

Table of Contents

| | |
|--|---|
| Integrating with External Systems (15 pages) | 1 |
| Technical Requirements | 1 |
| Implementing consumer-driven contracts | 1 |
| Exposing a REST-based API | 1 |
| Exposing an events-based API | 1 |
| Implementing an Anti-Corruption Layer | 1 |
| Legacy Application Migration Patterns | 1 |

Integrating with External Systems (15 pages)

Wholeness is not achieved by cutting off a portion of one's being, but by integration of the contraries.

— Carl Jung

Thus far, we have used DDD to implement a robust core for our application. However, most bounded contexts usually have both upstream and downstream dependencies which usually change at a pace which is different from these core components. To maintain both agility and reliability and enable loose coupling, it is important to create what DDD calls the anti-corruption layer in order to shield the core from everything that surrounds it. In this chapter, we will look at integrating with a legacy Inventory Management system. We will round off by looking at common patterns when integrating with legacy applications.

Technical Requirements

Implementing consumer-driven contracts

Exposing a REST-based API

Currently, we have a set of commands and an ability to handle them. However, there isn't an entry point to invoke these commands externally. Let's expose these via a RESTful interface.

Exposing an events-based API

Implementing an Anti-Corruption Layer

Legacy Application Migration Patterns