Response Metrics

S.No	Type	Description
1	Avegrage Reponse Time	Average Time an API responds to a request
2	Peak Response Time	Maximum Response time the API is capable of reaching
3	Error Rate	Mathematical percentage of Overall maxmium errors in API relative to requests. This may be a status code errors or response timeout errors

Response Time has two essential characteristics - **Average Response Time, Maximum Response Time**

S.No	Response Time	Significance
1	0.1 Second	Most preferrable response time where the user sees instantanious interactable activity in web
2	1.0 Second	Maximum acceptable limit. Though user wont feel interuppted, but may experience some delay
3	>3 second	Time when user starts great seeing visible delays. This should be greatly considered on how the response affects to mobile users or users with low speed network. Bounce rate is 45%
4	>5 second	Vast delays and site become unusable for most users. The site's bounce rate will increase to an average 70%

API Performance Benchmark As of 2021

- **An average user loses interest if page takes more than 3 seconds to load.**
- **The best server reponse time is under 327ms (World's Top 20% sites), 224ms(Top 10% sites)**
- **Anything less than 1.1 second is considered Good**
- **Response Time more than 1.5 seconds are considered poor**
- **Response Time more than 3 seconds are worst performing sites and has 45% bounce rate**

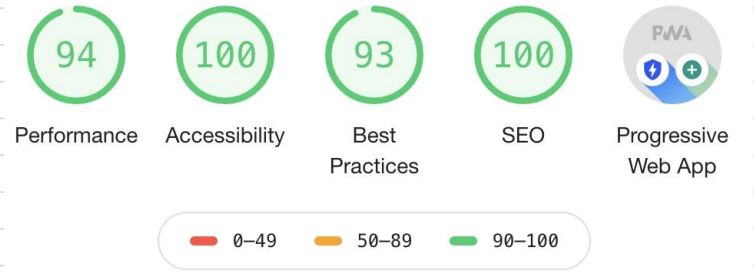
Benchmark Score			
5	< 224 ms		Outstanding
4.5	225-500ms		Excellent
4	500ms - 1s		Good
3.5	1.1 - 1.5 seconds		Ok
3	1.5-2 s		Caution
2.5	2.1-2.5		Unfavourable
2	2.6 - 3 seconds		Poor
1.5	3 -3.5seconds		Worst
1	> 3.5 seconds		Critical

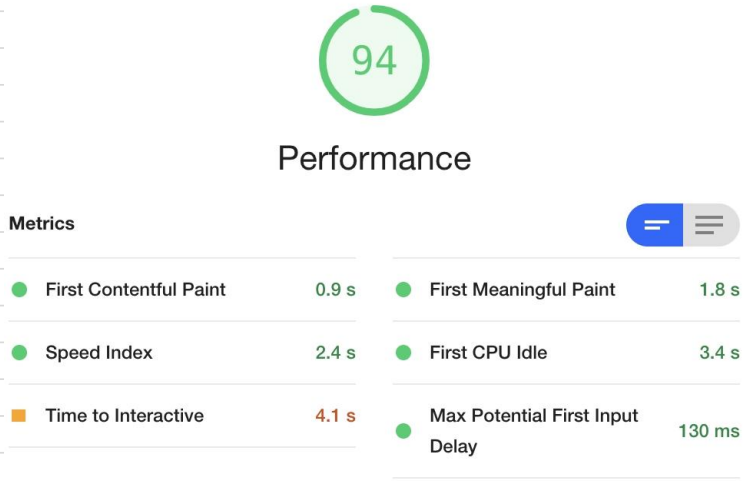
Benchmark must consider both Technology Stack(Database Serves, App Severs, Proxies, Programming Language Used, Framework used) and If optimal Resources are Available(CPU, Memory, Storage)

API Performance Guidelines	
	Since python is low in performance, Some complex data structures or general data structures handling in Python performs low compared to other compiled language like Java, .NET . So data must be manipulated carefully considering these. Consider BigO-Time(the amount of time the code takes) and Space(the amount of resources it uses on the process) complexity.
	Avoid nested conditionals and for loops as much as possible. Remove unused imports and stdout statements. Avoid too many variable declarations. Avoid changing variable types erratically once declared(If variable is assigned an int type. Make sure it stays that type until the end of its lifecycle

☑	Anything performant in Local development may not perform well under traffic. Because of the amount of concurrent requests the application handles depends on its gateway app server and how efficiently it uses its resources
☑	Database factors must be considered predominantly. For example using 5 standalone queries may perform well under low traffic. Because resources will be there to serve those requests. But same cannot be said when API started to receiver lot of requests. So using minimal queries leads to minimal connection management, Reduced CPU cycle and I/O of database and efficient overall resource usage. Also one of the APIs key performance metrics is related to reduced queries and joins.
☑	Always be aware of what goes into variables. For example selecting * from database and saving the results to variable will consume huge memory. This usage gets bigger when using pandas data frames. Tools can be used to profile APIs after getting prior approval from required persons(Tech Lead/Manager). Some sample tools are django-silk(Django), Flask-profiler(flask), python cprofile(generic), memory-profiler(python) or 3rd party Apps like Sentry(Error Tracking and Perfomance Analysis)
☑	Response Blocking user actions or long running process must be moved to background process. A basic example is sending mail.
☑	Third party API calls and and its associated network calls must be considered while designing APIs. if needed some can be moved as background Jobs.Or use a scheduler to store the data into database and retrive from it
☑	Use caching where neccessary. Caching should not be a shortcut for performance optimization for poor code, also it consumes significant resource Only go for caching when all other optimizations are done and the API is no longer optimizable. Consider Design Decisions
☑	Its always best to incorporate static analysis check(Pylint) and standard check(PEP8), security check(Bandit, Sonar cube) as early as possible, at the start of development cycle for increased code quality
☑	Starting with minimal unit tests are essential for improving quality. Modern development suggests having a self-tested code as one of the best practice and also is the root of TDD(Test-Driven-Development)
☑	Here are some links to Peformance improvement in python
	https://wiki.python.org/moin/PythonSpeed/PerformanceTips

	Guido van Rossum wrote a much more detailed (and succinct) examination of loops is worth reading especillay https://www.python.org/doc/essays/list2str/
Recommendations Based on Google Page Insights	
☑	Server response time measures how long it takes to load the necessary HTML to begin rendering the page from your server, subtracting out the network latency between Google and your server. There may be variance from one run to the next, but the differences should not be too large. In fact, highly variable server response time may indicate an underlying performance issue
☑	You should reduce your server response time under 200ms. There are dozens of potential factors which may slow down the response of your server: slow application logic, slow database queries, slow routing, frameworks, libraries, resource CPU starvation, or memory starvation
☑	You need to consider all of these factors to improve your server's response time. The first step to uncovering why server response time is high is to measure. Then, with data in hand, consult the appropriate guides for how to address the problem. Once the issues are resolved, you must continue measuring your server response times and address any future performance bottlenecks
☑	**Gather** and **inspect** existing performance and data. If none is available, evaluate using an automated web application monitoring solution(there are hosted and open source versions available for most platforms), or add custom instrumentation.
☑	**Identify** and **fix** top performance bottlenecks. If you are using a popular web framework, or content management platform, consult the documentation for performance optimization best practices
☑	**Monitor** and **alert** for any future performance regressions!


94
Performance
100
Accessibility
93
Best Practices
100
SEO
PWA
Progressive Web App
0–49
50–89
90–100


94
Performance
Metrics
First Contentful Paint 0.9 s
First Meaningful Paint 1.8 s
Speed Index 2.4 s
First CPU Idle 3.4 s
Time to Interactive 4.1 s
Max Potential First Input Delay 130 ms