Clean Architecture

Agenda

- Introduction
- Key Principles
- Pros & Cons
- Solution Demo
- Conclusion

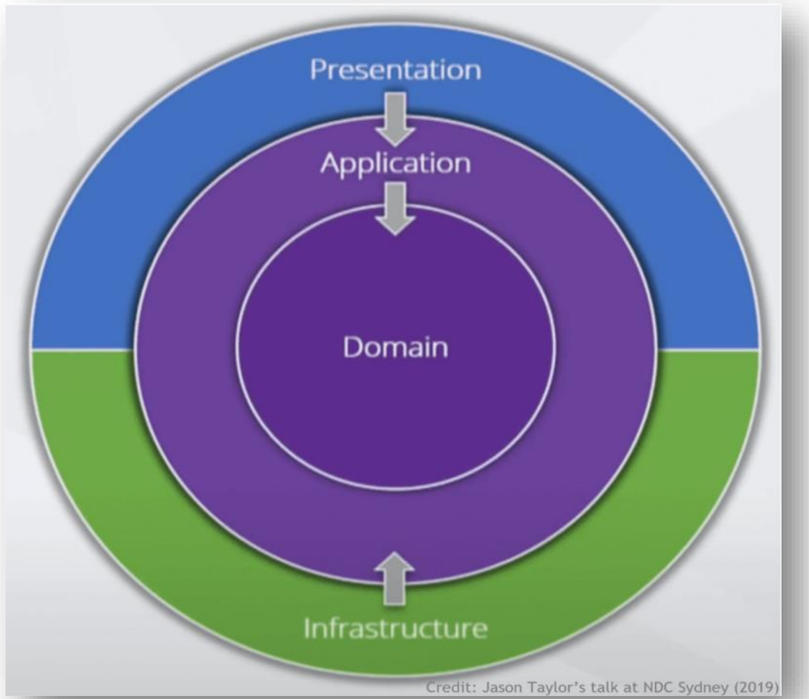
Introduction

The architecture of a software system is the shape given to that system by those who build it. The form of that shape is in the division of that system into components, the arrangement of those components, and the ways in which those components communicate with each other.

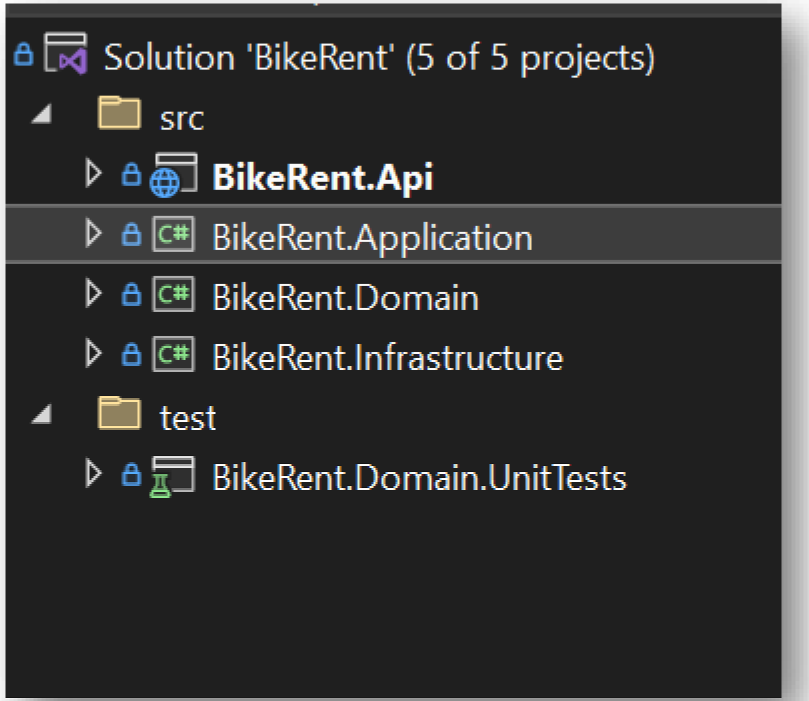
What is clean architecture?

The clean architecture, is a way to structure software systems to achieve separation of concerns, improve testability, and maintainability. The core idea is to organize the system in such a way that the business logic is independent of frameworks, UI, databases, and other externalities. This is achieved by organizing code into layers and defining clear boundaries and dependencies.

Introduction



=


Solution 'BikeRent' (5 of 5 projects)
src
BikeRent.Api
BikeRent.Application
BikeRent.Domain
BikeRent.Infrastructure
test
BikeRent.Domain.UnitTests

Clean Architecture: Layers

Presentation layer - responsible for the user interface and user interactions.

Application layer - responsible for the use cases and business logic. It includes services and use case classes.

Infrastructure layer - responsible for the interface adapters and frameworks. It includes database access, web services, and other external systems.

Domain layer - responsible for the entities and domain logic. It includes entities, domain services, and domain events.

Key Principles: Separation of Concerns

Clean Architecture enforces a clear separation between the core business logic and external details.

Layers have distinct responsibilities: the innermost layer holds essential business rules, while outer layers handle technical implementation and delivery mechanisms.

Key Principles: Dependency Rule

Dependencies flow inward towards the core business logic.

High-level modules are independent of low-level modules, ensuring that changes in external components do not impact the core business logic.

Key Principles: Testability

Encourages writing unit tests for business logic independently of external dependencies.

Allows for efficient and comprehensive testing of core functionality without needing integration tests for minor changes.

Key Principles: Platform Independence

Ensures business logic is independent of the implementation framework or platform.

Facilitates easy technology switches or adaptation to various platforms without rewriting core logic.

Pros and Cons

What are pros and cons of a Clean Architecture?

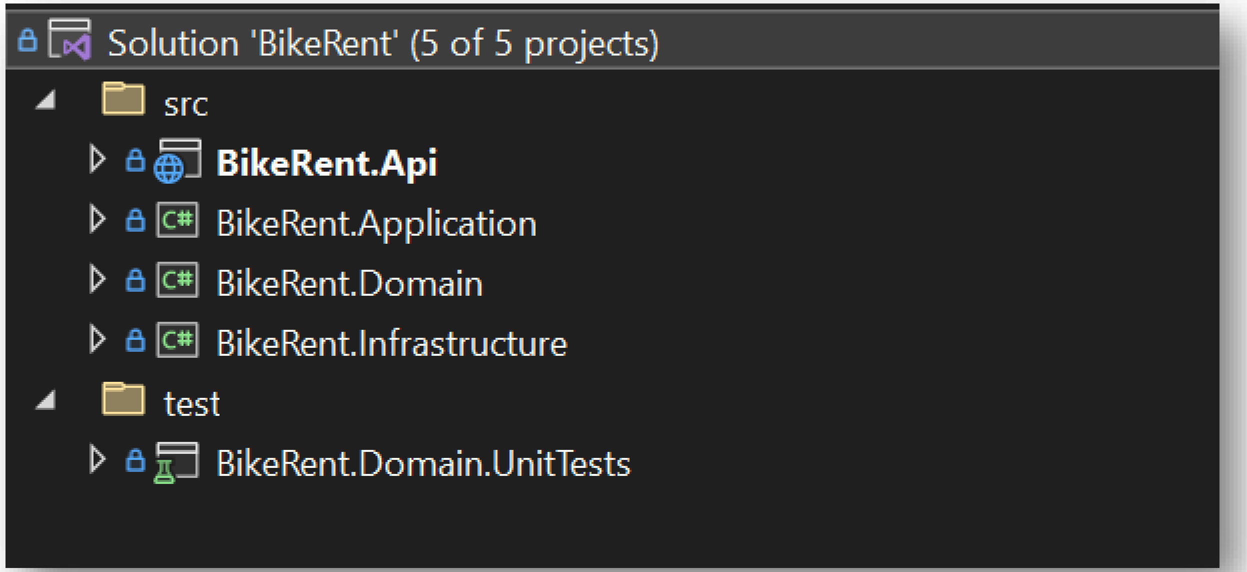
Pros

1. Separation of Concerns:
2. Flexibility:
3. Testability
4. Independence of Frameworks
5. Resilience to Change

Cons

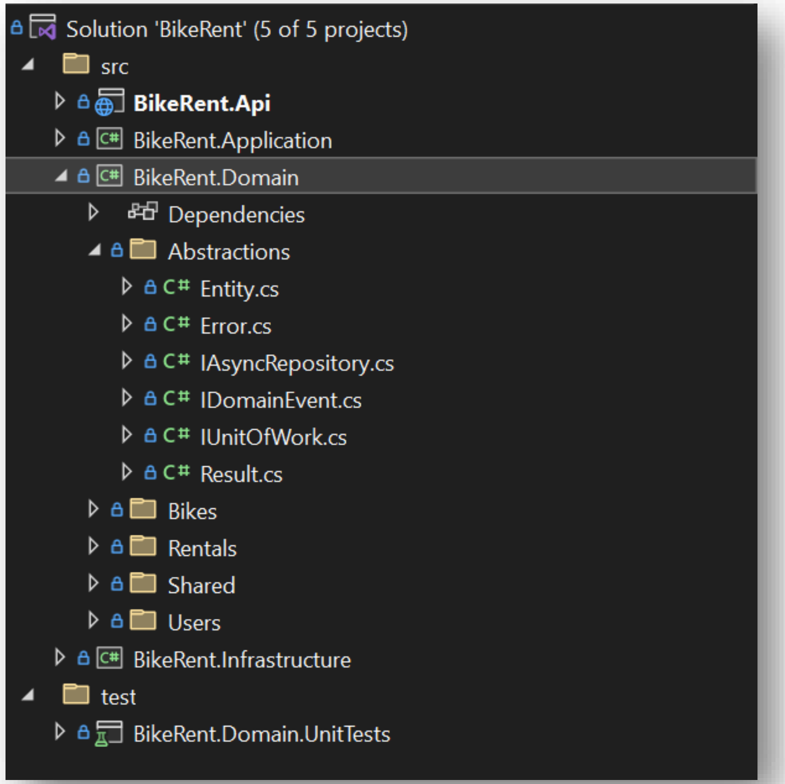
1. Complexity
2. Overhead
3. Learning Curve
4. Boilerplate Code
5. Initial Investment

Solution Demo



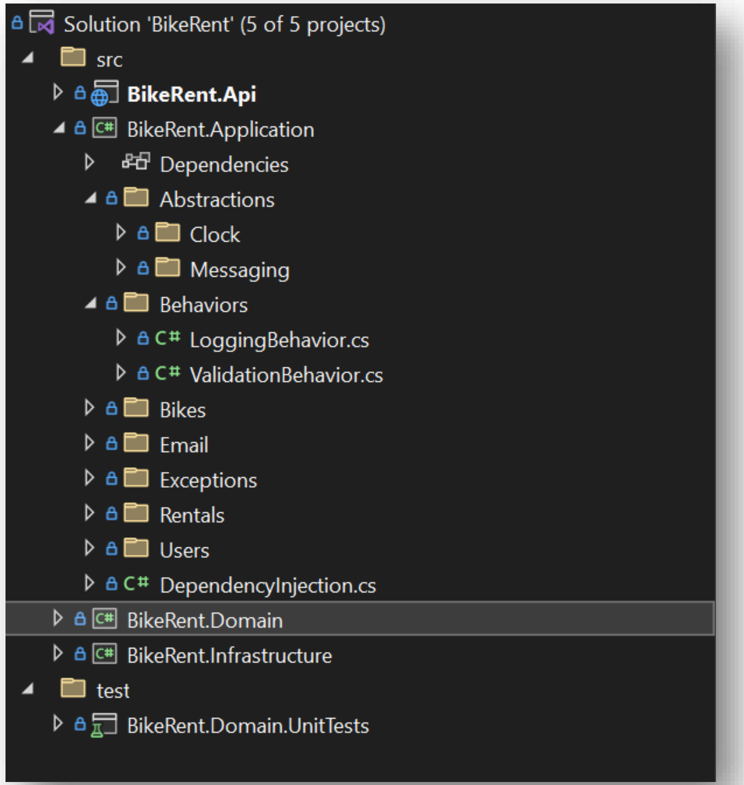
```
Solution 'BikeRent' (5 of 5 projects)
  src
    BikeRent.Api
    BikeRent.Application
    BikeRent.Domain
    BikeRent.Infrastructure
  test
    BikeRent.Domain.UnitTests
```

Domain Layer



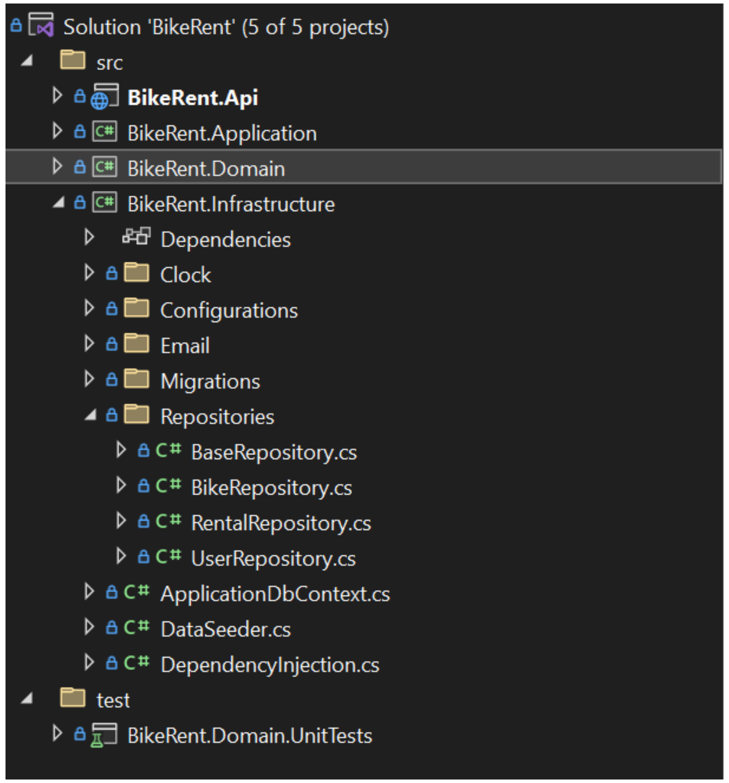
```
Solution 'BikeRent' (5 of 5 projects)
  src
    BikeRent.Api
    BikeRent.Application
    BikeRent.Domain
      Dependencies
      Abstractions
        Entity.cs
        Error.cs
        IAsyncRepository.cs
        IDomainEvent.cs
        IUnitOfWork.cs
        Result.cs
      Bikes
      Rentals
      Shared
      Users
    BikeRent.Infrastructure
  test
    BikeRent.Domain.UnitTests
```

Application Layer



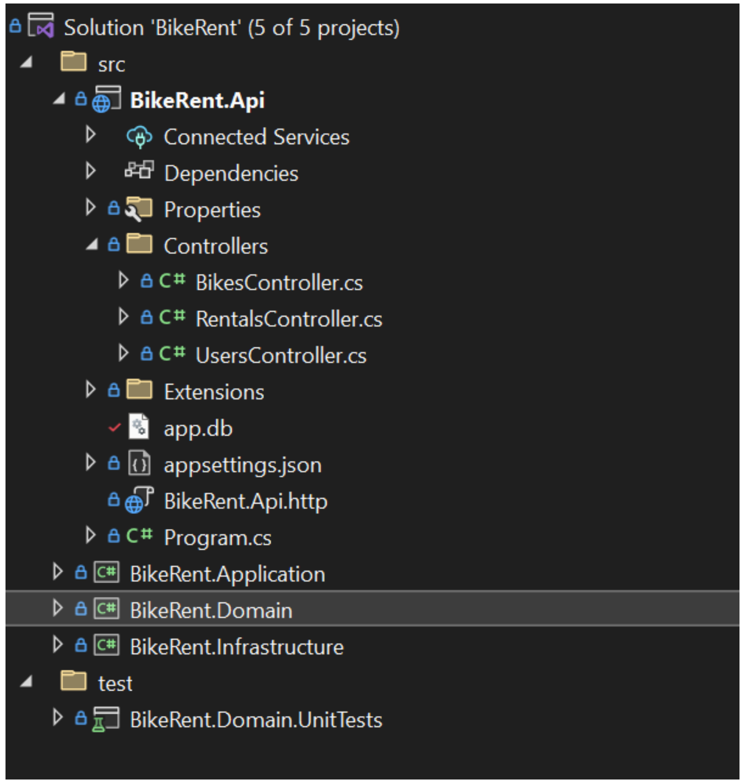
```
Solution 'BikeRent' (5 of 5 projects)
  src
    BikeRent.Api
    BikeRent.Application
      Dependencies
      Abstractions
        Clock
        Messaging
      Behaviors
        LoggingBehavior.cs
        ValidationBehavior.cs
      Bikes
      Email
      Exceptions
      Rentals
      Users
      DependencyInjection.cs
    BikeRent.Domain
    BikeRent.Infrastructure
  test
    BikeRent.Domain.UnitTests
```

Infrastructure Layer



```
Solution 'BikeRent' (5 of 5 projects)
  src
    BikeRent.Api
    BikeRent.Application
    BikeRent.Domain
    BikeRent.Infrastructure
      Dependencies
      Clock
      Configurations
      Email
      Migrations
      Repositories
        BaseRepository.cs
        BikeRepository.cs
        RentalRepository.cs
        UserRepository.cs
      ApplicationDbContext.cs
      DataSeeder.cs
      DependencyInjection.cs
  test
    BikeRent.Domain.UnitTests
```

Presentation Layer



```
Solution 'BikeRent' (5 of 5 projects)
  src
    BikeRent.Api
      Connected Services
      Dependencies
      Properties
      Controllers
        BikesController.cs
        RentalsController.cs
        UsersController.cs
      Extensions
      app.db
      appsettings.json
      BikeRent.Api.http
      Program.cs
    BikeRent.Application
    BikeRent.Domain
    BikeRent.Infrastructure
  test
    BikeRent.Domain.UnitTests
```

Conclusion

Clean Architecture is a valuable approach for building maintainable and adaptable software. It focuses on clear separation of concerns, framework independence, and easy testability, which makes it suitable for projects that need long-term reliability and flexibility. While it requires some initial effort, the long-term benefits are significant.

Repository

https://github.com/ilis08/BikeRent

Thank you

Illia Nankov

nankoviliya@gmail.com