WEBSITE EMPRESARIAL COM ARQUITETURA MVC UTILIZANDO DDD E ANGULARJS

BARBOSA, Aurélio Miguel dos Santos; VITOR, Diogo Cesarini; SILVA, Hiago Arthur Frenhan; JÚNIOR, Nivaldo Gonçalves Queiroz; RIBEIRO, Rodrigo Ademir[1].
CARVALHO, Jaqueline Corrêa Silva de; LUZ, Francisco Donizeti Vieira[2].

[1] Acadêmicos do Curso de Ciência da Computação da Universidade José do Rosário Vellano. UNIFENAS, Campus Alfenas – MG, 2016.
[2] Professores do Curso de Ciência da Computação da Universidade José do Rosário Vellano. UNIFENAS, Campus Alfenas – MG, 2016.

Resumo: Projetar um *website* empresarial, sendo realizado com a arquitetura MVC (*Model View Controller*), aliado a metodologia de design DDD (*Domain Driven Design*) para a criação do *back-end*, e o *framework* AngularJS para a interface do usuário (*front-end*). Motivado pela crescente evolução na forma com que estão sendo realizados projetos *web*, onde em sua maioria são estruturados seguindo os conceitos das camadas da arquitetura MVC. A estrutura bibliográfica foi em sua maioria fundamentada com artigos e pesquisas realizadas na internet, no interesse de obter conhecimento sobre tecnologias e arquiteturas que são dominantes no desenvolvimento de *web sites* empresarias no contexto atual. Concluiu-se com as pesquisas relacionadas ao tema, que para o desenvolvimento *web*, a arquitetura MVC sendo bem estruturada é a que obtém melhor desempenho, tanto no quesito do resultado final, quanto para a facilidade de quem está desenvolvendo. O projeto fica mais bem estruturado, facilitando a construção e a manutenção, tanto para quem projetou quanto para outro desenvolvedor que posteriormente tem que realizar alguma manutenção, isso ocorre em razão das camadas dividirem as funções em classes e pastas diferentes.
Palavras chave: MVC, desenvolvimento web, web site empresarial, DDD, angularjs.

Abstract: Designing a business website, being done with the MVC architecture (Model View Controller), combined with DDD design methodology (Domain Driven Design) to create the back-end, and the AngularJS framework for the user interface (front-end). Motivacionado the growing evolution in the way being made web projects, which are mostly structured following the concepts of the MVC architecture layers. The bibliographic structure was mostly based articles and research conducted on the Internet, in the interest of gaining knowledge about technologies and architectures that are dominant in the development of web sites entrepreneurial in the current context. It was concluded with related research, which for the web development, MVC architecture and well structured, is the one that gets better performance, both in the question of the final result, and for ease of who is developing. The project is better structured, facilitating construction and maintenance, both for those who designed, and for another developer who later have to perform some maintenance, this occurs because of the layers divide the functions into different classes and folders.
Keyword: MVC, web development, business web site, DDD, angularjs.

1. Introdução

Segundo Gamma et al. (2010), surgiu na década de 1970 uma arquitetura que foi desenvolvida para ser usada em projetos de interface visual na linguagem de

programação *Smalltalk*, a esta arquitetura recebeu o nome de MVC (*Model*, *View*, *Controller*).

Atualmente, as aplicações *web* contam com um *back-end* e *front-end* fracamente acoplado, facilitando a migração para outra linguagem ou tecnologia.

O desenvolvimento *web* vem ganhando cada vez mais notoriedade, sendo que, atualmente, a maioria das empresas e clientes opta por trabalhar em tal maneira. Com essa grande demanda de profissionais competentes no mercado de trabalho, referente ao desenvolvimento *web*, e sendo ainda mais específico, usando a arquitetura MVC, e tecnologias usadas em associação com a mesma, impulsionaram a realização de uma *web site* empresarial, direcionado a clientes envolvidos com trabalhos na área de manutenção de computadores e suporte, facilitando o contato com o cliente e a solução dos problemas reais por eles encontrados, buscando por fim, experiência necessária para estar mais bem preparado para novos desafios.

Diante do exposto, este estudo teve como objetivo geral reunir a integração de um *back-end* com um *front-end* usando a arquitetura MVC em um web site empresarial. E objetivos específicos manipular *frameworks* no *front-end* para maior interatividade do usuário para com o sistema; e, organizar um *back-end* usando a metodologia de *design* DDD.

As hipóteses norteadoras compreenderam: 1) com a arquitetura MVC desenvolve-se um *website* melhor estruturado que a arquitetura em três camadas; 2) na arquitetura MVC é possível obter melhor divisão de tarefas entre os integrantes da equipe; 3) com uma estrutura fracamente acoplada devido ao *back-end* robusto, há maior facilidade de migração do projeto; 4) com as camadas do *back-end* utilizando DDD (*Domain Driven Design*), facilita o desenvolvimento dentro de uma empresa de grande porte, onde uma *website* é dividida em vários setores; e 5) o desenvolvimento do *front-end* fica mais fácil e interativo usando Angular JS.

2. Referencial Teórico

2.1 Model View Controller (MVC)

O MVC é um padrão de arquitetura de *software*. Com a extensão da complexidade das aplicações desenvolvidas torna-se fundamental a separação entre os dados (*Model*) e o *layout* (*View*). Assim, modificações feitas no *layout* não afetam a manipulação de dados, e estes poderão ser reorganizados sem mudar o *layout*. O MVC soluciona este problema através da separação das tarefas de acesso aos dados e lógica de negócio, lógica de apresentação e de interação com o utilizador, introduzindo um componente entre os dois: o *Controller*. MVC é utilizado em padrões de projeto de *software*, mas o MVC envolve mais da arquitetura de uma aplicação do que é típico para um padrão de projeto (LAMIM, 2012).

O principal objetivo do MVC é separar dados lógicos (*Model*) da interface do usuário (*View*) e as ações da aplicação (*Controller*). A ideia é deixar que uma mensagem do banco de dados seja acessada e visualizada através de várias interfaces. Em MVC, o *Model* não sabe quantas nem quais as interfaces com o usuário estão exibindo seu estado, a *View* não se importa de onde está recebendo os dados, mas ela tem que garantir que sua aparência reflita o estado do modelo, ou seja, sempre que os

estados do modelo mudam, o modelo notifica as *Views* para que as mesmas se atualizem (BAPTISTELLA, 2009).

O *Model* é utilizado para manipular informações de forma mais detalhada, sendo recomendado que, sempre que possível, se utilize dos modelos para realizar consultas, cálculos e todas as regras de negócio do *site* ou sistema. É o modelo que tem acesso a toda e qualquer informação sendo essa vinda de um banco de dados, arquivo XML (BASTOS, 2011).

Segundo Lemos et al. (2013, p. 8) "a *View* é a camada de apresentação da aplicação, o que será visualizado pelo usuário final, não importando quais dados e de qual lugar tenham vindo, mas, sim, de como serão exibidas essas informações".

O *Controller* é responsável por controlar todo o fluxo do programa. Comparando ao ser humano é como se fosse o cérebro e o coração do aplicativo. É nele que se decide "se, o que, quando, onde" e tudo o mais que faz com que a lógica funcione. Desde o que deve ser consultado no banco de dados à tela que vai ser exibida ao usuário (ZEMEL, 2009).

Com o aumento da complexidade dos sistemas/*sites* desenvolvidos atualmente, essa arquitetura tem como foco dividir um grande problema em vários outros menores e de menor complexidade. Sendo assim, qualquer tipo de alteração em uma das camadas não interfere nas demais, propiciando a atualização de *layouts*, alteração nas regras de negócio e adição de novos recursos. Em caso de grandes projetos, o MVC facilita muito a divisão de tarefas entre a equipe (BASTOS, 2011).

De acordo com Pires (2012) as principais vantagens do MVC são maior praticidade em gerenciar o desenvolvimento da aplicação ao dividir o aplicativo em três camadas; não utiliza o estado de exibição nem formulários baseados no servidor, tornando a estrutura MVC ideal para desenvolvedores que desejam controle completo sobre o comportamento da aplicação; utiliza o padrão *Front Controller* que processa as solicitações do aplicativo *Web* através de um único controlador permitindo ao aplicativo suportar a infraestrutura de roteamento; fornece um melhor suporte para desenvolvimento controlado por testes (TDD – *test-driven development*); e funciona bem com aplicativos *Web* que são suportados por grandes equipes de desenvolvedores e com *Web designers* que precisem de um grande grau de controle sobre o comportamento do aplicativo.

Segundo Pires (2012) as principais vantagens do *Web Forms* são suportar diversos tipos de controles e eventos; utilizar o padrão *Page Controller* que adiciona toda a funcionalidade em páginas individuais; manter o estado do formulário no servidor, facilitando o gerenciamento de informações e menor complexidade para o desenvolvimento de aplicativos.

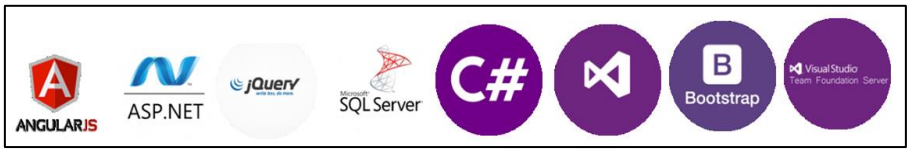
Já Zaccanini (2010) demonstra algumas desvantagens, como o controle é muito difícil referente ao HTML gerado; a arquitetura pode ser tecnicamente fácil, não visa a utilização de padrões arquiteturais, mesmo suportando-os; o último estado da página do servidor é armazenado dentro da página de cliente como um campo oculto, o famoso e conhecido *View state*; dificuldades de compartilhamento e integração com bibliotecas de *JavaScript*; geralmente a lógica de negócio e a apresentação da página ficam no mesmo arquivo.

3. Material e Métodos

O projeto *web* usa a arquitetura MVC (*Model View Controller*) com o *framework Bootstrap* e *AngularJS* como na FIG. 1, para uma melhor estruturação da interface do sistema (*Front-End/ View*) obtendo assim, menor complexidade durante o desenvolvimento da aplicação, e também maior controle sobre o projeto, dividindo a interface, dos dados e funções.

Para o desenvolvimento da regra de negócios e dados ficaram divididos em duas camadas são elas respectivamente a de Infraestrutura e Domínio (*Model*). Obtemos maior interação entre os integrantes do grupo, e maior produtividade de trabalho, utilizando a metodologia *SCRUM*, definindo um *Scrum Master* e prazos para cada fração do trabalho.

Figura 1. Ferramentas utilizadas no desenvolvimento do site.



Fonte: elaboração própria partir de imagens obtidas pelo *google* imagens

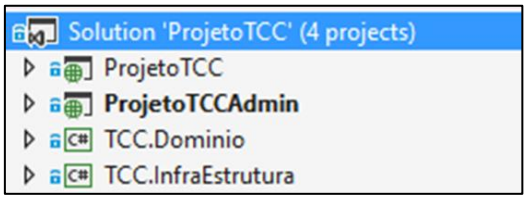
4. Implementação

4.1 DDD (Domain Driven Design)

O projeto prático foi um *Website* que tem a funcionalidade de facilitar o trabalho de uma assistência técnica, tanto para o agendamento de serviços na casa do cliente, quanto para abertura de ordens de serviço, solicitando os serviços que deseja, e marcando assim, somente para a busca do equipamento. Sistema bastante prático para pequenas empresas de assistência técnica em equipamentos eletrônicos, ou até para pessoas físicas que prestam estes serviços.

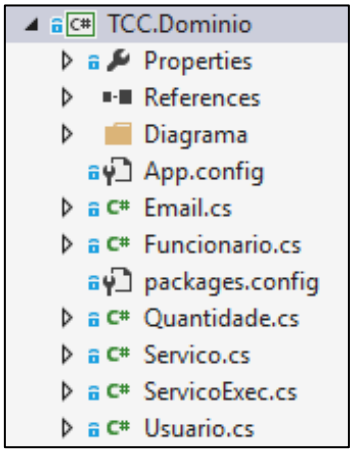
As camadas de domínio são onde foram criadas as classes do projeto, fica a modelagem a ser replicada no banco de dados, em outras palavras é o *Model*. Ele ficou em uma camada independente, pois nesse caso específico tem-se dois projetos que usam o mesmo modelo (Projeto TCC e Projeto TCC Admin), por otimização foi feita esta isolação, ganhando maior agilidade do que se precisassem criar dois modelos, um em cada projeto como nas FIG. 2, 3 e 4.

Figura 2. Camadas do projeto



Fonte: elaboração própria obtida a partir do sistema desenvolvido.

Figura 3. Camada de domínio



Fonte: elaboração própria obtida a partir do sistema desenvolvido.

Figura 4. Classe funcionário, incluída na camada domínio

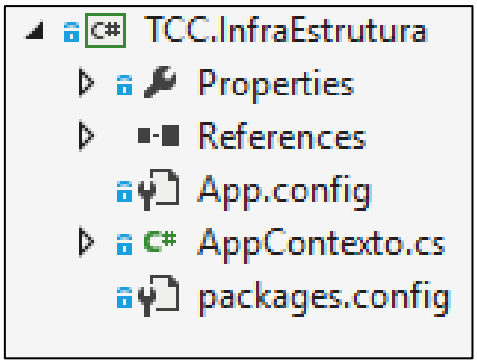
```
public class Funcionario
{
    [Key]
    7 references
    public int Id { get; set; }

    [Required(ErrorMessage = "Nome do técnico Obrigatorio")]
    [MaxLength(150)]
    5 references
    public string Nome { get; set; }

    [Required(ErrorMessage = "Senha Obrigatorio")]
    [MaxLength(200)]
    [MinLength(6)]
    4 references
    public string Senha { get; set; }

    [Required(ErrorMessage = "Email do técnico Obrigatorio")]
    [MaxLength(150)]

    6 references
    public string Email { get; set; }

    2 references
    public string Celular { get; set; }
    0 references
    public string Status { get; set; }

    3 references
    public string Permissao { get; set; }

    0 references
    public List<ServicoExec> ListServicoExec { get; set; }
}
```

Fonte: elaboração própria obtida a partir do sistema desenvolvido.

Na camada de infraestrutura é feita a interação com o banco de dados, onde é criada a classe AppContexto que utiliza os recursos do DbContext, como o DbSet, DbInitializer, entre outros que ajudam na comunicação com os dados, como pesquisa,

inserção e exclusão. É aqui também, que fica a responsabilidade de gerar o banco de dados, utilizando a metodologia Code-First, após a camada Domínio concluída, só chamar a classe AppContexto pelo projeto, que o banco criará automaticamente, caso não exista, como é mostrado nas FIG. 5 e 6.

Figura 5. Camada de infraestrutura



Fonte: elaboração própria obtida a partir do sistema desenvolvido.

Figura 6. Classe APPCONTEXTO

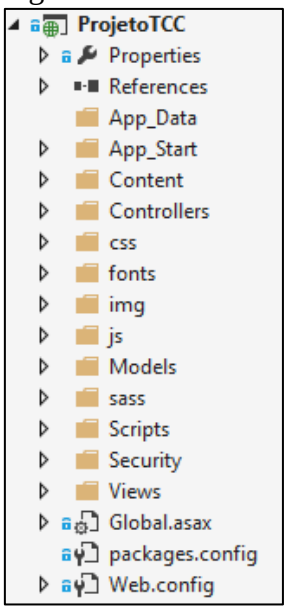
```
public class AppContexto:DbContext
{
    0 references
    static AppContexto()
    {
        // Not initialize database
        //  Database.SetInitializer<ProjectDatabase>(null);
        // Database initialize
        Database.SetInitializer<AppContexto>(new DbInitializer());
        using (AppContexto db = new AppContexto())
            db.Database.Initialize(false);
    }

    15 references
    public DbSet<Usuario> Usuario { get; set; }
    1 reference
    public DbSet<Quantidade> Quantidade { get; set; }
    15 references
    public DbSet<Funcionario> Funcionario { get; set; }
    21 references
    public DbSet<Servico> Servico { get; set; }
    17 references
    public DbSet<ServicoExec> ServicoExec { get; set; }

    1 reference
    class DbInitializer : CreateDatabaseIfNotExists<AppContexto>
    {
        1 reference
        protected override void Seed(AppContexto context)
        {
            base.Seed(context);
        }
    }
```

Fonte: elaboração própria obtida a partir do sistema desenvolvido.

Nesta camada, existe também um recurso que o usuário pode usar chamado Migration. Com ele pode fazer alterações no seu banco de dados, sem precisar recriar o banco. Após atualizar seu domínio para alterar seu banco, só precisa ativar o Migration. Ele é ativado pelo Packet Manager Console, pertencente ao NuGetPackage Manager.

Parte do projeto, como mostra a FIG. 7, é dedicado ao cliente da empresa responsável pelo serviço, aplicação da qual foi criada com a arquitetura MVC, onde o cliente tem a liberdade de solicitar serviços abrindo ordens de serviço e as mantendo em observação. Temos um exemplo de view na FIG. 8.

Figura 7. Estrutura do projeto responsável pela aplicação do cliente.



Fonte: elaboração própria obtida a partir do sistema desenvolvido.

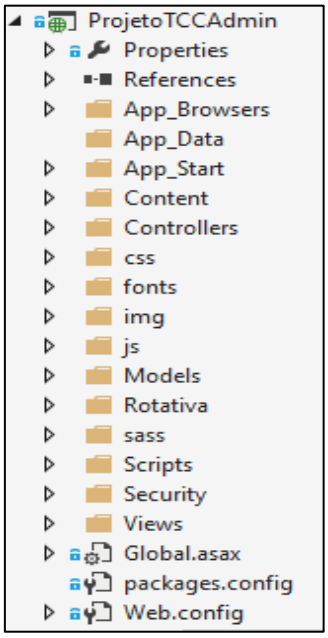
Figura 8. View da página inicial do cliente

```
<div class="container">

    <div class="col-md-12" id="customer-orders">

        <div class="row">
            O estorno de uma O.S.(Ordem de Serviço) só pode ser feito antes de 24h após a abertura de tal.
            Caso queira estornar uma O.S. após este prazo, entre em contato conosco e solucionaremos seu problema!
            <div class="box col-md-12">

                <div class="table-responsive">
                    <table class="table table-hover" align="center">
                        <thead>
                            <tr>
                                <th>OS</th>
                                <th>Data</th>
                                <th>Valor</th>
                                <th>Status</th>
                                <th>Funcionário</th>
                                <th>Ações</th>
                            </tr>
                        </thead>
                        <tbody>
                            @foreach(var v in Lista)
                            {
                                <tr>
                                    <th>@v.OS</th>
                                    <th>@v.Data.ToShortDateString()</th>
                                    <th>R$ @v.Valor</th>
                                    @if(@v.Situacao == "Em Aberto")
```

Fonte: elaboração própria obtida a partir do sistema desenvolvido.

Como na FIG. 9, a aplicação do administrador é onde a empresa solicitante do projeto controla clientes, funcionários e as ordens de serviços abertas pelos mesmos, além de cadastrar novos serviços, mostrado na FIG. 10 um exemplo de controller.

Figura 9. Estrutura do projeto responsável pela aplicação do administrador



Fonte: elaboração própria obtida a partir do sistema desenvolvido.

Figura 10. Exemplo de controller

```
namespace ProjetoTCC.Controllers
{
    0 references
    public class HomeController : Controller
    {
        //
        // GET: /Home/
        static List<Servico> listaservico = new List<Servico>();
        AppContexto contexto = new AppContexto();

        [PermissaoFiltro(Roles = "Adm")]
        0 references
        public ActionResult Index(int? pagina)
        {

            int tamanhoPagina = 10;
            int numeroPagina = pagina ?? 1;
            List<Funcionario> func = contexto.Funcionario.ToList();
            List<ServicoExec> se = new List<ServicoExec>();
            se = contexto.ServicoExec.Include(c => c.UsuarioUnico).ToList();
            ViewBag.Func = func;
            ViewBag.SE = se;
            ViewBag.Lista = (PagedList<ServicoExec>)se.ToPagedList(numeroPagina, tamanhoPagina);
            return View();
        }
```

Fonte: elaboração própria obtida a partir do sistema desenvolvido.

4.2 Entity Framework

Conforme definido pela Microsoft o Entity Framework tem como base o conceito de domínio que, na prática, é o grande detentor da informação, sendo o responsável pela exibição, inclusão, exclusão e atualização da informação dos dados de entidades, seus atributos e relacionamentos (MACORATTI, 2015).

Tecnicamente, o Entity Framework funciona de uma maneira bastante intuitiva. Primeiramente cria-se um modelo entidade-relacionamento que vai definir as entidades contidas dentro do escopo da aplicação. Uma vantagem percebida do momento da criação desse modelo é que ele pode ser gerado a partir de uma base de dados já existente, fato que facilita uma possível migração de um sistema legado. Na versão Entity Framework 6.1 foram incluídos vários recursos relacionados ao Code-First e assistência de conexão a banco de dados. Vieram recursos como o DbSet e DbContext, que são comandos para interação com os dados, e o Migration para quando houver alterações no DBContext ou na arquitetura do banco de dados. Possui também a biblioteca DataAnotations, que permite o uso do Index e AllowHtml, entre vários outros recursos que auxiliam em um melhor desenvolvimento do projeto (ALVES, 2013).

O Entity Framework permite consultar, inserir, atualizar e excluir dados, usando Common Language Runtime (CLR) objetos (conhecidos como entidades). O Entity Framework mapeia as entidades e relacionamentos que são definidas no seu modelo para um banco de dados. O Entity Framework oferece facilidades para fazer o seguinte: materializam dados retornados do banco de dados como objetos de entidade; acompanhar as mudanças que foram feitas para os objetos; lidar com concorrência; propagar alterações objeto de volta para o banco de dados; e vincular objetos para controles (ALVES, 2013).

A classe principal que é responsável pela interação com objetos de dados como é *System.Data.Entity.DbContext* (muitas vezes referida como o contexto). A classe de contexto administra a entidade objetos durante o tempo de execução, o que inclui preencher objetos com dados de um banco de dados, controle de alterações, e persistência de dados no banco de dados (ALVES, 2013).

Por padrão, o contexto gerencia conexões para o banco de dados. O contexto abre e fecha conexões conforme necessário. Por exemplo, o contexto abre uma conexão para executar uma consulta e, em seguida, fecha a conexão quando todos os conjuntos de resultados foram processadas (ALVES, 2013).

Há casos em que precisa quer ter mais controle sobre quando a conexão abre e fecha. Por exemplo, quando se trabalha com o SQL Server Compact, abrindo e fechando a mesma ligação é caro. É possível gerenciar esse processo manualmente usando a conexão de propriedade (ALVES, 2013).

No *Multithreading* o contexto não é thread-safe. Podendo ainda criar um aplicativo com vários segmentos, desde que uma instância da mesma classe de entidade não é controlada por vários contextos ao mesmo tempo (ALVES, 2013).

4.3 Migration

Quando se desenvolve uma nova aplicação, o modelo de dados muda frequentemente, e cada vez que o modelo é alterado, ele fica fora de sincronia com o banco de dados. Quando adicionar, remover ou alterar classes de entidade ou alterar a classe DbContext, a próxima vez que executar o aplicativo automaticamente exclui o

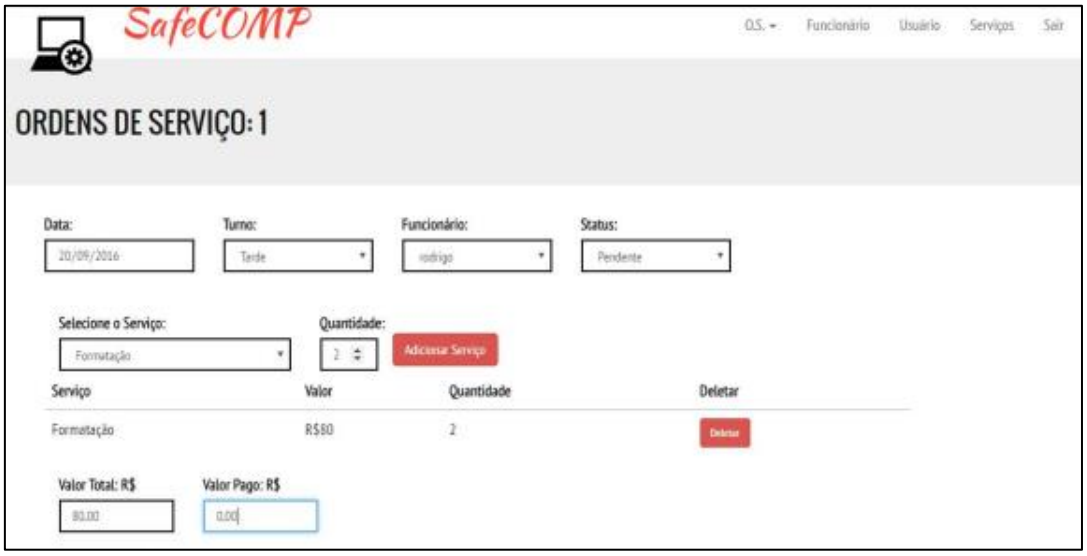
banco de dados existente, cria um novo que corresponde ao modelo, e permanece com as sementes com os dados de teste.

Este método de manter o banco de dados em sincronia com o modelo de dados funciona bem até implantar o aplicativo para produção. Quando o aplicativo está sendo executado na produção geralmente é o armazenamento de dados que se deseja manter, e se não quiser perder tudo cada vez que ocorre uma mudança como a adição de uma nova coluna. O recurso migration resolve este problema, utilizando CodeFirst para atualizar o esquema de banco de dados em vez de deletar e recriar o banco de dados.

4.4 AngularJS

Sempre que for abrir uma ordem de serviço, nesta página terá os serviços que poderá ser adicionado e abaixo uma tabela com os serviços que contém os serviços já adicionados como mostra a FIG. 11. E para adicionar os serviços à tabela sem que precise atualizar a página, usamos o angular, que edita a página sem atualização.

Figura 11. Tela para abrir uma ordem de serviço



Fonte: elaboração própria obtida a partir do sistema desenvolvido.

Segundo a documentação oficial o Controller é o comportamento por trás do DOM e é também responsável por inicializar e/ou adicionar comportamentos ao objeto $scope, o qual permite a comunicação entre a View e o seu Controller. A diretiva ***ng-controller*** é utilizada para definir em qual parte do documento HTML ele será utilizado.

Os módulos são componentes que realizam a inicialização e encapsulam os controllers, diretivas, services e routes de uma aplicação AngularJS e são definidas pela diretiva ng-app.

De acordo com a documentação oficial, Services são objetos Singletons que realizam tarefas específicas comuns em uma aplicação WEB. Esses componentes podem ser utilizados para compartilhar informações entre controllers, servir como

comunicação com o servidor e também como a camada que contém a lógica de negócio, deixando o Controller cuidar somente do fluxo da aplicação.

As diretivas são elementos do framework que trazem ‘um novo poder ao HTML. Elas são definidas dentro do documento HTML e são utilizadas para que a manipulação do DOM seja feita de forma mais transparente e/ou para adicionar novos comportamentos às tags existentes.

Os Filters é outro recurso muito interessante e nativo do framework é a aplicação de filters. Os filtros servem para transformar o resultado de uma expressão, aplicando algum tipo de formatação ou restrição nos dados.

4.5 Bootstrap

Para criar um layout bonito e responsivo, foi usado os recursos do Bootstrap, criando em cada projeto uma Master Page. E essa usada em todo o respectivo projeto, sem ser necessário preocupar-se em recriar layouts para todas as páginas (Views).

No projeto é adicionado css com alterações específicas para o uso do bootstrap desejados, o mesmo ocorre com as classes js. Após adicioná-los no projeto, é necessário referenciá-los dentro da página a ser usada, no caso do projeto, todas foram referenciadas dentro da MasterPage.

5. Resultados e Discussão

5.1 Resultados

Conforme mostra a FIG. 12, onde traz a opinião de dois grandes sites envolvendo desenvolvimento web, o primeiro framework acompanhado de uma metodologia no ranking, é o ASP.NET MVC, que a cada dia ganha mais espaço na área de desenvolvimento web. Na lista também aparece um único framework de front-end que é o AngularJS, que vem substituindo todos outros.

Figura 12. Ranking de frameworks segundo os principais sites de pesquisa sobre computação

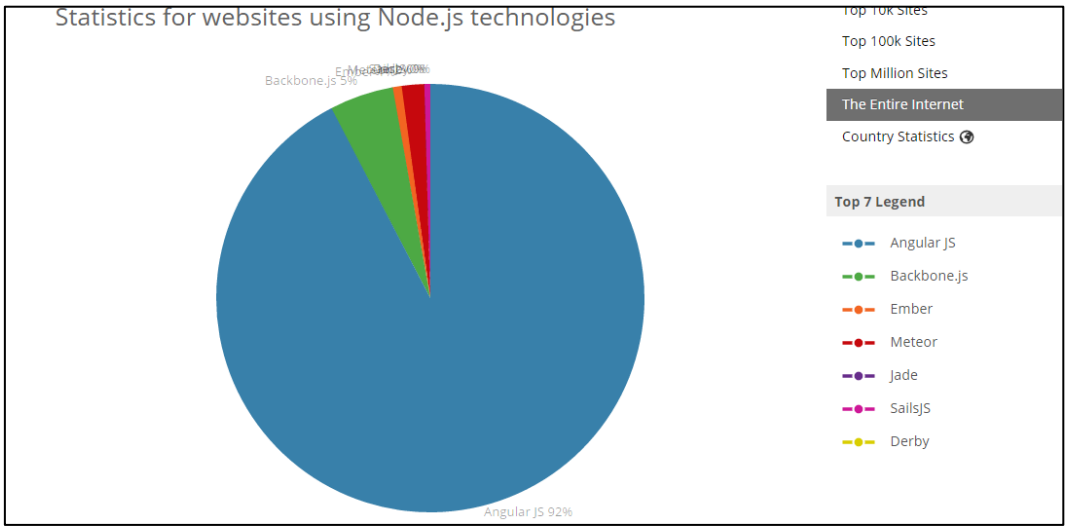
Rankings

Framework	Github Score	Stack Overflow Score	Overall Score
ASP.NET		100	100
AngularJS	100	96	98
Ruby on Rails	95	98	96
ASP.NET MVC		94	94
Django	90	93	91

Fonte: http://hotframeworks.com/

Como mostra a FIG. 13, o AngularJS é a tecnologia com maior preferência (92%) entre os sites que utilizam do Node.js, e com grande vantagem em cima dos concorrentes.

Figura 13. Estatística segundo sites que usam tecnologia NODE.JS



Fonte: http://trends.builtwith.com/framework/node.js

5.2 Discussão

Os resultados encontrados no presente estudo sugerem que a arquitetura MVC, vem atraindo cada vez mais os desenvolvedores web, mas ainda está abaixo do ASP.Net sem uma arquitetura estabelecida, por ter menos normas e obter maior liberdade, mas como tal liberdade tem grande risco de comprometer o projeto, a arquitetura MVC já é requisitada em boa parte das empresas. Indicam que sites que utilizam das novas tecnologias, já querem também que o front-end de seu site, tenha desempenhos mais modernos, e o que mais pode propor isso, é o AngularJS, com alteração na página e interação com o usuário em tempo real.

6. Conclusão

Concluiu-se de acordo com tudo visto no andamento do projeto que a utilização da arquitetura MVC facilita o relacionamento e divisão de tarefas entre as camadas do back-end e front-end. O back-end utilizando- se da metodologia de design DDD, contribui para a criação da modelagem de dados de forma simples e concisa. O front-end por sua vez, utilizando asp.net MVC e Angular.js, trouxeram grande interatividade e funcionalidade para o projeto.

Por todos esses aspectos percebemos que as tecnologias usadas são provenientes de boas funcionalidades e que a união das mesmas, trouxeram um projeto modular,

simples e interativo. Ainda contribuíram para um grande enriquecimento de conteúdo entre os integrantes.

REFERÊNCIAS

ALVES, Tiago Bento. **Entity Framework code-first.** 2013. Disponível em: <http://www.devmedia.com.br/entity-framework-code-first/29705> Acesso em: 12/ Julho/ 2016.

BAPTISTELLA, Adriano José. **Abordando a arquitetura MVC, e Design Patterns: Observer, Composite, Strategy.** 2009.Disponível em: <http://www.webartigos.com/artigos/abordando-a-arquitetura-mvc-e-design-patterns-observer-composite-strategy/20878/>Acesso em: 5/ Fevereiro/ 2016.

BASTOS, Daniel Flores. **O que é Model-View-Controller (MVC)?.** 2011. Disponível em: <http://blog.thiagobelem.net/o-que-e-e-como-funciona-o-jquery> Acesso em: 5/ Fevereiro/ 2016.

GAMMA, Erich et al. **Padrões de Projeto:** Soluções reutilizáveis de software Orientado a Objetos. Porto Alegre: Bookman, 2000.

LAMIN, Jonathan. **MVC – O padrão de arquitetura de software.** 2012. Disponível em: <https://www.oficinadanet.com.br/artigo/1687/mvc_-_o_padrao_de_arquitetura_de_software> Acesso em: 16/ Julho/ 2016.

LEMOS, Maximilian Ferreira de et al. **Aplicabilidade da arquitetura MVC em uma aplicação web (WebApps).** RE3C-Revista Eletrônica Científica de Ciência da Computação, v.8, n.1, p.1-17, nov.2013.

MACORATTI, José Carlos. **Entity Framework – apresentação e arquitetura.** 2015. Disponível em: <http://imasters.com.br/framework/entity-framework-apresentacao-e-arquitetura/?trace=1519021197&source=single> Acesso em: 14/ Fevereiro/ 2016.

PIRES, Eduardo. **Desenvolvimento Web com .Net – MVC x WebForms.** 2012. Disponível em: < http://www.eduardopires.net.br/2012/07/desenvolvimento-web-mvc-x-webforms/>Acesso em: 17/ Julho/ 2016.

ZACCANINI, Rafael.**O que é e por que utilizar o ASP.NET MVC?.**2010. Disponível em: <http://www.devmedia.com.br/o-que-e-e-por-que-utilizar-o-asp-net-mvc/18544>Acesso em: 17/ Julho/ 2016.

ZEMEL, Tárcio. **MVC (Model – View – Controller).**2009. Disponível em: <http://codeigniterbrasil.com/passos-iniciais/mvc-model-view-controller/> Acesso em: 20/ Fevereiro/ 2016.