Create a function to set up and tear down a gRPC server with configurable interceptors and thread pool workers.

Function Description: The function "_set_up " initializes and starts a gRPC server with optional interceptors, while "_tear_down" gracefully stops the server. The server is configured to handle gRPC requests concurrently using a thread pool.

Input:

- " interceptors " (Optional[List[grpc.ServerInterceptor]]): A list of gRPC server interceptors for request/response processing. Defaults to None.

- " server " (grpc.Server): The gRPC server instance to be stopped (for "_tear_down").

- Output:

- " _set_up " returns the initialized and started "grpc.Server" instance.

- " _tear_down" has no return value (stops the server).",

import grpc

from concurrent import futures

import futures

```
from concurrent import futures
from typing import List, Optional
def _set_up(interceptors: Optional[List[grpc.ServerInterceptor]] = None):
    server = grpc.server(
        futures.ThreadPoolExecutor(max_workers=2),
        interceptors=interceptors,
    )
    add_gRPCTestServiceServicer_to_server(TestService(), server)
    server.add_insecure_port("[::]:{}".format(PORT))
    server.start()
    return server

def _tear_down(server: grpc.Server):
    server.stop(None)

from concurrent import futures
from typing import List, Optional
def _set_up(interceptors: Optional
server = grpc.server(
    futures.ThreadPoolExecutor(
        interceptors=interceptors if i
    )
    server.start()
    return server

def _tear_down(server: grpc.Server)
server.stop(0)
```

"Instruction":
"Task Description:

```
def _set_up(interceptors: Optional[List[grpc.ServerInterceptor]] = None) -> grpc.Server:
    server = grpc.server(
      futures.ThreadPoolExecutor(max_workers=10),
      interceptors=interceptors if interceptors else []
    )
    server.start()
    return server

def _tear_down(server: grpc.Server) -> None:
    server.stop(o)
```