Mindtree
A Larsen & Toubro Group Company

A best practices guide to performance **principles and patterns**

Introduction

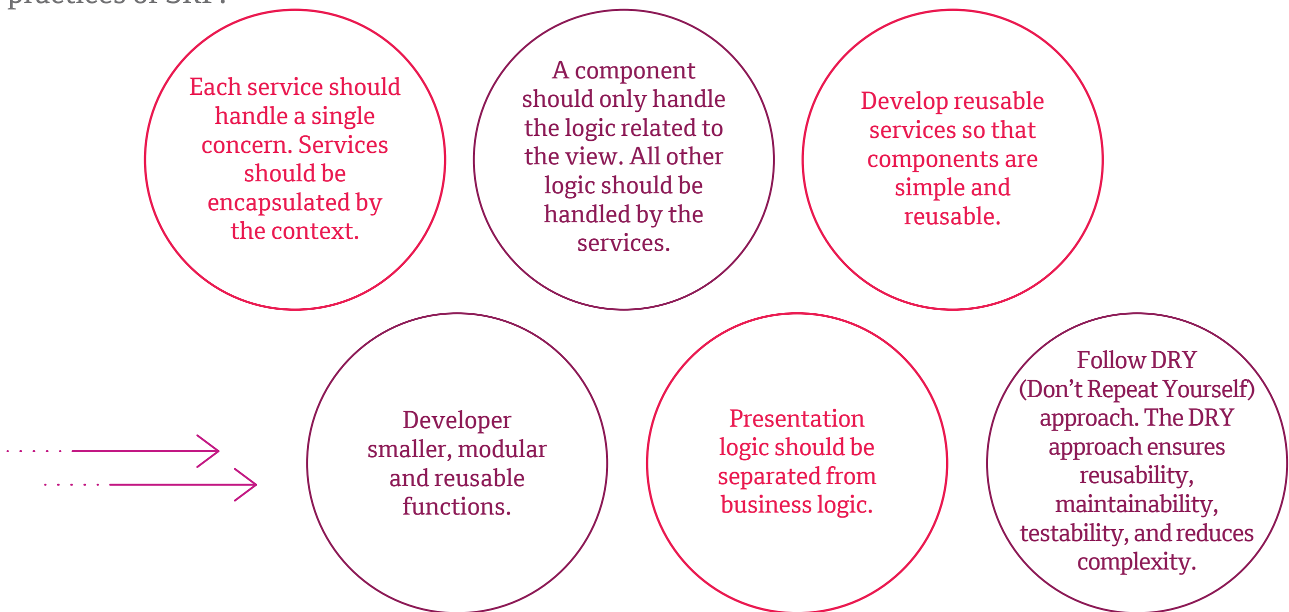
Performance principles provide a best practices-based approach to design optimal performance. In this whitepaper we discuss the main performance-related principles and the primary bottleneck scenarios and patterns. We also discuss performance validation methods.

Performance related architecture principles

Here are the key performance-related optimization principles:

Single Responsibility Principle (SRP)

The SRP principle states that modules should do one thing. By applying the SRP to components and services, it is easier to maintain, modularize, test, and extend the code. Here are some best practices of SRP:



Responsive design

Ideally, we should leverage responsive design to cater to multiple devices and browsers. The responsive design utilizes HTML 5, CSS 3 and media queries to render the web page optimally across various browsers and devices.

Single code base

Develop the web platform using a tool that uses a single code base for web and for various mobile devices. Isomorphic applications use a single code base for client and server side. For instance, we can use IONIC4 based Hybrid cross-platform development using Angular7 with a single source code for both iOS & Android mobile apps.

Optimal maintenance cost and shorter time to market

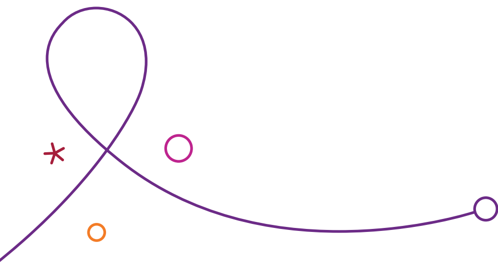
A single code base improves maintainability and results in effective cost reduction for implementation, support, and maintenance. The single code base also accelerates time to market.

Separation of concerns

Create different modules and components to handle data, UI rendering logic, business logic and communication logic. This pattern improves code testability, reusability, and extensibility.

Testability

The modules and components should follow the single responsibility principle (SRP) that separates concerns. This makes the code easily testable.



Plug and play architecture

The server APIs must be flexible enough to support various clients (such as browsers, mobile apps, tablets, wearables, watches, kiosks, etc.); various deployment models (such as on-premise, cloud or serverless architecture); various protocols (such as HTTP, HTTPS, REST, SMTP, etc.); and be platform agnostic (across cloud, database, UI framework etc.).

Extensibility

The APIs should be decomposed based on business functionality. API-driven design should enable incremental addition of business capabilities.

Service governance

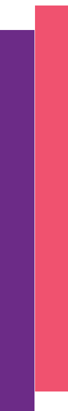
Normally the integration middleware systems provide the service governance. They handle the concerns such as:

- Parsing incoming requests or inject headers in response. For example, we could use body-parser middleware for parsing the HTTP request.
- Authenticating requests before processing. For example, we could use passport to handle various authentication strategies (such as OAuth, Email, AD, third party authentication etc.). Middleware such as helmet can secure the APIs through headers.
- Logging request and response.
- Handling CORS (Cross Origin Resource Sharing).
- Proxying requests through middleware such as http-proxy.
- Error-handling by gracefully handling application errors, connection errors and data errors.
- Compress the HTTP response. For example, compression middleware can be used for HTTP response compression.
- Cache the required and static data.
- Perform any required data transformation.

The API façade pattern can be used at middleware to orchestrate the system-specific calls and abstract the details from the caller.

Asynchronous API invocations

- Most modern web platforms invoke the server APIs asynchronously. Such asynchronous calls prevent blocks in request processing and we can execute the tasks in parallel.



Performance bottleneck scenarios and patterns

Here's an overview of key performance bottleneck scenarios and the performance patterns to address them.

Component causing bottleneck	Bottleneck scenario at high load	Web performance optimization pattern
User-agent layer		
Page level web objects	> 100 objects per page impact page size and page load times.	Minimize objects per page and load resources asynchronously.
Resource requests	>20 synchronous resource requests per page impact page load time.	Minimal HTTP requests. Avoid long-running scripts. Minify and merge resources to minimize resource requests.
Inline image & inline script	Inline image and inline script increase page load time by 31% and 16% respectively.	Externalize images and scripts. Avoid inline scripts and images.
Web objects in critical path	HTML Parsing & JS execution forms 35% of critical path and creates bottleneck.	Avoid long-running scripts. Use asynchronous scripts.
Web server layer		
3rd party script/external object	Creates front-end SPOF for long running scripts.	Use async scripts and test 3rd party objects. Use time out to avoid blocking.
Scripts	Synchronous request of long-running scripts/files blocks the page and creates a single point of failure.	Asynchronous resource request and on-demand loading. Use the iframe of the 3rd party scripts. Real user monitoring of performance metrics.
Application server layer		
Server response	Impacts TTFB and latency and page load time by 10%.	CDN usage and connection caching.
Server configuration	Improper connection pool size, connection pool setting, threads' pool size impacts performance at heavy load.	Fine tune and test the application server settings.
Network layer		
DNS lookup	DNS lookup forms 13% of critical path.	DNS caching and connection caching.

Table 1: Performance bottleneck scenarios

Performance NFR compliance validation

In this section, I will discuss the key methods to test the performance NFR compliance. The main way to ensure the performance NFR compliance is to perform a thorough, iterative, and layer-wise performance test. Additionally, we should setup a robust performance monitoring system to constantly monitor and alert performance incidents.

Performance validation

We need to understand the application usage patterns and simulate the user load during performance testing. Based on the application needs and performance SLAs, we conduct various kinds of performance testing. The common types of performance testing are:

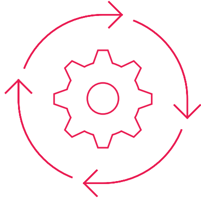
- **Load testing:** Measure the system performance at pre-defined load conditions and for a specified concurrent user load. Record the change in system behavior and performance with the increase in the load. Monitor the system resources such as CPU, memory, and network utilization during the load testing.

- **Stress testing:** Measure the system behavior and performance under stress conditions. The system will be subjected to peak loads, sudden spikes and extended high load scenarios as part of this testing. We will discover the application's break point and understand the maximum load levels the system can handle without performance degradation. Remember that the system will suffer from resource depletion during this testing. We use the following inputs for stress testing:

  - Identify peak limits – number of transactions, concurrent users, 24/7 availability etc.
  - How many concurrent users at peak and average load?
  - How many concurrent transactions at peak and average load?
  - Endurance testing: The system will be subjected to load testing for an extended duration (usually 48-72 hours) to measure system performance and system behavior. We can identify any potential memory leaks, buffer overflow issues, and hardware-related issues during this testing.
  - Scalability testing: We will test the system with various workloads based on the workload model. During this testing, we iteratively increase the concurrent user load and check the system performance. We must identify the key workload and mitigate the bottlenecks that impede the application scalability. Start with 10% of data, then scale up to maximum with increasing data volume periodically.
  - Reliability and availability testing: We will test the reliability and availability of the system during load testing and stress testing. We will also check the MTTF (Mean Time between Failures) for the system.
  - Performance benchmarking testing: We compare the application performance vis-à-vis the performance of earlier versions of the application. We also compare the application perfor mance with applications in the same category as well as competitive applications.
  - Volume testing: We need to test the system with data volume and content similar to production.

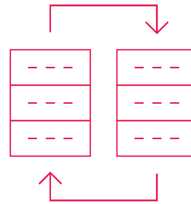
Key performance testing metrics

During performance testing, we record and report these key performance metrics:



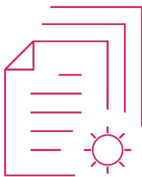
Response time:

We test the overall response time for pages, transactions and business processes. This entails testing the response time at various user loads.



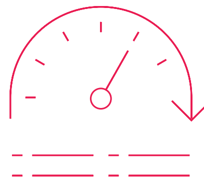
System scalability:

We test the system with various workloads based on the workload model. We iteratively increase the concurrent user load and check if the system scales successfully at various loads.



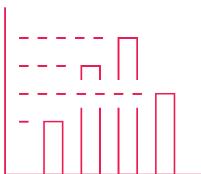
Reliability and availability:

We test the reliability and availability of the system during load testing and stress testing. We also check the MTTF (Mean Time between Failures) for the system.



Resource utilization:

We monitor the system resources such as CPU, memory, network bandwidth and input/output activities during various loads.



Efficiency:

We check if the resource utilization is healthy and within the agreed thresholds during various loads. For instance, we monitor if the CPU utilization is within 80% during the entire duration of load testing.

Recoverability:

We test how well the system recovers from failure and how well the system handles errors gracefully.

Performance monitoring tools

We could leverage a range of open source and commercial tools for performance monitoring. Table 2 lists the popular performance monitoring tools.

Real-time event monitoring, visualizations and data queries	Prometheus & Grafana
Search engine	Elasticsearch
Synthetic monitoring tool	• DynaTrace (Commercial) • Selenium • Lighthouse • Webpagetest.org
Database monitoring	• Automatic Workload Repository (AWR) • Fluentd
Log monitoring	• Splunk • Fluentd
Message streaming Notification	Alert manager
Container monitoring	• Node exporter • Docker stats • cAdvisor • Prometheus
Web page monitoring (page size, page response time, number of requests, asset load time, etc.)	• Webpagetest.org • Site speed (https://www.sitespeed.io/) • Google page speed insights (https://developers.google.com/speed/pagespeed/insights/) • Pingdom (commercial) • Silk performance manager (commercial) • Uptrends (commercial) • https://web.dev/measure/
Development tools/ page auditing	• Google Chrome developer tools • Test my site (https://www.thinkwithgoogle.com/feature/testmysite/) • Google Chrome lighthouse • HTTP Watch • https://speedrank.app/en • Fiddler • Firebug • Web tracing framework http://google.github.io/tracing-framework/ • Timeline tool https://developers.google.com/web/tools/chrome-devtools/evaluate-performance/timeline-tool#profile-painting

Multi-geo web performance testing	• https://performance.sucuri.net/
Cloud monitoring	• AWS CloudWatch
Website speed test	• https://tools.keycdn.com/speed
Load testing	• BlazeMeter • Apache JMeter
Web site latency test	• Ping test (https://tools.keycdn.com/ping)
Real user monitoring (RUM)	• New relic • SpeedCurve (https://speedcurve.com/)

Table 2: Performance monitoring tools

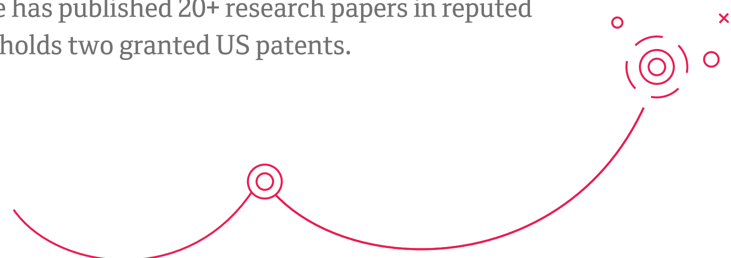
Conclusion

In this whitepaper we discussed the key performance-related architecture principles and best practices to design a high-performance application. In addition, we examined bottleneck scenarios and patterns. Lastly, we considered performance validation scenarios.

Dr. Shailesh Kumar Shivakumar

Solution Architect

Dr. Shailesh Kumar Shivakumar has 19+ years of experience across a wide spectrum of digital technologies, including enterprise portals, content management systems, lean portals, and microservices. He holds a PhD degree in computer science and has authored eight technical books published by the world's top academic publishers. He has also authored several academic content pieces for various undergraduate and postgraduate programs. He has published 20+ research papers in reputed international journals and holds two granted US patents.



About Mindtree

Mindtree [NSE: MINDTREE] is a global technology consulting and services company, helping enterprises marry scale with agility to achieve competitive advantage. "Born digital," in 1999 and now a Larsen & Toubro Group Company, Mindtree applies its deep domain knowledge to 260 enterprise client engagements to break down silos, make sense of digital complexity and bring new initiatives to market faster. We enable IT to move at the speed of business, leveraging emerging technologies and the efficiencies of Continuous Delivery to spur business innovation. Operating in 24 countries across the world, we're consistently regarded as one of the best places to work, embodied every day by our winning culture made up of over 27,000 entrepreneurial, collaborative and dedicated "Mindtree Minds."