**Употреба на Singleton патернот во кодот**

*Singleton патернот во кодот е искористен за да обезбеди дека постои само една инстанца на `****aiohttp.ClientSession****` во python скриптата именувана* ***test.py****. Овој патерн е клучен за правилно управување со ресурси и за избегнување на потенцијални грешки.*

**Предности на Singleton патернот**

1. Ефикасно испоработување на ресурси

- Се избегнува креирање на повеќе инстанци на `ClientSession`, што би можело да потроши повеќе меморија и мрежни ресурси.

2. Централизирано управување со сесиите

- Употребата е унифицирана на секаде во кодот, што го прави кодот доста по едноставен.

**Функционирање на кодот**

1. Креирање на инстанца:

@staticmethod

def get\_instance():

if SingletonSession.\_instance is None:

SingletonSession.\_instance = aiohttp.ClientSession()

return SingletonSession.\_instanc

Доколку `\_instance` е `None`, се креира нова `ClientSession` и се чува во `\_instance`.

Подоцна секој повик на `get\_instance` го враќа истиот инстанц.

2. Затворање на сесијата:

@staticmethod

async def close\_instance():

if SingletonSession.\_instance is not None:

await SingletonSession.\_instance.close()

SingletonSession.\_instance = None

Ова обезбедува дека сесијата е правилно затворена и дека ресурсите се ослободени.

3. Употреба во кодот:

session = SingletonSession.get\_instance()

#Користење на сесијата

tasks = []

for company in companies:

date\_ranges = get\_date\_ranges\_for\_update(company, "results.csv")

tasks.append(fetch\_missing\_data(session, company, date\_ranges))

all\_data\_batches = await asyncio.gather(\*tasks)

# Затворање на сесијата

await SingletonSession.close\_instance()