
- Remember that the special token `";;"` is only used to end an expression in the interpreter. Thus, it must never appear in any file you turn in. Regardless, the interpreter is a powerful ally, learn to use it at its best as soon as possible!
- The subject can be modified up to 4 hours before the final turn-in time.
- In case you're wondering, no coding style is enforced during the OCaml piscine. You can use any style you like, no restrictions. But remember that a code your peer-evaluator can't read is a code he or she can't grade. As usual, big functions are a weak style.
- You will NOT be graded by a program, unless explicitly stated in the subject. Therefore, you are given a certain amount of freedom in how you choose to do the

exercises. However, some piscine day might explicitly cancel this rule, and you will have to respect directions and outputs perfectly.

- Only the requested files must be turned in and thus present on the repository during the peer-evaluation.
- Even if the subject of an exercise is short, it's worth spending some time on it to be absolutely sure you understand what's expected of you, and that you did it in the best possible way.
- By Odin, by Thor! Use your brain!!!

Chapter III

Today's specific rules


- Today, you're gonna develop a single site. It will feature articles a user will be able to consult. Logged-in users will be able to publish new articles or save some in a favorite list.
- You can name this site as you like and give it the focus you prefer (news, fan fictions, erotic novellas...).
- You can choose the database you like as long as it is compatible with Django's native ORM.
- Your repo will have the form of a single Django project. It will not split in exercises, as usual. Each one will add a functionality to the project. The functionality and its implementation will be graded.
- Implementing a "hard" url is STRICTLY prohibited. You must refer to the URL's **name**. Either in the views or the templates.
- Implementing a view in the form of a functions is STRICTLY prohibited. You must only use **generic views**.
- Remember exercises will be evaluated in project's order.
- English is the default language of your site. The content of the database can be anything. If you have a soft spot for ancient greek, go for it.
- You must leave the default administration application.



Don't waste time on what's not required!

Chapter IV

Exercise 00

	Exercise 00
Exercise 00: Model building - Generic Class	
Turn-in directory : <i>ex00/</i>	
Files to turn in :	
Allowed functions :	

Create a new project. You can name it and set the application structure as you like. Think of a logical structure. The easier it is to understand, the easier the evaluation.

Implement the following models:

- **Article:** Content of the article and a few metadata. It must feature the following fields:
 - **title:** Article's title. Character chain 64 max size. Non null.
 - **author:** Article's author. References a record of the User model. Non null.
 - **created:** Creation's complete date and time. Must be automatically filled when created. Non null.
 - **synopsis:** Article's abstract. Character chain. Max size 312. Non null.
 - **content:** The article. It's a text type. Non null.

The `__str()` method must be 'overrode' to send 'title'

- **UserFavouriteArticle:** User's favorite articles. Must feature the following fields:
 - **user:** References a record of the User model. Non null.
 - **article:** References a record of the Article model. Non null.

The `__str__` method must be overridden to send 'title' included in the Article model.

Once all of this is done, we can tackle the real goal of this first exercise.

Using only generic views (except 'View' that you can't inherit directly), you must implement the following functionalities to your site. Each functionality must have its own URL:

Articles: HTML page displaying every field as an HTML table (except the `content`) of every recorded article in the `Article` table).

The table must have a **header** indicating the title of each column.

Home: Mandatory URL: '127.0.0.1:8000'. Redirects to **Articles**

Login: HTML page displaying a POST type form. Logs an logged-out user thanks to a username and a password. In case of an error, the page must display a message describing said error. If successful, the view must redirect to 'Home'.


You must also provide at least five articles examples from three different users. Provide fixtures if necessary. The article's content doesn't matter. Basic 'lorem ipsum' can be enough.



No css formatting is required in this exercise.

Chapter V

Exercise 01

	Exercise 01
Exercise 01: Generic Class again	
Turn-in directory : <i>ex01/</i>	
Files to turn in :	
Allowed functions :	

Using only generic views (except '**View**' that you can't inherit directly), you must implement the following functionalities to your site. Each functionality must have its own URL:

Publications: HTML page displaying as HTML tables the fields '**title**', '**synopsis**' and '**created**', of every article recorded in the '**Article**' model whose user is currently logged-in.

For each article, you also must implement a link which URL must include the article's identification leading to the '**Detail**' functionality of said article.

The table must have a 'header' indicating the title of each column.

Detail: HTML page displaying every field of a given article located in the database. Its identification must be in the URL.

Field disposition is free.

For each article in the **Articles** functionality present in the HTML table from the previous exercise you will also add a link towards this article's '**Detail**'.

Logout: A link logging out a logged in user. You can place the link wherever you want as long as it is visible and accessible. Once logged out, the user is redirected towards '**Home**'.


Favourites: HTML page displaying the titles of the current user's favorite articles as a list of links.

Each link - the URL of which must include the article's identification - must lead to the article's '**Detail**' functionality.

You must provide at least one user with at least two different favorite articles.

Chapter VI

Exercise 02

	Exercise 02
Exercise 02: Generic Class - CreateView	
Turn-in directory : <i>ex02/</i>	
Files to turn in :	
Allowed functions :	

Using only `CreateView`, you must implement the following functionalities to your site. Each functionality must have its own URL:

Register: HTML page featuring a 'POST' type form allowing a logged out user to create a new user account.

The form must require the login, a password and a password confirmation. this form must be accessible to a URL exclusively dedicated to it and ending with 'register'.

Publish: HTML page featuring a 'POST' type form allowing a logged-in user to publish a new article. The 'author' field must not be displayed. It must be completed in the view during the validation process. You have to use a 'form' object created by your view to generate your form (no handwritten `<input>` tag for the fields in your form!)

Add a link towards this functionality in the **Publications** functionality template.

Add to favourite: HTML page containing a 'POST' type form located in the detail page of an article. No field must be visible. The 'article' field must be pre-filled with the current article ID and during validation, 'user' field must be filled with the logged user ID. This mechanism allows to add the current article in the logged user's favorite list.




No css formatting is required in this exercise.



Did you know that Django proposes ready-made forms?

Chapter VII

Exercise 03

	Exercise 03
Exercise 03: Template tags and Filters	
Turn-in directory : <i>ex03/</i>	
Files to turn in :	
Allowed functions :	

In this exercise, you will create a menu that must be visible from EVERY page of the site.

Every link in this menu lead to functionalities **YOU HAVE ALREADY CREATED**. If something is missing, start questioning yourself.

This menu must feature the following elements:

- **Home:** A link to the '**Home**' functionality (that redirects towards '**Articles**', remember?).
- **Last Articles:** A link to the '**Article**' functionality. You can adapt the name of this link according to the theme of you site (but it must be in English!).
- If a user is not registered:
 - **Register:** A link to the '**Register**' functionality.
 - **Login:** '**Login**' functionality. It's a little different here, because it's not just a link, you must include the whole form IN the menu.

This means it will be accessible from every page, not only from the dedicated page you have created.

However, this page must always display error messages if the form is invalid.

- If a user is registered:
 - Favourites: Link to the 'Favourites' functionality.
 - Publications: Link to the 'Publications' functionality.
 - Logged as <user>: Simple text indicating that the user is logged. Of course, <user> must be replaced by the name of the current user.
 - Logout: 'Logout' functionality. Now, you have found a place to set up this link.

Using tags and filters, modify the templates that list every article so that:


- The abstract is reduced to 20 characters maximum. The follow-up must be replaced by suspension points. You must have an example ready to demonstrate it works.
- The list of articles must be sorted out by date, from the newest to the oldest.
- An additional column mentions how long the article has been published for.



No css formatting is required in this exercise.

Chapter VIII


Exercise 04

	Exercise 04
Exercise 04: Bootstrap	
Turn-in directory : <i>ex04/</i>	
Files to turn in :	
Allowed functions :	

Using Bootstrap, give your menus the same CSS formatting as the one in the provided image in today's resources.

Chapter IX

Exercise 05

	Exercise 05
Exercise 05: Internationalization	
Turn-in directory : <i>ex05/</i>	
Files to turn in :	
Allowed functions :	

Translate the whole **Articles** functionality of the site, as well as the menu (which is also visible from here) English depending on the prefix in the URL.

For instance:

If my URL is: `127.0.0.1:8000/en/articles`, the whole content will be in English.

If this URL is: `127.0.0.1:8000/<??>/articles`, the whole content will be in the language indicated in `<this>` section of the address.


You won't have to translate the database content or the site's name.

The site's default language must be English.

Add a link allowing to switch language on this page.

Chapter X

Exercise 06

	Exercise 06
Exercise 06: Testing	
Turn-in directory : <i>ex06/</i>	
Files to turn in :	
Allowed functions :	

Using the framework integrated to **Django**, create the tests allowing to check that the site behaves as follows:

- **favourites** views, **publications** and **publish** as well as their templates are only accessible by registered users.
- A registered user cannot access the new user creation form.
- A user cannot add the same article twice in their favorite list.

Your tests must be explicitly named, clearly specifying what they're testing.

Fix all the errors your tests might have ran into.