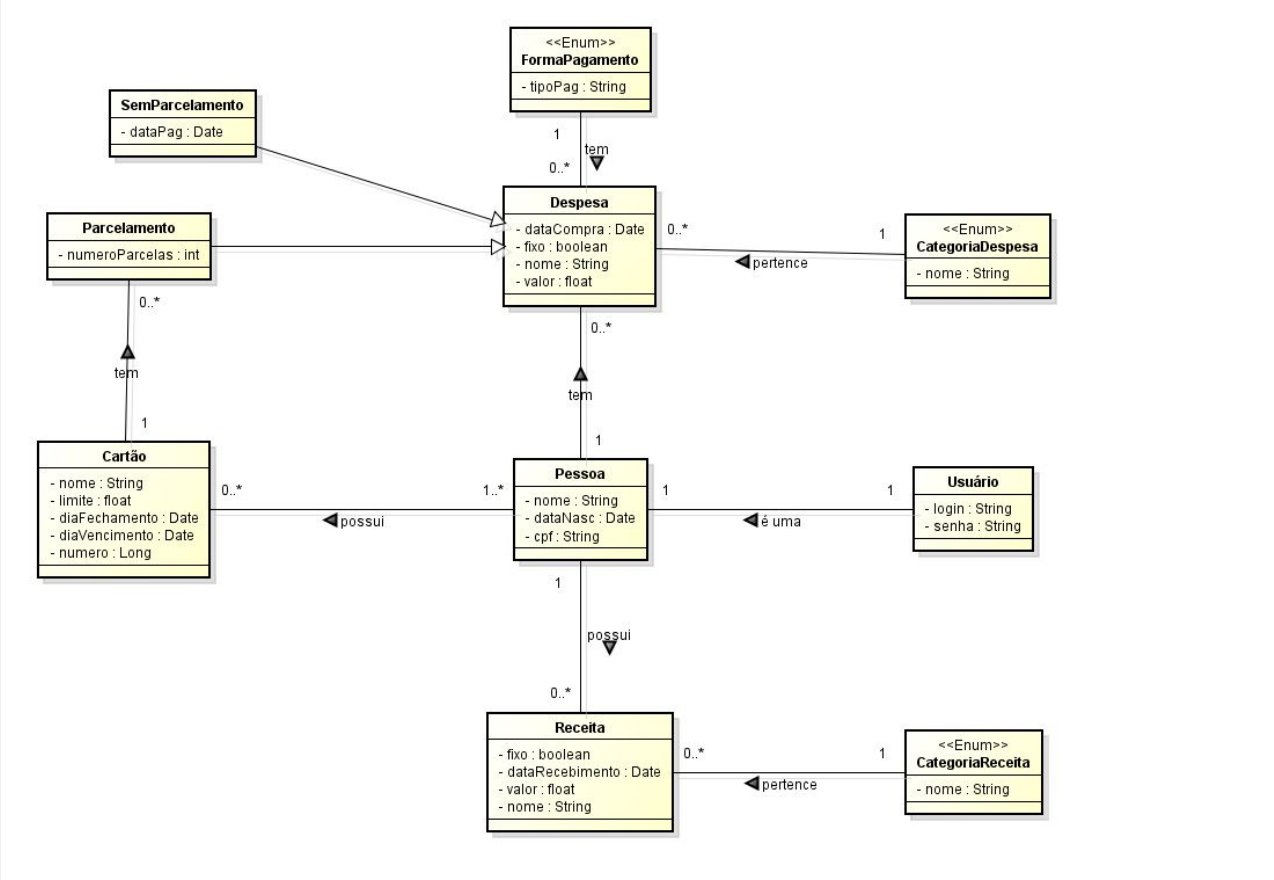
Ctrl+Money

Alunos: Brendon Mauro
Jennifer de Castro
Joel Will
Larissa Motta

Diagrama de Classes



Padrões

- Builder :Padrão Criativo
- Repository => DAO :Padrão Estrutural
- Algumas Frameworks substitui alguns padrões. Como:
  - AutoMapper => Adapter :Padrão Estrutural
- Cadeia de Responsabilidade :Padrão Comportamental
- Template :Padrão Comportamental

Builder

- O Padrão Builder permite a separação da construção de um objeto complexo da sua representação, de forma que o mesmo processo de construção possa criar diferentes representações.

- A classe VisaoGeralViewModel é uma classe complexa para ser instanciada, pois contém um resumo das despesas e receitas, e também cálculos efetuados com as mesmas. O padrão builder foi implementado a fim de abstrair a instanciação desta classe, de organizar e facilitar alterações, além de é claro, deixar o código mais limpo (entenda como legível).

Builder

Classe Builder

```
namespace CtrlMoney.Models
{
    public class VisaoGeralVMBuilder
    {

        private decimal TotalDespesa { get; set; }
        private decimal TotalReceita { get; set; }
        private decimal Caixa { get; set; }
        private Dictionary<string, decimal> CategoriaDespesaValue { get; set; }
        private Dictionary<string, decimal> CategoriaReceitaValue { get; set; }

        public VisaoGeralVMBuilder calcTotalDespesa()
        {
            TotalDespesa = CategoriaDespesaValue.Values.Sum();
            return this;
        }

        public VisaoGeralVMBuilder calcTotalReceita()
        {
            TotalReceita = CategoriaReceitaValue.Values.Sum();
            return this;
        }

        public VisaoGeralVMBuilder calcCaixa()
        {
            Caixa = TotalReceita - TotalDespesa;
            return this;
        }
```

Builder

Classe Builder (continuação)

```
        public VisaoGeralVMBuilder ctrCategoriaDespesa(List<Despesa> despesas)
        {
            CategoriaDespesaValue = new Dictionary<string, decimal>();
            foreach (CategoriaDespesa item in Enum.GetValues(typeof(CategoriaDespesa)))
            {
                decimal valor = 0;
                valor += despesas.Where(p => p is Parcelamento && p.Categoria.Equals(item)).Cast<Parcelamento>().Sum(p => p.Valor / p.NumParcelas);
                valor += despesas.Where(p => p is SemParcelamento && p.Categoria.Equals(item)).Cast<SemParcelamento>().Sum(p => p.Valor);
                CategoriaDespesaValue[item.ToString()] = valor;
            }
            return this;
        }

        public VisaoGeralVMBuilder ctrCategoriaReceita(List<Receita> receitas, ReceitaAPL receitaAPL)
        {
            CategoriaReceitaValue = receitaAPL.GetAllReceitasMes(receitas);
            return this;
        }

        public VisaoGeralViewModel build()
        {
            return new VisaoGeralViewModel(TotalDespesa, TotalReceita, Caixa, CategoriaDespesaValue, CategoriaReceitaValue);
        }
    }
}
```

Builder

Classe Director

```
namespace CtrlMoney.Models
{
    public class VisaoGeralDirector
    {
        private List<Despesa> despesas;
        private List<Receita> receitas;
        private ReceitaAPL receitaAPL;

        public VisaoGeralDirector(List<Despesa> despesas, List<Receita> receitas, ReceitaAPL receitaAPL)
        {
            this.despesas = despesas;
            this.receitas = receitas;
            this.receitaAPL = receitaAPL;
        }

        public VisaoGeralViewModel build()
        {
            return new VisaoGeralVMBuilder()
                .ctrCategoriaDespesa(despesas)
                .ctrCategoriaReceita(receitas, receitaAPL)
                .calcTotalDespesa()
                .calcTotalReceita()
                .calcCaixa()
                .build();
        }
    }
}
```

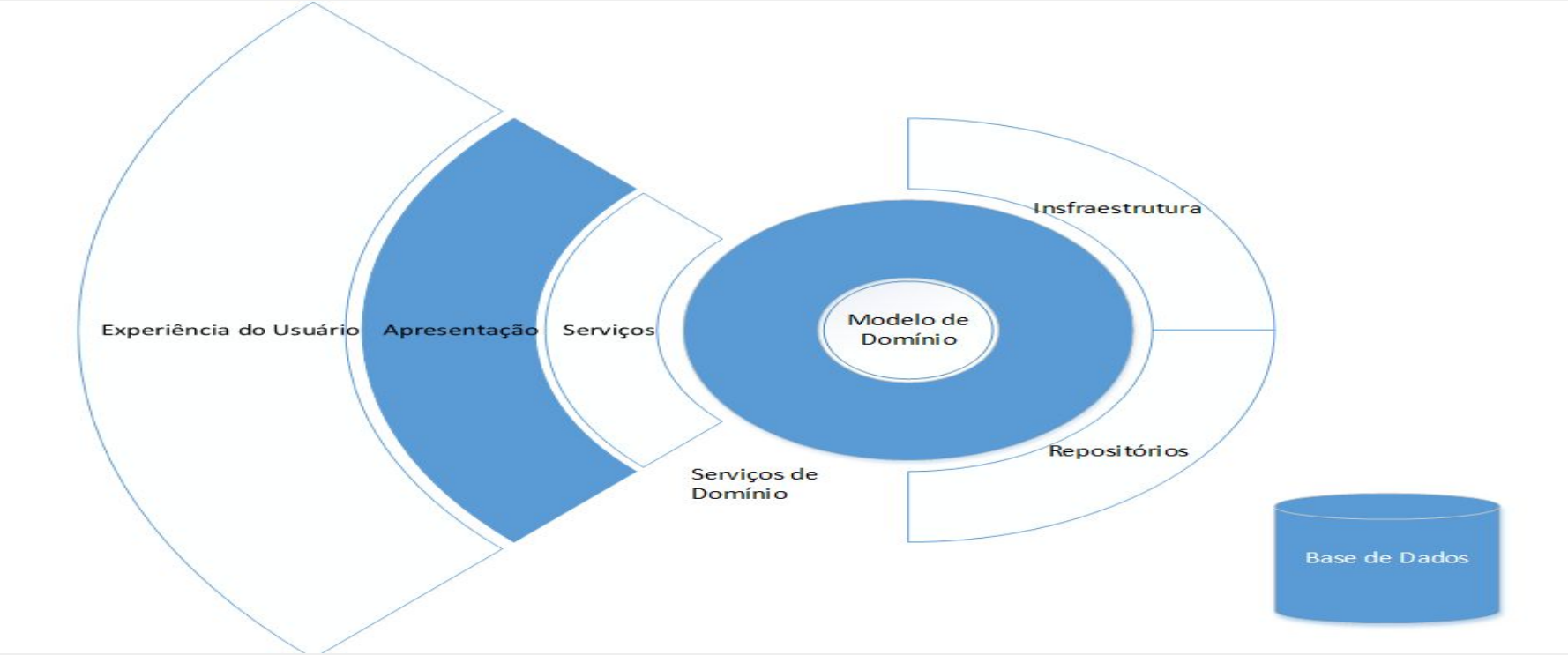
Repository

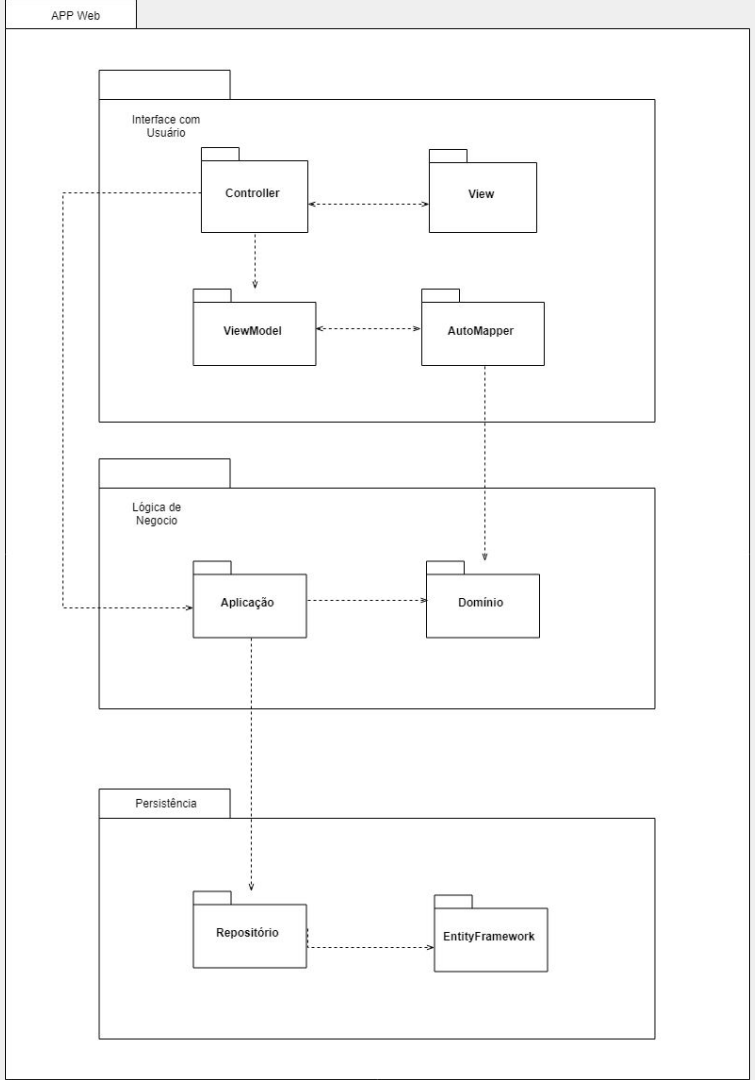
- O padrão Repository permite realizar o isolamento entre a camada de acesso a dados de sua aplicação e sua camada de apresentação e camada de negócios.
- Podendo realizar a persistência e a separação de interesses em seu código de acesso a dados visto que ele encapsula a lógica necessária para persistir os objetos do domínio na fonte de armazenamento de dados.
- O código de exemplo é uma interface genérica, onde contém as principais chamadas ao banco. Caso seja necessário a implementação de uma função específica apenas para uma classe, cria-se uma interface para essa classe que extende a genérica e coloca-se a nova assinatura. Facilitando assim, a mudança do método de acesso a dados sem ter grandes impactos no restante do código.

Interface Genérica do Repositório

```
namespace Repositorio
{
    public interface IRepositorioGenerico<TEntidade, TChave>
        where TEntidade : class
    {
        List<TEntidade> Selecionar();
        TEntidade SelecionarPorId(TChave id);
        void Inserir(TEntidade entidade);
        void Alterar(TEntidade entidade);
        void Excluir(TEntidade entidade);
    }
}
```

Repository


Experiência do Usuário
Apresentação
Serviços
Serviços de Domínio
Modelo de Domínio
Insfraestrutura
Repositórios
Base de Dados


APP Web
Interface com Usuário
Controller
View
ViewModel
AutoMapper
Lógica de Negocio
Aplicação
Domínio
Persistência
Repositório
EntityFramework

Frameworks: AutoMapper - Adapter

- O Padrão adapter adapta uma classe existente para fazer parte do sistema.
- O Padrão arquitetural escolhido foi o MVC e implementamos dois tipos de classes de “transmissão de dados” as viewModels e as de Dominio.
- O Framework Automapper é utilizado para adaptar uma viewModel (classe modelada para conversar com a camada de interface com o usuário) para a Dominio (classe modelada para conversar com a camada de regra de negócios e persistência) e vice e versa.

Frameworks: AutoMapper - Adapter

```
namespace CtrlMoney.AutoMapper
{
    public class DominioToViewModelProfile : Profile
    {
        public DominioToViewModelProfile()
        {
            CreateMap<Pessoa, PessoaUsuarioViewModel>()
                .ForMember(p => p.Login, opt => opt.MapFrom(src => src.Usuario.Login))
                .ForMember(p => p.Senha, opt => opt.MapFrom(src => src.Usuario.Senha));

            CreateMap<Usuario, PessoaUsuarioViewModel>()
                .ForMember(p => p.Nome, opt => opt.MapFrom(src => src.Pessoa.Nome))
                .ForMember(p => p.CPF, opt => opt.MapFrom(src => src.Pessoa.CPF))
                .ForMember(p => p.DataNasc, opt => opt.MapFrom(src => src.Pessoa.DataNasc));
            CreateMap<Cartao, CartaoViewModel>()
                .ForMember(c => c.Nome, opt => opt.MapFrom(src => src.Nome))
                .ForMember(c => c.Limite, opt => opt.MapFrom(src => src.Limite))
                .ForMember(c => c.DiaFechamento, opt => opt.MapFrom(src => src.DiaFechamento))
                .ForMember(c => c.DiaVencimento, opt => opt.MapFrom(src => src.DiaVencimento))
                .ForMember(c => c.Numero, opt => opt.MapFrom(src => src.Numero));


            CreateMap<SemParcelamento, SemParcelamentoViewModel>();
            CreateMap<Parcelamento, ParcelamentoViewModel>();

            CreateMap<Receita, ReceitaViewModel>();
        }
    }
}
```

Frameworks: AutoMapper - Adapter

```
namespace CtrlMoney.AutoMapper
{
    public class ViewModelToDominioProfile : Profile
    {

        public ViewModelToDominioProfile()
        {
            CreateMap<PessoaUsuarioViewModel, Pessoa>();
            CreateMap<PessoaUsuarioViewModel, Usuario>();
            CreateMap<ParcelamentoViewModel, Parcelamento>();
            CreateMap<SemParcelamentoViewModel, SemParcelamento>();
            CreateMap<ReceitaViewModel, Receita>();
            CreateMap<CartaoViewModel, Cartao>();
        }
    }
}
```

Cadeia de Responsabilidade

- O Padrão Cadeia de Responsabilidade é utilizado para evitar acoplamento de “sender” com “receiver” reduzindo o vínculo se tornando mais flexível.

- Foi utilizado para calcular o total de cada categoria das receitas de um mês e colocar em um dicionário. Permitindo que se uma nova categoria de receita seja adicionada futuramente no sistema, a única coisa a ser feita será a criação de uma classe para esta categoria, em que esta classe herda da classe mais abstrata da cadeia e implementa seus métodos abstratos, sem alterar o restante do sistema.

Cadeia de Responsabilidade

Chamada da Função

```
public Dictionary<string,decimal> GetAllReceitasMes(List<Receita> receitas)
{
    AbstractClassCategoriaReceita receitaSalario = new ReceitaSalario();
    AbstractClassCategoriaReceita receitaVendas = new ReceitaVenda();
    AbstractClassCategoriaReceita receitaPensao = new ReceitaPensao();
    AbstractClassCategoriaReceita receitaOutros = new ReceitaOutros();

    receitaSalario.SetNext(receitaVendas);
    receitaVendas.SetNext(receitaPensao);
    receitaPensao.SetNext(receitaOutros);

    var dicionarioReceita = new Dictionary<string, decimal>();
    receitaSalario.EfetuarCalculo(dicionarioReceita, receitas);

    return dicionarioReceita;
}
```

Cadeia de Responsabilidade

Classe Abstrata

```
namespace Dominio
{
    public abstract class AbstractClassCategoriaReceita
    {
        protected AbstractClassCategoriaReceita next;

        public void SetNext(AbstractClassCategoriaReceita forma)
        {
            if (next == null)
            {
                next = forma;
            }
            else
            {
                next.SetNext(forma);
            }
        }

        public void EfetuarCalculo(Dictionary<string,decimal> dicReceitas, List<Receita> receitas)
        {
            dicReceitas = Adicionar(dicReceitas, receitas);
            if (next != null)
            {
                next.EfetuarCalculo(dicReceitas, receitas);
            }
        }

        protected abstract Dictionary<string, decimal> Adicionar(Dictionary<string, decimal> dicionario, List<Receita> receitas);
    }
}
```

Cadeia de Responsabilidade

Classe

```
namespace Dominio
{
    public class ReceitaOutros : AbstractClassCategoriaReceita
    {
        protected CategoriaReceita categoria = CategoriaReceita.Outros;

        protected override Dictionary<string, decimal> Adicionar(Dictionary<string, decimal> dicionario, List<Receita> receitas)
        {
            dicionario[categoria.ToString()] = receitas.Where(r => r.Categoria == categoria).Sum(r => r.Valor);
            return dicionario;
        }
    }
}
```

Template

- O Padrão template define os métodos para as classes concreta implementarem, reutilizando o código sem perder o controle.
- Esse padrão foi usado para definir os métodos que uma classe de configuração de uma entidade (classe domínio) do EntityFramework precisa implementar, mas quem define a ordem da execução dos métodos é a classe pai.

Template

```
namespace EntityAcessoDados.TypeConfig
{
    abstract class GenericConfig<TEntidade> : EntityTypeConfiguration<TEntidade>
        where TEntidade : class
    {
        public GenericConfig()
        {
            ConfigurarNomeTabela();
            ConfigurarCamposTabela();
            ConfigurarChavePrimaria();
            ConfigurarChavesEstrangeiras();
        }

        protected abstract void ConfigurarChavesEstrangeiras();

        protected abstract void ConfigurarChavePrimaria();

        protected abstract void ConfigurarCamposTabela();

        protected abstract void ConfigurarNomeTabela();
    }
}
```

Template

```
namespace EntityAcessoDados.TypeConfig
{
    class CartaoConfig : GenericConfig<Cartao>
    {
        protected override void ConfigurarCamposTabela()
        {
            Property(p => p.Id)
                .IsRequired()
                .HasDatabaseGeneratedOption(System.ComponentModel.DataAnnotations.Schema.DatabaseGeneratedOption.Identity)
                .HasColumnName("id");

            Property(p => p.Numero)
                .IsRequired()
                .HasColumnName("numero");

            Property(p => p.Nome)
                .IsRequired()
                .HasColumnName("nome")
                .HasMaxLength(50);

            Property(p => p.Limite)
                .IsRequired()
                .HasColumnName("limite");

            Property(p => p.DiaFechamento)
                .IsRequired()
                .HasColumnName("dia_fechamento");

            Property(p => p.DiaVencimento)
                .IsRequired()
                .HasColumnName("dia_vencimento");
        }
```

Template

```
        protected override void ConfigurarChavePrimaria()
        {
            HasKey(p => p.Id);
        }

        protected override void ConfigurarChavesEstrangeiras()
        {
            HasMany(p => p.Parcelamentos)
                .WithRequired(p => p.Cartao)
                .HasForeignKey(p => p.CartaoId);
        }

        protected override void ConfigurarNomeTabela()
        {
            ToTable("cartao");
        }
    }
}
```

REFERÊNCIAS

https://www.devmedia.com.br/padroes-comportamentais-no-mvc/28707

https://www.devmedia.com.br/dal-design-patterns-unit-of-work-e-repository/33919

https://pt.stackoverflow.com/questions/12927/qual-a-diferen%C3%A7a-entre-dao-e-repository

http://www.macoratti.net/11/10/net_pr1.htm