

FIA P GRADUAÇÃO

ENTERPRISE APPLICATION DEVELOPMENT

Prof. Me. Thiago T. I. Yamamoto

#08 – DESIGN PATTERNS



thiagoyama



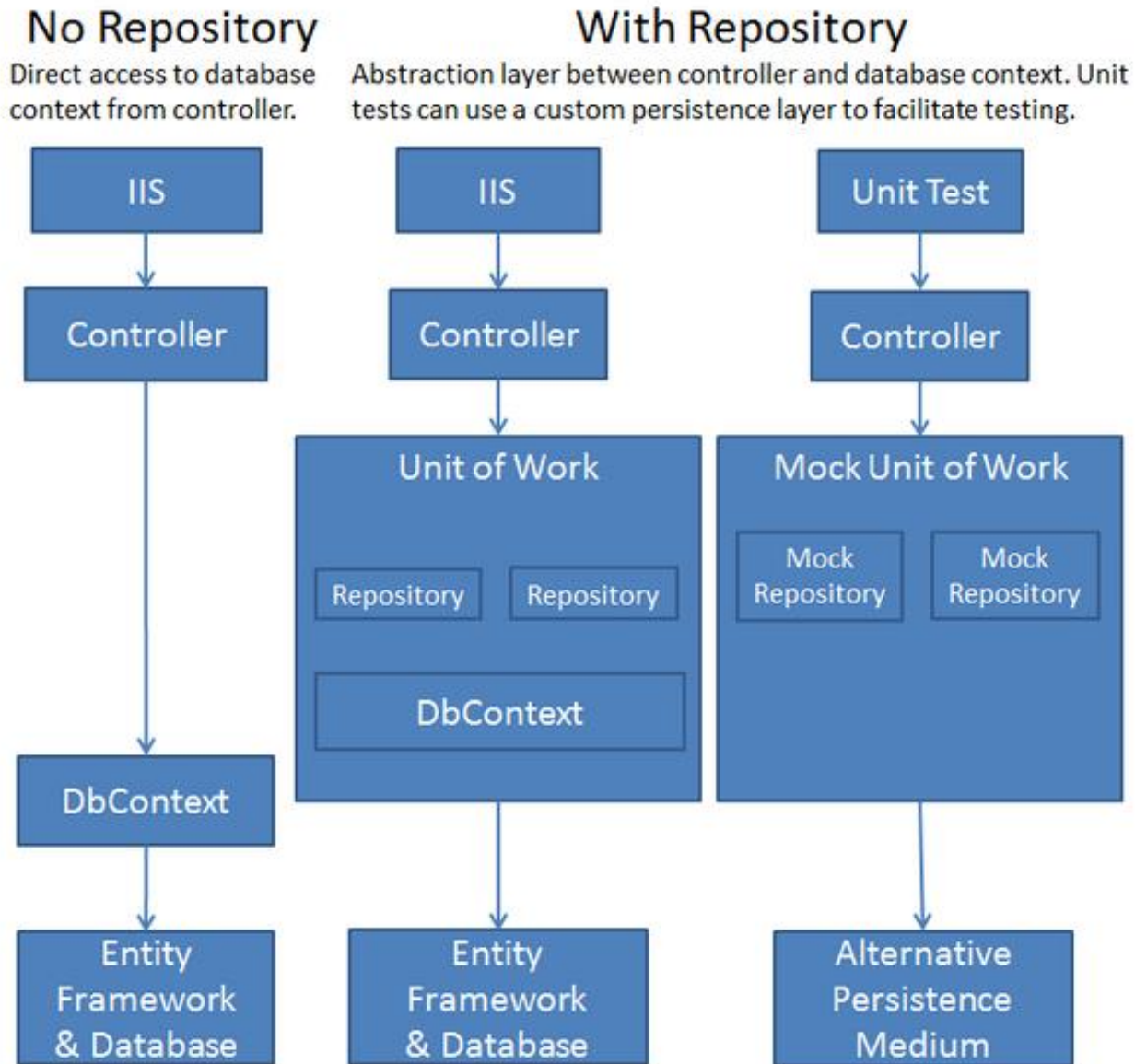
thiagoyama@gmail.com

#08 – DESIGN PATTERNS

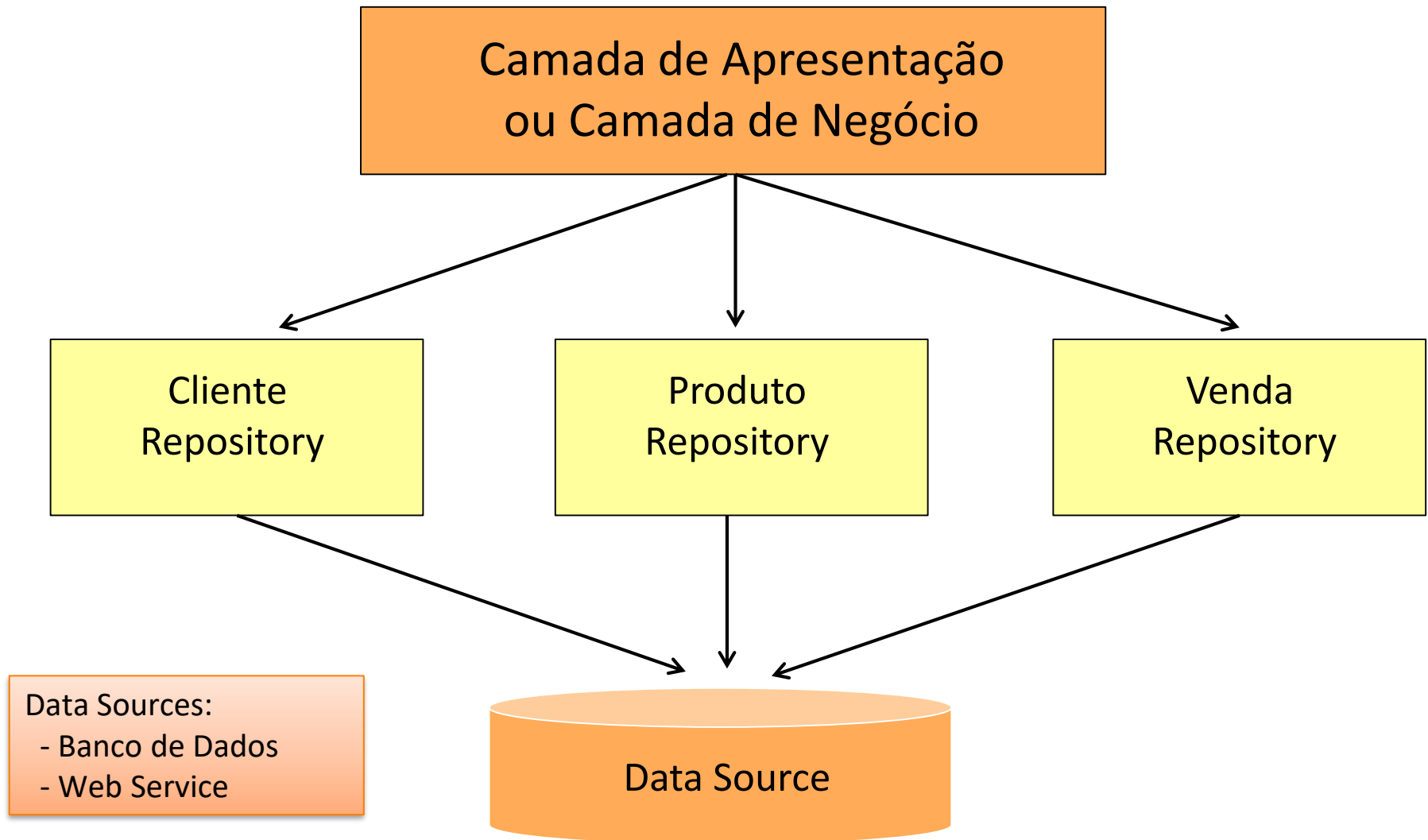
- Repository
- Unit of Work
- Utilizando o Design Patterns
- View Model

REPOSITORY

- Design Pattern para Acesso a Dados: permite realizar o isolamento entre a camada de acesso a dados (DAL) e a camada de apresentação ou camada de negócios.
- Benefícios:
 - Separação de conceitos;
 - Menos erros e códigos duplicados;
 - Facilita os testes automatizados;
 - Facilita a manutenção;



- Encapsula o acesso a dados:



- Primeiramente, precisamos definir uma interface que irá atuar como a nossa fachada de acesso aos dados:

```
public interface IClienteRepository
{
    IEnumerable<Cliente> GetAll();
    Cliente GetById(int clienteId);
    void Insert(Cliente cliente);
    void Delete(int clienteId);
    void Update(Cliente cliente);
    ICollection<Cliente>
        SearchFor(Expression<Func<Cliente, bool>> predicate);
}
```


- Depois, vamos construir a classe que irá implementar as interfaces: `IClienteRepository`.
 - Possui a propriedade `ClienteContext`;
 - Construtor que recebe o context por parâmetro.

```
public class ClienteRepository: IClienteRepository
{
    private ClienteContext _context;

    //Construtor
    public ClienteRepository(ClienteContext context)
    {
        _context = context;
    }
}
```

- Operações básicas: Buscar todos, Buscar por Id e Inserir

```
public IEnumerable<Cliente> GetAll()
{
    return _context.Clientes.ToList();
}

public Cliente GetById(int clienteId)
{
    return _context.Clientes.Find(clienteId);
}

public void Insert(Cliente cliente)
{
    _context.Clientes.Add(cliente);
}
```

REPOSITORY - IMPLEMENTAÇÃO

- Operações básicas: Deletar, Alterar e Salvar.

```
public void Delete(int clienteId)
{
    Cliente cliente = _context.Clientes.Find(clienteId);
    _context.Clientes.Remove(cliente);
}

public void Update(Cliente cliente)
{
    _context.Entry(cliente).State = EntityState.Modified;
}

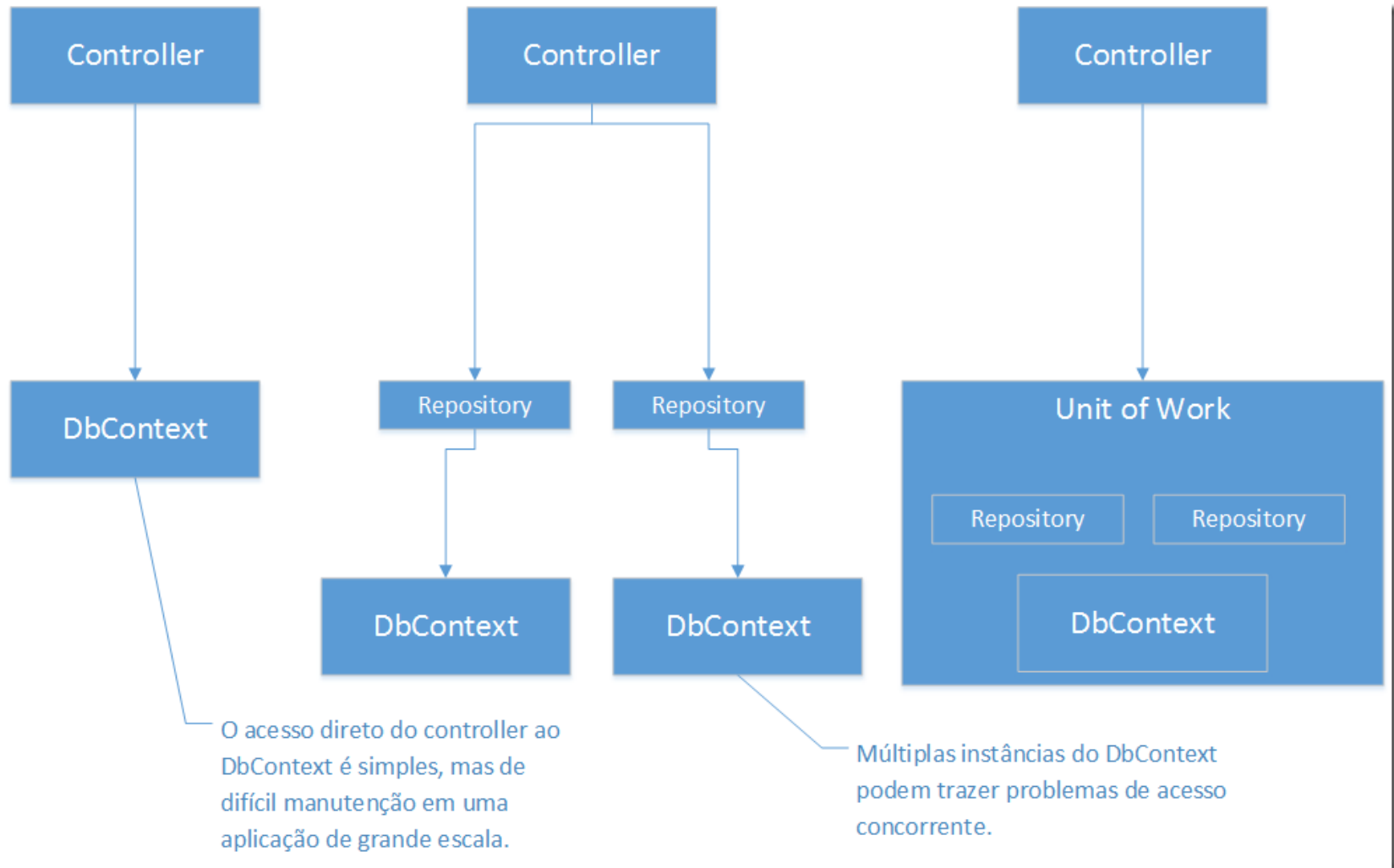
public ICollection<Cliente>
    SearchFor(Expression<Func<Cliente, bool>> predicate)
{
    return _context.Clientes.Where(predicate).ToList();
}
```

UNIT OF WORK

- Padrão Unidade de trabalho: um dos padrões de design mais comuns no desenvolvimento de sistemas;
- Quando existem vários repositórios no projeto, eles devem compartilhar o mesmo contexto de banco de dados.

De acordo com Martin Fowler:

“Mantém uma lista de objetos afetados por uma transação e coordena a escrita de mudanças e trata possíveis problemas de concorrência”



- É uma classe que implementa IDisposable e deve possuir as propriedades:
 - Contexto do banco de dados;
 - Repositories ;

```
public class UnitOfWork: IDisposable
{
    private ClienteContext _context = new ClienteContext();
    private IClienteRepository _clienteRepository;
    //....
}
```

- Devemos implementar somente o método **get** do Repository. Este método deve realizar uma validação, caso não exista uma instância do repository, ela cria um novo, caso contrário, retorna o repository que já existe.

```
public IClienteRepository ClienteRepository
{
    get
    {
        if (clienteRepository == null)
        {
            _clienteRepository = new
            ClienteRepository(context);
        }
        return _clienteRepository;
    }
}
```


- Podemos implementar um método **Save()**, para salvar as alterações no banco de dados:

```
public void Save()  
{  
    _context.SaveChanges();  
}
```

- Dispose é utilizado para liberar os recursos após utilizado;

```
public void Dispose()
{
    if (_contexto != null)
    {
        _context.Dispose();
    }
    GC.SuppressFinalize(this);
}
```

UTILIZAÇÃO DOS DESIGN PATTERNS NO CONTROLLER

- Para utilizar o repository, devemos declarar a unidade de trabalho (unit of work) no controller:

```
public class ClienteController : Controller
{
    UnitOfWork _unit = new UnitOfWork();
    //...
}
```

- Depois utilizamos o unit para acessar o repository e seus métodos:

```
[HttpPost]
public ActionResult Create(Cliente cliente)
{
    _unit.ClienteRepository.Insert(cliente);
    _unit.Save();
    return View();
}

public ActionResult List()
{
    return View(_unit.ClienteRepository.GetAll());
}
```

```
[HttpPost]
public ActionResult Remove(int id)
{
    _unit.ClienteRepository.Delete(id);
    _unit.Save();
    return RedirectToAction("List");
}

[HttpPost]
public ActionResult Edit(Cliente cliente)
{
    _unit.ClienteRepository.Update(cliente);
    _unit.Save();
    return RedirectToAction("List");
}
```

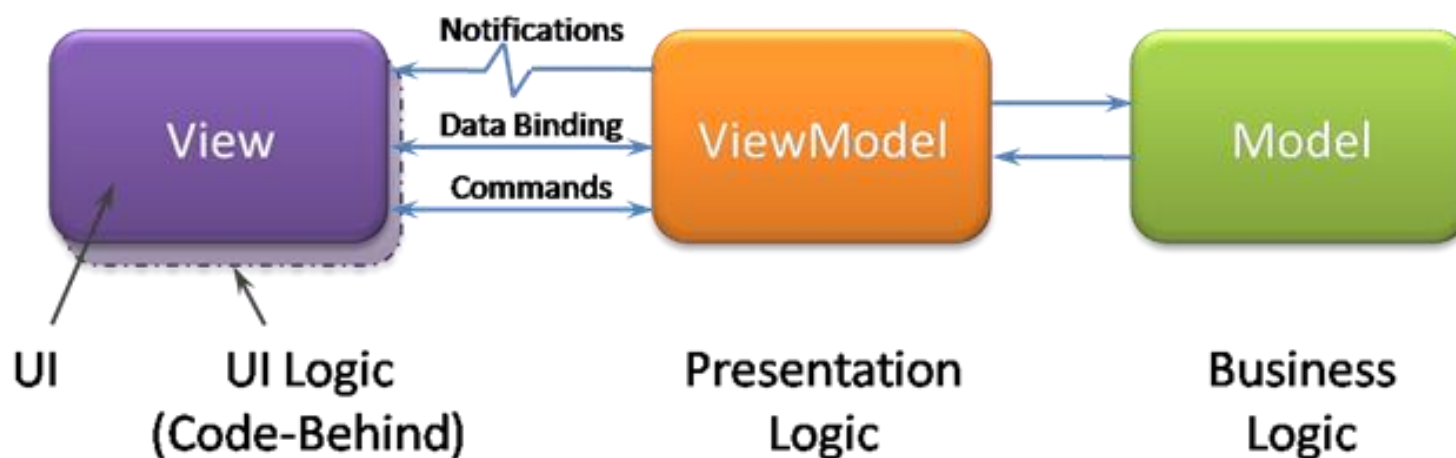
```
[HttpGet]
public ActionResult Buscar(string nome)
{
    var lista =
        _unit.ClienteRepository.SearchFor(c => c.Name == nome);
    return View(lista);
}
```

- Por fim, precisamos liberar o recurso com o método **Dispose()**

```
protected override void Dispose(bool disposing)
{
    _unit.Dispose();
    base.Dispose(disposing);
}
```


VIEW MODEL PATTERN

- Um padrão de Projetos que visa melhorar a organização do código e gestão das informações utilizados na View;
- Permite modelar várias entidades e informações que serão exibidas na View;



Copyright © 2013 - 2017 - Prof. Me. Thiago T. I. Yamamoto

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).

O sucesso nasce do querer, da determinação e persistência em se chegar a um objetivo. Mesmo não atingindo o alvo, quem busca e vence obstáculos, no mínimo fará coisas admiráveis