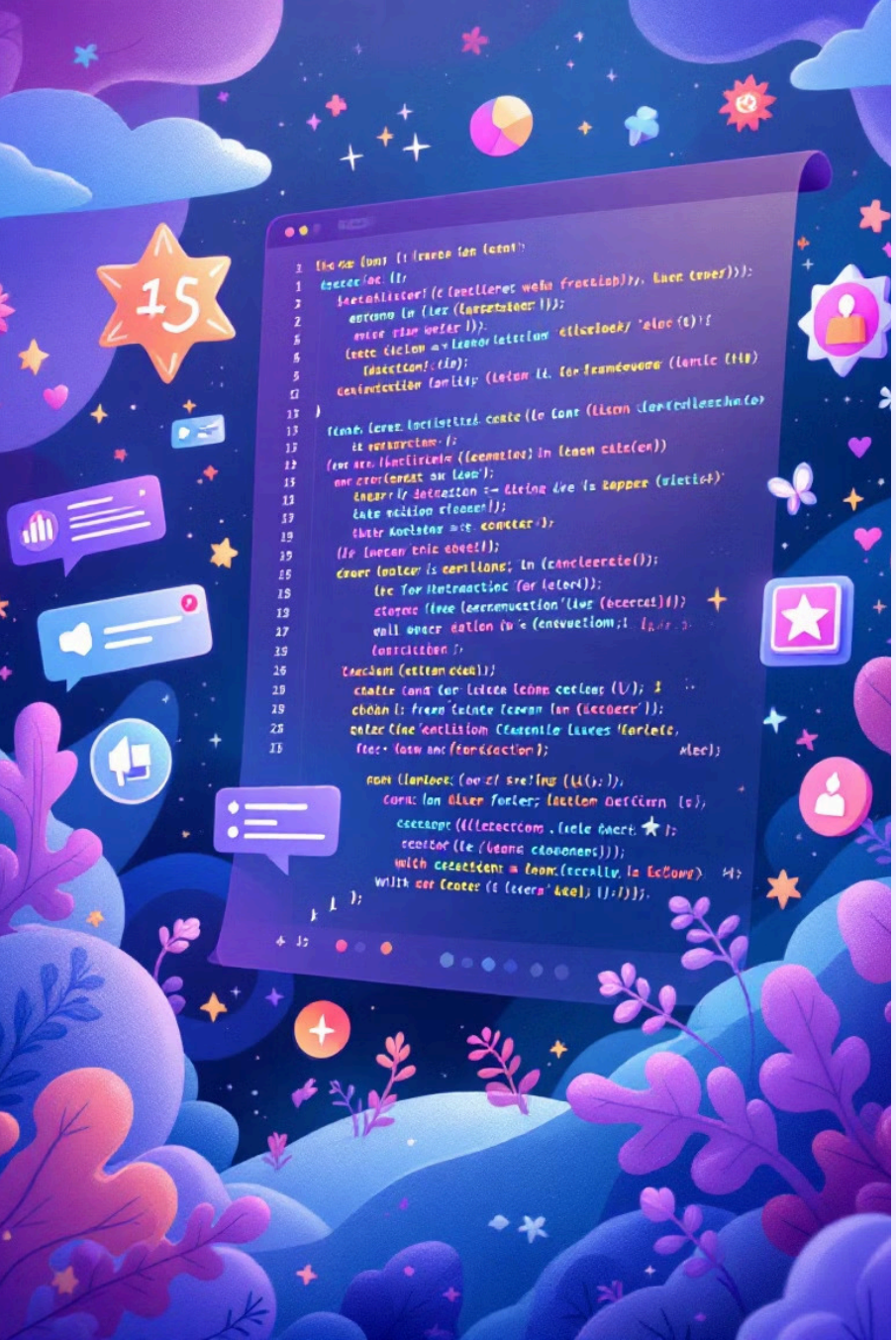
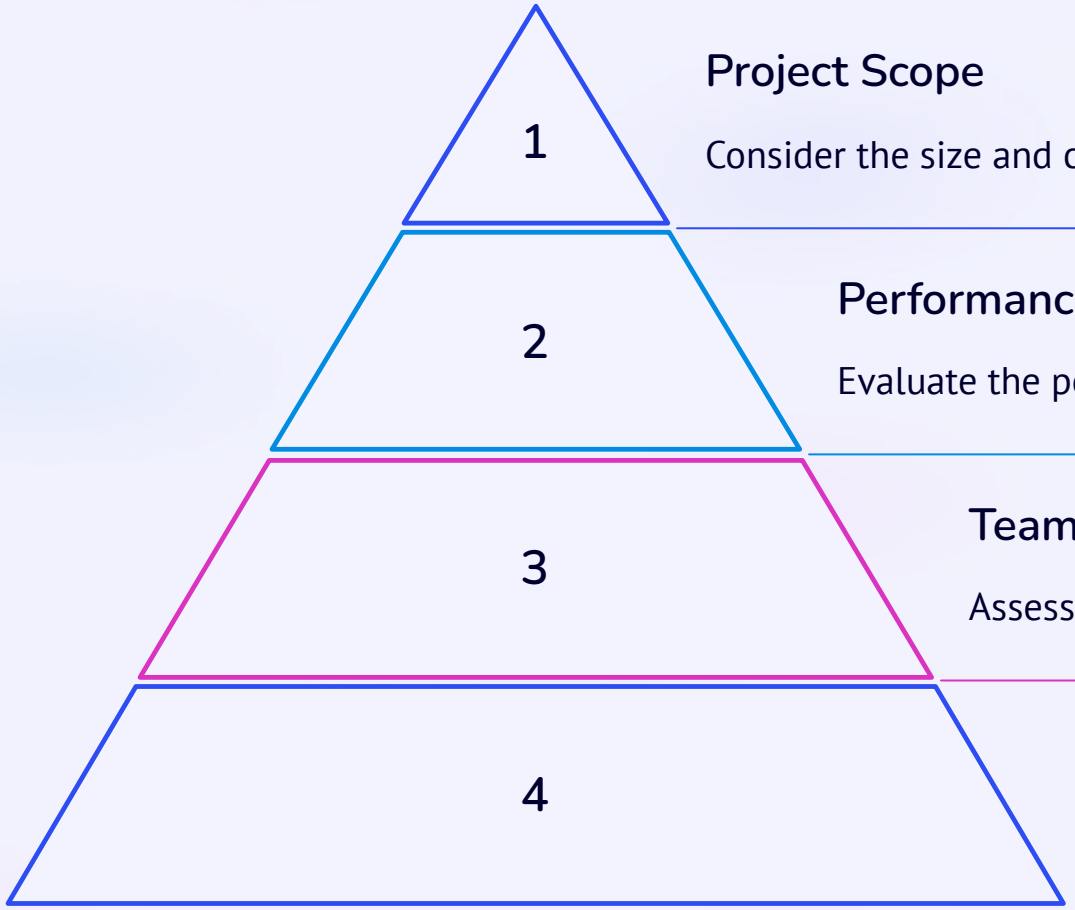


Web Frameworks in Python: A Comparative Guide

Today, we'll delve into the world of Python web frameworks, comparing popular options to help you choose the best fit for your next project.

Choosing the Right Framework



Project Scope

Consider the size and complexity of your application.

Performance

Evaluate the performance requirements for your application.

Team Expertise

Assess the skills and experience of your development team.

Development Speed

Prioritize rapid development and efficient deployment.

Django:

Robust and Feature-Rich

Follows MVT architecture.

All-in-One

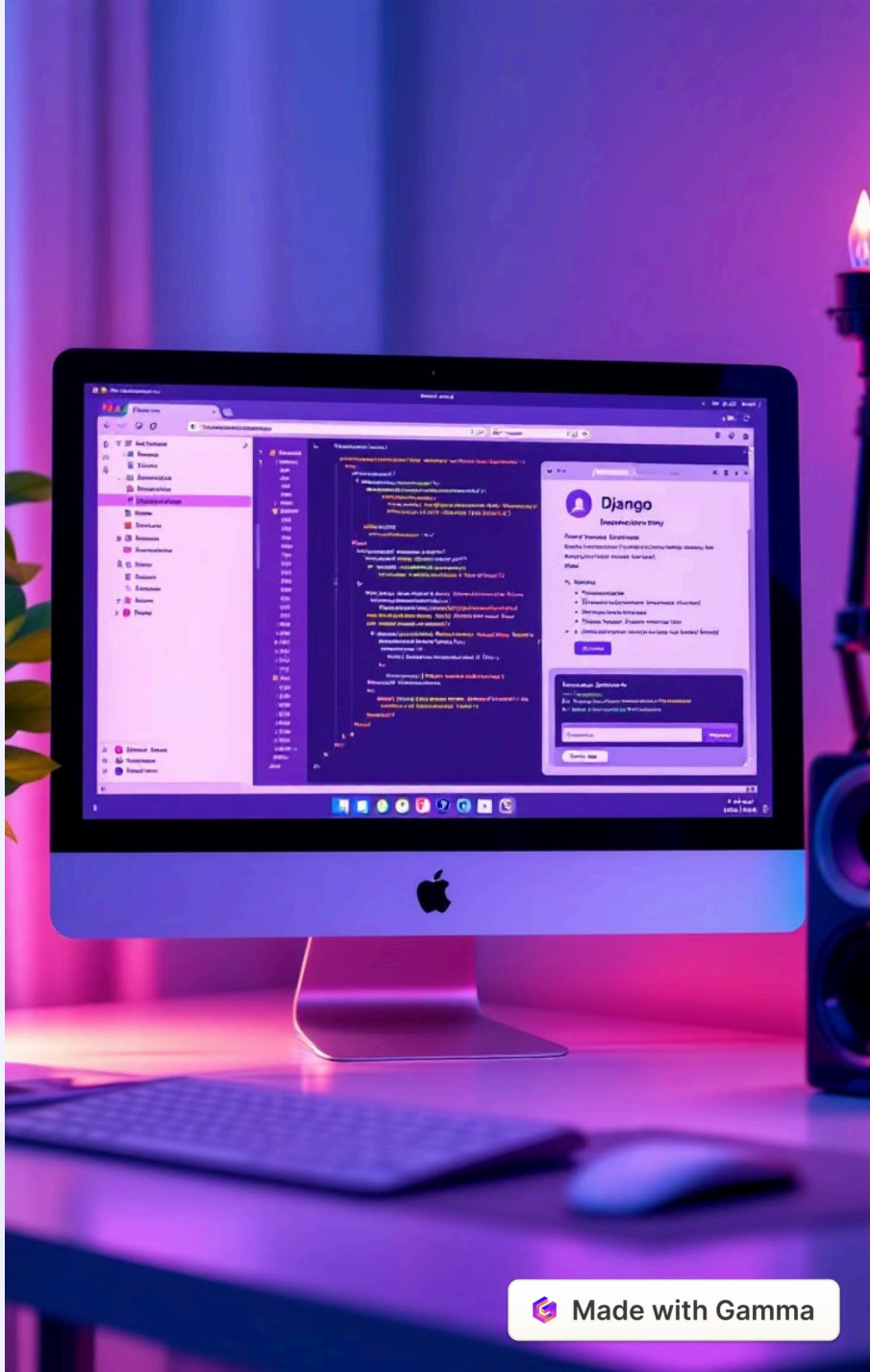
Django provides a comprehensive suite of features for building complex web applications, including a powerful ORM and templating engine.

Rapid Development

Django's batteries-included approach allows for rapid development and efficient deployment, ideal for large-scale projects.

Strong Community

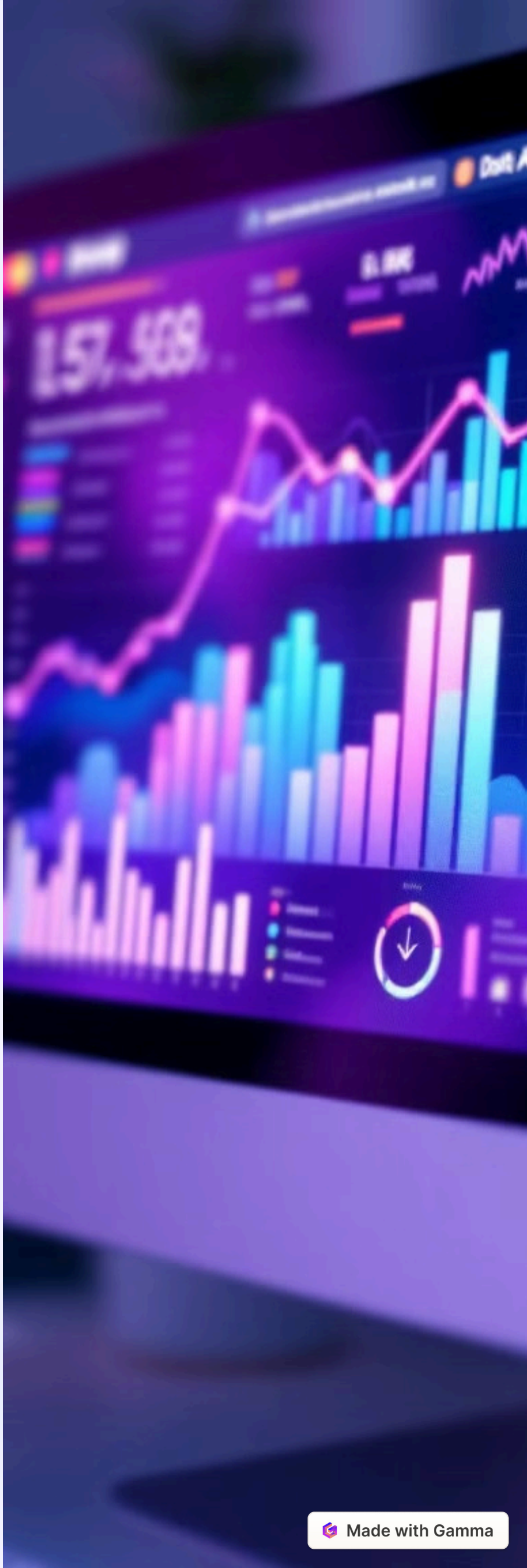
Django benefits from a vast and active community, providing ample resources and support.



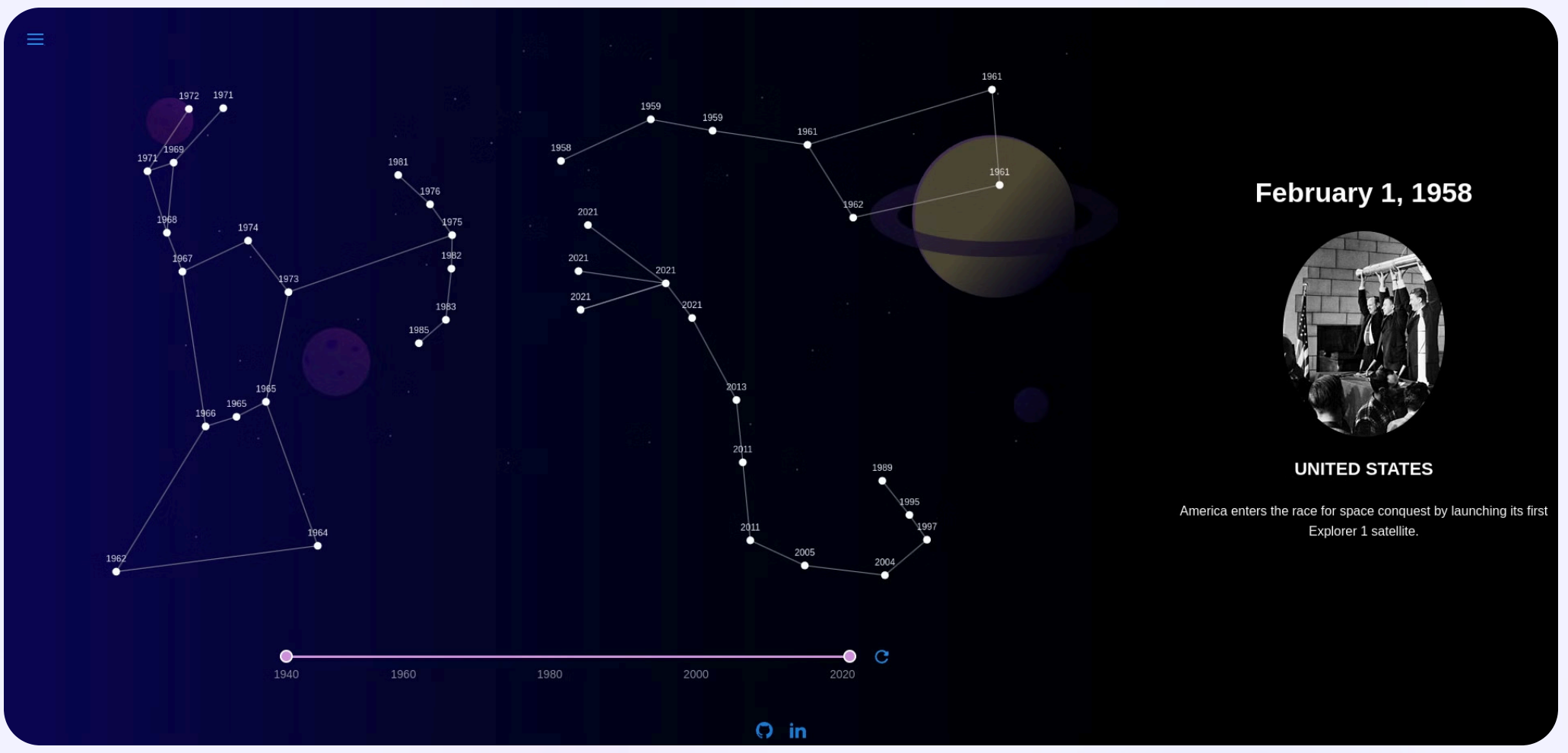
Dash: low-code framework for rapidly building data apps in Python

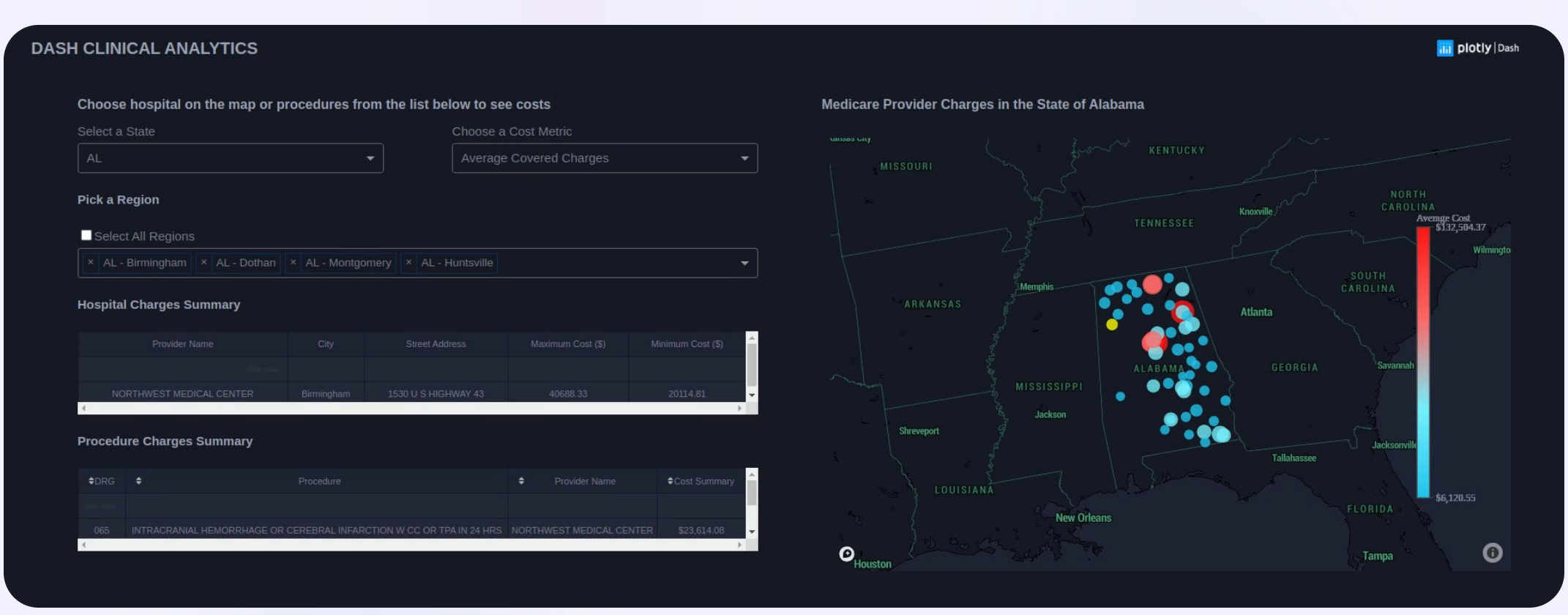
- Component-based: HTML, DataTables, Canvas, Slider
- Databricks integration
- Community: leaflet, parallel computing

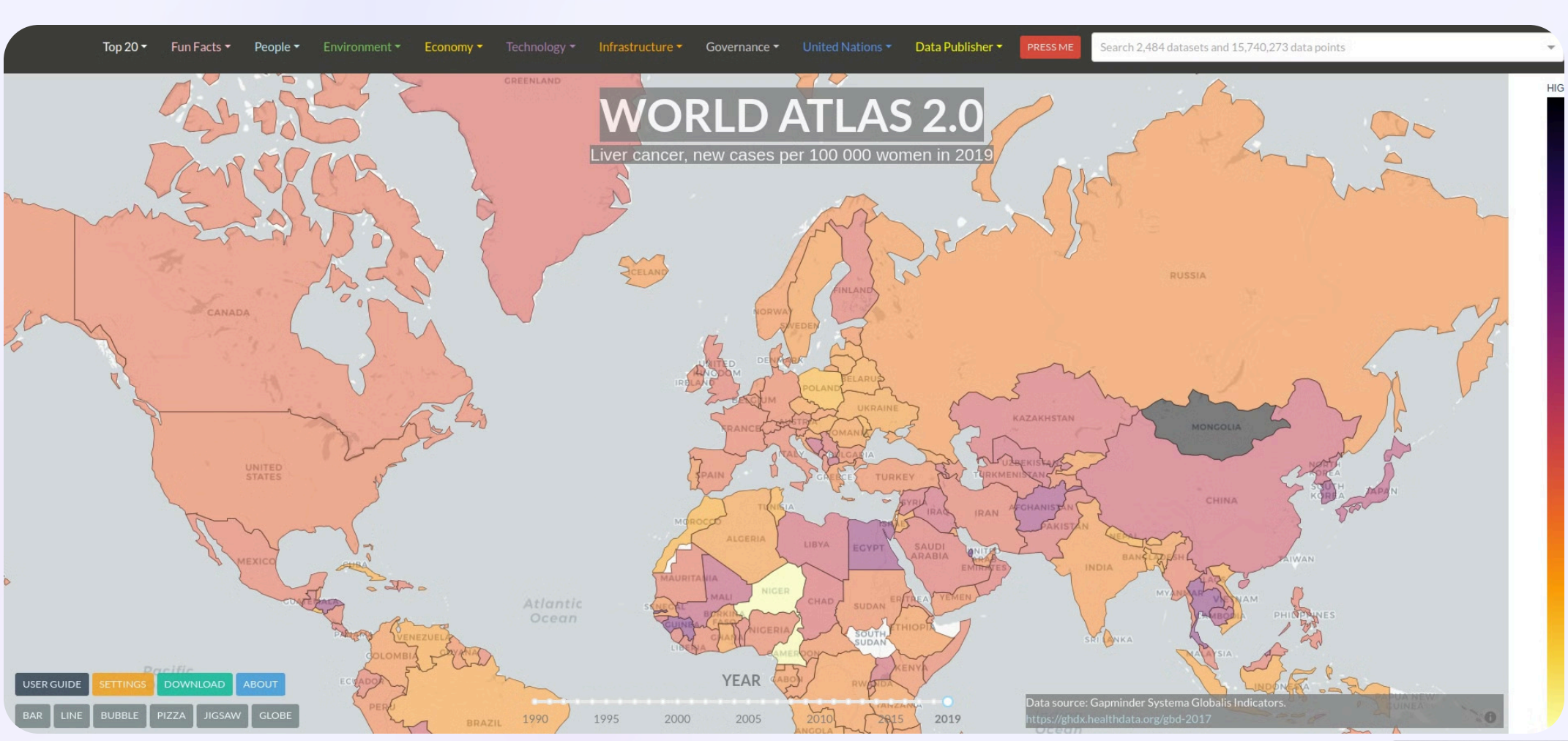
```
from dash import Dash, html, dcc, callback, Output, Input
import plotly.express as px
import pandas as pd

df =
pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/
master/gapminder_unfiltered.csv')

app = Dash()

app.layout = [
    html.H1(children='Title of Dash App', style=
{'textAlign':'center'}),
    dcc.Dropdown(df.country.unique(), 'Canada', id='dropdown-
selection'),
    dcc.Graph(id='graph-content')
]

@callback(
    Output('graph-content', 'figure'),
    Input('dropdown-selection', 'value')
)
def update_graph(value):
    dff = df[df.country==value]
    return px.line(dff, x='year', y='pop')

if __name__ == '__main__':
    app.run(debug=True)
```

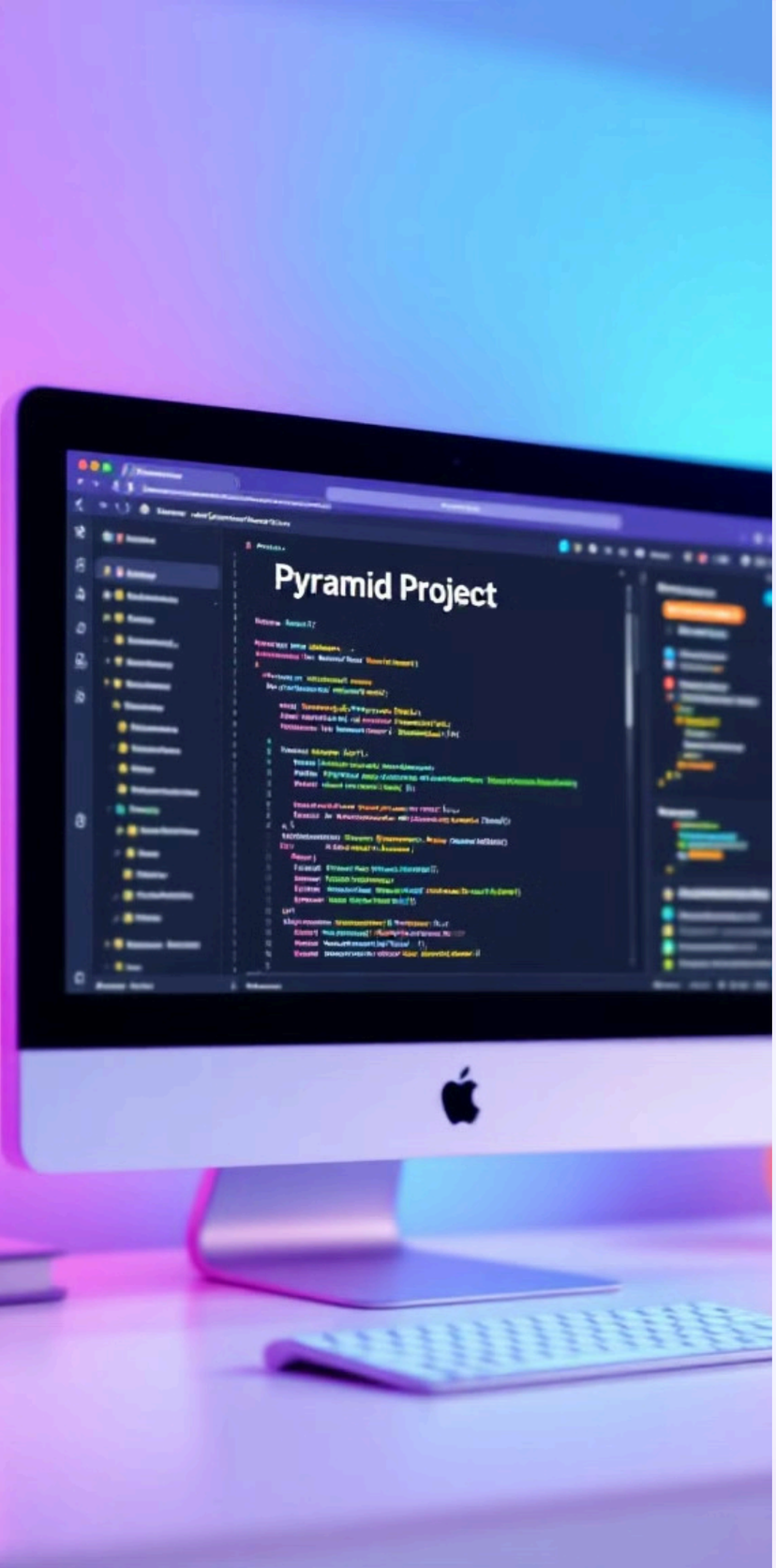


Dash: Examples







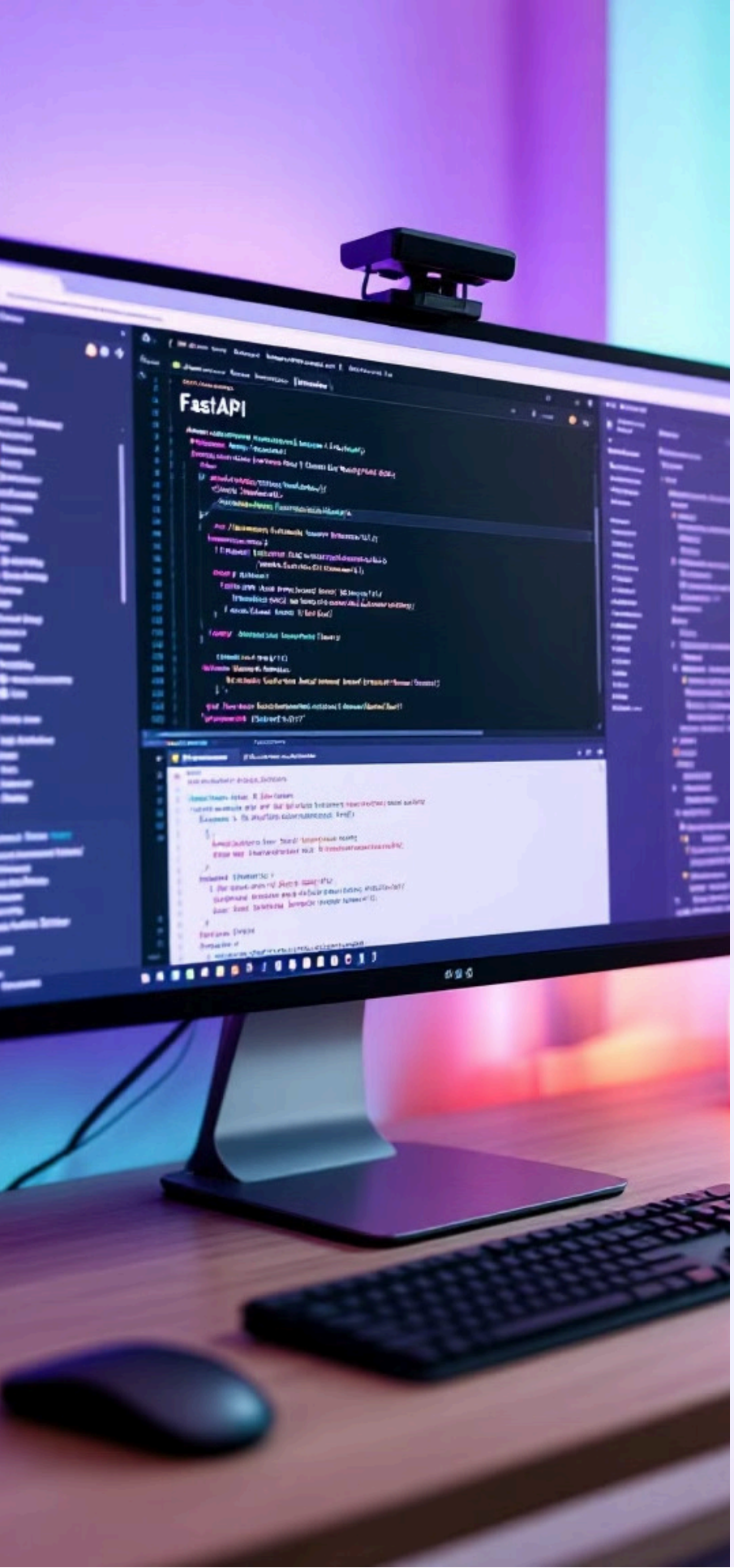


Pyramid

Pyramid is designed to scale efficiently, handling complex projects and growing user bases.

- Debugtoolbar for easy debugging.
- Easy configuration, up and running.
- Support object-oriented views.

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response

def hello_world(request):
    return Response('Hello World!')

if __name__ == '__main__':
    with Configurator() as config:
        config.add_route('hello', '/')
        config.add_view(hello_world, route_name='hello')
        app = config.make_wsgi_app()
    server = make_server('0.0.0.0', 6543, app)
    server.serve_forever()
```



FastAPI: High-Performance and Asynchronous

1

Performance

FastAPI prioritizes performance, offering speed and efficiency thanks to its asynchronous nature.

2

Asynchronous Programming

It leverages async/await, allowing for concurrent operations and improved responsiveness in demanding applications.

3

Modern Features

FastAPI incorporates modern features like automatic documentation and type hinting for better code clarity.

```
from fastapi import FastAPI

app = FastAPI()
@app.get("/")
async def root():
    return {"message": "Hello World"}
```

Starlette

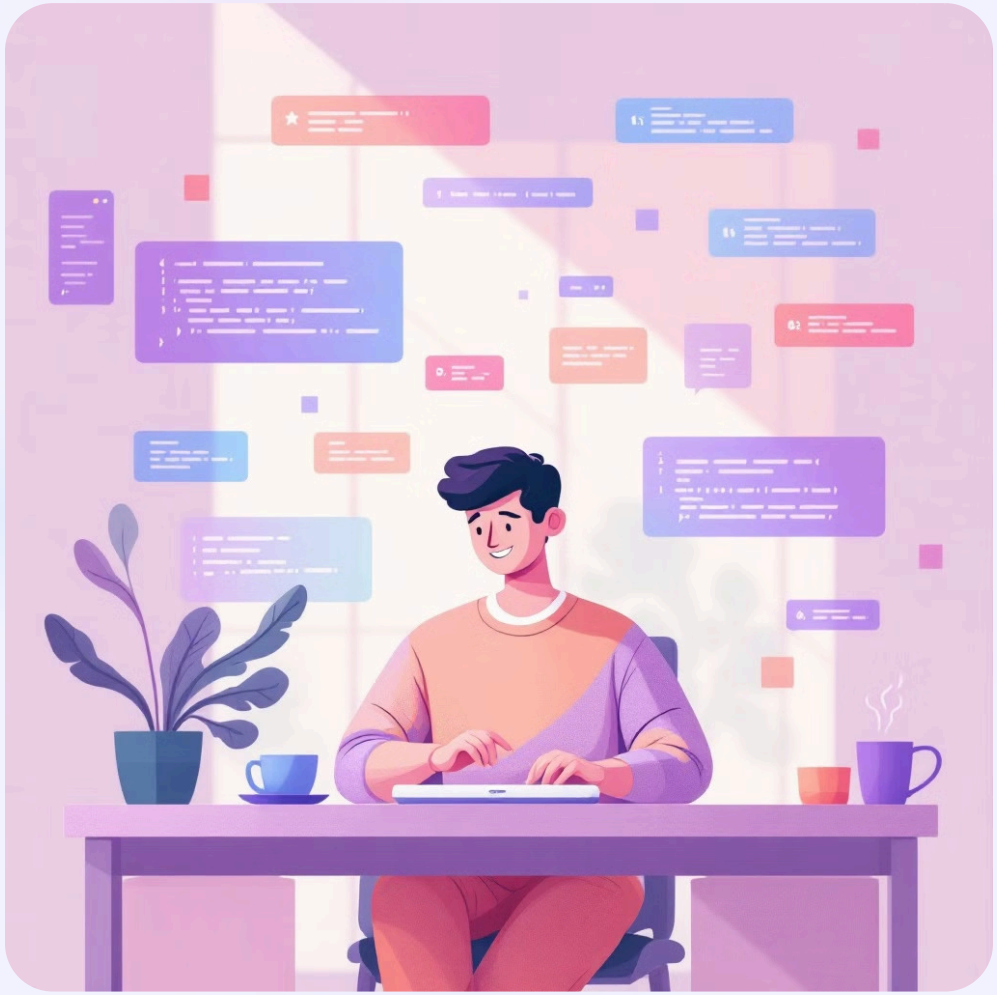
- Few deppendencies.
- A lightweight, low-complexity HTTP web framework.
- Test client built on httpx.

```
from starlette.applications import Starlette
from starlette.responses import JSONResponse
from starlette.routing import Route

async def homepage(request):
    return JSONResponse({'hello': 'world'})

app = Starlette(debug=True, routes=[
    Route('/', homepage),
])
```

CherryPy

- Built-in and flexible tools:
  caching, sessions, testing, profiling, coverage,...
- Server-side functions such as pooling and background proccessing.
- Object-oriented support.
- Publish/subscribe pattern.

```
import cherrypy

class StringGenerator(object):
    @cherrypy.expose
    def index(self):
        return "Hello world!"
    @cherrypy.expose
    def hello(self):
        return "hello"

if __name__ == '__main__':
    cherrypy.quickstart(StringGenerator())
```



Robyn: Rust runtime!

- Automatic OpenAPI generation
- Sync and Async Function Support
- WebSockets, Middlewares, Dependency Injection
- CLI tools
- A multithreaded Runtime
- GraphQL support with Strawberry

```
from robyn import Robyn

app = Robyn(__file__)

@app.get("/")
async def h(request):
    return "Hello, world!"

app.start(port=8080)
```

Run time!

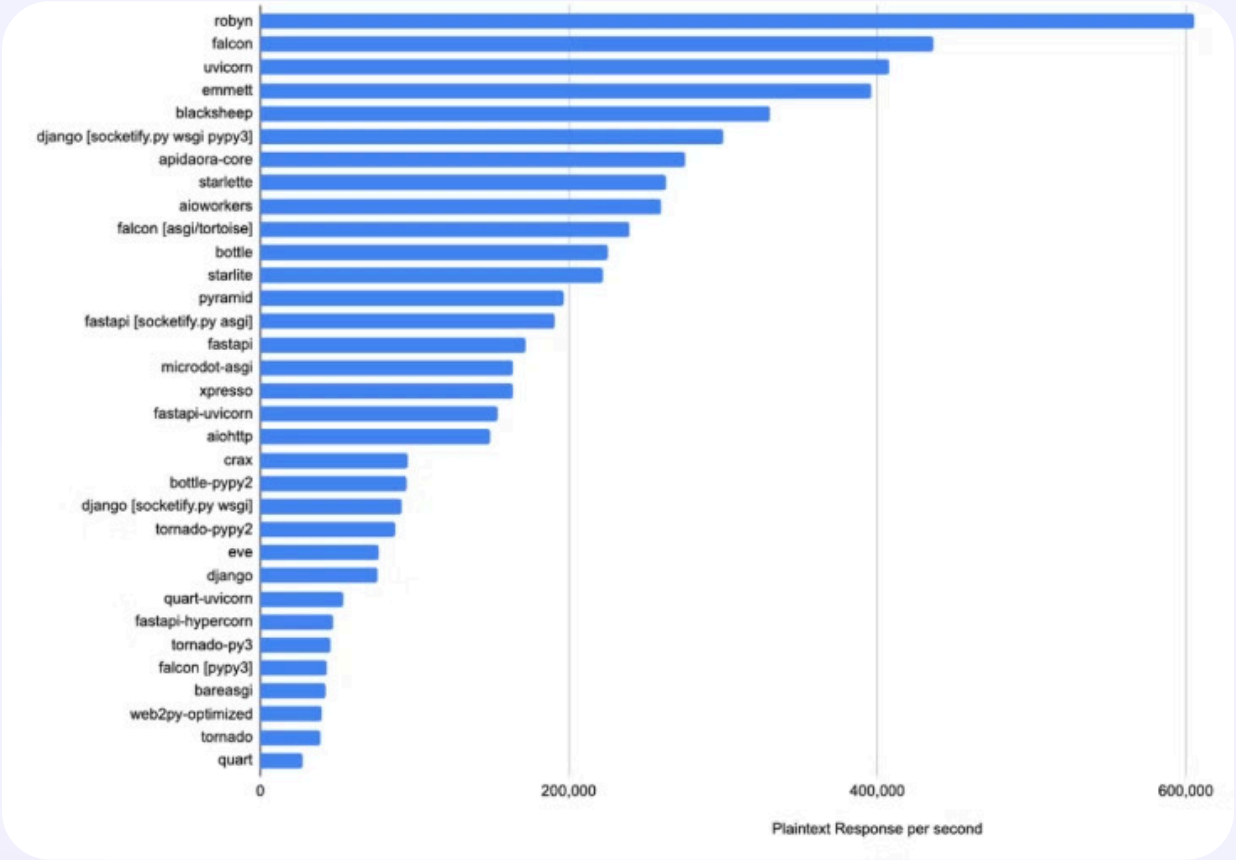
uvicorn: ASGI server

- alternatives: daphne, hypercorn
- support long-lived connections
- suitable for websockets, async apps, ...
- Often used during development and testing
- Can auto-reload application on code changes
- Single-process server by default
- Not ideal for high-traffic production

gunicorn: WSGI server

- suitable for sync apps
- Typically used in production deployments
- Not designed for auto-reloading in development
- Can spawn multiple worker processes
- Designed for production with multiple workers

Benchmarks


robyn
falcon
uvicorn
emmett
blacksheep
django [socketify.py wsgi pypy3]
apidaora-core
starlette
aioworkers
falcon [asgi/tortoise]
bottle
starlite
pyramid
fastapi [socketify.py asgi]
fastapi
microdot-asgi
xpresso
fastapi-uvicorn
aiohttp
crax
bottle-pypy2
django [socketify.py wsgi]
tornado-pypy2
eve
django
quart-uvicorn
fastapi-hypercorn
tornado-py3
falcon [pypy3]
bareasgi
web2py-optimized
tornado
quart
0
200,000
400,000
600,000
Plaintext Response per second


Made with Gamma

Community

Framework	GitHub	Stackoverflow2024
Django	81.2k	12%
FastAPI	78K	9.9%
Flask	68.2K	12.9%
Dash	21.5k	
Pyramid	4k	
Robyn	4.5k	

Locust

Load testing tool!

•

```
from locust import HttpUser, between, task

class WebsiteUser(HttpUser):
    wait_time = between(5, 15)
    def on_start(self):
        self.client.post("/login", {
            "username": "test_user",
            "password": ""
        })

    @task
    def index(self):
        self.client.get("/")
       self.client.get("/static/assets.js")

    @task
    def about(self):
        self.client.get("/about/")
```

Let's connect!