Hands on Domain Driven Design

# Solution Structure

## Marketplace Event Sourcing

This assembly contains all the classes and shared logic that are needed to save, load, and project events across all bounded contexts

* TypedId
  + **Definition**: Abstract base class for ids in the system
  + **Usage:** Ids like User Id is now represented as a class that inherits TypedId
  + **Reasons:**
    - Leverages C# type system to force the correct type for derived ids
    - Introduces shared logic like simple validation when initializing an object
  + **Challenges:**
    - Serialization from C# CLR type to json. The Id now is an object which requires more work to customize json serialization to produce simple value instead of object value
    - ORMs like EF Core treats needs to be configured because now the Id used to be a primitive value and now it is a CLR type.
* TypeMapper: TODO written in code
* ISubscription
  + **Definition:** Interface for projecting an event
  + **Usage:** inherited by multiple store providers like RavenDb, SQL Server, and Event Store. These take and event and do some projections on it and save it to its underlying store
* IInternalEventHandler
  + **Definition:** Interface that provides a method to apply an event to Entity. This is implemented by the aggregate root and the entity classes. When an event is going to be handled by an entity, it should be handled also by the Aggregate Root.
* ICheckpointStore
  + **Definition:** Interfaces used to provide functionality to load and store a checkpoint
  + **Checkpoint:** is a state that tells what was the last event we read and projected successfully. So when we restart the app, it continues projection from last event we read not from the start of the stream.
* Entity
  + **Definition:** A class with mandatory Action<DomainEvent> which is a delegate of event handler passed from the Aggregate Root to the entity
  + **The When Method:** it takes an event and applies it to the entity state. It should never fail. It can only fail when validating the state on the aggregate root not the entity.
* AggregateRoot
  + **Definition:** root of entity tree. Controls the validity of the tree. And the tree is stored all together.
  + **The When Method** takes event and applies it to the state of the aggregate
  + **The Apply Method** calls the When Method, then checks the validity of the aggregate, and finally it adds the event to list of changes.

## Marketplace Event Store