Hexagonal Architecture

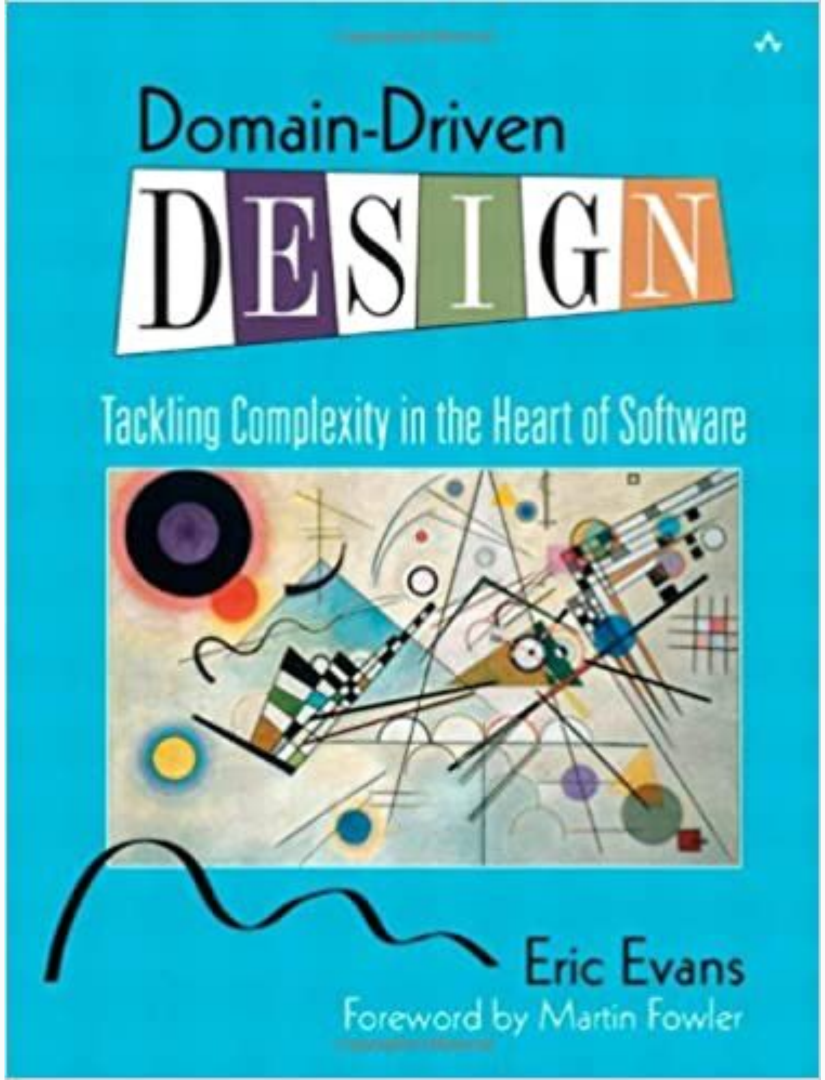
Murat Öz

- Domain-Driven Design
- N-Layered Architecture
- **Hexagonal Architecture**

Domain-Driven Design

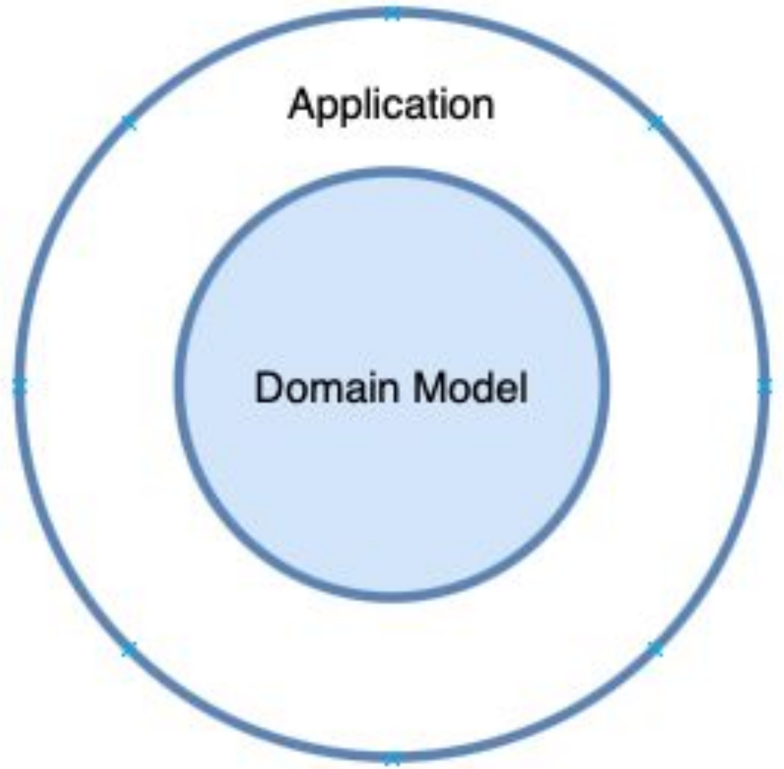
Domain-Driven Design

Eric Evans

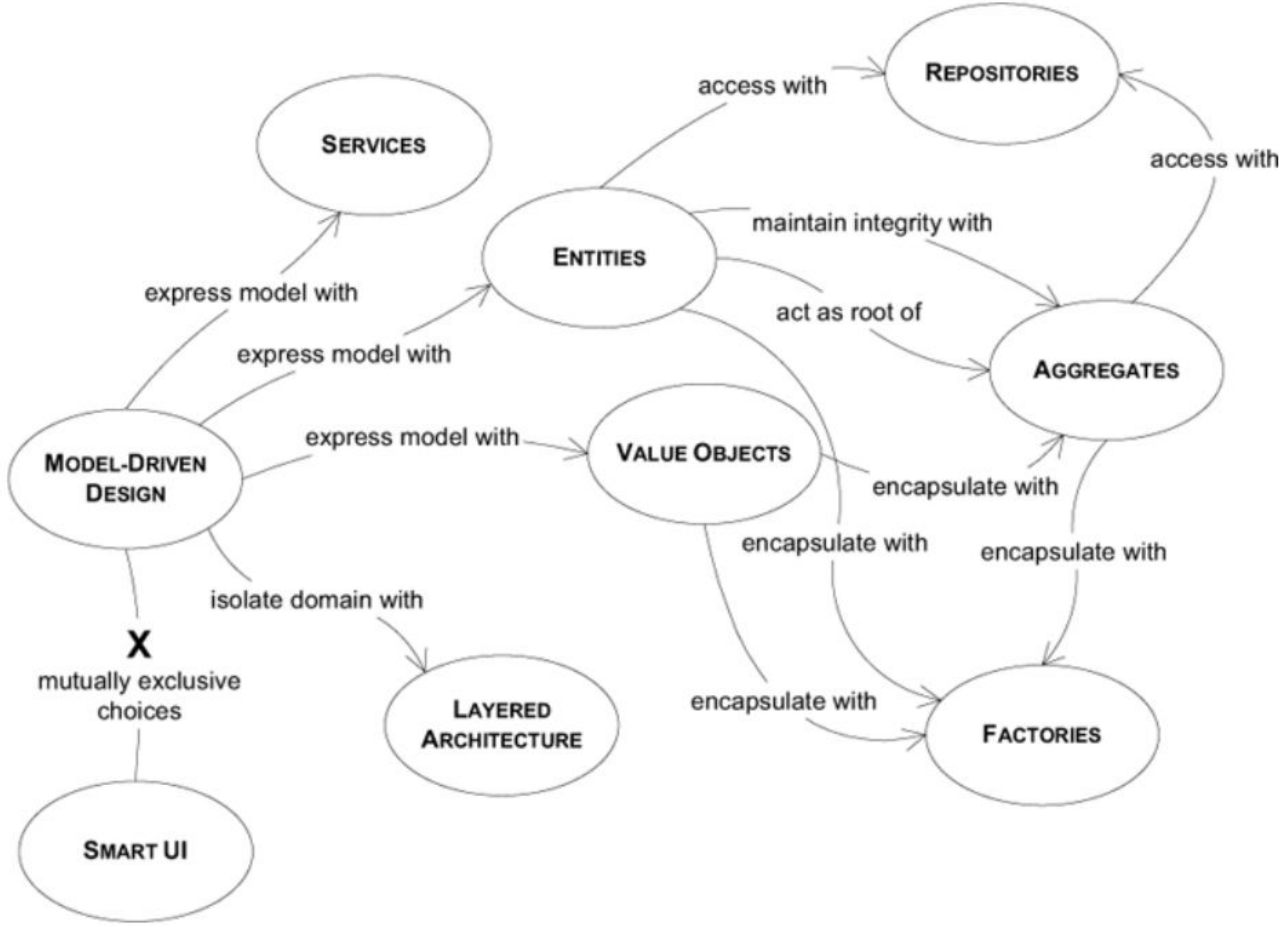

Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler

DOMAIN-DRIVEN DESIGN

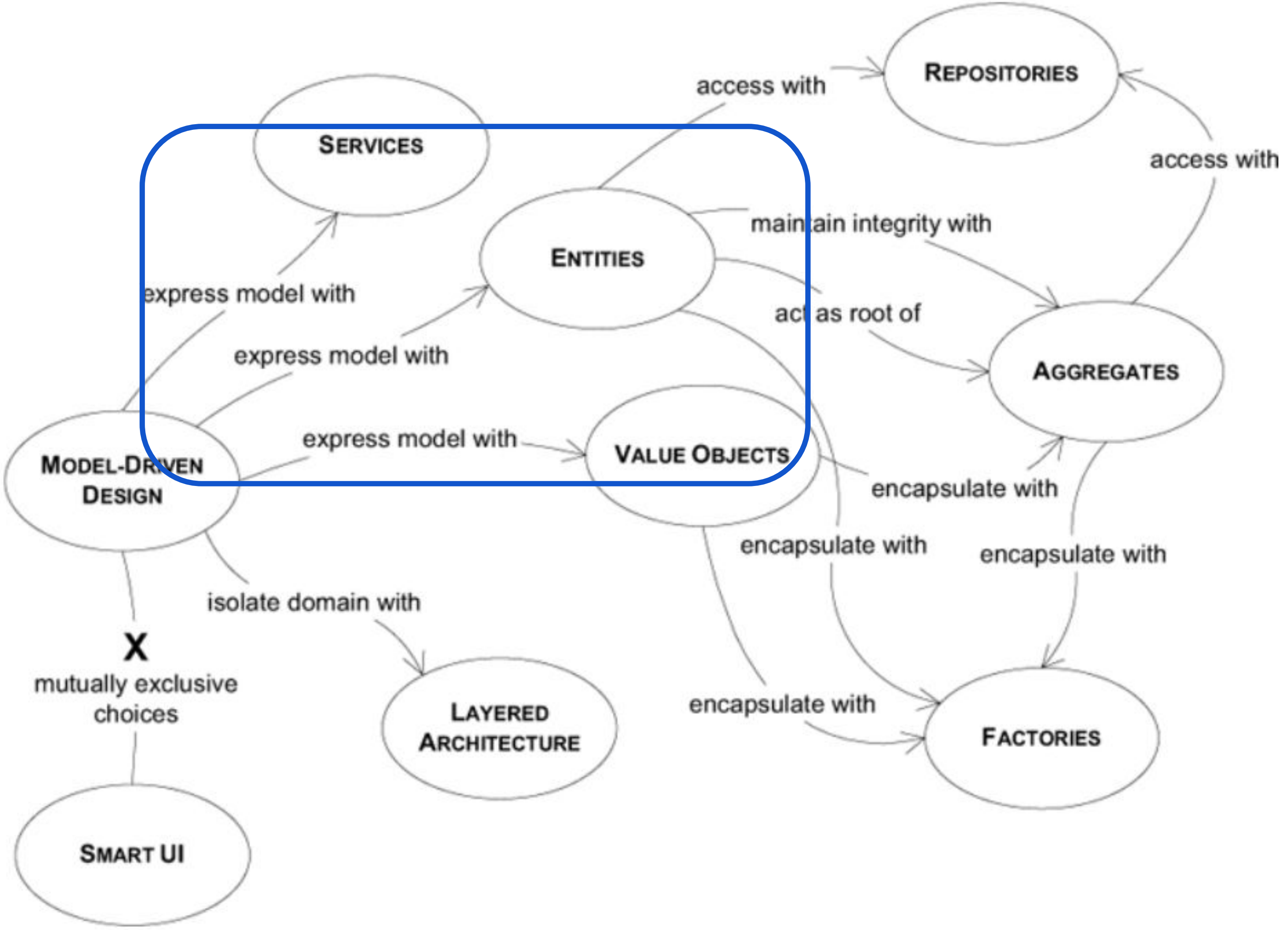
- Domain modellemesini sistemden soyutlamak



DOMAIN DRIVEN DESIGN



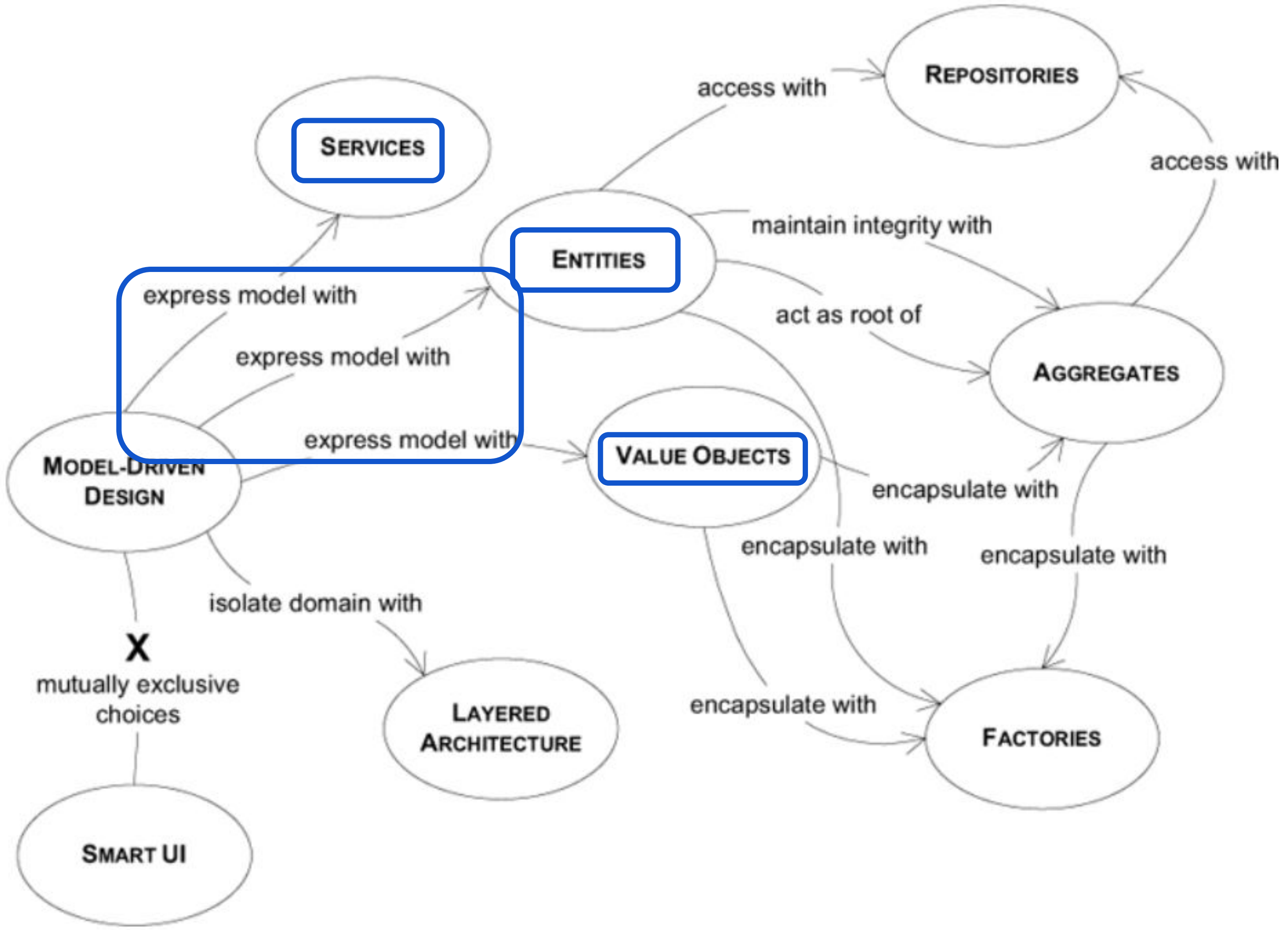
DOMAIN DRIVEN DESIGN



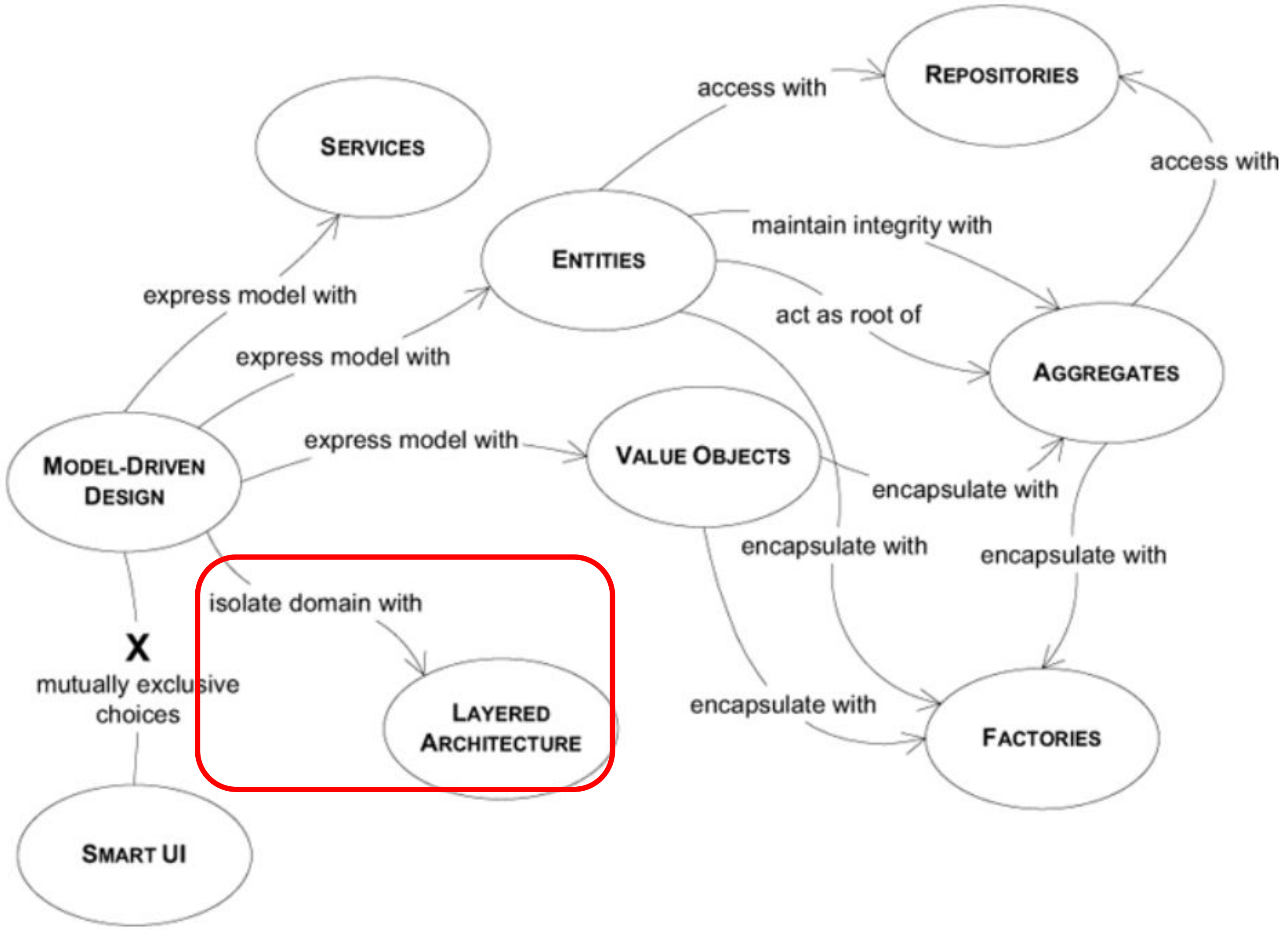
Domain Modeli Tanımlamak

DOMAIN DRIVEN DESIGN

Domain Modeli Tanımlamak

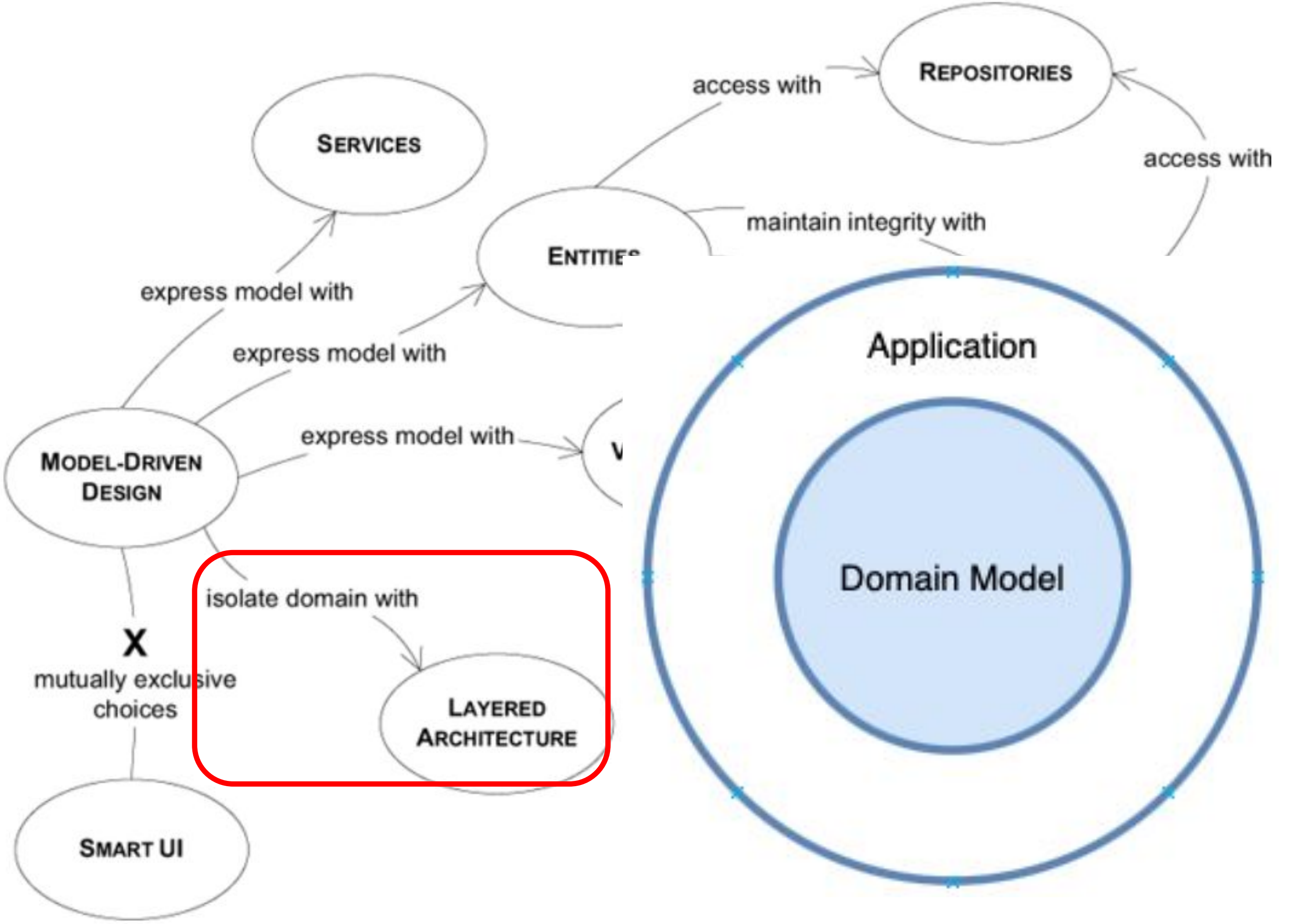


DOMAIN DRIVEN DESIGN



Katmanlı Mimari

DOMAIN DRIVEN DESIGN

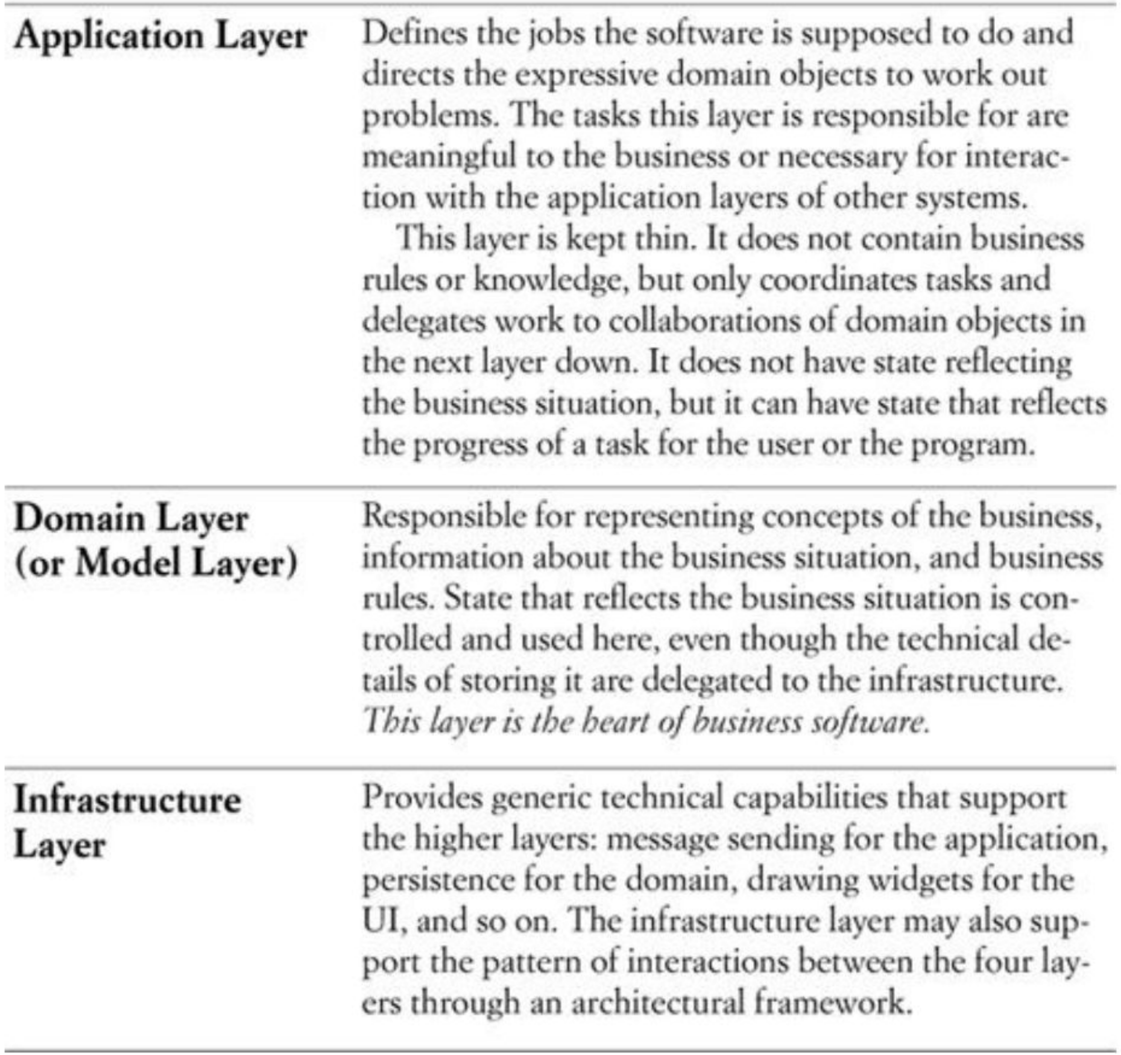


Katmanlı Mimari

Domain Abstraction

Layered Architecture

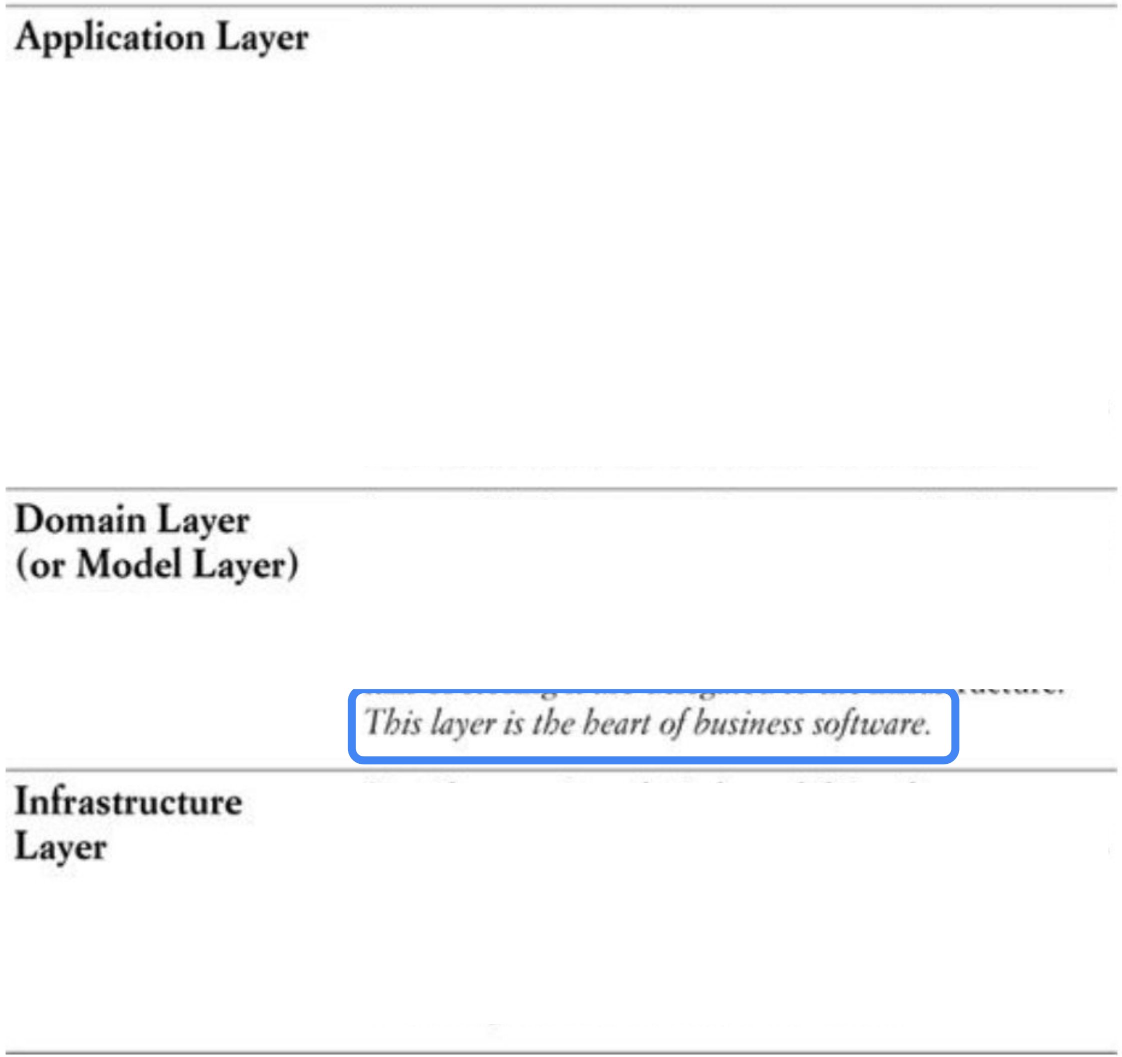
Source: Eric Evans - Domain Driven Design



Application Layer	Defines the jobs the software is supposed to do and directs the expressive domain objects to work out problems. The tasks this layer is responsible for are meaningful to the business or necessary for interaction with the application layers of other systems. This layer is kept thin. It does not contain business rules or knowledge, but only coordinates tasks and delegates work to collaborations of domain objects in the next layer down. It does not have state reflecting the business situation, but it can have state that reflects the progress of a task for the user or the program.
Domain Layer (or Model Layer)	Responsible for representing concepts of the business, information about the business situation, and business rules. State that reflects the business situation is controlled and used here, even though the technical details of storing it are delegated to the infrastructure. *This layer is the heart of business software.*
Infrastructure Layer	Provides generic technical capabilities that support the higher layers: message sending for the application, persistence for the domain, drawing widgets for the UI, and so on. The infrastructure layer may also support the pattern of interactions between the four layers through an architectural framework.

Domain Abstraction

Layered Architecture



Application Layer

Domain Layer (or Model Layer)

This layer is the heart of business software.

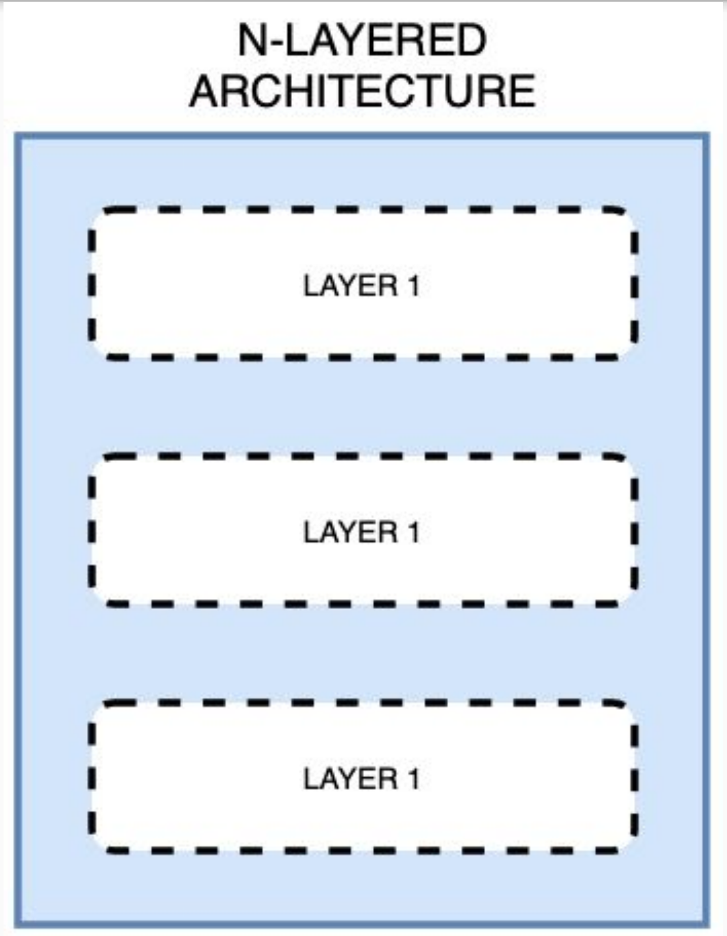
Infrastructure Layer

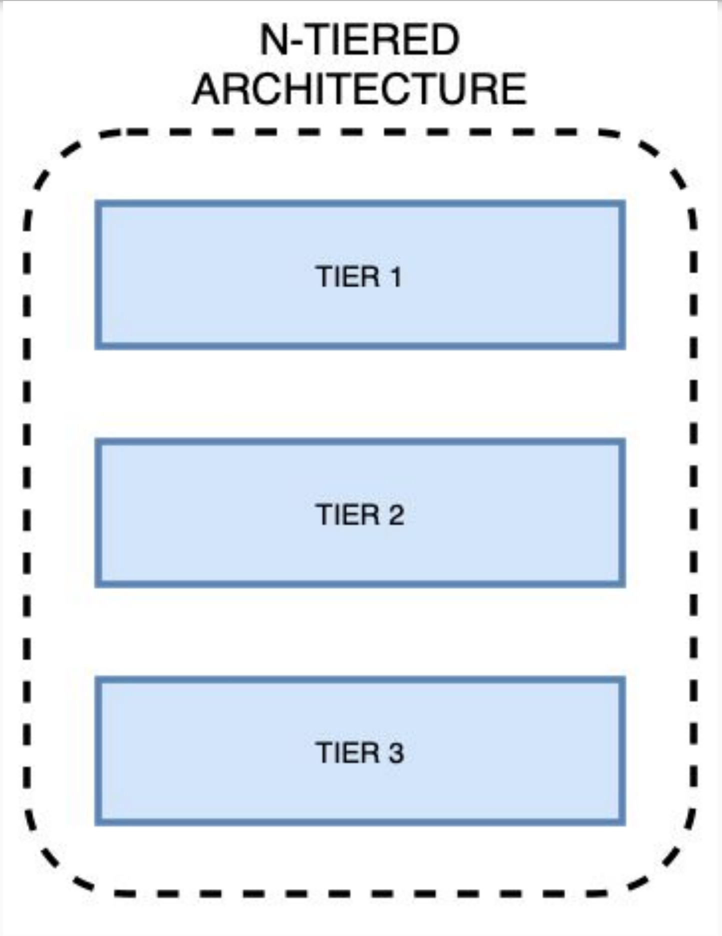
Source: Eric Evans - Domain Driven Design

N-Layered Architecture

Katmanlı Mimari

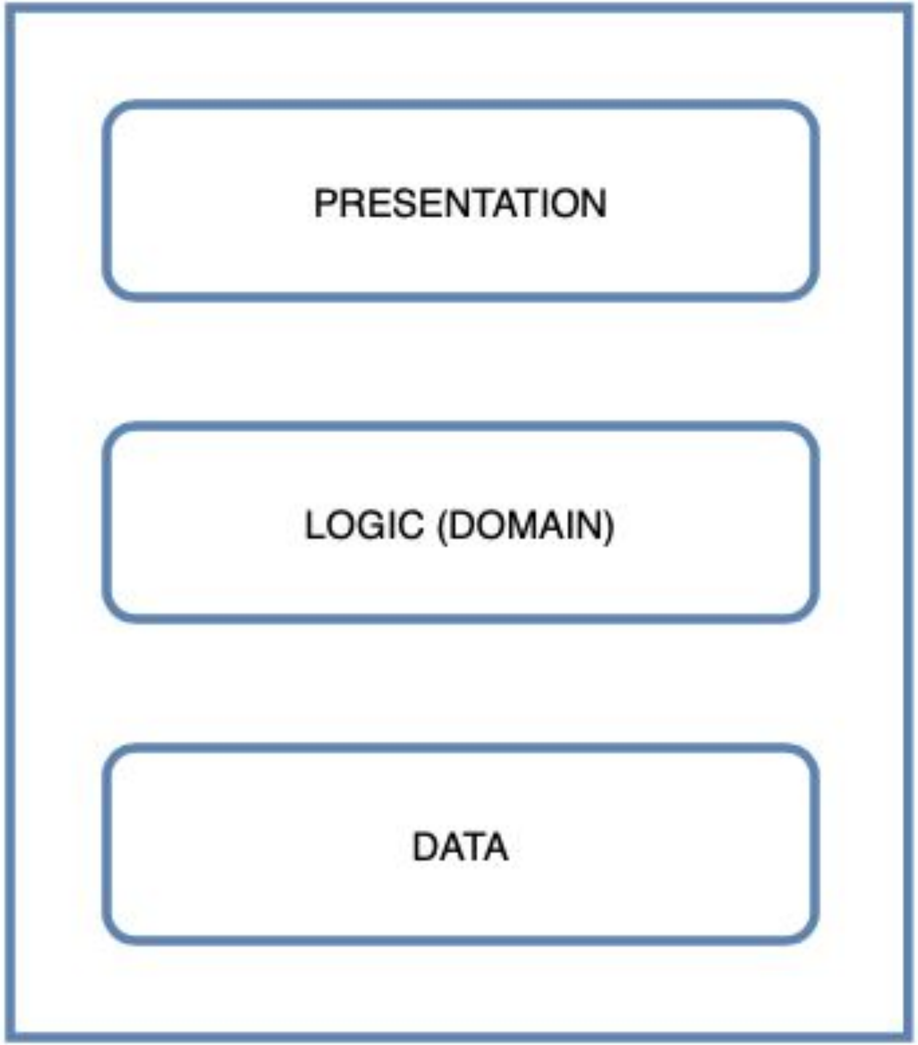
N-LAYERED vs N-TIERED



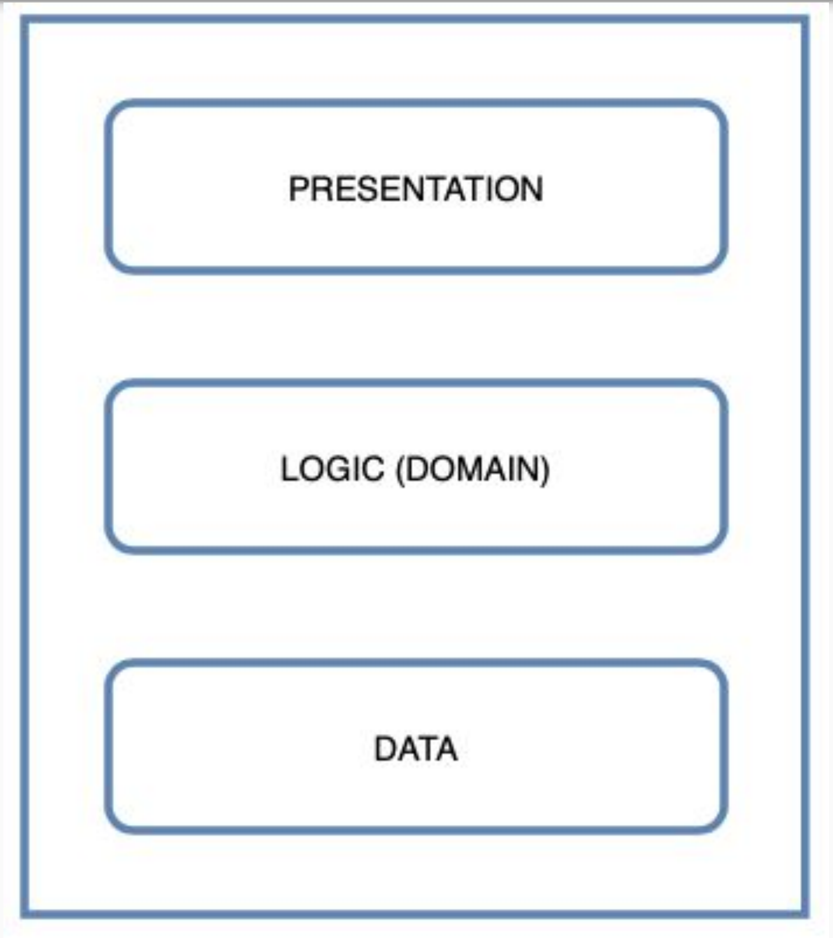


N-Layered Architecture

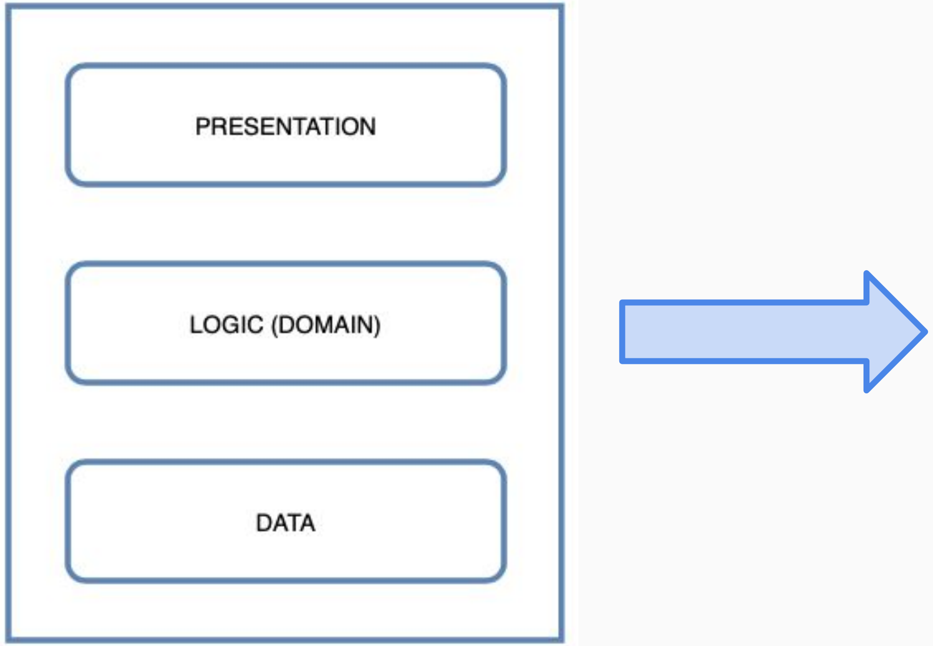
Katmanlı Mimari

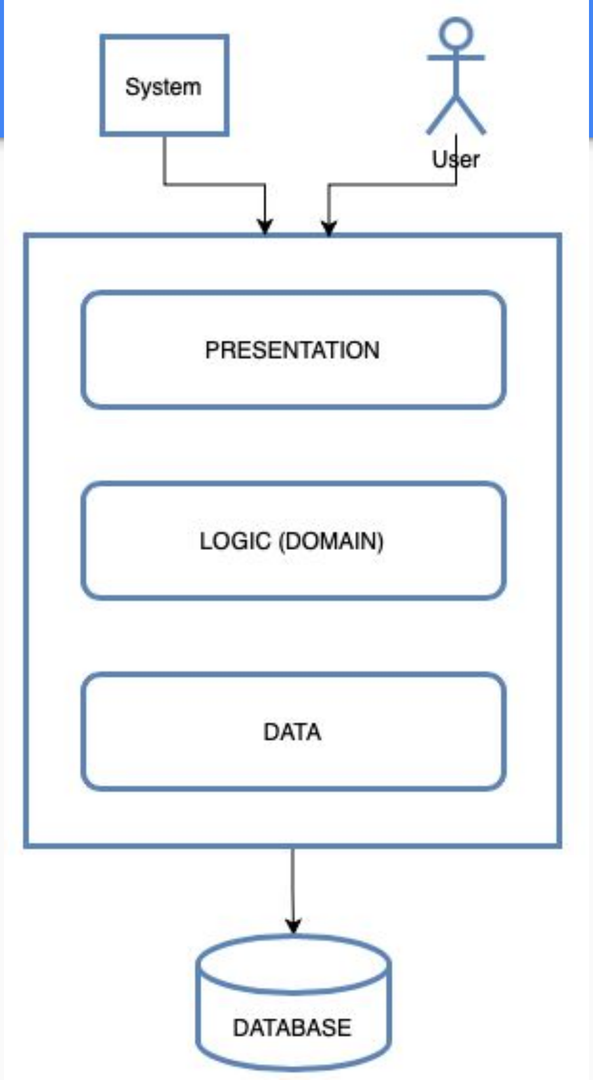


Katmanlı Mimari

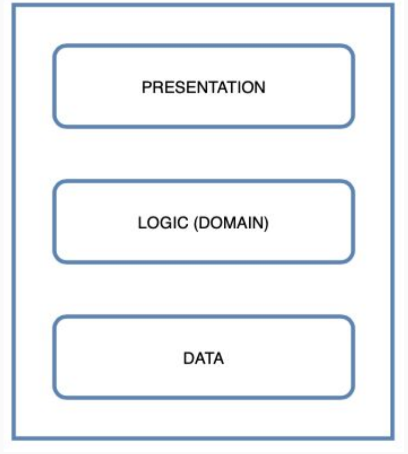


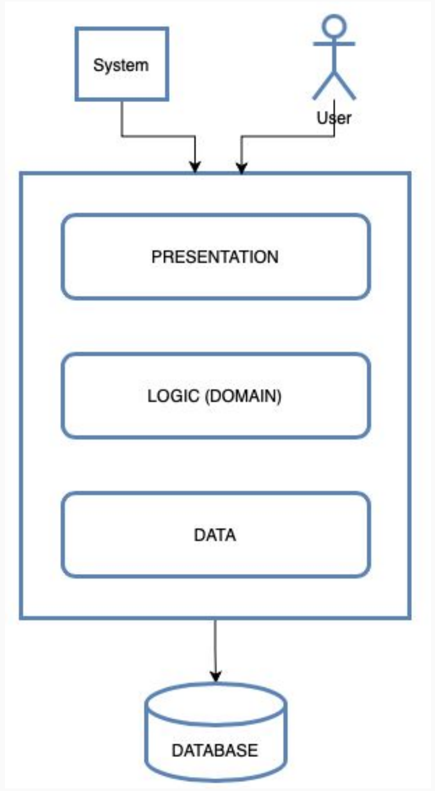
Katmanlı Mimari

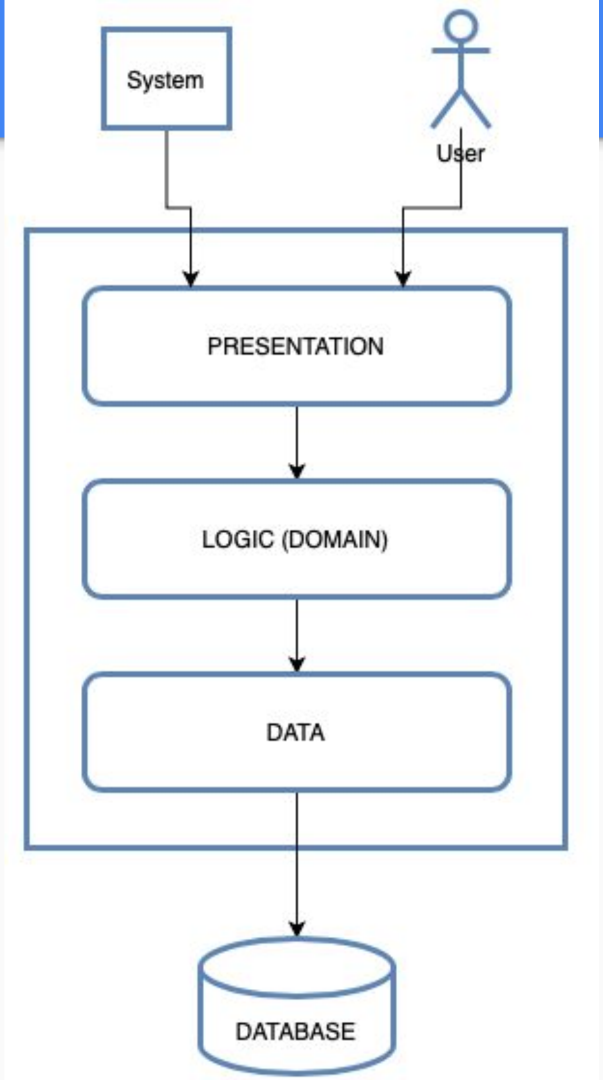




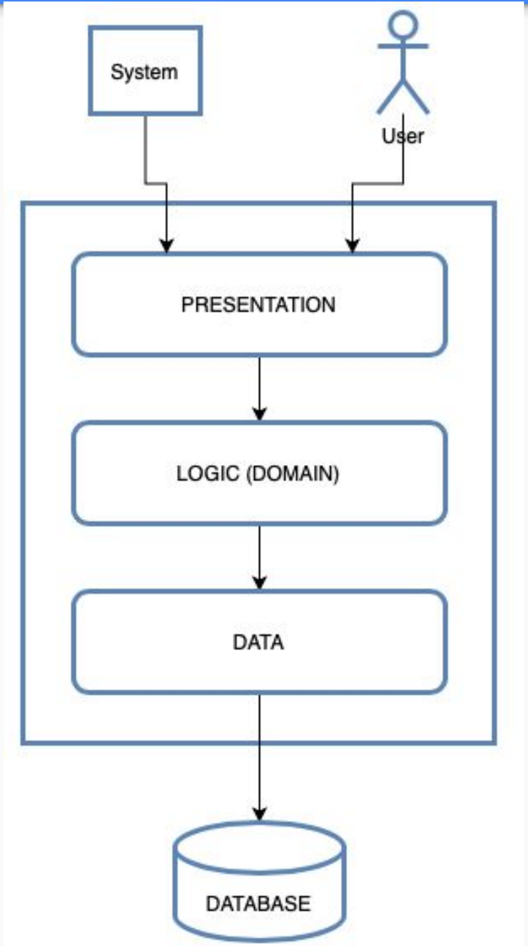
Katmanlı Mimari



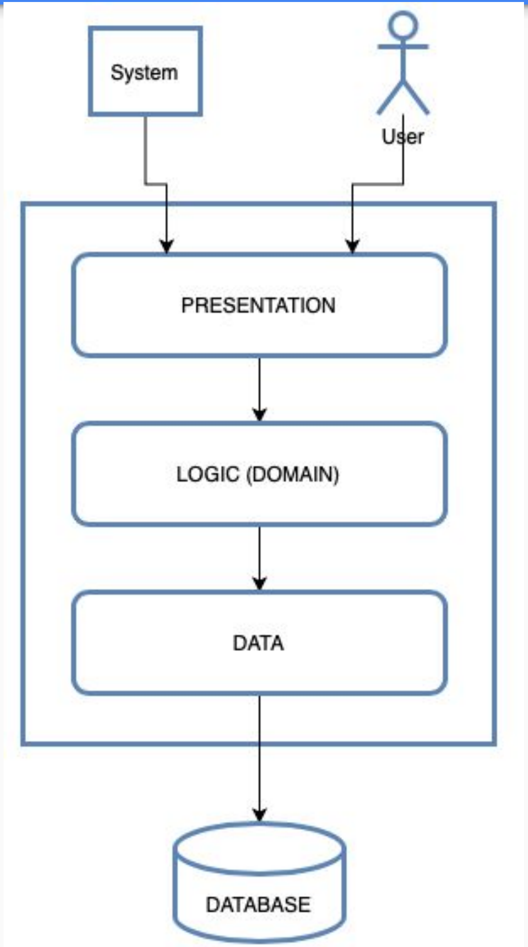


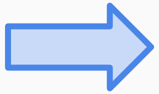


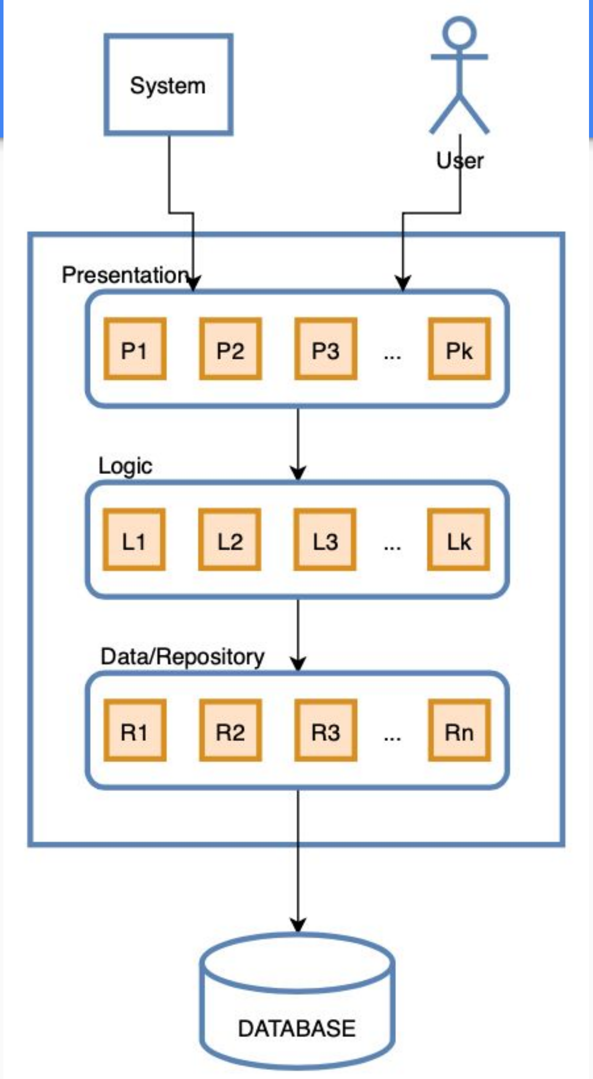
Katmanlı Mimari



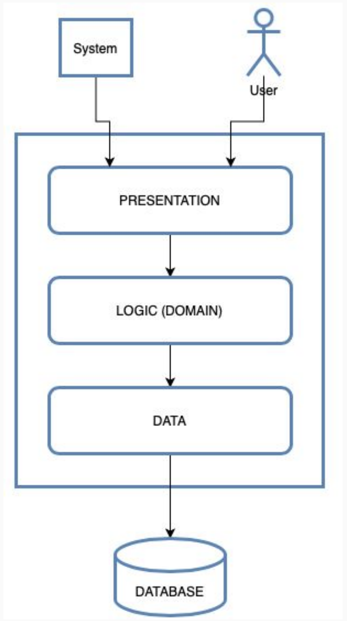
Katmanlı Mimari

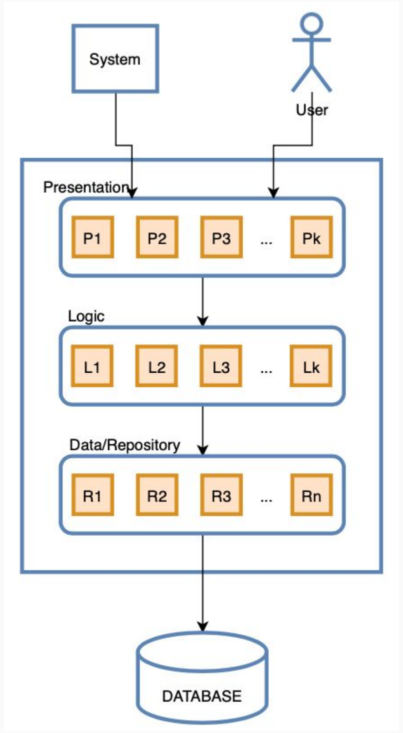


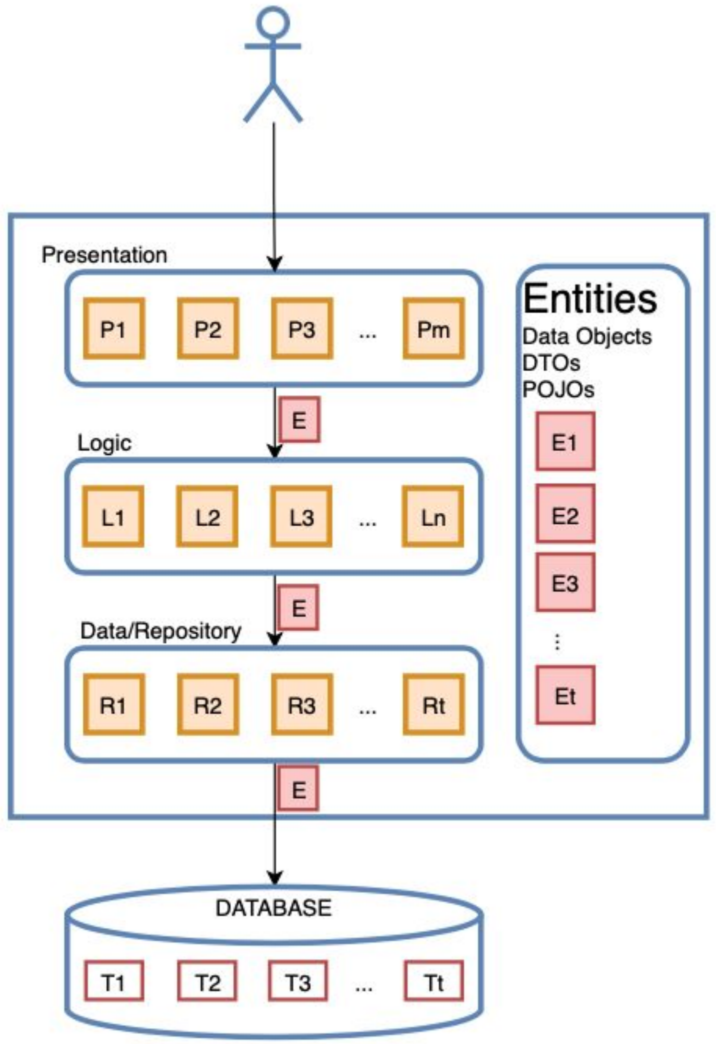




Katmanlı Mimari





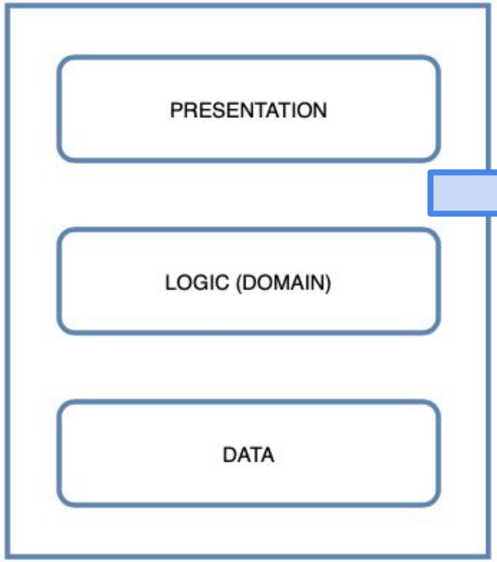


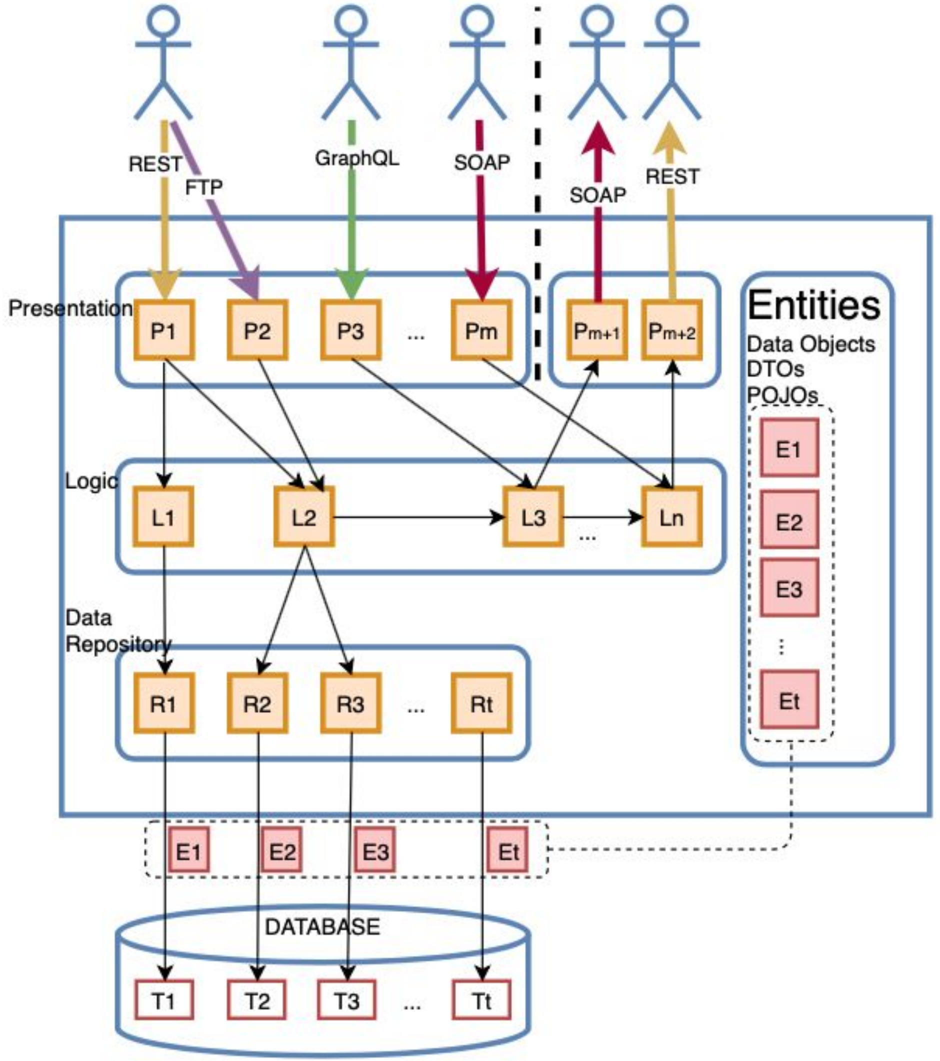
Katmanlı Mimari

Application Layer

Domain Layer (or Model Layer)

Infrastructure Layer





Hexagonal Architecture
Ports & Adapters Architecture

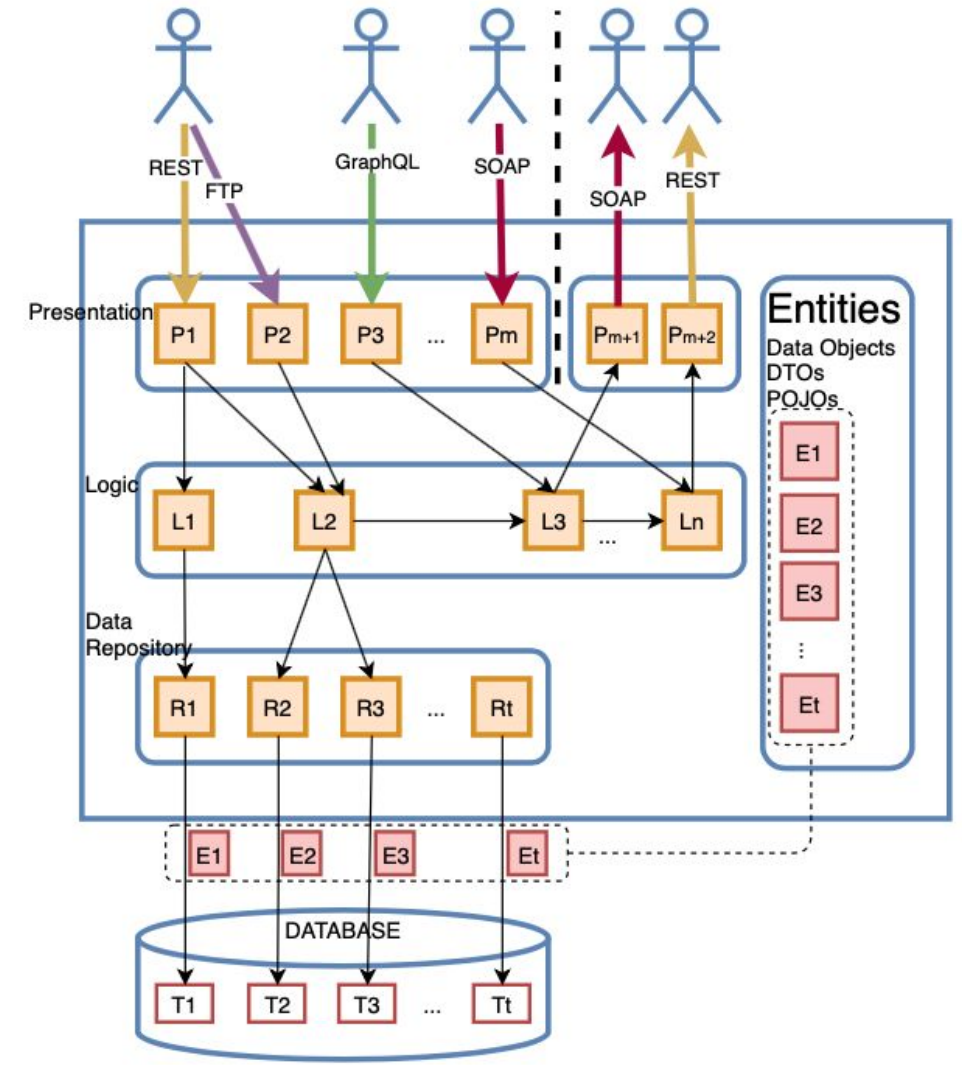
Altıgen Mimari

Dr. Alistair Cockburn

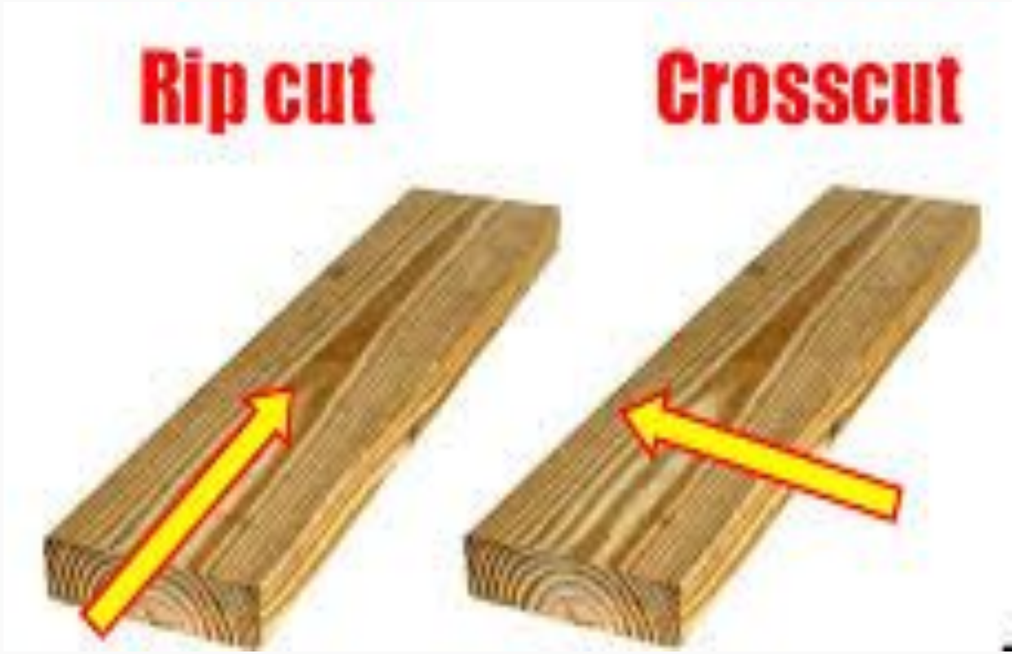
alistair.cockburn.us

N-Tiered Architecture

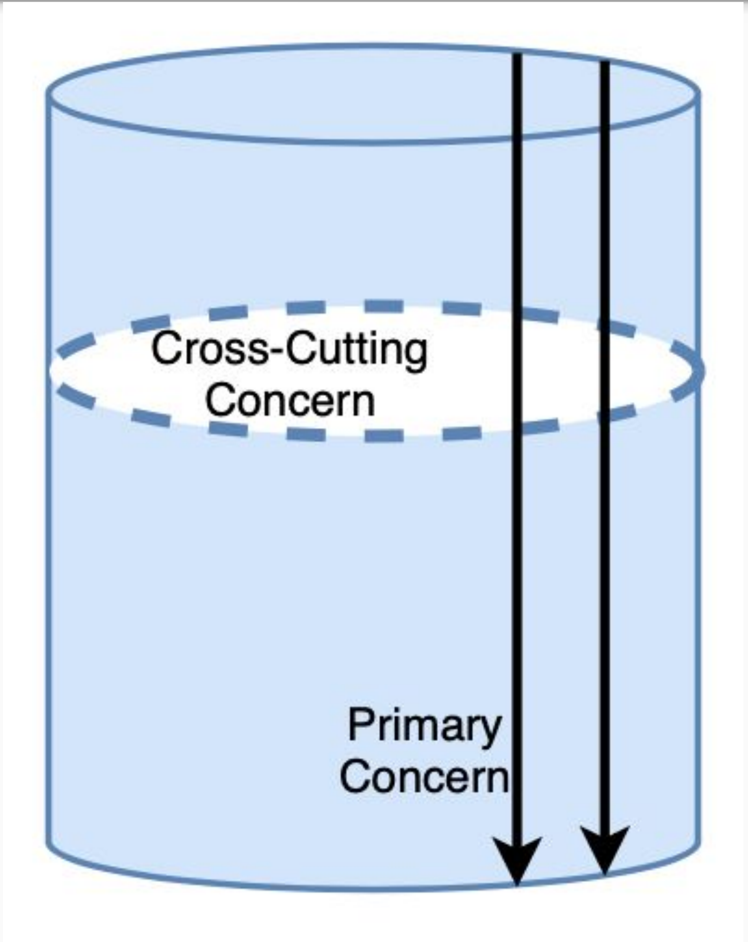
Katmanlı Mimari



CROSS CUT



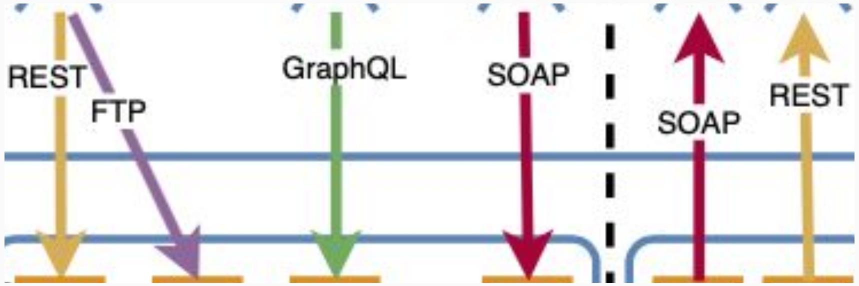
Cross-Cutting Concerns (Enine Kesen / Kesitsel)

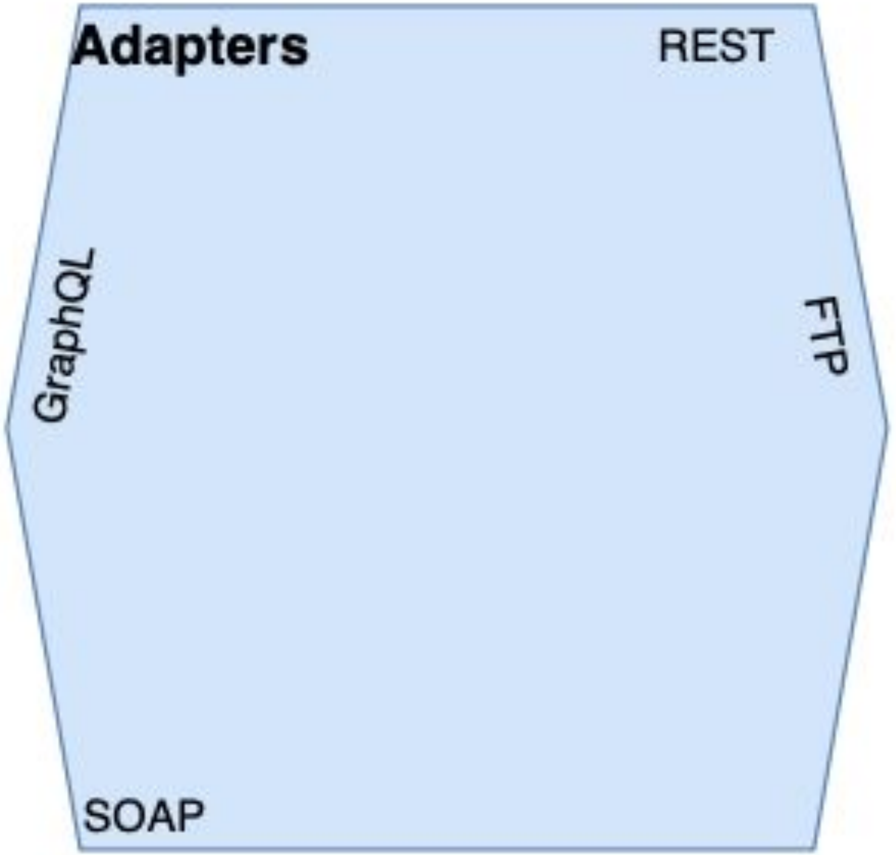


- Synchronization
- Transaction
- Audit
- Authentication
- Authorization

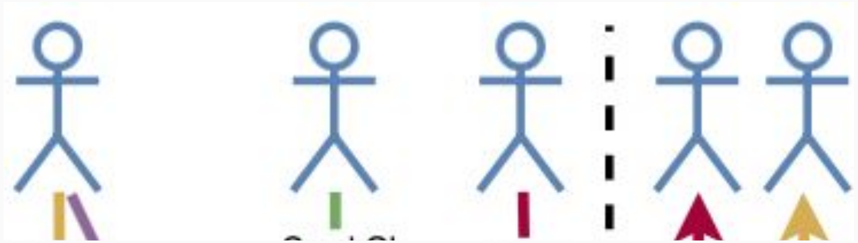
- **Transformation**

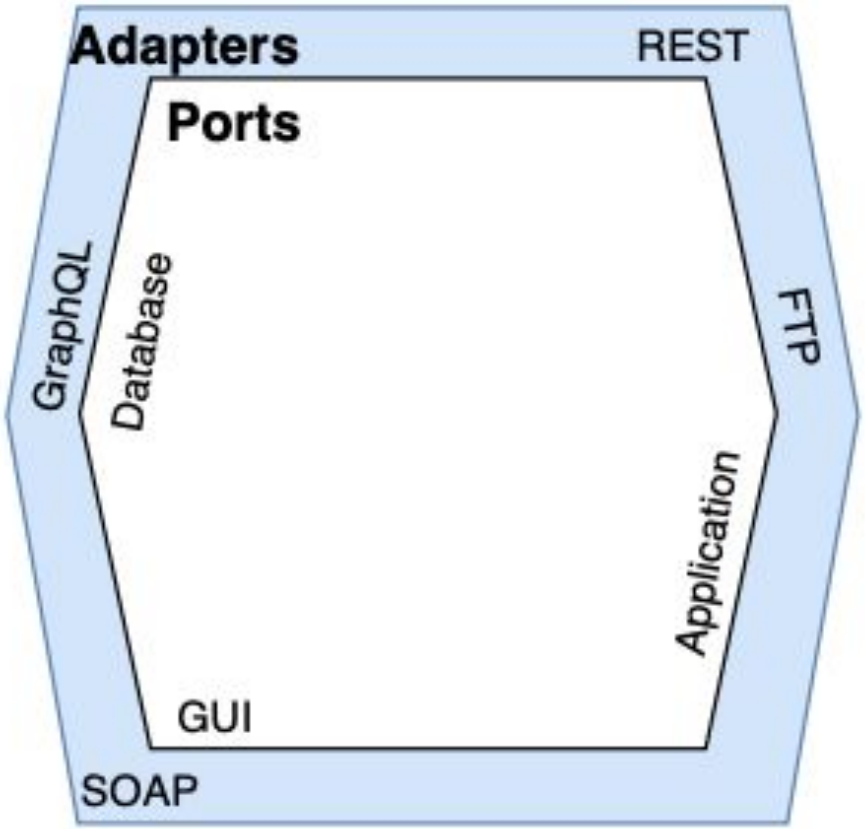
Problem: Data Transfer Formats & Protocols



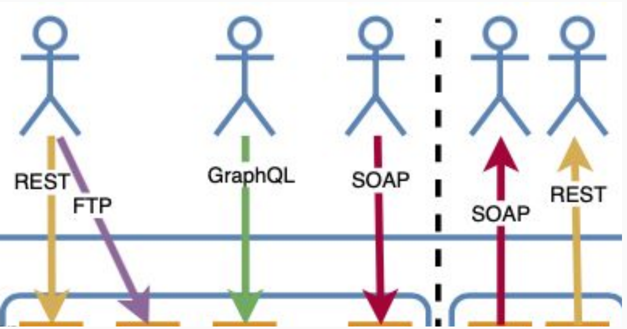


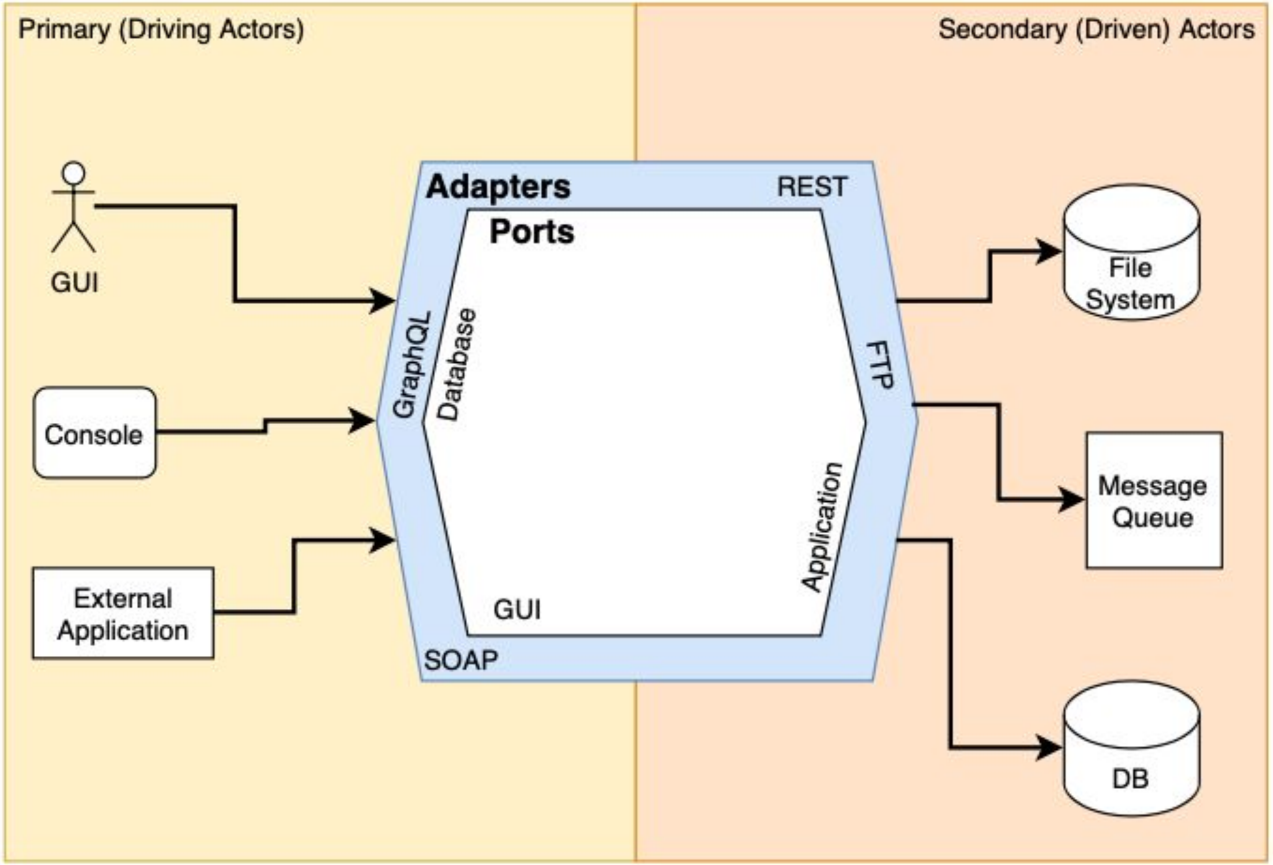
Problem: Numerous Actors



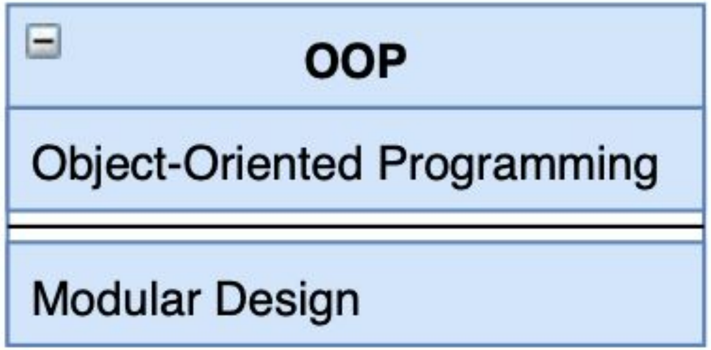


Problem: Data Direction





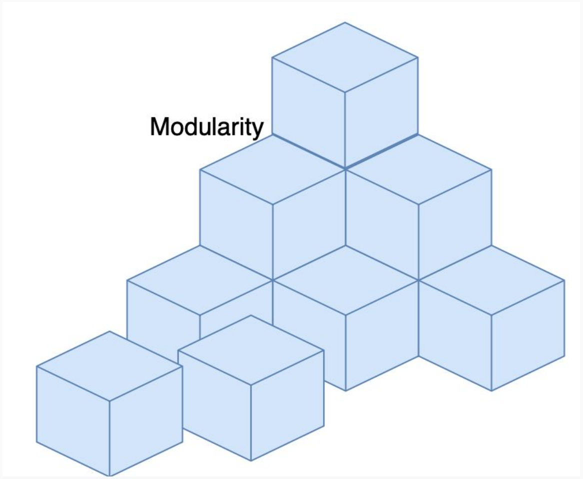
SOLID - Dependency Inversion Principle

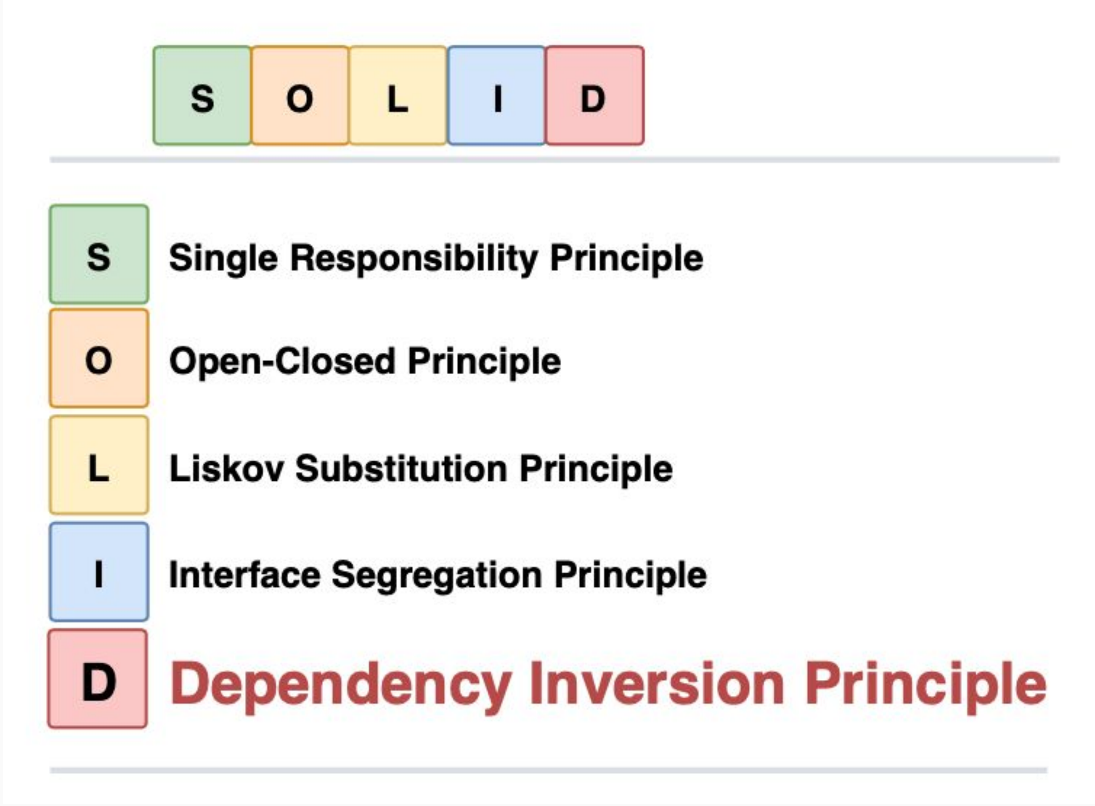


OOP

Object-Oriented Programming

Modular Design



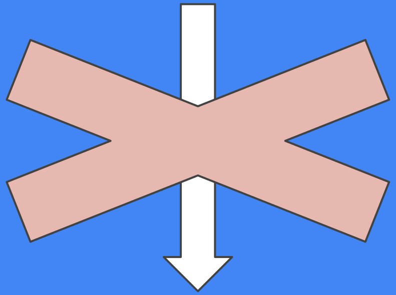


S O L I D

S	Single Responsibility Principle
O	Open-Closed Principle
L	Liskov Substitution Principle
I	Interface Segregation Principle
D	Dependency Inversion Principle

N-Tiered Architecture

Dependency Inversion Principle



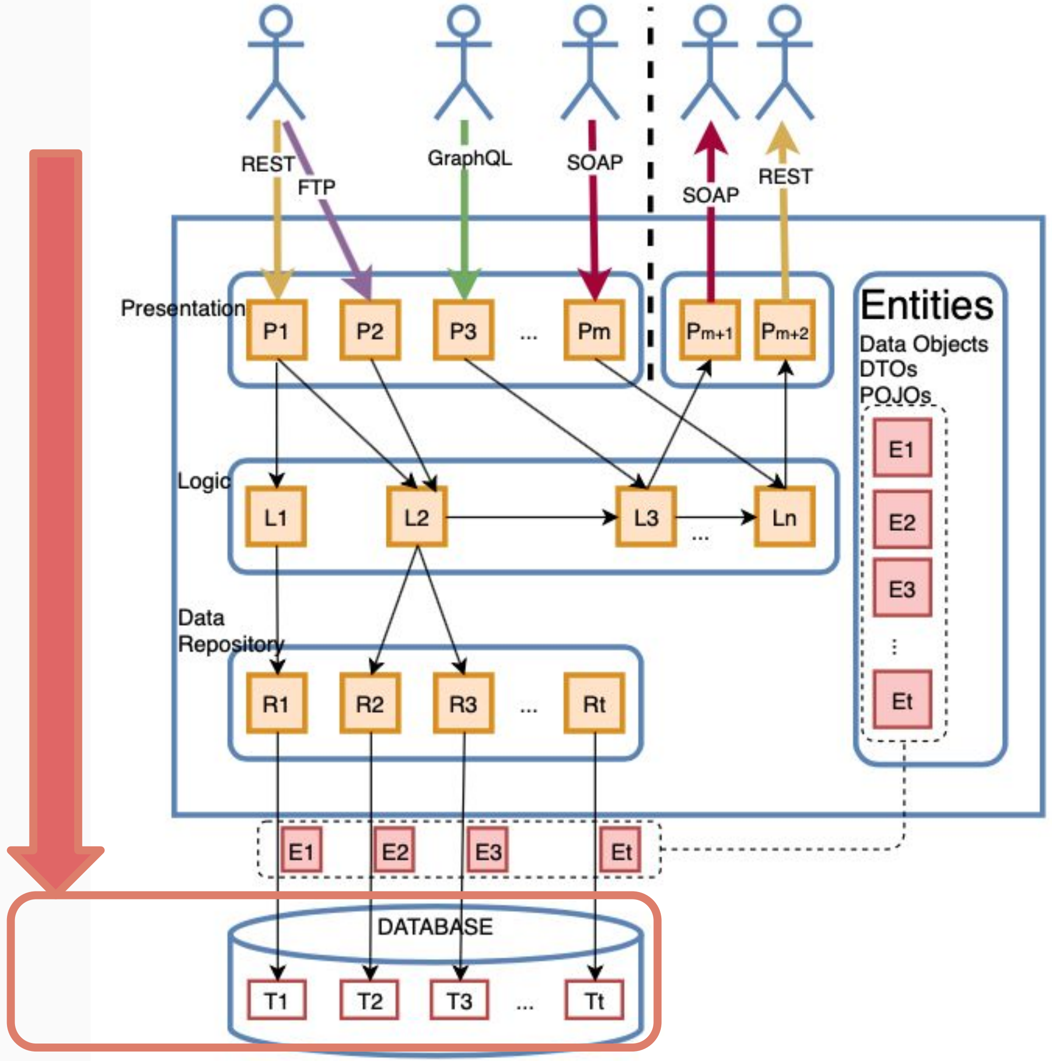
DB = Highest Abstraction Level

Database Centric Architecture

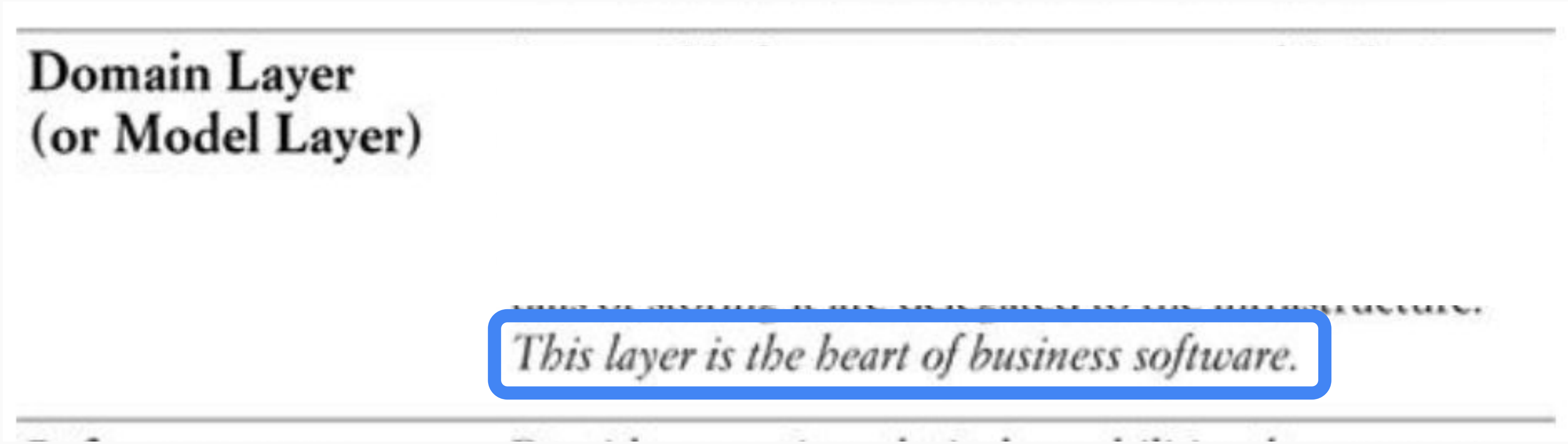
(Data-Driven Design)

vs

Domain-Driven Design



Domain-Driven Design

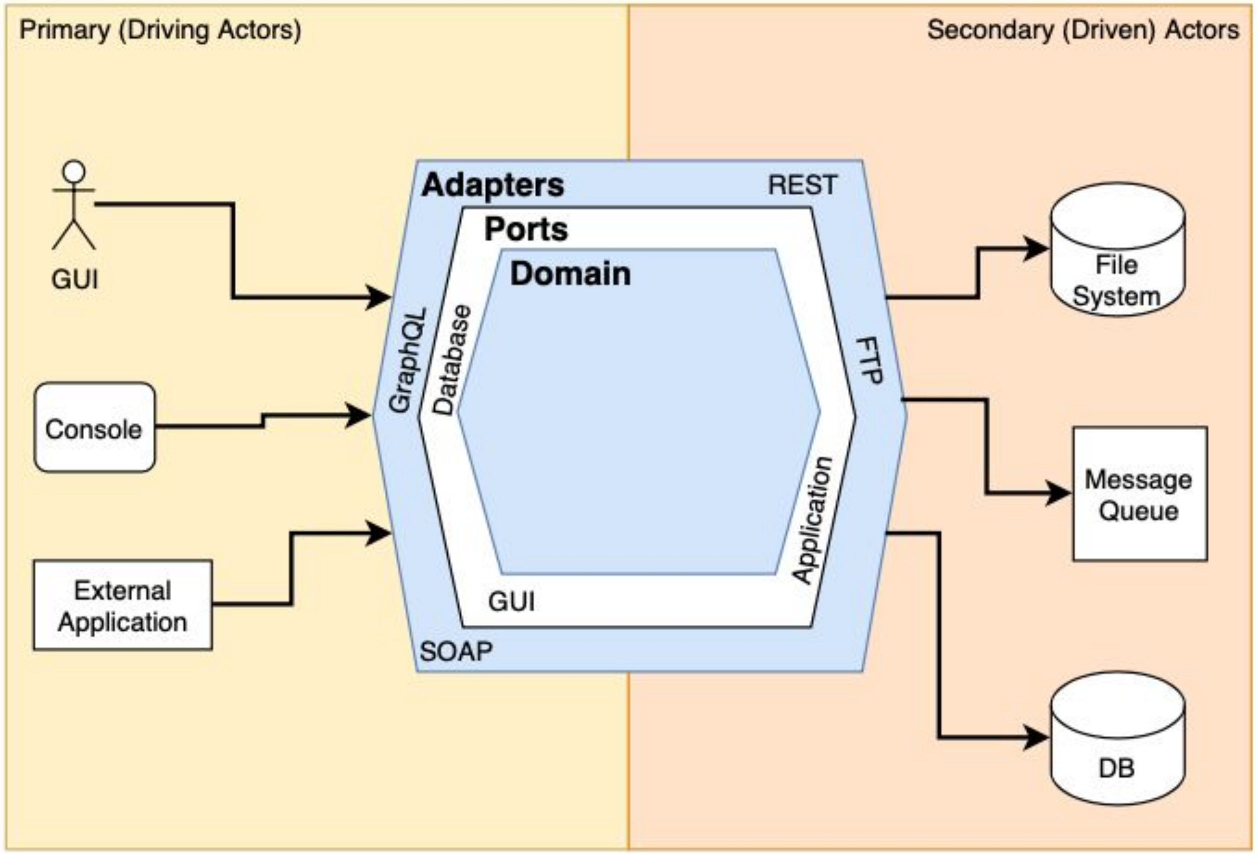


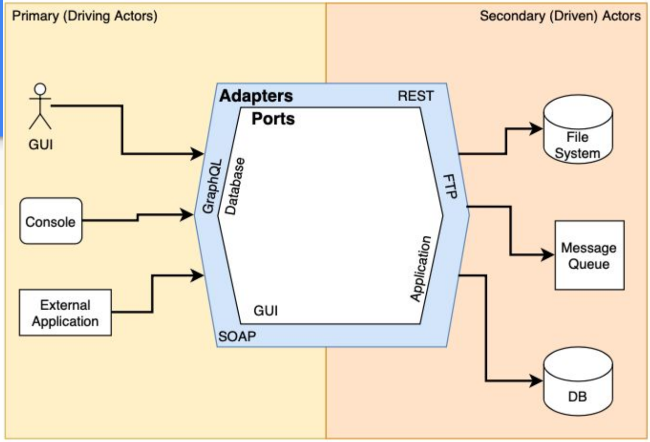
Domain Layer (or Model Layer)

This layer is the heart of business software.

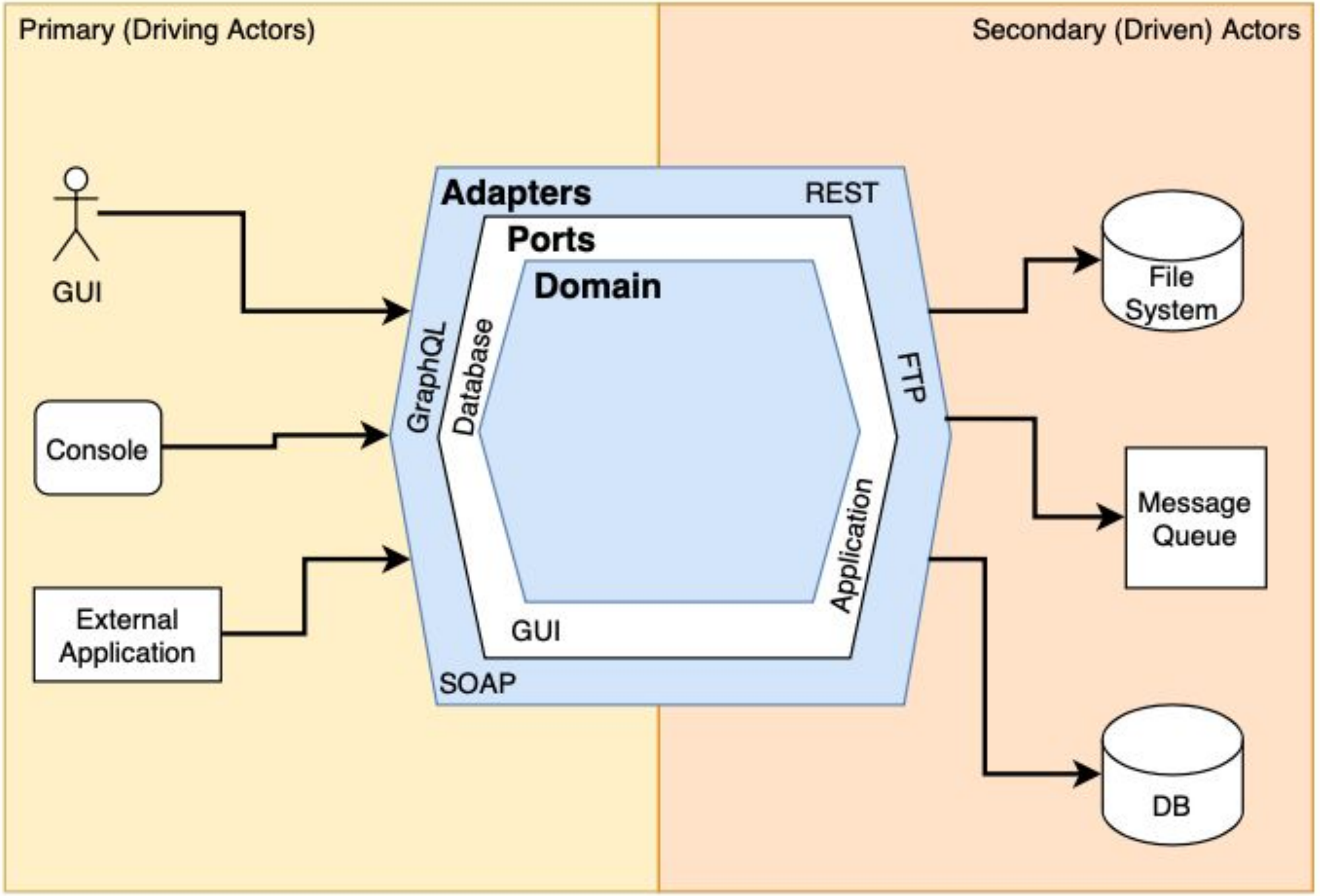
Source: Eric Evans - Domain Driven Design

Hexagonal Domain Model



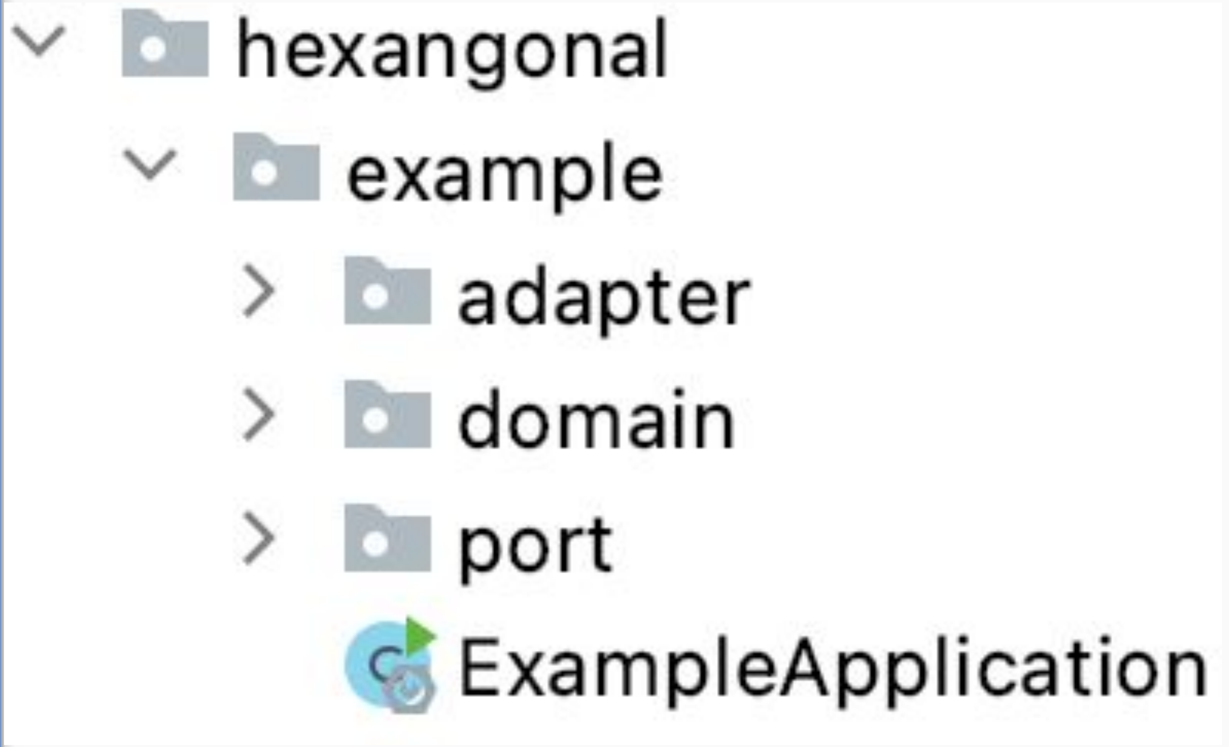


Hexagonal


Primary (Driving Actors)
Secondary (Driven) Actors
GUI
Console
External Application
Adapters
REST
Ports
Domain
GraphQL
Database
FTP
Application
GUI
SOAP
File System
Message Queue
DB

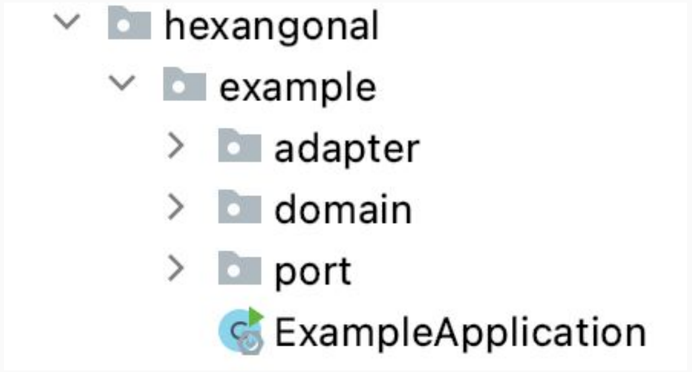
Hexagonal
Spring Boot Example

(Oversimplified)

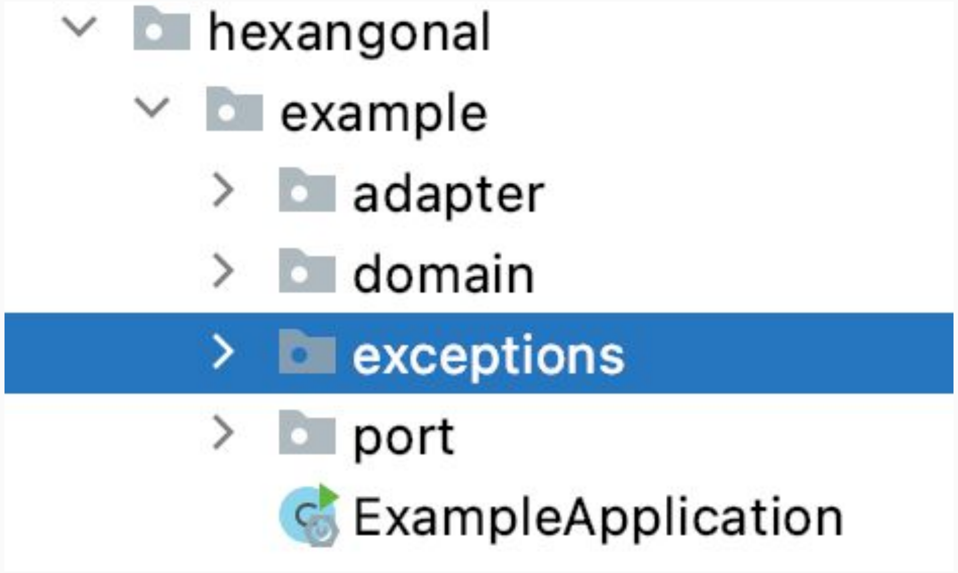


```
hexangonal
  example
    adapter
    domain
    port
    ExampleApplication
```

Hexagonal
Spring Boot Example



```
hexangonal
  example
    adapter
    domain
    port
    ExampleApplication
```

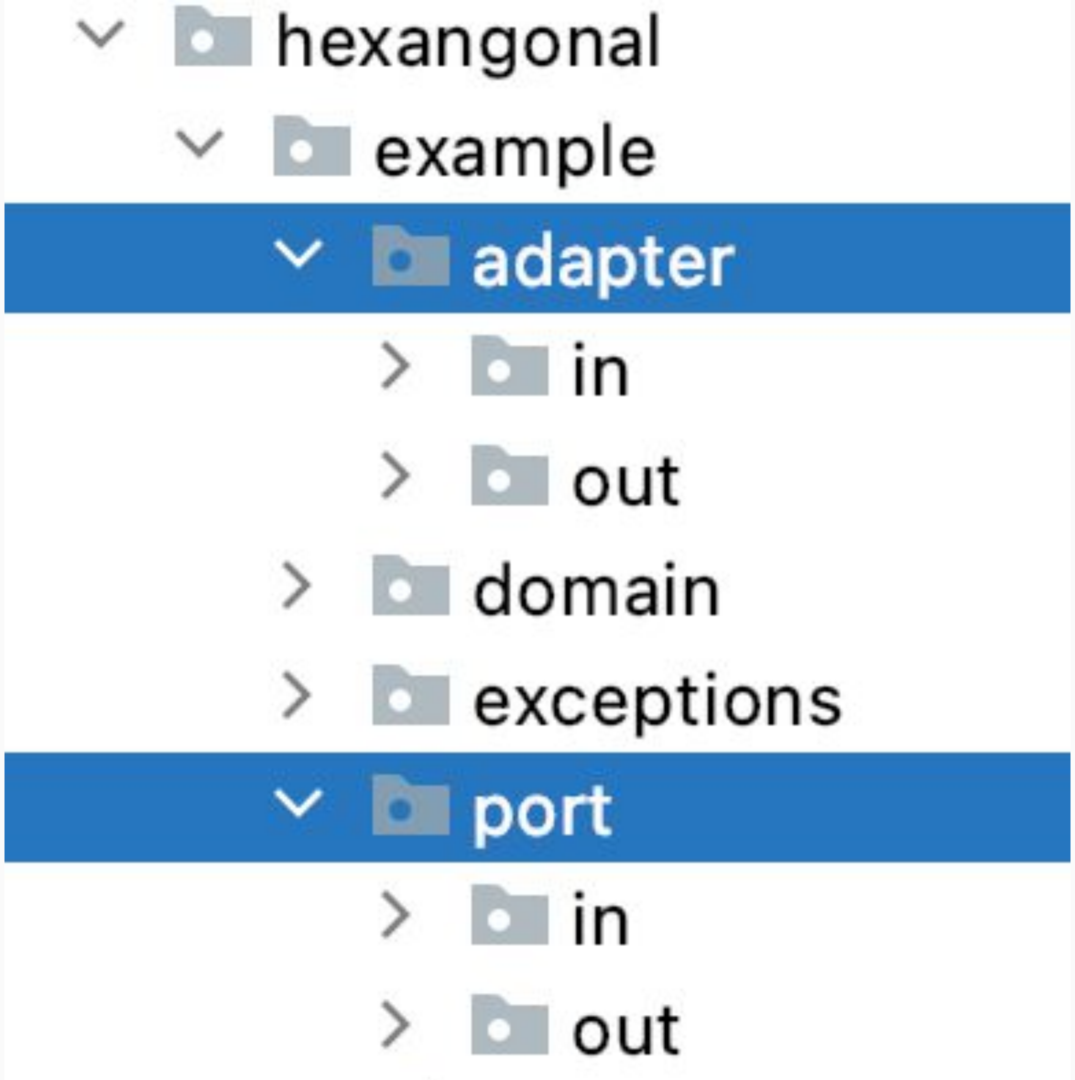


```
hexangonal
  example
    adapter
    domain
    exceptions
    port
    ExampleApplication
```

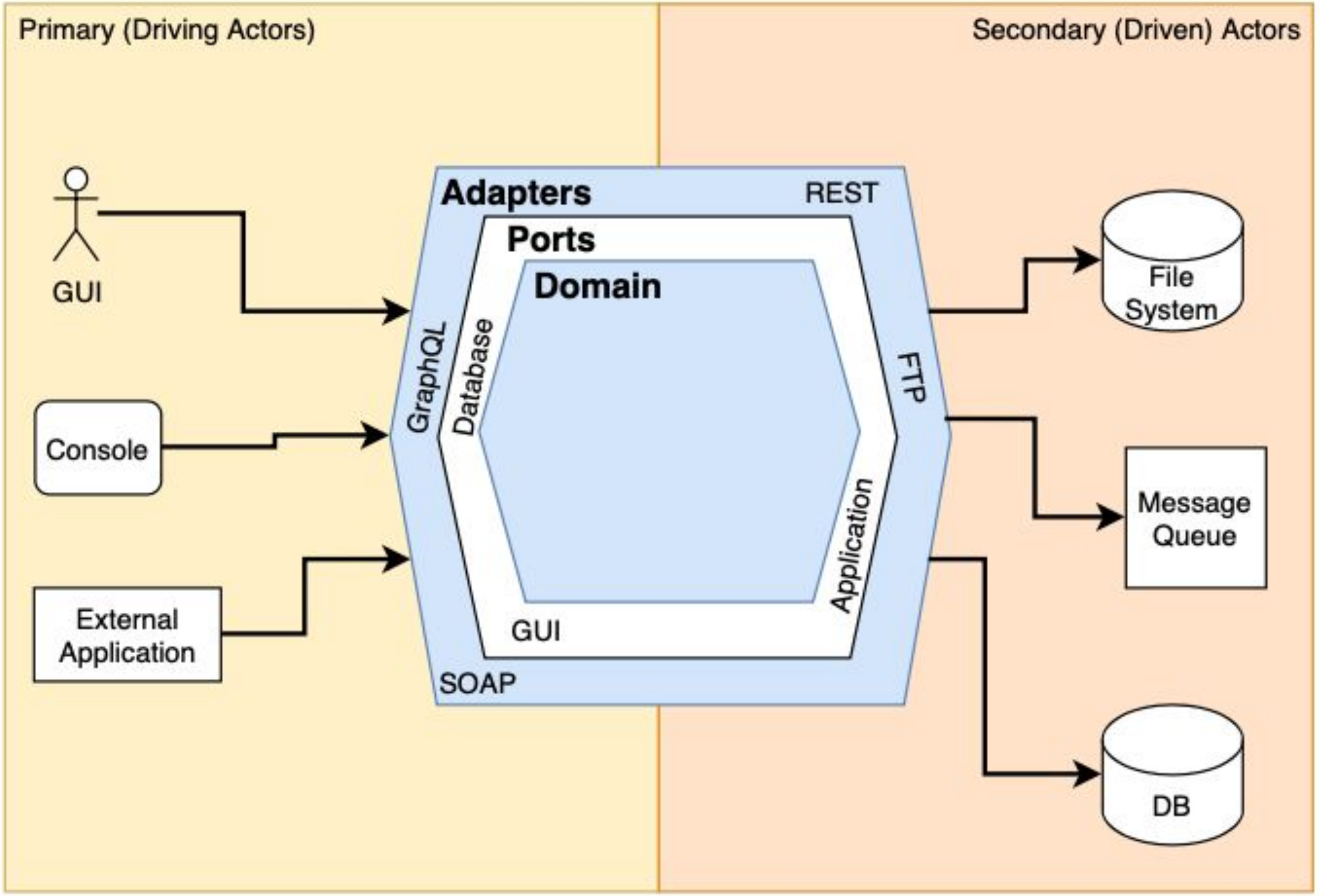
(Oversimplified)

Hexagonal
Spring Boot Example

(Oversimplified)

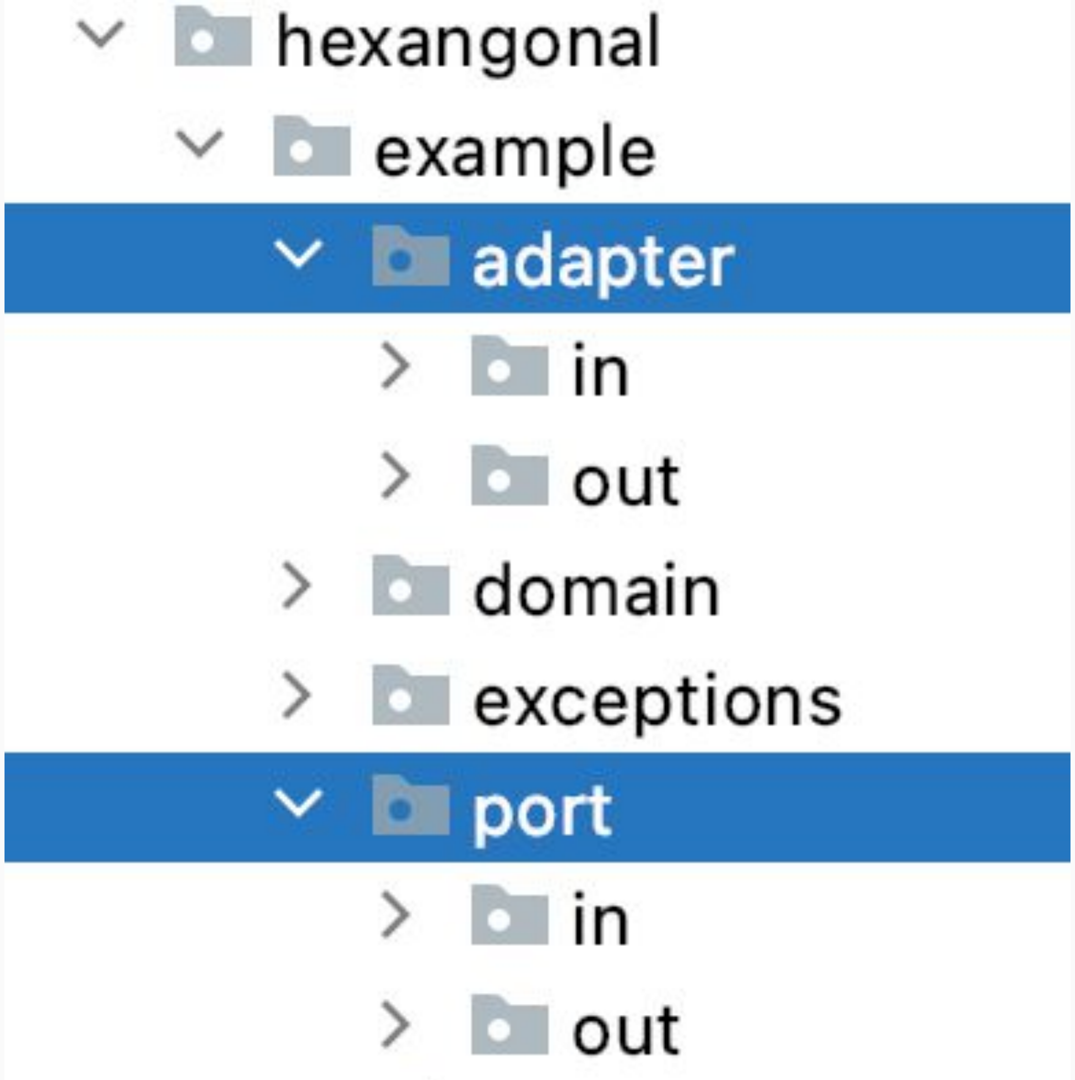


Hexagonal


Primary (Driving Actors)
Secondary (Driven) Actors
GUI
Console
External Application
Adapters
REST
Ports
Domain
GraphQL
Database
FTP
Application
GUI
SOAP
File System
Message Queue
DB

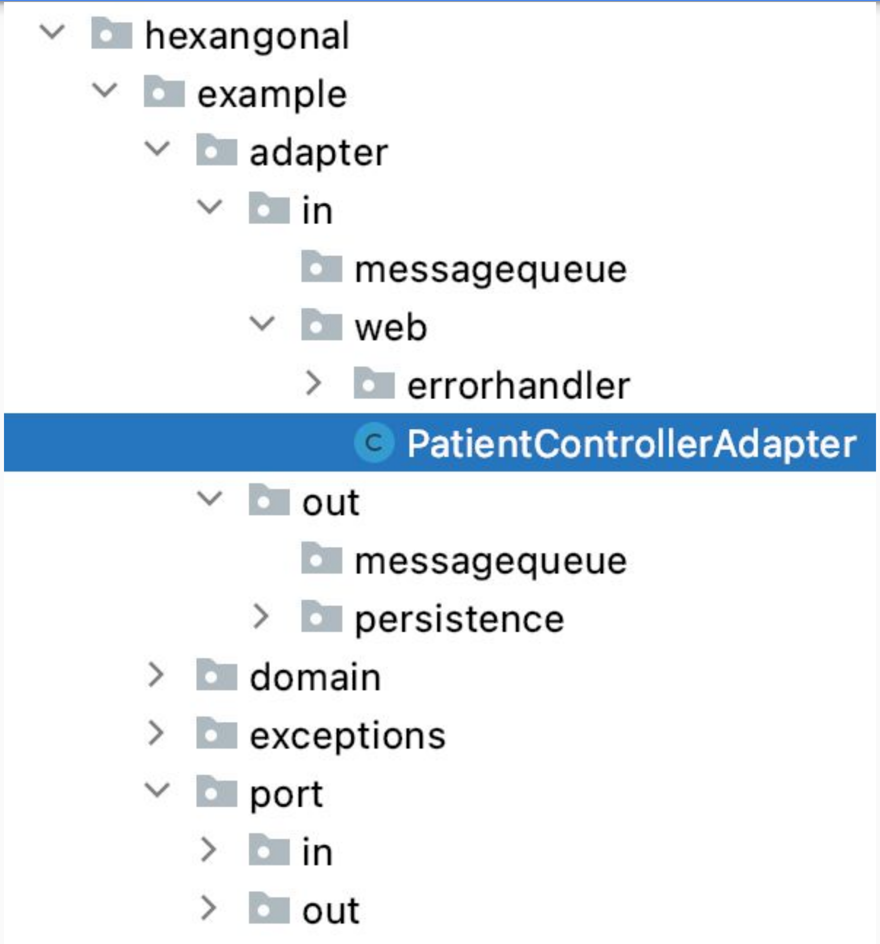
Hexagonal Spring Boot Example

(Oversimplified)



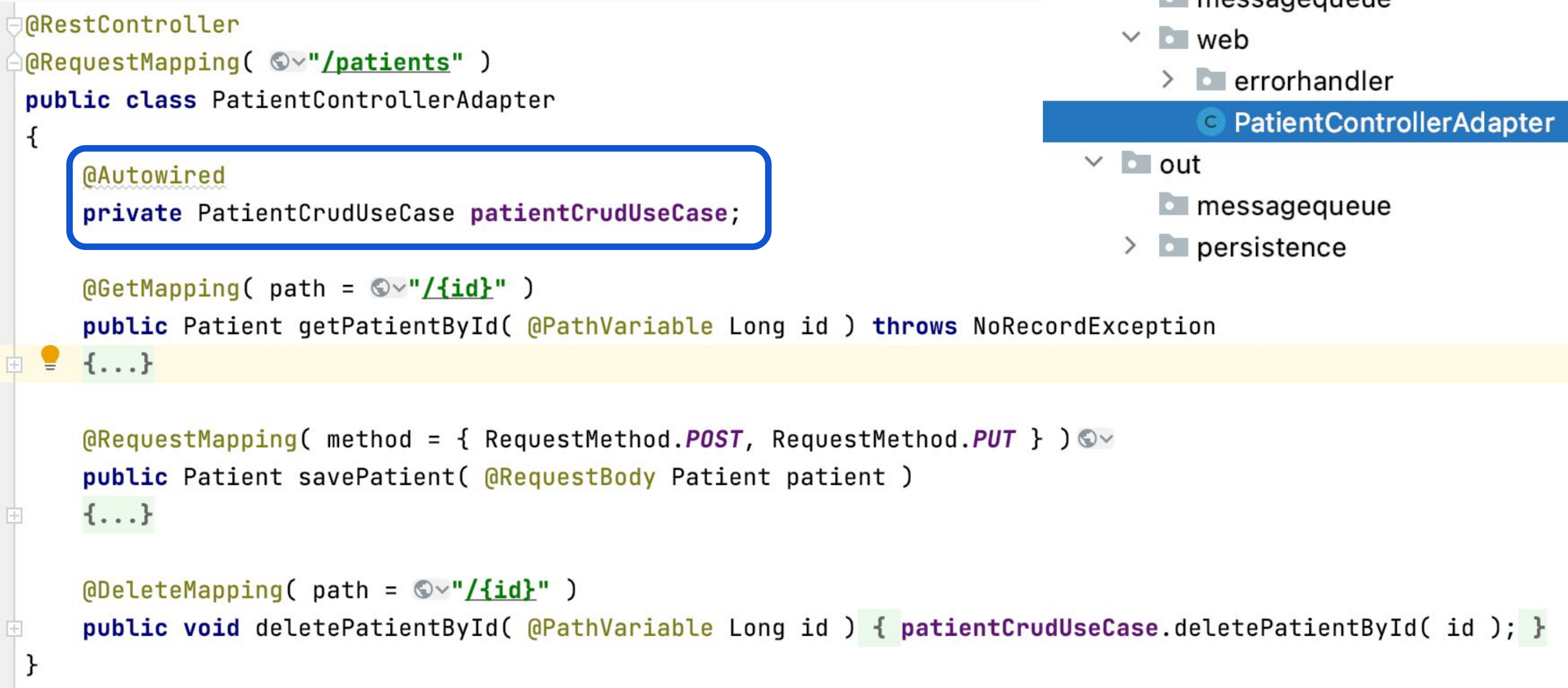
```
hexangonal
  example
    adapter
      in
      out
    domain
    exceptions
    port
      in
      out
```

In Adapter Example

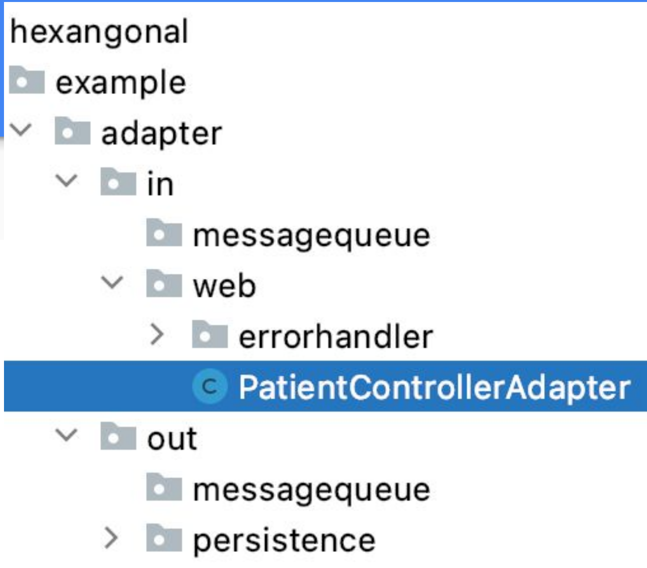


```
hexangonal
  example
    adapter
      in
        messagequeue
        web
          errorhandler
          PatientControllerAdapter
      out
        messagequeue
        persistence
    domain
    exceptions
    port
      in
      out
```

In Adapter Example

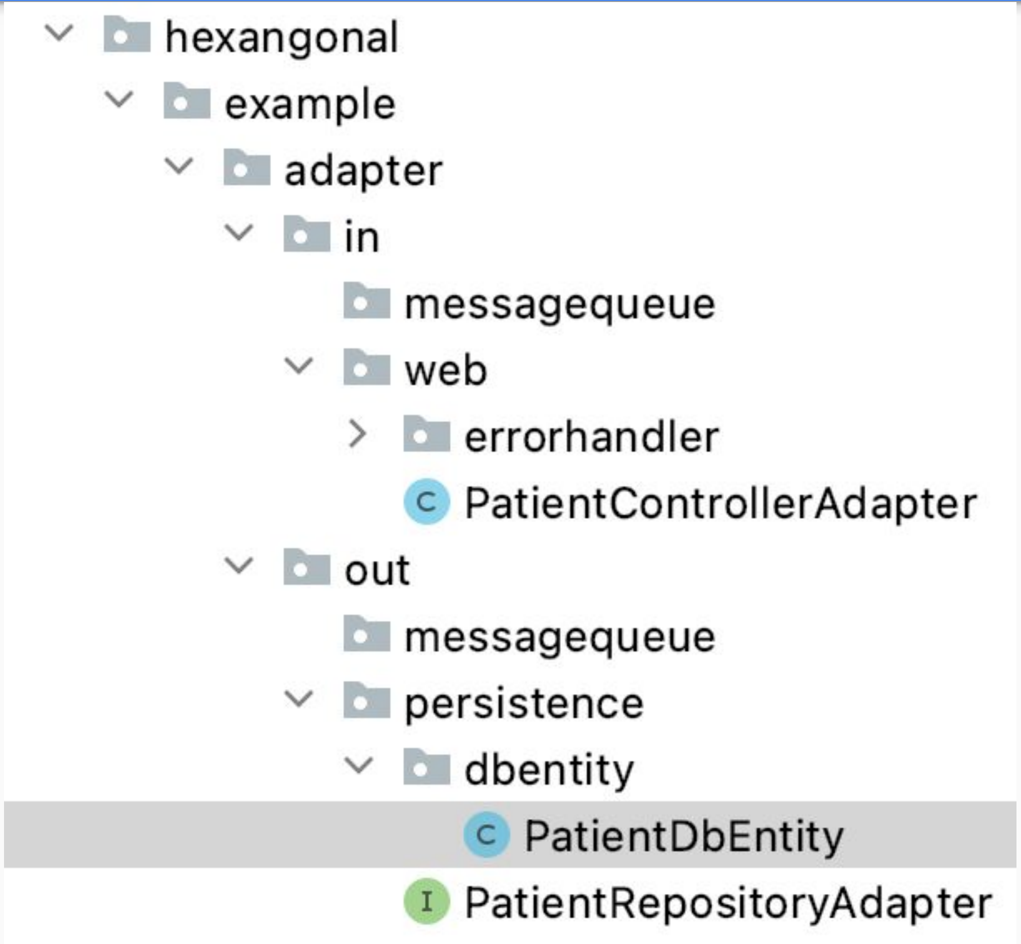


```java
@RestController
@RequestMapping( "/patients" )
public class PatientControllerAdapter
{
    @Autowired
    private PatientCrudUseCase patientCrudUseCase;

    @GetMapping( path = "/{id}" )
    public Patient getPatientById( @PathVariable Long id ) throws NoRecordException
    {...}

    @RequestMapping( method = { RequestMethod.POST, RequestMethod.PUT } )
    public Patient savePatient( @RequestBody Patient patient )
    {...}

    @DeleteMapping( path = "/{id}" )
    public void deletePatientById( @PathVariable Long id ) { patientCrudUseCase.deletePatientById( id ); }
}
```



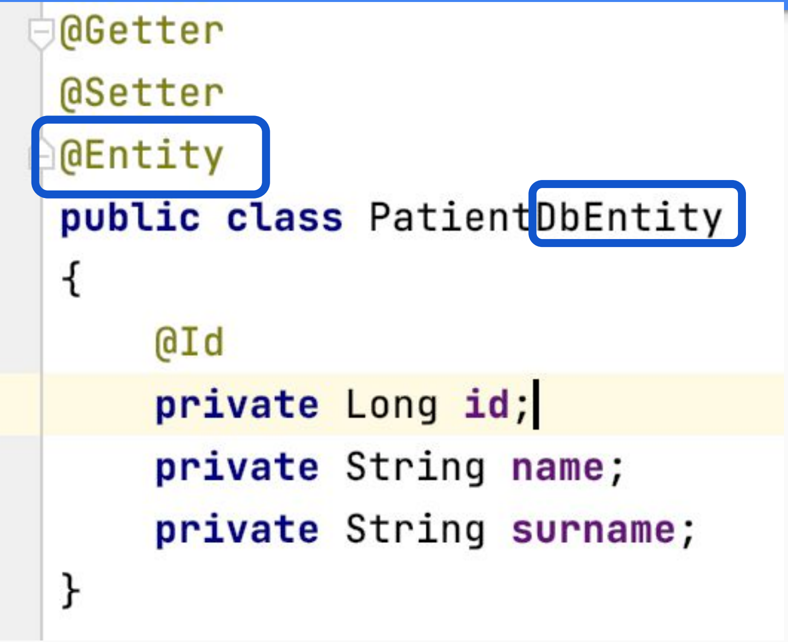
```
hexangonal
example
  adapter
    in
      messagequeue
      web
        errorhandler
        PatientControllerAdapter
    out
      messagequeue
      persistence
```

Out Adapter Example

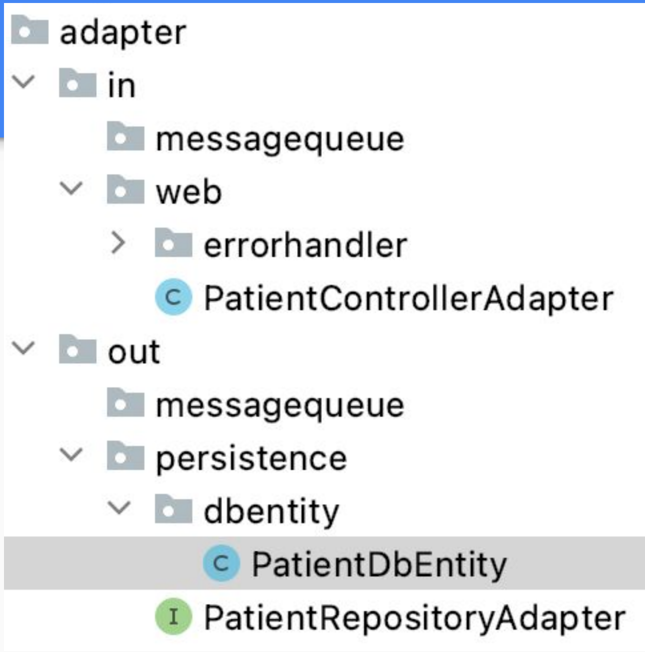


```
⌄ hexangonal
  ⌄ example
    ⌄ adapter
      ⌄ in
          messagequeue
        ⌄ web
          › errorhandler
            PatientControllerAdapter
      ⌄ out
          messagequeue
        ⌄ persistence
          ⌄ dbentity
              PatientDbEntity
            PatientRepositoryAdapter
```

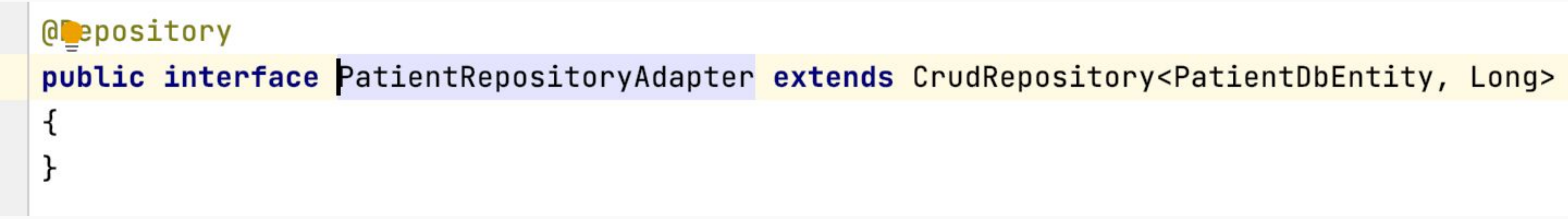
Out Adapter Example



```
@Getter
@Setter
@Entity
public class PatientDbEntity
{
    @Id
    private Long id;
    private String name;
    private String surname;
}
```

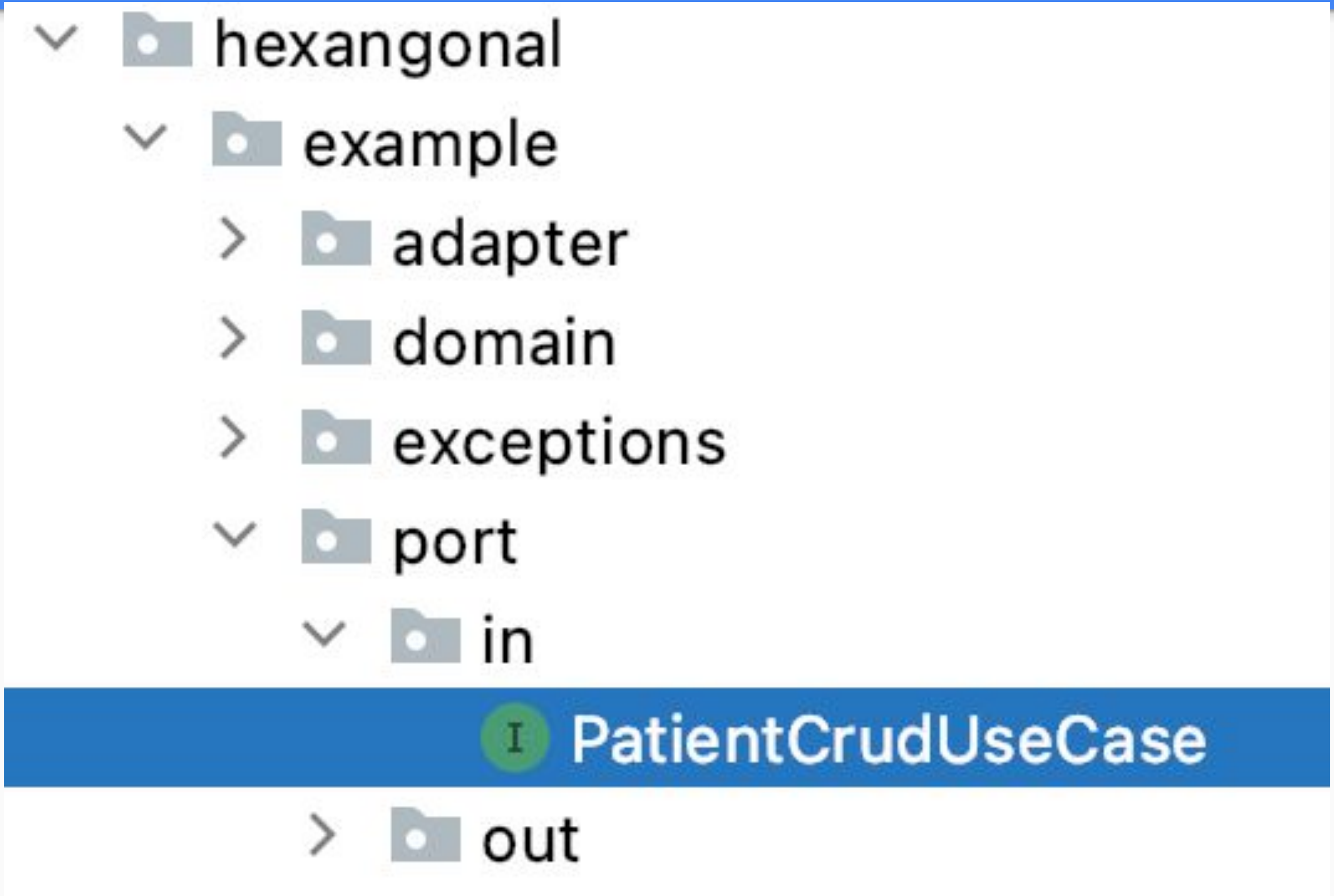


```
adapter
  in
    messagequeue
    web
      errorhandler
      PatientControllerAdapter
  out
    messagequeue
    persistence
      dbentity
        PatientDbEntity
      PatientRepositoryAdapter
```



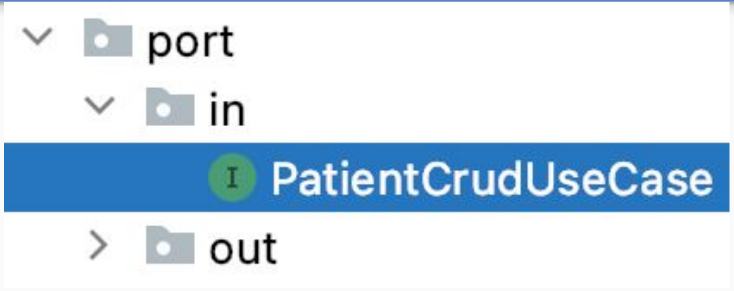
```
@Repository
public interface PatientRepositoryAdapter extends CrudRepository<PatientDbEntity, Long>
{
}
```

In Port Example



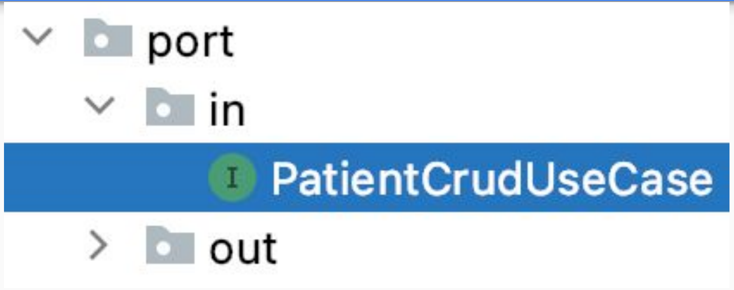
```
hexangonal
  example
    adapter
    domain
    exceptions
    port
      in
        PatientCrudUseCase
      out
```

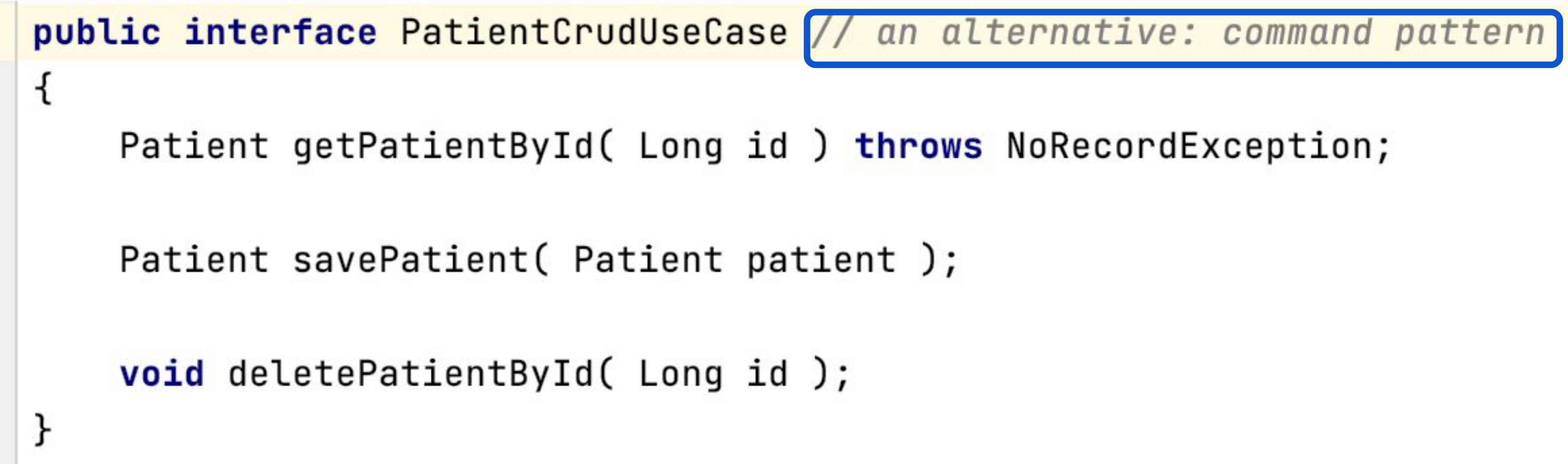
In Port Example



```
public interface PatientCrudUseCase // an alternative: command pattern
{
    Patient getPatientById( Long id ) throws NoRecordException;

    Patient savePatient( Patient patient );

    void deletePatientById( Long id );
}
```

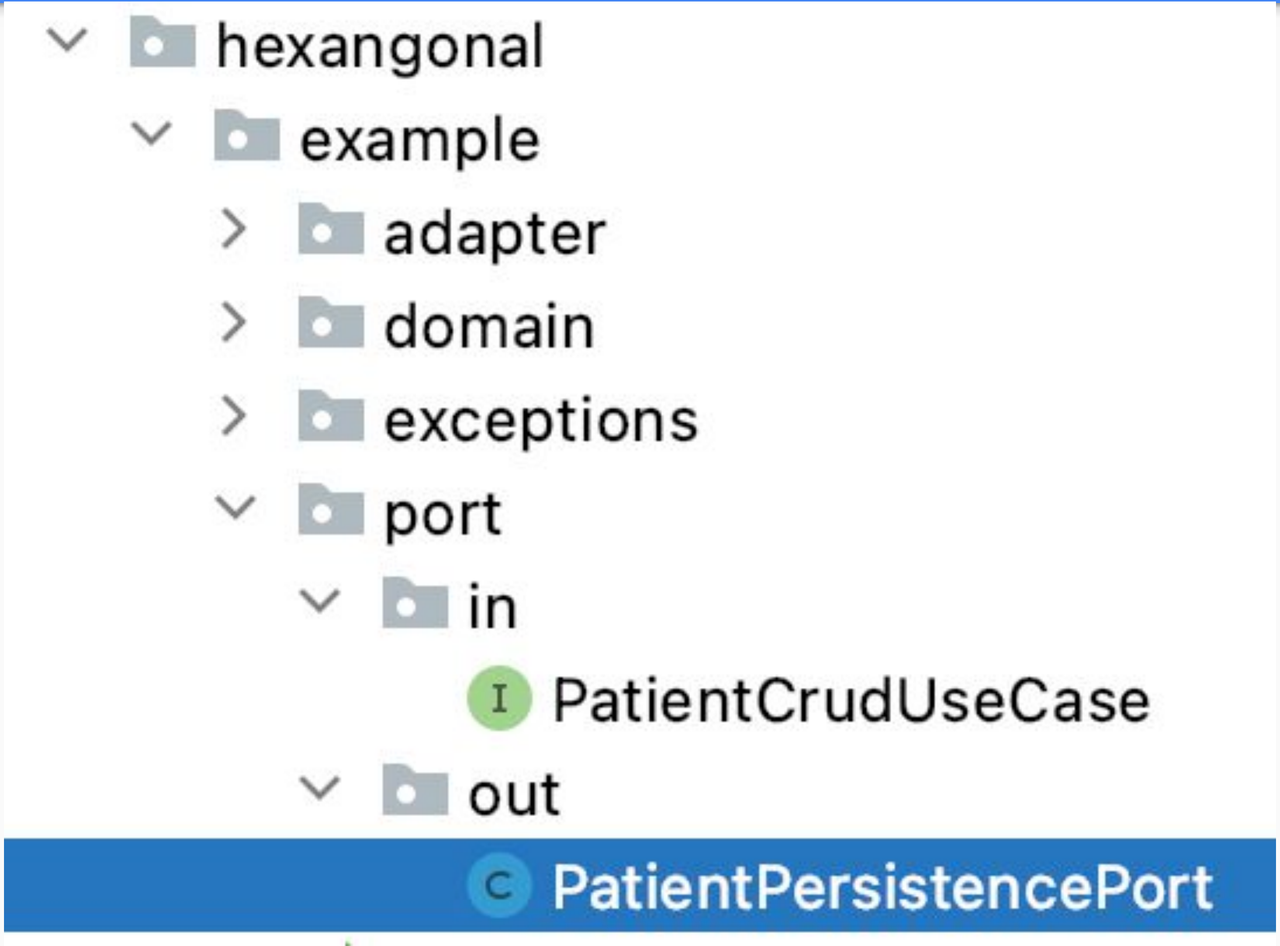
In Port Example





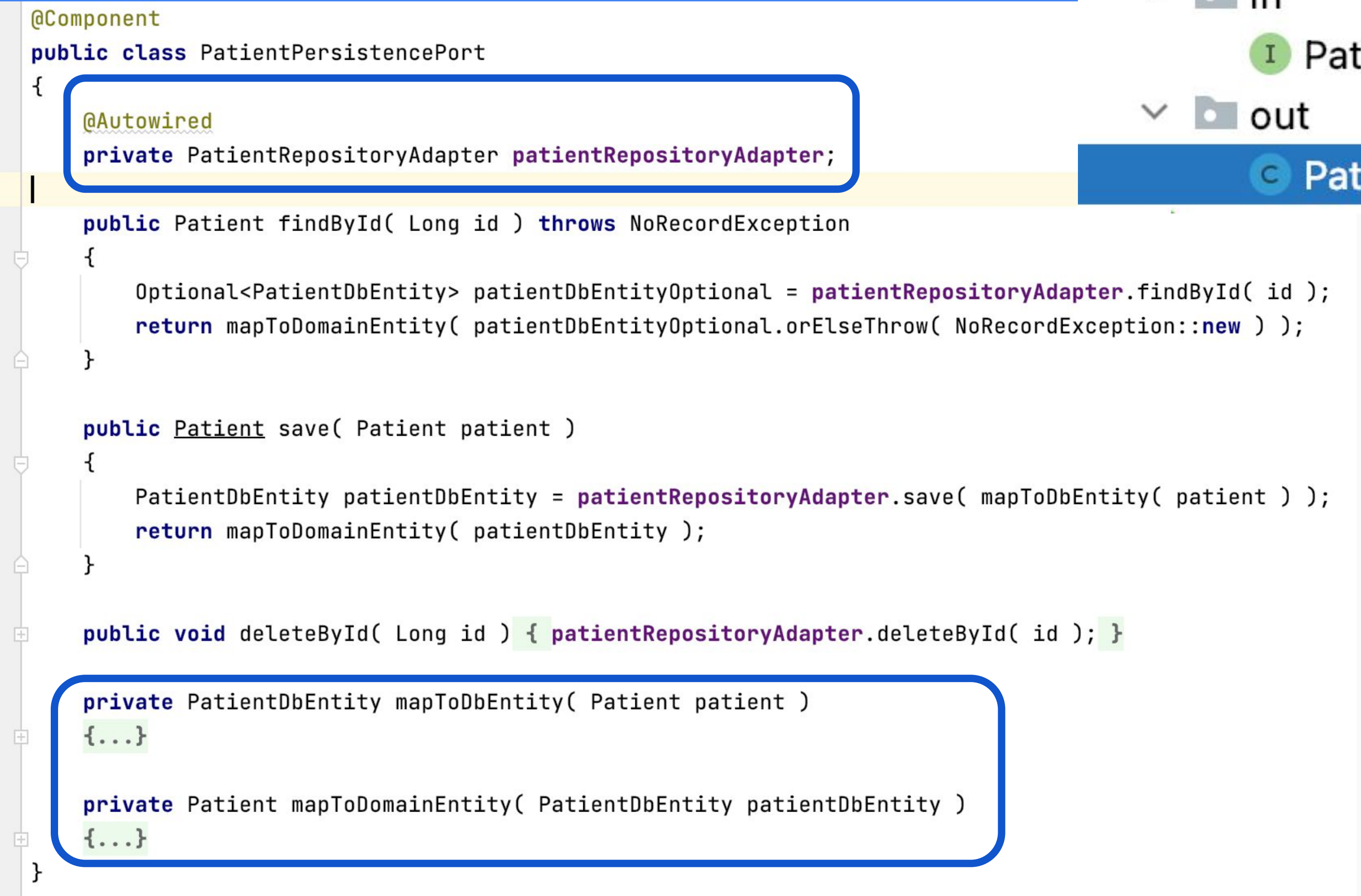
```
public interface PatientCrudUseCase // an alternative: command pattern
{
    Patient getPatientById( Long id ) throws NoRecordException;

    Patient savePatient( Patient patient );

    void deletePatientById( Long id );
}
```

Out Port Example

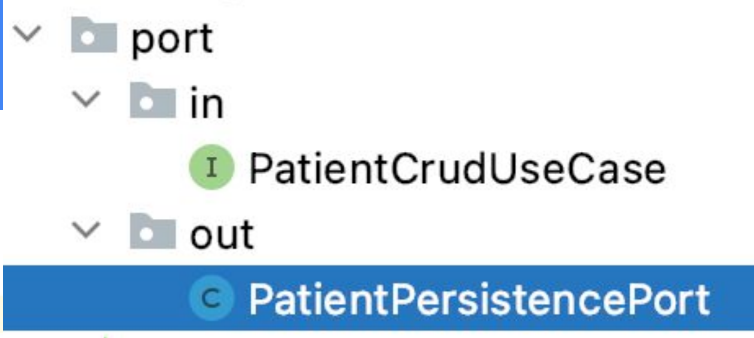


hexangonal
example
adapter
domain
exceptions
port
in
PatientCrudUseCase
out
PatientPersistencePort

Out Port Example

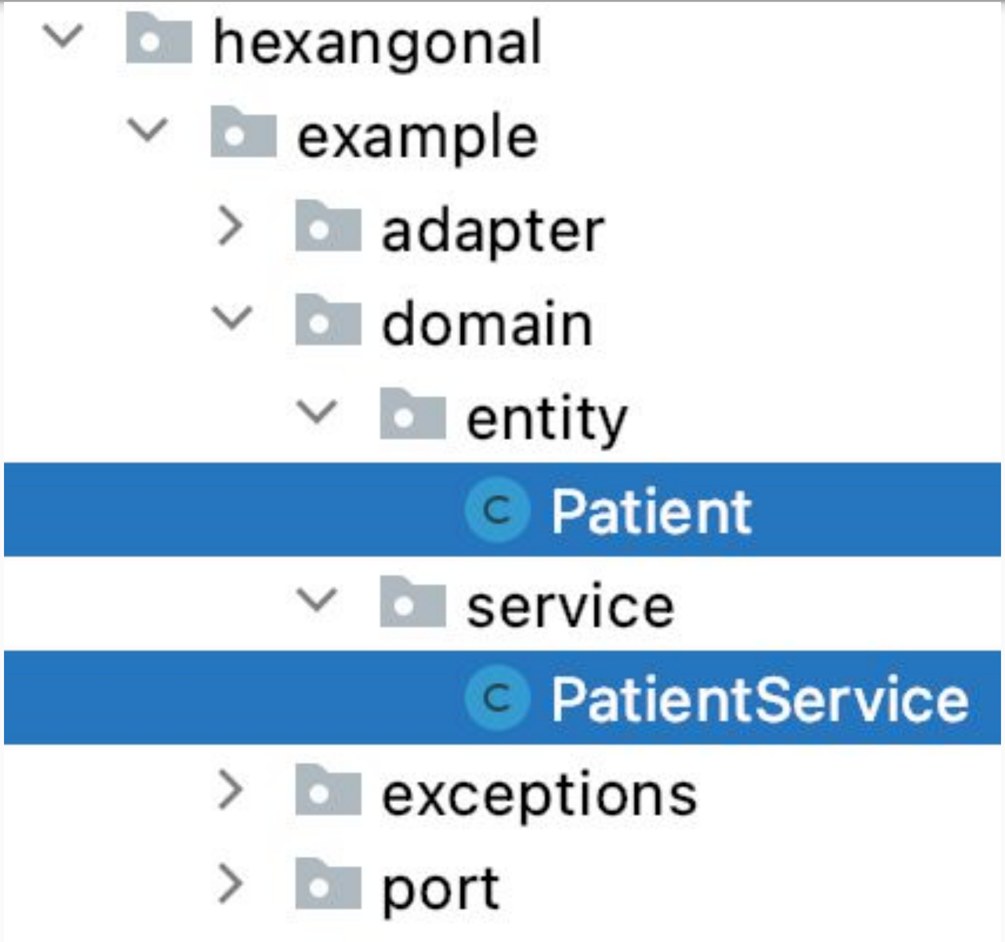


```
@Component
public class PatientPersistencePort
{
    @Autowired
    private PatientRepositoryAdapter patientRepositoryAdapter;

    public Patient findById( Long id ) throws NoRecordException
    {
        Optional<PatientDbEntity> patientDbEntityOptional = patientRepositoryAdapter.findById( id );
        return mapToDomainEntity( patientDbEntityOptional.orElseThrow( NoRecordException::new ) );
    }

    public Patient save( Patient patient )
    {
        PatientDbEntity patientDbEntity = patientRepositoryAdapter.save( mapToDbEntity( patient ) );
        return mapToDomainEntity( patientDbEntity );
    }

    public void deleteById( Long id ) { patientRepositoryAdapter.deleteById( id ); }

    private PatientDbEntity mapToDbEntity( Patient patient )
    {...}

    private Patient mapToDomainEntity( PatientDbEntity patientDbEntity )
    {...}
}
```



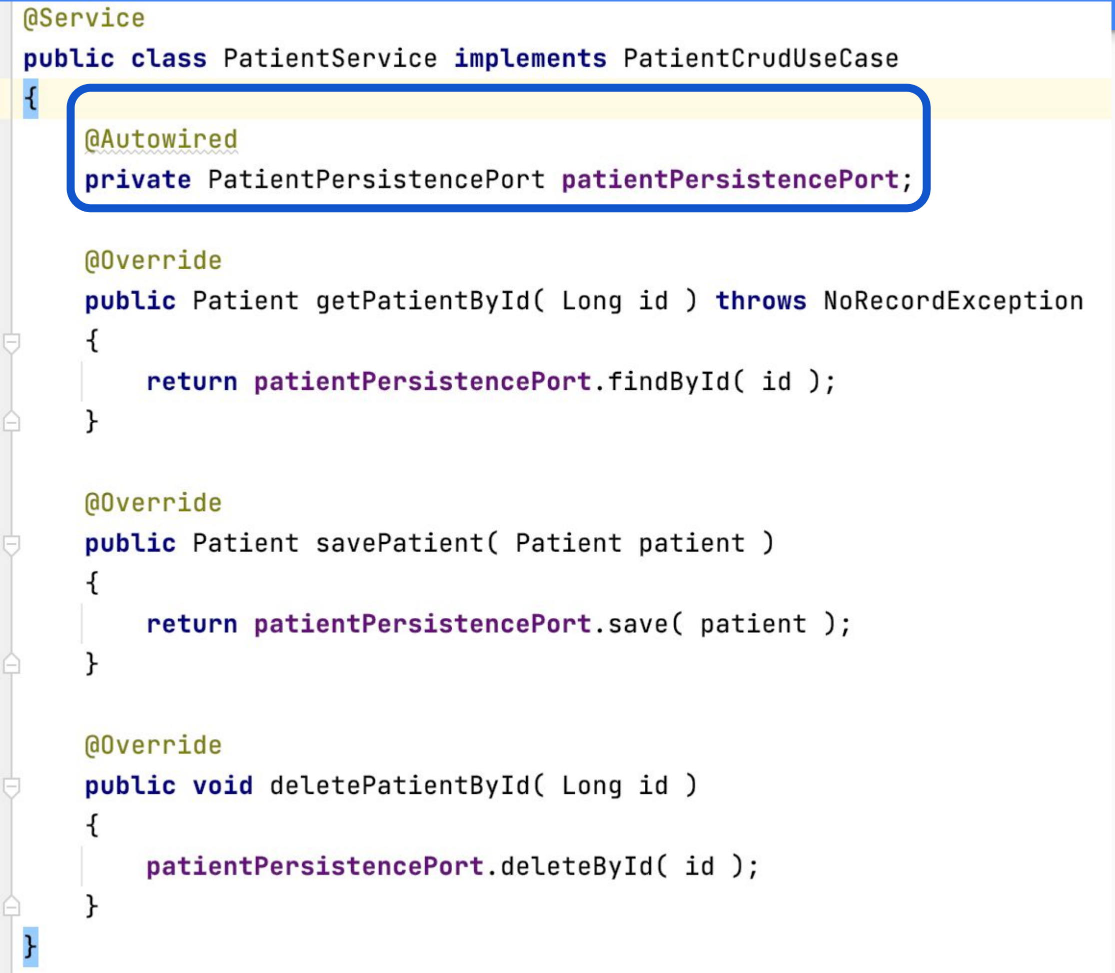
```
port
  in
    PatientCrudUseCase
  out
    PatientPersistencePort
```

Domain Example

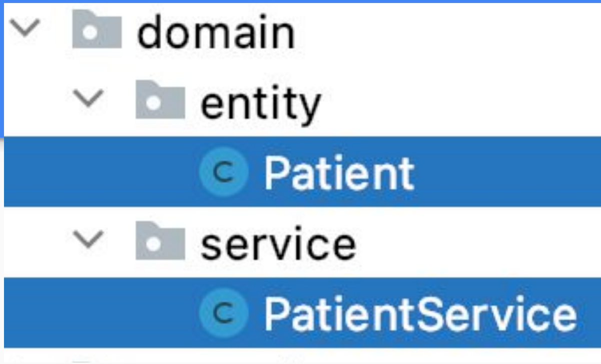


```
hexangonal
  example
    adapter
    domain
      entity
        Patient
      service
        PatientService
    exceptions
    port
```

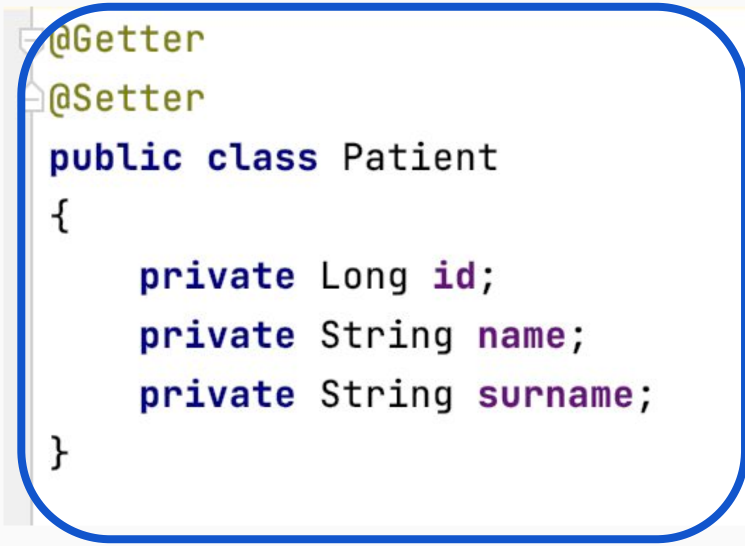
Domain Example



```
@Service
public class PatientService implements PatientCrudUseCase
{
    @Autowired
    private PatientPersistencePort patientPersistencePort;

    @Override
    public Patient getPatientById( Long id ) throws NoRecordException
    {
        return patientPersistencePort.findById( id );
    }

    @Override
    public Patient savePatient( Patient patient )
    {
        return patientPersistencePort.save( patient );
    }

    @Override
    public void deletePatientById( Long id )
    {
        patientPersistencePort.deleteById( id );
    }
}
```



```
domain
  entity
    Patient
  service
    PatientService
```

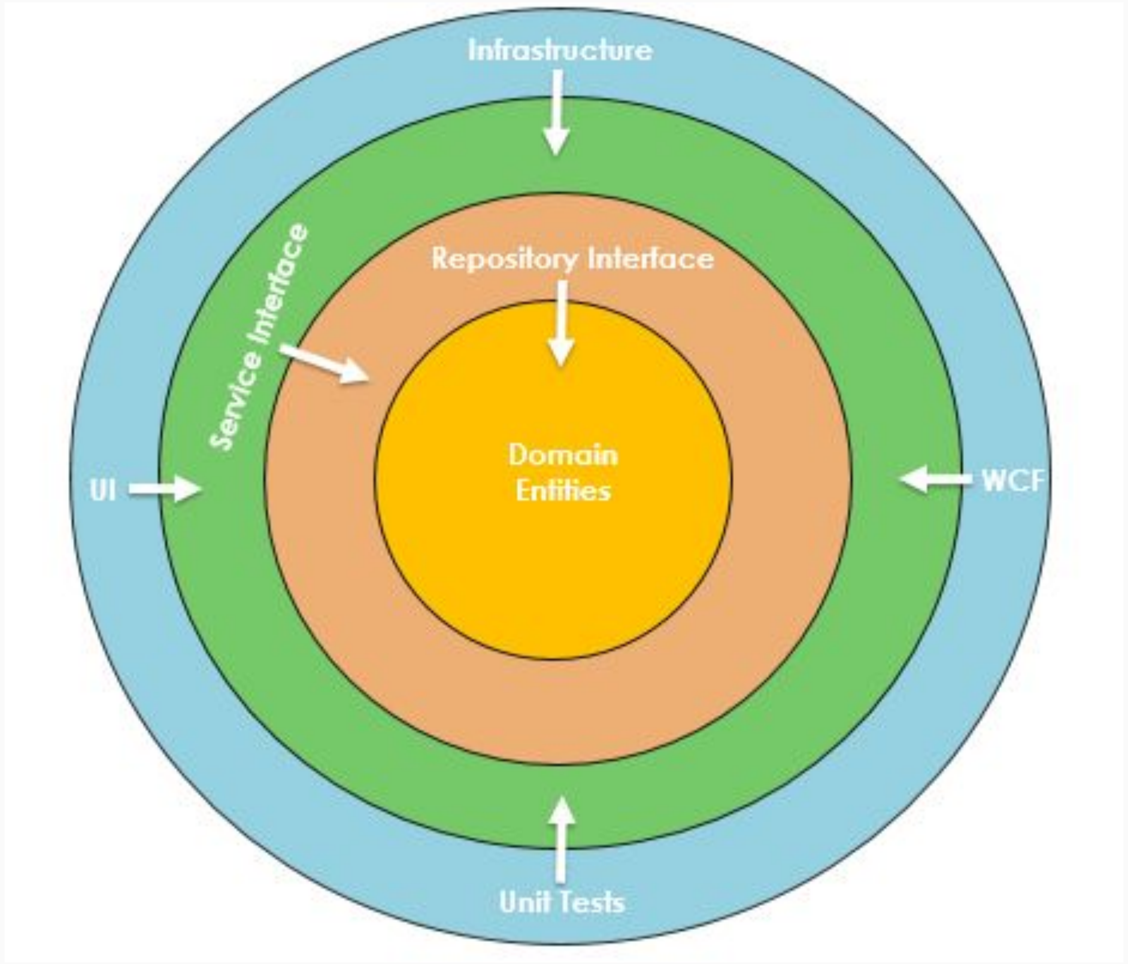


```
@Getter
@Setter
public class Patient
{
    private Long id;
    private String name;
    private String surname;
}
```

Variants

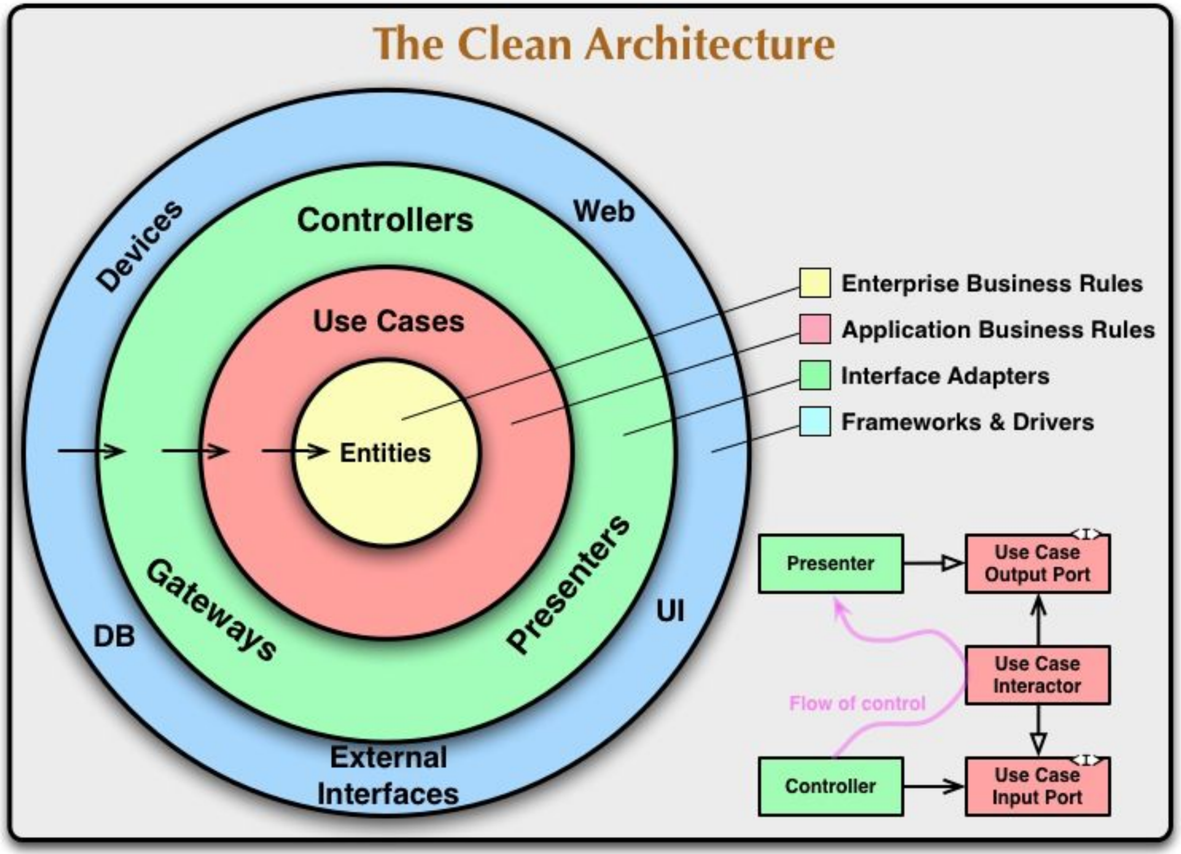
Onion Architecture

Lasagna Antipattern ???



Source:
https://www.codeguru.com/csharp/csharp/cs_misc/designtechniques/understanding-onion-architecture.html

Clean Architecture
(by Robert Martin)



Source: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

```
System.out.println( "Thanks!" );
```