

LIMITED
TIME **SALE**

Offer ends in:

2d 09h 08m 33s

EN

TUTORIAL ▾

Category ▾

[Home](#) > [About Python](#) > [Learn Python](#)

FastAPI Tutorial: An Introduction to Using FastAPI

Explore the FastAPI framework and discover how you can use it to create APIs in Python

Contents

Sep 2022 · 13 min read



Moez Ali

Data Scientist, Founder & Creator of PyCaret

TOPICS

Python

API (Application Programming Interface) are the backbone of modern architecture because they allow for applications to be modular and decoupled. This means that you can build applications built quickly and easily, allowing you to easily maintain and update them.

APIs are also very important in machine learning because they allow different applications to share data and work together, saving time and effort. There are many different frameworks for building APIs in Python. Some of the most popular frameworks for creating APIs in Python are Django, Flask, and FastAPI. This tutorial is a deep dive into one of the frameworks called FastAPI.

What is an API?

API stands for Application Programming Interface. An API is a software intermediary that allows two applications to talk to each other. When you use an application on your phone, the application connects to the Internet and sends data to a server. The server then processes the data and sends it back to your phone. The application on your phone then interprets the data and presents it to you in a readable way.

An API is like a waiter in a restaurant. The waiter takes your order and sends it to the kitchen. The kitchen then prepares the food and sends it back to the waiter. The waiter then brings the food to you.

In the same way, an API takes a request from an application and sends it to a server. The server then processes the request and sends the data back to the application. The application then interprets the data and presents it to the user.

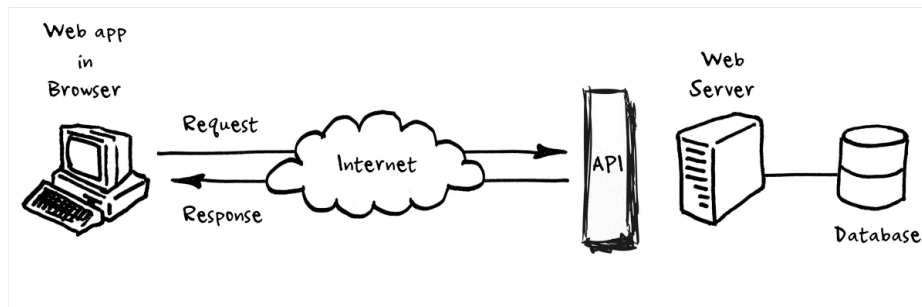


Image Source: <https://www.techfunnel.com/wp-content/uploads/2021/07/api.png>

If you want to learn more about machine learning pipelines, APIs, and MLOps, you can check out our [Machine Learning, Pipelines, Deployment and MLOps Tutorial](#).

What is FastAPI

FastAPI is a high-performing web framework for building APIs with Python 3.7+ based on standard Python type hints. It helps developers build applications quickly and efficiently. FastAPI is built on top of the Starlette web server and includes features that make building web applications easier, such as automatic data validation, error handling, and interactive API docs.

We will look at all these features individually in this section. First, let's look at key features as pointed out in the original documentation of FastAPI.

- **Performance:** On par with NodeJS and the Go language.
- **Speed:** Increase the development speed 2-3X.
- **Easy:** Great editor support. Completion everywhere. Easy to learn and use.
- **Robust:** Production-ready code with automatic interactive documentation.
- **OpenAPI based:** Fully compatible with OpenAPI and JSON Schema.

Installing FastAPI

FastAPI requires Python 3.7+. It can be installed using pip. You will need to install FastAPI and the ASGI server 'uvicorn'.

```
..  
  
# install fastapi  
pip install fastapi  
  
# install uvicorn  
pip install uvicorn  
..
```

 Explain code

 OpenAI

Create a simple API

Let's directly get into creating a very simple toy API. I am using VS Code to implement this, but you can use any editor you like.

```
...  
  
from typing import Union
```

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"Hello": "World"}

@app.get("/items/{item_id}")
def read_item(item_id: int, q: Union[str, None] = None):
    return {"item_id": item_id, "q": q}
...

```

[Explain code](#) OpenAI

(Example reproduced from the [original documentation](#)). Thanks to @tiangolo.

Now using a command line terminal, run this API with the following command:

```
...

uvicorn main:app --reload

...

```



POWERED BY DATACAMP WORKSPACE

``main`` is the name of the Python file, and ``app`` is the variable that stores the FastAPI class. You can name them whatever you like. Once you run the above command, you will see something like this on your terminal:

```
(base) C:\Users\owner\fastapi>uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['C:\\Users\\owner\\fastapi']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [19124] using statreload
INFO:     Started server process [32540]
INFO:     Waiting for application startup.
INFO:     Application startup complete.

```

Head over to the link in your browser, and if you see a page that shows ``Hello World``, the API is up and running.

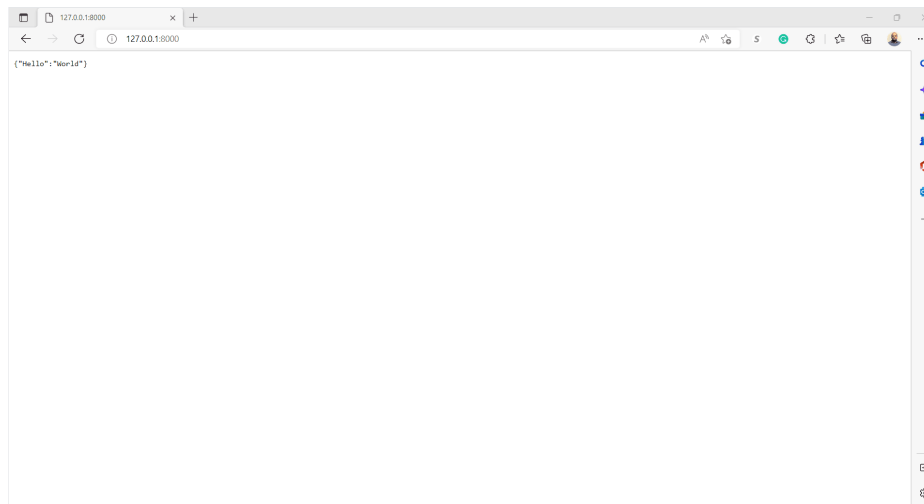


Image by Author

Congratulations on building your first API.

Interactive API Docs

FastAPI generates a "schema" with all your APIs using the [OpenAPI](#) standard for defining APIs. A "schema" is a definition or description of something. Not the code that implements it, but just an abstract description. The OpenAPI schema is what powers the two interactive documentation systems included in FastAPI.

To see the documentation, just add `/docs` to the url (`http://127.0.0.1:8000/docs`). This link will show automatic interactive API documentation.

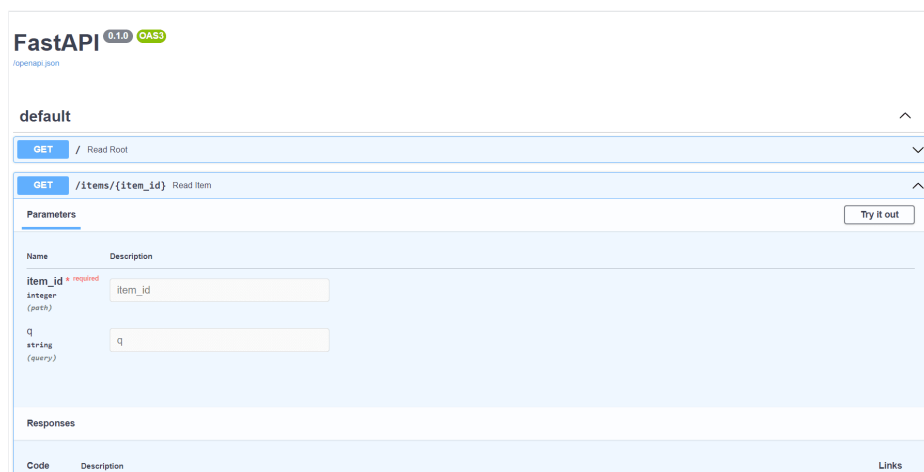


Image by Author

Click on 'Try it out' button in the top right corner to test the API.

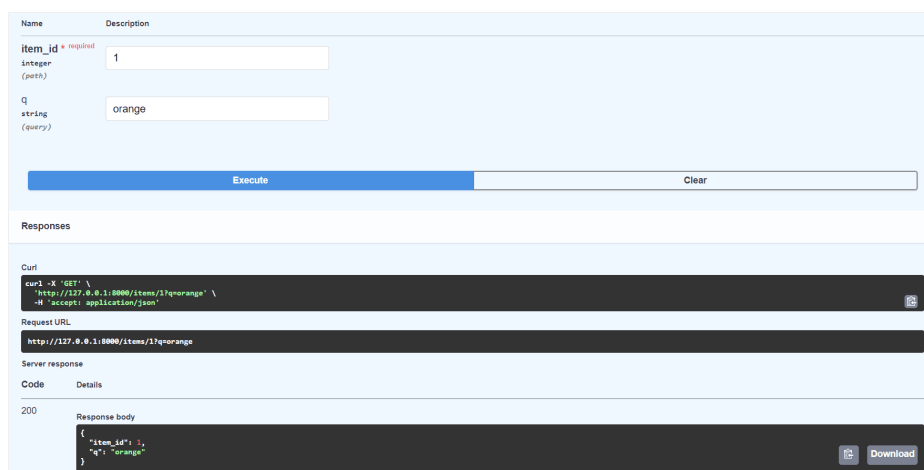


Image by Author

You can see the response body is a dictionary. This is the return call of the `read_item` function defined in the `main.py`. You can also see the request url `http://127.0.0.1:8000/items/1?q=orange`. `1` and `orange` were our inputs to the API.

There is an alternative interactive automatic documentation available in FastAPI. To check that out, go to `http://127.0.0.1:8000/redoc`. This is what it looks like:

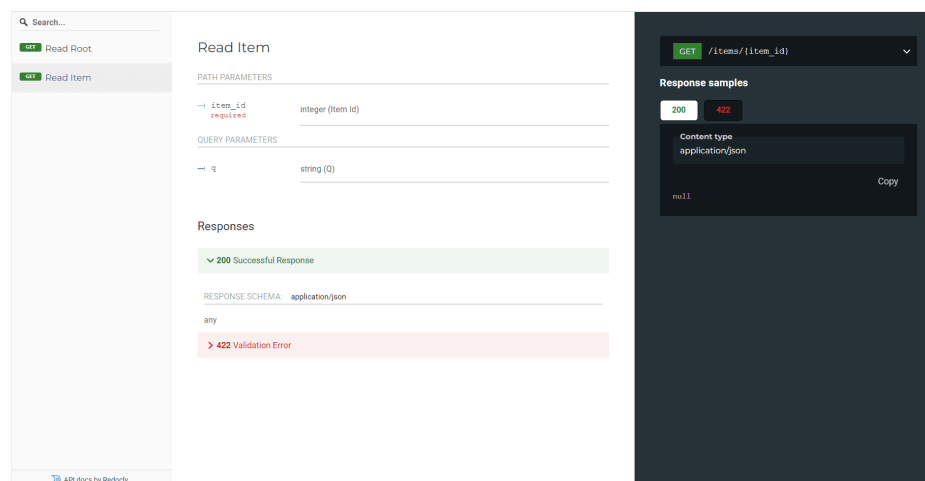


Image by Author

More advanced examples

While building an API, the "path" defines the route or endpoint of the request. However, there is one more choice we have to make here, i.e., "Operation." The word `operation` here refers to one of the HTTP "methods." By using one (or more) of these so-called "methods," you can communicate with each of the several paths supported by the HTTP protocol. Typically, you would use:

- **POST:** to create data.
- **GET:** to read data.
- **PUT:** to update data.
- **DELETE:** to delete data.
- And a few other advanced ones

FastAPI supports all of the http methods.

The fact that FastAPI is built on Python type hints is yet another key aspect of this framework. Type hints are supported in versions of Python 3.6 and later. The type hints are a specialized kind of syntax that makes it possible to declare the type of a variable.

Declaring types for your variables enables editors and other tools to provide you with improved assistance. Let's see an advanced example.

We will modify our `main.py` file to include a new `PUT` request which will take multiple inputs of different data types.

```
...

from typing import Union

from fastapi import FastAPI

from pydantic import BaseModel
```

```
app = FastAPI()

class Item(BaseModel):
    name: str

    price: float

    is_offer: Union[bool, None] = None

@app.get("/")
def read_root():
    return {"Hello": "World"}

@app.get("/items/{item_id}")
def read_item(item_id: int, q: Union[str, None] = None):
    return {"item_id": item_id, "q": q}

@app.put("/items/{item_id}")
def update_item(item_id: int, item: Item):
    return {"item_name": item.name, "item_id": item_id}

...
```

[Hide code explanation](#)

- This code is written in Python and uses the FastAPI and Pydantic libraries.
- First, the necessary libraries are imported using the `import` statement.
- The `typing` library is used to define the type of variables, and the `Union` class is used to specify that a variable can have multiple types.
- The `FastAPI` class is used to create a web application, and the `BaseModel` class from Pydantic is used to define the structure of the data that will be sent and received by the application.
- Next, an instance of the `FastAPI` class is created using the `app = FastAPI()` statement.
- Then, a new class called `Item` is defined, which inherits from the `BaseModel` class.
- This class has three attributes: `name`, `price`, and `is_offer`.
- The `is_offer` attribute is optional and can be either a boolean or `None`.
- After that, three endpoints are defined using the `app.get()` and `app.put()` decorators.
- The `@app.get("/")` decorator defines the root endpoint, which returns a simple JSON object with the message "Hello World".
- The `@app.get("/items/{item_id}")` decorator defines an endpoint that takes an `item_id` parameter and an optional `q` parameter, and returns a JSON object with the `item_id` and `q` values.
- The `@app.put("/items/{item_id}")` decorator defines an endpoint that takes an `item_id` parameter and an `item` parameter of type `Item`, and returns a JSON object with the `item_name` and `item_id` values.
- Overall, this code sets up a basic web application with three endpoints that can receive and return JSON data.

Was this helpful? ☒ Yes ☐ No

(Example reproduced from the [original documentation](#)). Thanks to @tiangolo.

The change:

A `put` request is added that takes two inputs. `item_id` is an integer and `item` type is pointing to custom class `Item` created and inheriting the `BaseModel` from `pydantic`. The class `Item` contains three attributes: `name`, `price`, `is_offer`, and all of them have different data types.

FastAPI will check:

- `name` is a `str`.
- `price` is a `float`.
- `is_offer` is a bool, if present.

The benefit of using type hints is that you declare once the types of parameters, body, etc. as function parameters with standard Python (3.6+). You will get:

- Editor support, including auto-completion, type checks
- Data Validation
- Conversion of input data
- Conversion of output data
- Errors that are easy to understand.

Comparison of FastAPI with Django and Flask

All three of these frameworks are Python web frameworks that you can use to develop web applications. They each have their own strengths and weaknesses.

Django is a full-featured framework that includes everything you need to get started, including a built-in ORM and an admin panel. It can be a bit overwhelming for beginners, but its comprehensive documentation makes it easy to learn.

Flask is a microframework that is lightweight and easy to get started with. It doesn't include as many features as Django, but it's perfect for simple projects.

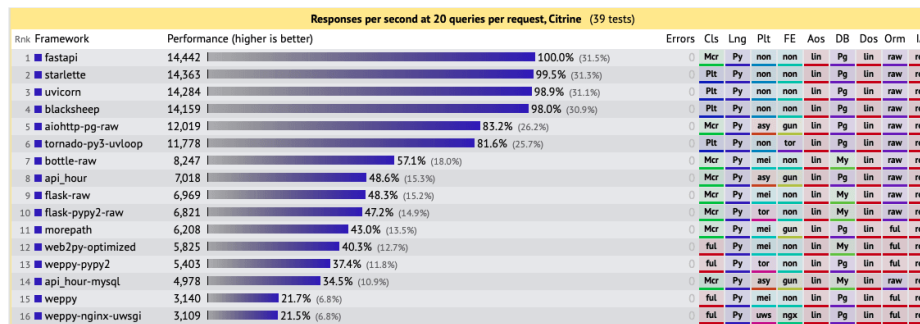
FastAPI is a new framework that is designed to be fast and easy to use. It includes features like automatic data validation and documentation.

	Django	Flask	FastAPI
Community	Big. 66K GitHub stars	Big. 61K GitHub stars	Big. 50K GitHub stars
Performance	Django is massive. It isn't the best in terms of performance.	Flask is a micro web framework. It performs better than Django.	FastAPI is one of the fastest web frameworks with native async support that adds to the efficiency of the framework.
Async support	Yes, with limited latency.	No. Needs Asyncio	FastAPI provides native async support.
Ease of use	Django is massive and hence a bit	Flask is easy to learn and pretty	FastAPI is the simplest of all three.

	complicated to learn.	straightforward in use.	
Interactive Documentation	Not interactive	No	Yes (OpenAI, Redoc)
Data Verification	No	No	Yes

FastAPI Performance Benchmarks

According to the results of tests run by [techempower](#), FastAPI is superior to all other frameworks in terms of its overall performance.



Source: <https://www.techempower.com/benchmarks/>

Example: Building an end-to-end machine learning Pipeline with PyCaret and deploying with FastAPI

In this section, we will quickly build a machine learning pipeline and then create an API to serve the model. We will use a low-code Python library [PyCaret](#) to build a pipeline and create an API. PyCaret has integration with FastAPI, which makes it extremely easy to create and serve machine learning models as an API.

PyCaret

PyCaret is an open-source, low-code machine learning library in Python that automates machine learning workflows. It is an end-to-end machine learning and model management tool that speeds up the experiment cycle exponentially and makes you more productive.

```

...

pip install pycaret

...

...

import pycaret

pycaret.__version__

>>> 2.3.10

...

```




We will use the `insurance` dataset in this example. It is a regression use-case for predicting medical charges based on age, sex, BMI, and region.

```
...  
  
from pycaret.datasets import get_data  
  
data = get_data('insurance')  
  
...
```

POWERED BY DATACAMP WORKSPACE

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

Image by Author

Next we will initialize the `setup` function from pycaret. This function initializes the experiment in PyCaret and creates the transformation pipeline based on all the parameters passed in the function.

The `setup` must first executed before any other function. It requires two parameters to work `data` and `target`. All the remaining arguments in `setup` are optional. The `setup` function is responsible for controlling the entirety of the data preprocessing procedures. Check out [the PyCaret documentation](#) for a comprehensive rundown of all of the preprocessing procedures that are supported in PyCaret.

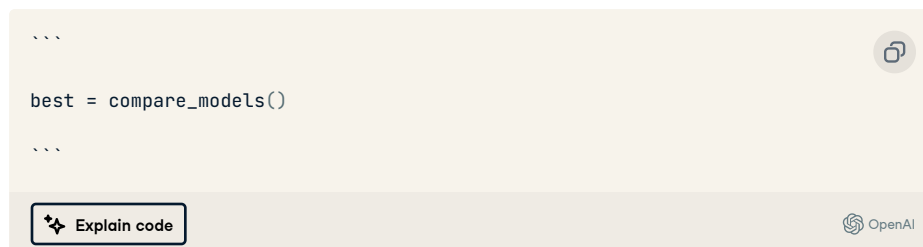
```
...  
  
from pycaret.regression import *  
  
s = setup(data, target = 'charges')  
  
...
```



	Description	Value
0	session_id	1350
1	Target	charges
2	Original Data	(1338, 7)
3	Missing Values	False
4	Numeric Features	2
5	Categorical Features	4
6	Ordinal Features	False
7	High Cardinality Features	False
8	High Cardinality Method	None
9	Transformed Train Set	(936, 14)
10	Transformed Test Set	(402, 14)
11	Shuffle Train-Test	True
12	Stratify Train-Test	False
13	Fold Generator	KFold
14	Fold Number	10
15	CPU Jobs	-1
16	Use GPU	False
17	Log Experiment	False

Image by Author Output truncated.

Once the setup is finished, we can start model training and selection with just one line of code: `compare_models`. Through the use of cross-validation, this function trains and evaluates the model performance of all estimators within the model library. The result of this function is a scoring grid that contains the average scores obtained from cross-validation.




	Model	MAE	MSE	RMSE	R2	RMSLE	MAPE	TT (Sec)
gbr	Gradient Boosting Regressor	2520.6229	20769829.9785	4525.8866	0.8568	0.3947	0.2709	0.0230
catboost	CatBoost Regressor	2711.2094	22678972.2643	4730.0674	0.8437	0.4191	0.2886	0.4740
lightgbm	Light Gradient Boosting Machine	2855.9507	23314394.3696	4791.7314	0.8394	0.4880	0.3225	0.2960
rf	Random Forest Regressor	2689.3003	23521544.1910	4826.4161	0.8380	0.4269	0.2863	0.0870
ada	AdaBoost Regressor	3833.7662	25143449.8335	4987.6102	0.8265	0.5675	0.6121	0.0080
et	Extra Trees Regressor	2690.2509	26168744.7162	5092.3057	0.8195	0.4448	0.2821	0.0790
xgboost	Extreme Gradient Boosting	3108.6635	28860047.6000	5336.6700	0.7998	0.4893	0.3525	0.2740
llar	Lasso Least Angle Regression	4213.1579	35666189.6567	5947.2977	0.7551	0.5901	0.4213	0.0060
ridge	Ridge Regression	4231.2149	35681675.4000	5949.1723	0.7550	0.5831	0.4252	0.0050
lasso	Lasso Regression	4218.5314	35675973.2000	5948.1665	0.7550	0.5968	0.4235	0.0060
br	Bayesian Ridge	4228.1661	35684770.8810	5949.2847	0.7550	0.5846	0.4248	0.0050
lr	Linear Regression	4219.3180	35688562.0000	5949.2035	0.7549	0.6007	0.4236	0.6830
lar	Least Angle Regression	4276.7246	36316676.7397	5995.9336	0.7502	0.6042	0.4403	0.0050
dt	Decision Tree Regressor	2987.3224	39904963.5364	6267.2071	0.7280	0.4740	0.2795	0.0110
huber	Huber Regressor	3483.0094	46557209.1038	6773.2554	0.6803	0.4893	0.2349	0.2230
omp	Orthogonal Matching Pursuit	5756.4046	57504547.0291	7551.4122	0.6063	0.7200	0.8625	0.0060
par	Passive Aggressive Regressor	4927.7735	64161499.7943	7976.0997	0.5625	0.6802	0.4169	0.0070
en	Elastic Net	7229.1871	87894292.8000	9364.7893	0.4006	0.7166	0.8866	0.0070
knn	K Neighbors Regressor	7753.2965	124523234.4000	11119.9351	0.1540	0.8098	0.8460	0.0150
dummy	Dummy Regressor	9117.1966	148041516.8000	12143.8482	-0.0043	0.9955	1.5027	0.0070

Image by Author

Based on this the best performing model is `Gradient Boosting Regressor`. If we want, we can analyze the model through visualization and further try to improve performance through hyperparameter tuning or model ensembling, but we will not do that in this tutorial.

We will get straight into building the API of the best model to serve predictions in production using FastAPI. Remember, **PyCaret has integration with FastAPI, so it can automatically create a REST API from the model using the `create_api` function.**

```
...
create_api (best, 'insurance_prediction_model')
...
```

 Hide code explanation

 OpenAI

- This code snippet is incomplete and does not provide enough information to explain how it works.
- It appears to be calling a function named `create_api` with two arguments: `best` and `'insurance_prediction_model'`.
- However, without the context of the rest of the code and the definition of the `create_api` function, it is impossible to provide a complete explanation.

Was this helpful? ☒ Yes ☐ No

API successfully created. This function only creates a POST API, it doesn't run it automatically.
To run your API, please run this command --> `!python insurance_prediction_model.py`

Now to run this API, open the command prompt terminal and navigate to the folder where your Notebook is and run the following command `python insurance_prediction_model.py`.

```
Transformation Pipeline and Model Successfully Loaded
[12mInfo[0m: Started server process [12mInfo[0m]
[12mInfo[0m: Waiting for application startup.
[12mInfo[0m: Application startup complete.
[12mInfo[0m: [Error 1000] error while attempting to bind on address ('127.0.0.1', 8000): only one usage of each socket address (protocol/network address/port) is normally permitted
[12mInfo[0m: Waiting for application shutdown.
[12mInfo[0m: Application shutdown complete.
```

Head over to `http://127.0.0.1:8000/docs` and you will see the same UI as you have seen earlier in this tutorial

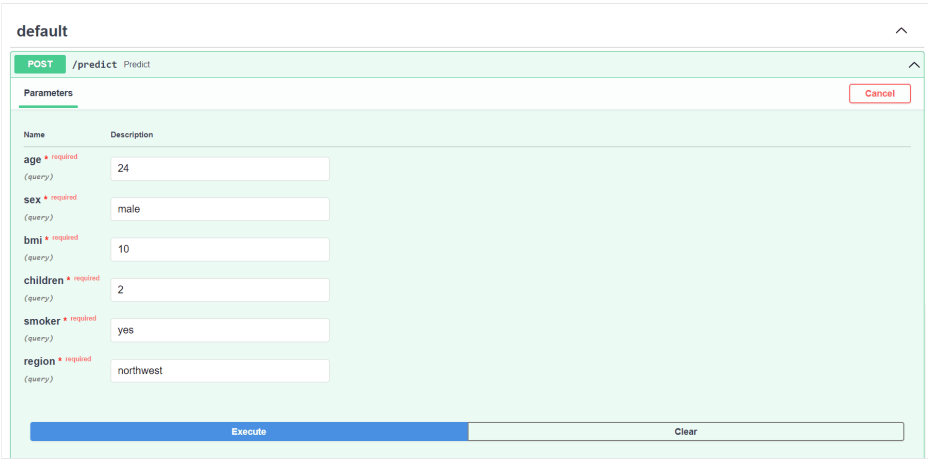


Image by Author

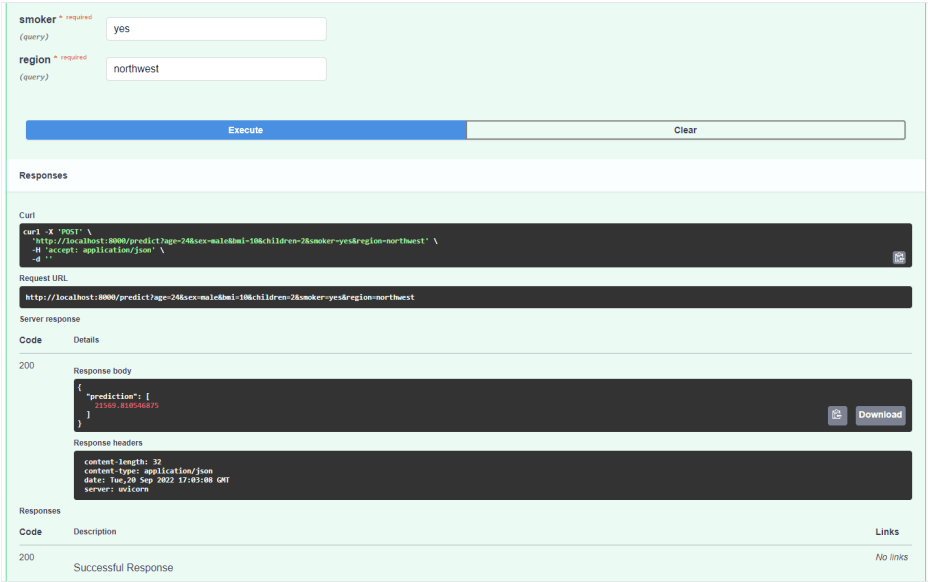


Image by Author

You can also see alternative interactive documentation by going to `http://127.0.0.1:8000/redoc`

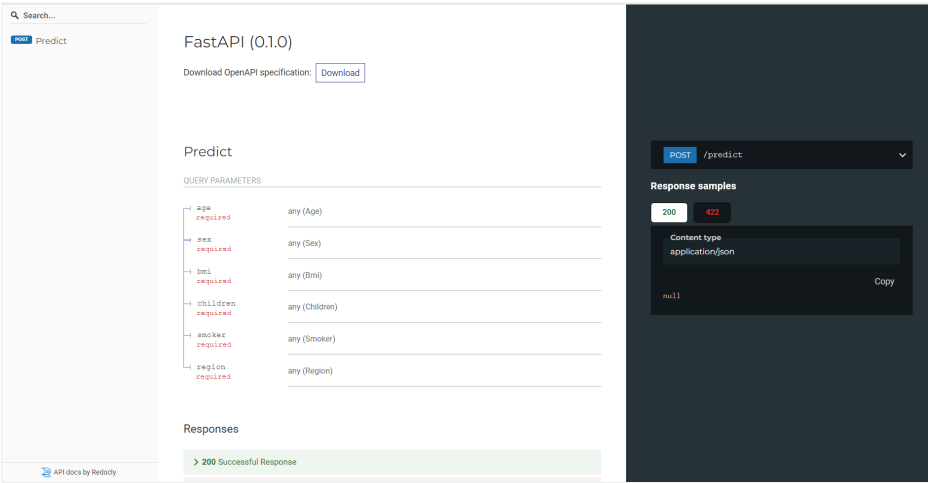


Image by Author

Remember, both the default and redoc interactive documentation are powered using OpenAPI standards.

If you want to see the file that PyCaret created when you used `'create_api'` function you can check out the file in the same folder as your Notebook. It looks like this:

```
...  
  
import pandas as pd  
  
from pycaret.regression import load_model, predict_model  
  
from fastapi import FastAPI  
  
import uvicorn  
  
  
# Create the app  
  
app = FastAPI()  
  
  
# Load trained Pipeline  
  
model = load_model('my_lr_api')  
  
  
# Define predict function  
  
@app.post('/predict')  
  
def predict(age, sex, bmi, children, smoker, region):  
  
    data = pd.DataFrame([[age, sex, bmi, children, smoker, region]])  
  
    data.columns = ['age', 'sex', 'bmi', 'children', 'smoker', 'region']  
  
    predictions = predict_model(model, data=data)  
  
    return {'prediction': list(predictions['Label'])}  
  
  
if __name__ == '__main__':  
  
    uvicorn.run(app, host='127.0.0.1', port=8000)  
  
...
```

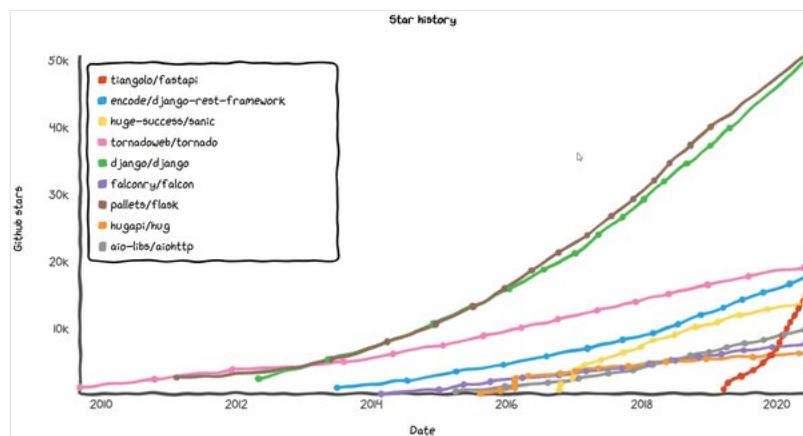
[🔗 Explain code](#) OpenAI

Conclusion

The article discussed the idea of API and why they are used. API is important in modern architecture because it allows different software programs to share data and functionality. This makes it possible to create complex systems that are more reliable and easier to maintain.

We then deep dive into a relatively new framework in Python called FastAPI. FastAPI is a newer API that is designed to be easy to use and efficient. It is a great option for developers

who want to create an API quickly and easily.



GitHub star history - created using star-history.com

If you also want to learn how to create a simple API from a machine learning model in Python using Flask, check out this easy-to-follow, [Turning Machine Learning Models into APIs in Python tutorial](#).

FastAPI FAQs

Is FastAPI asynchronous? ^

FastAPI supports asynchronous code out of the box using the `async/await` Python keywords.

Is FastAPI production ready? v

How is FastAPI different from Flask? v

Does FastAPI also have a built-in development server? v

Is FastAPI supported on Python 2? v

Is FastAPI available in Python only? v

Is FastAPI faster than Go programming language? v

Is FastAPI compatible with Pydantic? v

TOPICS

Python

Python Courses

Preprocessing for Machine Learning in Python

• Beginner ⌚ 4 hr 👤 38.2K

In this course you'll learn how to get your cleaned data ready for modeling.

See Details →

Start Course

See More →

Related



How to Learn Python From
Scratch in 2023: An Expert...

Matt Crabtree



10 Essential Python Skills All
Data Scientists Should Master

Thaylise Nakamoto



Distributed Processing using
Ray framework in Python

Moez Ali

See More →

Grow your data skills with DataCamp for Mobile

Make progress on the go with our mobile courses and daily 5-minute coding challenges.



LEARN

Learn Python

Learn R

Learn AI

Learn SQL

Learn Power BI

Learn Tableau

Learn Data Engineering

Assessments

[Career Tracks](#)[Skill Tracks](#)[Courses](#)[Data Science Roadmap](#)

DATA COURSES

[Upcoming Courses](#)[Python Courses](#)[R Courses](#)[SQL Courses](#)[Power BI Courses](#)[Tableau Courses](#)[Spreadsheets Courses](#)[Data Analysis Courses](#)[Data Visualization Courses](#)[Machine Learning Courses](#)[Data Engineering Courses](#)

WORKSPACE

[Get Started](#)[Templates](#)[Integrations](#)[Documentation](#)

CERTIFICATION

[Certifications](#)[Data Scientist](#)[Data Analyst](#)[Data Engineer](#)[Hire Data Professionals](#)

RESOURCES

[Resource Center](#)[Upcoming Events](#)[Blog](#)[Tutorials](#)[Open Source](#)[RDocumentation](#)[Course Editor](#)[Book a Demo with DataCamp for Business](#)

[Data Portfolio](#)[Portfolio Leaderboard](#)

PLANS

[Pricing](#)[For Business](#)[For Universities](#)[Discounts, Promos & Sales](#)[DataCamp Donates](#)

SUPPORT

[Help Center](#)[Become an Instructor](#)[Become an Affiliate](#)

ABOUT

[About Us](#)[Learner Stories](#)[Careers](#)[Press](#)[Leadership](#)[Contact Us](#)[Privacy Policy](#)[Cookie Notice](#)[Do Not Sell My Personal Information](#)[Accessibility](#)[Security](#)[Terms of Use](#)

© 2023 DataCamp, Inc. All Rights Reserved.