The Austrian Journal of Technical and Natural Sciences, 2026 No 3–4
ISSN 2310-5607

PREMIER Publishing
ppublishing.org

DOI:10.29013/AJT-26-3.4-119-125

APPLICATION OF SOLID PRINCIPLES IN BACKEND DEVELOPMENT OF MEDIA PLATFORMS: ARCHITECTURAL RESILIENCE AND IMPACT ON THE PRODUCT LIFE CYCLE

Georgii Andreev [1]

[1] Technical Director, Lead Software Developer, Individual Entrepreneur

Moscow, Russia

Cite: *Andreev G. (2026). Application of SOLID principles in backend development of media platforms: architectural resilience and impact on the product life cycle. Austrian Journal of Technical and Natural Sciences 2026, No 3–4.* https://doi.org/10.29013/AJT-26-3.4-119-125

Abstract

This article examines the application of SOLID principles in the architecture of backend systems for modern media platforms. The importance of each of the five principles of object-oriented design in creating resilient and scalable software systems is discussed. The need for modularity and loose coupling in reducing system load and improving the efficiency of the product life cycle is highlighted. Particular attention is paid to the details of implementing SOLID principles in PHP and Python-based backend systems using typing, abstractions, and testing frameworks. The article also discusses the risks involved in the architecture of anti-patterns and the role of SOLID principles in overcoming such risks. It was concluded that the SOLID principles provide an important methodological foundation for the development of software systems with high architectural resilience.

Keywords: *SOLID principles, architectural resilience, media platform, PHP, Python, scalability, product life cycle*

Introduction

Contemporary media technologies involve complex, high-load software applications supported by robust backend infrastructures for handling user requests, content delivery, and integration with external services. The growing number of users and software features makes architecture a critical factor an essential role in determining the scalability and resilience of the backend systems. Contemporary technologies have an ever-changing and rapidly evolving landscape, and the software product development lifecycle relies heavily upon the solid foundation of architecture.

One of the most well-known methods for ensuring architectural sustainability is the use of SOLID principles, which consist of a set of five guidelines for object-oriented design. These principles aim to reduce module coupling, enhance cohesion, and promote extensibility and testability of software components. In the context of backend programming in media platforms, SOLID acts as

a conceptual framework for creating modular architecture that can grow in an efficient manner. The framework is particularly important in PHP and Python-based applications, in which the flexibility of the programming language requires architectural discipline.

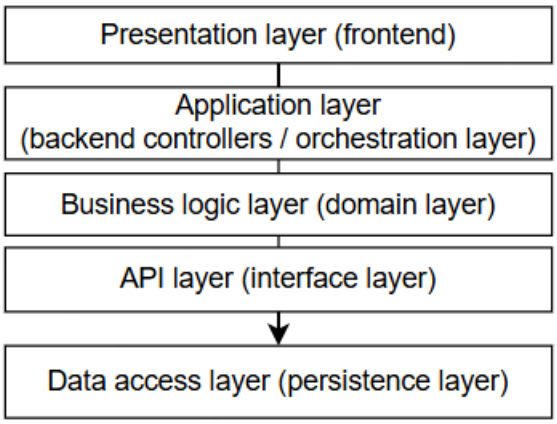
The objective of this article is to analyze the role of SOLID principles in enhancing the architectural stability of backend systems in media platforms and to evaluate their influence on the software product lifecycle. The study explores how these principles are implemented in multilayered system structures, APIs, and user logic modules, and demonstrates how adherence to architectural standards contributes to faster, more reliable, and more maintainable backend development.

Main part

Structure of backend architecture in media platforms and scalability challenges

Modern media platforms are built on principles of modular, hierarchically organized architecture aimed at ensuring resilience, extensibility, and technical maintainability under high load and rapid functional growth (Bogutskii A., 2025). The most widespread model is a multi-layer architectural approach, in which the logical and functional separation of the system components provides for the possibility of isolated data processing, independent development of the modules, and loosening of coupling between layers (fig. 1).

Figure 1. *Multi-layer architecture of a media platform: logical organization of the backend system*



A multi-layer architecture of a media platform is organized so that each layer addresses a clearly defined set of responsibilities, thereby simplifying scalability. The presentation layer, which enables the interaction of the user with the system through Web and mobile interfaces, constitutes the top level. The application layer then takes over for the routing of requests, invocation of appropriate services, and process mediation between the user interface and business logic. Essentially, the business logic layer is responsible for the implementation of the domain rules, entity management, and computation. This makes the business logic layer the functional definition of the platform. At the interface boundary, the API surface, or the controllers/endpoints, standardize access to the platform services, including the integration of other applications. Finally, the data access layer is responsible for controlling the access to the data stored in the databases or other data systems.

However, there are several challenges that are experienced by the multi-layer system. Some of the problems include the high level of coupling between the modules, inefficient allocation of responsibilities, and the lack of flexibility in the components. A frequent issue is the blending of business logic with infrastructure code, which leads to the emergence of a “monolith,” where changing one element requires substantial rework of others. Such a structure complicates scaling, slows down development and testing, and significantly reduces fault resilience. This problem becomes particularly acute in systems that process streaming data or real-time multimedia con-

tent, where resilience and scalability are critical requirements (Smirnov A., 2025).

As software systems grow in complexity, architectural sustainability increasingly depends not only on selecting appropriate technical solutions but also on applying sound structural principles in code organization. During the evolution of media platforms, typical architectural defects are often observed that hinder scalability, modularity, and maintainability. These defects are commonly classified as anti-patterns – recurring ineffective design practices that lead to architectural degradation (table 1).

Table 1. *Common architectural anti-patterns in backend systems of media platforms (Silva N. et al., 2025; Røhne A. L. et al., 2024)*

Anti-pattern name	Description	Technical characteristics
God object	A class or module that performs too many functions, violating the Single Responsibility Principle.	Large classes with many methods and dependencies.
Spaghetti code	Unstructured, tangled logic that reduces code readability and maintainability.	Deeply nested conditionals, duplicated code, lack of modularity.
Tight coupling	Components are directly dependent on each other, reducing flexibility and reusability.	Classes instantiate other classes directly; no interfaces.
Lack of separation of concerns	Business logic is intermixed with infrastructure or presentation logic.	Business rules placed inside controllers or request handlers.
Hard-coded dependencies	Concrete implementations are hardwired into components, hindering testability and extensibility.	Direct use of specific classes; no dependency injection or abstraction.

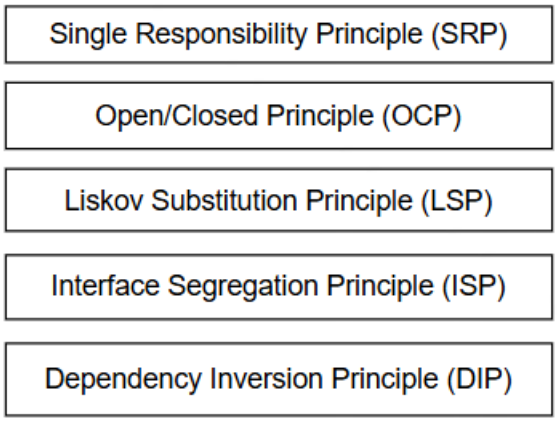
These architectural weaknesses result in inflexibility for the system to adapt to changing demands and requirements. Architectural weaknesses systematic fix requires employing architectural design principles concerning modularity, loose coupling, and change resilience. In this context, the adoption of SOLID principles becomes a justified and methodologically grounded approach, serving as an engineering foundation for the design and evolution of backend systems in media platforms.

SOLID principles as a response to architectural risks

One of the most effective approaches to overcoming the architectural limitations discussed in the previous section is the systematic application of the SOLID principles (Ramachandrappa N. C., 2024). This acronym encompasses five fundamental principles of object-oriented design (fig. 2).

All the mentioned principles are meant to reduce the structural coupling, to introduce the feature of modularity, and to improve the changeability of the software architecture, thus making the SOLID an important engineering tool for designing scalable backend systems.

Figure 2. *SOLID principles*



The **Single Responsibility Principle (SRP)** is defined by the fact that a class or module should change for only one reason, that is, it must serve a single purpose within a business domain. SRP is an effective means

to isolate a business logic from infrastructural logic. Using SRP can help to avoid several anti-patterns, for instance, God Object, from a system, while it is also a means to break a system into well-defined autonomous components.

The **Open/Closed Principle (OCP)** requires software entities to be open for extension but closed for modification. This is typically achieved through the creation of abstractions where new functionality should be introduced without modifying existing code. This becomes especially useful in media platform design when adding new content types, processing algorithms, or API interfaces with third-party services because such enhancements should be possible without affecting the existing stability of the system.

The **Liskov Substitution Principle (LSP)** defines that an object of a descendant class should replace an object of its base class without disrupting the program logic in any way. The correct operation of the system under the use of "inheritance" and "polymorphism" in its program logic could be severely disturbed by violated LSP principles.

The **Interface Segregation Principle (ISP)** states that clients should not be forced to depend on interfaces they do not use. This implies designing narrowly focused interfaces rather than broad, overloaded ones. In the context of APIs and service layer logic in media platforms, this reduces coupling between modules and simplifies unit and integration testing.

Lastly, the **Dependency Inversion Principle (DIP)** states the requirement for high-level modules to depend on abstractions rather than concrete implementations. In practice, DIP is often applied using inversion of control and dependency injection mechanisms, although these techniques represent implementation approaches rather than the principle itself. By decoupling objects and classes, DIP enhances architectural extensibility. In the context of backend systems for personalized content delivery and algorithmic decision-making, DIP plays a critical role in separating core business logic from specific personalization mechanisms and data-processing implementations. Such separation facilitates controlled evolution, auditability, and the substitution of algorithmic components without affecting higher-level system behavior, which is particularly important for ensuring transparency, traceability, and controlled modification of personalization logic in media platforms. This property aligns with contemporary engineering approaches in high-load backend environments that incorporate automated decision-making and personalization logic (Garifullin R., 2025).

Therefore, the SOLID principles provide a robust conceptual framework for designing architectures that are resilient to technical debt and the complexity associated with system growth. By applying these principles, developers can reduce the prevalence of common coding anti-patterns and place their software systems in a better position to manage complexity as new functionality is introduced.

Applying SOLID principles in Web Backends with PHP and Python: implementation and tooling

The application of SOLID principles in backend development using PHP and Python represents a significant engineering practice, enhancing the resilience, modularity, and adaptability of system architectures (Sharma S., 2024). Although PHP and Python differ in their typing systems and runtime characteristics, both languages provide strong support for object-oriented abstractions – such as interfaces and traits in PHP, and abstract base classes, protocols, and type annotations in Python – that facilitate SOLID-compliant designs.

Dependency management is an important aspect of applying SOLID principles in both PHP and Python backend systems. The DIP is one of the key considerations in the application of the SOLID design principles to the application architectures and designs found in the media platform space, including the abstraction from the specific implementations found in the use of the database, the payment mechanism, or the API.

The implementation of the SRP in PHP and Python involves the **use of heavy logic separation between layers.** The layers include the business layer, the controller layer, the repository layer, and the adapter layer. This is particularly useful when the application is structured in layers. The use of this principle helps in the prevention of the accu-

mulation of responsibility in any module and hence the creation of isolated modules. Similarly, the use of the OCP is mainly achieved using design patterns such as the Strategy design pattern and the Factory design pattern.

API interfaces play a central role in backend applications. In the context of SOLID, the ISP is particularly important, as it encourages the design of narrowly scoped contracts between system components. For example, different media platform services – such as user management, content publishing, and analytics – should expose independent interfaces rather than inheriting from broad, overloaded structures. This reduces component coupling and simplifies reuse.

In addition, adherence to SOLID principles significantly **simplifies unit testing and mocking.** Architectures based on abstractions and loosely coupled interfaces allow dependencies to be isolated effectively during tests. This is especially relevant for CI/CD processes and automated verification of complex system functionality. As noted in contemporary engineering literature, resilient architectures are achieved not only through formal modularity but also through design practices that constrain and control complexity growth.

Thus, applying SOLID principles in backends improves architectural quality, reduces maintenance costs, increases development velocity, and enhances the reliability of functional evolution in media platforms. This makes SOLID a foundational methodological framework for building scalable and adaptable systems.

Impact of architectural conformance to SOLID principles on the media product lifecycle

Architectural design principles, particularly SOLID, have a systematic impact on all stages of the life cycle of backend systems for media platforms – from initial design to operation, scaling, and continuous change delivery. Under conditions of highly dynamic user behavior and frequent releases, architectural decisions largely determine a system's ability to adapt to new requirements without quality degradation or increased operational risk. Applying SOLID provides a foundation for predictable architectural evolution, reducing the likelihood of technical debt accumulation and architectural decay at later life-cycle stages.

One of the key effects of SOLID conformance is improved change manageability and lower maintenance costs. Modularity, loose coupling, and reliance on abstractions allow changes to be localized within specific functional areas, which is especially important for media platforms that evolve iteratively and are subject to frequent adjustments in business logic, recommendation algorithms, and integrations with external services.

Table 2. *Impact of SOLID principles on key aspects of the media platform life cycle (Qiao R. & Yin Z., 2024)*

SOLID principle	Architectural effect	Life-cycle impact
SRP	Clear separation of component responsibilities.	Simplified maintenance and refactoring.
OCP	Extensibility without modifying existing code.	Reduced risk when introducing new functionality.
LSP	Correct substitutability and predictable behavior.	Increased stability during logic scaling.
ISP	Narrow, role-specific interfaces.	Improved testability and reusability.
DIP	Dependence on abstractions rather than implementations.	Reduced technical debt and increased adaptability.

The practical relevance of architectural approaches focused on modularity and loose coupling is supported by empirical software engineering research. Studies comparing production systems with different levels of architectural maturity report that projects exhibiting clear separation of concerns and abstraction-driven design achieve better out-

comes across key life-cycle metrics (Yanakiev I. et al., 2025; Alwi M. et al., 2025). In particular, reductions in change lead time, lower release failure rates, and shorter mean time to recovery have been observed. These findings align with widely used continuous delivery indicators – such as deployment frequency, lead time for changes, change failure rate, and mean time to restore – and indicate that architectural decisions conceptually aligned with SOLID positively affect system resilience and controllability in highly changeable environments.

Therefore, the implementation of SOLID principles in the backend design of media services has a wide-ranging effect on the product lifecycle, independent of software quality characteristics. This is because it establishes an architecturally sustainable trajectory, enables the faster delivery of changes, increases operational stability, and reduces operational costs. As complexity grows and functionalities are strained, SOLID principles serve not only as design guidelines but also as practical tools for managing architectural risk throughout the lifecycle of media products implemented in PHP and Python.

Conclusion

In the context of increasing media platform complexity and rising requirements for software system resilience, the SOLID principles represent an effective methodological framework for the design and evolution of backend architectures. Their application ensures modularity, loose coupling, and predictability in the behavior of software components, thereby improving scalability, testability, and maintainability. Applying these principles in PHP and Python backends, with available mechanisms for abstraction and typing, supports the development of adaptive software architectures. Therefore, consistent application of SOLID principles in backend development for media platforms enhances software architecture and promotes sustainable evolution.

References

Bogutskii, A. (2025) Enhancing the resilience of stream processing platforms: engineering approaches to scalability and fault tolerance. *ISJ Theoretical & Applied Science*, – 08(148). – P. 170–175.

Smirnov, A. (2025) Modern methods of backend system performance optimization: algorithmic, architectural, and infrastructural aspects. *International Journal of Advanced Research in Science, Communication and Technology*, – 5(3). – P. 262–266.

Silva, N., Rodrigues, E. and Conte, T. (2025) Evaluating Strategies for Teaching Micro Frontends: Do Anti-patterns Help? *Simpósio Brasileiro de Engenharia de Software (SBES)*, – P. 522–532.

Røhne, A. L., Pronk, B. and Akesson, B. (2024) Graph-based Anti-Pattern Detection in Microservice Applications. In: 2024 50th Euromicro Conference on Software Engineering and Advanced Applications (SEAA*)*. *IEEE*, – P. 341–349.

Ramachandrappa, N. C. (2024) SOLID Design Principles in Software Engineering. *International Journal of Computer Trends and Technology*, – 72(9). – P. 18–23. URL: https://doi.org/10.14445/22312803/ijctt-v72i9p104

Garifullin, R. (2025) Ethical aspects of personalizing web content using Artificial Intelligence. *International Journal of Scientific Research and Engineering Development*, – 8(4). – P. 1457–1460.

Sharma, S. (2024) Modern Backend Development Technologies: A Comparative Review and Case Study. *International Conference on Emerging Trends in Expert Applications & Security*, – P. 139–151.

Qiao, R. and Yin, Z. (2024) Construction of life cycle prediction model for digital media based on virtual reality. *International Journal of Mechatronics and Applied Mechanics*, – 15. – P. 28–34.

Yanakiev, I., Lazar, B. M. and Capiluppi, A. (2025) Applying SOLID principles for the refactoring of legacy code: an experience report. *Journal of Systems and Software*, – 220. – 112254 p. URL: https://doi.org/10.1016/j.jss.2024.112254

Alwi, M., Kajang, G. and Irwanto, M. M. (2025) Implementing Modular Layouts to Increase Scalability of Production Facilities. *Journal of the American Institute*, – 2(1). – P. 64–72. URL: https://doi.org/10.71364/r93vyj15

submitted 03.03.2026;
accepted for publication 17.03.2026;
published 30.04.2026

Contact: g_andreev@outlook.com