Build an Xbox Controller Abstraction Layer in Python Using XInput API

Bartosz Konikiewicz · Follow
Published in Level Up Coding
3 min read · Dec 10, 2020

Listen Share More

Or: how to achieve simple goal with exaggerated amount of code — Part 2

Photo by Kamil S on Unsplash

In the previous part of the tutorial, you've made Python to detect gamepad input. So far, so good. In this one, you will:

- Write a stub of a wrapper class.
- Add a configuration file.

Let's start, shall we?

The Wrapper

Start with creating an XInput class of your own. We need a way to tell the script which gamepad is connected and what settings to use:

```python
from configparser import ConfigParser


class XInput:
    def __init__(self, config_file, gamepad_number=0):
        self.gamepad_number = gamepad_number
        self.state = XInputState()
        self.gamepad = self.state.Gamepad
        config = ConfigParser()
        config.read(config_file)
        self.config = config['gamepad']
```

xinput.py hosted with ❤ by GitHub view raw

- Pass configuration (a path to an `ini` file; more on it in a while) file and gamepad number to initialize method. Despite the variable appearance, I won't cover handling multiple input devices, but the option's there if you need it.
- Init the state.
- Read the config ad assign the `gamepad` section to the class variable `config`.

Next, some constants need to get defined:

```python
class XInput:
    API = ctypes.windll.xinput1_4
    TRIGGERS = {
        'LEFT_TRIGGER': 'bLeftTrigger',
        'RIGHT_TRIGGER': 'bRightTrigger',
    }
    THUMBS = {
        'LEFT_THUMB_X': 'sThumbLX',
        'LEFT_THUMB_-X': 'sThumbLX',
        'LEFT_THUMB_Y': 'sThumbLY',
        'LEFT_THUMB_-Y': 'sThumbLY',
        'RIGHT_THUMB_X': 'sThumbRX',
        'RIGHT_THUMB_-X': 'sThumbRX',
        'RIGHT_THUMB_Y': 'sThumbRY',
        'RIGHT_THUMB_-Y': 'sThumbRY',
    }
    AXES = {
        **TRIGGERS,
        **THUMBS
    }
    BUTTONS = {
        'DPAD_UP': 0x0001,
        'DPAD_DOWN': 0x0002,
        'DPAD_LEFT': 0x0004,
        'DPAD_RIGHT': 0x0008,
        'START': 0x0010,
        'BACK': 0x0020,
        'LEFT_THUMB': 0x0040,
        'RIGHT_THUMB': 0x0080,
        'LEFT_SHOULDER': 0x0100,
        'RIGHT_SHOULDER': 0x0200,
        'A': 0x1000,
        'B': 0x2000,
        'X': 0x4000,
        'Y': 0x8000
    }
    TRIGGER_MAGNITUDE = 256
    THUMB_MAGNITUDE = 32768
    MOTOR_MAGNITUDE = 65535
    ERROR_SUCCESS = 0
```

xinput.py hosted with ❤ by GitHub view raw

Lots of stuff, but it's mostly just mapping:

- Import `xinput1_4` library. It's the latest version dedicated for Windows 10. If you wish to support other versions as well, you can check the docs for reference.

- Define `TRIGGERS` and `THUMBS` mapping. They directly correspond with `XInputGamepad` structure fields, just without the `XINPUT_GAMEPAD_` prefix. Using mapped values is just more convenient. Besides, you need a way to determine which set of buttons have been pressed.
- Merge `TRIGGERS` and `THUMBS` in `AXES` dictionary. Technically triggers are both buttons and axes, but because they are pressure-sensitive, it would be more logical to treat them as the axes.
- `BUTTONS` mapping. Reference.
- Set maximum `TRIGGER`, `MOTOR` and `THUMB_MAGNITUDE`. The values are suggested by XInputGamepad and XInputVibration documentation pages.
- `ERROR_SUCCESS` is an error code you get when successfully fetching the state of the controller. If selected controller is unplugged, you'd get different code.

Configuration

There are no settings at the moment, so let's create the `ini` file now. In the future, you could put buttons mappings in there. Name the file `default.ini` and set the

```
THUMB_DEAD_ZONE = 0.2
TRIGGER_DEAD_ZONE = 0.2
THUMB_SENSITIVITY = 1
TRIGGER_SENSITIVITY = 1
```

- Only one section: `[gamepad]`.
- Axes dead zone and sensitivity.

And that's it. That's everything that can (and needs to) be adjusted.

Get State

Now, we can fetch the state:

```python
def get_state(self):
    previous_state = self.state.dwPacketNumber
    error_code = self.API.XInputGetState(
        ctypes.wintypes.WORD(self.gamepad_number),
        ctypes.pointer(self.state)
    )
    if error_code != self.ERROR_SUCCESS:
        raise Exception('Gamepad number {} is not connected'.format(self.gamepad_number))
    return previous_state != self.state.dwPacketNumber
```

xinput.py hosted with ❤️ by GitHub view raw

- It looks similar to the code you've written in the previous part of the tutorial, the only real difference being that it's now a method.
- There is only a single XInput API method for both obtaining the controller state and checking if the gamepad's connected, meaning that you have to implement both of these features in a single function.

That's everything for this part of the tutorial. In the last one, you will interact with the gamepad more heavily. It's going to be meaty, I promise.

Final words (for now)

Thanks for reading!

You can find the full code on my GitHub.

Python Xinput Xbox Gamepad Mapper

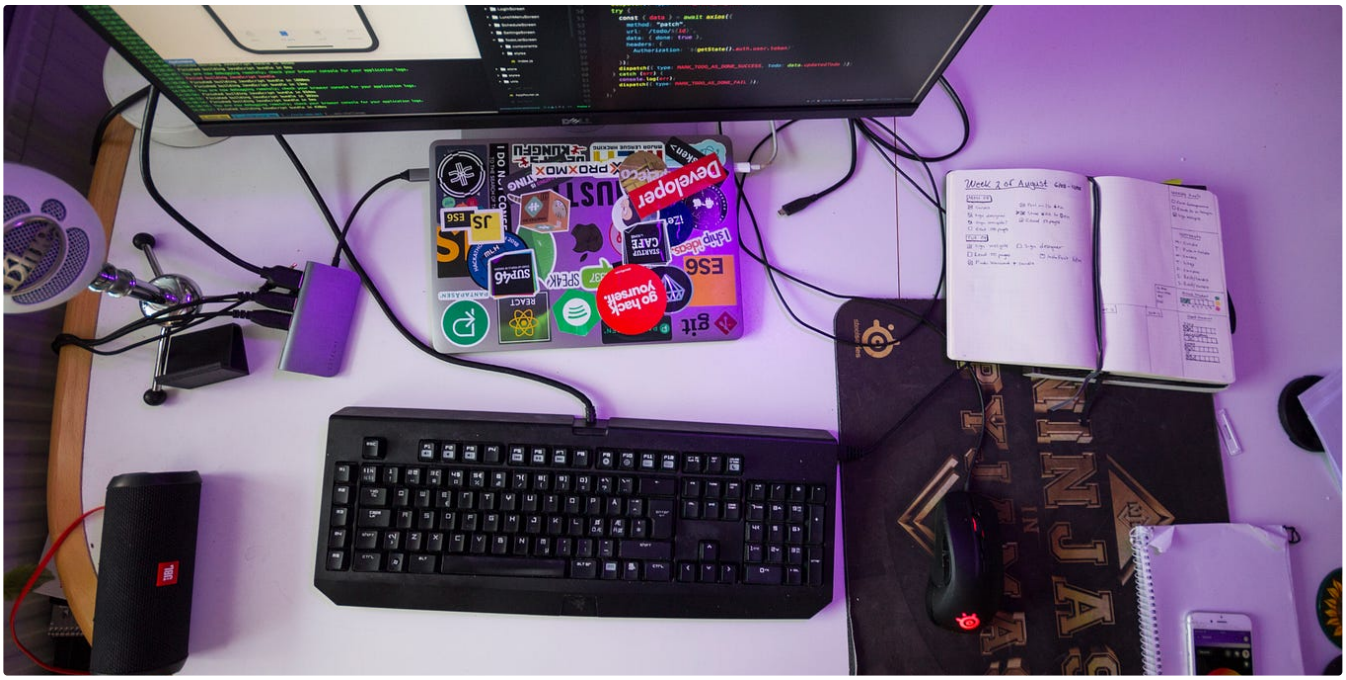
Follow

Written by Bartosz Konikiewicz

24 Followers · Writer for Level Up Coding

This is not the greatest bio in the world. This is just a tribute.

More from Bartosz Konikiewicz and Level Up Coding



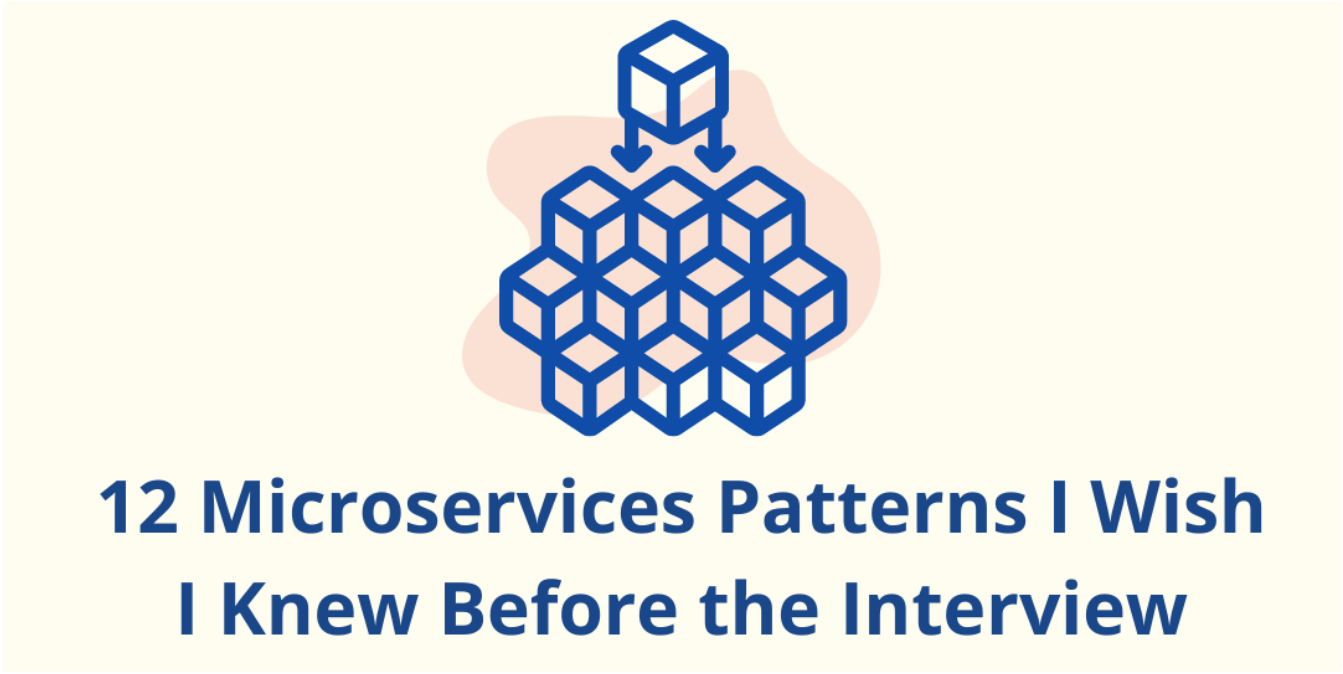
Bartosz Konikiewicz in Level Up Coding

Create a Controlled Radio Group in React, Formik, Material UI, and TypeScript

Or: how not to lose five hours at your 9 to 5

4 min read · Apr 25, 2021

113 1



Arslan Ahmad in Level Up Coding

12 Microservices Patterns I Wish I Knew Before the System Design Interview

Mastering the Art of Scalable and Resilient Systems with Essential Microservices Design Patterns

· 13 min read · May 16

1K 9

Attila Vágó in Level Up Coding

Martin Fowler Was Right: Microservices Suck*

A pragmatic view on Amazon's unexpected stance on serverless microservice architecture.

✦ · 5 min read · May 9

2K 36

Bartosz Konikiewicz in Level Up Coding

Handle Registration in FastAPI and Tortoise ORM

And shed a tear or two in the process—Part 3

4 min read · Dec 9, 2020

11 1

See all from Bartosz Konikiewicz

See all from Level Up Coding

Recommended from Medium

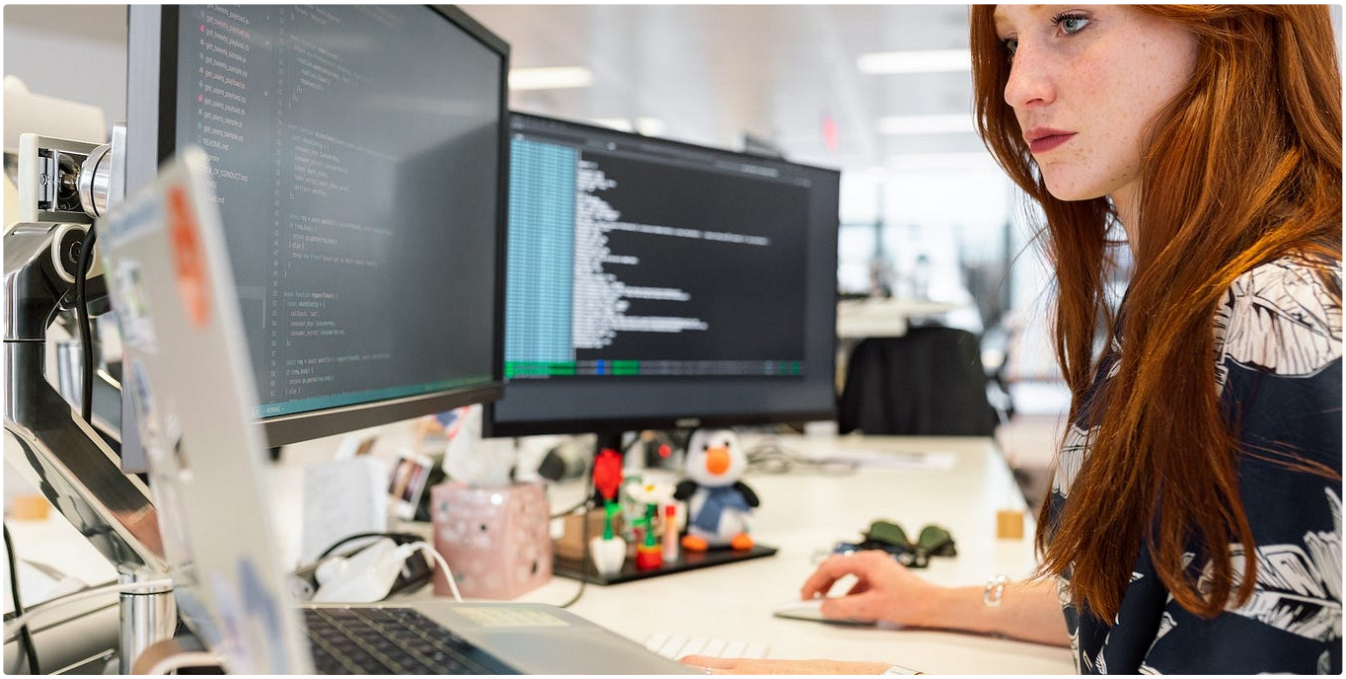
Aysel Aydin

An Introduction to Python FastAPI & Swagger UI

In this Python tutorial, you will learn about FastAPI that a Web framework for developing RESTful APIs in Python.

3 min read · Jan 7

373



The Coding Diaries in The Coding Diaries

Why Experienced Programmers Fail Coding Interviews

A friend of mine recently joined a FAANG company as an engineering manager, and found themselves in the position of recruiting for...

· 5 min read · Nov 2, 2022

2.9K 64

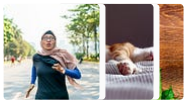
Lists

Staff Picks

320 stories · 81 saves

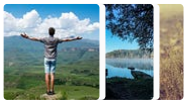
Stories to Help You Level-Up at Work

19 stories · 40 saves



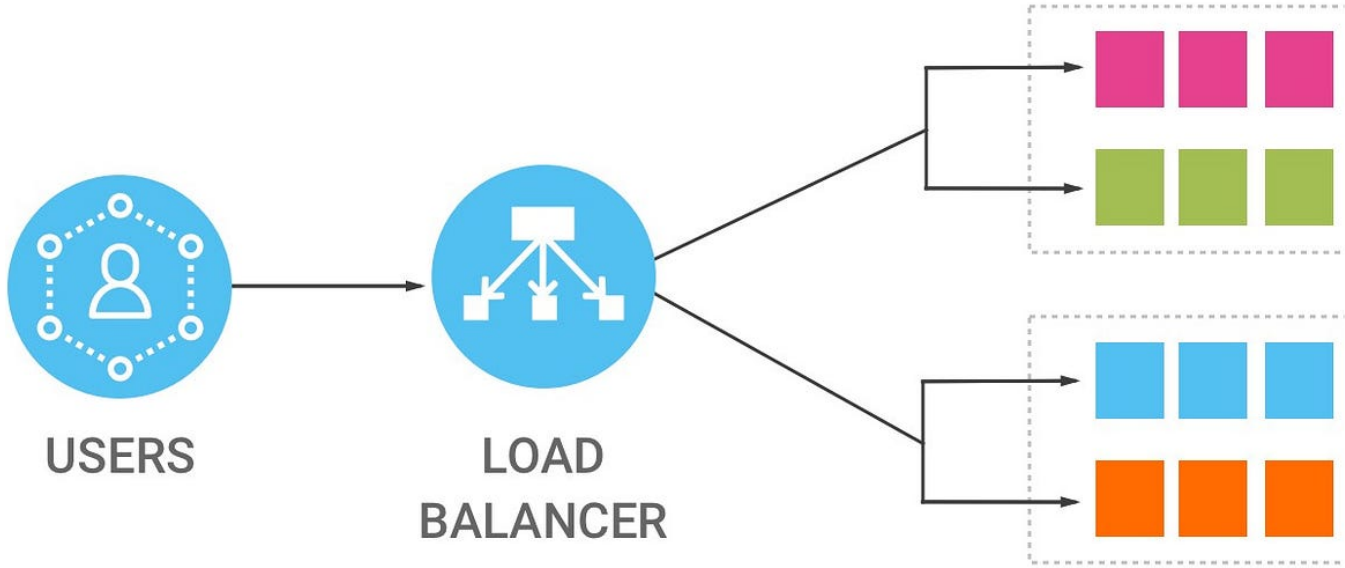
Self-Improvement 101

20 stories · 87 saves



Productivity 101

20 stories · 88 saves



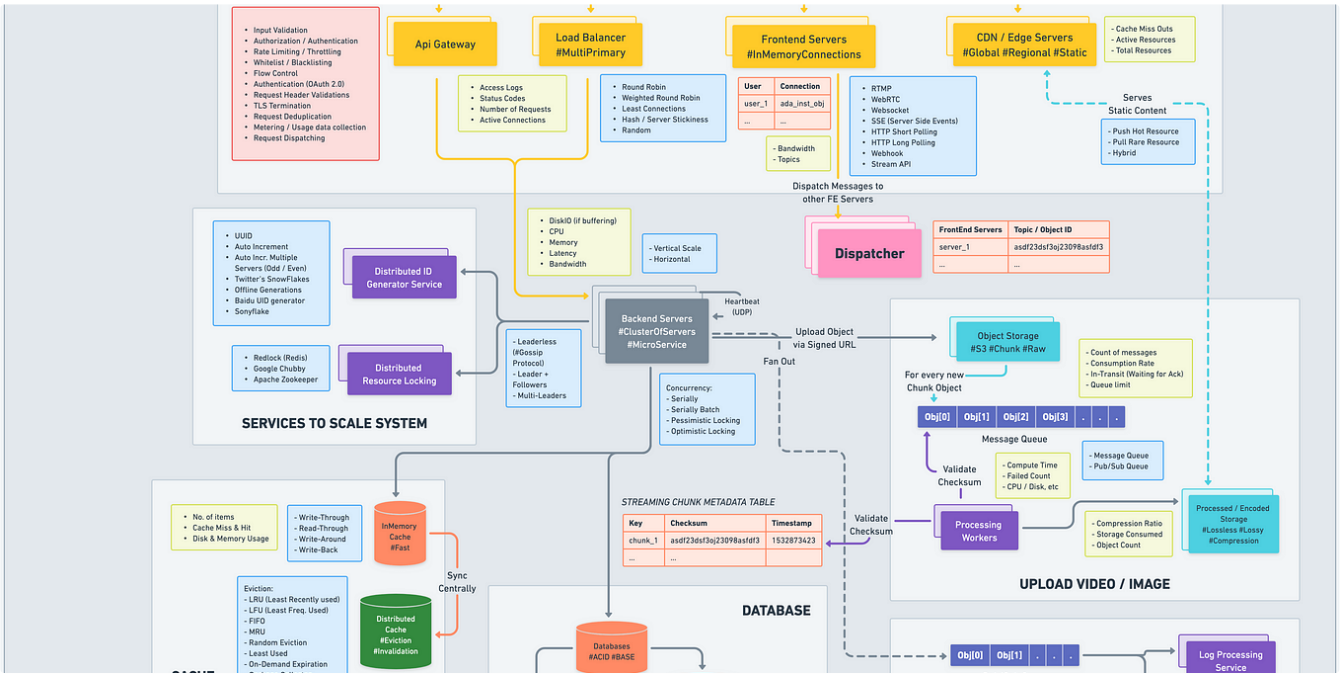
Daniel Pollak in anecdotes engineering

FastAPI Scale with non-asyncio requests handling

FastAPI is a web server framework that is particularly well-suited for use with asyncio-compatible flows. However, there are still many...

6 min read · Dec 25, 2022

186 3



Love Sharma in Dev Genius

System Design Blueprint: The Ultimate Guide

Developing a robust, scalable, and efficient system can be daunting. However, understanding the key concepts and components can make the...

✨ · 9 min read · Apr 20

1.1K 11

Jacob Marks, Ph.D. in Towards Data Science

How I Turned My Company's Docs into a Searchable Database with OpenAI

And how you can do the same with your docs

15 min read · Apr 25

2.9K 39


Hussein Nasser

How to Become a Good Backend Engineer (Fundamentals)

I have been a backend engineer for over 18 years and I witnessed technologies come and go but one thing always remain constant; The first...

· 11 min read · Dec 3, 2022

5.3K 40