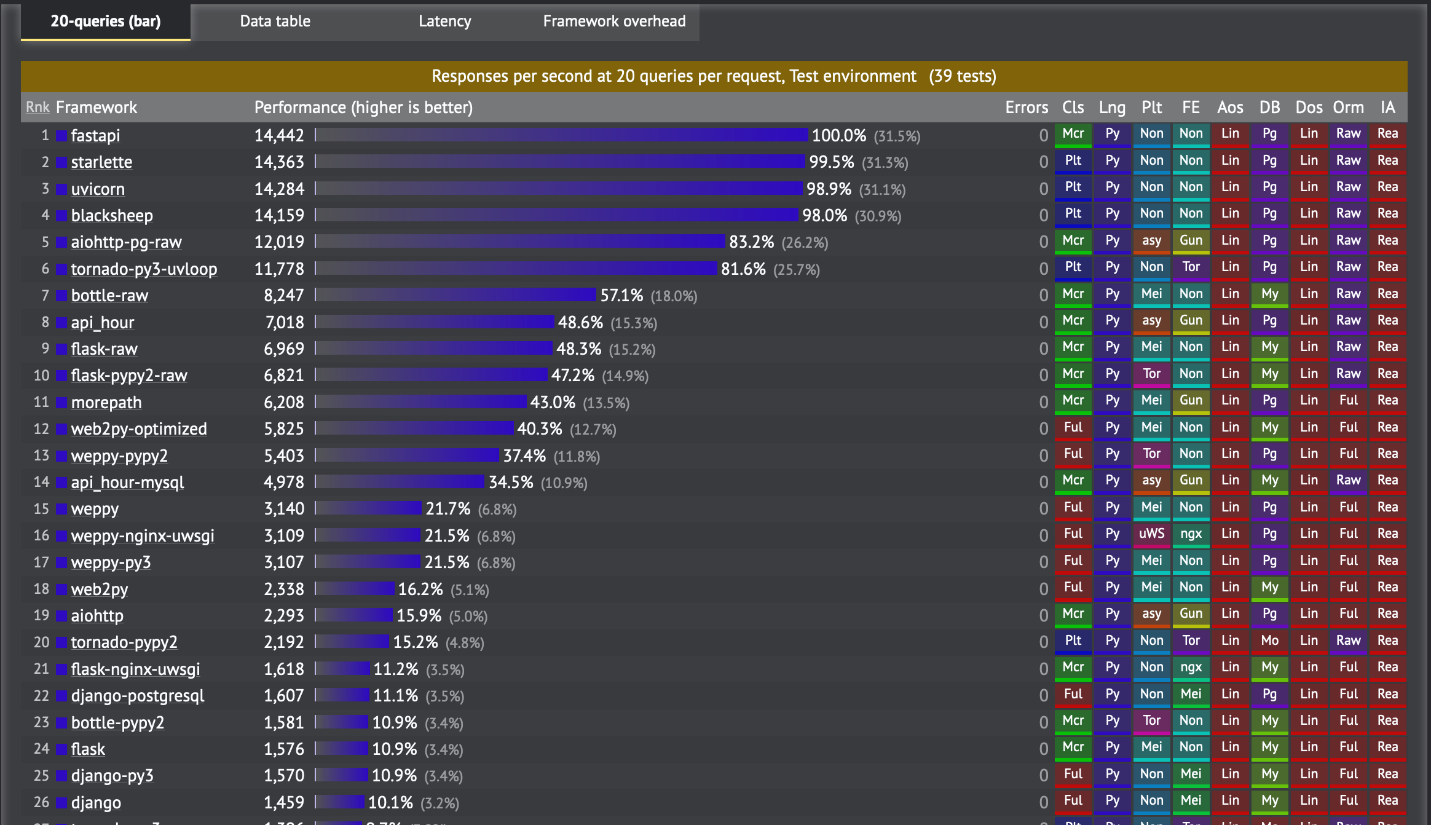
FAST API

*FastAPI is a modern, async, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints.*



* **Fast**: Very high performance, on par with **NodeJS** and **Go** (with Starlette and Pydantic). One of the fastest Python frameworks available.
* **Fast to code**: Increase the speed to develop features by about 200% to 300%. \*
* **Fewer bugs**: Reduce about 40% of human (developer) induced errors. \*
* **Intuitive**: Great editor support. Completion everywhere. Less time debugging.
* **Easy**: Designed to be easy to use and learn. Less time reading docs with built-in swagger UI support
* **Short**: Minimize code duplication. Multiple features from each parameter declaration. Fewer bugs.
* **Robust**: Get production-ready code. With automatic interactive documentation with git support for CI/CD.
* **Standards-based**: Based on (and fully compatible with) the open standards for APIs(Open API)

Setup:

Install Python 3.9.x from python.org web site.

Upgrade pip with latest version:

d:\>python -m pip install --upgrade pip

install Pipenv via PyPy by running  for pipfile configuration.

d:\>pip install pipenv

Activate the virtual environment by running the command

D:\>md fast-api-demo

D:\>cd fast-api-demo

D:\fast-api-demo>pipenv shell

…….

Successfully created virtual environment!

Virtualenv location: C:\Users\asus\.virtualenvs\fast-api-demo-ez6srkeY

Creating a Pipfile for this project...

Launching subshell in virtual environment...

Microsoft Windows [Version 10.0.18363.1440]

(c) 2019 Microsoft Corporation. All rights reserved.

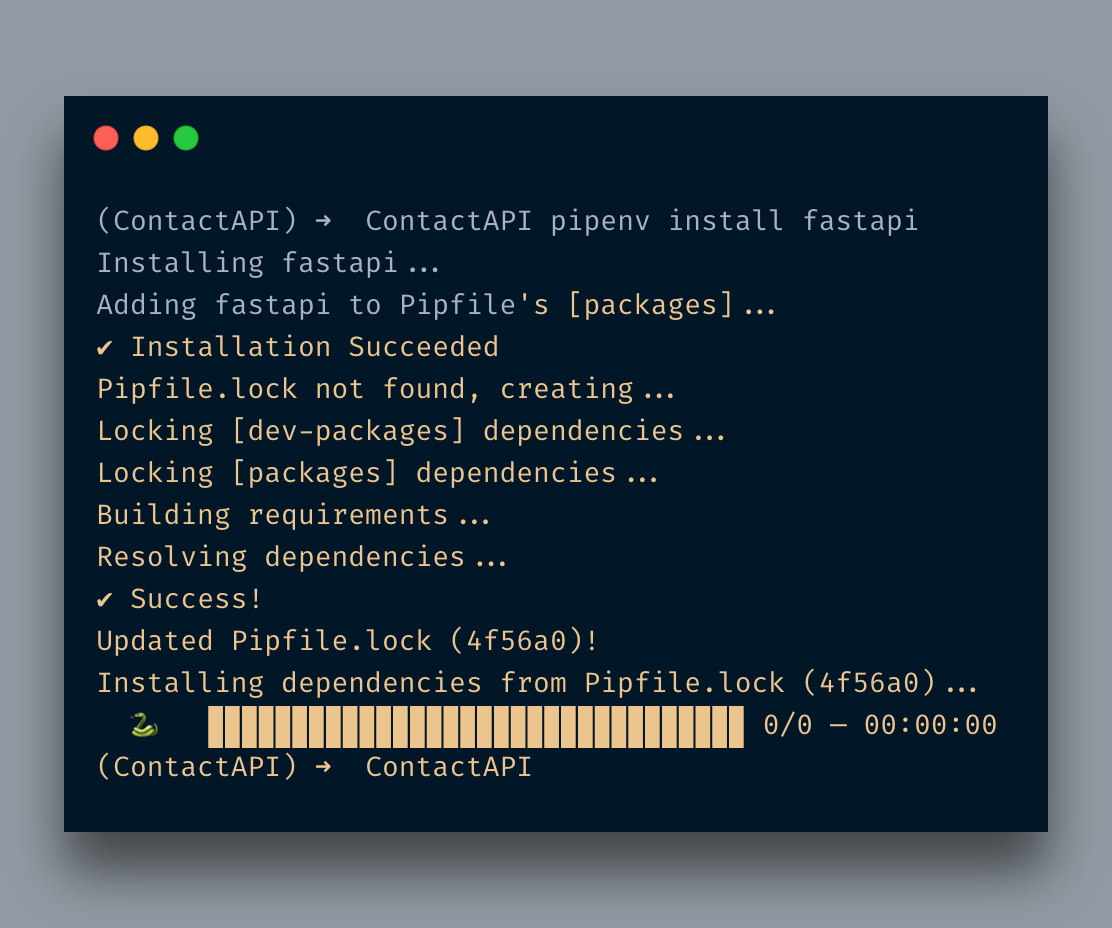
(fast-api-demo-ez6srkeY) D:\fast-api-demo>

Observe pipfile content.

install **FastAPI** by running the following command:

(fast-api-demo-ez6srkeY) D:\fast-api-demo>

pipenv install fastapi



create a file called main.py

from fastapi import FastAPI

app = FastAPI()

@app.get("/")

def home():

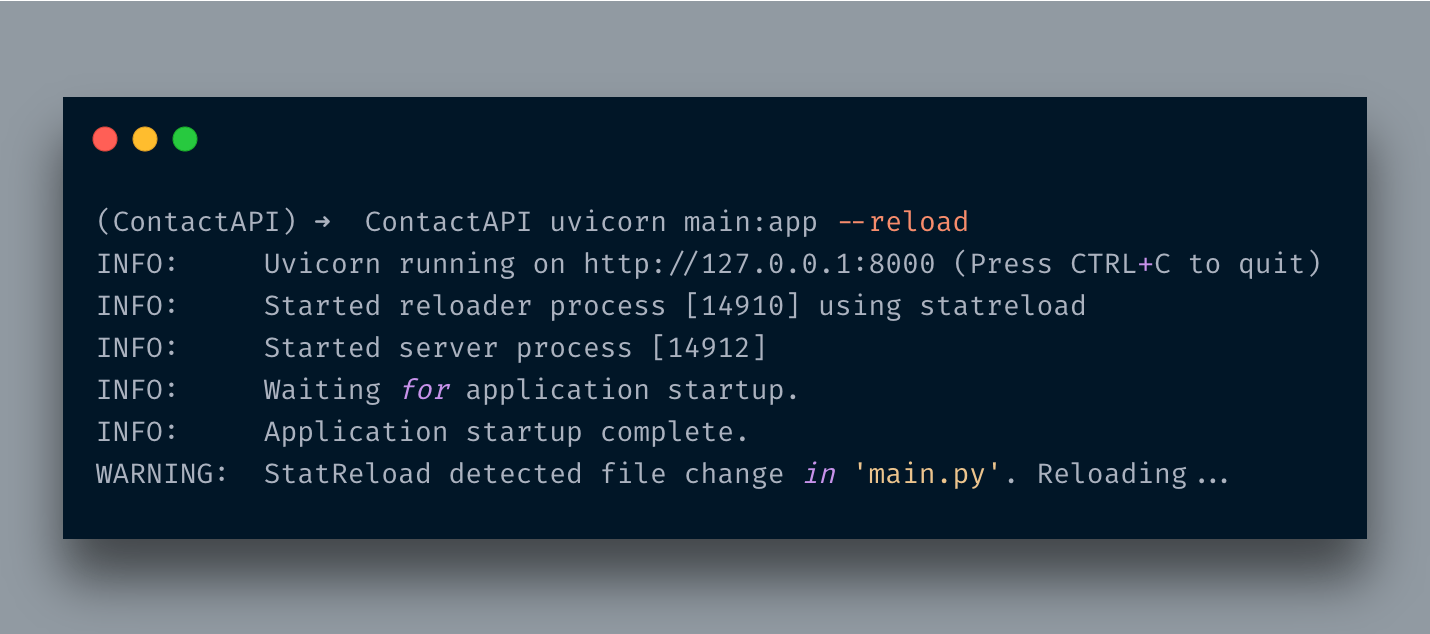
    return {"Hello": "FastAPI"}

FastAPI comes with the [uvicorn](https://www.uvicorn.org/) which is an ASGI server. You will simply be running the command

(fast-api-demo-ez6srkeY) D:\fast-api-demo>

uvicorn main:app –reload

provide the file name(**main**.py) and the class object(**app**) and it will initiate the server.  --reload flag so that it reloads itself after every change. ( we can change filename and app to something..)



Visit http://localhost:8000/ and see the message in JSON format {"Hello":"FastAPI"}

FastAPI provides an API document engine too. If you visit http://localhost:8000/docs which is using the Swagger UI interface.

 Need something fancy then visit <http://localhost:8080/redoc>

FastAPI also provides an OpenAPI version of API endpoints, like this <http://127.0.0.1:8000/openapi.json>

Path and Parameters

[@app](http://twitter.com/app).get("/contact/{contact\_id}")  
def contact\_details(contact\_id: int):  
return {'contact\_id': contact\_id}

contact\_details that accepts only an int parameter and just returns it as it in a dict format.

Query String

What if you pass extra data in the form of query strings? For instance your API end-point returns loads of records hence you need pagination. Well, no issue, you can fetch that info as well.

First, we will import the Optional type:

from typing import Optional

[@app](http://twitter.com/app).get("/contact/{contact\_id}")  
def contact\_details(contact\_id: int, page: Optional[int] = 1):  
 if page:  
 return {'contact\_id': contact\_id, 'page': page}  
 return {'contact\_id': contact\_id}

Pydantic models actually help in data validation, It means it makes sure that the data which is being passed is valid, if not otherwise it returns an error.

from typing import Optionalfrom fastapi import FastAPI  
from pydantic import BaseModel  
app = FastAPI()class Contact(BaseModel):  
 contact\_id:int  
 first\_name:str  
 last\_name:str  
 user\_name:str  
 password:str

[@app](http://twitter.com/app).post('/contact')  
async def create\_contact(contact: Contact):  
 return contact

Async converted to coroutine:

Go to <http://localhost:8080/docs> and check

As expected it just returned the Contact object in JSON format.

# What is Response Model

As the name suggests, a Response Model is a model that is used while sending a response against a request. Basically, when you just used a model it just returns all fields. By using a response model you can control what kind of data should be returned back to the user

class Contact(BaseModel):  
 contact\_id:int  
 first\_name:str  
 last\_name:str  
 user\_name:str  
 password:strclass ContactOut(BaseModel):  
 contact\_id:int  
 first\_name:str  
 last\_name:str  
 user\_name:str[@app](http://twitter.com/app).get("/")  
def home():  
 return {"Hello": "FastAPI"}[@app](http://twitter.com/app).post('/contact', response\_model=ContactOut)  
async def create\_contact(contact: Contact):  
 return contact

The only thing which is different here is the absence of the password field. In order to use it, we are going to assign it in the response\_model parameter of the post decorator. That's it. Now when I run hit the same URL it will not return the password field.

Or

[@app](http://twitter.com/app).post('/contact', response\_model=Contact, response\_model\_exclude={"password"})  
async def create\_contact(contact: Contact):  
return contact

You can also attach metadata with your API endpoint.

[@app](http://twitter.com/app).post('/contact', response\_model=Contact, response\_model\_exclude={"password"},description="Create a single contact")  
async def create\_contact(contact: Contact):  
 return contact

Observe metadata in docs of swagger.

# Error handling in FastAPI

It is always possible that you do not get the required info. FastAPI provides HTTPException class to deal with such situations.

[@app](http://twitter.com/app).get("/contact/{id}", response\_model=Contact, response\_model\_exclude={"password"},description="Fetch a single contact")  
async def contact\_details(id: int):  
 if id < 1:  
 raise HTTPException(status\_code=404, detail="The required contact details not found")  
 contact = Contact(contact\_id=id, first\_name='Sriram', last\_name='murthy', user\_name='welcome1', password='welcome34')  
 return contact

Custom headers:

from fastapi import FastAPI, HTTPException, Response[@app](http://twitter.com/app).get("/contact/{id}", response\_model=Contact, response\_model\_exclude={"password"},  
 description="Fetch a single contact")  
async def contact\_details(id: int, response: Response):  
 response.headers["X-LOL"] = "1"  
 if id < 1:  
 raise HTTPException(status\_code=404, detail="The required contact details not found")  
 contact = Contact(contact\_id=id, first\_name='Adnan', last\_name='Siddiqi', user\_name='adnan1', password='adn34')  
 return contact