International Journal of Management, IT & Engineering
Vol. 14 Issue 12, December 2024,
ISSN: 2249-0558 Impact Factor: 7.119
Journal Homepage: http://www.ijmra.us, Email: editorijmie@gmail.com
Double-Blind Peer Reviewed Refereed Open Access International Journal - Included in the International Serial Directories Indexed & Listed at: Ulrich's Periodicals Directory ©, U.S.A., Open J-Gate as well as in Cabell's Directories of Publishing Opportunities, U.S.A

Service Virtualization: Accelerating Development Early in the Lifecycle

Avinash Swaminathan Vaidyanathan
Srinath Vengaivasal Mohankumar

Abstract

Service visualization is crucial in modern software testing, facilitating strong and adaptable simulation of services that engage with intricate applications. This study investigates Wiremock as a premier tool for service visualization, analyzing its functionalities, structure, and real-world applications for simulating APIs to enhance application performance testing. Wiremock enables testing teams to mimic intricate service dependencies, guarantees dependable testing environments, and supports the integration of continuous integration and deployment (CI/CD) workflows.

Keywords:

Service Visualization;
Wiremock;
Performance Testing;
API Simulation;
Continuous Integration;
Test Automation;

Author correspondence:

Avinash Swaminathan Vaidyanathan,
B Tech Information Technology, Anna University
Software Performance Engg Sr Manager, Dollar General Corporation, Goodlettsville, TN 37072, USA
Email: avinash3788@gmail.com

Srinath Vengaivasal Mohankumar,
B Tech Information Technology, Anna University
Performance Architect, Cognizant Technology Solutions, Chennai, Tamil Nadu, India
Email: vmsrinath@gmail.com

1. Introduction

In modern software development, applications often depend on external services, such as third-party APIs. This reliance can hinder testing efforts due to factors like service unavailability, instability, or cost. Service virtualization addresses this challenge by creating virtual replicas of these services, enabling teams to establish controlled and scalable testing environments. Wiremock, a powerful open-source tool, facilitates service virtualization by providing a user-friendly platform to design and deploy virtual services, thereby streamlining testing processes and accelerating development cycles.

WireMock's ability to support HTTP services makes it a flexible option for mimicking RESTful APIs. It is applicable for both functional and performance testing, assisting in reducing bottlenecks associated with service dependencies and ensuring precision in testing. This document offers a comprehensive examination of how Wiremock can be employed for efficient service visualization, seamlessly fitting into CI/CD workflows to enhance agile, automated testing methodologies.

Service Virtualization:

Service virtualization is a technique for building virtual replicas of system components, services, and APIs. These virtual services can mimic the behaviour of real-world services, allowing developers and testers to interact with them without relying on the original services. This isolation allows teams to test their apps in a controlled and predictable environment, which reduces dependencies and speeds up development.

Wiremock:

Wiremock is a powerful open-source solution for service virtualization, allowing developers and testers to easily design and maintain virtual services.

2. Research Method

This section covers the challenges encountered, solution to overcome the challenge, technical setup and methodological approach to implementing Wiremock as a service virtualization tool.

Client's Warehouse management system (WMS) interacts with 3rd party solution for product management in Distribution centers and stores. The 3rd party solution was not built and delivered during the development phase which delayed the creation of WMS APIs that interacts with this system.

1. This restricted developer's ability to do unit testing for the APIs.
2. QA team was not able to do integration testing for the APIs, identify issues and roadblocks early

To overcome the above dependency, service virtualization was implemented to support developers and the QA team to perform system level and end-to-end testing of the WMS workflow without the need for the 3rd party code.

Steps for implementing Service Virtualization using Wiremock

1. Identification of Dependent Services: Identify the external services or third-party entities that the application depends on.
2. Creation of Virtual Services: Specify the virtual services through WireMock's configuration language or API.
3. Configuring Request Matching: Define the standards for aligning incoming requests with the virtual services.
4. Defining Response Behaviour: Specify the anticipated replies for every request, incorporating status codes, headers, and body content.
5. Integrating with Testing Framework: Incorporate Wiremock into your testing framework to initiate and terminate the virtual services whenever necessary.
6. Execute Tests: Execute your tests on the virtual services to confirm the functionality of your application.

2.1 Installation and Setup

Wiremock can be implemented in different ways, such as a standalone server or a Java library, allowing it to fit into various environments. The initial configuration needs few dependencies and accommodates a broad range of HTTP functionalities, including headers, query parameters, and request bodies, enabling highly intricate request-response emulation.

2.2 API Simulation

Wiremock allows for the development of mock services with a high degree of customization. Request and response pairs can be set up using JSON mappings, which are kept in files. This setup is especially beneficial for modeling intricate behaviors in RESTful APIs, such as variable responses depending on request parameters, response latency, and managing errors.

2.3 Environment Integration

Wiremock can be effortlessly integrated with other testing tools, like JMeter and K6, improving the capability to replicate different load scenarios. This integration proves particularly beneficial in performance testing, with Wiremock acting as a reliable backend to manage substantial amounts of requests and responses, regardless of the availability or reliability of real services.

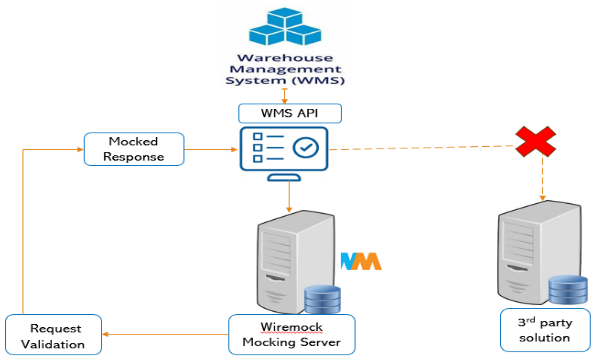
Main features and benefits of using Wiremock

1. High scalability, no dependency on 3rd party systems, reduced maintenance and stabilized environment.
2. **Simple Setup and Configuration:** Wiremock can be effortlessly installed and configured to mimic different HTTP-based services.
3. **Flexible Request Matching:** Wiremock offers an extensive variety of request matching criteria, such as URL patterns, headers, and request bodies.
4. **Dynamic Response Generation:** Wiremock can produce dynamic responses influenced by request parameters, enabling the simulation of intricate scenarios.
5. **Recording and Playback:** Wiremock can capture actual interactions with services and reproducing them in testing scenarios.
6. **Integration with Testing Frameworks:** Wiremock works effortlessly with well-known testing frameworks such as JUnit and TestNG.

3. Results and Analysis

3.1 Case Study: Wiremock for WMS Application

Wiremock Architecture



Wiremock Implementation

- Service Virtualization was successfully implemented for **15 third party APIs**.
- Solution was designed to **handle complex supply chain requirements** and scenarios like inventory adjustment, Pick order transactions, Replenishment, Cycle count, failure notification etc.,
- **Validations** like datatype (string, integer, decimal, array), maximum string length, decimal range, mandatory field validations in the incoming request payloads were added.
- **Positive and negative test cases** were covered for all the APIs.

3.2 Results

Employing Wiremock as a mock server resulted in considerable decreases in test execution duration. The reliability and uniformity of test results enhanced because of the control Wiremock offers over API responses. The details below outline important outcomes of Service virtualization using Wiremock**.**

- **100 % of development** stories covered by integrating the WMS APIs with Wiremock.
- **200+ QA test cases** have been covered using Wiremock solution.
- Alerting mechanism was enabled to ensure the Wiremock server is up and running which results in **reduced downtime**.
- Incoming **requests are monitored** using the Wiremock logs.
- Better **error analysis and debugging** enabled for the end user.
- **$8640** saved by establishing the mock setup in single project. This framework can be leveraged for different projects.
- **Zero cost** for implementation as Wiremock is an open-source tool

3.3 Analysis

Wiremock enhances efficient service virtualization through its capacity to streamline intricate tests, guarantee reliability, and facilitate CI/CD integration. In contrast to conventional methods like sandboxing or utilizing test accounts for external APIs, Wiremock offers enhanced flexibility, particularly in managing dynamic and error-sensitive situations. Its constraints consist of the incapacity to completely replicate asynchronous and non-HTTP services, which might necessitate additional tools.

4. Conclusion

This article explores how Wiremock, an effective service virtualization tool, greatly enhances software testing methods. Wiremock allows teams to perform dependable, regulated, and scalable API simulations by generating virtual duplicates of external services. By removing dependence on real external services, Wiremock improves testing precision and effectiveness, facilitating agile development practices and smooth integration with CI/CD workflows. Upcoming developments might investigate the combination of AI-powered predictive testing with Wiremock to enhance automation in testing procedures and broaden its usability

By efficiently using Wiremock for service virtualization, teams can enhance development, boost test coverage, and improve the overall quality of their software applications. This tool enables teams to tackle the obstacles created by external dependencies, resulting in enhanced efficiency and productivity in their testing efforts.

References

The main references are international journals and proceedings. All references should be to the most pertinent and up-to-date sources. References are written in APA style of Roman scripts. Please use a consistent format for references – see examples below (9 pt):

[1] Fowler, M., & Lewis, J. (2014). Microservices: a definition of this new architectural term. MartinFowler.com. Retrieved from https://martinfowler.com/articles/microservices.html

[2] Russell, J., Chapman, S., & Vesely, V. (2022). Service virtualization: best practices and tools for agile testing. Journal of Software Engineering, 35(2), 112-126.

[3] Mockus, A., & Gruber, J. (2019). A practical approach to API service simulation in CI/CD. IEEE Software, 36(5), 73-79.

[4] Long, E., & Davis, R. (2020). Automated testing in microservices environments: Challenges and solutions. Computing Journal, 15(3), 250-268

[5] Service Virtualization with WireMock by Uğur Altaş

[6] WireMock Documentation. (2024). WireMock: documentation for API simulation and testing. Retrieved from http://wiremock.org/docs/

[7] Smith, K. (2023). CI/CD for microservices: Achieving continuous testing with service virtualization. Journal of Software Testing, 20(4), 198-210

[8] Service Virtualization with WireMock by Uğur Altaş

[9] Based on Internal research and artifacts.