

**Basic Concept of Locust**:

🡪Locust is easy to use, event based, scriptable, scalable, extendable and distributed performance testing framework.

🡪 Use this URL for practice,

<http://www.automationpractice.pl/index.php>

🡪 We will hatch users that will do certain activity on the application in use and locust will provide us with out of the box reports of REST/API calls.

🡪 We can create client – server architecture in python with locust where we can distribute our load and write test scenarios.

🡪 Locust uses lightweight greenevent which helps us spawn multiple users on a single machine.

**Setup Development Environment**:

🡪 Need Python and PIP

🡪 pip3 install locust.

Sample Script:

from locust import User, task, between

class MyUser(User):

    wait\_time = between(1, 2)

    @task

    def my\_task1(self):

        print("executing my\_task1")

    @task

    def my\_task2(self):

        print("executing my\_task2")

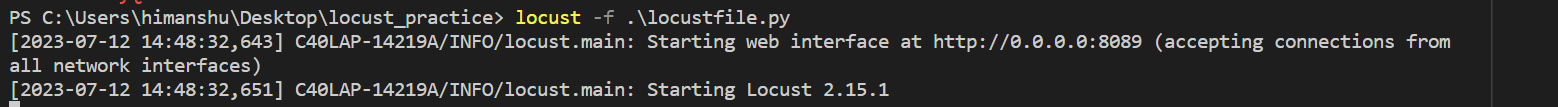
🡪 MyUser class is inheriting from User class, which comes from locust library. *Used to simulate a real user*.

🡪 Inside @task which is a decorator we define a function which is basically a *task that user will perform on the application*.

🡪 wait\_time is specifying the time between *two consecutive task executions*. Here (1, 2) means after executing task#1 it will wait between 1 to 2 second to start task#2.

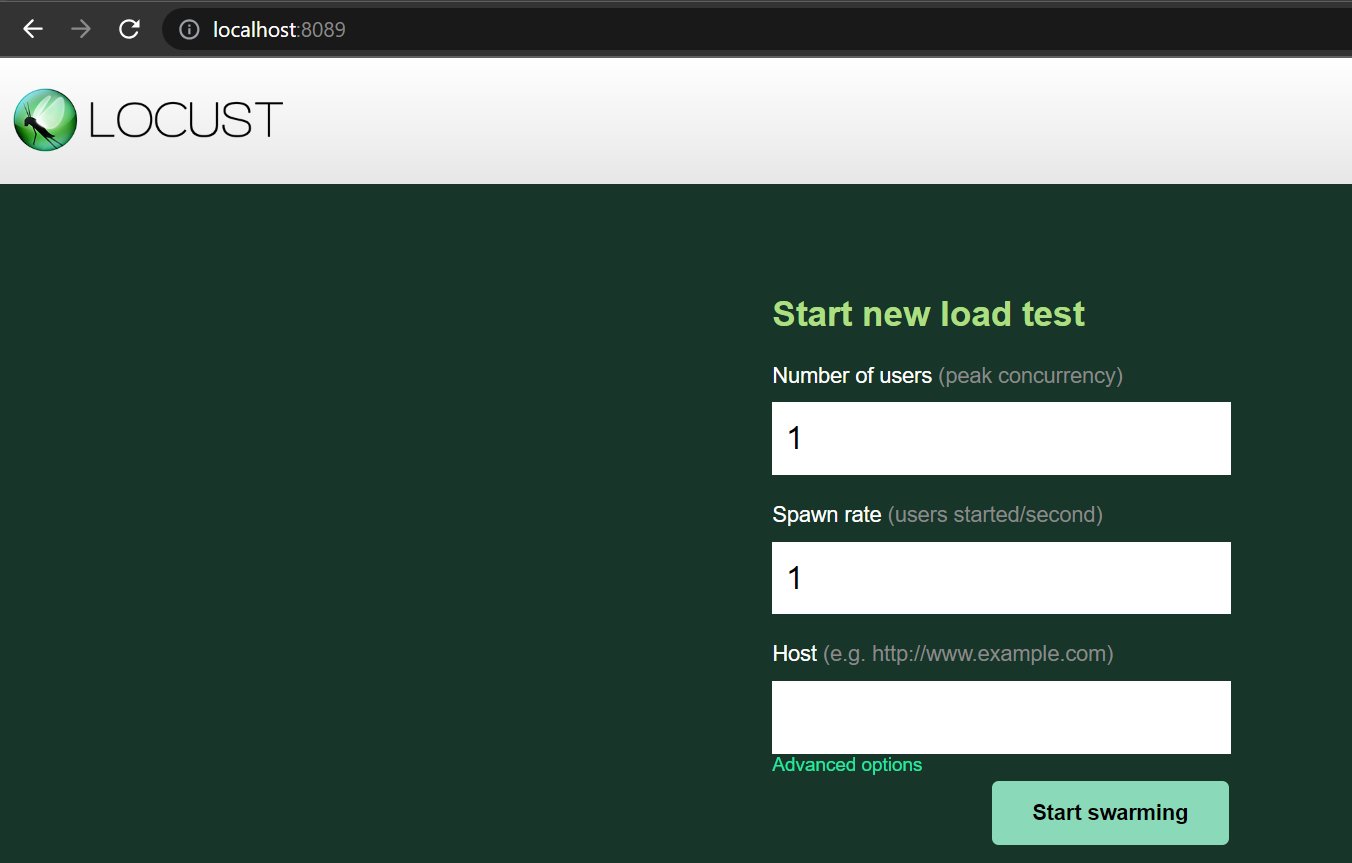
To run locust,

locust -f .\locustfile.py ,

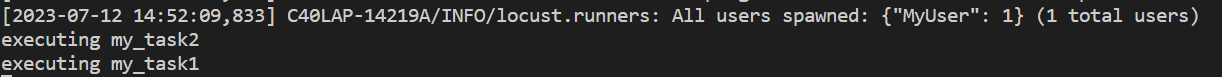


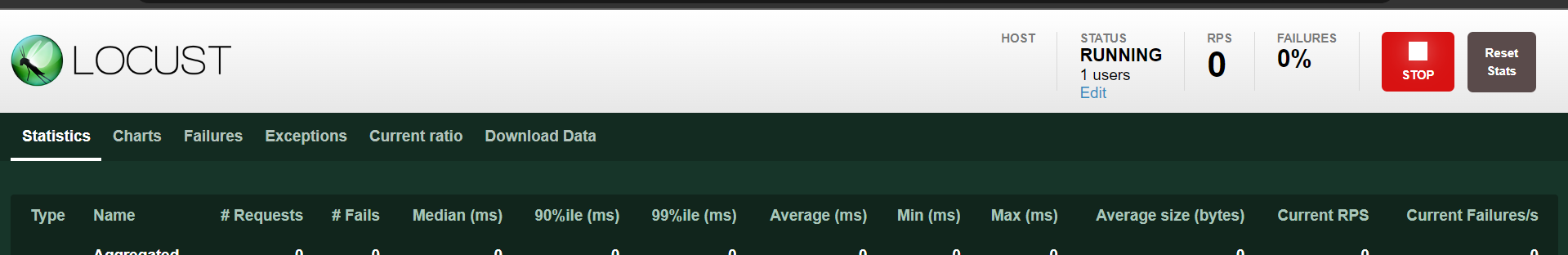
Go to <http://localhost:8089/>

You will see, our locust web UI will appear,

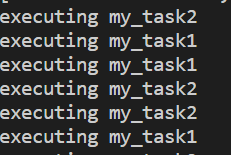


Here if we do not give any REST API in Host, we will start seeing our print statements in console.

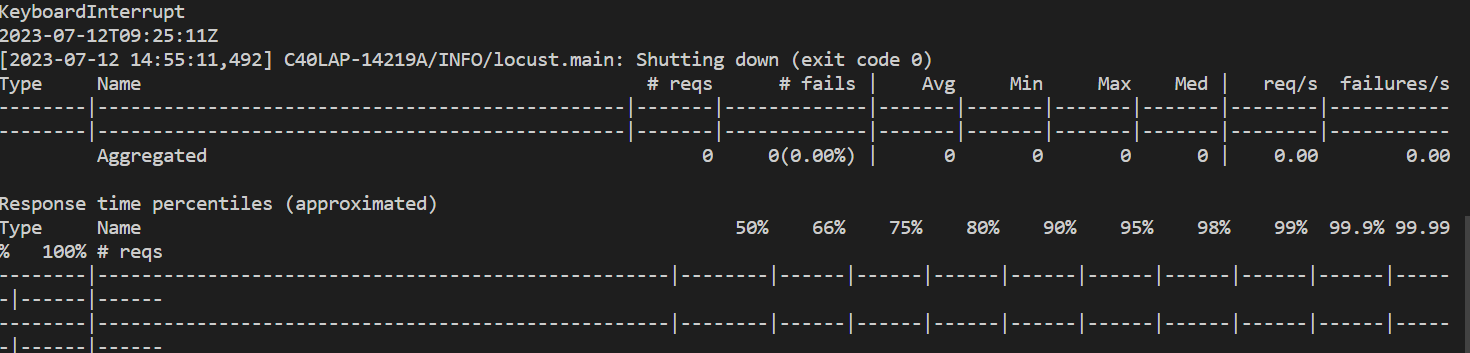




Only one user is running and its executing both tasks but in random manner(not sequential).



Press ctrl+c to stop execution.



We do not see any response here, because we are not hitting any actual API.

**Explanation of User, TaskSet and Sequential TaskSet**:

1. *Basic User and TaskSet class*:

class SearchProduct(TaskSet):

    @task

    def search\_men\_products(self):

        print("Searching men products")

    @task

    def search\_kids\_products(self):

        print("Searching kids products")

class MyUser(User):

    wait\_time = between(1, 2)

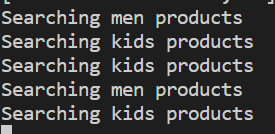
    tasks = [SearchProduct]

🡪 Here, our *SearchProduct* class is inheriting from TaskSet base class, which is how we should define our task in real projects.

🡪 Then in our user class we can set tasks attribute to this *SearchProduct* class as a ***List***(we can define multiple TaskSet classes in this [ ] list).

Note: We can also give number of user and spawn rate from command line

locust -f .\locustfile.py -u 1 -r 1



There is no sequential way of execution and it does not stop.

Problem here is randomness and no priority set for one task over the other.

1. *TaskSet class with Weight*:

class SearchProduct(TaskSet):

    @task(4)

    def search\_men\_products(self):

        print("Searching men products")

    @task(1)

    def search\_kids\_products(self):

        print("Searching kids products")

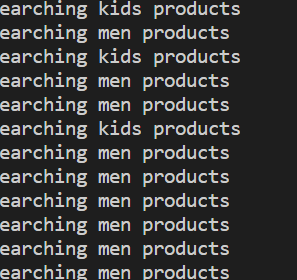
class MyUser(User):

    wait\_time = between(1, 2)

    tasks = [SearchProduct]

Notice here I have given an attribute to task decorator. First task has 4 and second one has 1. This argument is called *weight*.

*It means User has 4 times more probability to pick up the task with @task(4) weight then the other task*.

🡨 It is executing task with more weight more frequently.

In real life scenario of an e-commerce store , the task of searching for a product is more frequent then adding an item in shopping cart.

1. *Sequential TaskSet class*:

class SearchProduct(SequentialTaskSet):

    @task()

    def search\_men\_products(self):

        print("Searching men products")

    @task()

    def search\_kids\_products(self):

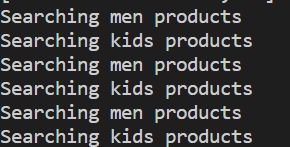
        print("Searching kids products")

class MyUser(User):

    wait\_time = between(1, 2)

    tasks = [SearchProduct]

To remove the randomness of the task execution, we can introduce a SequentialTaskSet class.



Now it will maintain its sequence of task execution.

1. *Multiple TaskSet Class*:

from locust import User, task, between, TaskSet, SequentialTaskSet

class SearchProduct(SequentialTaskSet):

    @task

    def search\_men\_products(self):

        print("Searching men products")

    @task

    def search\_kids\_products(self):

        print("Searching kids products")

class ViewCart(SequentialTaskSet):

    @task

    def get\_cart\_items():

        print("get all cart items")

    @task

    def search\_cart\_item():

        print("Searching item from cart")

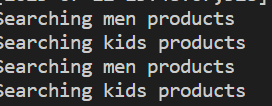
class MyUser(User):

    wait\_time = between(1, 2)

    tasks = [SearchProduct, ViewCart]

Here we have two Sequential TaskSet classes defined and in the user class we defined both of them in form of list.

Let us see what happens,



*Locust user is executing only one TaskSet class, It is not executing the other one, which is an issue for solving this we need to add an* ***interrupt***.

1. *TaskSet class with Interrupt*:

class SearchProduct(SequentialTaskSet):

    @task

    def search\_men\_products(self):

        print("Searching men products")

    @task

    def search\_kids\_products(self):

        print("Searching kids products")

    @task

    def exit\_task\_execution(self):

        self.interrupt() 🡪 Added an interrupt here

class ViewCart(SequentialTaskSet):

    @task

    def get\_cart\_items(self):

        print("get all cart items")

    @task

    def search\_cart\_item(self):

        print("Searching item from cart")

    @task

    def exit\_task\_execution(self):

        self.interrupt()

class MyUser(User):

    wait\_time = between(1, 2)

    tasks = [SearchProduct, ViewCart]

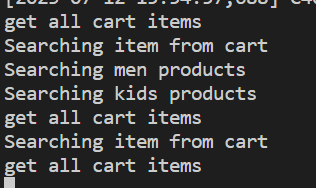
As you can see we have added another task in both our TaskSet,

    @task

    def exit\_task\_execution(self):

        self.interrupt()

it will be used to get out of the task loop and switch to another TaskSet.



So now it is picking up tasks from both the TaskSet randomly, issue fixed.

1. *Multiple TaskSet with weight(****closest to real world****)*:

We can define the weightage of TaskSet in our User class like this,

class MyUser(User):

    wait\_time = between(1, 2)

    tasks = {SearchProduct: 4, ViewCart: 1} 🡪 As a dictionary

Now probability of SearchProduct TaskSet execution is 4 times higher than ViewCart TaskSet.

**Explanation**

**on\_test\_start, on\_test\_stop, on\_start, on\_stop**:

Sometimes we need functionalities like opening our browser before starting test execution and closing it after it is complete.

1. *Executing only once before and after suite*:

These methods will not be inside TaskSet or User class but rather be at module level(means in file).

@events.test\_start.add\_listener

def on\_test\_start(\*\*kwargs):

    print(".................Initializing Load Test..................ON\_TEST\_START")

@events.test\_stop.add\_listener

def on\_test\_stop(\*\*kwargs):

    print("..................Load Test completed.......................ON\_TEST\_STOP")

Locust will create an event at the start using @events.test\_start.add\_listener where it will execute the underlying function and when the test is completed it will execute @events.test\_stop.add\_listener.

1. *Executing for each user locust hatch*:

class MyUser(User):

    def on\_start(self):

        print("MyUser : Hatching new User ...")

    def on\_stop(self):

        print("MyUser: Deleting User ...")

This on\_start methods is *going to execute when Locust will hatch the user*.

on\_stop method when all users completed their execution.

1. *Execute for each time for TaskSet execution*:

class SearchProduct(SequentialTaskSet):

    def on\_start(self):

        print("SearchProduct: Tasks execution Started")

    def on\_stop(self):

        print("SearchProduct: Tasks execution completed")

    @task

    def search\_men\_products(self):

        print("Searching men products")

    @task

    def search\_kids\_products(self):

        print("Searching kids products")

    @task

    def exit\_task\_execution(self):

        self.interrupt()

on\_start inside TaskSet will execute every time a user starts executing task in the task sets.

Complete code below…

from locust import User, task, between, TaskSet, SequentialTaskSet, events

@events.test\_start.add\_listener

def on\_test\_start(\*\*kwargs):

    print(".................Initializing Load Test..................ON\_TEST\_START")

@events.test\_stop.add\_listener

def on\_test\_stop(\*\*kwargs):

    print("..................Load Test completed.......................ON\_TEST\_STOP")

class SearchProduct(SequentialTaskSet):

    def on\_start(self):

        print("SearchProduct: Tasks execution Started")

    def on\_stop(self):

        print("SearchProduct: Tasks execution completed")

    @task

    def search\_men\_products(self):

        print("Searching men products")

    @task

    def search\_kids\_products(self):

        print("Searching kids products")

    @task

    def exit\_task\_execution(self):

        self.interrupt()

class MyUser(User):

    def on\_start(self):

        print("MyUser : Hatching new User ...")

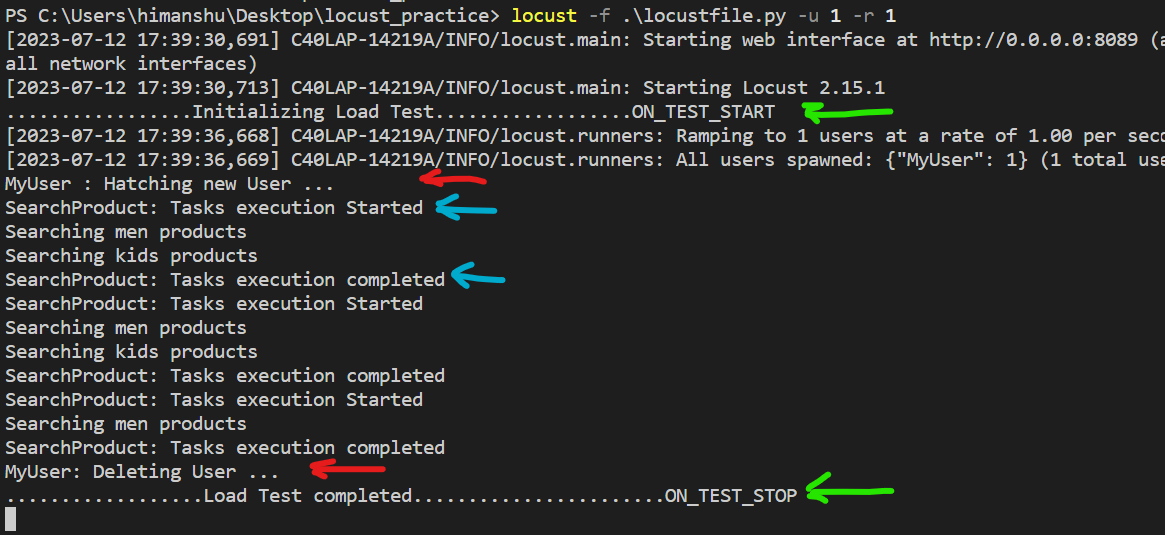
    def on\_stop(self):

        print("MyUser: Deleting User ...")

    wait\_time = between(1, 2)

    tasks = [SearchProduct]

Result:



🡪 As we can see even before hatching a new user, we got on\_test\_start .

🡪 Then our user hatched and afterwards our task execution started.

🡪 After stopping our execution we can see on\_stop methods in User class action.

**Rest execution using Locust HTTPUser class**:

We use --host command line parameter to define our host before swarming.

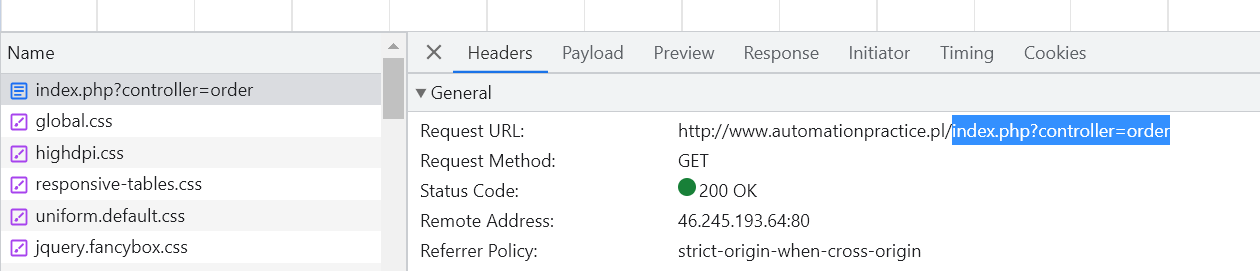
Note that host will remain fixed while we only provide endpoints in our task. For example our host is

<http://www.automationpractice.pl/>

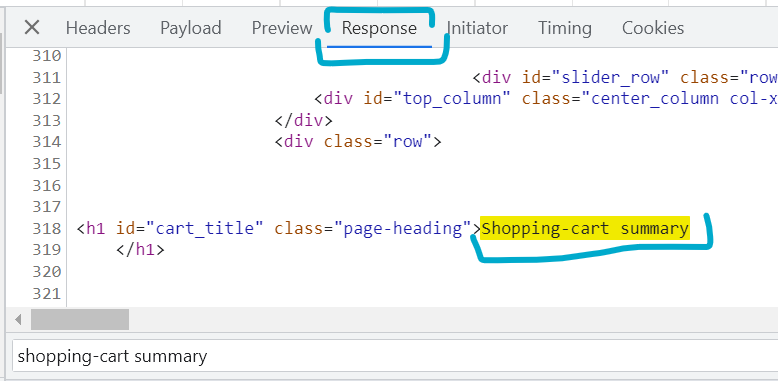
We will add this parameter from command line.

*Activity*:

Go to page and see all items in the cart



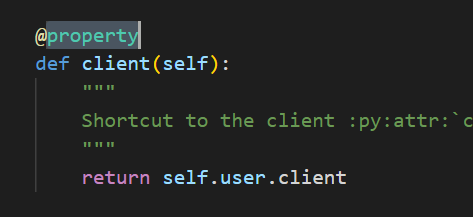
This is a GET API which we need to test. We will try to verify whether this Shopping-cart summary is present in this APIs response or not



*Prerequisites to know before executing this*,

🡪 Our Locust User will inherit from *HTTPUser* class instead of User class.

Note: *HTTPUser* class itself inherits from locust User class but Main thing we get from this HTTPUser class is ***client***(*which is not present in its parent class*).

🡨 *client* property

This client uses *session* which creates a *HTTP session* and with its help we can hit REST API calls.



With this client we can hit GET, POST all kinds of request.

🡪 This client property has a get method which takes URL “end point” and *catch\_response* = True which means I will catch the *response* and do tasks or validations with this response.

🡪 If *response. status\_code != 200* give response. failure (“ ”) otherwise response. success(“ ”)

🡪 Run this command to add host automatically,

locust -f .\locustfile.py -u 1 --host http://www.automationpractice.pl -r 1

Code:

from locust import User, task, between, TaskSet, SequentialTaskSet, events, HttpUser

class ViewCart(SequentialTaskSet):

    @task

    def get\_all\_cart\_item(self):

        with self.client.get(

            "/index.php?controller=order", catch\_response=True

        ) as response:

            if response.status\_code != 200:

                response.failure(

                    "Failed to get all cart items" + str(response.status\_code)

                )

            else:

                if "Shopping-cart summary" in response.text:

                    response.success()

                else:

                    response.failure("Failed to get cart items " + response.txt)

    @task

    def exit\_navigation(self):

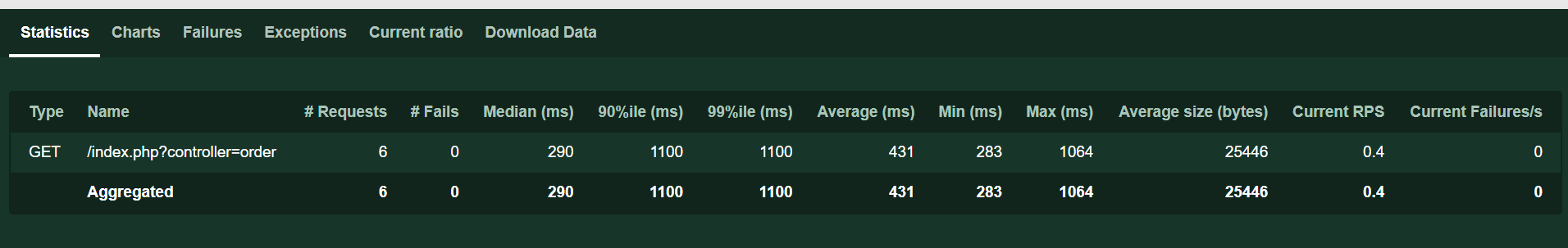
        self.interrupt()

class MyUser(HttpUser):

    wait\_time = between(1, 2)

    tasks = [ViewCart]

We can see we are hitting this end point,



Now we will fail it by giving wrong end point and see failures.

