Photo by Meriç Dağlı on Unsplash

TDD and Hexagonal Architecture in Microservices

Valentina Cupać
Founder & Technical Coach @ Optivem

About the speaker

Valentina Cupać coaches development teams in TDD & Clean Architecture to increase quality, accelerate delivery and scale teams.

Previously, she worked as a Senior Developer, Technical Lead & Solutions Architect.

Graduated from University of Sydney - Computer Science, Maths and Finance.

I write articles about TDD & Clean Architecture.

Connect with me or follow me:

LinkedIn: linkedin.com/in/valentinacupac
YouTube: youtube.com/@valentinacupac
Twitter: twitter.com/valentinacupac
GitHub: github.com/valentinacupac

Agenda

1. Hexagonal Architecture - Designing Testable Microservices using Hexagonal Architecture
2. Test Automation - Testing Microservices using Unit Testing, Integration Testing, Component Testing
3. TDD & Microservices - Applying TDD in Microservices with Hexagonal Architecture
4. Code Demo - Banking Kata on GitHub (Java)

Photo by Joel Filipe on Unsplash

1. Hexagonal Architecture

Foundational Overview

Hexagonal Architecture - Motivation

“Create your application to work **without either a UI or a database** so you can run automated regression-tests against the application, work when the database becomes unavailable, and link applications together without any user involvement.”

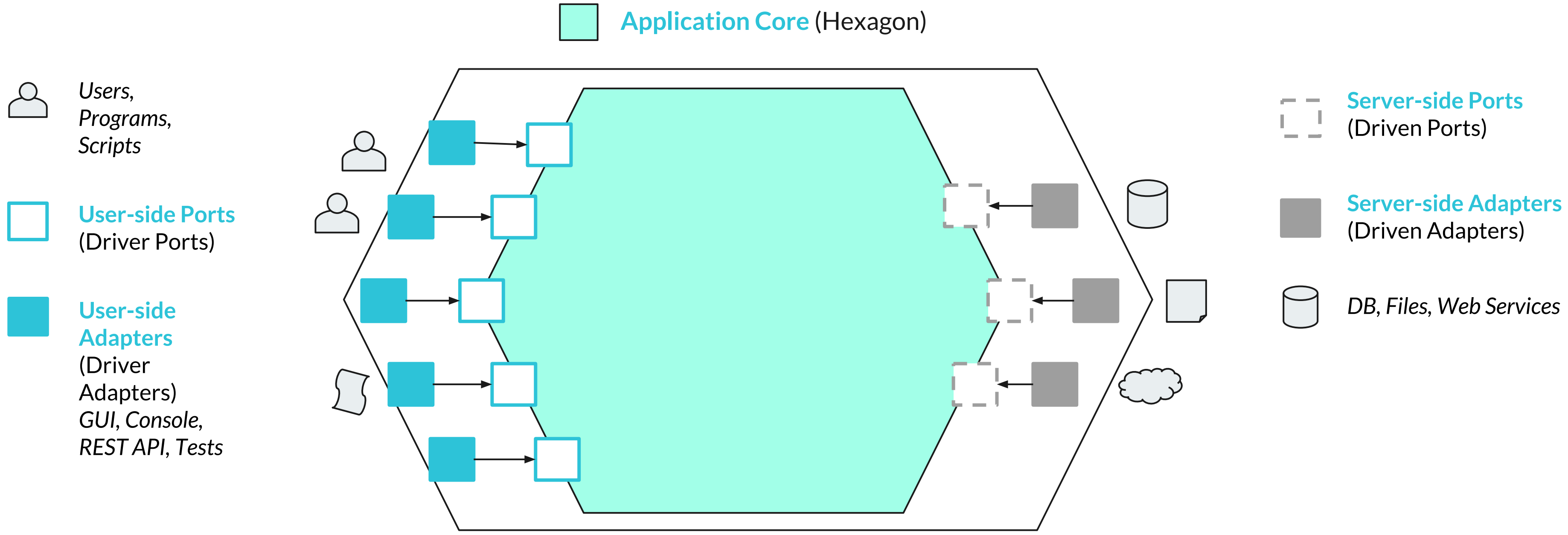
“Allow an application to **equally be driven by users, programs, automated test** or batch scripts, and to be **developed and tested in isolation** from its eventual run-time devices and databases.”

- Alistair Cockburn

https://alistair.cockburn.us/hexagonal-architecture/

Hexagonal Architecture

Adapted from https://alistair.cockburn.us/hexagonal-architecture/


Application Core (Hexagon)
Users, Programs, Scripts
User-side Ports (Driver Ports)
User-side Adapters (Driver Adapters) GUI, Console, REST API, Tests
Server-side Ports (Driven Ports)
Server-side Adapters (Driven Adapters)
DB, Files, Web Services

Foundations - TDD, Hexagonal & Clean Architecture

TDD and Clean Architecture - Driven by Behaviour
Hosted by: Java User Group Switzerland & Software Crafts Romandie Community
https://www.youtube.com/watch?v=3wxiQB2-m2k

TDD and Clean Architecture - Use Case Driven Development
Hosted by: Software Craftsmanship Luxembourg
https://www.youtube.com/watch?v=IZWLnn2fNko

TDD and Clean Architecture - Use Case Driven and Domain Driven Design
Hosted by: Ticino Software Craft
https://www.youtube.com/watch?v=UubZZOPP500

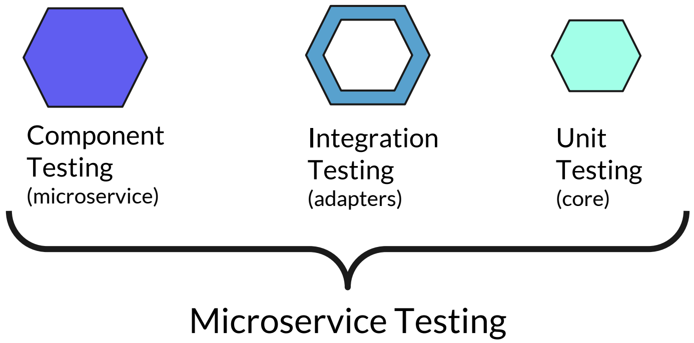
TDD in Hexagonal Architecture and Clean Architecture
Hosted by: Tech Excellence
https://www.youtube.com/watch?v=WAoqGzVDHc0

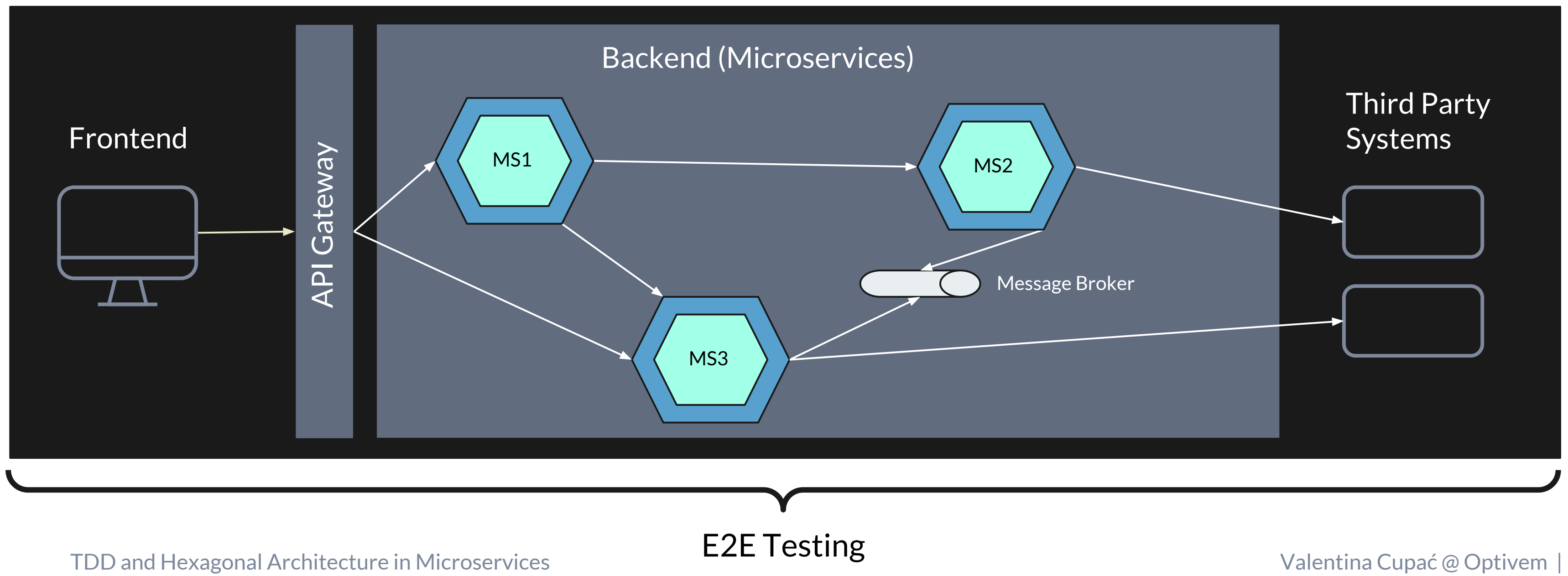
Photo by Meriç Dağlı on Unsplash

2. Test Automation

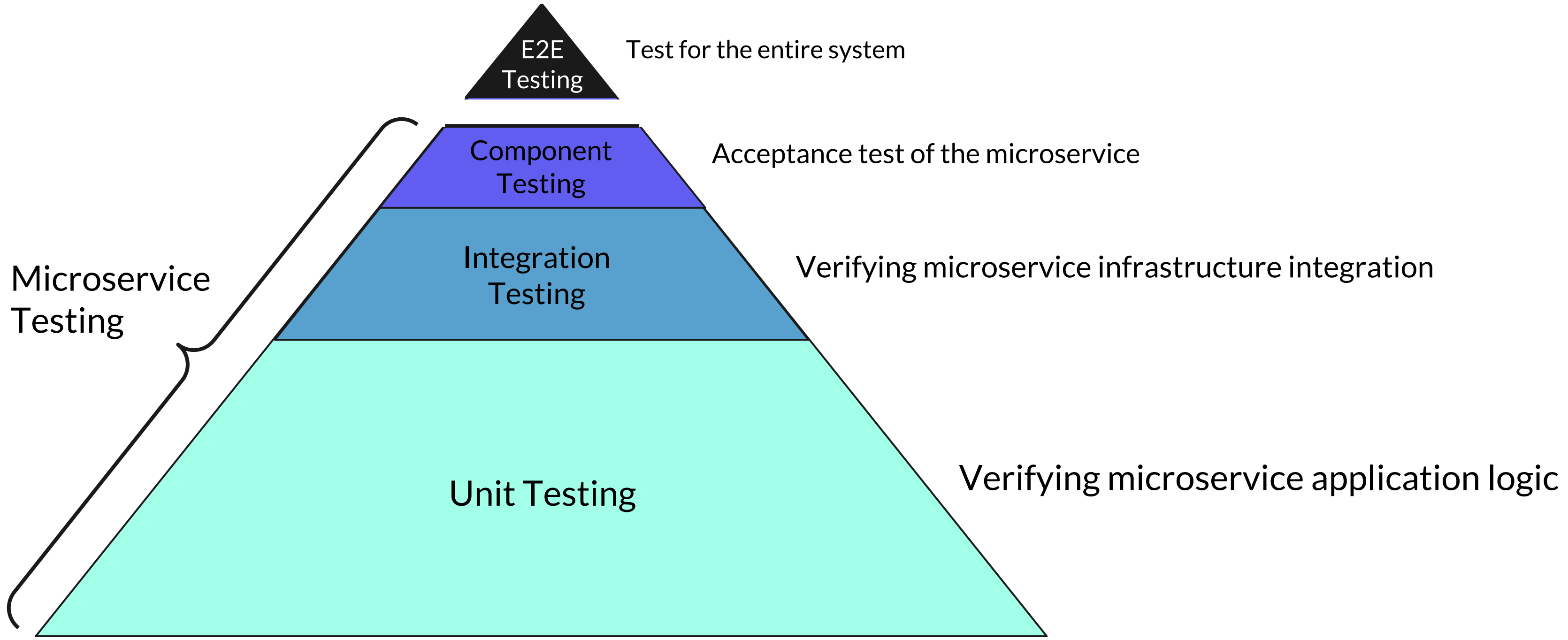
Test Pyramid & Deployment Pipeline

Microservice Architecture


Component Testing (microservice)
Integration Testing (adapters)
Unit Testing (core)
Microservice Testing



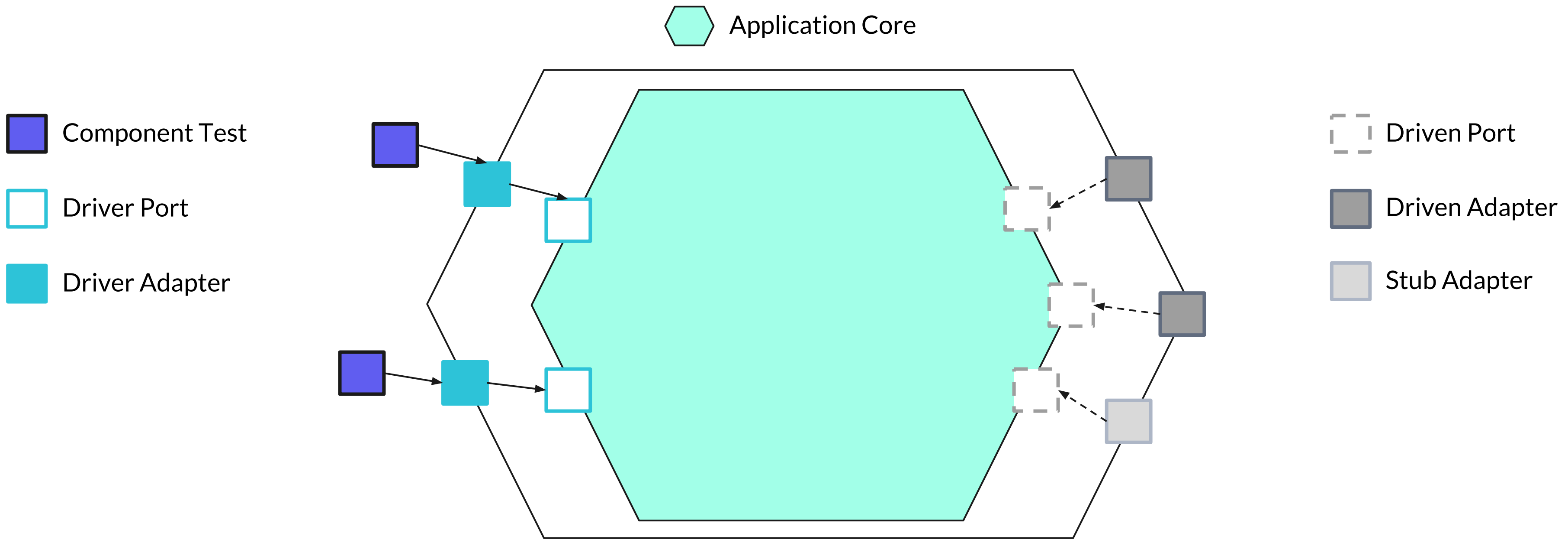
Test Pyramid


E2E Testing
Test for the entire system
Component Testing
Acceptance test of the microservice
Integration Testing
Verifying microservice infrastructure integration
Unit Testing
Verifying microservice application logic
Microservice Testing

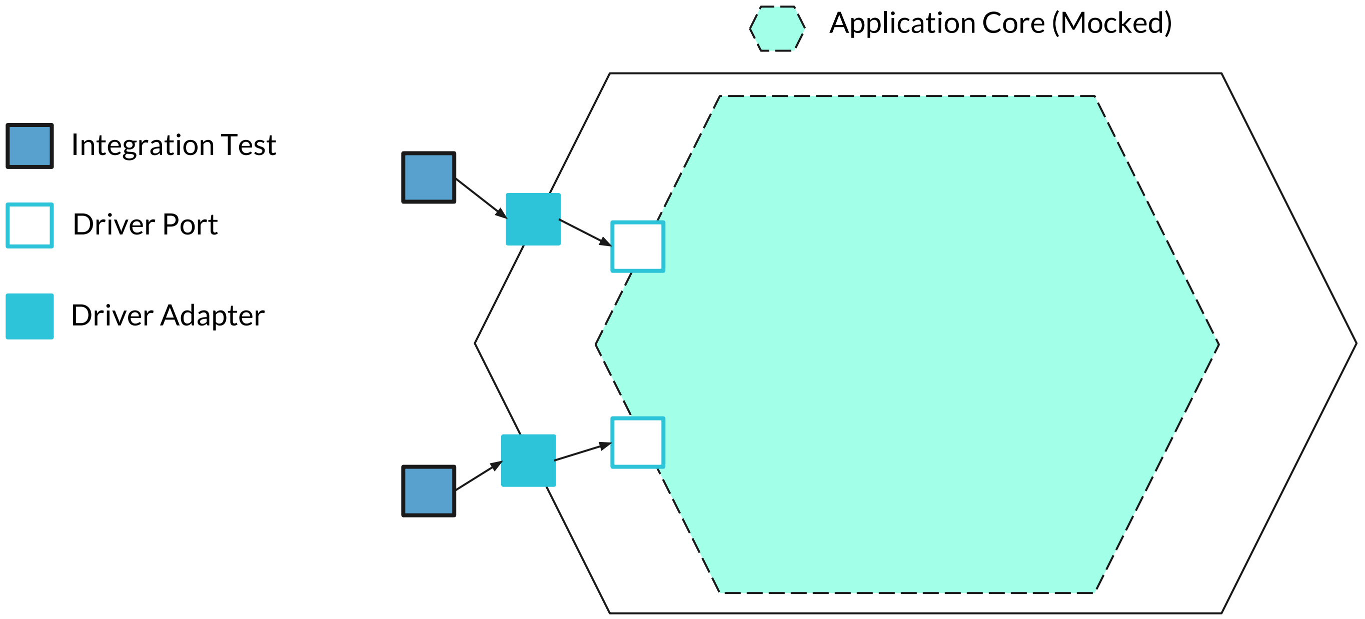
Component Testing

In-Process Component Testing
→ use only Stub Adapters.

Out-of-process Component Testing
→ use (Real) Driven Adapters for Infrastructure Services & Stub Adapters for other microservices & third party systems.



Integration Testing - Driver Side

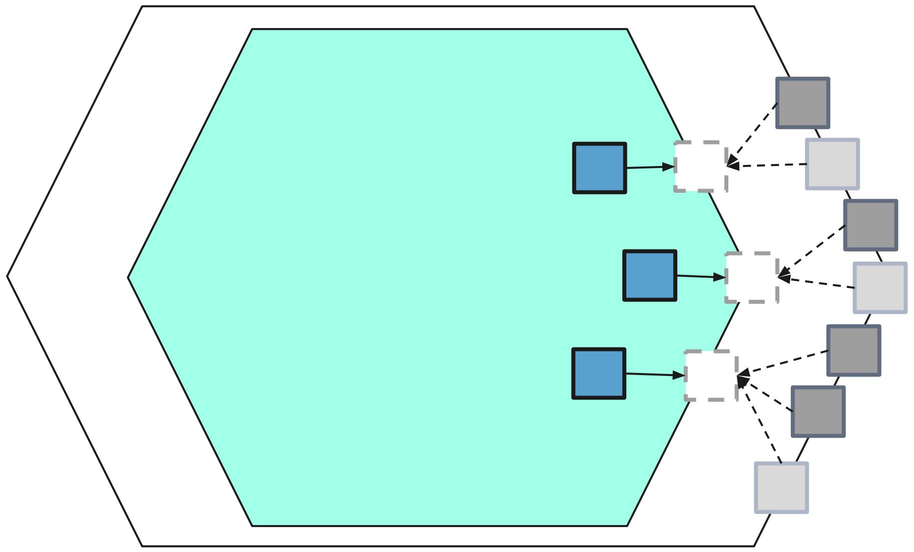


Driver Adapters
HTTP API
Message Consumer

Integration Tests
Test the Driver Adapters by mocking out the Driver Ports (Use Cases) to simulate various scenarios

Integration Testing - Driven Side

⬡ Application Core



🟦 Integration Test

⬚ Driven Port

⬛ Driven Adapter

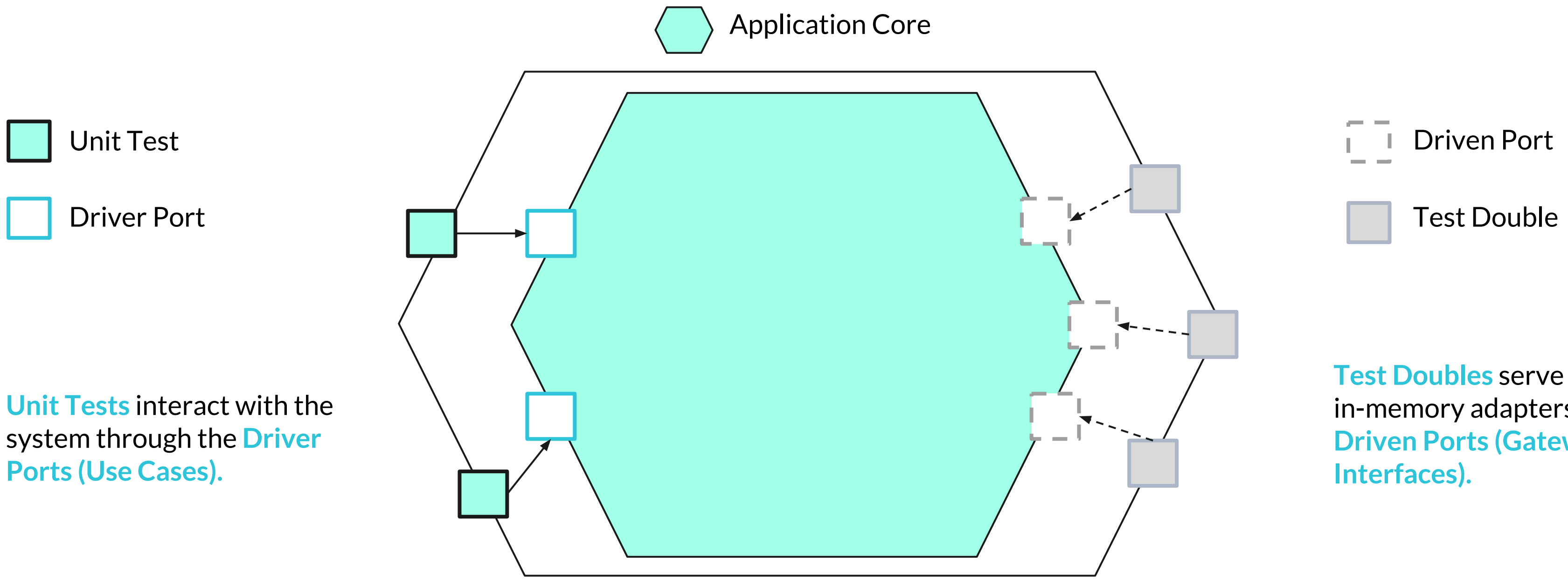
⬜ Test Double

Driven Adapters:
HTTP Client, FTP Client, SMTP Client, DB Client, File System Message Publisher, System Clock, Random Number Generator

Integration Tests:
Target the Driven Ports, executed against the Driven Adapters & Fake Adapters

Note: This diagram illustrates the approach of using **Sociable Unit Tests** for **Driver Ports**.

Unit Testing



Unit Tests interact with the system through the Driver Ports (Use Cases).

Test Doubles serve as in-memory adapters for the Driven Ports (Gateway Interfaces).

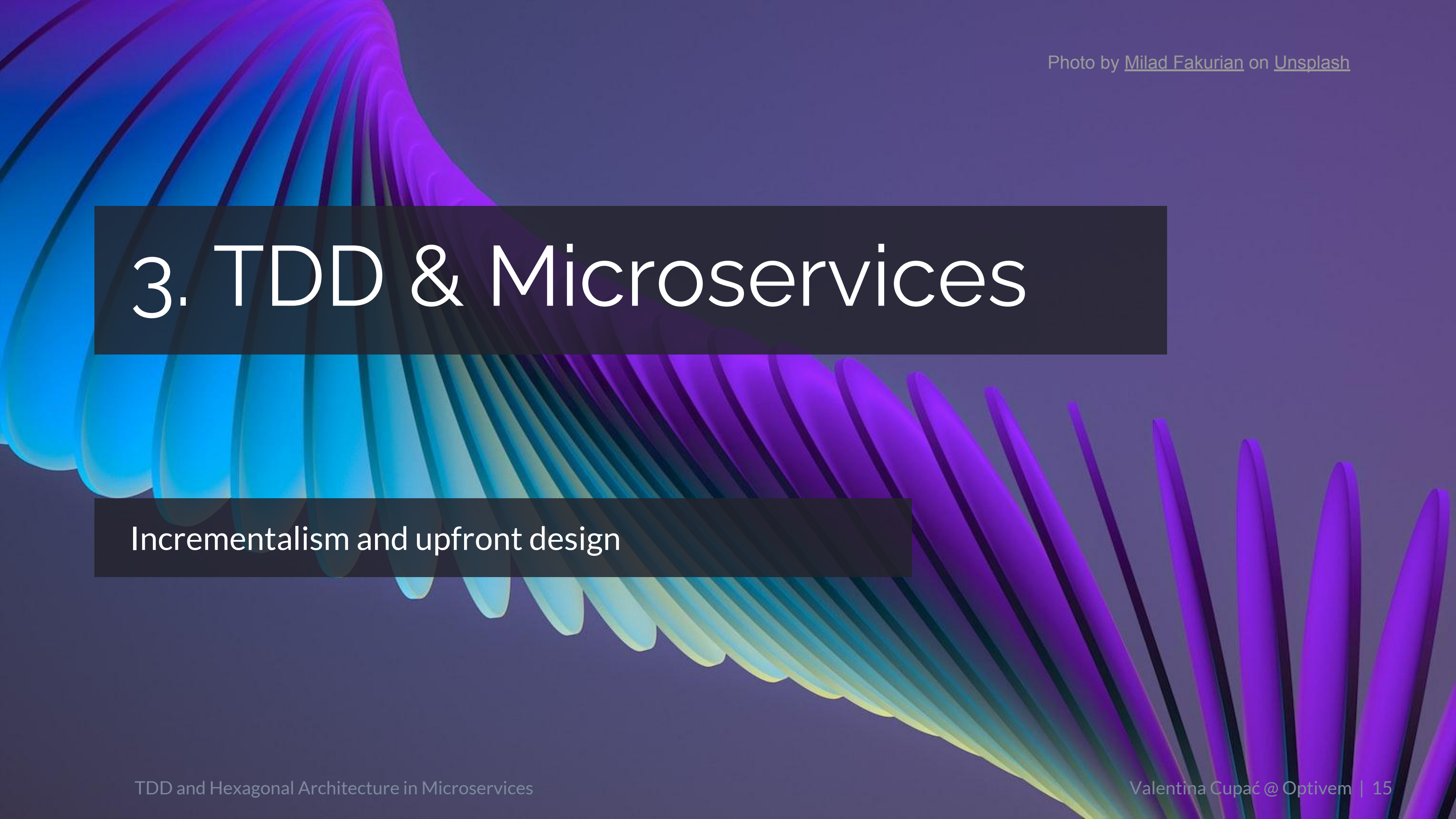


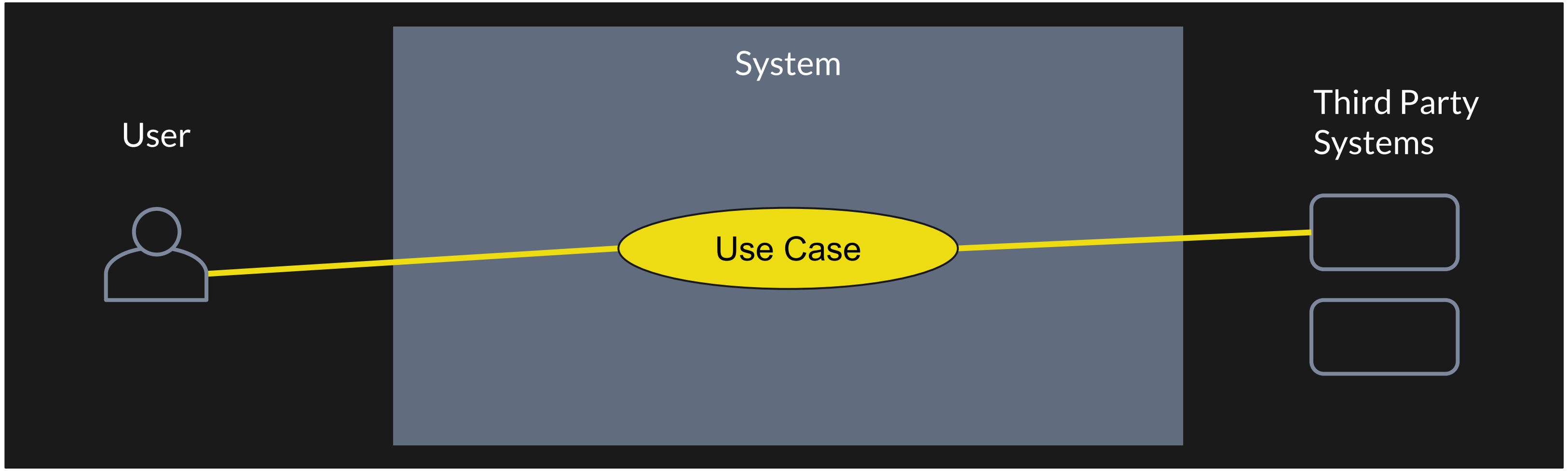
Photo by Milad Fakurian on Unsplash

3. TDD & Microservices

Incrementalism and upfront design

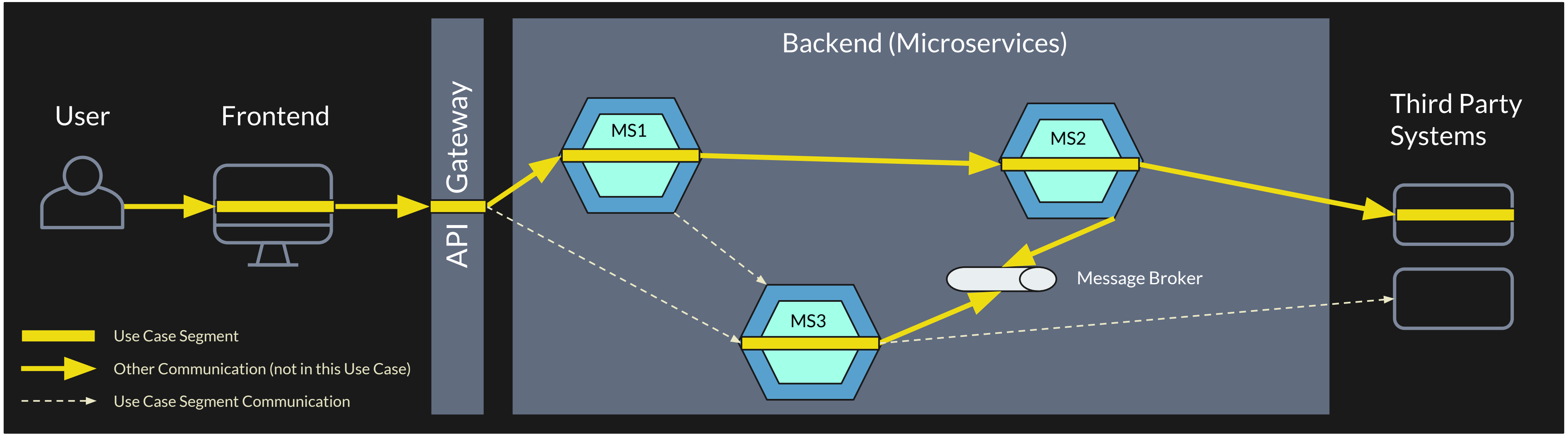
Step 1. Use Case Diagram & Use Case Narrative

Create a Use Case Diagram & write the Use Case Narrative, so that we can understand the interactions between Primary Actors (User), the System, and Secondary Actors (Third Party Systems). *Our System is a black box, we do not think about frontend/backend nor microservices.*



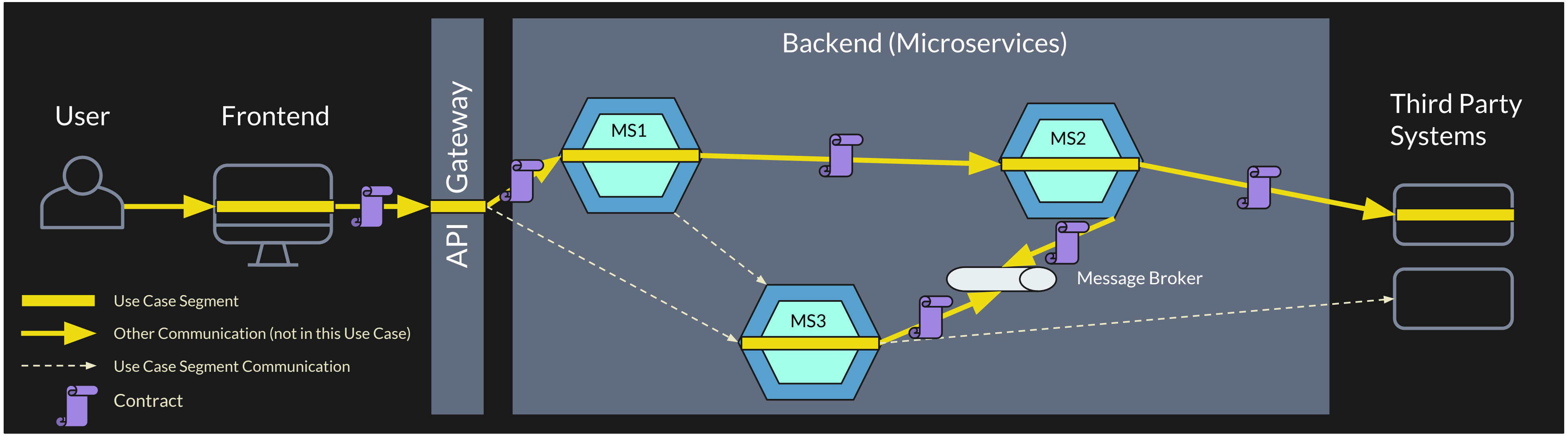
Step 2. Use Case Decomposition - FE & BE

Zoom into the System itself, and decompose the Use Case to identify the responsibilities and interactions between Frontend and Backend. Zoom into the Backend Microservices to identify how the Use Case will be decomposed across the Microservices.



Step 3. Use Case Decomposition - Contract Tests

Create the contracts for the “break points” in the Use Case Decomposition - between Frontend & API Gateway, API Gateway & Microservices, between Microservices themselves and between Microservices & Third Party Systems. These contracts are for the request/response and messages.



Step 4. Parallel Team Development with TDD

Each team works in parallel using TDD. Frontend Team develops the frontend, and each Microservice Team can develop their Microservice using TDD. Each Microservice is tested and deployed in isolation, hence each team can work incrementally without being blocked by other teams.

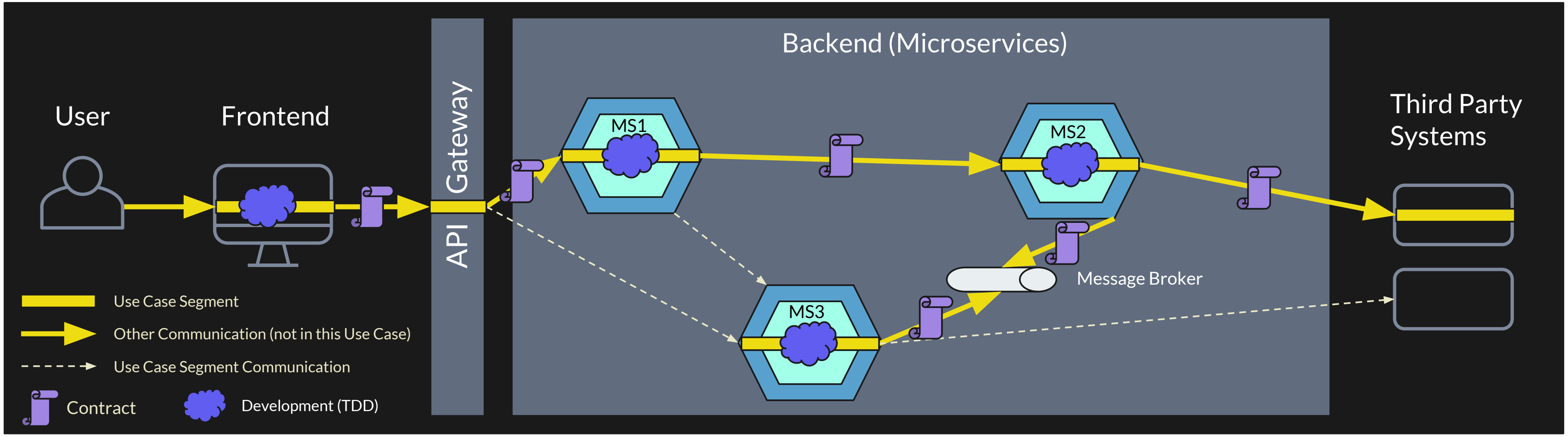


Photo by Ash Edmonds on Unsplash

4. Code Demo

Banking Kata on GitHub (Java)

GitHub Code Demo

The following open source GitHub projects illustrate **TDD & Clean Architecture** with a **Use Case Driven Development** (UCDD) and **Domain Driven Design (DDD)** approach. They show an incremental and iterative approach to implementing use cases with a robust test suite - by primarily coupling tests to use cases.

Banking Kata (Java) https://github.com/valentinacupac/banking-kata-java

Banking Kata (.NET) https://github.com/valentinacupac/banking-kata-dotnet

I am continuing development on these projects; you can **follow me on GitHub** https://github.com/valentinacupac to get further updates. You're welcome to **contribute**, see the README.md file. *Feel free to contact me if you have any questions, feedback or suggestions regarding these demo projects.*

Conclusion

Hexagonal Architecture helps us develop and test our application in isolation from external technology concerns - in isolation from UI & DB.

Microservice Architecture helps us rapidly deliver large and complex applications by splitting them into independently testable & deployable services organized by business capabilities and developed by small teams.

Each Microservice is independently testable through Unit Tests, Integration Tests and Component Tests, which are executed on the Microservice's Pipeline. We have very few E2E Tests which span the entire system. ✓

Benefits are higher testability resulting in lower maintenance costs → higher ROI

References

Hexagonal Architecture

Hexagonal Architecture (Alistair Cockburn)
https://alistair.cockburn.us/hexagonal-architecture/

Hexagonal Architecture (Juan Manuel Garrido de Paz)
https://jmgarridopaz.github.io/

Microservices & Hexagonal Architecture

Microservices (Chris Richardson)
https://microservices.io/

Microservices Patterns (Chris Richardson)
https://microservices.io/book

Photo by Mahdi Soheili on Unsplash

Thank You

Valentina Cupać @ Optivem
Founder Technical Coach

E valentina.cupac@optivem.com
W www.optivem.com

Connect on: LinkedIn | Twitter | YouTube | GitHub | Instagram