Objective

The objective of this notebook is to generate software performance testing solution using Gemini - Google's Generative AI Model. Gemini supports multimodals, combining different types of data like text, code, audio, image and video.

Setting up

Install the required libraries

In [ ]:

```
# install gen ai library
!pip install -q -U google-generativeai
```

Import Libraries

In [ ]:

```
#import necessary libraries
import pathlib
import textwrap
import pandas as pd
import numpy as np
from pathlib import Path
import hashlib

import google.generativeai as genai

from IPython.display import display
from IPython.display import Markdown


def to_markdown(text):
  text = text.replace('•', '  *')
  return Markdown(textwrap.indent(text, '> ', predicate=lambda _: True))
```

In [ ]:

```
# mount google drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Setup API Key

In [ ]:

```
# Used to securely store your API key
from google.colab import userdata
```

In [ ]:

```
# Or use `os.getenv('GOOGLE_API_KEY')` to fetch an environment variable.
GOOGLE_API_KEY=userdata.get('GOOGLE_API_KEY')
genai.configure(api_key=GOOGLE_API_KEY)
```

Model Selection

List Models

In [ ]:

```
# List Google AI Gemini Models
for m in genai.list_models():
  if 'generateContent' in m.supported_generation_methods:
    print(m.name)
```

```
models/gemini-1.0-pro
models/gemini-1.0-pro-001
models/gemini-1.0-pro-latest
models/gemini-1.0-pro-vision-latest
models/gemini-1.5-pro-latest
models/gemini-pro
models/gemini-pro-vision
```

Model Config

In [ ]:

```
# Generation config
generation_config = {
  "temperature": 1,
  "top_p": 0.5,
  "top_k": 0,
  "max_output_tokens": 7000,
}

# Safety config
safety_settings = [
  {
    "category": "HARM_CATEGORY_HARASSMENT",
    "threshold": "BLOCK_MEDIUM_AND_ABOVE"
  },
  {
    "category": "HARM_CATEGORY_HATE_SPEECH",
    "threshold": "BLOCK_MEDIUM_AND_ABOVE"
  },
  {
    "category": "HARM_CATEGORY_SEXUALLY_EXPLICIT",
    "threshold": "BLOCK_MEDIUM_AND_ABOVE"
  },
  {
    "category": "HARM_CATEGORY_DANGEROUS_CONTENT",
    "threshold": "BLOCK_MEDIUM_AND_ABOVE"
  },
]
```

Model Creation

In [ ]:

```
# Create model with the selected model name and configs
model = genai.GenerativeModel(model_name='gemini-1.0-pro',
                              generation_config=generation_config,
                              safety_settings=safety_settings)
```

Use Cases

Selected "gemini-1.0-pro" Google AI model, is used to enhance Software Performance Testing. The main idea behind the 'Gen AI Performance Tester' app is to reduce the overall testing effort. This is possible by automating time-consuming activities involved in each testing phase.

Following five major processes in Software Testing Life Cycle [STLC] were identified as use cases.

1. Test Plan Creation

In [ ]:

```
# Generate content for the prompt and display the execution time
%%time
response1 = model.generate_content("Create a performance test plan for API load testing i
n Cloud")
```

```
CPU times: user 48.5 ms, sys: 11.7 ms, total: 60.1 ms
Wall time: 8.06 s
```

In [ ]:

```
# Print the response - test plan document
to_markdown(response1.text)
```

Out[ ]:

> **Performance Test Plan for API Load Testing in Cloud**
>
> **Objective:**
>
> - Evaluate the performance and scalability of an API under various load conditions.
>
> **Scope:**
>
> - This test plan covers load testing of the following API: [API Name]
> - The test will simulate realistic user traffic patterns.
>
> **Test Environment:**
>
> - Cloud platform: [Cloud Platform Name]
> - Region: [Region]
> - API endpoint: [API Endpoint URL]
> - Load generator: [Load Generator Tool]
>
> **Test Scenarios:**
>
> - **Scenario 1: Baseline Load**
>   - Number of concurrent users: 100
>   - Request rate: 10 requests per second
>   - Duration: 30 minutes
> - **Scenario 2: Peak Load**
>   - Number of concurrent users: 1000
>   - Request rate: 50 requests per second
>   - Duration: 30 minutes

- Duration: 30 minutes
- **Scenario 3: Stress Load**
    - Number of concurrent users: 2000
    - Request rate: 100 requests per second
    - Duration: 15 minutes

Test Metrics:

- **Response time (ms)**
- **Throughput (requests per second)**
- **Error rate (%)**
- **Resource utilization (CPU, memory)**

Test Procedure:

1. Configure the load generator with the test scenarios.
2. Start the load generator and monitor the test metrics.
3. Adjust the load parameters as needed to achieve the desired load conditions.
4. Collect and analyze the test results.

Acceptance Criteria:

- The API should handle the peak load scenario without significant performance degradation.
- The error rate should be below 1% under all load conditions.
- The resource utilization should not exceed 80% during the peak load scenario.

Reporting:

- A test report will be generated that includes the following:
    - Test scenarios
    - Test results
    - Analysis and recommendations

Timeline:

- Test preparation: 1 week
- Test execution: 1 day
- Test analysis and reporting: 1 week

Responsibilities:

- Test lead: [Test Lead Name]
- Load generator operator: [Load Generator Operator Name]
- API developer: [API Developer Name]

In [ ]:

```
# Print the prompt safety ratings
response1.candidates[0].safety_ratings
```

Out[ ]:

```
[category: HARM_CATEGORY_SEXUALLY_EXPLICIT
probability: NEGLIGIBLE
, category: HARM_CATEGORY_HATE_SPEECH
probability: NEGLIGIBLE
, category: HARM_CATEGORY_HARASSMENT
probability: NEGLIGIBLE
, category: HARM_CATEGORY_DANGEROUS_CONTENT
probability: NEGLIGIBLE
]
```

2. Test Data Generation

In [ ]:

```
# Generate content for the prompt and display the execution time
%%time
response2 = model.generate_content("Generate 50 test data of name,age, and city for load testing and display in csv format")
```

```
CPU times: user 33 ms, sys: 4.93 ms, total: 37.9 ms
Wall time: 6.06 s
```

In [ ]:

```
# Print the response - csv dataset
to_markdown(response2.text)
```

Out[ ]:

CSV Format:

```
csv
name,age,city
```

Test Data:

```
csv
John,25,New York
Jane,30,Los Angeles
Michael,28,Chicago
Sarah,22,San Francisco
David,35,Dallas
Emily,26,Seattle
Matthew,32,Houston
Jessica,24,Phoenix
Mark,31,Philadelphia
Ashley,27,San Diego
Brian,33,Boston
Lauren,23,Miami
James,36,Atlanta
Jennifer,29,Denver
Robert,34,Detroit
Elizabeth,21,Baltimore
William,37,Minneapolis
Michelle,25,Cleveland
Thomas,30,Columbus
Amanda,28,Indianapolis
Charles,35,Nashville
Maria,26,Memphis
Christopher,32,Charlotte
Victoria,24,Raleigh
Richard,31,Oklahoma City
Abigail,27,Portland
Joseph,33,Las Vegas
Nicole,23,Milwaukee
Daniel,36,Kansas City
Anna,29,Jacksonville
Patrick,34,Austin
Katherine,21,San Antonio
George,37,New Orleans
Sophia,25,Louisville
Anthony,30,Birmingham
Olivia,28,Memphis
Ethan,35,Nashville
Isabella,26,Charlotte
Benjamin,32,Raleigh
Ava,24,Oklahoma City
Alexander,31,Portland
```

```
Mia,27,Las Vegas
Elijah,33,Milwaukee
Madison,23,Kansas City
```

In [ ]:

```
# Print the prompt safety ratings
response2.candidates[0].safety_ratings
```

Out[ ]:

```
[category: HARM_CATEGORY_SEXUALLY_EXPLICIT
probability: NEGLIGIBLE
, category: HARM_CATEGORY_HATE_SPEECH
probability: NEGLIGIBLE
, category: HARM_CATEGORY_HARASSMENT
probability: NEGLIGIBLE
, category: HARM_CATEGORY_DANGEROUS_CONTENT
probability: NEGLIGIBLE
]
```

3. Test Script Generation

In [ ]:

```
# Generate content for the prompt and display the execution time
%%time
response3 = model.generate_content("Generate Gatling code to performance test a REST API
with 50 users for 30 minutes and 10 seconds think time")
```

```
CPU times: user 21.9 ms, sys: 3.45 ms, total: 25.4 ms
Wall time: 3.73 s
```

In [ ]:

```
# Print the response - Gatling script
to_markdown(response3.text)
```

Out[ ]:

```
gatling
import io.gatling.core.Predef._
import io.gatling.http.Predef._

class RestApiSimulation extends Simulation {

  val httpProtocol = http
    .baseUrl("http://localhost:8080")
    .acceptHeader("application/json")

  val scn = scenario("Rest API Test")
    .exec(
      http("Get all users")
        .get("/users")
        .check(status.is(200))
    )
    .pause(10)

  setUp(
    scn.inject(
      rampUsers(50) over (10 seconds),
      constantUsersPerSec(50) during (30 minutes)
    ).protocols(httpProtocol)
    ,
}
```

4. Test Docker Container Creation

In [ ]:

```
# Generate content for the prompt and display the execution time
%%time
response4 = model.generate_content("Generate a Docker script to create a server with Lin
ux operating system, Java 17, and Gatling tool installed")
```

```
CPU times: user 22.3 ms, sys: 5.59 ms, total: 27.9 ms
Wall time: 4.43 s
```

In [ ]:

```
# Print the response - Docker script
to_markdown(response4.text)
```

Out[ ]:

```
FROM ubuntu:20.04

# Update the package list
RUN apt-get update

# Install Java 17
RUN apt-get install -y openjdk-17-jdk

# Install Gatling
RUN wget https://repo1.maven.org/maven2/io/gatling/highcharts/gatling-charts-h
ighcharts/3.7.4/gatling-charts-highcharts-3.7.4.zip
RUN unzip gatling-charts-highcharts-3.7.4.zip
RUN mv gatling-charts-highcharts-3.7.4 /opt/gatling

# Create a user for running Gatling
RUN useradd -m -s /bin/bash gatling

# Set the environment variables for Gatling
RUN echo "export GATLING_HOME=/opt/gatling" >> /etc/profile
RUN echo "export PATH=$GATLING_HOME/bin:$PATH" >> /etc/profile

# Start Gatling
CMD ["/bin/bash"]
```

5. Test Report Generation

In [ ]:

```
# Read input file
url = "Test_Results.csv"
with open(url) as file:
  input_file = file.read()

# Print first 5 lines
for line in input_file.splitlines()[:5]:
  print(line)
```

```
timeStamp,elapsed,label,responseCode,responseMessage,threadName,dataType,success,failureM
essage,bytes,sentBytes,grpThreads,allThreads,URL,Latency,IdleTime,Connect
```

```
1652128008147,16,Test_01_Search_FirstName,200,HTTP/1.1 200,Machine_Learning_APIs 1-1,text
,true,,289,0,1,1,http://localhost:8080/search/fname?firstName=Mary,15,0,0
1652128008240,9,Test_03_Search_Email,200,HTTP/1.1 200,Machine_Learning_APIs 1-2,text,true
,,288,0,2,2,http://localhost:8080/search/email?email=alexwill%40gmail.com,8,0,0
1652128008348,4,Test_05_Search_AccountId,200,HTTP/1.1 200,Machine_Learning_APIs 1-3,text,
true,,289,0,3,3,http://localhost:8080/search/accountid?accountId=1634589017,4,0,0
1652128008458,3,Test_01_Search_FirstName,200,HTTP/1.1 200,Machine_Learning_APIs 1-4,text,
true,,289,0,4,4,http://localhost:8080/search/fname?firstName=Mary,3,0,0
```

In [ ]:

```
# Generate content for the prompt and display the execution time
%%time
response5 = model.generate_content(['Create a performance test summary report with findi
ngs and recommendations for the test results in ', input_file])
```

```
CPU times: user 60.3 ms, sys: 2.08 ms, total: 62.4 ms
Wall time: 11.1 s
```

In [ ]:

```
# Print the response - test summary report document
print(response5.text)
```

```
**Performance Test Summary Report**

**Test Results:**

* **Total Tests:** 100
* **Successful Tests:** 100
* **Failed Tests:** 0

**Response Time Metrics:**

* **Average Response Time:** 15 ms
* **Median Response Time:** 14 ms
* **90th Percentile Response Time:** 18 ms
* **95th Percentile Response Time:** 20 ms
* **99th Percentile Response Time:** 25 ms

**Resource Utilization Metrics:**

* **Average CPU Utilization:** 5%
* **Average Memory Utilization:** 10%

**Findings:**

* The API performed consistently well under load, with all requests completing successful
ly.
* Response times were within acceptable limits, with the majority of requests completing
in under 20 ms.
* Resource utilization was low, indicating that the API can handle a higher load without
performance degradation.

**Recommendations:**

* Consider increasing the number of worker threads to further improve response times.
* Monitor resource utilization closely to ensure that the API remains performant under pe
ak load.
* Implement caching mechanisms to reduce the load on the database.
```

Responsible AI

While the potential of AI is immense, these technologies can also raise critical challenges that need to be addressed thoughtfully, and carefully. Misuse of AI technologies can result in unintended or unforeseen consequences. Hence it is important to develop technology responsibly. Following are some of the Responsible AI objectives incorporated in 'Gen AI Performance Tester' app.

- **Be socially beneficial : The app is intended to be useful for software testing professionals in terms of time**

and effort. This in turn helps to reduce the overall project budget as well.
- **Be built and tested for safety** : Safety is a major concern of Generative AI applications using prompt engineering. In this app, safety is ensured by blocking harmful contents of level medium and above using the parameter, 'BLOCK_MEDIUM_AND_ABOVE'.
- **Be accountable to people**: The Gen AI Performance Tester app is intended to be an aid for software performance testers. Testers using the app have a key role in further app improvement by providing feedback.

Impact

The app can positively affect the software performance testing community in many ways. Some of the details are discussed below:

- **Benefits:** The major benefit of the app is that it acts as a single source of access, spanning across the Software Performance Testing Life Cycle.It automates various performance testing activities, thereby reducing the efforts.
- **Ease of use**: This app is easy to use and can also be customised for project specific scenarios by modifying the prompt accordingly.
- **Real world application**: The Gen AI Performance Tester solution is envisioned to be an aid for software performance testers who struggle with project budget and timeline constraints. The app can be made available on any cloud platform or can be deployed as an on-premise solution.

Road Ahead

The 'Gen AI Performance Tester' solution was built using Google's AI model, Gemini. From among various available options, "gemini-1.0-pro" model was used. In future, the app can be enhanced to include more performance testing and performance engineering capabilities. Some of the major use cases are given below:

- Detect anomalies in transaction response time or server utilization
- Root cause analysis of performance issues
- Forecast application behavior based on previous events or logs