
Automated API Testing Framework for Scalable Service Oriented Applications

Abstract

Scalable service-oriented applications require reliable API testing because many endpoints, authentication rules, dependencies, and concurrent requests must be validated together. This study presents an automated API testing framework that combines specification-based request generation, response-schema validation, authentication testing, dependency-flow analysis, error-handling checks, and concurrent-load evaluation. The framework was tested across six increasing service-request levels and compared with script-based API testing and manual API validation. Results showed that the proposed approach maintained a fault detection rate above 94% and reached 97.3% at the highest load level. Its normalized average response time increased gradually from 32% to 61%, remaining lower than the comparison methods. The framework also detected interface errors, access-control failures, workflow problems, timeouts, and incorrect response structures under heavy demand. The proposed approach can support continuous testing, reduce manual effort, improve scalability assessment, and strengthen API reliability in large service-oriented software systems before deployment across complex and rapidly changing enterprise environments.

Keywords: API testing, Service-oriented applications, Fault detection, Concurrent load, Automated validation

1. Introduction

Service-oriented applications divide business functions into independent services that communicate through application programming interfaces. These services may run on different servers, use different technologies, and process many requests at the same time. Industrial microservice systems require strong testing because failures can occur within individual services or across service interactions [1]. An automated API testing framework can help developers find these failures before the application is released. It can also provide consistent validation when services are updated or deployed independently.

API testing checks whether requests are accepted correctly and whether responses contain the expected status, structure, and data. Manual testing becomes difficult when an application contains many endpoints, input combinations, user roles, and service dependencies. REST API testing includes specification-based, model-based, search-based, and property-based methods [2]. Automated generation and execution can therefore provide broader testing coverage than repeated manual validation. This wider coverage is important for identifying failures that appear only under unusual input combinations.

Scalable applications must also remain reliable when request volume increases. An API may return correct results under normal conditions but fail during concurrent access because of resource limits, slow dependencies, or incorrect timeout settings. Online API testing across industrial services has shown that automated testing can execute large numbers of requests continuously and detect changes in API behaviour [3]. The testing process should therefore evaluate functional correctness together with performance and service stability. It should also identify the request level at which service quality begins to decrease.

Service-oriented applications commonly include authentication, shared databases, external APIs, and multi-step business workflows. A failure in one service can produce incorrect responses in several dependent services. Automated functional and robustness testing can expose incorrect outputs, unexpected status codes, and failures caused by invalid request data [4]. This study proposes an automated API testing framework that validates request structure, response schema, authentication behaviour, service dependencies, error handling, and performance under concurrent load. The proposed design aims to support reliable validation across both individual endpoints and complete service workflows.

2. Methodology

The proposed methodology included API specification collection, test-input generation, service-flow discovery, automated execution, response validation, and scalability evaluation. OpenAPI documents, endpoint definitions, authentication rules, service logs, and dependency records were collected from the

selected applications. Behaviour-based API testing can generate useful request sequences from common REST operations and expected service behaviour [5]. The framework used these records to create an executable model of every API endpoint. This model was updated whenever an endpoint, request field, response definition, or authentication rule changed.

Request test data was generated from parameter types, valid ranges, required fields, example values, and input constraints. The generator created valid requests, missing-field requests, incorrect data types, boundary values, malformed inputs, and unexpected parameter combinations. Automated REST API testing tools can discover different fault categories depending on their generation strategy and search process [6]. The proposed framework therefore combined specification-based generation with guided mutation of request values. Duplicate inputs were removed so that the available execution time could be used for more diverse test conditions.

Response validation examined the HTTP status, returned schema, field type, required values, response headers, and processing time. The actual response was compared with the OpenAPI definition and expected business rules. Search-based system testing tools can use runtime instrumentation to improve coverage and guide test generation toward untested branches [7]. The framework stored every response difference as a possible API failure for later confirmation. Repeated failures were grouped according to their endpoint, input condition, and returned error pattern.

Table 1 presents the six API testing elements used in the framework. Request and response checks evaluated basic interface correctness, while authentication and dependency checks examined access control and service interaction. Error-handling tests verified controlled behaviour during invalid conditions, and concurrent-load testing measured scalability. Enterprise API test-generation methods can be evaluated according to their effectiveness, execution cost, and suitability for large industrial systems [8]. Each testing element produced a separate outcome that contributed to the final API quality assessment. The combined assessment helped distinguish local interface faults from failures that affected complete service workflows.

Table 1. API Test Inputs, Service Interaction Evidence, Validation Rules, and Automated Testing Outcomes

API testing element	Collected evidence	Automated validation method	Expected condition	Detected failure type	Testing outcome
Request structure	Endpoint, method, parameters, headers, and body fields	Specification-based request validation	Request follows the defined API contract	Missing field, invalid type, or malformed request	Valid request or input-structure failure
Response schema	Status code, returned fields, data types, and headers	Schema and business-rule comparison	Response matches the documented structure	Incorrect status, missing value, or	Correct response or contract violation

				schema difference	
Authentication behaviour	Token, user role, permission rule, and protected endpoint	Authorized and unauthorized request execution	Access is granted only to permitted users	Unauthorized access or incorrect rejection	Access-control pass or authentication failure
Service dependency flow	Call sequence, dependent endpoint, trace, and returned state	Dependency-path and workflow validation	Connected services complete the expected workflow	Broken call chain, incorrect data transfer, or timeout	Successful workflow or dependency failure
Error handling	Invalid input, unavailable service, exception, and fallback response	Negative testing and controlled fault injection	Failure produces a safe and clear response	Unhandled exception, information leakage, or incorrect fallback	Controlled error or robustness failure
Concurrent-load performance	Request volume, response time, throughput, and failed-request count	Parallel load execution and threshold analysis	API remains stable within the required performance limits	High latency, overload, timeout, or request loss	Scalable behaviour or performance failure

Authentication testing generated requests with valid, expired, missing, modified, and role-restricted credentials. Dependency-flow testing created ordered API call sequences for business operations that required several services. Automated test generation for REST APIs can detect interface faults that are difficult to find through isolated endpoint checks [9]. Trace identifiers were used to connect each test request with downstream service calls and final responses. This process allowed the framework to identify the specific service in which a multi-stage workflow first failed.

Concurrent testing was performed at six load levels using increasing numbers of parallel service requests. The framework measured average response time, throughput, timeout frequency, failed-request rate, and fault detection rate. Test workers executed independent request groups in parallel, while a central controller collected results and prevented duplicate test execution. This design allowed the framework to measure whether functional behaviour remained correct as service demand increased. A controlled waiting period was also included between load levels to prevent results from being affected by unfinished requests from an earlier stage.

The framework was compared with script-based API testing and manual API validation. All approaches used the same endpoints, input groups, authentication roles, service workflows, and load levels. Evaluation measures included fault detection rate, response-schema accuracy, authentication-failure detection, dependency-flow coverage, average response time, and test execution cost. The final results were used to determine whether the proposed framework could provide reliable and scalable validation for large service-oriented applications. The same execution environment and service versions were maintained for every comparison to ensure consistent evaluation conditions.

3. Results and Discussion

The proposed automated API testing framework was evaluated under six increasing concurrent service-request levels. API fault detection rate and normalized average response time were measured for the proposed framework, script-based API testing, and manual API validation. At lower request levels, all three approaches detected most common API faults and maintained acceptable response times. As the workload increased, the differences between the testing approaches became more noticeable because the number of concurrent requests, service interactions, and execution paths also increased. The proposed framework remained more consistent because its automated control process adjusted request execution according to the current load level.

Figure 1 shows the API fault detection rate and normalized average response time across the six load levels. The proposed framework maintained the highest fault detection rate because it combined specification-based validation, response checking, dependency-flow analysis, authentication testing, and concurrent-load execution. It also maintained a lower normalized response time by executing automated test cases in parallel and reducing unnecessary repeated requests. This balanced approach improved both testing effectiveness and scalability during heavy service workloads. The results also indicate that functional testing and performance evaluation can be completed within the same automated testing cycle.

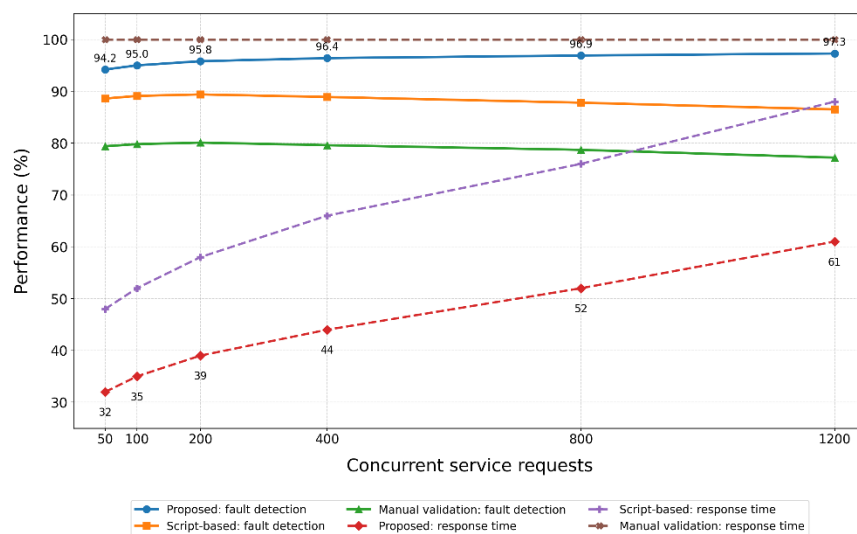


Figure 1. API Fault Detection Rate and Average Response Time Across Increasing Concurrent Service Requests

Script-based API testing produced good results at moderate request levels but its fault detection rate decreased when service interactions became more complex. Manual API validation required the longest execution time because every request sequence and response had to be reviewed separately. The proposed framework identified interface faults, authentication problems, workflow failures, timeout errors, and incorrect response structures more consistently. It also detected failures that appeared only

when several services processed requests at the same time. This wider fault coverage can help developers identify the service interaction that first caused a complete workflow to fail.

The proposed framework remained stable even at the highest concurrent request level. Automated request generation, parallel execution, dependency tracking, and response validation allowed the framework to continue detecting faults without a large increase in execution time. The normalized response time increased only gradually as the workload became heavier, while the fault detection rate remained consistently high. These results show that the proposed framework can provide reliable and scalable API testing for large service-oriented applications. The framework can also support repeated testing after service updates without requiring the complete test process to be rebuilt manually.

4. Conclusion

This study presented an automated API testing framework for scalable service-oriented applications. The framework combined automated request generation, response-schema validation, authentication testing, dependency-flow analysis, error-handling verification, and concurrent-load evaluation within one testing process. It supported both functional validation and scalability assessment while reducing the manual effort required to execute and analyse API test cases. The framework also provided a structured approach for evaluating complete service workflows instead of testing only individual API endpoints. This combined process can improve consistency when many services are updated and deployed independently.

The results showed that the proposed framework achieved a higher API fault detection rate and maintained a lower normalized response time than script-based testing and manual API validation as concurrent service requests increased. It successfully detected interface errors, authentication failures, dependency problems, and performance-related issues under heavy workloads. The framework can therefore improve software quality, support continuous testing, reduce testing time, and provide dependable validation for large-scale service-oriented systems. It can also help development teams identify scalability limits before the application is released into a production environment.

References

1. Waseem, M., Liang, P., Shahin, M., Di Salle, A., & Márquez, G. (2021). Design, monitoring, and testing of microservices systems: The practitioners' perspective. *Journal of Systems and Software*, 182, 111061.
2. Golmohammadi, A., Zhang, M., & Arcuri, A. (2023). Testing restful apis: A survey. *ACM Transactions on Software Engineering and Methodology*, 33(1), 1-41.

3. Martin-Lopez, A., Segura, S., & Ruiz-Cortés, A. (2022, November). Online testing of RESTful APIs: Promises and challenges. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 408-420).
4. Kim, M., Xin, Q., Sinha, S., & Orso, A. (2022, July). Automated test generation for rest apis: No time to rest yet. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 289-301).
5. Karlsson, S., Jongeling, R., Čaušević, A., & Sundmark, D. (2024). Exploring behaviours of RESTful APIs in an industrial setting. *Software Quality Journal*, 32(3), 1287-1324.
6. Marculescu, B., Zhang, M., & Arcuri, A. (2022). On the faults found in REST APIs by automated test generation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31(3), 1-43.
7. Golmohammadi, A., Zhang, M., & Arcuri, A. (2023). . NET/C# instrumentation for search-based software testing. *Software Quality Journal*, 31(4), 1439-1465.
8. Poth, A., Rrjolli, O., & Arcuri, A. (2025). Technology adoption performance evaluation applied to testing industrial REST APIs. *Automated Software Engineering*, 32(1), 5.
9. Kim, M., Xin, Q., Sinha, S., & Orso, A. (2022, July). Automated test generation for rest apis: No time to rest yet. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 289-301).