Guida sviluppo backend

February 22, 2025

Contents

1 Introduzione

Questa guida contiene le informazioni legate alle norme di sviluppo per la creazione del backend del progetto sui giochi di ruolo. Nello specifico, verrà spiegato come creare un modulo in python e come integrarlo all'interno di un proprio progetto.
La necessita' di uno sviluppo modulare risiede nel rischio di conflitti legati all'unione di più file, nello specifico tutti i conflitti legati alla ripetizione di nomi per funzioni e/o classi.

Sono inoltre specificate alcune regole per lo sviluppo dei moduli e del loro impiego nel progetto finale.

2 Come creare un modulo in Python

Come anticipato nell'introduzione, per garantire l'assenza di conflitti verranno utilizzati i moduli python, ma come si crea un modulo?

2.1 Definizioni: modulo nel contesto di python

Per modulo si intende un insieme di funzioni o classi raccolte in un unico gruppo. Parlando in termini più spicci, un modulo in python è una libreria.

Pertanto, quello che si andra' a fare e' creare tante librerie che saranno d'appoggio al server principale.

2.2 Creazione di un modulo

Quando si crea un modulo, la prima cosa che si deve fare e' creare una cartella, il cui nome sara' lo stesso del modulo (se la cartella si chiama "Pippo", il modulo si chiamera' "Pippo"). La cartella conterra' principalmente due file: *__init__.py*, che serve all'interprete di python per capire che la cartella in questione e' un modulo, e *script.py*, che conterra' tutte le funzioni del modulo.

2.2.1 script.py

***script.py*, come detto sopra, contiene tutte le funzioni o classi del modulo. Al suo interno si possono importare altri moduli per utilizzarne le funzionalita' o semplificare determinate operazioni (e.g.: importo openpyxl per creare un modulo che lavora sui file excel e svolgo certe istruzioni dentro *script.py* per semplificare il programma dove il nuovo modulo verra' utilizzato).**

2.2.2 __init.py__

***__init.py__* serve a rendere accessibili le funzioni o classi contenute in *script.py*. Per farlo, *__init.py__* contiene un import delle funzioni o classi di *script.py* che potranno essere utilizzate quando verra' importato il modulo (alcune funzioni o classi di *script.py* potrebbero fare uso di funzioni o classi di supporto modellate per svolgere compiti talmente specifici da risultare inutili altrove).**

2.3 Esempio di modulo

Prendiamo in considerazione un modulo che deve operare sui file excel. Le operazioni principali su un file excel devono essere apertura, lettura, scrittura e salvataggio. Il file *__init.py__* conterra' quindi gli import di queste 4 funzioni.

__init.py__

```
from .script import open_file, close_file, read_cell, write_cell
# nel caso si debbano importare tutte le funzioni,
# si puo' scrivere "from .script import *"
```

Nel file *script.py*, invece, andremo a definire le 4 funzioni, utilizzando *openpyxl* come modulo di supporto

script.py

```
    import openpyxl as op

    def open_file(path):
        """
        loads a workbook from specified path and returns it
        """
        return op.load_workbook(path)


    def close_file(workbook, path = None):
        """
        saves and closes a workbook.
        if path is not specified, the function raises an exception
        """
        if path != None:
            workbook.save(path)
            workbook.close()
        else:
            raise Exception("the path was not specified")


    def read_cell(sheet, row, column)
        """
        returns the value of the cell at coordinates (row, column)
        from the specified sheet
        """
        return sheet.cell(row=row, column=column).value


    def write_cell(sheet, row, column, new_val)
        """
        writes the specified value in the cell at coordinates
        (row, column) from specified sheet
        """
        sheet.cell(row=row, column=column).value = new_val
```

Adesso sai tutto quello che c'è da sapere per creare un modulo in python.

3 Integrare un modulo custom in Python

Ora che abbiamo creato un modulo per importarlo bastano due semplici passaggi:
1) Copia la cartella del tuo modulo nella cartella del progetto

Percorso

```
    progetto
     |
     |___ venv
     |___ modulo
     |___ main.py
```

2) In main.py aggiungere la linea "import modulo"

main.py

```
# resto degli import
import modulo

#programma
```

Adesso il modulo e' pronto per essere utilizzato nel tuo programma.

4 Regole generali per il progetto

4.1 URL per le chiamate REST

Gli URL a cui i client faranno le richieste devono avere un prefisso identificativo del gruppo (e.g.: il gruppo 1 userà il prefisso "g1", il gruppo 2 "g2", il gruppo 3 "g3" e cosi' via per tutti i gruppi).
Il prefisso impiegato dovra' essere separato tramite underscore dal resto dell'URL (e.g.: "/g1_getnome")

4.2 Nome dei moduli

Il nome dei moduli seguira' una regola simile: il modulo della missione 1 potrebbe chiamarsi "mis1", il modulo della missione 2 "mis2" e cosi' via.

4.3 Contenuti dei moduli

I moduli impiegati nel progetto dovranno contenere
OBBLIGATORIAMENTE le funzioni *check_get* e *check_post*, come sotto implementate.
Inoltre, i moduli conterranno tutte le altre funzioni o classi necessarie per il funzionamento della missione o della porzione di sito assegnata.

script.py

```
def check_get(path):
    # codice base per il controllo di un url
    if path.endswith("url/esempio_1"):
        return funzione_a()


def check_post(path, data):
    if path.endswith("url/esempio_2"):
        return funzione_b(data)

# altre funzioni
```

***check_get* e *check_post* sono le uniche funzioni che possono essere utilizzate del modulo creato. *__init.py__* sara' quindi:**

__init.py__

```
from .script import check_get, check_post
```

4.4 Utilizzo dei moduli

I moduli importati dovranno essere utilizzati solo nelle funzioni *check_get* e *check_post* del server, come sotto riportato

main.py

```
#altri import
import mis1
import mis2
.
.
.
import misn

#implementazione del programma

def check_get(path):
    if path.startswith("/g1_"):
        return mis1.check_get(path)
    if path.startswith("/g2_"):
        return mis2.check_get(path)
    .
    .
    .
    if path.startswith("/gn_"):
        return misn.check_get(path)


def check_post(path, data):
    if path.startswith("/g1_"):
        return mis1.check_post(path, data)
    if path.startswith("/g2_"):
        return mis2.check_post(path, data)
    .
    .
    .
    if path.startswith("/gn_"):
        return misn.check_post(path, data)
```