```
import os, shutil

base_dir = "/content/Day28_LoadTest_PilotPack"
results_dir = os.path.join(base_dir, "results")

# Remove old folders if they exist
if os.path.exists(base_dir):
    shutil.rmtree(base_dir)

# Create folder structure
os.makedirs(results_dir, exist_ok=True)

print("✅ Project folder created at", base_dir)
```

```
✅ Project folder created at /content/Day28_LoadTest_PilotPack
```

```
import os, shutil

base_dir = "/content/Day28_LoadTest_PilotPack"
results_dir = os.path.join(base_dir, "results")

# Remove old folders if they exist
if os.path.exists(base_dir):
    shutil.rmtree(base_dir)

# Create folder structure
os.makedirs(results_dir, exist_ok=True)

print("✅ Project folder created at", base_dir)
```

```
✅ Project folder created at /content/Day28_LoadTest_PilotPack
```

```
load_test_code = """#!/usr/bin/env python3
import os, asyncio, aiohttp, time, csv
```

```
API_URL = os.getenv("API_URL", "https://postman-echo.com/post")
SESSIONS = int(os.getenv("SESSIONS", 50))
CONCURRENCY = int(os.getenv("CONCURRENCY", 10))
MESSAGES_PER_SESSION = int(os.getenv("MESSAGES_PER_SESSION", 5))
THINK_TIME = float(os.getenv("THINK_TIME", 0.1))
OUTPUT_DIR = os.getenv("OUTPUT_DIR", "/content/Day28_LoadTest_PilotPack/results")

os.makedirs(OUTPUT_DIR, exist_ok=True)

def create_payload(session_id, msg_id):
    return {"session": session_id, "message_id": msg_id, "text": "Hello"}

async def send_request(session, payload):
    t0 = time.time()
    try:
        async with session.post(API_URL, json=payload) as resp:
            await resp.text()
            return time.time() - t0, resp.status
    except:
        return time.time() - t0, None

async def run_session(session_id):
    async with aiohttp.ClientSession() as s:
        results = []
        for m in range(MESSAGES_PER_SESSION):
            payload = create_payload(session_id, m)
            elapsed, status = await send_request(s, payload)
            results.append((elapsed, status))
            await asyncio.sleep(THINK_TIME)
        return results

async def main():
    tasks = [run_session(i) for i in range(SESSIONS)]
    all_results = await asyncio.gather(*tasks)
    rows = [(sid, i, r[0], r[1]) for sid, session in enumerate(all_results) for i, r in enumerate(session)]
    csv_file = os.path.join(OUTPUT_DIR, "results.csv")
    with open(csv_file, "w", newline="") as f:
        w = csv.writer(f)
```

```
        w.writerow(["session_id","msg_id","elapsed_sec","status"])
        w.writerows(rows)
    print("✅ Results saved to", csv_file)

if __name__ == "__main__":
    asyncio.run(main())
"""

# Save the script
load_test_path = "/content/Day28_LoadTest_PilotPack/load_test.py"
with open(load_test_path, "w") as f:
    f.write(load_test_code)

print("✅ load_test.py created")
```

```
✅ load_test.py created
```

```
import subprocess
import sys

# Create requirements.txt
req_path = "/content/Day28_LoadTest_PilotPack/requirements.txt"
with open(req_path, "w") as f:
    f.write("aiohttp\n")

print("✅ requirements.txt created")

# Install dependencies safely in Colab
subprocess.check_call([sys.executable, "-m", "pip", "install", "-r", req_path])
print("✅ Dependencies installed")
```

```
✅ requirements.txt created
✅ Dependencies installed
```

```
import os

os.environ["API_URL"] = "https://postman-echo.com/post"
os.environ["SESSIONS"] = "100"  # total concurrent chat sessions (50-100)
os.environ["CONCURRENCY"] = "10"  # messages at a time
os.environ["MESSAGES_PER_SESSION"] = "5"
os.environ["THINK_TIME"] = "0.1"  # seconds delay between messages
os.environ["OUTPUT_DIR"] = "/content/Day28_LoadTest_PilotPack/results"

print("✅ Environment variables set")
```

```
✅ Environment variables set
```

```
import subprocess
import sys

load_test_path = "/content/Day28_LoadTest_PilotPack/load_test.py"

# Run the load test
subprocess.check_call([sys.executable, load_test_path])
print("✅ Load test completed")
```

```
✅ Load test completed
```

```
import pandas as pd

# Load CSV results
csv_file = "/content/Day28_LoadTest_PilotPack/results/results.csv"
df = pd.read_csv(csv_file)

# Metrics
total_requests = len(df)
successful_requests = len(df[df['status'] == 200])
success_rate = successful_requests / total_requests * 100
avg_response = df['elapsed_sec'].mean()
min_response = df['elapsed_sec'].min()
```

```
max_response = df['elapsed_sec'].max()

print(f"✅ Total requests: {total_requests}")
print(f"✅ Successful (200): {successful_requests} ({success_rate:.2f}%)")
print(f"✅ Avg response time: {avg_response:.3f} sec")
print(f"✅ Min response time: {min_response:.3f} sec")
print(f"✅ Max response time: {max_response:.3f} sec")
```

```
✅ Total requests: 500
✅ Successful (200): 500 (100.00%)
✅ Avg response time: 0.242 sec
✅ Min response time: 0.062 sec
✅ Max response time: 0.995 sec
```