Create a REST API for an existing SQL Database with Python and Tornado in 10 Minutes.

Klaas (khz) · Follow

9 min read · Jan 9, 2020

The problem:

Sometimes just want to explore .

You don't really want to create a full blown application for that or reverse engineer the models or schemas but just want to connect, explore and expose as easy as possible. If this is just for reporting, you don't even have to bother with transactions or logic.

PythonOnWheels is ideal for this scenario. It is a Tornado based python 3 web framework with a focus on making your development workflow easy and of less boilerplate.

> *It is actually more a glue of some great existing python libraries than a full blown framework itself.*

We will take an (unknown) sample DB, connect to it. Reflect the Schema and expose an HTTP API so that we can retrieve data from the DB over HTTP / REST.

We will use curl and the insomnia REST client in the following examples. Curl is THE cli tool and insomnia is a really great HTTP GUI multiplattform REST client application.

Scenario:

- You already have a SQL (or NoSQL) DB and want to expose an HTTP API to make the data accessible.

Plan:

- Create a PythonOnWheels app
- Connect to the existing DB
- We use the chinook SQLite sample DB from sqlitetutorial.net
- Use reflection to auto-map the DB Tables to our models
- And finally expose a JSON REST API

Estimated time to build our base App:

- 10 Minutes (what ?? Read on .. it's probably even quicker ;)

To make it a little "harder" we assume that the existing DB is a SQL DB

The techni called reflection. We know already how to reflect an existing SQL DB from the documentation. We will use the same methodology but another sample DB taken from the SQlite tutorials.

This is the schema of the sample Database

The sample Database and the schema image is taken from www.sqlitetutorial.net.

Basics

- generate a new PythonOnWheels Application (I named it chinook for this example)
- Download the sample DB (250KB) and copy it to your Application directory

Configuration for the Database:

- Change the DB configuration in your *<appname>/config.py* database section.
- Change the "*dbname*" field to chinook.db

Initialize the SQL environment (once) and go:

We setup the SQLAlchemy / Alembinc environment in the backend with this command:

```
python init_sqldb_environment.py
```

> *This is a typical example of PythonOnWheels taking away the boilerplate but is based on the very strong foundation os sqlalchemy and alembic*

Reflect (auto-map) the employees Table

First we need to generate our model.

```
python generate_model.py -n employee -t sql
```

This generates a model class which we normally use to create a table and to insert data. In this case we want to make it exactly the other way round. We want to reflect an existing table. So we need to open the generated model and adapt 4 things:

- make the schema empty, *schema = {}*
- We wil

- set a custom table name: employees
- using the __tablename__ attribute.
- disable using the PoW schema enhancements (id, created_at, last_updated) using: _use_pow_schema_attrs = False
- These enhancements normally help us in the workflow but are not needed in this case
- set the autoload parameter to enable reflection in __table_args__ :
- add “autoload” : True

This is how the final model should look like:

```
@relation.setup_sql_schema()
class Employee(Base, metaclass=PowBaseMeta):

    #
    # cerberus style schema
    #
    schema = { }
    # if you want to define a custom tablename for this model:
    __tablename__ = "employees"
    # if you dont want to use the pow schema extension
    _use_pow_schema_attrs= False
    # define class attributes/variables here that should be included
in to_dict()
    # conversion and also handed to the encoders but that are NOT
part of the schema.
    include_attributes=[]
    # Add sqlalchemy table_args here. Add "autoload" : True for
database reflection
    __table_args__ = { "extend_existing": True, "autoload" : True  }

    #
    # init
    #
    def __init__(self, **kwargs):
        self.setup_instance_values()
        self.init_on_load(**kwargs)
    #
    # your model's methods down here
    #
```

Believe it or not, this is it for the DB part

We are nov To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy, including cookie policy. Query the Db, to upd

```
>>> from chinook.models.sql.employee import Employee
cls: <class 'chinook.models.sql.employee.Employee'>, name: Employee
dict_keys(['__module__', 'schema', '__tablename__',
'_use_pow_schema_attrs', 'include_attributes', '__table_args__',
'__init__', '__doc__'])
not adding pow schema attrs
setup_schema:employee
```

Create an instance and show the schema

```
>>> e=Employee()
trying to find possible observer in
chinook.models.sql.employee_observer.EmployeeObserver
>>> e
EmployeeId          : None (primary key)
LastName            : None
FirstName           : None
Title               : None
ReportsTo           : None (employees.EmployeeId)
BirthDate           : None
HireDate            : None
Address             : None
City                : None
State               : None
Country             : None
PostalCode          : None
Phone               : None
```



```
r=e.find_all()
>>> for elem in r:
...     print(elem.FirstName)
...
Andrew
Nancy
Jane
Margaret
Steve
Michael
```

```
Robert
Laura
>>>
```


Alright. Seems to work. There are 8 employees in the sample DB and we can access them.

Finally for the model, let's show the details of one employee.

```
>>> r.count()
8
>>> r[0]
EmployeeId          : 1 (primary key)
LastName            : Adams
FirstName           : Andrew
Title               : General Manager
ReportsTo           : None (employees.EmployeeId)
BirthDate           : 1962-02-18 00:00:00
HireDate            : 2002-08-14 00:00:00
Address             : 11120 Jasper Ave NW
City                : Edmonton
State               : AB
Country             : Canada
PostalCode          : T5K 2N1
Phone               : +1 (780) 428-9482
Fax                 : +1 (780) 428-3457
Email               : andrew@chinookcorp.com
>>>
```

Ok. so now we know anything about employee Andrew Adams.

Let's go ahead and create an API to make the data accessible for client applications

You will be maybe surprised that this step is even easier than the ones before. (And they weren't really hard as well honestly.) We want to create a REST API.

To create an API we need to generate a REST handler

```
python generate_handler.py -n employee -t sql --rest
```

This generates a handler called Employee for us that is automatically linked to our Employee model. This is where

convention over configuration

comes into l give a hint of which type the model is) than PythonOnWheels automatically includes links to the model into the handler. The

— *rest*

switch tells PythonOnWheels to also add the standard

CRUD

methods to the handler to Create, Read, Update and Delete data (employees)

You can find the generated handler in: handlers/employee.py . We will not go into details about the handler here since it has already everything we need. It can handle HTTP requests and will respond the right way that you typically expect for a REST API.

This is the API we generated.

See the extract of our handler below. The PythonOnWheels server will route all requests coming to /employee to the Employee handler. Based on the URI and HTTP method it will call the according handler methods. So a GET to /employee will call the list() method of the Employee handler. Which will by default return a list of all Employees. This should be normally exactly what we want.

```
@app.add_rest_routes("employee")
class Employee(PowHandler):
    """
    every pow handler automatically gets these RESTful routes
    when you add the : app.add_rest_routes() decorator.

    1  GET    /employee                          #=> list
    2  GET    /employee/<uuid:identifier>        #=> show
    3  GET    /employee/new                      #=> new
    4  GET    /employee/<uuid:identifier>/edit   #=> edit
    5  GET    /employee/page/<uuid:identifier>   #=> page
    6  GET    /employee/search                   #=> search
    7  PUT    /employee/<uuid:identifier>        #=> update
    8  PUT    /employee                          #=> update (You
have to send the id as json payload)
    9  POST   /employee                          #=> create
```

```
    10 DELETE /employee/<uuid:identifier>          #=> destroy
...
```


Everything prepared. Ready to serve!

Now, since we have a model that is successfully connected to the Database and a handler that can respond to HTTP requests and is linked to the model. We are ready to serve our API.

```
python server.py

-----------------------------------------------------------------
Final routes (order matters from here on ;)
-----------------------------------------------------------------
ROUTE 0 : pattern: /(robots\.txt)$
handler: StaticFileHandler
ROUTE 1 : pattern: /(favicon\.ico)$
handler: StaticFileHandler
ROUTE 2 : pattern: /static/(.*)$
handler: StaticFileHandler
ROUTE 3 : pattern: /employee/search/?(?:/?\.\w+)?/?$
handler: Employee
ROUTE 4 : pattern: /employee/list/?(?:/?\.\w+)?/?$
handler: Employee
ROUTE 5 : pattern: /employee/new/?(?:/?\.\w+)?/?$
handler: Employee
ROUTE 6 : pattern: /employee/page/?(?P<page>[0-9\-a-zA-Z]+)?/?(?:/?
handler: Employee
ROUTE 7 : pattern: /employee/show/?(?P<id>[0-9\-a-zA-Z]+)?/?(?:/?\.
handler: Employee
ROUTE 8 : pattern: /employee/(?P<id>[0-9\-a-zA-Z]+)/edit/?(?:/?\.\w
handler: Employee
ROUTE 9 : pattern: /employee/(?P<id>[0-9\-a-zA-Z]+)?/?(?:/?\.\w+)?/
handler: Employee
ROUTE 10: pattern: /employee/?(?:/?\.\w+)?/?$
handler: Employee
ROUTE 11: pattern: /test/(?P<id>\d+)$
handler: PyTestHandler
ROUTE 12: pattern: /test/(?P<identifier>[A-Fa-f0-9]{8}-[A-Fa-f0-9]{
handler: PyTestHandler
ROUTE 13: pattern: /testresults(?:/?\.\w+)?/?$
handler: PyTestHandler
ROUTE 14: pattern: /index/([0-9]+)(?:/?\.\w+)?/?$
handler: IndexdHandler
ROUTE 15: pattern: /index/(?P<identifier>[A-Fa-f0-9]{8}-[A-Fa-f0-9]
handler: IndexdHandler
ROUTE 16: pattern: /hello(?:/?\.\w+)?/?$
handler: HelloHandler
ROUTE 17: pattern: /(?:/?\.\w+)?/?$
handler: IndexdHandler
ROUTE 18: pattern: .*(?:/?\.\w+)?/?$
```

```
handler: ErrorHandler
-------
startin
-------
visit: http://localhost:8080
starting...
```


Let's test our API with curl.

The first test is to list all employees via HTTP.

curl

And this is what comes back:

curl -H "Content-Type: application/json" -X GET http://localhost:8080/employee


```
{"message": "employee, index", "http_status": 200, "prev": null,
"next": null, "data": [{"EmployeeId": 1, "LastName": "Adams",
"FirstName": "Andrew", "Title": "General Manager", "BirthDate":
"1962-02-18 00:00:00", "HireDate": "2002-08-14 00:00:00", "Address":
"11120 Jasper Ave NW", "City": "Edmonton", "State": "AB", "Country":
"Canada", "PostalCode": "T5K 2N1", "Phone": "+1 (780) 428-9482",
"Fax": "+1 (780) 428-3457", "Email": "andrew@chinookcorp.com"},
{"EmployeeId": 2, "LastName": "Edwards", "FirstName": "Nancy",
"Title": "Sales Manager", "BirthDate": "1958-12-08 00:00:00",
"HireDate": "2002-05-01 00:00:00", "Address": "825 8 Ave SW",
"City": "Calgary", "State": "AB", "Country": "Canada", "PostalCode":
"T2P 2T3", "Phone": "+1 (403) 262-3443", "Fax": "+1 (403) 262-3322",
"Email": "nancy@chinookcorp.com"}, {"EmployeeId": 3, "LastName":
"Peacock", "FirstName": "Jane", "Title": "Sales Support Agent",
"BirthDate": "1973-08-29 00:00:00", "HireDate": "2002-04-01
00:00:00", "Address": "1111 6 Ave SW", "City": "Calgary", "State":
"AB", "Country": "Canada", "PostalCode": "T2P 5M5", "Phone": "+1
(403) 262-3443", "Fax": "+1 (403) 262-6712", "Email":
"jane@chinookcorp.com"}, {"EmployeeId": 4, "LastName": "Park",
"FirstName": "Margaret", "Title": "Sales Support Agent",
"BirthDate": "1947-09-19 00:00:00", "HireDate": "2003-05-03
00:00:00", "Address": "683 10 Street SW", "City": "Calgary",
"State": "AB", "Country": "Canada", "PostalCode": "T2P 5G3",
"Phone": "+1 (403) 263-4423", "Fax": "+1 (403) 263-4289", "Email":
"margaret@chinookcorp.com"}, {"EmployeeId": 5, "LastName":
"Johnson", "FirstName": "Steve", "Title": "Sales Support Agent",
"BirthDate": "1965-03-03 00:00:00", "HireDate": "2003-10-17
00:00:00", "Address": "7727B 41 Ave", "City": "Calgary", "State":
"AB", "Country": "Canada", "PostalCode": "T3B 1Y7", "Phone": "1
(780) 836-9987", "Fax": "1 (780) 836-9543", "Email":
"steve@chinookcorp.com"}, {"EmployeeId": 6, "LastName": "Mitchell",
"FirstName": "Michael", "Title": "IT Manager", "BirthDate": "1973-
07-01 00:00:00", "HireDate": "2003-10-17 00:00:00", "Address": "5827
```

```
Bowness Road NW", "City": "Calgary", "State": "AB", "Country":
"Canada                                                    7",
"Fax":                                                     "},
{"Emplo
"Title".  "IT Staff",  "BirthDate":  "1970-05-29 00:00:00",  "Hi eDate":
"2004-01-02 00:00:00", "Address": "590 Columbia Boulevard West",
"City": "Lethbridge", "State": "AB", "Country": "Canada",
"PostalCode": "T1K 5N8", "Phone": "+1 (403) 456-9986", "Fax": "+1
(403) 456-8485", "Email": "robert@chinookcorp.com"}, {"EmployeeId":
8, "LastName": "Callahan", "FirstName": "Laura", "Title": "IT
Staff", "BirthDate": "1968-01-09 00:00:00", "HireDate": "2004-03-04
00:00:00", "Address": "923 7 ST NW", "City": "Lethbridge", "State":
"AB", "Country": "Canada", "PostalCode": "T1H 1Y8", "Phone": "+1
(403) 467-3351", "Fax": "+1 (403) 467-8772", "Email":
"laura@chinookcorp.com"}]}
```


So this is working! We get back the 8 employees.

Here is the nicer formatted view using insomnia.

Done!

You can of course also use the other API endpoints to update, delete or create new data. Just give it a try.

This is what PythonOnWheels is all about

> Make you focus on your App and not on the boilerplate and boring stuff to get there.

I think the API Creation for an existing datasource is a good example for the workflow in PythonOnWheels. We didn't have to configure a lot (just the DB name). We generated the model and the handler and didn't need to change a lot there to get what we wanted. The model is completely working and normally does not need to be changed at all. We acutally added value since the reflected model has a cerberus schema which adds validation out of the box for us.

Hope you enjoyed it!

If so I am happy to receive some feedback via twitter @pythononwheels. Or check the PythonOnWheels Homepage.

Credit: Photo by vipul uthaiah on Unsplash

API Python Sql NoSQL Database

Follow

Written by Klaas (khz)

32 Followers

11 to 1 pm spare time software developer. Mostly python.

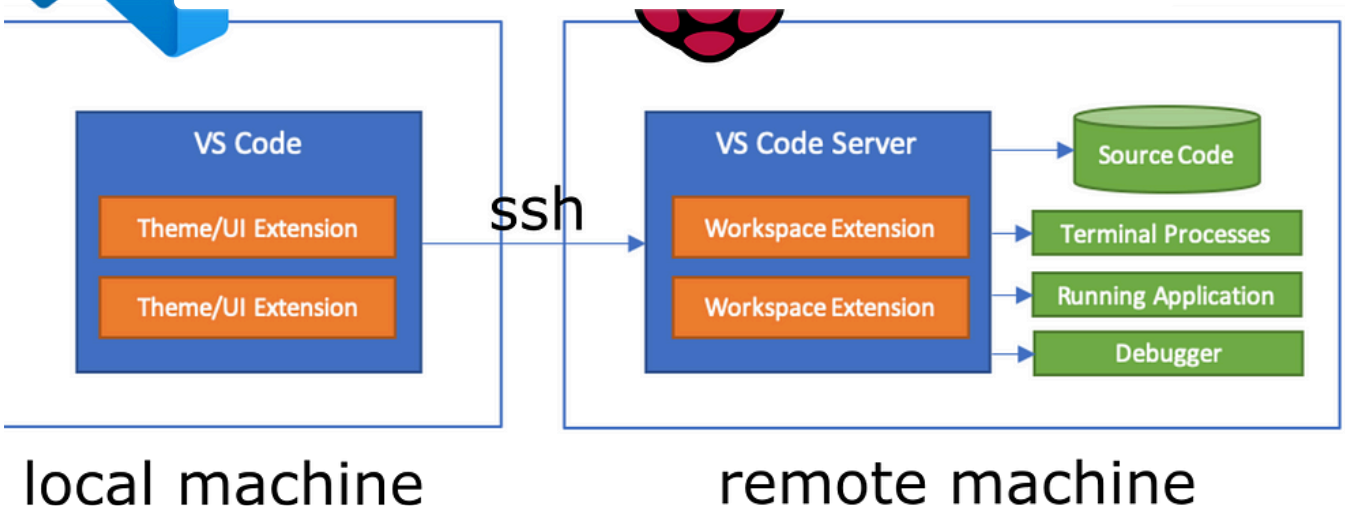
More from Klaas (khz)

Klaas (khz)

Reflecting an existing SQL Database schema with python in minutes with PythonOnWheels.

PythonOnWheels is a layer of glue around some great existing libraries/modules and tools to make your python life easier for the boring...

May 19, 2019 7



Klaas (khz)

Remote development on a raspberry pi with ssh and VSCode

Remote development is a really cool and useful feature of VSCode. It makes developing on the remote raspi feeling like you were developing...

Dec 28, 2019 72 8

Klaas (khz)

Switching from a Mac mini 2012 to an Intel NUC8i5BEK and Ubuntu.

I started buying (used) Macs starting with the white iMac 20", I think this was around 2008. I was really attracted to have a silent...

Sep 10, 2019

Klaas (khz) in Analytics Vidhya

Validating SQL and NoSQL data models with python and cerberus for a Tornado/PythonOnWheels web...

In this short (hands-on) tutorial I will give you a very basic introduction into using the great cerberus library to define and validate...

Sep 4, 2019 31

See all from Klaas (khz)

Recommended from Medium

Amazon.com — Seattle, WA
Software Development Engineer — Mar 2020 – May 2021

- [illegible] ctions
- Implemented [illegible] nt CSRF,
- [illegible] insactions and reduce call center costs by $25 Million
- Recovered Saudi Arabia checkout failure impacting 4000+ customers due to incorrect GET form redirection

Projects

NinjaPrep.io (React)
- Platform to offer coding problem practice with built in code editor and written + video solutions in React
- Utilized Nginx to reverse proxy IP address on Digital Ocean hosts
- Developed using Styled-Components for 95% CSS styling to ensure proper CSS scoping
- Implemented Docker with Seccomp to safely run user submitted code with < 2.2s runtime

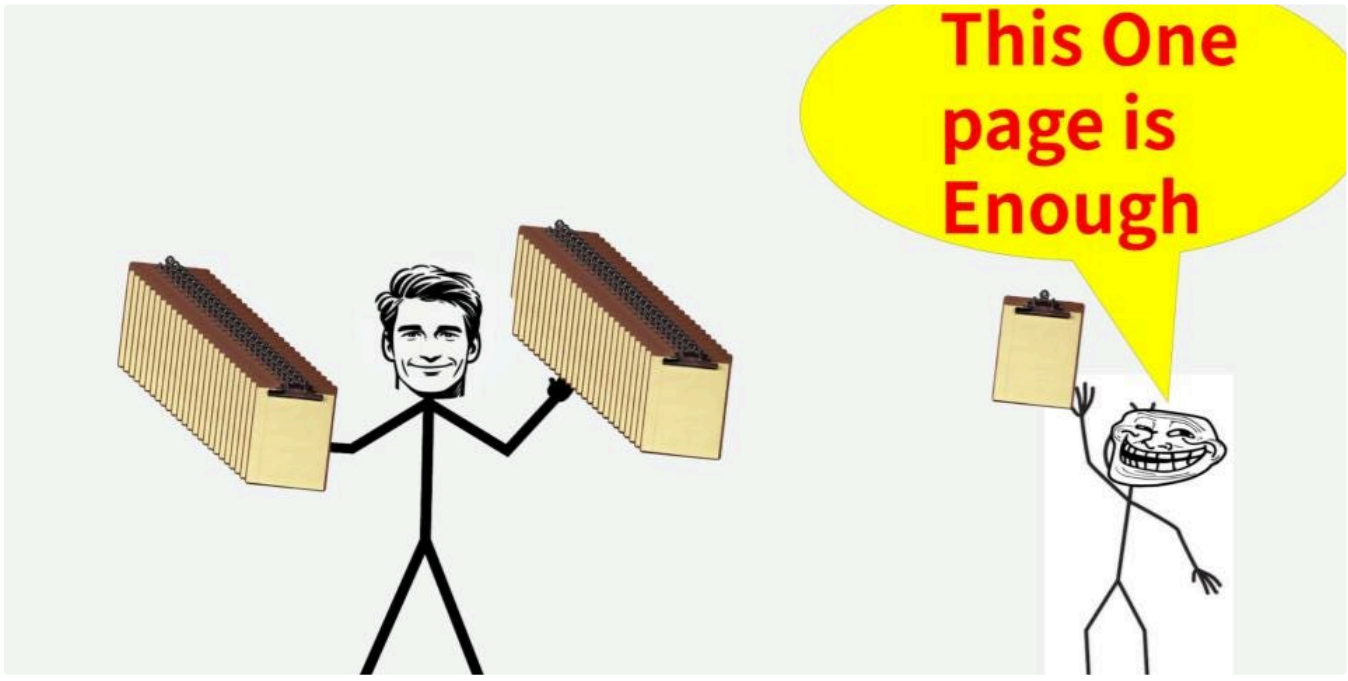
HeatMap (JavaScript)
- Visualized Google Takeout location data of location history using Google Maps API and Google Maps heatmap code with React
- Included local file system storage to reliably handle 5mb of location history data
- Implemented Express to include routing between pages and jQuery to parse Google Map and implement heatmap overlay

Alexander Nguyen in Level Up Coding

The resume that got a software engineer a $300,000 job at Google.

1-page. Well-formatted.

Jun 1 · 23K · 470



Ajay Parmar in Top Python Libraries

11-Python Packages That Turn Hundreds of Lines of Code into One

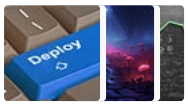
🚀 11 Python Libraries That Turn 100 Lines of Code Into One: A Fascinating Journey

Sep 28 · 668 · 6

Lists

Coding & Development

11 stories · 837 saves



Predictive Modeling w/ Python

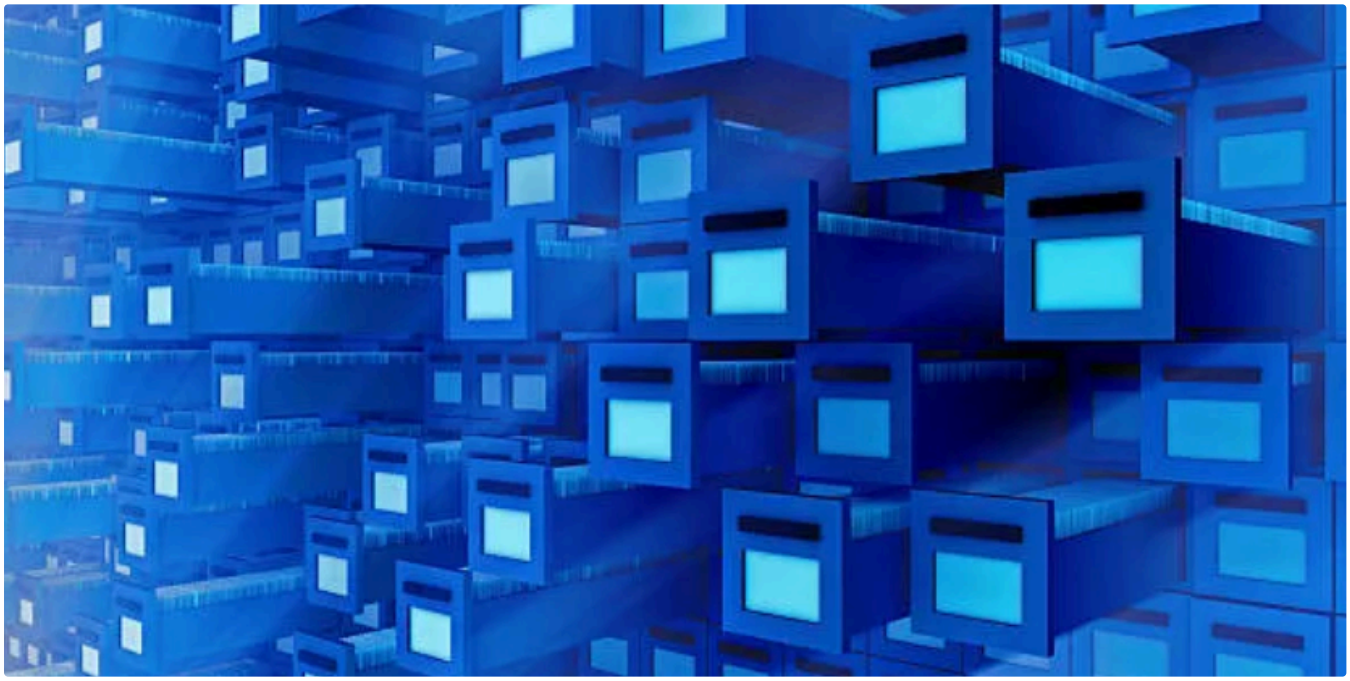
20 stories · 1584 saves

Practical Guides to Machine Learning

10 stories · 1926 saves

ChatGPT

21 stories · 828 saves

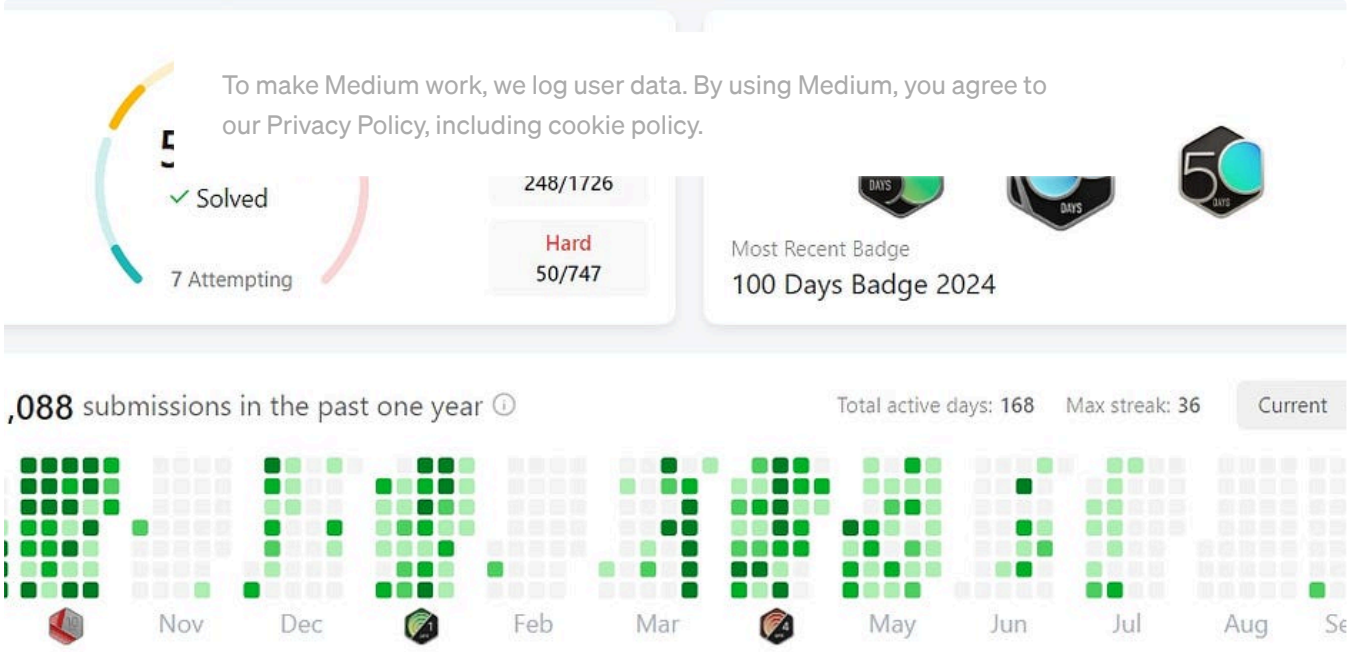


Paul Hoke

The Ultimate Guide to SQLAlchemy: Powering Your Python Database Operations

Embark on a comprehensive journey through SQLAlchemy, mastering Python's most powerful ORM from basic concepts to advanced techniques, with...

Aug 23



Surabhi Gupta in Code Like A Girl

Why 500 LeetCode Problems Changed My Life

How I Prepared for DSA and Secured a Role at Microsoft

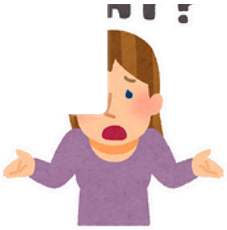
Sep 26 2.1K 54

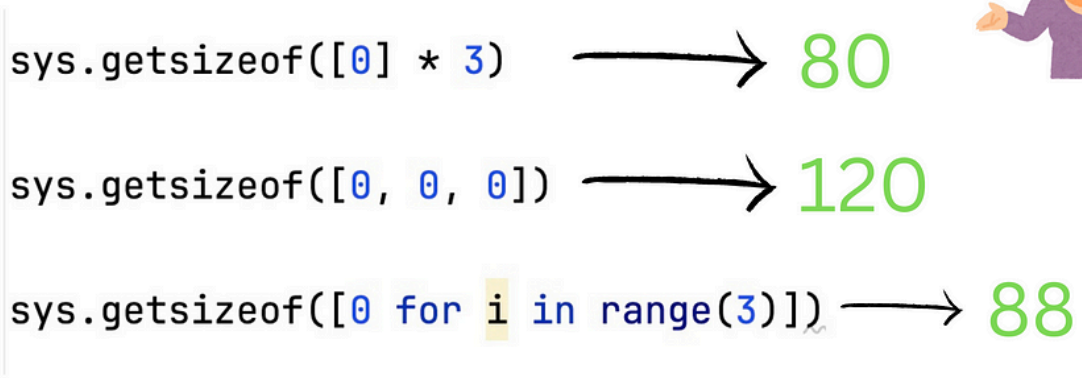
Amir Lavasani

How to Structure Your FastAPI Projects

Part 1: Blueprint

May 14 587 9





Shuai Li in Programming Domain

Most Developers Failed with this Senior-Level Python Interview Question

What is the difference between [0] * 3 and [0, 0, 0]?

Aug 9 3.3K 48