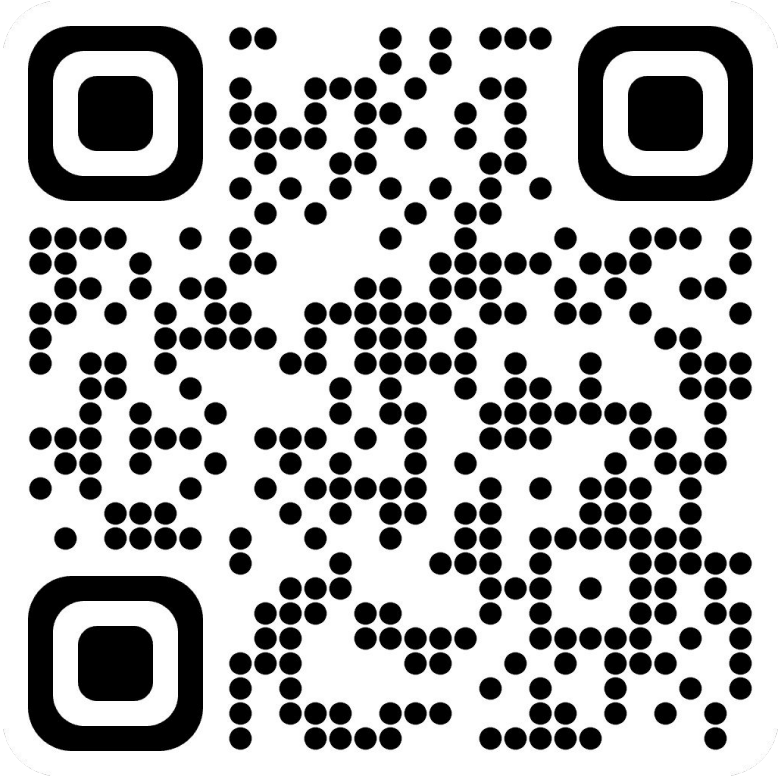
# Mónadas en Python

## Menos Excepciones y Más Tipado en Nuestro Código


Artur Costa-Pazo

Alice



https://github.com/alice-biometrics/pycones23

Discord

# Mónada

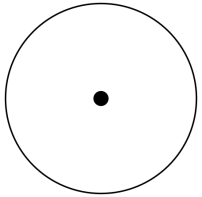
# Mónada

“Eso es un concepto filosófico, ¿no?”

# Mónada

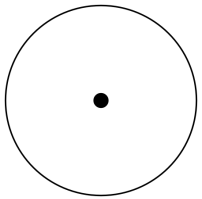
“Eso es un concepto filosófico, ¿no?”

# Mónada



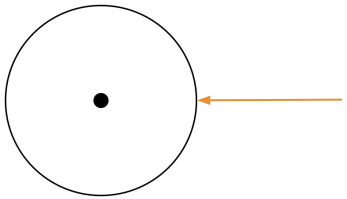
del griego *μονάς* monas, "**unidad**"
de *μόνος* monos, "**uno**", "**solo**", "**único**"

# Mónada



Término usado en filosofía que significa “**unidad**”, utilizado de diferentes maneras por los filósofos.

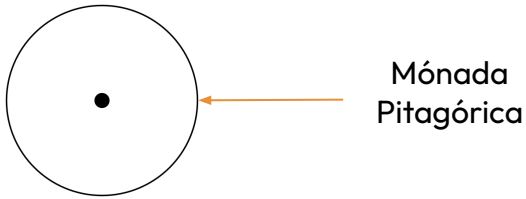
# Mónada



Mónada
Pitagórica

Término usado en filosofía que significa “**unidad**”, utilizado de diferentes maneras por los filósofos.

# Mónada




**Cada una de las sustancias indivisibles, pero de naturaleza distinta**, que componen el universo, según el sistema de Leibniz, filósofo y matemático alemán del siglo XVII.

# ¿Quien soy?

**Artur Costa-Pazo** 🗿

*Research Software Engineer*

# ¿Quien soy?

**Artur Costa-Pazo** 🗿

*Research Software Engineer*

*PhD on Face Presentation Attack Detection*

*Head of Engineering at* Alice

# ¿Quien soy?

**Artur Costa-Pazo** 🗿

*Research Software Engineer*

*PhD on Face Presentation Attack Detection*

*Head of Engineering at* Alice

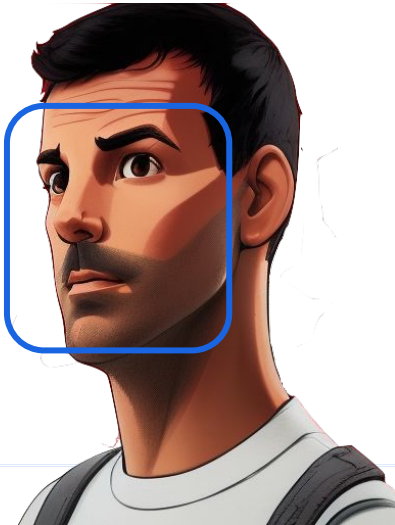

# ¿Quien soy?

**Artur Costa-Pazo** 🗿

*Research Software Engineer*

*PhD on Face Presentation Attack Detection*

*Head of Engineering at* Alice

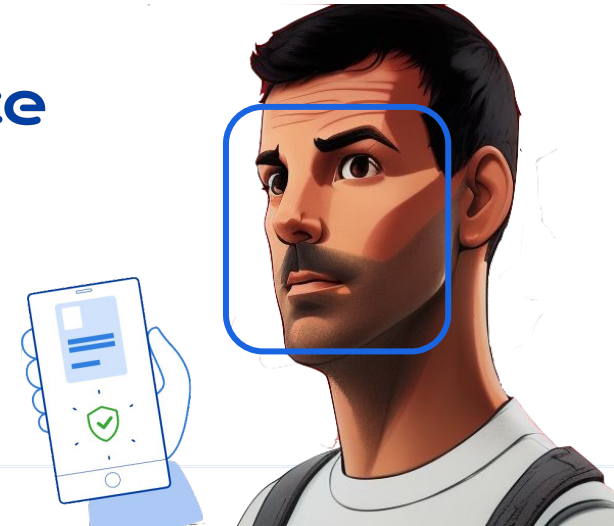
# ¿Quien soy?

**Artur Costa-Pazo** 🗿

*Research Software Engineer*

*PhD on Face Presentation Attack Detection*

*Head of Engineering at* Alice

# ¿Quien soy?

**Artur Costa-Pazo** 🗿

*Research Software Engineer*

*PhD on Face Presentation Attack Detection*

*Head of Engineering at* Alice


# ¿Por qué vienes a hablar de mónadas?

En Alice llevamos 4 años utilizando mónadas en todos nuestro servicios gracias a la ayuda del paquete Python **meiga** 🧙.

# ¿Por qué vienes a hablar de mónadas?

En Alice llevamos 4 años utilizando mónadas en todos nuestro servicios gracias a la ayuda del paquete Python **meiga** 🧙.

```
pip install meiga
```

# ¿Por qué vienes a hablar de mónadas?

En Alice llevamos 4 años utilizando mónadas en todos nuestro servicios gracias a la ayuda del paquete Python **meiga** 🧙.

```
pip install meiga
```

https://github.com/alice-biometrics/meiga

# ¿Por qué vienes a hablar de mónadas?

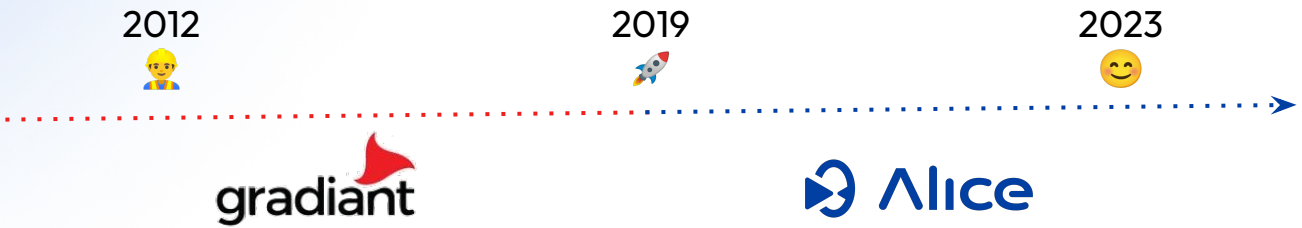
En Alice llevamos 4 años utilizando mónadas en todos nuestro servicios gracias a la ayuda del paquete Python **meiga** 🧙.

```
pip install meiga ⭐
```

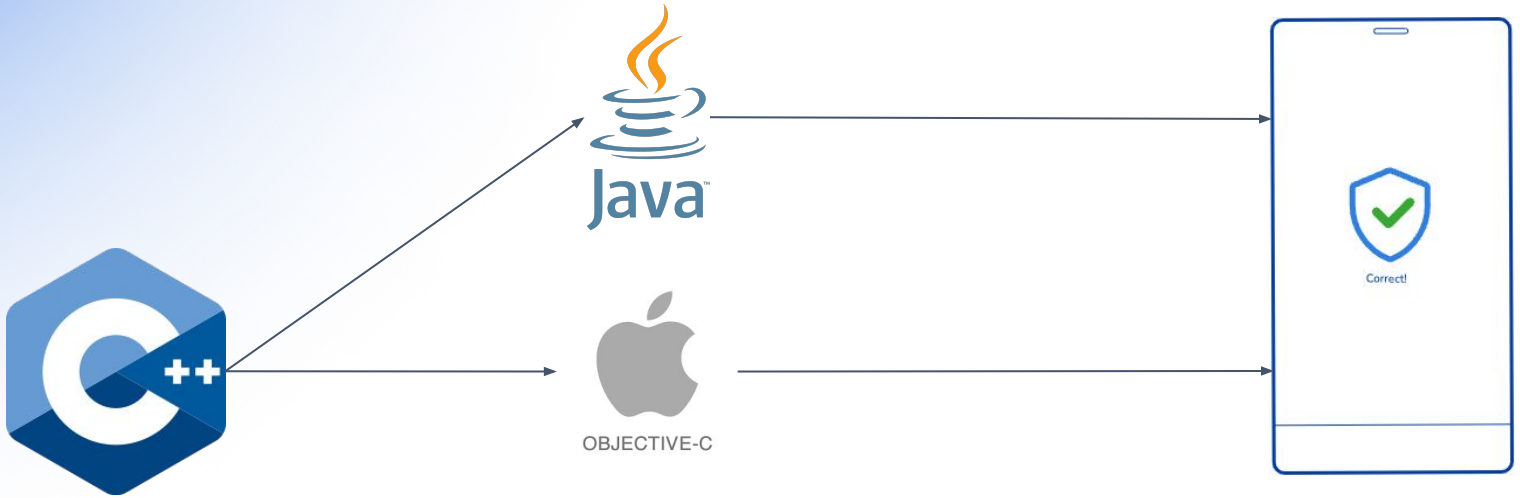
https://github.com/alice-biometrics/meiga

# ¿Y cómo empezasteis a utilizarlas?

# ¿Y cómo empezasteis a utilizarlas?



# ¿Y cómo empezasteis a utilizarlas?




Modelos de Visión Artificial y proto AI.

# ¿Y cómo empezasteis a utilizarlas?

Modelos de Visión Artificial y proto AI.

```
#include <jni.h>
#include <iostream>
#include "TestJNIPrimitive.h"
using namespace std;

JNIEXPORT jdouble JNICALL Java_TestJNIPrimitive_average
          (JNIEnv *env, jobject obj, jint n1, jint n2) {
   jdouble result;
   cout << "In C++, the numbers are " << n1 << " and " << n2 << endl;
   result = ((jdouble)n1 + n2) / 2.0;
   // jint is mapped to int, jdouble is mapped to double
```

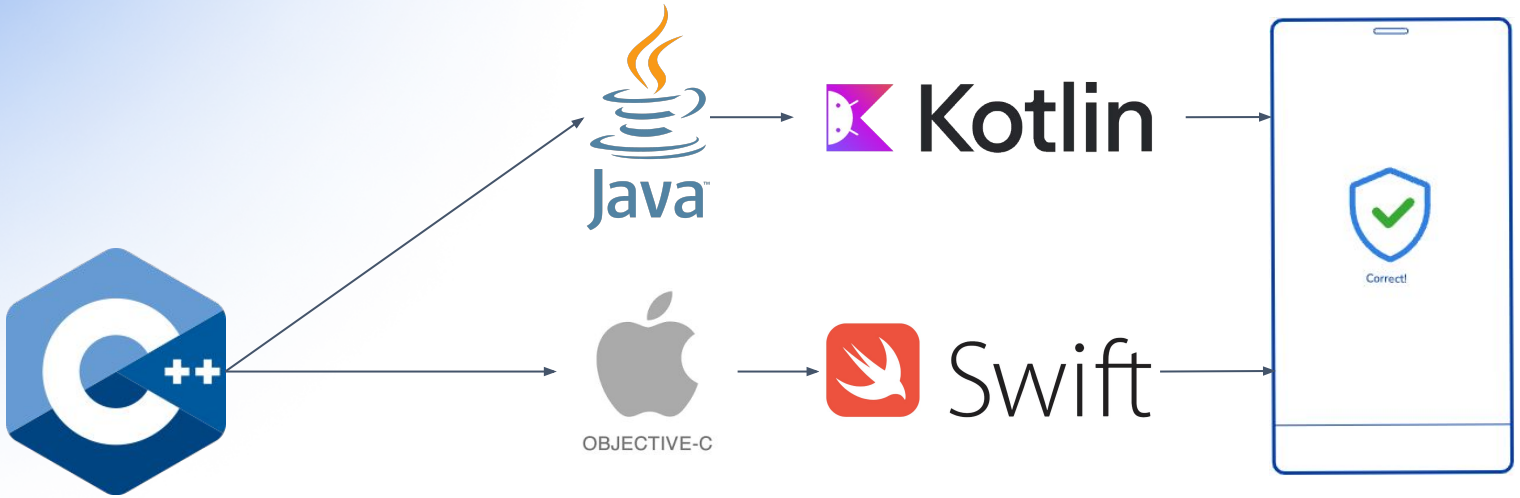
```
@implementation MyObjective_c_file
…
 – (void)myMethod: (NSString *) dataPayload {

// Your logic here
}

@end

void callFromCpp(const char *dataPayload) {
 NSString *convertedString = [[NSString alloc] initWithCString:
dataPayload encoding:NSUTF8StringEncoding];

 MyObjective_c_file myInstance = [[MyObjective_c_file alloc] init];
 [MyObjective_c_file myMethod: convertedString];
}
```

# ¿Y cómo empezasteis a utilizarlas?

```
#include <jni.h>
#include <iostream>
#include "TestJNIPrimitive.h"
using namespace std;

JNIEXPORT jdouble JNICALL Java_TestJNIPrimitive_average
          (JNIEnv *env, jobject obj, jint n1, jint n2) {
   jdouble result;
   cout << "In C++, the numbers are " << n1 << " and " << n2 << endl;
   result = ((jdouble)n1 + n2) / 2.0;
   // jint is mapped to int, jdouble is mapped to double
```

```
@implementation MyObjective_c_file
…
 – (void)myMethod: (NSString *) dataPayload {

// Your logic here
}

@end

void callFromCpp(const char *dataPayload) {
 NSString *convertedString = [[NSString alloc] initWithCString:
dataPayload encoding:NSUTF8StringEncoding];

 MyObjective_c_file myInstance = [[MyObjective_c_file alloc] init];
 [MyObjective_c_file myMethod: convertedString];
}
```

# ¿Y cómo empezasteis a utilizarlas?




Modelos de Visión Artificial y proto AI.

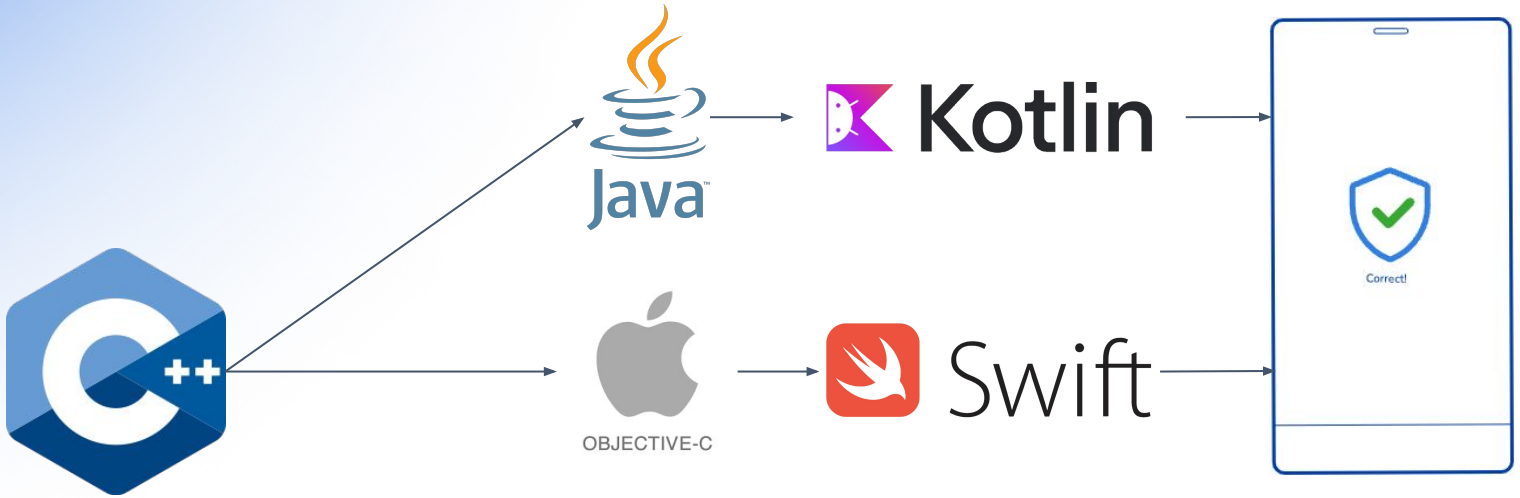
# ¿Y cómo empezasteis a utilizarlas?


```swift
typealias JSONObject = [String: Any]

enum JSONError: Error {
    case noSuchKey(String)
    case typeMismatch
}

func stringForKey(json: JSONObject, key: String) -> Result<String, JSONError> {
    guard let value = json[key] else {
        return .failure(.noSuchKey(key))
    }

    guard let value = value as? String else {
        return .failure(.typeMismatch)
    }

    return .success(value)
}
```

# ¿Y cómo empezasteis a utilizarlas?

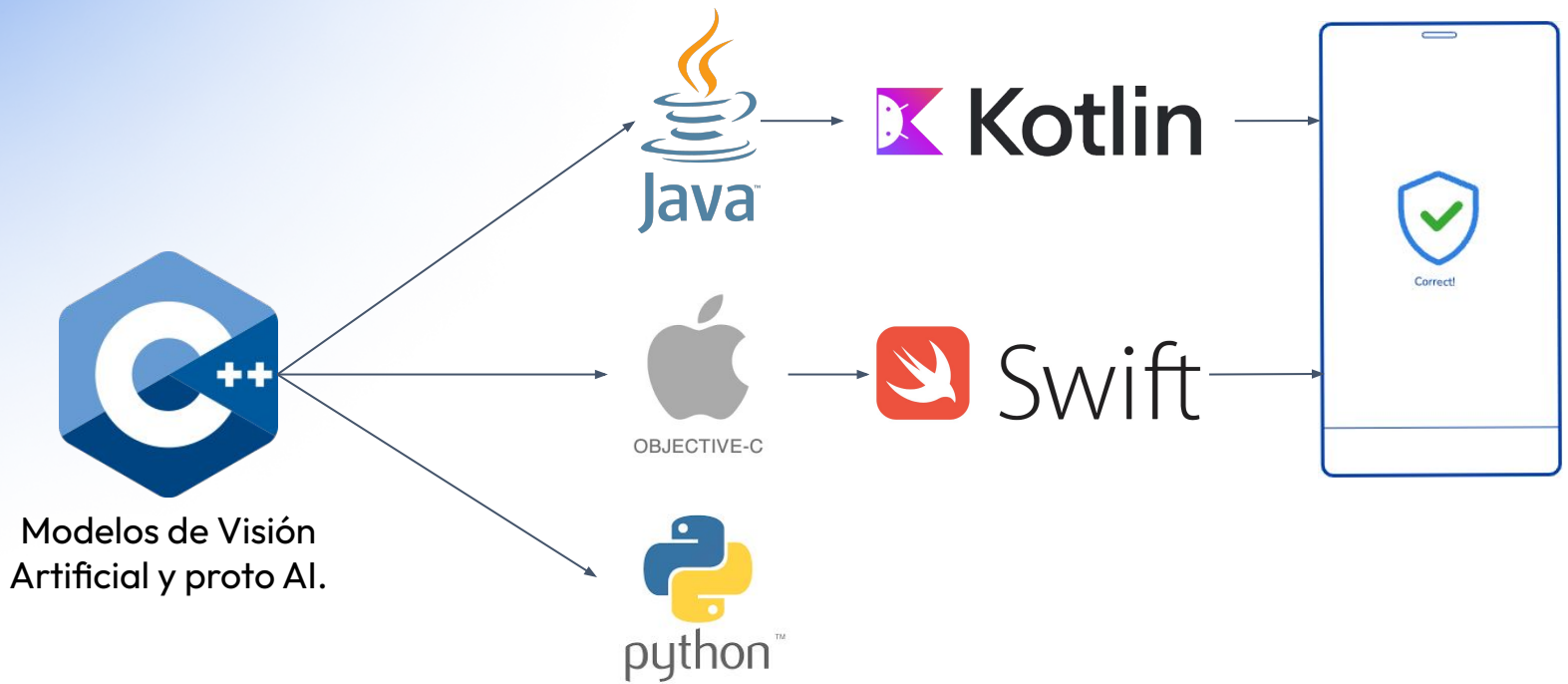

```swift
typealias JSONObject = [String: Any]

enum JSONError: Error {
    case noSuchKey(String)
    case typeMismatch
}

func stringForKey(json: JSONObject, key: String) -> Result<String, JSONError> {
    guard let value = json[key] else {
        return .failure(.noSuchKey(key))
    }

    guard let value = value as? String else {
        return .failure(.typeMismatch)
    }

    return .success(value)
}
```

# ¿Y cómo empezasteis a utilizarlas?




Modelos de Visión Artificial y proto AI.

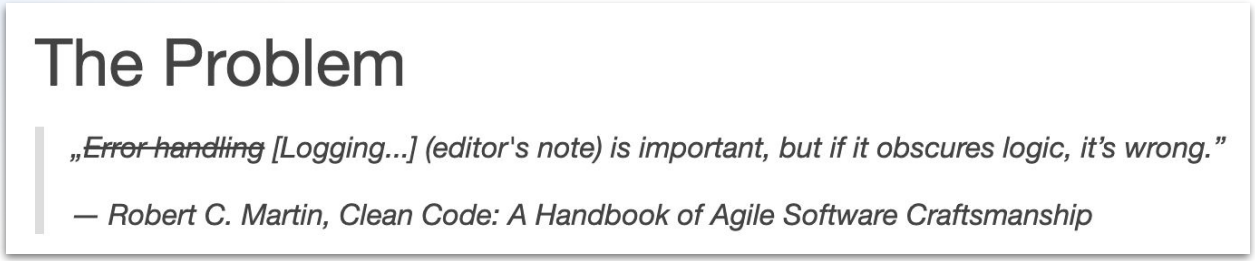
# ¿Y cómo empezasteis a utilizarlas?




Modelos de Visión Artificial y proto AI.

# ¿Y cómo empezasteis a utilizarlas?

Blog Post:
[Remove visual noise of logging code with python decorators]



# ¿Y cómo empezasteis a utilizarlas?

```
import requests
import logging

logger = logging.getLogger(__name__)


class DownloadFailedError(Exception):
    pass


def download(url):
    logger.debug("Starting to download {:s}".format(url))
    response = requests.get(url)

    if response.status_code != 200:
        logger.error("Download was not successful due to response status {:d}"
                     .format(response.status))
        raise DownloadFailedError

    filename = url.split("/")[-1]
    filename = filename or "index.html"
    logger.debug("Generated filename '{:s}' from url '{:s}'".format(filename, url))

    logger.debug("Writing downloaded content into file")
    with open(filename, "wb") as f:
        f.write(response.content)
    logger.info("Downloading of '{:s}' into file '{:s}' was successful"
                .format(url, filename))
```

# ¿Y cómo empezasteis a utilizarlas?

```
import requests
import logging

logger = logging.getLogger(__name__)


class DownloadFailedError(Exception):
    pass


def download(url):
    logger.debug("Starting to download {:s}".format(url))
    response = requests.get(url)

    if response.status_code != 200:
        logger.error("Download was not successful due to response status {:d}"
                     .format(response.status))
        raise DownloadFailedError

    filename = url.split("/")[-1]
    filename = filename or "index.html"
    logger.debug("Generated filename '{:s}' from url '{:s}'".format(filename, url))

    logger.debug("Writing downloaded content into file")
    with open(filename, "wb") as f:
        f.write(response.content)
    logger.info("Downloading of '{:s}' into file '{:s}' was successful"
                .format(url, filename))
```

# ¿Y cómo empezasteis a utilizarlas?

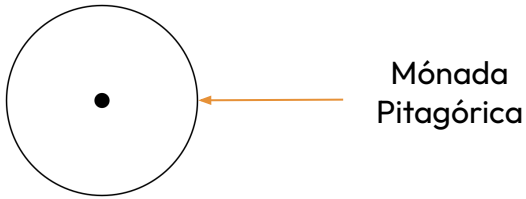
```
@log_on_start(DEBUG, "Starting to download {url:s}")
@log_on_end(INFO, "Downloading of '{url:s}' into file '{result:s}' was successful")
def download(url):
    content = content_from_url(url)
    filename = url_to_filename(url)

    write_to_file(filename, content)
    return filename
```

# ¿Y cómo empezasteis a utilizarlas?


➕ *Remove visual noise...* = 🌱 (meiga 🧙‍♀️)

# Mónada

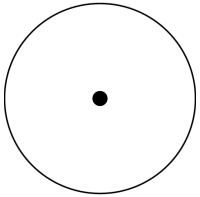
# Mónada




**Cada una de las sustancias indivisibles, pero de naturaleza distinta**, que componen el universo, según el sistema de Leibniz, filósofo y matemático alemán del siglo XVII.

# Mónada

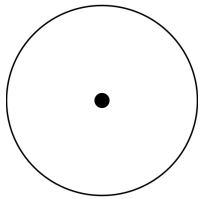
Success



Failure

{ value }

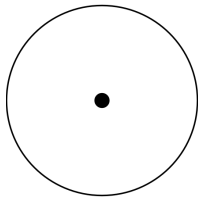
# Mónada

Success  Failure

{ value }

```
def foo() → Result[Success,Failure]:
```

# Mónada

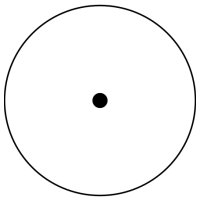
Success  Failure

{ value }

```
def foo() → Result[str,Failure]:
```

# Mónada

Success



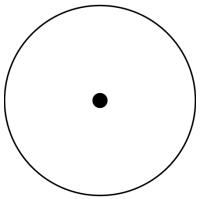
Failure

{ value }

```
def foo() → Result[int,Failure]:
```

# Mónada

Success

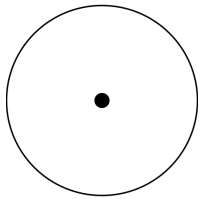


Failure

{ value }

```
def foo() → Result[str|int,Failure]:
```

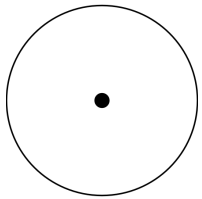
# Mónada

Success  Failure

{ value }

```
def foo() → Result[list,Failure]:
```

# Mónada

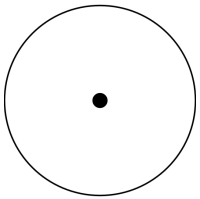
Success  Failure

{ value }

```
def foo() → Result[MyModel,Failure]:
```

# Mónada

Success



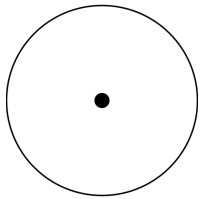
Failure

{ value }

```
def foo() → Result[MyModel,NotFound]:
```

# Mónada

Success



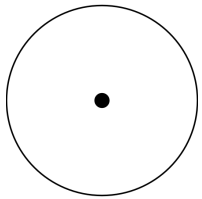
Failure

{ value }

```
def foo() → Result[MyModel,NotAllowed]:
```

# Mónada

Success

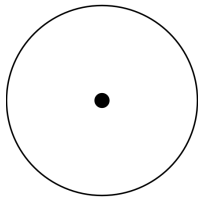


Failure

{ value }

```
def foo() → Result[MyModel,NotFound|NotAllowed]:
```

# Mónada

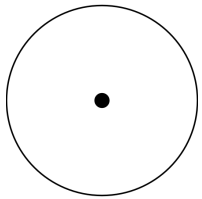
Success  Failure

{ value }

```
def foo() → Result[MyModel,MyError]:
```

# Mónada

Success



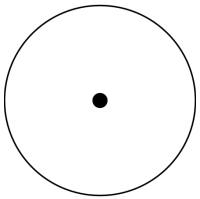
Failure

{ value }

```
def foo() → Result[Success,Failure]:
```

# Mónada

Ok

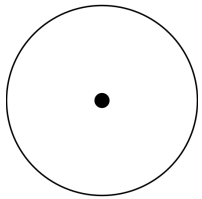


Err

{ value }

```
def foo() → Result[Ok,Err]:
```

# Mónada

Left  Right

{ value }

```
def foo() → Either[Left,Right]:
```

Haskell

Either


SHOW ME THE CODE

# Ejemplo

```
dictionary: dict[str, Any] = {"key1": "1", "key2": 2}

value: str = string_from_key(dictionary, "key1")
```

Queremos una función que nos devuelva valores de un dict:

# Ejemplo

```
dictionary: dict[str, Any] = {"key1": "1", "key2": 2}

value: str = string_from_key(dictionary, "key1")
```

Queremos una función que nos devuelva valores de un dict.

- Solo queremos que nos devuelva `str`.

# Ejemplo

```python
dictionary: dict[str, Any] = {"key1": "1", "key2": 2}

value: str = string_from_key(dictionary, "invalid_key")
```

Queremos una función que nos devuelva valores de un dict.

- Solo queremos que nos devuelva `str`.
- Si la *key* que le pasamos no existe queremos que nos devuelva un error.

# Ejemplo

```
dictionary: dict[str, Any] = {"key1": "1", "key2": 2}

value: str = string_from_key(dictionary, "key2")
```

Queremos una función que nos devuelva valores de un dict.

- Solo queremos que nos devuelva `str`.
- Si la *key* que le pasamos no existe queremos que nos devuelva un error.
- Si el valor recuperado no es un `str` queremos que nos devuelva un error.

Con Excepciones

```python
dictionary: dict[str, Any] = {"key1": "1", "key2": 2}

value: str = string_from_key(dictionary, "key1")
```

```python
class NoSuchKey(Exception): ...
class TypeMismatch(Exception): ...

def string_from_key(dictionary: dict, key: str) -> str:
    if key not in dictionary.keys():
        raise NoSuchKey()

    value = dictionary.get(key)
    if not isinstance(value, str):
        raise TypeMismatch()

    return value
```

```python
dictionary: dict[str, Any] = {"key1": "1", "key2": 2}

value: str = string_from_key(dictionary, "key1")
```

Con Excepciones

```python
class NoSuchKey(Exception): ...
class TypeMismatch(Exception): ...

def string_from_key(dictionary: dict, key: str) -> str:
    if key not in dictionary.keys():
        raise NoSuchKey()

    value = dictionary.get(key)
    if not isinstance(value, str):
        raise TypeMismatch()

    return value
```

Return type hint

Exceptions

```python
dictionary: dict[str, Any] = {"key1": "1", "key2": 2}

value: str = string_from_key(dictionary, "key1")
```

Con Mónadas

```python
class NoSuchKey(Error): ...
class TypeMismatch(Error): ...

def string_from_key(dictionary: dict, key: str) -> Result[str, Error]:
    if key not in dictionary.keys():
        return Failure(NoSuchKey())

    value = dictionary.get(key)
    if not isinstance(value, str):
        return Failure(TypeMismatch())

    return Success(value)
```

```python
dictionary: dict[str, Any] = {"key1": "1", "key2": 2}

value: str = string_from_key(dictionary, "key1")
```

Con Mónadas

```python
class NoSuchKey(Error): ...
class TypeMismatch(Error): ...

def string_from_key(dictionary: dict, key: str) -> Result[str, Error]:
    if key not in dictionary.keys():
        return Failure(NoSuchKey())

    value = dictionary.get(key)
    if not isinstance(value, str):
        return Failure(TypeMismatch())

    return Success(value)
```

Monad

Failure

Success

Con Excepciones

```python
class NoSuchKey(Exception): ...
class TypeMismatch(Exception): ...

def string_from_key(dictionary: dict, key: str) -> str:
    if key not in dictionary.keys()
        raise NoSuchKey()

    value = dictionary.get(key)
    if not isinstance(value, str):
        raise TypeMismatch()

    return value
```

Return type hint

Exceptions

Con Mónadas

```python
class NoSuchKey(Error): ...
class TypeMismatch(Error): ...

def string_from_key(dictionary: dict, key: str) -> Result[str, Error]:
    if key not in dictionary.keys():
        return Failure(NoSuchKey())

    value = dictionary.get(key)
    if not isinstance(value, str):
        return Failure(TypeMismatch())

    return Success(value)
```

Monad

Failure

Success

Con Excepciones

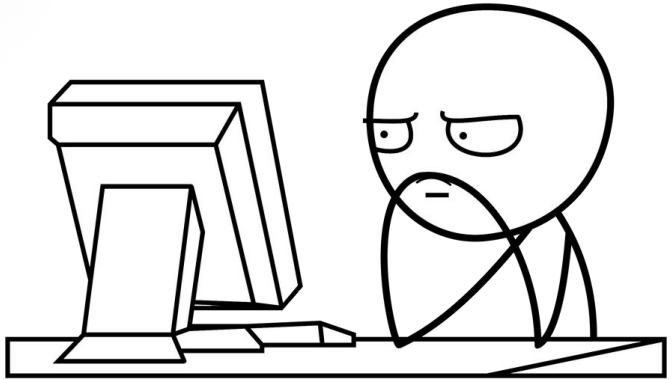
```python
class NoSuchKey(Exception): ...
class TypeMismatch(Exception): ...

@to_result
def string_from_key(dictionary: dict, key: str) -> str:
    if key not in dictionary.keys()
        raise NoSuchKey()

    value = dictionary.get(key)
    if not isinstance(value, str):
        raise TypeMismatch()

    return value
```

Con Mónadas

```python
class NoSuchKey(Error): ...
class TypeMismatch(Error): ...

def string_from_key(dictionary: dict, key: str) -> Result[str, Error]:
    if key not in dictionary.keys():
        return Failure(NoSuchKey())

    value = dictionary.get(key)
    if not isinstance(value, str):
        return Failure(TypeMismatch())

    return Success(value)
```

Monad

# ¿Por qué utilizar mónadas?



# ¿Por qué utilizar mónadas?

# ¿Por qué utilizar mónadas?

Nos ayuda a tener un mejor tipado.

Nos facilita el testing.

Nos habilita a usar funciones derivadas de la programación funcional.

Nos ayuda a crear "código limpio".

# ¿Por qué utilizar mónadas?

➡️ Nos ayuda a tener un mejor tipado.

Nos facilita el testing.

Nos habilita a usar funciones derivadas de la programación funcional.

Nos ayuda a crear código limpio.

# Mejor Tipado

```
value = string_from_key(dictionary, “key1”)
```

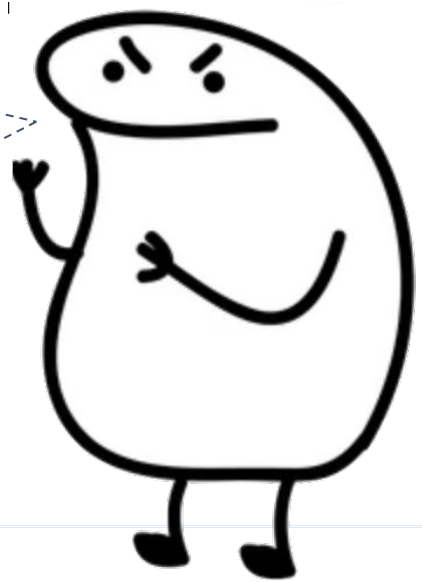
```
value: str = string_from_key(dictionary, “key1”)
```

# Mejor Tipado

```
value = string_from_key(dictionary, “key1”)
```

```
value: str = string_from_key(dictionary, “key1”)
```

Y si falla…..
¿Cuáles son las excepciones que tengo que capturar?
🤔

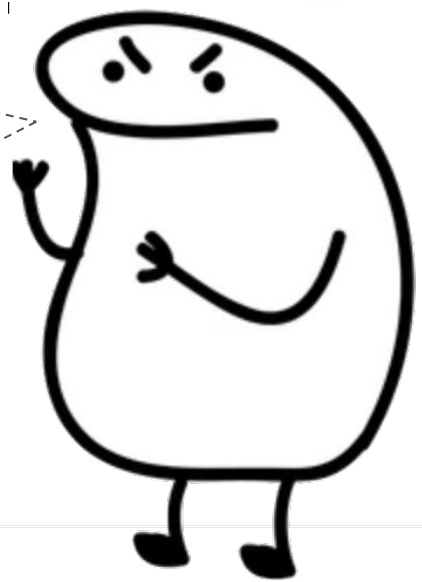


# Mejor Tipado

```
value = string_from_key(dictionary, “key1”)
```

```
value: str = string_from_key(dictionary, “key1”)
```

Y si falla…..
¿Cuáles son las excepciones que tengo que capturar?
🤔

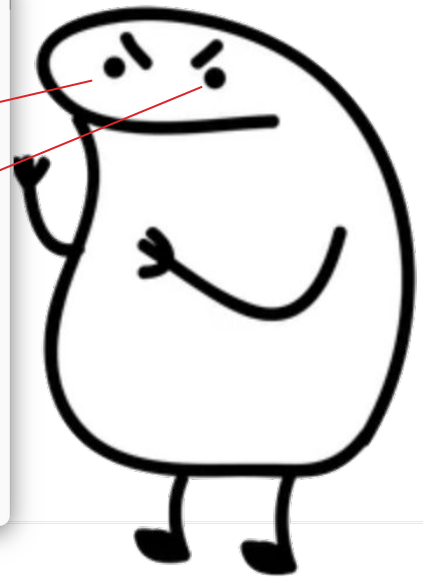


# Mejor Tipado

```
value = string_from_key(dictionary, “key1”)
```

```
value: str = string_from_key(dictionary, “key1”)
```

```
class NoSuchKey(Exception): ...
class TypeMismatch(Exception): ...

def string_from_key(dictionary: dict, key: str) → str:
    if key not in dictionary.keys()
        raise NoSuchKey()

    value = dictionary.get(key)
    if not isinstance(value, str):
        raise TypeMismatch()

    return value
```

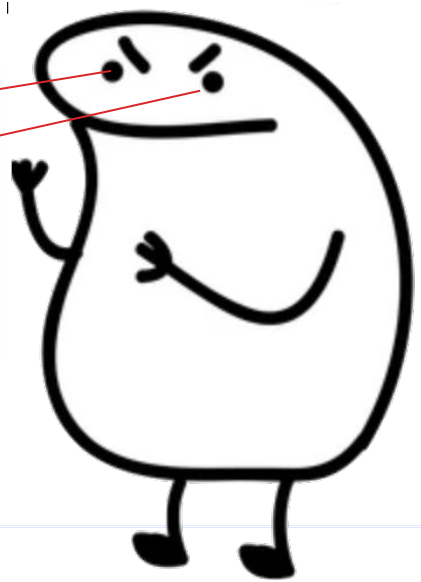


# Mejor Tipado

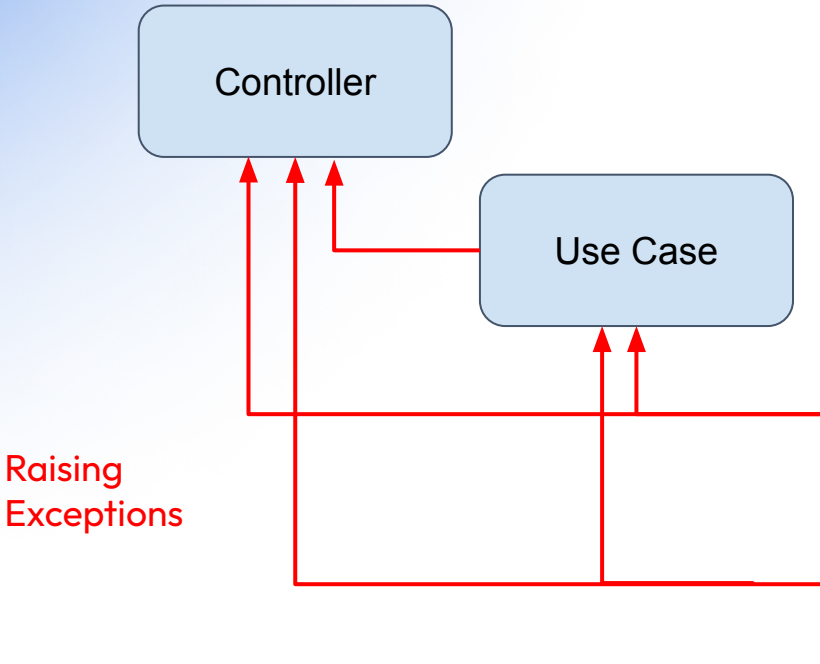
```
value = string_from_key(dictionary, “key1”)
```

```
value: str = string_from_key(dictionary, “key1”)
```

```
try:
  value: str = string_from_key(dictionary, "key1")
except NoSuchKey:
  print("Given key do not exist")
except TypeMismatch:
  print("Retrieved value is not of the expected type")
```

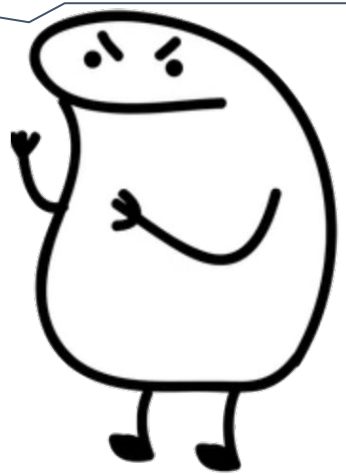


# Mejor Tipado




```python
try:
  value: str = string_from_key(dictionary, "key1")
except NoSuchKey:
  print("Given key do not exist")
except TypeMismatch:
  print("Retrieved value is not of the expected type")
```

except ?, except ?, except ?



# Mejor Tipado

```
value = string_from_key(dictionary, “key1”)
```

```
value: str = string_from_key(dictionary, “key1”)
```

¿Y qué pasa si utilizo
la clase Result?

# Mejor Tipado

```
result = string_from_key(dictionary, “key1”)
```

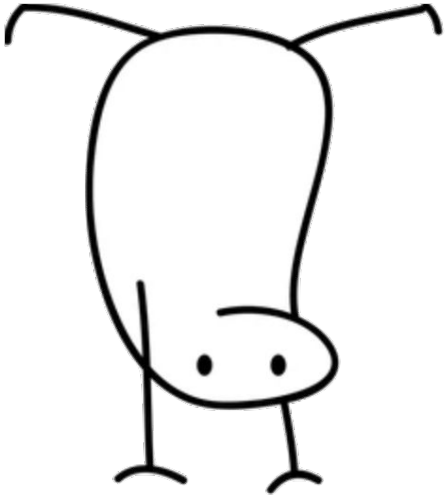
```
result: Result[str, NoSuchKey | TypeMismatch] = string_from_key(dictionary, “key1”)
```

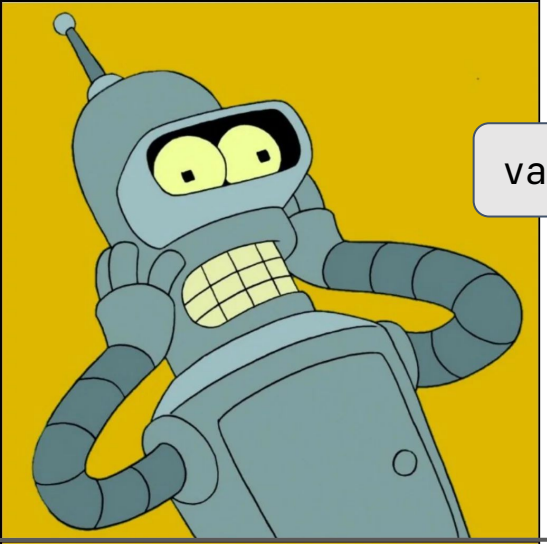
# Mejor Tipado

```
result = string_from_key(dictionary, “key1”)
```

```
result: Result[str, NoSuchKey | TypeMismatch] = string_from_key(dictionary, “key1”)
```

```
result = string_from_key(dictionary, "key1")
match result:
  case Success(value):
    print(value)
  case Failure(NoSuchKey):
    print("Given key do not exist")
  case Failure(TypeMismatch):
    print("Retrieved value is not of the expected type")
```

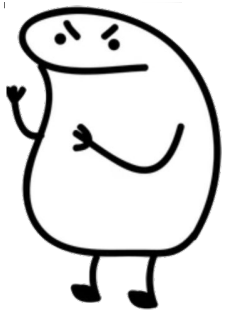


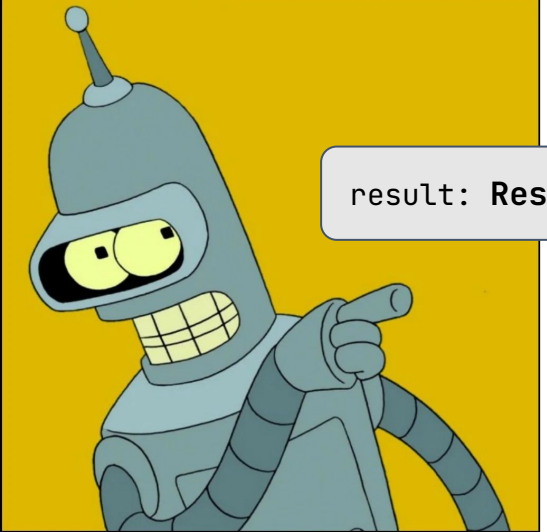


```
value = string_from_key(dictionary, “key1”)
```

```
value: str = string_from_key(dictionary, “key1”)
```

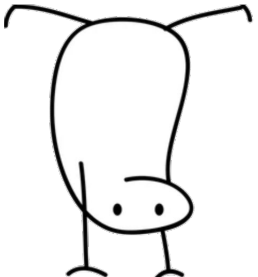
Y si falla…..
¿Cuáles son las excepciones que tengo que capturar?
🤔





```
result = string_from_key(dictionary, “key1”)
```

```
result: Result[str, NoSuchKey | TypeMismatch] = string_from_key(dictionary, “key1”)
```



# ¿Por qué utilizar mónadas?

✅ Nos ayuda a tener un mejor tipado.

➡️ Nos facilita el testing.

Nos habilita a usar funciones derivadas de la programación funcional.

Nos ayuda a crear "código limpio".

# Testing

```python
@pytest.mark.unit
class TestExampleWithoutMeiga:
    dictionary = {"key1": "value", "key2": 2}

    def should_return_a_str(self):
        value = string_from_key(
            dictionary=self.dictionary,
            key="key1"
        )
        assert isinstance(value, str)

    def should_raises_non_such_key_exception(self):
        with pytest.raises(NoSuchKey):
          _ = string_from_key(
              dictionary=self.dictionary,
              key="invalid_key"
          )

    def should_raises_type_mismatch_exception(self):
        with pytest.raises(TypeMismatch):
          value = string_from_key(
             dictionary=self.dictionary,
             key="key2"
          )
          assert not isinstance(value, str)
```

```python
@pytest.mark.unit
class TestExampleWithMeiga:
    dictionary = {"key1": "value", "key2": 2}

    def should_return_a_str(self):
        result = string_from_key(
            dictionary=self.dictionary,
            key="key1"
        )
        result.assert_success(value_is_instance_of=str)


    def should_raises_non_such_key_exception(self):
        result = string_from_key(
            dictionary=self.dictionary,
            key="invalid_key"
        )
        result.assert_failure(value_is_instance_of=NoSuchKey)


    def should_raises_type_mismatch_exception(self):
        result = string_from_key(
           dictionary=self.dictionary,
           key="key2"
        )
        result.assert_failure(value_is_instance_of=TypeMismatch)
```

# Testing

```python
@pytest.mark.unit
class TestExampleWithoutMeiga:
    dictionary = {"key1": "value", "key2": 2}

    def should_return_a_str(self):
        value = string_from_key(
            dictionary=self.dictionary,
            key="key1"
        )
        assert isinstance(value, str)

    def should_raises_non_such_key_exception(self):
        with pytest.raises(NoSuchKey):
            _ = string_from_key(
                dictionary=self.dictionary,
                key="invalid_key"
            )

    def should_raises_type_mismatch_exception(self):
        with pytest.raises(TypeMismatch):
            value = string_from_key(
                dictionary=self.dictionary,
                key="key2"
            )
            assert not isinstance(value, str)
```

```python
@pytest.mark.unit
class TestExampleWithMeiga:
    dictionary = {"key1": "value", "key2": 2}

    def should_return_a_str(self):
        result = string_from_key(
            dictionary=self.dictionary,
            key="key1"
        )
        result.assert_success(value_is_instance_of=str)

    def should_raises_non_such_key_exception(self):
        result = string_from_key(
            dictionary=self.dictionary,
            key="invalid_key"
        )
        result.assert_failure(value_is_instance_of=NoSuchKey)

    def should_raises_type_mismatch_exception(self):
        result = string_from_key(
            dictionary=self.dictionary,
            key="key2"
        )
        result.assert_failure(value_is_instance_of=TypeMismatch)
```

# Testing

```python
@pytest.mark.unit
class TestExampleWithoutMeiga:
    dictionary = {"key1": "value", "key2": 2}

    def should_return_a_str(self):
        value = string_from_key(
            dictionary=self.dictionary,
            key="key1"
        )
        assert isinstance(value, str)

    def should_raises_non_such_key_exception(self):
        with pytest.raises(NoSuchKey):
            _ = string_from_key(
                dictionary=self.dictionary,
                key="invalid_key"
            )

    def should_raises_type_mismatch_exception(self):
        with pytest.raises(TypeMismatch):
            value = string_from_key(
                dictionary=self.dictionary,
                key="key2"
            )
            assert not isinstance(value, str)
```

```python
@pytest.mark.unit
class TestExampleWithMeiga:
    dictionary = {"key1": "value", "key2": 2}

    def should_return_a_str(self):
        result = string_from_key(
            dictionary=self.dictionary,
            key="key1"
        )
        result.assert_success(
          value_is_instance_of=str,
          value_is_equal_to="value"
        )

    def should_raises_non_such_key_exception(self):
        result = string_from_key(
            dictionary=self.dictionary,
            key="invalid_key"
        )
        result.assert_failure(value_is_instance_of=NoSuchKey)

    def should_raises_type_mismatch_exception(self):
        result = string_from_key(
            dictionary=self.dictionary,
            key="key2"
        )
        result.assert_failure(value_is_instance_of=TypeMismatch)
```

# Testing

```
def should_fail_when_app_service_error(self) → None:                    Arrange
    self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
    self.mock_app_service.execute = Mock(return_value=Failure(AppServiceError()))
                                                                          Act
    use_case = UserUpdater(self.mock_repository, self.mock_app_service)
    result = use_case.execute(UuidMother.any())
                                                                          Assert
    result.assert_failure(value_is_instance_of=AppServiceError())
    self.mock_repository.retrieve.assert_called_once()
    self.mock_app_service.execute.assert_called_once()
```

# ¿Por qué utilizar mónadas?

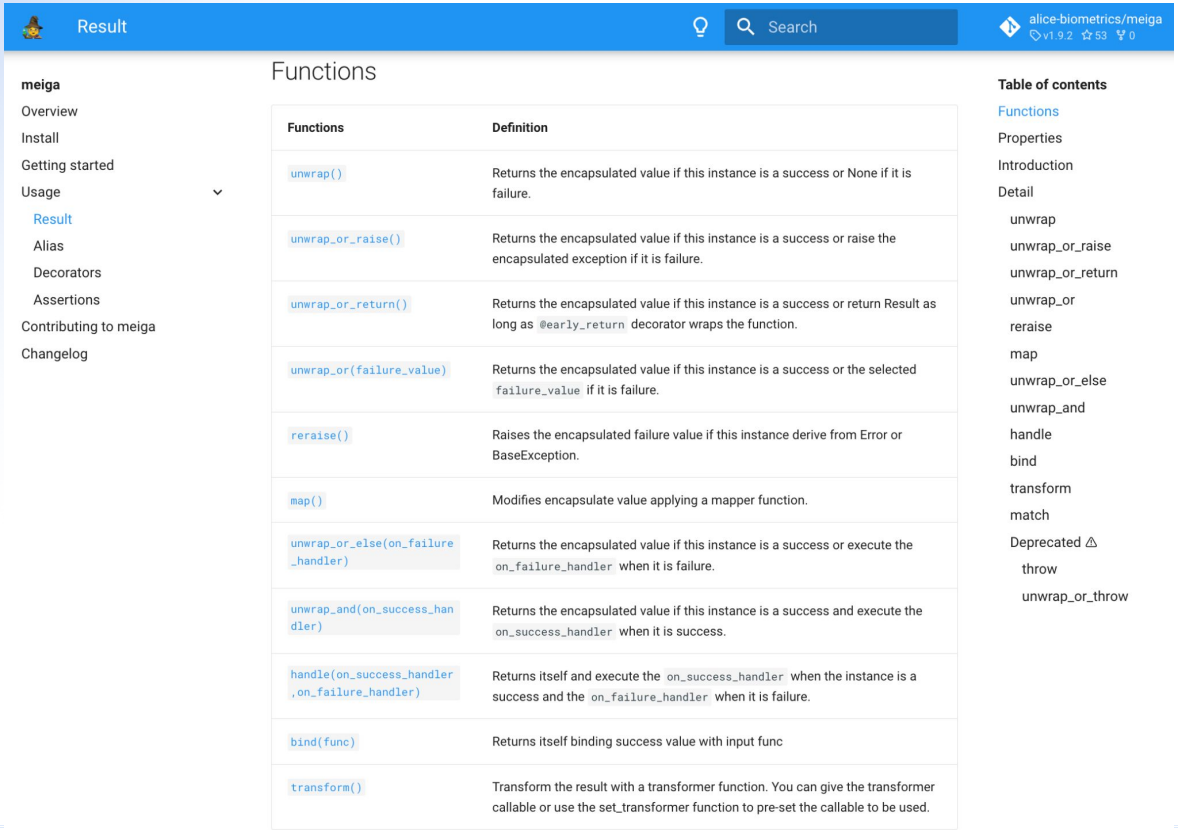
✅ Nos ayuda a tener un mejor tipado.

✅ Nos facilita el testing.

➡️ Nos habilita a usar funciones derivadas de la programación funcional.

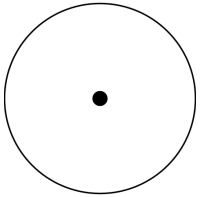
Nos ayuda a crear ”código limpio”.

# Programación funcional



# bind

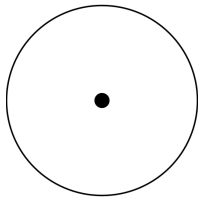
Success



Failure

{ value }

# bind

Success  Failure

*f(* { value } *)* if Success

# bind

```
user = {
     “name”: “rosalia de castro”,
     “age”: 186
}
```

`f( user )` Nombre en Mayúsculas

↓

`f( user )` Aniversario 🥳

↓

`f( user )` E ti, de onde ves sendo?

# bind

```
user = {
    “name”: “rosalia de castro”,
    “age”: 186
}
```

*f(* user *)* Nombre en Mayúsculas

*f(* user *)* Aniversario 🥳

*f(* user *)* E ti, de onde ves sendo?

```
user = {
    “name”: “ROSALIA DE CASTRO”,
    “age”: 187,
    “location”: “GALIZA”,
}
```

# bind

```
user = {
    “name”: “rosalia de castro”,
    “age”: 186
}
```

`f( user )` Nombre en Mayúsculas

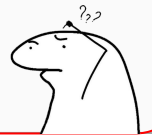
↓

`f( user )` Aniversario 🥳

↓

`f( user )` E ti, de onde ves sendo?

```python
from typing import Any

from meiga import Result, Success, Error

def upper_name(value: dict) -> dict:
    value.update({"name":value["name"].upper()})
    return value

def update_age(value: dict) -> dict:
    value.update({"age": value["age"] + 1})
    return value

def add_location(value: dict) -> dict:
    value.update({"location": "GALIZA"})
    return value

def update(user: dict) -> dict:
    if user:
        user = upper_name(user)
        if user:
            user = update_age(user)
            if user:
                user = add_location(user)
        return user
```



# bind

```
user = {
    “name”: “rosalia de castro”,
    “age”: 186
}
```

check

`f( user )` Nombre en Mayúsculas

check

`f( user )` Aniversario 🥳

check

`f( user )` E ti, de onde ves sendo?

```python
from typing import Any

from meiga import Result, Success, Error

def upper_name(value: dict) -> dict:
    value.update({"name":value["name"].upper()})
    return value

def update_age(value: dict) -> dict:
    value.update({"age": value["age"] + 1})
    return value

def add_location(value: dict) -> dict:
    value.update({"location": "GALIZA"})
    return value

def update(user: dict) -> dict:
    if user:
        user = upper_name(user)
        if user:
            user = update_age(user)
            if user:
                user = add_location(user)
        return user
```

# bind

```
user = {
    “name”: “rosalia de castro”,
    “age”: 186
}
```

check

`f( user )` Nombre en Mayúsculas

check

`f( user )` Aniversario 🥳

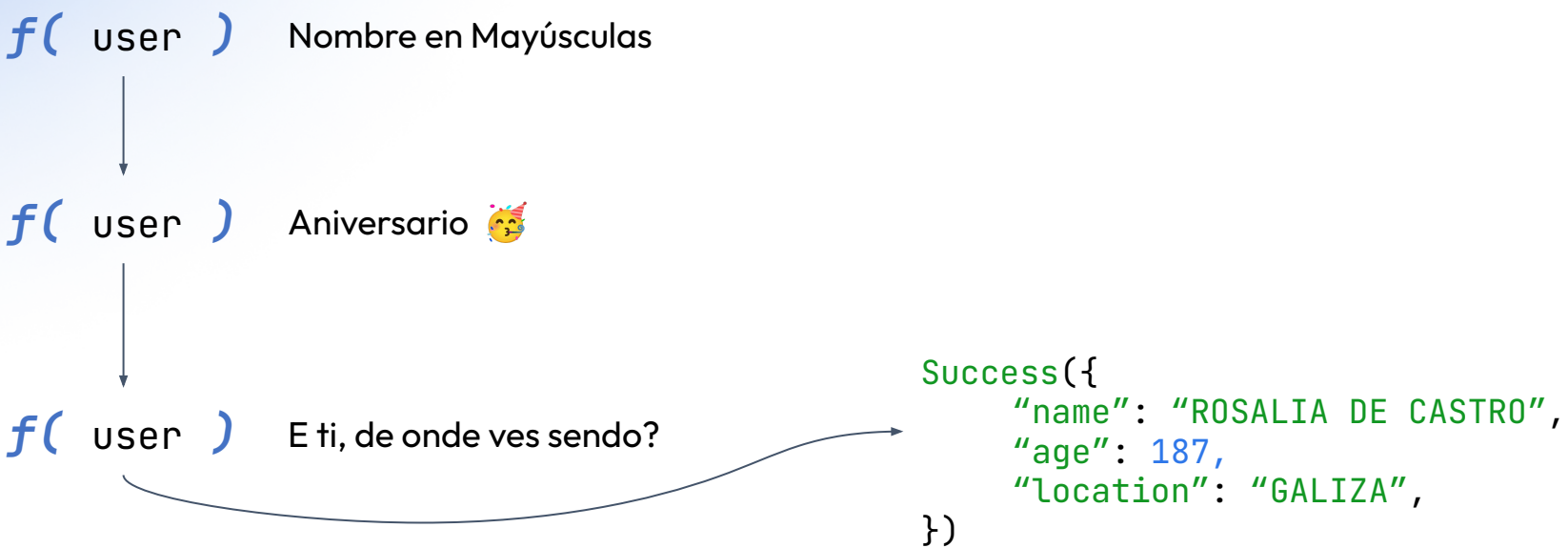
check

`f( user )` E ti, de onde ves sendo?

```python
from typing import Any

from meiga import Result, Success, Error

def upper_name(value: dict) -> dict:
    value.update({"name":value["name"].upper()})
    return value

def update_age(value: dict) -> dict:
    value.update({"age": value["age"] + 1})
    return value

def add_location(value: dict) -> dict:
    value.update({"location": "GALIZA"})
    return value

def update(user: dict) -> dict:
    if user:
        user = upper_name(user)
        if user:
            user = update_age(user)
            if user:
                user = add_location(user)
        return user
```

# bind

```
Success({
    “name”: “rosalia de castro”,
    “age”: 186
})
```




```
Success({
    “name”: “ROSALIA DE CASTRO”,
    “age”: 187,
    “location”: “GALIZA”,
})
```

# bind

```
Success({
    “name”: “rosalia de castro”,
    “age”: 186
})
```

`f( user )` Nombre en Mayúsculas

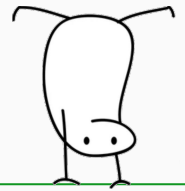


`f( user )` Aniversario 🥳

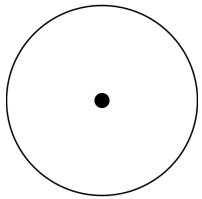


`f( user )` E ti, de onde ves sendo?

```python
from typing import Any

from meiga import Result, Success, Error

def upper_name(value: dict) -> dict:
    value.update({"name":value["name"].upper()})
    return value

def update_age(value: dict) -> dict:
    value.update({"age": value["age"] + 1})
    return value

def add_location(value: dict) -> dict:
    value.update({"location": "GALIZA"})
    return value

def update(result: Result[dict, Error]) -> Result[dict, Error]:
    return (
          result
          .bind(upper_name)
          .bind(update_age)
          .bind(add_location)
      )
```



# handle

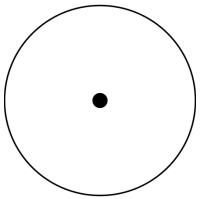
Success  Failure

*f(* { value }*, args)* if Success

*f(* { value }*, args)* if Failure

# handle

Success

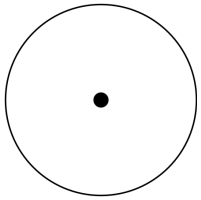


Failure

`f( { value }, args)` if Success → Publish to RabbitMQ

`f( { value }, args)` if Failure → Notify to Slack

# handle

Success  Failure

*f*( { value }, *args*) if Success

*f*( { value }, *args*) if Failure

```
.handle(
    on_success_handler: OnSuccessHandler(func=f, args=(arg1, arg2)),
    on_failure_handler: OnFailureHandler(func=f, args=(arg1, arg2))
)
```

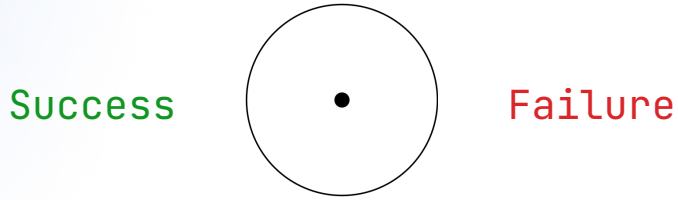
# handle


*f*( { value }, *args*) if Success

*f*( { value }, *args*) if Failure

```
.handle(
    on_success_handler: OnSuccessHandler(func=f, args=(arg1, arg2)),
    on_failure_handler: OnFailureHandler(func=f, args=(arg1, arg2))
)
```

# handle




*f*( { value }, *args*) if Success

*f*( { value }, *args*) if Failure

```
.handle(
    on_success_handler: OnSuccessHandler(func=f, args=(arg1, arg2)),
    on_failure_handler: OnFailureHandler(func=f, args=(arg1, arg2))
)
```

# handle

```python
from meiga import OnSuccessHandler, OnFailureHandler

def publish_event(domain_event_bus: DomainEventBus):
    domain_event_bus.publish(ValueRetrieved())

def notify_error(notifier: Notifier, use_case: str):
    notifier.post(use_case, "Value cannot be retrieved")

result = string_from_key(dictionary=user, key="key1")

result.handle(
  on_success_handler=OnSuccessHandler(
    func=publish_event, args=(domain_event_bus,)
  ),
  on_failure_handler=OnFailureHandler(
    func=notify_error, args=(notifier, "PyCon ES",)
  )
)
```

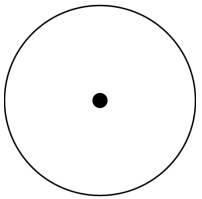
# handle

```python
from meiga import OnSuccessHandler, OnFailureHandler

def publish_event(domain_event_bus: DomainEventBus):
    domain_event_bus.publish(ValueRetrieved())

def notify_error(notifier: Notifier, use_case: str):
    notifier.post(use_case, "Value cannot be retrieved")

result = string_from_key(dictionary=user, key="key1")

result.handle(
  on_success_handler=OnSuccessHandler(
    func=publish_event, args=(domain_event_bus,)
  ),
  on_failure_handler=OnFailureHandler(
    func=notify_error, args=(notifier, "PyCon ES",)
  )
)
```

# handle

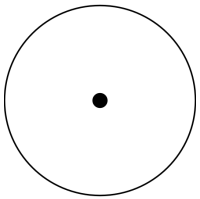
```python
from meiga import OnSuccessHandler, OnFailureHandler

def publish_event(domain_event_bus: DomainEventBus):
    domain_event_bus.publish(ValueRetrieved())

def notify_error(notifier: Notifier, use_case: str):
    notifier.post(use_case, "Value cannot be retrieved")

result = string_from_key(dictionary=user, key="key1")

result.handle(
  on_success_handler=OnSuccessHandler(
    func=publish_event, args=(domain_event_bus,)
  ),
  on_failure_handler=OnFailureHandler(
    func=notify_error, args=(notifier, "PyCon ES",)
  )
)
```

# map

Success  Failure

*f(* { value } *)* if Success or Failure

# transformer

Success  Failure

*f*( Result ) if Success or Failure

# map

```python
def mapper(value: str):
    if not isistance(value, str):
      return value
    return value.capitalize()

result = string_from_key(dictionary,key)

result.map(mapper)
```

# transformer

```python
def transformer(result: Result) -> tuple[int, str]:
    match result:
      case Success(value):
         return 200, value.capitalize()
      case Failure(NoSuchKey):
         raise 404, "Key not found"
      case Failure(TypeMissmatch):
         raise 422, "Given key doesn't have a str"


result = string_from_key(dictionary, key)
status_code, message = result.transform(transformer)
```

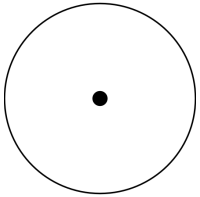
# map

```python
def mapper(value: str):
    if not isistance(value, str):
      return value
    return value.capitalize()

result = string_from_key(dictionary,key)

result.map(mapper)
```

# transformer

```python
def my_controller() -> Result[str, Error]:
  def transformer(result: Result) -> tuple[int, str]:
      match result:
        case Success(value):
           return 200, value.capitalize()
        case Failure(NoSuchKey):
           raise 404, "Key not found"
        case Failure(TypeMissmatch):
           raise 422, "Given key doesn't have a str"

  result = string_from_key(dictionary, key)
  result.set_transformer(transformer)
  return result


status_code, message = my_controller.transform()
```

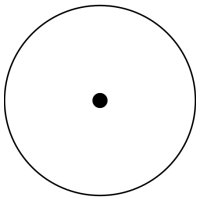
# unwrap

Success  Failure

{ value } if Success

None if Failure

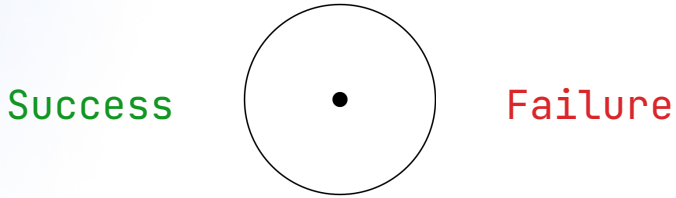
# unwrap_or

Success  Failure

{ value } if Success

***<given value>*** if Failure

# unwrap_or

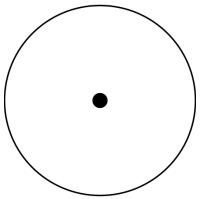



{ value } if Success

***<given value>*** if Failure

```
users = fetch_usernames().unwrap_or([])
for user in users:
    notify(user)
```

# unwrap_or_return

Success  Failure

{ value } if Success

***force return*** if Failure

# unwrap_or_return

```
from meiga import Result, isSuccess, early_return

def CreateUser(UseCase)
  sanitizer: UserSanitizer
  repository: UserRepository
  bus: DomainEventBus

  @early_return
  def execute(self, user: User) -> Result[bool, InvalidInput | UserAlreadyExist]:
    sanitized_user = self.sanitizer.execute(user).unwrap_or_return()
    self.repository.save(sanitized_user).unwrap_or_return()
    self.bus.publish(UserCreated()).unwrap_or_return()
    return isSuccess
```

# unwrap_or_return

```python
from meiga import Result, isSuccess, early_return

def CreateUser(UseCase)
  sanitizer: UserSanitizer
  repository: UserRepository
  bus: DomainEventBus

  @early_return
  def execute(self, user: User) -> Result[bool, InvalidInput | UserAlreadyExist]:
    sanitized_user = self.sanitizer.execute(user).unwrap_or_return()
    sanitized_user: User = self.sanitizer.execute(..
    self.bus.publish(UserCreated()).unwrap_or_return()
    return isSuccess
```

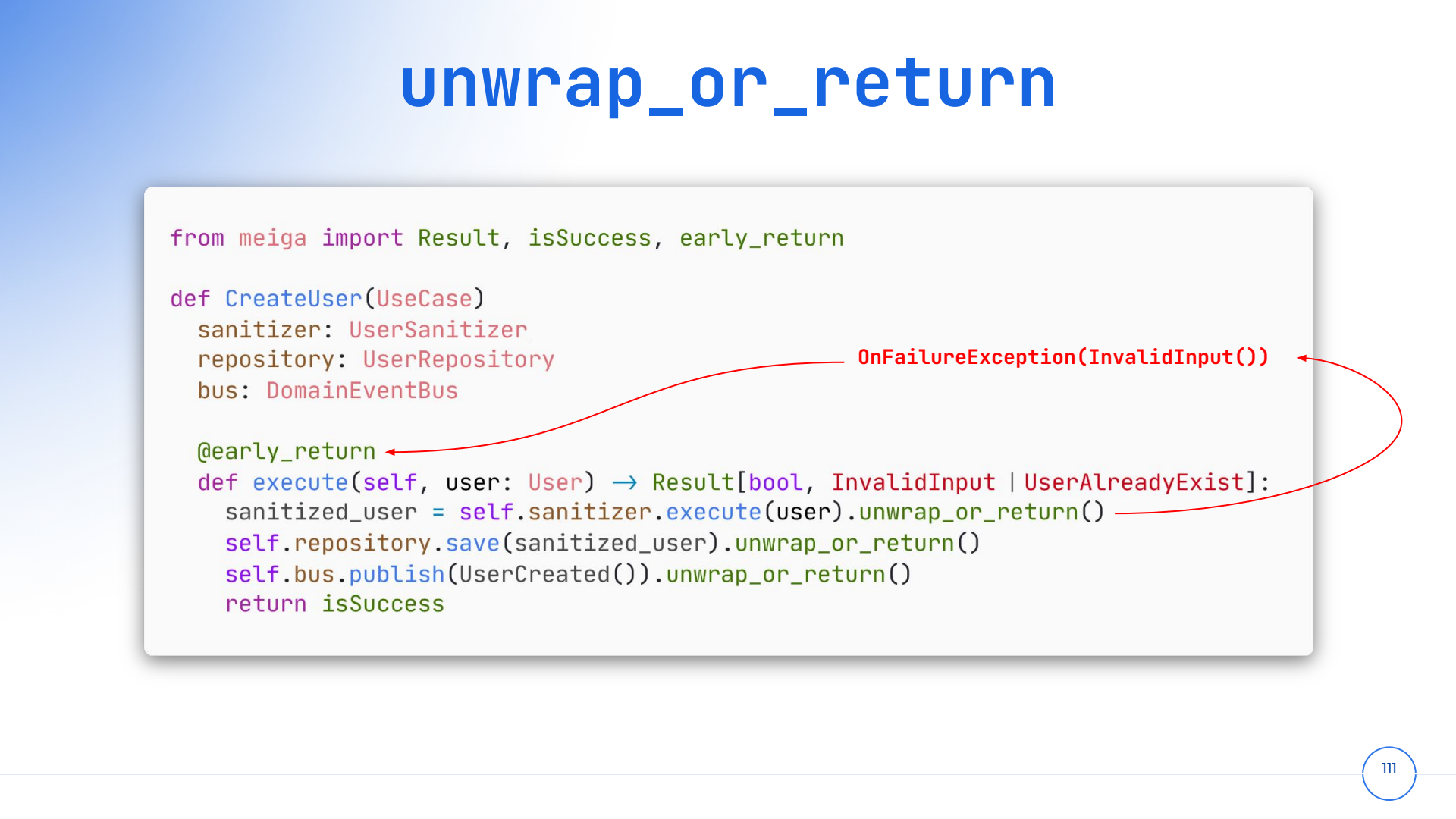
# unwrap_or_return

```python
from meiga import Result, isSuccess, early_return

def CreateUser(UseCase)
  sanitizer: UserSanitizer
  repository: UserRepository
  bus: DomainEventBus

  @early_return
  def execute(self, user: User) -> Result[bool, InvalidInput | UserAlreadyExist]:
    sanitized_user = self.sanitizer.execute(user).unwrap_or_return()
    self.repository.save(sanitized_user).unwrap_or_return()
    self.bus.publish(UserCreated()).unwrap_or_return()
    return isSuccess
```

```python
def execute(self, user: User) -> Result[User, InvalidInput]
```

# unwrap_or_return

```python
from meiga import Result, isSuccess, early_return

def CreateUser(UseCase)
  sanitizer: UserSanitizer
  repository: UserRepository
  bus: DomainEventBus

  @early_return
  def execute(self, user: User) -> Result[bool, InvalidInput | UserAlreadyExist]:
    sanitized_user = self.sanitizer.execute(user).unwrap_or_return()
    self.repository.save(sanitized_user).unwrap_or_return()
    self.bus.publish(UserCreated()).unwrap_or_return()
    return isSuccess
```

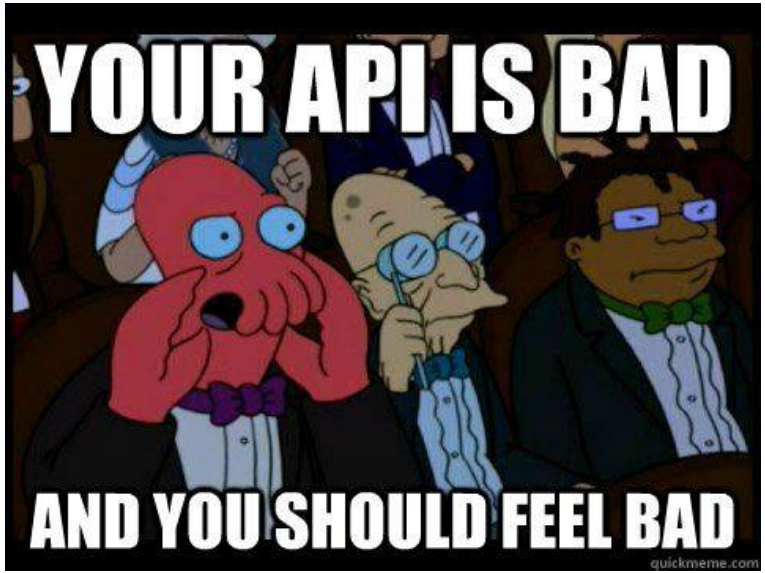


# ¿Por qué utilizar mónadas?

- ✅ Nos ayuda a tener un mejor tipado.
- ✅ Nos facilita el testing.
- ✅ Nos habilita a usar funciones derivadas de la programación funcional.
- ➡️ Nos ayuda a crear ”código limpio”.

# Dónde nos ayuda realmente

A la hora de desarrollar aplicaciones web en backend.



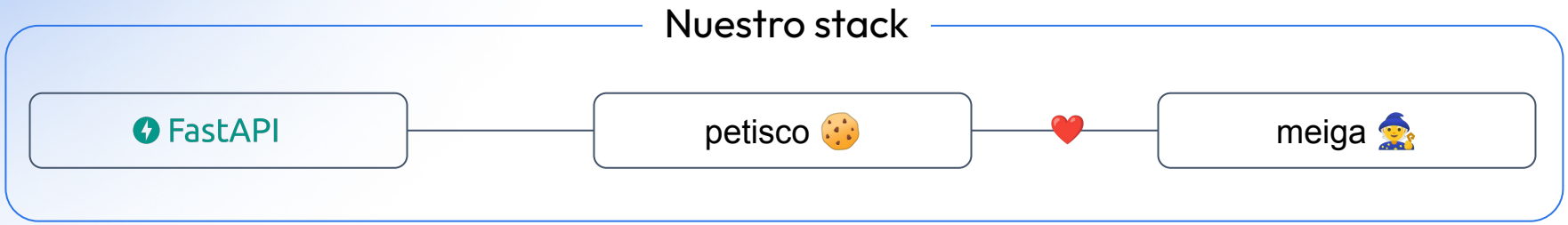
# Dónde nos ayuda realmente

A la hora de desarrollar aplicaciones web en backend.

- ✅ Nos ayuda a tener un mejor tipado.
- ✅ Nos facilita el testing.
- Mapear los errores de dominio a errores HTTP.
- Nos reduce el ruido visual de los casos de uso.

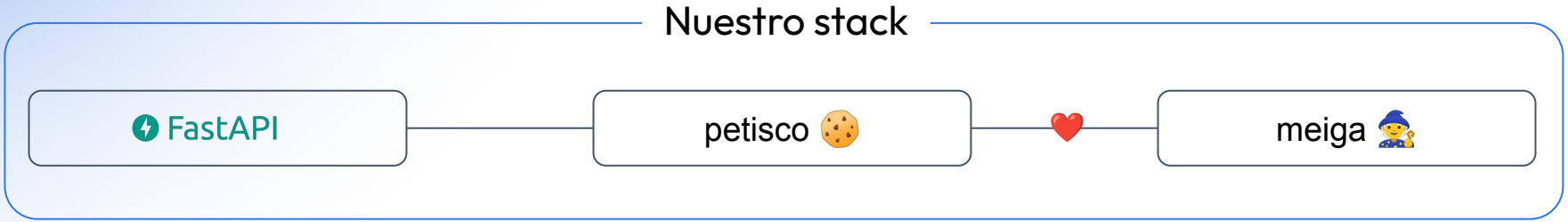
# Dónde nos ayuda realmente

A la hora de desarrollar aplicaciones web en backend.



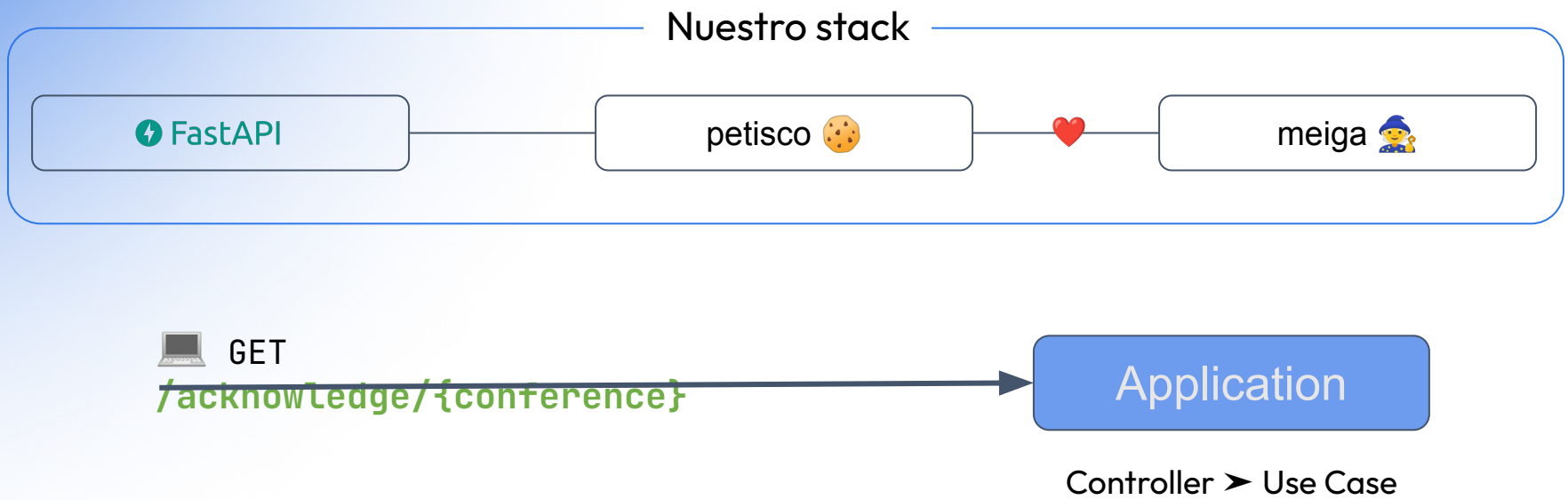
# Dónde nos ayuda realmente

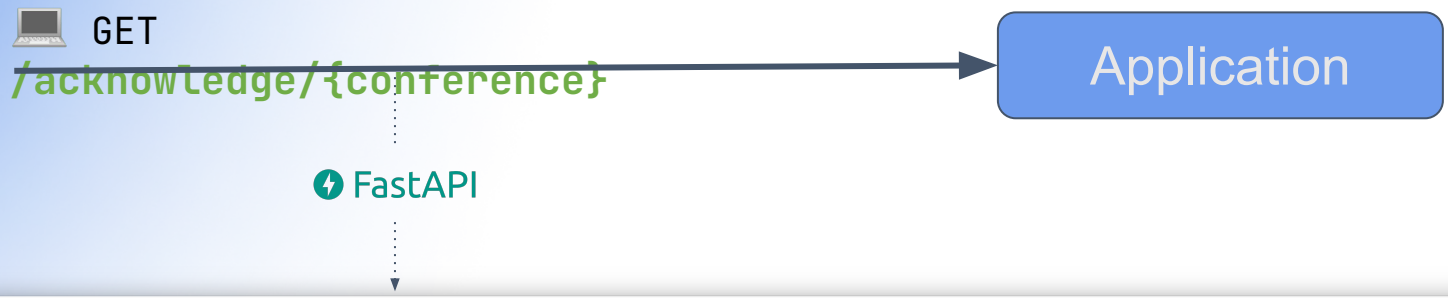
A la hora de desarrollar aplicaciones web en backend.




```
pip install petisco[fastapi] ⭐
```

https://github.com/alice-biometrics/petisco


Nuestro stack
FastAPI
petisco 🍪
❤️
meiga 🧙
💻 GET
/acknowledge/{conference}
Application
Controller ➤ Use Case




```python
from fastapi import FastAPI
from petisco.extra.fastapi import as_fastapi


app = FastAPI()

@app.get("acknowledge/{conference}")
def get_acknowledge(conference: str) -> HTML:
    result = GetAcknowledgeController().execute(conference)
    return as_fastapi(result)
```


```python
class GetAcknowledgeController(Controller):

    class Config:
        error_map = {
            OrganizersNotFound: HttpError(status_code=404, detail="Organizer not found"),
            SponsorsNotFound: HttpError(status_code=404, detail="Sponsors not found"),
            MessageTooLarge: HttpError(status_code=422, detail="The html message exceeds the limits"),
        }

    def execute(self, conference: str) -> Result[dict, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        return AcknowledgeCreator(
            organizers_repository=Container.get(OrganizersRepository),
            sponsors_repository=Container.get(SponsorsRepository),
            html_formatter=Container.get(HtmlFormatter)
        ).execute(conference)
```


```python
class GetAcknowledgeController(Controller):

    class Config:
        error_map = {
            OrganizersNotFound: HttpError(status_code=404, detail="Organizer not found"),
            SponsorsNotFound: HttpError(status_code=404, detail="Sponsors not found"),
            MessageTooLarge: HttpError(status_code=422, detail="The html message exceeds the limits"),
        }

    def execute(self, conference: str) -> Result[dict, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        return AcknowledgeCreator(
            organizers_repository=Container.get(OrganizersRepository),
            sponsors_repository=Container.get(SponsorsRepository),
            html_formatter=Container.get(HtmlFormatter)
        ).execute(conference)
```

Instanciar un Caso de Uso


```python
class GetAcknowledgeController(Controller):

    class Config:
        error_map = {
            OrganizersNotFound: HttpError(status_code=404, detail="Organizer not found"),
            SponsorsNotFound: HttpError(status_code=404, detail="Sponsors not found"),
            MessageTooLarge: HttpError(status_code=422, detail="The html message exceeds the limits"),
        }

    def execute(self, conference: str) -> Result[dict, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        return AcknowledgeCreator(
            organizers_repository=Container.get(OrganizersRepository),
            sponsors_repository=Container.get(SponsorsRepository),
            html_formatter=Container.get(HtmlFormatter)
        ).execute(conference)
```

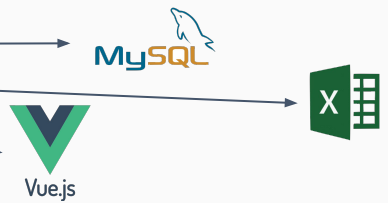
Inyecta las dependencias

💻 GET /acknowledge/{conference}

Application

Controller ➤ Use Case

```python
class GetAcknowledgeController(Controller):

    class Config:
        error_map = {
            OrganizersNotFound: HttpError(status_code=404, detail="Organizer not found"),
            SponsorsNotFound: HttpError(status_code=404, detail="Sponsors not found"),
            MessageTooLarge: HttpError(status_code=422, detail="The html message exceeds the limits"),
        }

    def execute(self, conference: str) -> Result[dict, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        return AcknowledgeCreator(
            organizers_repository=Container.get(OrganizersRepository),
            sponsors_repository=Container.get(SponsorsRepository),
            html_formatter=Container.get(HtmlFormatter)
        ).execute(conference)
```




```python
class GetAcknowledgeController(Controller):

    class Config:
        error_map = {
            OrganizersNotFound: HttpError(status_code=404, detail="Organizer not found"),
            SponsorsNotFound: HttpError(status_code=404, detail="Sponsors not found"),
            MessageTooLarge: HttpError(status_code=422, detail="The html message exceeds the limits"),
        }

    def execute(self, conference: str) -> Result[dict, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        return AcknowledgeCreator(
            organizers_repository=Container.get(OrganizersRepository),
            sponsors_repository=Container.get(SponsorsRepository),
            html_formatter=Container.get(HtmlFormatter)
        ).execute(conference)
```


```python
class GetAcknowledgeController(Controller):

    class Config:
        error_map = {
            OrganizersNotFound: HttpError(status_code=404, detail="Organizer not found"),
            SponsorsNotFound: HttpError(status_code=404, detail="Sponsors not found"),
            MessageTooLarge: HttpError(status_code=422, detail="The html message exceeds the limits"),
        }

    def execute(self, conference: str) -> Result[dict, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        return AcknowledgeCreator(
            organizers_repository=Container.get(OrganizersRepository),
            sponsors_repository=Container.get(SponsorsRepository),
            html_formatter=Container.get(HtmlFormatter)
        ).execute(conference)
```


```python
from meiga import Result, Error
from petisco import UseCase

class OrganizersNotFound(Error): ...
class SponsorsNotFound(Error): ...
class MessageTooLarge(Error): ...


class AcknowledgeCreator(UseCase):
    organizers_repository: OrganizersRepository
    sponsors_repository: SponsorsRepository
    html_formatter: HtmlFormatter

    def execute(
        self, conference: str
    ) -> Result[HTML, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        organizers = self.organizers_repository.retrieve(conference).unwrap_or_return()
        sponsors = self.sponsors_repository.retrive(conference).unwrap_or_return()
        return self.html_formatter.execute(organizers, sponsors)
```


```python
from meiga import Result, Error
from petisco import UseCase

class OrganizersNotFound(Error): ...
class SponsorsNotFound(Error): ...
class MessageTooLarge(Error): ...


class AcknowledgeCreator(UseCase):
    organizers_repository: OrganizersRepository
    sponsors_repository: SponsorsRepository
    html_formatter: HtmlFormatter

    def execute(
        self, conference: str
    ) -> Result[HTML, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        organizers = self.organizers_repository.retrieve(conference).unwrap_or_return()
        self.sponsors_repository.retrive(conference).unwrap_or_return()
        ...sors)
```

```
organizers: list[Organizer] = self.organizers_repository..
```


```python
from meiga import Result, Error
from petisco import UseCase

class OrganizersNotFound(Error): ...
class SponsorsNotFound(Error): ...
class MessageTooLarge(Error): ...


class AcknowledgeCreator(UseCase):
    organizers_repository: OrganizersRepository
    sponsors_repository: SponsorsRepository
    html_formatter: HtmlFormatter

    def execute(
        self, conference: str
    ) -> Result[HTML, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        organizers = self.organizers_repository.retrieve(conference).unwrap_or_return()
        sponsors = self.sponsors_repository.retrieve(conference).unwrap_or_return()
        return self.html_formatter.format(organizers, sponsors)
```

```python
def retrieve(self, conference: str) -> Result[list[Organizer], OrganizersNotFound]
```

💻 GET
/acknowledge/{conference}


Controller ➤ Use Case

```python
from meiga import Result, Error
from petisco import UseCase

class OrganizersNotFound(Error): ...
class SponsorsNotFound(Error): ...
class MessageTooLarge(Error): ...


class AcknowledgeCreator(UseCase):
    organizers_repository: OrganizersRepository
    sponsors_repository: SponsorsRepository
    html_formatter: HtmlFormatter

    def execute(
        self, conference: str
    ) -> Result[HTML, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        organizers = self.organizers_repository.retrieve(conference).unwrap_or_return()
        sponsors = self.sponsors_repository.retrive(conference).unwrap_or_return()
        self.html_formatter.execute(organizers, sponsors)
```

```
sponsors: list[Sponsor] = self.sponsors_repository..
```


```python
from meiga import Result, Error
from petisco import UseCase

class OrganizersNotFound(Error): ...
class SponsorsNotFound(Error): ...
class MessageTooLarge(Error): ...


class AcknowledgeCreator(UseCase):
    organizers_repository: OrganizersRepository
    sponsors_repository: SponsorsRepository
    html_formatter: HtmlFormatter

    def execute(
        self, conference: str
    ) -> Result[HTML, OrganizersNotFound | SponsorsNotFound | MessageTooLarge]:
        organizers = self.organizers_repository.retrieve(conference).unwrap_or_return()
        sponsors = self.sponsors_repository.retrive(conference).unwrap_or_return()
        return self.html_formatter.format(organizers, sponsors)
```

```python
def retrieve(self, conference: str) -> Result[list[Sponsor], SponsorsNotFound]
```

💻 GET /acknowledge/{conference}


Controller ➤ Use Case

```python
from meiga import Result, Error
from petisco import UseCase

class OrganizersNotFound(Error): ...
class SponsorsNotFound(Error): ...
class MessageTooLarge(Error): ...


class AcknowledgeCreator(UseCase):
    organizers_repository: OrganizersRepository
    sponsors_repository: SponsorsRepository
    html_formatter: HtmlFormatter

    def execute(
        self, confere
    ) -> Result[HTML,
        organizers =
        sponsors = se
        return self.html_formatter.execute(organizers, sponsors)
```

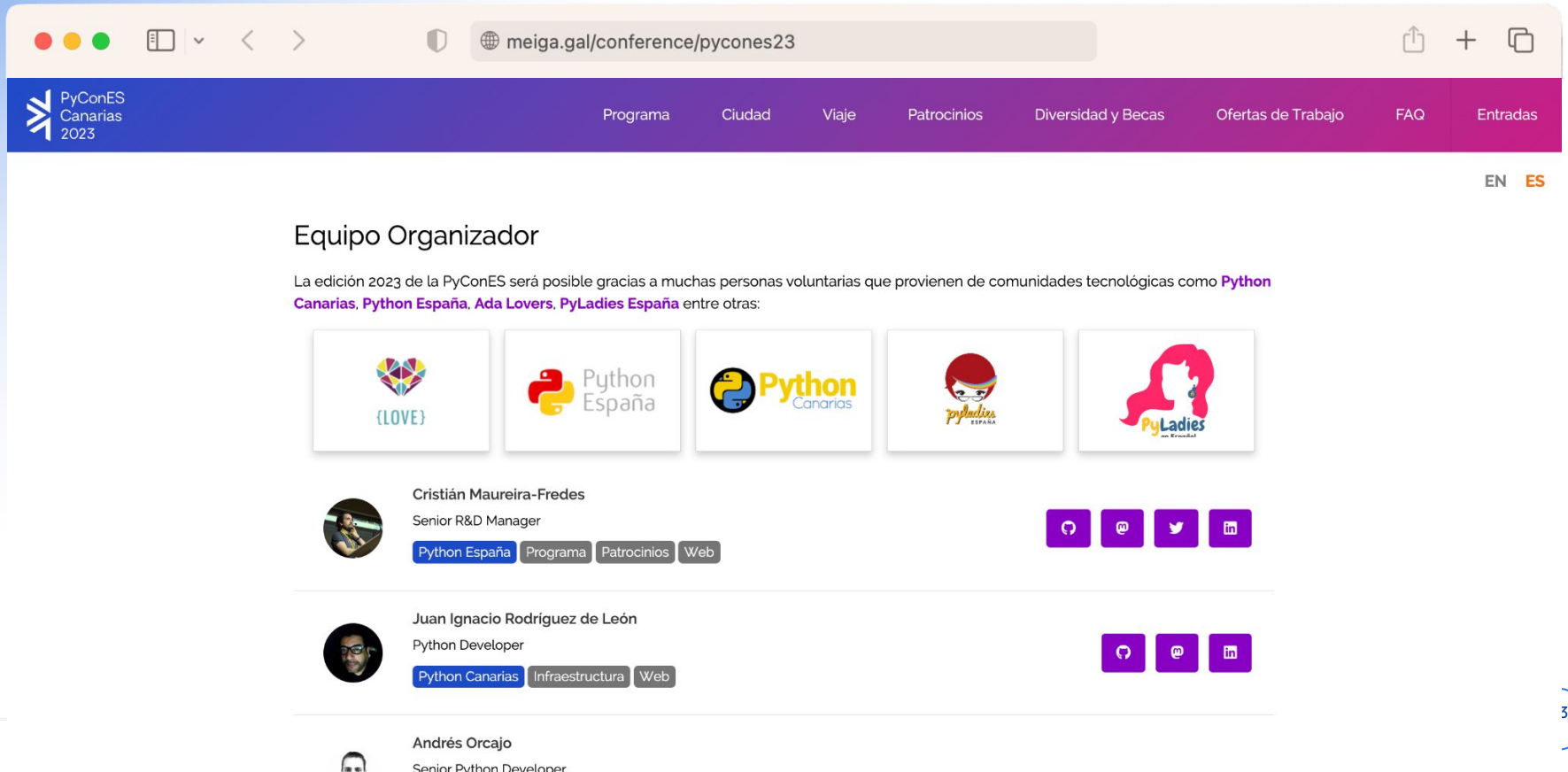
```python
def execute(self,
            organizers: list[Organizer],
            sponsors: list[Sponsor]) -> Result[HTML, MessageTooLarge]
```


GET
/acknowledge/{conference}
Application


1. Recupera los Organizadores.
2. Recupera los Patrocinadores.
3. Crea un HTML para presentar la info.

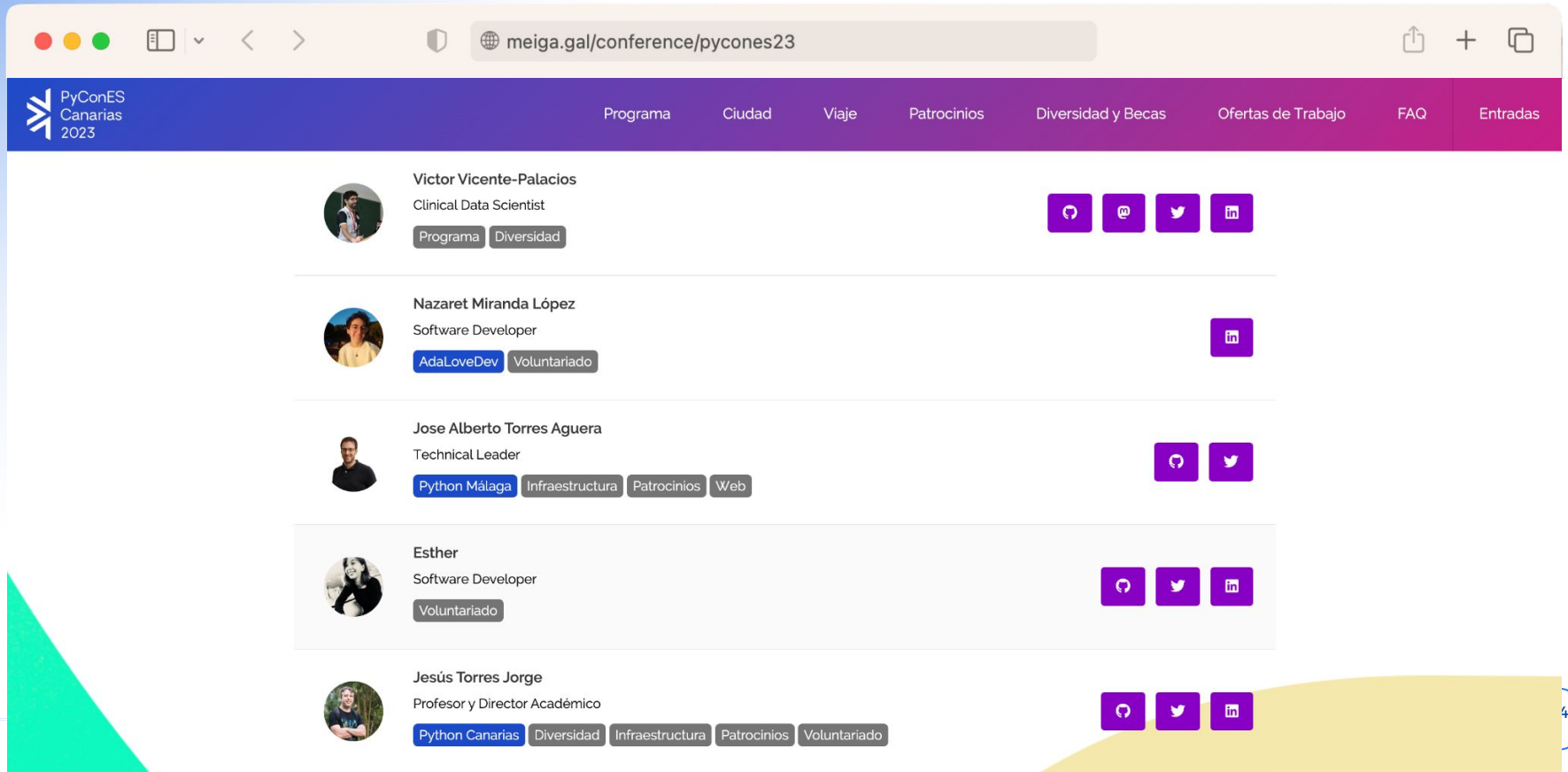
GET /acknowledge/pycones23

Application

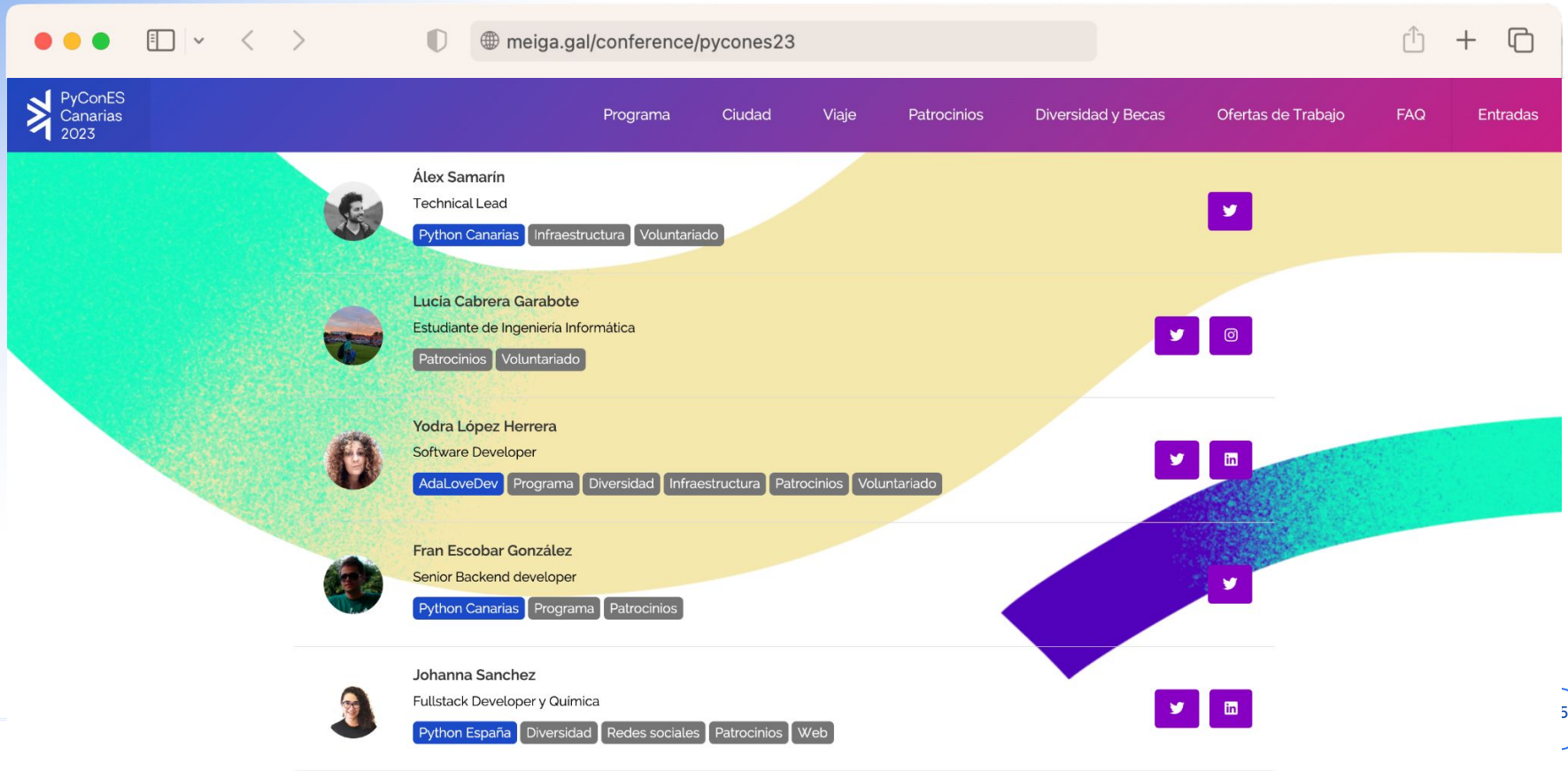


GET /acknowledge/pycones23

Application

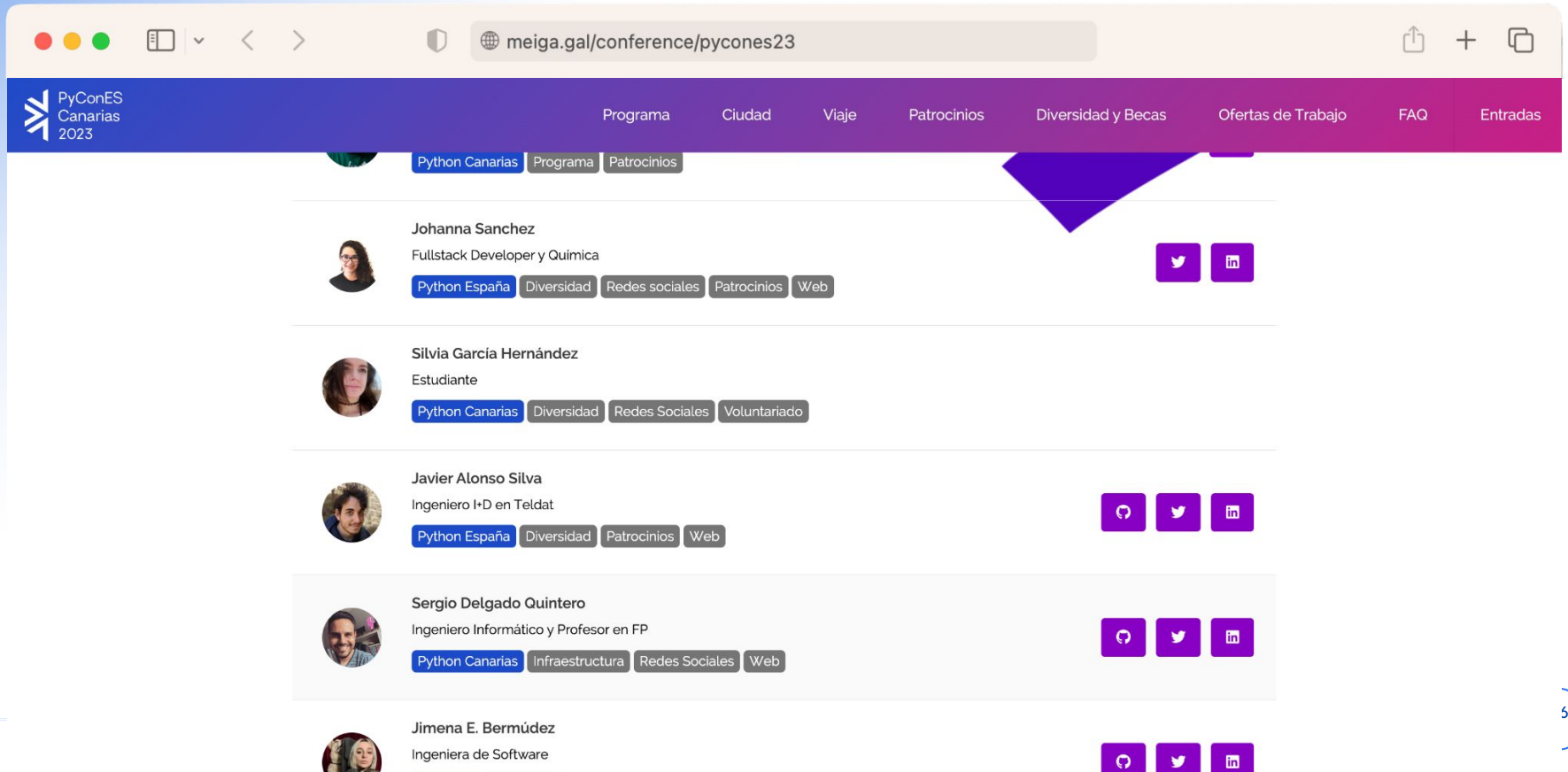


💻 `GET /acknowledge/pycones23` → Application

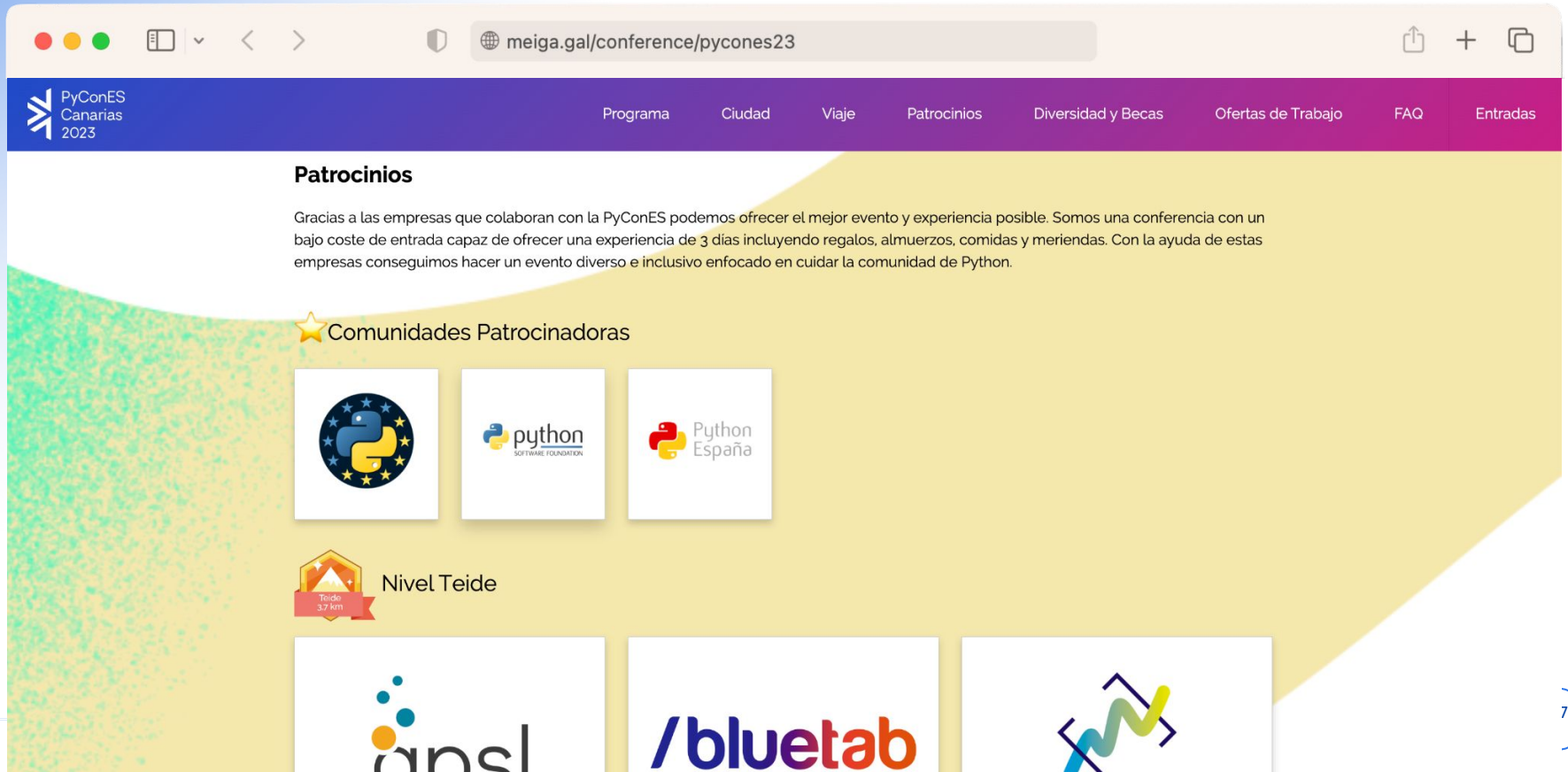


GET /acknowledge/pycones23

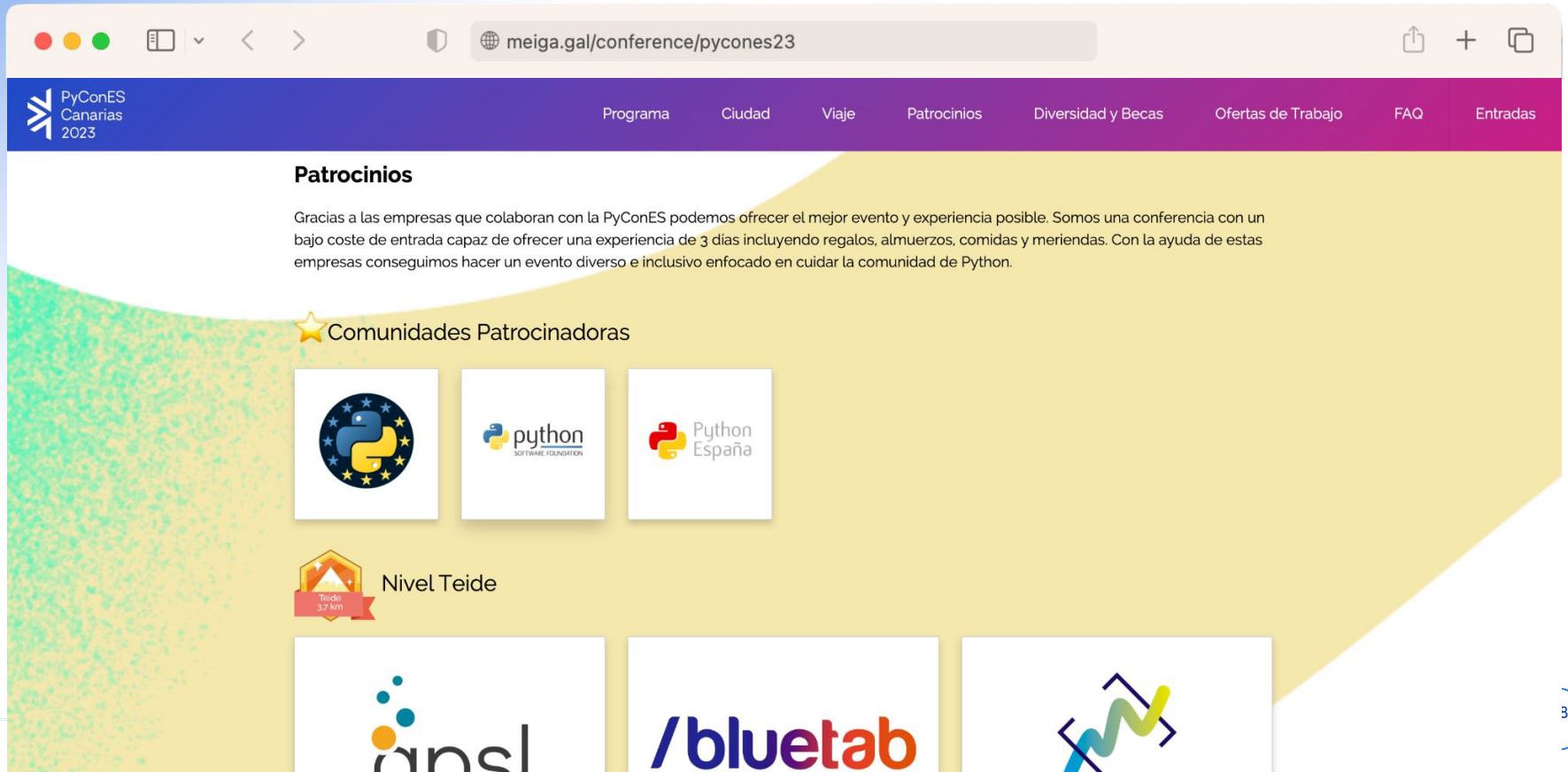
Application



💻 `GET /acknowledge/pycones23` → Application

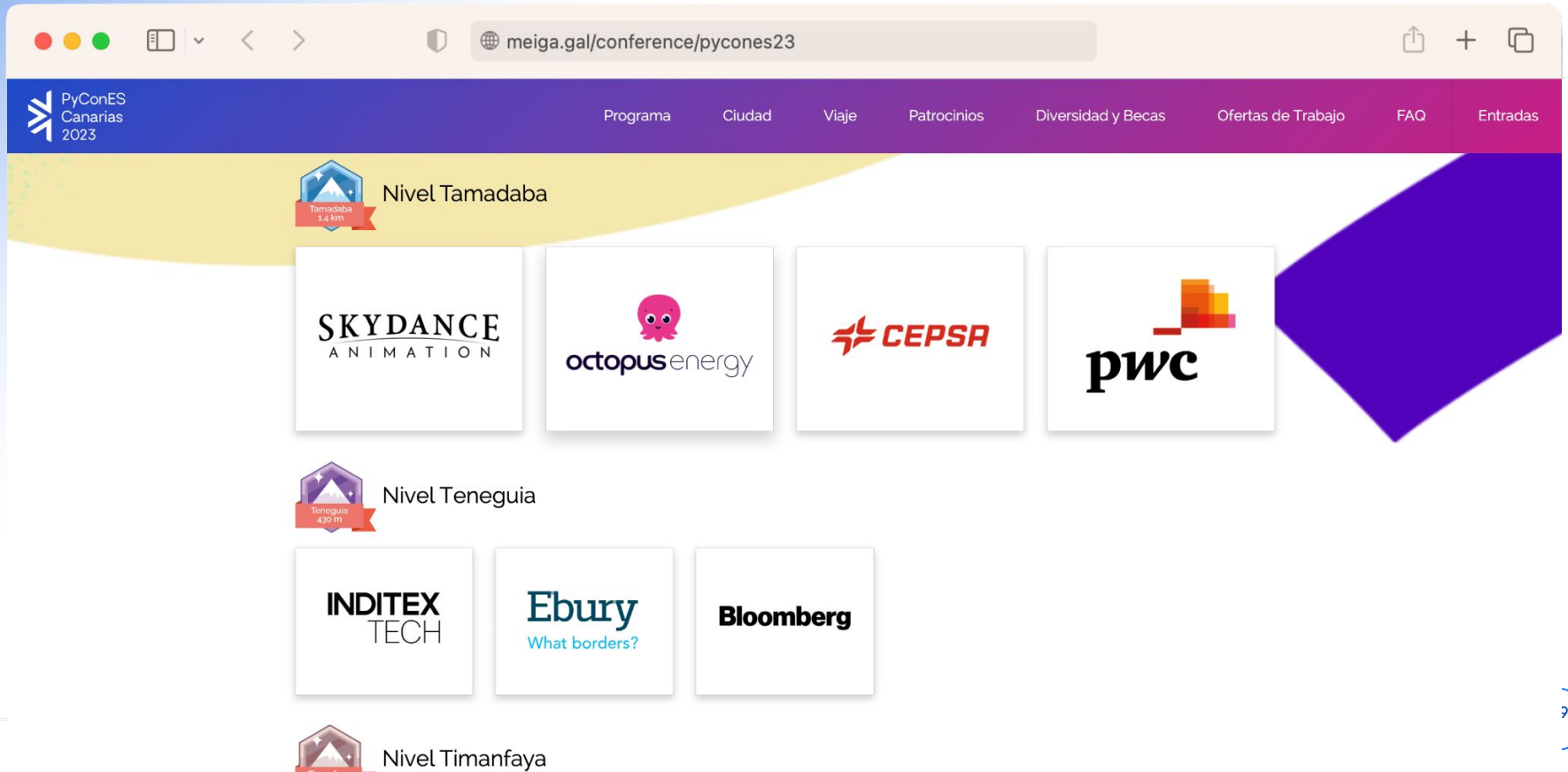


💻 `GET /acknowledge/pycones23` → Application

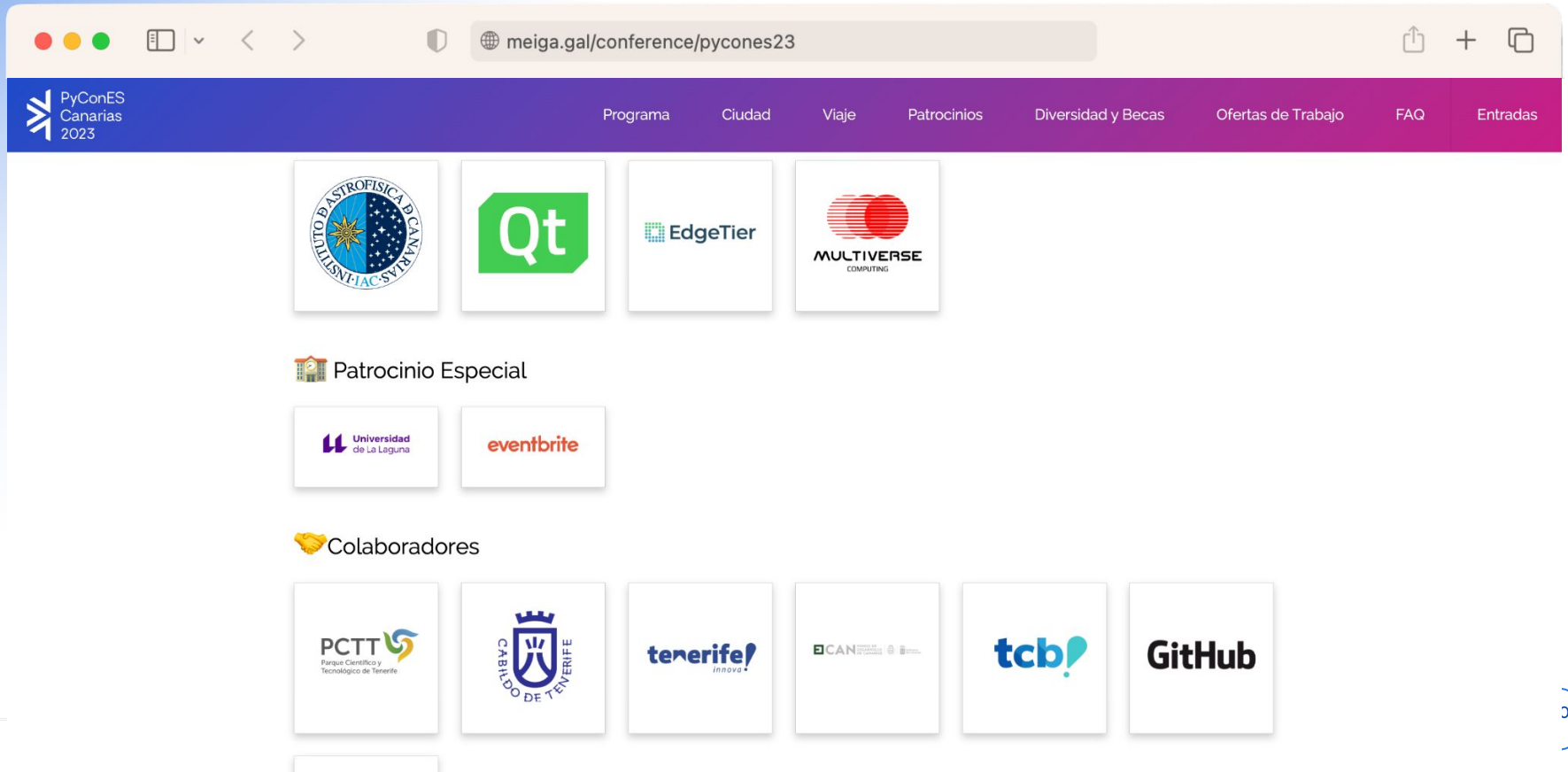


GET /acknowledge/pycones23

Application



💻 `GET /acknowledge/pycones23` → Application



## Malas noticias para los fans de Pitágoras

IFLSCIENCE

# Teorema de Pitágoras encontrado en una tablilla de arcilla 1.000 años más antigua que Pitágoras

*Es anterior a Pitágoras en más de 1.000 años.*

JAMES FELTON
Redactor sénior

La prueba está tallada en arcilla.
*Crédito de la imagen: Osama Shukir Muhammed Amin FRCP (Glasg) vía Wikimedia Commons ( CC BY-SA 4.0 ); girado, recortado*

**“La conclusión es ineludible. Los babilonios conocían la relación entre la longitud de la diagonal de un cuadrado y su lado: d=raíz cuadrada de 2”**, escribe el matemático Bruce Ratner en un artículo sobre el tema . “Este fue


En Alice estamos contratando

**Frontend | ML Engineer**


Grazas!

# Testing

```
class UserUpdater(UseCase):
    repository: Repository
    app_service: AppService

    def execute(self, user_id: Uuid) -> Result[bool, Error]:
        user = self.repository.retrive(user_id).unwrap_or_return()
        return self.app_service.execute(user)
```

# Testing

```python
class UserUpdater(UseCase):
    repository: Repository
    app_service: AppService

    def execute(self, user_id: Uuid) -> Result[bool, Error]:
        user = self.repository.retrive(user_id).unwrap_or_return()
        return self.app_service.execute(user)
```

```python
@pytest.mark.unit
class TestUserUpdater:
    mock_repository: Mock
    mock_app_service: Mock

    def setup_method(self) -> None:
        self.mock_repository = Mock(Repository)
        self.mock_app_service = Mock(AppService)

    def should_success(self) -> None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Success(True))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_success()
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()

    def should_fail_when_repository_not_found_user(self) -> None:
        self.mock_repository.retrieve = Mock(return_value=Failure(UserNotFound()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_failure(value_is_instance_of=UserNotFound)
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_not_called()

    def should_fail_when_app_service_error(self) -> None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Failure(AppServiceError()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_failure(value_is_instance_of=AppServiceError())
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()
```

# Testing

```python
class UserUpdater(UseCase):
    repository: Repository
    app_service: AppService

    def execute(self, user_id: Uuid) -> Result[bool, Error]:
        user = self.repository.retrive(user_id).unwrap_or_return()
        return self.app_service.execute(user)
```

Arrange: Configuración del  test.
Act: Acción a testear.
Assert: Comprobaciones necesarias.

```python
@pytest.mark.unit
class TestUserUpdater:
    mock_repository: Mock
    mock_app_service: Mock

    def setup_method(self) -> None:
        self.mock_repository = Mock(Repository)
        self.mock_app_service = Mock(AppService)

    def should_success(self) -> None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Success(True))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_success()
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()

    def should_fail_when_repository_not_found_user(self) -> None:
        self.mock_repository.retrieve = Mock(return_value=Failure(UserNotFound()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_failure(value_is_instance_of=UserNotFound)
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_not_called()

    def should_fail_when_app_service_error(self) -> None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Failure(AppServiceError()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_failure(value_is_instance_of=AppServiceError())
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()
```

# Testing

```python
class UserUpdater(UseCase):
    repository: Repository
    app_service: AppService

    def execute(self, user_id: Uuid) -> Result[bool, Error]:
        user = self.repository.retrive(user_id).unwrap_or_return()
        return self.app_service.execute(user)
```

```python
@pytest.mark.unit
class TestUserUpdater:
    mock_repository: Mock
    mock_app_service: Mock

    def setup_method(self) -> None:
        self.mock_repository = Mock(Repository)
        self.mock_app_service = Mock(AppService)

    def should_success(self) -> None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Success(True))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_success()
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()

    def should_fail_when_repository_not_found_user(self) -> None:
        self.mock_repository.retrieve = Mock(return_value=Failure(UserNotFound()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_failure(value_is_instance_of=UserNotFound)
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_not_called()

    def should_fail_when_app_service_error(self) -> None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Failure(AppServiceError()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_failure(value_is_instance_of=AppServiceError())
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()
```

# Testing

```python
class UserUpdater(UseCase):
    repository: Repository
    app_service: AppService

    def execute(self, user_id: Uuid) -> Result[bool, Error]:
        user = self.repository.retrive(user_id).unwrap_or_return()
        return self.app_service.execute(user)
```

```python
@pytest.mark.unit
class TestUserUpdater:
    mock_repository: Mock
    mock_app_service: Mock

    def setup_method(self) -> None:
        self.mock_repository = Mock(Repository)
        self.mock_app_service = Mock(AppService)

    def should_success(self) -> None:
```

```python
def should_success(self) -> None:
    # Arrange
    self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
    self.mock_app_service.execute = Mock(return_value=Success(True))

    # Act
    use_case = UserUpdater(self.mock_repository, self.mock_app_service)
    result = use_case.execute(UuidMother.any())

    # Assert
    result.assert_success()
    self.mock_repository.retrieve.assert_called_once()
    self.mock_app_service.execute.assert_called_once()
```

```python
        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_failure(value_is_instance_of=AppServiceError())
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()
```

# Testing

```python
class UserUpdater(UseCase):
    repository: Repository
    app_service: AppService

    def execute(self, user_id: Uuid) -> Result[bool, Error]:
        user = self.repository.retrive(user_id).unwrap_or_return()
        return self.app_service.execute(user)
```

```python
@pytest.mark.unit
class TestUserUpdater:
    mock_repository: Mock
    mock_app_service: Mock

    def setup_method(self) -> None:
        self.mock_repository = Mock(Repository)
        self.mock_app_service = Mock(AppService)

    def should_success(self) -> None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Success(True))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_success()
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()

    def should_fail_when_repository_not_found_user(self) -> None:
        self.mock_repository.retrieve = Mock(return_value=Failure(UserNotFound()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_failure(value_is_instance_of=UserNotFound)
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_not_called()

    def should_fail_when_app_service_error(self) -> None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Failure(AppServiceError()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_failure(value_is_instance_of=AppServiceError())
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()
```

# Testing

```python
class UserUpdater(UseCase):
    repository: Repository
    app_service: AppService

    def execute(self, user_id: Uuid) -> Result[bool, Error]:
        user = self.repository.retrive(user_id).unwrap_or_return()
        return self.app_service.execute(user)
```

```python
@pytest.mark.unit
class TestUserUpdater:
    mock_repository: Mock
    mock_app_service: Mock

    def setup_method(self) -> None:
        self.mock_repository = Mock(Repository)
        self.mock_app_service = Mock(AppService)

    def should_success(self) -> None:
```

```python
def should_fail_when_repository_not_found_user(self) -> None:
    # Arrange
    self.mock_repository.retrieve = Mock(return_value=Failure(UserNotFound()))

    # Act
    use_case = UserUpdater(self.mock_repository, self.mock_app_service)
    result = use_case.execute(UuidMother.any())

    # Assert
    result.assert_failure(value_is_instance_of=UserNotFound)
    self.mock_repository.retrieve.assert_called_once()
    self.mock_app_service.execute.assert_not_called()
```

```python
        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_failure(value_is_instance_of=AppServiceError())
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()
```

# Testing

```python
class UserUpdater(UseCase):
    repository: Repository
    app_service: AppService

    def execute(self, user_id: Uuid) -> Result[bool, Error]:
        user = self.repository.retrive(user_id).unwrap_or_return()
        return self.app_service.execute(user)
```

```python
@pytest.mark.unit
class TestUserUpdater:
    mock_repository: Mock
    mock_app_service: Mock

    def setup_method(self) -> None:
        self.mock_repository = Mock(Repository)
        self.mock_app_service = Mock(AppService)

    def should_success(self) -> None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Success(True))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_success()
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()

    def should_fail_when_repository_not_found_user(self) -> None:
        self.mock_repository.retrieve = Mock(return_value=Failure(UserNotFound()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_failure(value_is_instance_of=UserNotFound)
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_not_called()

    def should_fail_when_app_service_error(self) -> None:
        self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
        self.mock_app_service.execute = Mock(return_value=Failure(AppServiceError()))

        use_case = UserUpdater(self.mock_repository, self.mock_app_service)
        result = use_case.execute(UuidMother.any())

        result.assert_failure(value_is_instance_of=AppServiceError())
        self.mock_repository.retrieve.assert_called_once()
        self.mock_app_service.execute.assert_called_once()
```

# Testing

```python
class UserUpdater(UseCase):
    repository: Repository
    app_service: AppService

    def execute(self, user_id: Uuid) -> Result[bool, Error]:
        user = self.repository.retrive(user_id).unwrap_or_return()
        return self.app_service.execute(user)
```

```python
@pytest.mark.unit
class TestUserUpdater:
    mock_repository: Mock
    mock_app_service: Mock

    def setup_method(self) -> None:
        self.mock_repository = Mock(Repository)
        self.mock_app_service = Mock(AppService)
```

```python
def should_fail_when_app_service_error(self) -> None:
    # Arrange
    self.mock_repository.retrieve = Mock(return_value=Success(UserMother.any()))
    self.mock_app_service.execute = Mock(return_value=Failure(AppServiceError()))

    # Act
    use_case = UserUpdater(self.mock_repository, self.mock_app_service)
    result = use_case.execute(UuidMother.any())

    # Assert
    result.assert_failure(value_is_instance_of=AppServiceError())
    self.mock_repository.retrieve.assert_called_once()
    self.mock_app_service.execute.assert_called_once()
```