Notifier

The project is used to notify different mock targets i.e. SMS or Email. The notify endpoint receives a Notification request which contains 5 parameters: sender, recipient, title, body and type. The type of parameter is used to decide which mock target to perform.

1. If you were to review the following code, what feedback would you give? Please be specific and indicate any errors that might occur as well as other best practices and code refactoring that should be done.

The code has several issues affecting its maintainability and scalability. It violates several SOLID principles, making it difficult to extend and maintain. The Single Responsibility Principle (SRP) is broken because SmsNotifier and EmailNotifier handle both notification logic and the instantiation of their respective providers (SmsVendor and EmailVendor), while the providers themselves contain console printing logic instead of encapsulating it in an external provider interface. The Open/Closed Principle (OCP) is violated since adding a new notification type requires modifying the endpoint, making the system non-scalable—for instance, adding support for WhatsApp or Calls would require modifying the if condition in the endpoint. The Dependency Inversion Principle (DIP) is not followed because SmsNotifier and EmailNotifier directly depend on concrete classes (SmsVendor and EmailVendor) instead of interfaces, and there is no dependency injection, making the code hard to test and modify.

Additionally, the code has high coupling and scalability issues as MapPost("/notify", ...) directly handles notification instantiation, requiring API modifications for any changes. There is no centralized mechanism for managing notifications, and adding new channels would require modifying multiple code sections. The lack of dependency injection further complicates testing and maintenance since SmsNotifier and EmailNotifier instantiate SmsVendor and EmailVendor instead of receiving them as injected dependencies, and no dependency injection container is used.

The code also suffers from repetitive and non-reusable implementations, as SmsNotifier and EmailNotifier have similar Notify methods, indicating the need for a strategy or factory pattern. There is no clear separation between infrastructure and business logic. Furthermore, the lack of error handling and standardized responses means there is no proper exception handling if an error occurs, and the API lacks a consistent response structure, leading to inconsistencies for clients. Finally, the absence of unit testing makes it difficult to validate notification dispatch, and the tight coupling to SmsVendor and EmailVendor complicates test creation without real dependencies.

1. Rewrite the code by using the SOLID principles and hexagonal architecture. Please include unit tests on your code.
2. By using the open/close principle add two mock targets like WhatsApp and Call.