Zimic: Towards Compliant TypeScript API Mocks

Diego C. de Aquino, Everton L. G. Alves

Federal University of Campina Grande (UFCG), Campina Grande – PB – Brazil

diego.cruz.aquino@ccc.ufcg.edu.br, everton@computacao.ufcg.edu.br

Abstract. *Mocks are a common strategy to reduce flakiness and complexity in automated testing. API mocks, which intercept communications with external services, must align with the actual implementation to ensure realistic test scenarios. However, current solutions for TypeScript API mocking do not adequately support testers in declaring compliant API mocks. In this paper, we introduce Zimic, a novel approach to TypeScript API mocking that enhances compliance through static validation and test abstractions. We evaluated Zimic in two empirical studies involving students and professional testers. Our results show that Zimic improves mock compliance by 24.1%, test completeness by 27.2%, and readability by 15.1% compared to existing solutions.*

1. Overview

Unit and integration tests are widely used in software development to validate the behavior of an application [Torkar and Mankefors 2003]. However, features that rely on non-trivial components and external dependencies can be particularly challenging to test [Thomas and Hunt 2002]. When these dependencies are complex or outside the developer's control, testability can be significantly affected, often hindering readability and maintainability [Thomas and Hunt 2002, Mackinnon et al. 2000].

In such cases, *mocks* can reduce the complexity of tests. These objects simulate a concrete implementation, avoiding flakiness and helping developers to manipulate each scenario more easily [Mackinnon et al. 2000]. For example, Application Programming Interface (API) mocks are a class of mocks that are especially useful in web applications using HTTP and REST [Viglianisi et al. 2020]. They intercept communications with external services and return simulated responses.

A recurring challenge with mocks is maintaining their compliance with actual implementation [Spadini et al. 2017]. If mocks are not kept up-to-date as the real component evolves, tests may rely on unrealistic scenarios, undermining their confidence in the asserted behavior. Despite this, popular API mocking libraries in the TypeScript ecosystem, such as MSW[1], offer limited built-in features for compliance validation.

To address these challenges, we propose *Zimic*, a novel approach and tool[2] for API mocking. Zimic statically verifies each mock with a centralized schema, which defines the interface of the real service. The schema acts as a single source of truth and can be updated as the real service evolves, helping developers identify inconsistencies early in development. In addition, *Zimic* offers a set of test abstractions designed to encourage simplicity and clarity when declaring mocks and validating application behavior.

[1] https://mswjs.io

[2] https://zimic.dev

The Zimic approach (Figure 1) begins with the schema declaration (Step 1). In Step 2, interceptors reference the schema to validate new API mocks. Finally, in Step 3, requests matching the expectations of a handler receive the mock response.

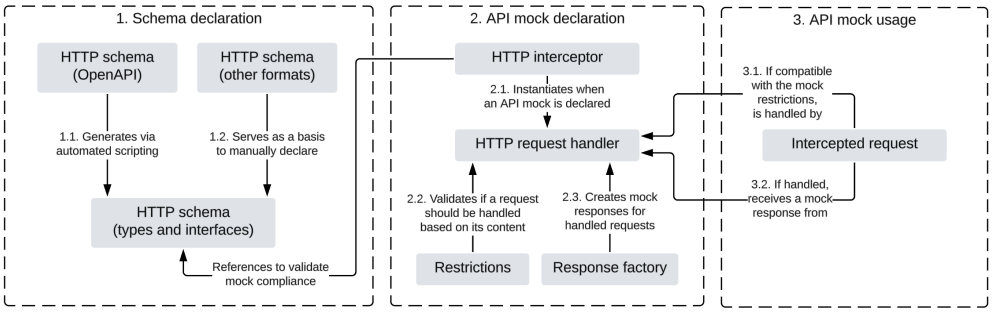

Figure 1. Overview of the Zimic approach.

We evaluated Zimic in two empirical studies involving 35 students and 6 professional testers. The participants implemented API mocks with Zimic and MSW and completed a survey with their opinions. Zimic received positive feedback and contributed to improve mock compliance by 24.1%, test completeness by 27.2%, and readability by 15.1%, although with a 32.6% and 27.1% increase in test creation and maintenance time.

This work originated from the first author's undergraduate thesis, completed as part of the 2024 Computer Science program at the Federal University of Campina Grande. Zimic was first introduced in a paper at an international conference [de Aquino and Alves 2025]. Additionally, a beta version reached 6,000 monthly downloads in the Node Package Manager (NPM) as of August 2025. These results highlight Zimic's contributions to both academia and practice.

Future work may extend this research by comparing Zimic with other API mocking tools. We also plan to grow the Zimic ecosystem with extended code validation, debugging tools, and additional network protocols such as WebSocket.

References

de Aquino, D. C. and Alves, E. L. G. (2025). Zimic: Bridging the gaps in api mocking for typescript projects. In *Proceedings of the IEEE International Computer Software and Applications Conference (COMPSAC)*. Accepted for publication.

Mackinnon, T., Freeman, S., and Craig, P. (2000). Endo-testing: unit testing with mock objects. *Extreme programming examined*, pages 287–301.

Spadini, D., Aniche, M., Bruntink, M., and Bacchelli, A. (2017). To mock or not to mock? an empirical study on mocking practices. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 402–412. IEEE.

Thomas, D. and Hunt, A. (2002). Mock objects. *IEEE Software*, 19(3):22–24.

Torkar, R. and Mankefors, S. (2003). A survey on testing and reuse. In *Proceedings 2003 Symposium on Security and Privacy*, pages 164–173. IEEE.

Viglianisi, E., Dallago, M., and Ceccato, M. (2020). Resttestgen: automated black-box testing of restful apis. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pages 142–152. IEEE.