

Setting Up a Flask Application in PyCharm

(/post/setting-up-a-flask-application-in-pycharm)

July 23 2018

Posted by [Miguel Grinberg \(/author/Miguel Grinberg\)](/author/Miguel Grinberg) under [Flask \(/category/Flask\)](/category/Flask), [Python \(/category/Python\)](/category/Python), [Programming \(/category/Programming\)](/category/Programming).

Tweet

Like

G+

Share

In this short article and video I want to give you a few tips on setting up a PyCharm project for your Flask application. The idea is to set up a Flask application so that it can be executed, debugged, and tested from inside PyCharm Community Edition (<https://www.jetbrains.com/pycharm/download>), which is fantastic IDE for Python that is completely free to download and use. If you want to see me go through the exercise, watch the video below. Then you can come to the article if you want a quick reference and summary of the steps required.

Creating a Flask Project in PyCharm

This part is very simple. All you need to do to create your Flask project is to start PyCharm, select **Open**, and then choose the top-level directory of your application. PyCharm will create a new project and open it. The left sidebar will now show you a tree structure with all the files in your project.

Adding a Run and Debug Configuration

One of the first things you'll probably want to do is run your project. PyCharm works with the concept of "configurations", which are sets of parameters that define a run or debug task. To create or edit configurations you have to select **Run|Edit Configurations...** from the menu. If this option appears

disabled, you just need to wait a little bit and let PyCharm complete its initial background parsing of the project.

To create a new configuration, click the **+** sign, and then select **Python**. For the **Name** field, enter a description of this configuration, such as "webapp". I like to check the **Single Instance Only** option, because for a web application it isn't very useful to run multiple instances.

Under **Script path** you need to select the **flask** tool, which is located in the **bin** directory of your virtual environment. Here you can use the little **...** button to the right to browse and select this script. If you store your virtual environments outside of your project, or if you use a virtual environment wrapper such as pipenv which has its own location for virtualenvs, you will need to find out where the virtualenv is located and browse to that directory and then down into bin to find the flask command. If your virtualenv is in the same directory as your project this is much easier to do. I should note that if you are doing this on Windows, your virtualenv is not going to have a "bin" subdirectory. On Windows look for a **Scripts** subdirectory, and inside it look for a **flask.exe** file to select.

In the **Parameters** field you are going to enter **run**. This combined with the previous option configure PyCharm to run the familiar **flask run** command.

As I'm sure you also remember, the flask command requires a `FLASK_APP` environment variable to point to your Flask application. This variable needs to be defined in the **Environment variables** section. The value of the variable will be the same value you use when you run your application from the command line. If you are using a **.flaskenv** file in your project, then you are all set, the environment variable will be imported from there.

The final change you need to make is to set the **Working directory** to point to the top-level directory of your project instead of to the location of the flask command.

You can now close the configuration window. The dropdown on the top-right of your main PyCharm window should now be set to the configuration you just added. If you now click the green "play" button your project should start, and if instead you click the green "bug" button the project will start under the debugger. Running with the debugger is very useful, because you can set breakpoints to stop the program execution and inspect variables or run a part of your application step by step.

Running Unit Tests

In addition to being able to run and debug your application, it is useful to have PyCharm run your unit tests. This requires a second configuration to be added to the project. So go back to the **Run|Edit Configurations...** menu option, and then click the **+** button once again, to create a new

configuration. This time select the **Python tests** configuration type, and then pick the test runner that you would like to use. In the video demonstration above I picked **Unittests**, which is the runner based on the Python's unittest package.

Set the name of the configuration to **tests** or something similar. In the **Target** field make sure **Script path** is selected, and then choose the directory where your tests are located by clicking on the ... button to the right. In the **Pattern** field, enter a file pattern that applies to all the modules that contain tests. A few common patterns are **test_*.py**, ***_test.py** or ***test*.py**. To complete the test configuration, set the **Working directory** field to the top-level directory of your project.

After you close the configurations window, the dropdown in the top right of the PyCharm window will have **tests** selected. You can use this dropdown to go back to the **webapp** configuration when you need to. With **tests** selected, you can click the green run button to run your unit tests. PyCharm detects you are running tests, so it uses a dedicated panel in the bottom portion of the window to show you how the tests are doing. Any tests that fail will be displayed, and if you click on each one you can see the output it produced. You can also opt to run the tests under the debugger by clicking the green bug button, and that gives you the power to set breakpoints and run a specific part of a test step by step.

PyCharm also adds a little run button on the sidebar, next to each unit test function or method, and this is extremely convenient, as it allows you to run or debug a single test just by clicking its button.

Improved Debug Configuration

If you've got to this point, your project is nicely configured to run under PyCharm, but there is one detail that is missing. When your Flask application crashes, you know that there are two possible outcomes. If debug mode is not enabled, the client receives a 500 status code, and if debug mode is enabled, you are thrown into the browser-based debugger. When you run your application under the PyCharm debugger, it would be nice if crashes would stop the application at the crash point so that you can inspect variables and figure out what went wrong. Unfortunately due to a bug that has existed in Flask for a long time, this does not work if you configure your PyCharm project as I shown you above. More specifically, this is a bug that affects Flask applications that are started with the `flask run` command, but works well with the old `app.run()` method of running the application.

If you want to take advantage of the PyCharm debugger when your application crashes, you have to switch to `app.run()`. First, make sure your application's main script (which is usually the script that you set in the `FLASK_APP` environment variable) has this at the bottom:

```
if __name__ == '__main__':  
    app.run(debug=True, use_debugger=False, use_reloader=False, passthrough_errors=True)
```

This creates an alternative way to start the Flask application, without affecting the `flask run` command, which you can continue to use when you need to. The `app.run()` call that I added enables debug mode, but turns off the debugger and the reloader, so that they don't interfere with PyCharm. The `passthrough_errors` option is the key to make crashes reach the PyCharm debugger. Unfortunately this option cannot be set when you start your application via `flask run`.

The next change is to modify the run configuration in PyCharm to start the application through the `app.run()` call. Select the **webapp** configuration in the top-right corner of the PyCharm window, and then go back to **Run|Edit Configurations....** Change the **Script path** field to point to your main script, the one that now has the `app.run()` call. The **Parameters** field must be empty, so clear the "run" that was there from before. The `FLASK_APP` environment variable does not need to be set when using `app.run()`, so you can remove it from the configuration, although it doesn't hurt anything to leave it there.

If you now start the application, any crashes triggered by exceptions will reach PyCharm, which will stop the application and show you the location of the crash, an interactive stack trace, and the ability to check variable values.

Conclusion

I hope this was a useful guide to get started with PyCharm. In a future post I will go through the same exercise with Visual Studio Code, which is another pretty good IDE for Python. Do you have any other useful set up steps when you work with PyCharm? Let me know below in the comments!

Tweet

Like

G+

Share

12 comments



#1 Mycelias said a month ago

Michael, I'm using Windows 10 and cannot find the `flask.exe` to select when creating the Run/Debug configuration. I see it in my project directory, but cannot select in the Script Path field under the Configuration tab. What is the issue?



#2 Miguel Grinberg said a month ago

@Mycelias: I don't have a Windows machine at hand to test this. Can you at least write the path, instead of selecting it?



#3 WhiskeyDance said 18 days ago

@Mycelias: you can use "Module name" option instead of "Script path", just write "flask" in the field.



#4 WhiskeyDance said 18 days ago

Miguel, I have some issues configuring my PyCharm to catch errors: I configured it to run the main script itself instead of using "flask run" command, but when the app crashes I'm getting only logs in PyCharm debug console, it doesn't stop at the crash point. Strange thing is that instead of one traceback in the console I get multiple tracebacks about the same error - one traceback for each thread. Why do I have multiple threads? I configured Run config just like you did (and checked "Single instance only" as well). My PyCharm version is 2018.2.1 CE. Can you please check logs in your debug console when your app crashes (with the config from this article) and help me with that problem? Thank you in advance.



#5 Miguel Grinberg said 17 days ago

@WhiskeyDance: you should have a "Python Exception Breakpoint" set to "Any Exception" in your project for PyCharm to break into the debugger. This is added by PyCharm by default, at least in my own projects it always gets added. Maybe it is missing for you?



#6 WhiskeyDance said 17 days ago

Miguel, it's already set to "Any Exception", all other settings of PyCharm are set by default (except project related stuff like an interpreter path). If I set a breakpoint manually in PyCharm then app stops when it get to the breakpoint and the IDE works as intended. But exception breakpoints do not work for some reason, PyCharm debugger doesn't catch them. Maybe it's a problem with 2018.2.1 CE. I'm still interested what you get in your console (how many threads) when your app crashes.

When I start my debugger PyCharm executes the following command (I see it in debugger console):

```
path\to\project\venv\Scripts\python.exe "C:\Program Files\JetBrains\PyCharm Community Edition 2018.2\helpers\pydev\pydevd.py" --multiproc --qt-support=auto --client 127.0.0.1 --port 54527 --file path\to\project\run.py
```

I'm using Win10 if that matters.



#7 Miguel Grinberg said 17 days ago

@WhiskeyDance: Not sure what you are asking with regards to threads. The number of threads you have will depend on how many requests are being handled, or else these must be threads that your application starts that are not associated with a request. Are you running in debug mode, as I indicated above? If you were in debug mode and PyCharm isn't stopping, then you should be getting the interactive debugger in the browser. Sounds like you are just getting a stack trace in the console, which means you are probably not running in debug mode. Try enabling debug mode as I did above.



#8 WhiskeyDance said 17 days ago

Miguel, I tried to do this with your flasky project and everything worked! However after I upgraded flask to the latest version I get the same problems with debugger, so that's the case.

#9 Miguel Grinberg said 17 days ago



@WhiskeyDance: Yeah, looks like you are correct, Flask 1.0 does not handle this the right way. I'll investigate.



#10 WhiskeyDance said 17 days ago

Miguel, thank you. Also, awesome blog and mega-tutorial :)



#11 Miguel Grinberg said 17 days ago

@WhiskeyDance: okay, this is actually not an issue. Flask 1.x runs in multithreaded mode by default, so any exception raised in a request handler will exist in the context of a thread. In previous Flask releases the server was single-threaded, so exceptions were raised in the main thread, causing the application to stop. The PyCharm debugger is configured to stop on termination by default, not when a thread raises an exception. I was able to get the debugger to catch the exception by setting the "On Raise" and "Ignore library files" options in the exception breakpoint configuration. Give that a try.



#12 WhiskeyDance said 17 days ago

Miguel, I tried it, it works, thanks. But when you press F9 (resume program) the debugger will jump to the same exception breakpoint from another thread. To solve this I added threaded=False parameter to app.run(), and everything works as intended now. Thank you.

««

«

»

»»

Leave a Comment

Name

URL

Email

Comment

Captcha

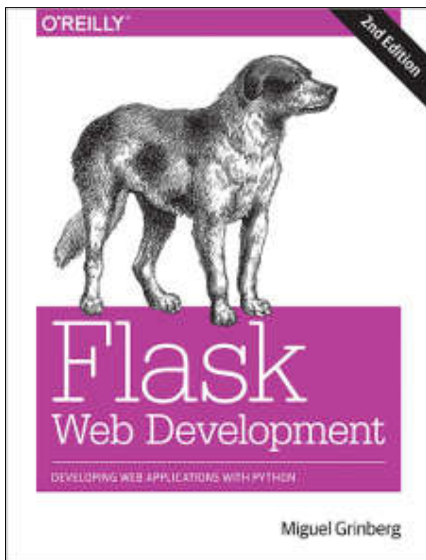
I'm not a robot

reCAPTCHA
Privacy - Terms

Submit

Flask Web Development, 2nd Edition

If you want to learn modern web development techniques with Python and Flask, you may find the second edition of my O'Reilly book (https://www.amazon.com/gp/product/1491991739/ref=as_li_tl?ie=UTF8&tag=thirdwish-20&camp=1789&creative=9325&linkCode=as2&creativeASIN=1491991739&linkId=10f6689564c9c4cda68802397c58e906) useful:



([https://www.amazon.com/gp/product/1491991739/ref=as_li_tl?ie=UTF8&tag=thirdwish-](https://www.amazon.com/gp/product/1491991739/ref=as_li_tl?ie=UTF8&tag=thirdwish-20&camp=1789&creative=9325&linkCode=as2&creativeASIN=1491991739&linkId=10f6689564c9c4cda68802397c58e906)

[20&camp=1789&creative=9325&linkCode=as2&creativeASIN=1491991739&linkId=10f6689564c9c4cda68802397c58e906](https://www.amazon.com/gp/product/1491991739/ref=as_li_tl?ie=UTF8&tag=thirdwish-20&camp=1789&creative=9325&linkCode=as2&creativeASIN=1491991739&linkId=10f6689564c9c4cda68802397c58e906))

About Miguel

Welcome to my blog!







I'm a software engineer, photographer and filmmaker in Portland, Oregon, USA.




























You can also find me on Facebook (<https://www.facebook.com/miguelgrinbergblog>), Google+ (<https://plus.google.com/u/0/117786742456929977820>), LinkedIn (<http://www.linkedin.com/in/miguelgrinberg>), Github (<https://github.com/miguelgrinberg>) and Twitter (<https://twitter.com/#!/miguelgrinberg>).



Thank you for visiting!

Categories

-  (/category/AWS/feed) **AWS (/category/AWS)** (1)
-  (/category/Arduino/feed) **Arduino (/category/Arduino)** (7)
-  (/category/Authentication/feed) **Authentication (/category/Authentication)** (6)
-  (/category/Blog/feed) **Blog (/category/Blog)** (1)
-  (/category/C++/feed) **C++ (/category/C++)** (5)
-  (/category/Cloud/feed) **Cloud (/category/Cloud)** (6)

	(/category/Database/feed)	Database (/category/Database)	(11)
	(/category/Docker/feed)	Docker (/category/Docker)	(1)
	(/category/Filmmaking/feed)	Film making (/category/Film making)	(6)
	(/category/Flask/feed)	Flask (/category/Flask)	(78)
	(/category/Games/feed)	Games (/category/Games)	(1)
	(/category/HTML5/feed)	HTML5 (/category/HTML5)	(1)
	(/category/Heroku/feed)	Heroku (/category/Heroku)	(1)
	(/category/JavaScript/feed)	JavaScript (/category/JavaScript)	(8)
	(/category/Microservices/feed)	Microservices (/category/Microservices)	(2)
	(/category/Movie Reviews/feed)	Movie Reviews (/category/Movie Reviews)	(5)
	(/category/Netflix/feed)	Netflix (/category/Netflix)	(5)
	(/category/Node.js/feed)	Node.js (/category/Node.js)	(1)
	(/category/OpenStack/feed)	OpenStack (/category/OpenStack)	(1)
	(/category/Personal/feed)	Personal (/category/Personal)	(2)
	(/category/Photography/feed)	Photography (/category/Photography)	(7)
	(/category/Product Reviews/feed)	Product Reviews (/category/Product Reviews)	(2)
	(/category/Programming/feed)	Programming (/category/Programming)	(95)
	(/category/Project Management/feed)	Project Management (/category/Project Management)	(1)
	(/category/Python/feed)	Python (/category/Python)	(90)
	(/category/REST/feed)	REST (/category/REST)	(6)
	(/category/Rackspace/feed)	Rackspace (/category/Rackspace)	(1)
	(/category/Raspberry Pi/feed)	Raspberry Pi (/category/Raspberry Pi)	(7)
	(/category/Robotics/feed)	Robotics (/category/Robotics)	(6)
	(/category/Security/feed)	Security (/category/Security)	(10)
	(/category/Video/feed)	Video (/category/Video)	(5)
	(/category/Webcast/feed)	Webcast (/category/Webcast)	(3)
	(/category/Windows/feed)	Windows (/category/Windows)	(1)

© 2012-2018 by Miguel Grinberg. All rights reserved. Questions? (mailto:webmaster_at_miguelgrinberg_dot_com)