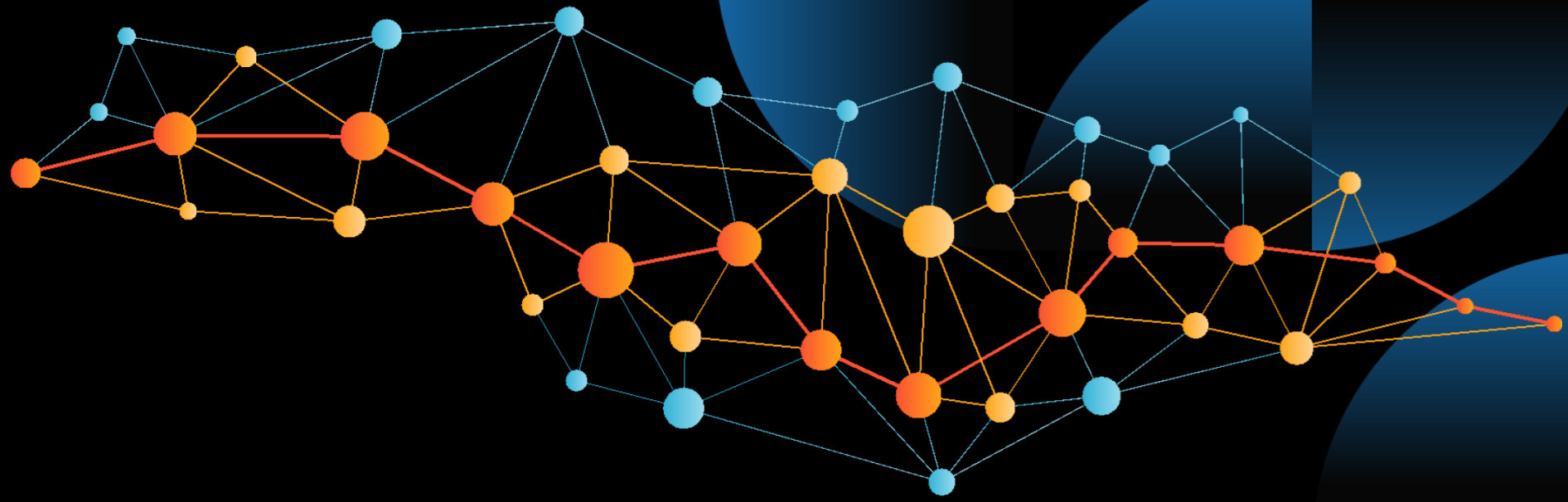
.NET DEVELOPER DAYS

CODEDESIGN



Florin Coroș

Implementing Clean Architecture

CodeDesign

Florin Coroș

Software Architect Consultant

Technical Trainer

Founder of Code Design

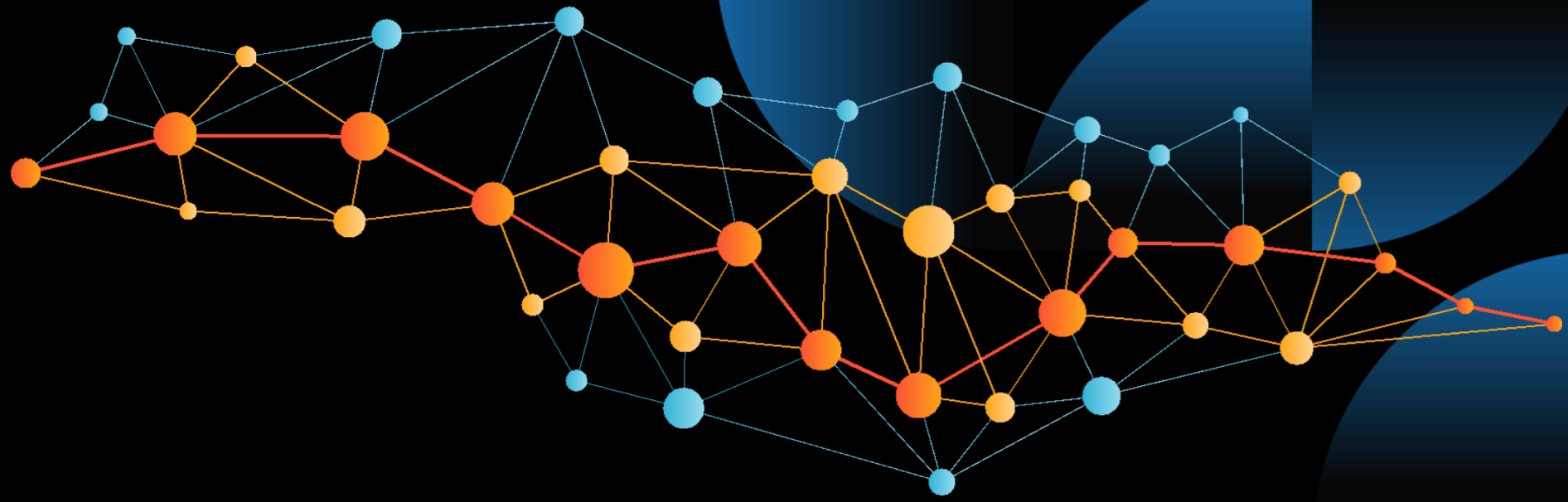
enjoing playing GO

enjoing traveling

onCodeDesign.com/DevDaysPL

.NET DEVELOPER DAYS

CODEDESIGN



Florin Coroș

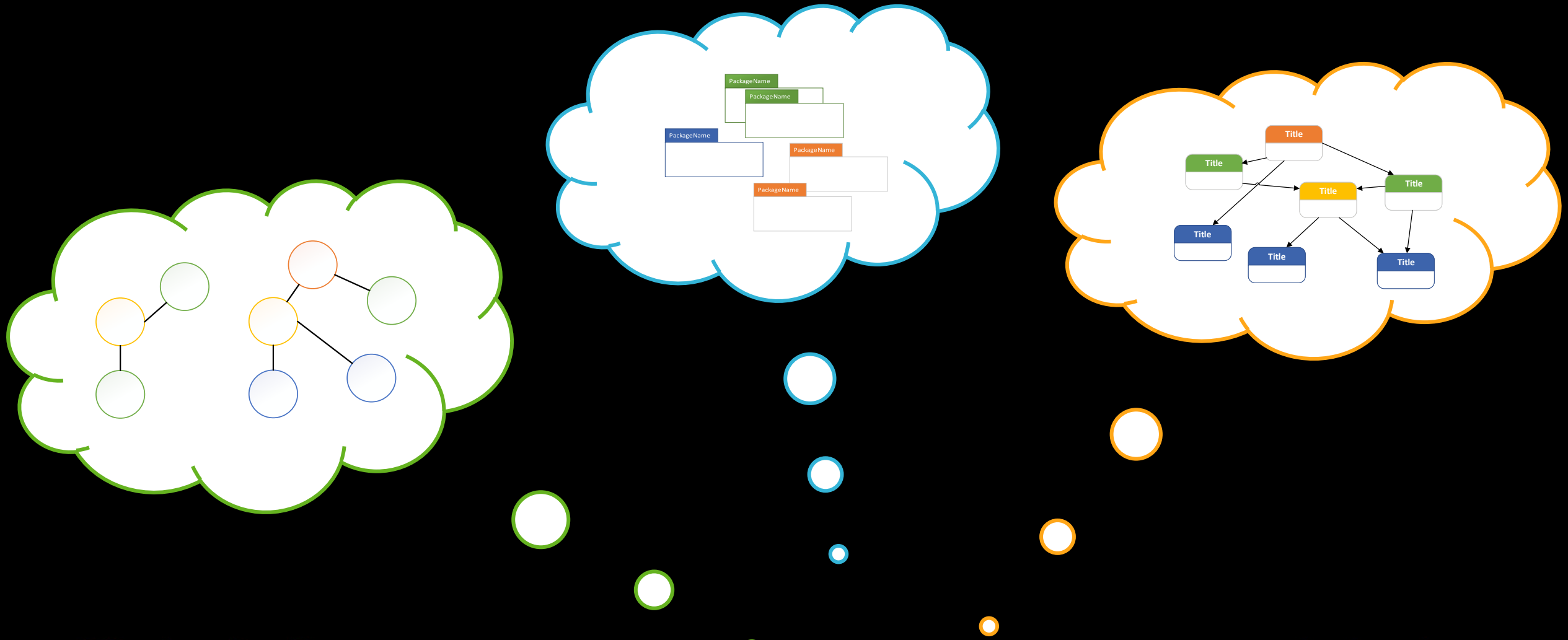
Implementing Clean Architecture

Why is Architecture Important

The goal of software architecture is to minimize the human resources required to build and maintain the required system

-- Robert C. Martin, Clean Architecture

Decompose - Separation of Concerns

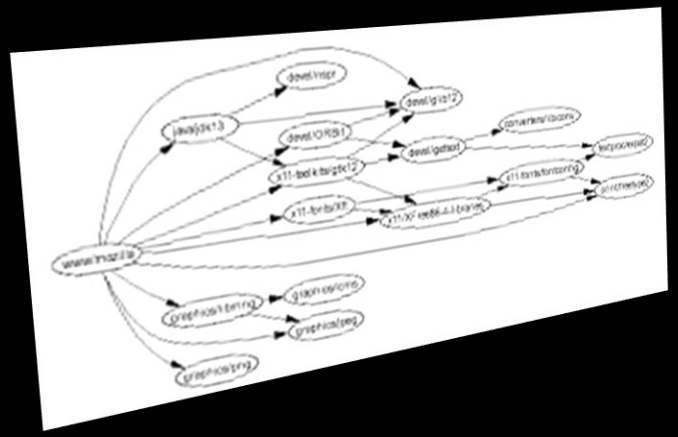


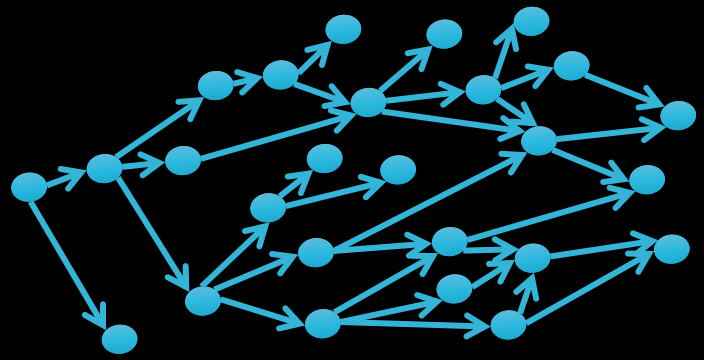
Manage the Complexity and Size

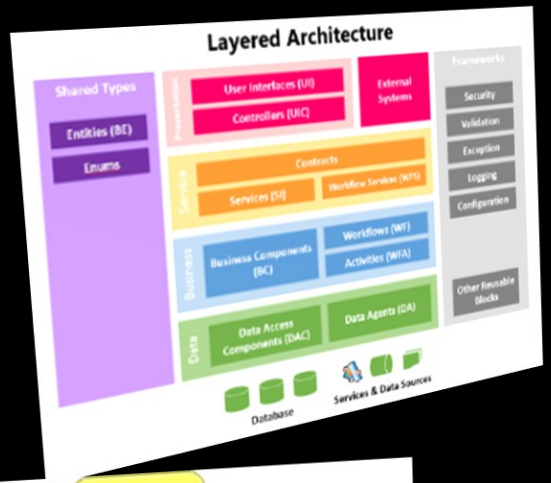
when projects do fail for reasons that are primarily technical, the reason is often

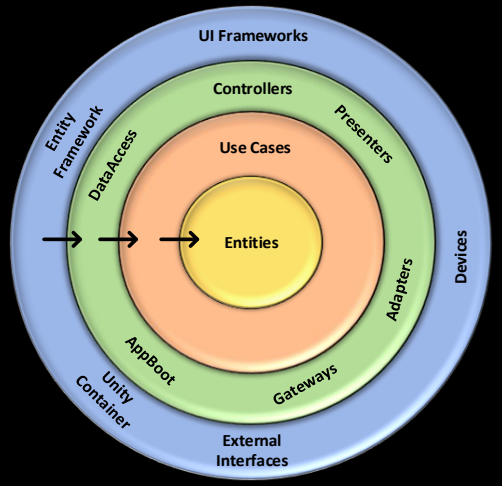
uncontrolled complexity

The Architecture is Separation of Concerns and a Set of Rules





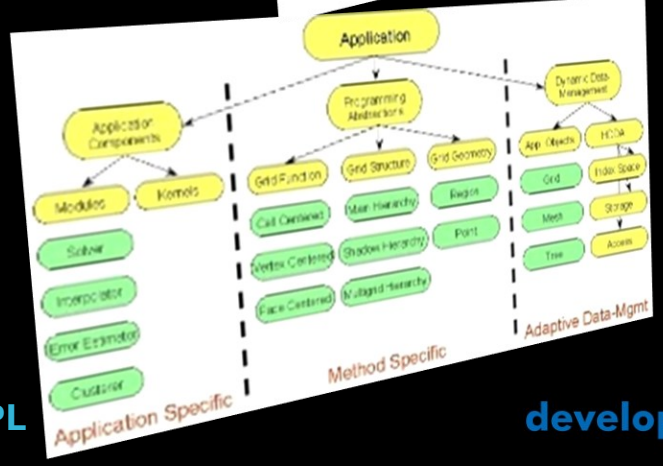




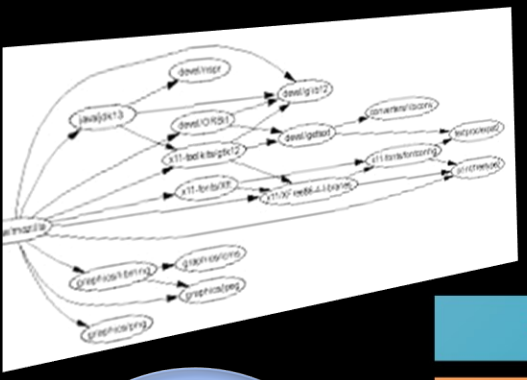
Presentation Layer

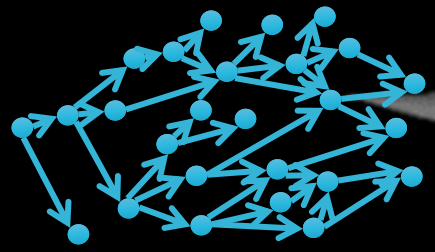
Business Logic Layer

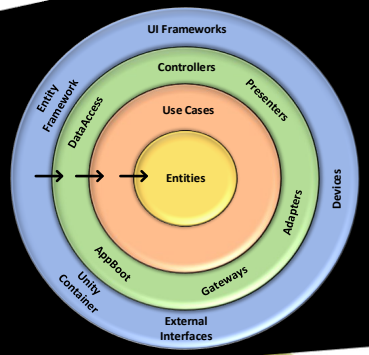
Data Access Layer



Implementing the Architecture



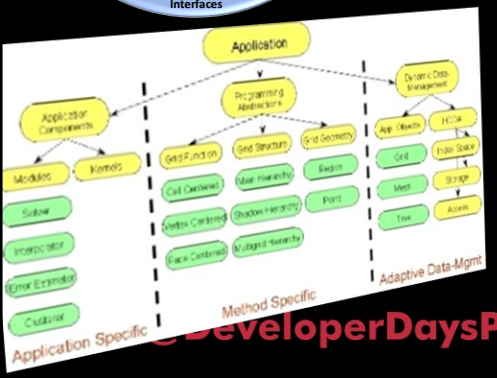




Presentation Layer

Business Logic Layer

Data Access Layer




```
private EfContext db = ne
//...
public A
```

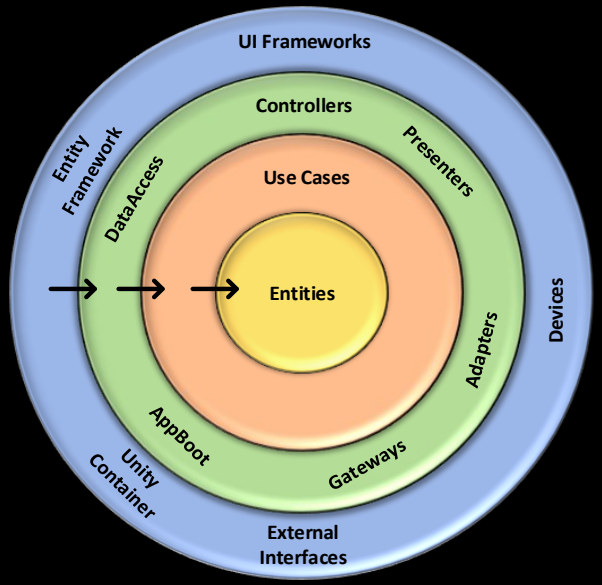
```
public void UpdateTaxStatus(TaxStatus item)
{
    // validations
    if (DateTimeUtils.IsStartDateBeforeEndDate(item.StartDate, item.EndDate) == false)
    {
        throw new ValidationAidaException(Localize.GetLocalizedResource("Start date cannot
    }

    // check if some hotel period will overlap with another period
    List<Hotel> hotels = DB.TaxStatusesHotels.Where(c => c.TaxStatusID == item.ID).Select(
    foreach (Hotel hotel in hotels)
    {
        TaxStatusesHotel taxStatusesHotel;
        bool isOverlapping = IsHotelHavingOverlappingTaxStatus(hotel.ID, item.ID, item.St
        if (isOverlapping)
        {
            string errorMessage = string.Format(Localize.GetLocalizedResource("The Hotel
                taxStatusesHotel.Hotel.Name,
                taxStatusesHotel.TaxStatus.Name,
```

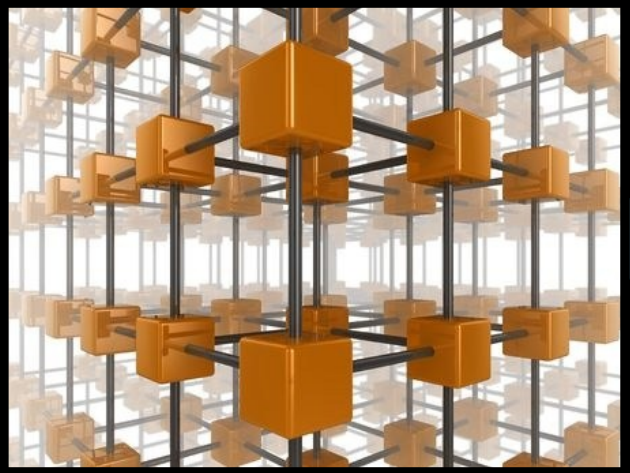
```
private rea

public List<
{
    var pers
        .Orc
        .Pag
        .Tol

    //. . .
}
```

```
IDbCommand
cmd.Parame

using (IDb
{
    cmd.Co
    using
    {
        wh
        {

        }
    }
}
```

```
foreach (Hotel hotel in hotels)
{
    TaxStatusesHotel taxStatusesHotel;
    bool isOverlapping = IsHotelHavingOverlappingTaxStatus(hotel.ID, item.ID, item.StartDat
    if (isOverlapping)
    {
        string errorMessage = string.Format(Localize.GetLocalizedResource("The Hotel '{0}'
            taxStatusesHotel.Hotel.Name,
            taxStatusesHotel.TaxStatus.Name,
            taxStatusesHotel.TaxStatus.StartDate.ToShortDateString(),
            taxStatusesHotel.TaxStatus.EndDate.ToShortDateString());
        throw new ValidationAidaException(string.Format(errorMessage, item.ID), ScreenName
    }
}

base.Update(item, true);
}

public void DeleteTaxStatus(long itemID)
{
    TaxStatus item = DB.TaxStatuses.FirstOrDefault(c => c.ID == itemID);
    if (item == null)
    {
        throw new ValidationAidaException(
            string.Format(Localize.GetLocalizedResource("There is no object with such ID: ({0})
    }
    if (item.StartDate <= DateTime.Now)
    {
        throw new ValidationAidaException(
            Localize.GetLocalizedResource(Localize.GetLocalizedResource("You cannot delete a pa
```

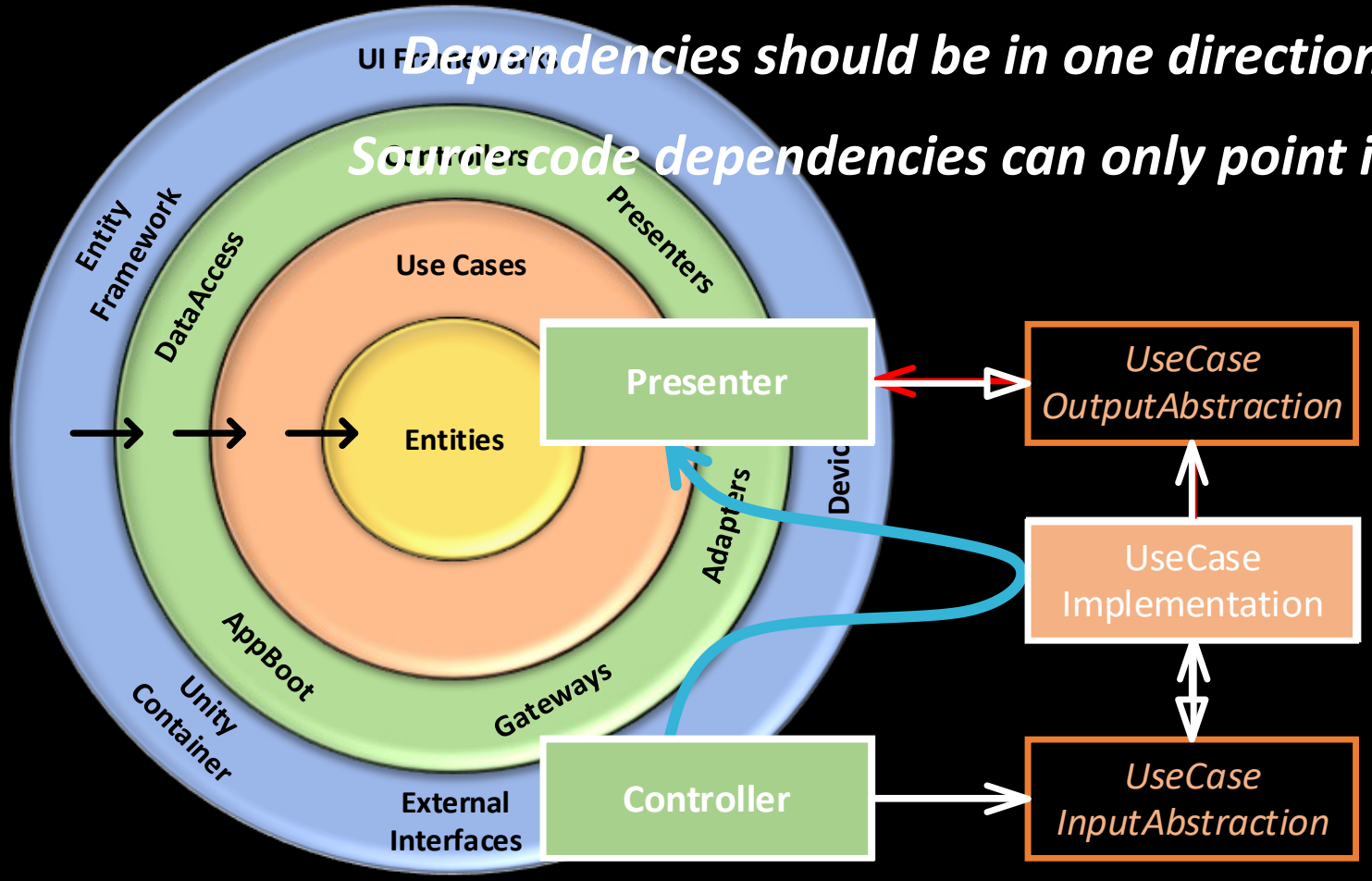
Structure that Supports the Architecture





```
public void UpdateTaxStatus(TaxStatus item)
{
    // validations
    if (DateTimeUtils.IsStartDateBeforeEndDate(item.StartDate, item.EndDate) == false)
    {
        throw new ValidationAidaException(Localize.GetLocalizedResource("Start date cannot be
    }

    // check if some hotel period will overlap with another period
    List<Hotel> hotels = DB.TaxStatusesHotels.Where(c => c.TaxStatusID == item.ID).Select(c =>
    foreach (Hotel hotel in hotels)
    {
        TaxStatusesHotel taxStatusesHotel;
        bool isOverlapping = IsHotelHavingOverlappingTaxStatus(hotel.ID, item.ID, item.StartDat
        if (isOverlapping)
foreach (Hotel hotel in hotels)
{
    TaxStatusesHotel taxStatusesHotel;
    bool isOverlapping = IsHotelHavingOverlappingTaxStatus(hotel.ID, item.ID, item.StartDat
    if (isOverlapping)
    {
        string errorMessage = string.Format(Localize.GetLocalizedResource("The Hotel '{0}'
            taxStatusesHotel.Hotel.Name,
            taxStatusesHotel.TaxStatus.Name,
            taxStatusesHotel.TaxStatus.StartDate.ToShortDateString(),
            taxStatusesHotel.TaxStatus.EndDate.ToShortDateString());
        throw new ValidationAidaException(string.Format(errorMessage, item.ID), ScreenName,
    }
}

base.Update(item, true);
}

public void DeleteTaxStatus(long itemID)
{
    TaxStatus item = DB.TaxStatuses.FirstOrDefault(c => c.ID == itemID);
    if (item == null)
    {
        throw new ValidationAidaException(
            string.Format(Localize.GetLocalizedResource("There is no object with such ID: ({0})
    }
    if (item.StartDate <= DateTime.Now)
    {
        throw new ValidationAidaException(
            Localize.GetLocalizedResource(Localize.GetLocalizedResource("You cannot delete a pa
    }

    base.Delete(DB.TaxStatuses, itemID, true);
    //DB.TaxStatuses.Remove(item);
    DB.SaveChanges();
}
#endregion
```

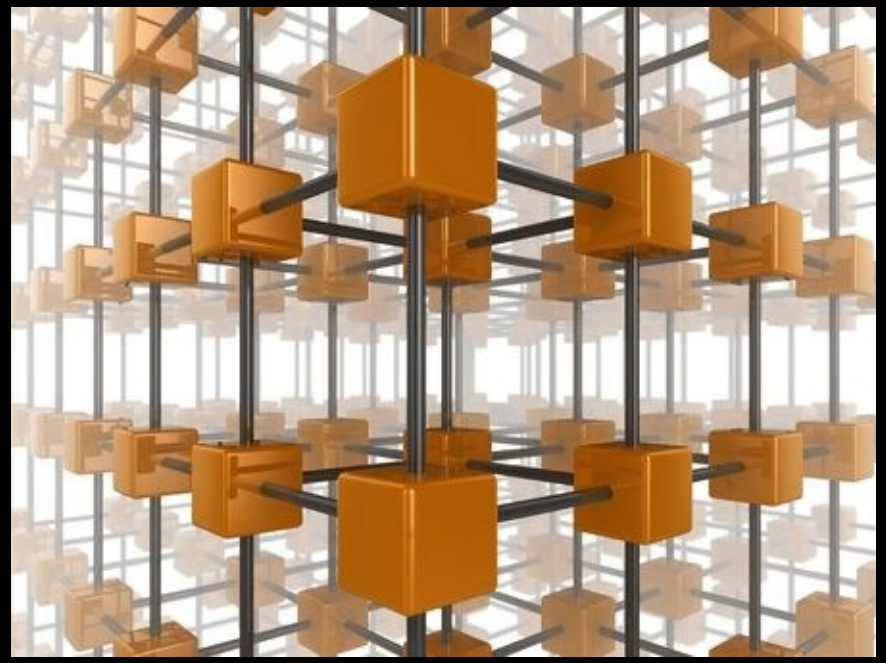
Clean Architecture



http://blog.8thlight.com/uncle-bob/2012/08/13/the-clean-architecture.html

Implementing Clean Architecture through Structure

Create a structure that makes it difficult to write bad code and it makes it easy to write good code, code that follows the architecture and the design

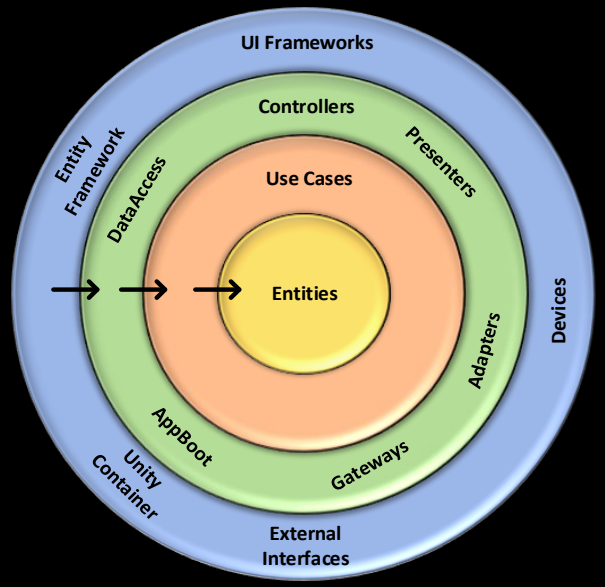


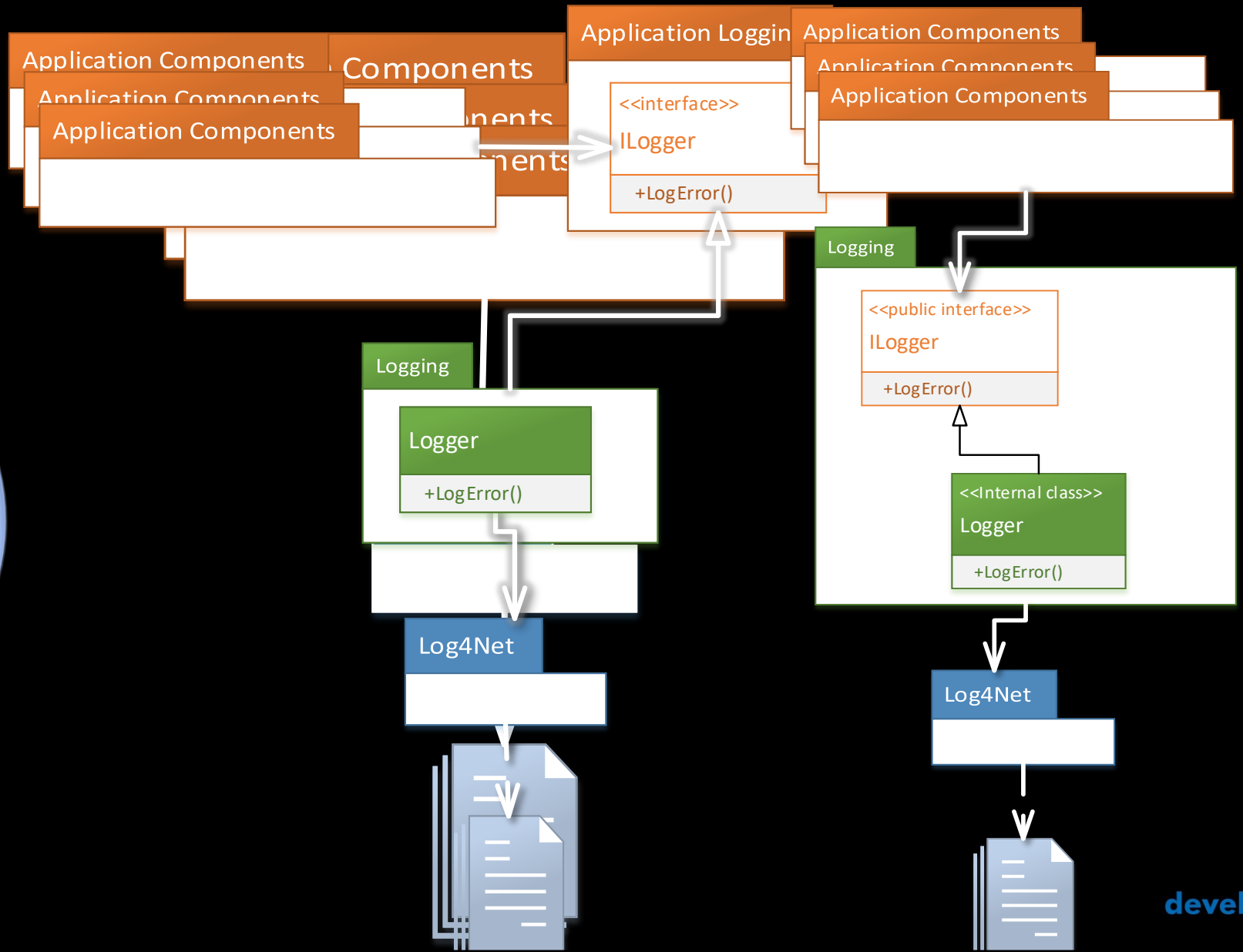
Hide external frameworks to enforce the way they are used

Use assemblies and references among them to **enforce dependencies rules**

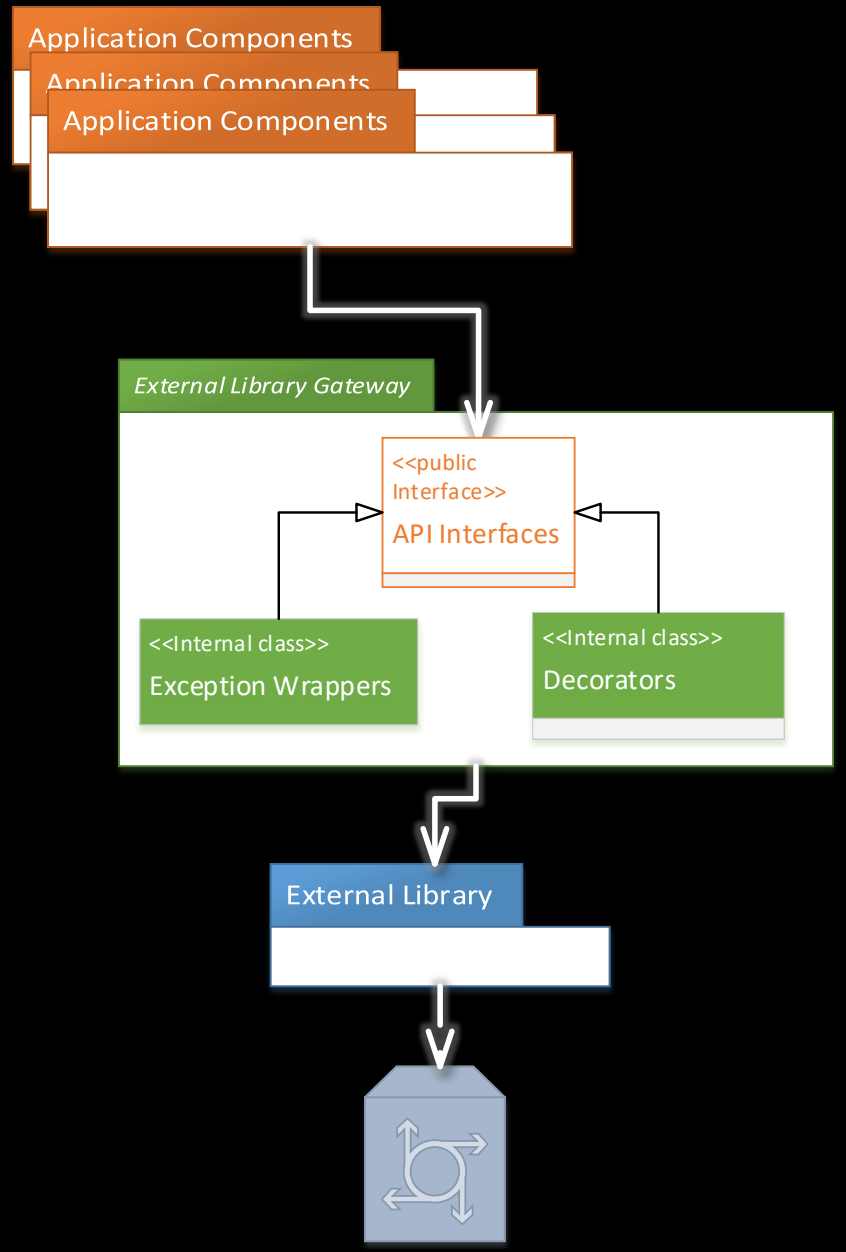
Enforce *Constructor Dependency Injection* that encourages *Programming Against Interfaces*

Creating a Structure for Clean Architecture

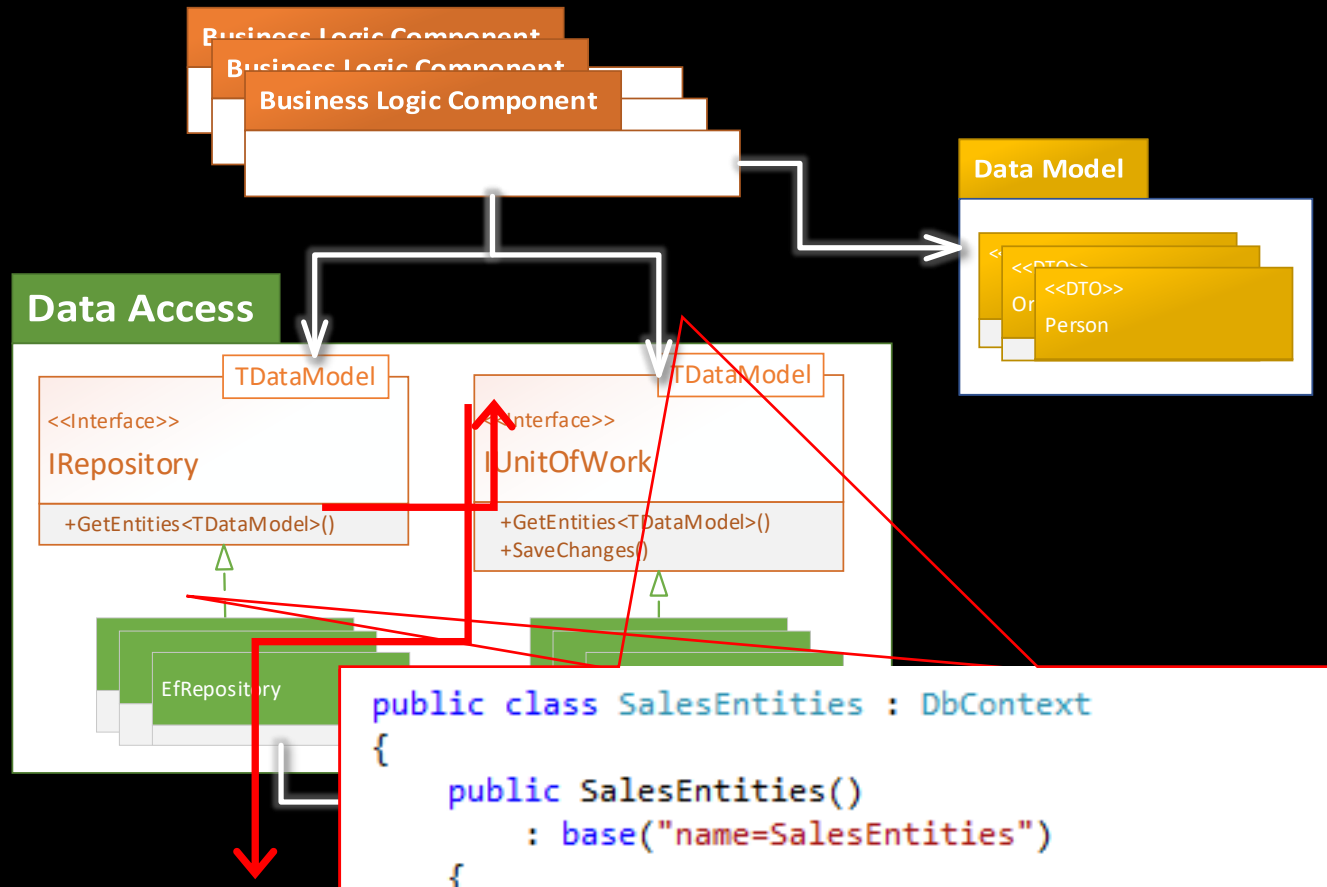




Hide External Libraries from App Code



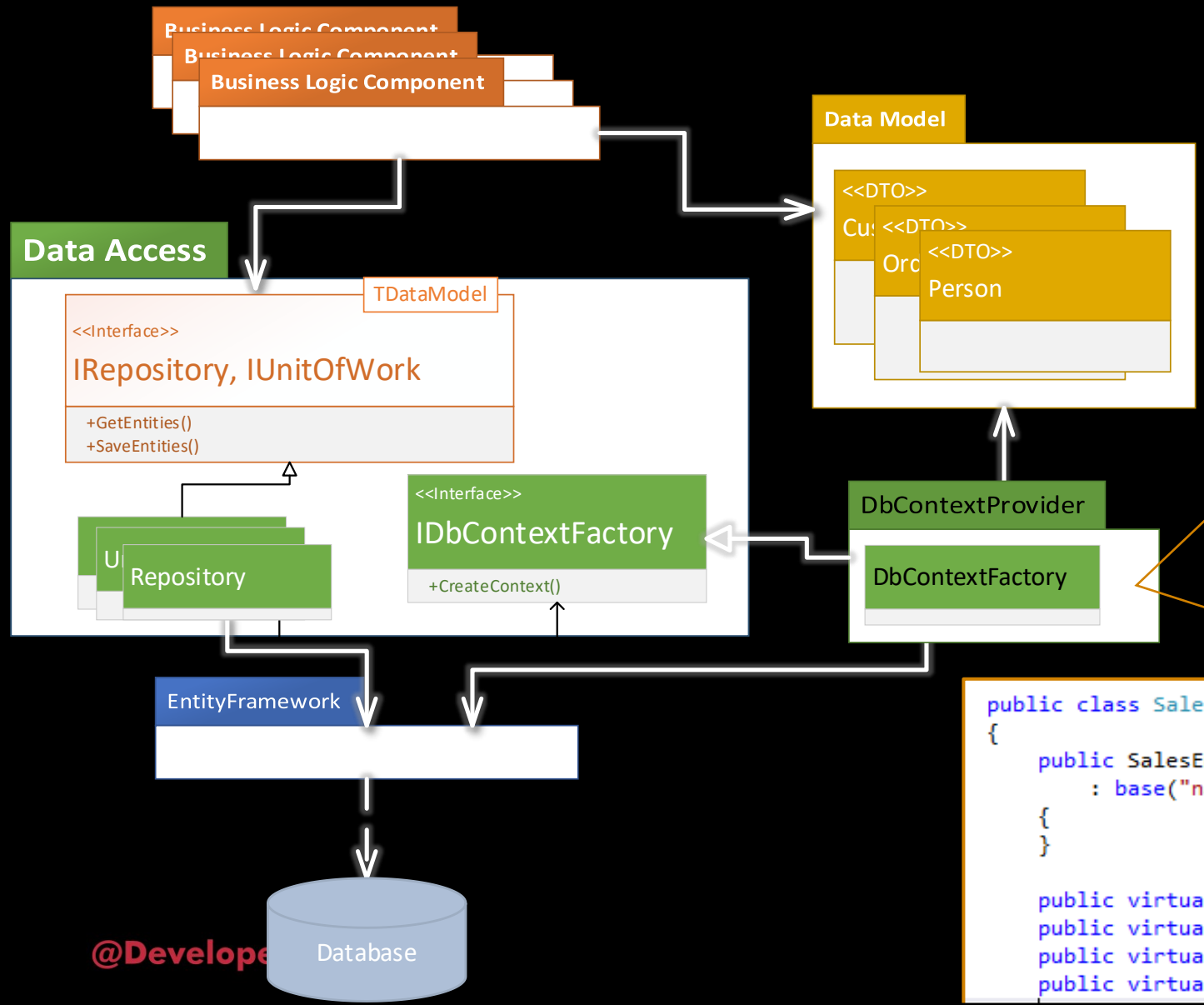
Enforce Separation of Data Access Concerns




```
public class SalesEntities : DbContext
{
    public SalesEntities()
        : base("name=SalesEntities")
    {
    }

    public virtual DbSet<Customer> Customers { get; set; }
    public virtual DbSet<SalesOrderHeader> SalesOrderHeaders { get; set; }
    public virtual DbSet<SalesOrderDetail> SalesOrderDetails { get; set; }
    public virtual DbSet<CountryRegionCurrency> CountryRegionCurrencies { get; set;
```

/AppInfra-Training/Infra/DataAccess

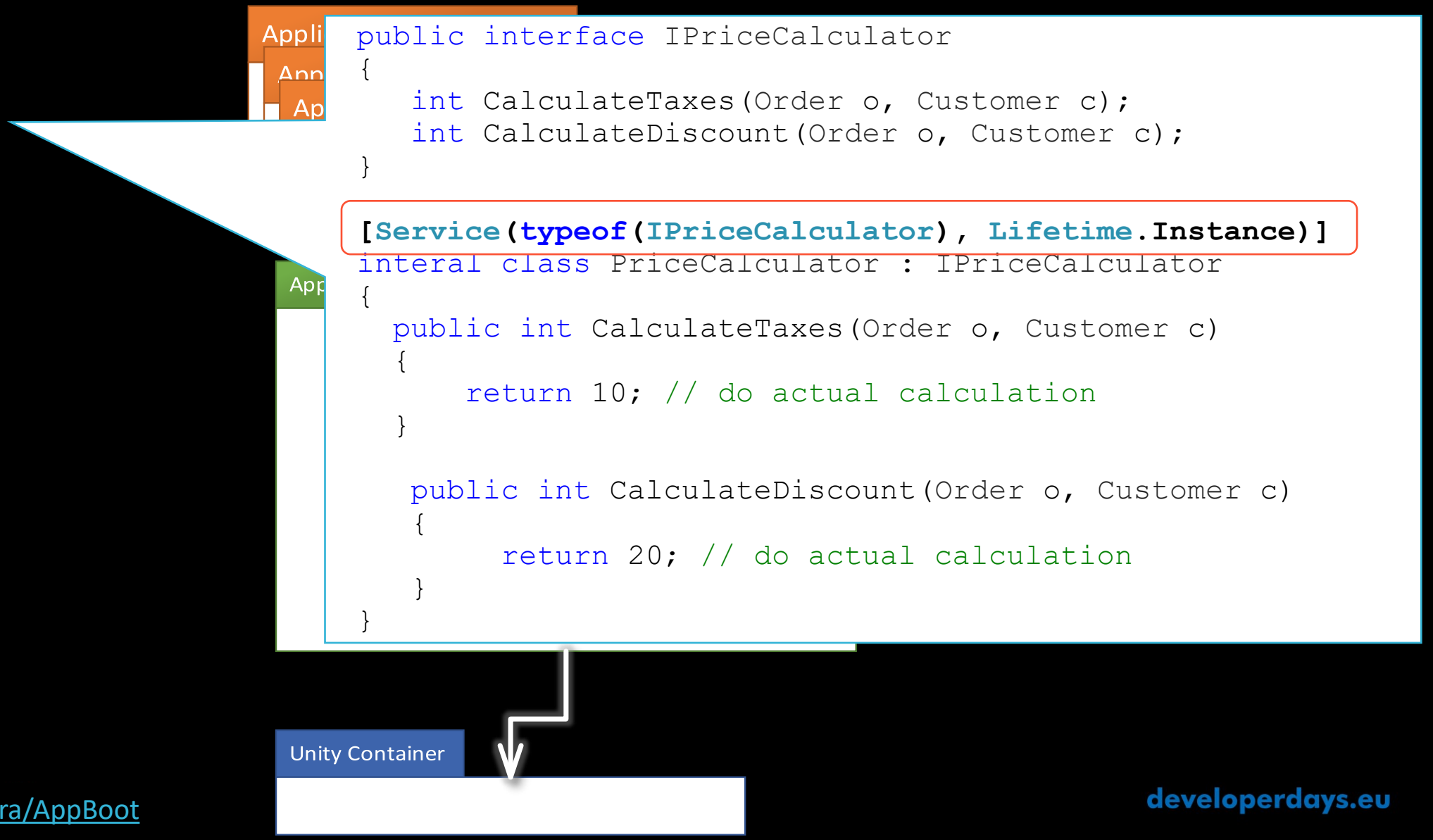
Factor-out to Enforce Separation of Data Access




```
[Service(typeof(IDbContextFactory))]
public class DbContextFactory : IDbContextFactory
{
    public IDbContextWrapper CreateContext()
    {
        var context =  new SalesEntities();
        return new DbContextWrapper(context);
    }
}
```

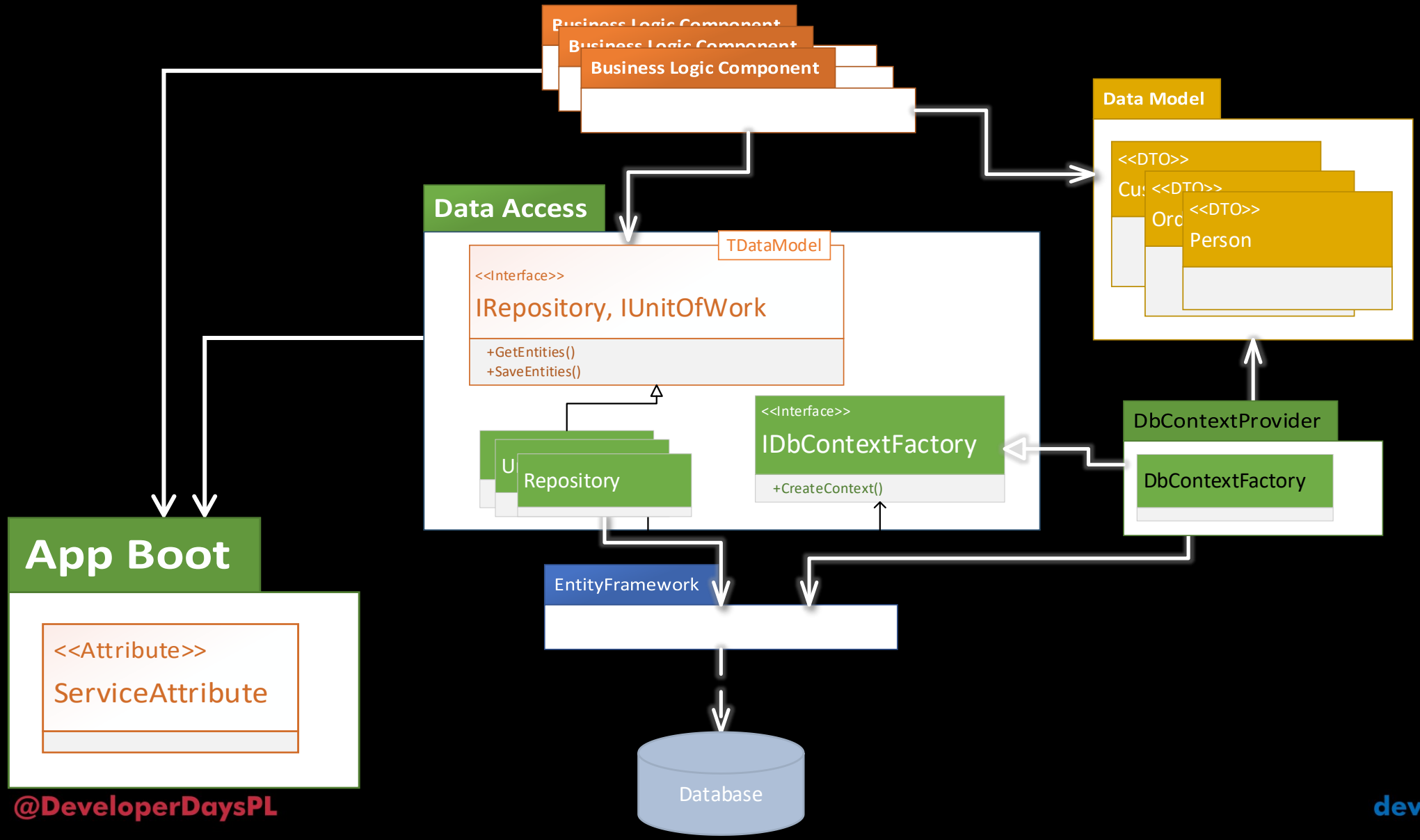
```
public class SalesEntities : DbContext
{
    public SalesEntities()
        : base("name=SalesEntities")
    {
    }

    public virtual DbSet<Customer> Customers { get; set; }
    public virtual DbSet<SalesOrderHeader> SalesOrderHeaders { get; set; }
    public virtual DbSet<SalesOrderDetail> SalesOrderDetails { get; set; }
    public virtual DbSet<CountryRegionCurrency> CountryRegionCurrencies { get; set; }
```

AppBoot Hides the DI Framework under Abstractions



/AppInfra-Training/Infra/AppBoot

AppBoot: DI Abstractions & Type Discovery



Patterns on how most of the code is written

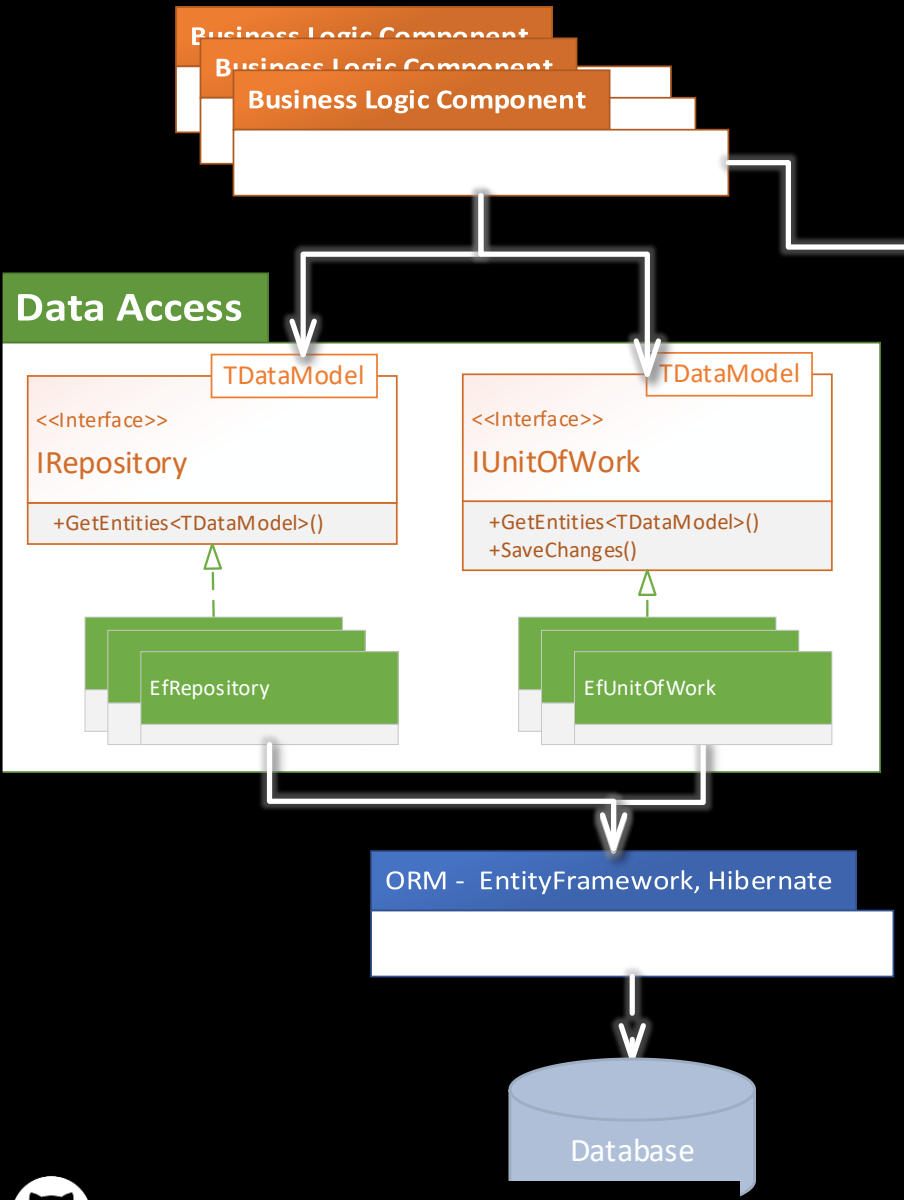
```
[Service(typeof (IOrderingService))]
private class OrderingService : IOrderingService
{
   private readonly IRepository repository;
   private readonly IPriceCalculator calculator;
   private readonly IApprovalService orderApproval;

 public OrderingService(IRepository repository, IPriceCalculator calculator, IApprovalService orderApproval)
   {
       this.repository = repository;
       this.calculator = calculator;
       this.orderApproval = orderApproval;
   }

    public SalesOrderInfo[] GetOrdersInfo(string customerName)
    {
        var orders = repository.GetEntities<SalesOrderHeader>()
        ...
        return orders.ToArray();
    }

    public SalesOrderResult PlaceOrder(string customerName, OrderRequest request)
    {
        ...
    }
}
```

Enforce Constructor DI to Prevent Circular Dependencies

```
[Service(typeof(IApprovalService))]
class ApprovalService : IApprovalService
{
  private readonly IPriceCalculator priceCalculator;

  public ApprovalService(IPriceCalculator priceCalculator)
  {
    this.priceCalculator = priceCalculator;
  }
...
}
```

```
[Service(typeof (IPriceCalculator), Lifetime.Instance)]
public class PriceCalculator : IPriceCalculator
{
  private readonly IApprovalService approvalService;

  public PriceCalculator(IApprovalService approvalService)
  {
    this.approvalService = approvalService;
  }
...
}
```

Encapsulate Data Access Concerns




```
[Service(typeof (IRepository))]
internal class EfRepository : IRepository, IDisposable
{
   private readonly IDbContextFactory contextFactory;
   private readonly IInterceptorsResolver interceptorsResolver;
   private DbContext context;
   private readonly IEnumerable<IEntityInterceptor> interceptors;

   public EfRepository(IDbContextFactory contextFactory,
                       IInterceptorsResolver resolver)
   {
      this.contextFactory = contextFactory;
      this.interceptorsResolver = interceptorsResolver;
      this.interceptors =
         resolver.GetGlobalInterceptors();
   }
...
}
```

/AppInfra-Training/Infra/DataAccess

onCodeDesign.com/DevDaysPL

developerdays.eu

Create Separated Patterns for Read-Only and Read-Write

```
public interface IRepository
{
  IQueryable<TDbEntity> GetEntities<TDbEntity>() where TDbEntity : class;
  IUnitOfWork CreateUnitOfWork();
}


public interface IUnitOfWork : IRepository, IDisposable
{
  void SaveChanges();
  void Add<T>(T entity) where T : class;
  void Delete<T>(T entity) where T : class;
  void BeginTransactionScope(SimplifiedIsolationLevel isolation);
}
```

Patterns for Read-Only data

```
public class OrdersController : Controller
{
  private readonly IRepository repository;
  public OrdersController(IRepository repository)
  {
     this.repository = repository;
  }

  public IActionResult Index(string customer)
  {
     var orders = repository.GetEntities<SalesOrderHeader>()
        .Where(soh => soh.Customer.Person.LastName == customer)
        .Select(soh => new OrdersListViewModel
        {
           CustomerName = customer,
           Number = soh.SalesOrderNumber,
           SalesPersonName = soh.SalesPerson,
           DueDate = soh.DueDate,
        });

    return View(orders);
  }
...
}
```

Patterns for Read-Write data

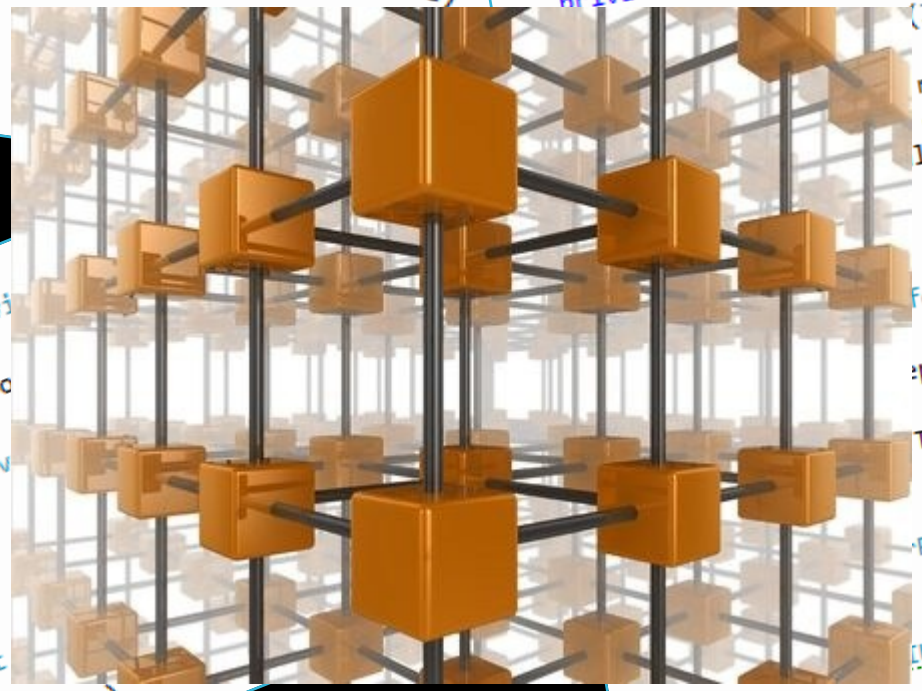
```
public class OrdersController : Controller
{
...
  [HttpPost]
  [ValidateAntiForgeryToken]
   public IActionResult PlaceOrder(OrderRequestViewModel model)
   {
      ...
    using (IUnitOfWork uof = repository.CreateUnitOfWork())
    {
        SalesOrderHeader order = uof.GetEntities<SalesOrderHeader>()
            .FirstOrDefault(o => o.CustomerID == c.ID && o.OrderDate.Month == DateTime.Now.Month);
        if (order == null)
        {
              order = new SalesOrderHeader {Customer = c};
              uof.Add(order);
        }
        AddRequestToOrder(model, order);

      uof.SaveChanges();
    }
     ...
   }
}
```

Consistency Creates Optimizations Opportunities

```
internal class EfRepository : IRepository, IDisposable
{
  public IQueryable<T> GetEntities<T>() where T : class
  {
     return Context.Set<T>().AsNoTracking();
  }

  public IUnitOfWork CreateUnitOfWork()
  {
     return new EfUnitOfWork(contextFactory, interceptorsResolver);
  }

  private sealed class EfUnitOfWork : IUnitOfWork
  {
    private DbContext context;
    private TransactionScope transactionScope;

    private readonly IDbContextFactory contextFactory;

    public EfUnitOfWork(IDbContextFactory contextFactory, IInterceptorsResolver resolver)
    {
       this.contextFactory = contextFactory;
       this.interceptorsResolver = interceptorsResolver;
    }
}
```

Create Development Patterns in Code

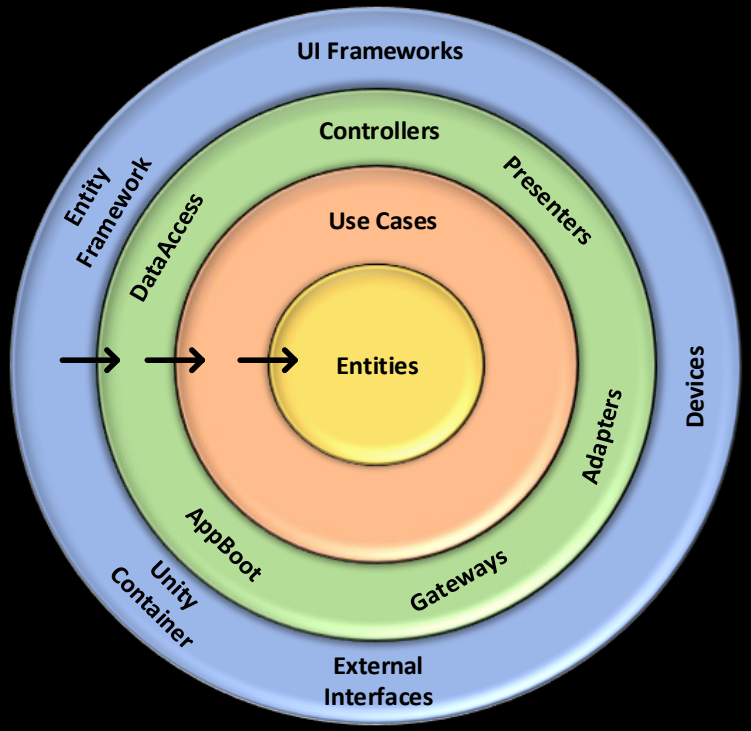
```
[Service(typeof (IPriceCalculator), Lifetime.Instance)]
public class PriceCalculator : IPriceCalculator
{
    public decimal CalculateTaxes(OrderRequest o, Customer c)
    {
        // do actual calculation
        return 10;
    }
```

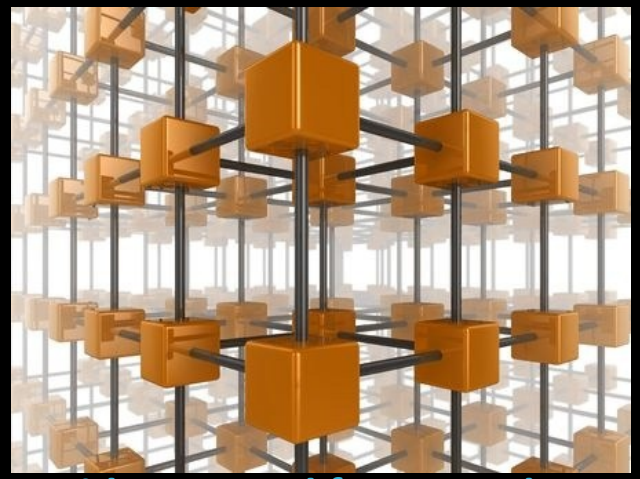


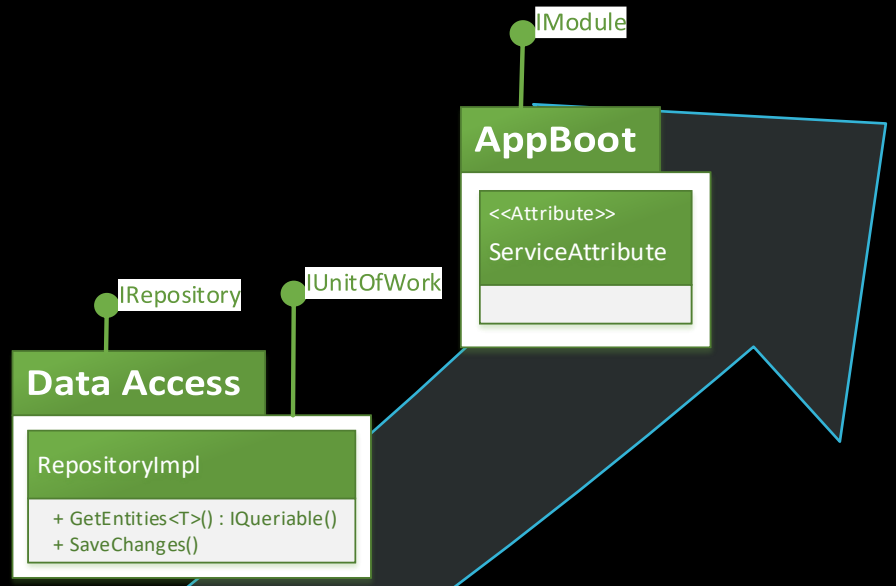
```
[Service(typeof (IOrderingService))]
class OrderingService : IOrderingService
{
    private readonly IRepository repository;
    private readonly IPriceCalculator calculator;
    private readonly IApprovalService orderApproval;

    (IRepository repository, IPriceCalculator calculator, IApprova

    repository;
    calculator;
    l = orderApproval;

    fo[] GetOrdersInfo(string customerName)

    epository.GetEntities<SalesOrderHeader>();

    ToArray();

    Result PlaceOrder(string customerName, OrderRequest request)

    IUnitOfWork uof = repository.CreateUnitOfWork())

    {
        ..
    }

    var taxes = calculator.CalculateTaxes(order, customer);
    var discount = calculator.CalculateDiscount(order, customer);
    ...
    new SalesOrderResult {State = OrderResultState.Invalid};
```

```
ervice(typeof(IApprovalService))]
lass CompositeApprovalService : IApprovalServi
{
    private readonly IApprovalService[] appro

    public CompositeApprovalService(IApprov
    {
        this.approvals = approvals;
    }

    public bool Approve(ApproveRequest
    {
        foreach (var approval in approvals)
        {
```

Implementing Clean Architecture through Structure







Hide external frameworks to enforce the way they are used

Use assemblies and references among them to **enforce dependencies rules**

Enforce *Constructor Dependency Injection* that encourages *Programming Against Interfaces*

CODEDESIGN

Please rate this session using

.NET DeveloperDays Mobile App
(available in AppStore & Google Play)

my LinkedIn

florin@onCodeDesign.com
linkedin.com/in/florincoros
oncodedesing.com/training
github.com/onCodeDesign/AppInfra-Training
calendly.com/code-design/florin-short-call

onCodeDesign.com/DevDaysPL