Python task

Dear applicant, below you can find the assignments for this task.
Take your time to read the instructions carefully.
You will have one hour to complete the assignments and submit the code.

Remarks:

1. You are allowed to use the internet.
2. Don’t forget to submit the tasks with valid dependencies (pip requirements).
3. Make sure that the code is tested before submitting it and provide valid usage examples.
4. Use Python 3.7 and PyCharm, make sure to conform to PEP8 conventions.
5. Upload your code to a public git repository, with a short README file on how to run your code.

Magic List

Create a Python class that implements a simplified list by skipping boundary checks when possible.

While regular lists behave like:

```
>>> a = list()
>>> a[0] = 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
```

Our magic list should allow:

```
>>> a = MagicList()
>>> a[0] = 5
>>> print(a)
[5]
```

The Magic List should also support initializing assigned types when *cls_type* is provided to its constructor:

```
@dataclass
class Person:
  age: int = 1

>>> a = MagicList(cls_type=Person)
>>> a[0].age = 5
>>> print(a)
[Person(age=5)]
```

The list should also enforce its indexes continuity:

```
>>> a = MagicList(cls_type=Person)
>>> a[1].age = 5
Traceback (most recent call last):
File "<stdin>", line 2, in <module>
IndexError: list index out of range
```

API

Create a Python web-server implemented with Sanic.

1. Create a login method for a user (You can store the user in a local file - username and password), and implement a JWT authentication mechanism (You can use any JWT python package you want).
2. The web-server should accept input data to a route with a POST method and return a normalized version of it.

The input would be a JSON that looks like:

```
[
        {
                "name": "device",
                "strVal": "iPhone",
                "metadata": "not interesting"
        },
        {
                "name": "isAuthorized",
                "boolVal": "false",
                "lastSeen": "not interesting"
        }
]
```

The output result should look like:

```
{
        "device": "iPhone",
        "isAuthorized": "false"
}
```

Normalization logic:

You should rely on the assumption that the field "name" will **always** exist and represents the key in the normalized output dict. Another assumption is that the field that represents the value of that key would contain the string "**val**". Other fields that don't participate in the result (like metadata, lastSeen, or any other unrelated value) should be omitted.

- the conversion should be done using a **one-liner**.
- The post route should be allowed to a logged-in user **ONLY**.

SERVERLESS

Deploy the same route you used for conversion in the sanic api (2), to AWS using the Serverless framework.
(For this part ignore user authentication)