**Sumário**

[1 Sobre a Iniciativa 2](#_Toc61874330)

[2 Objetivos 2](#_Toc61874331)

[3 Objetivos do requisito 2](#_Toc61874332)

[4 Arquitetura 2](#_Toc61874333)

[5 ESPECIFICAÇÃO DA INFRAESTRUTURA 13](#_Toc61874334)

[6 GERÊNCIA DE CONFIGURAÇÃO 14](#_Toc61874335)

[7 Referências 15](#_Toc61874336)

[8 Histórico de alterações 15](#_Toc61874337)

# Sobre a Iniciativa

|  |  |
| --- | --- |
| **Nome** | CARTAODIGITAL – Digitalizar o processo de entrega de benefícios de final de ano. |
| **Cliente** | Martins Karina (HSS-Ct) |
| **Contato** | Juliana Rodrigues (HSS-Ct) |

# Objetivos

Criação de um sistema a fim de digitalizar todo processo de entrega de benefícios aos colaboradores, objetivando a economia de papel e economia do tempo gasto do setor responsável para preenchimento manual, separação e entrega de vouchers.

# Objetivos do requisito

Definição dos itens de arquitetura do software.

Como repositório de dados, a aplicação utilizará o Microsoft SQL Server 2016.

# Arquitetura

Para o desenvolvimento da aplicação, deverá ser criado uma infraestrutura web que será disponibilizada como serviço.

A Infra e o ciclo de desenvolvimento se dará da seguinte forma:

O módulo de serviço será desenvolvido utilizando o Web API.

**Arquitetura Web API:**

A arquitetura Web API funcionará como serviço SOA. Esse serviço fará a troca de mensagens entre os diversos clientes que a utilizaram através de handlers.

Basicamente esses handlers recebem a instância de uma classe do tipo HttpRequestMessage, que traz toda a solicitação do usuário, e retornam a instância da classe HttpResponseMessage, contendo a resposta gerada para aquela solicitação. E como já ficou subentendido, podemos ter vários handlers adicionados ao pipeline, onde cada um deles pode ser responsável por executar uma tarefa distinta, como logging, autenticação, autorização, etc. A imagem abaixo ilustra esse fluxo.

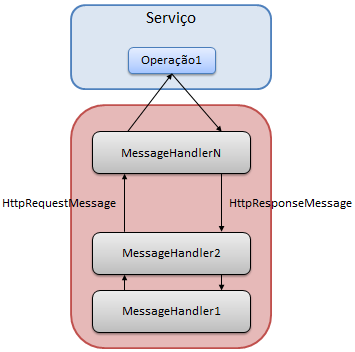


Figura 1 - Estrutura dos message handlers

**Entendendo como funciona o serviço:**

O cliente fará o envio de mensagens para a API, e receberá as respostas para cada requisição.

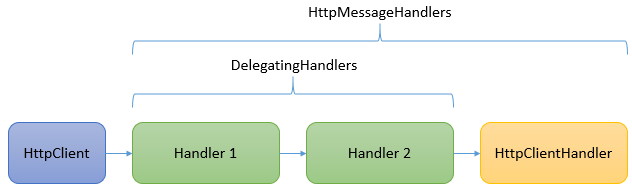


Figura 2 - Estrutura dos message handelrs do lado do cliente.

O serviço implementará o padrão de iniciativa MVC (Model View Controller), o padrão será utilizado para orquestrar a troca de mensagens entre aplicações e Serviço (Front end e Back And).

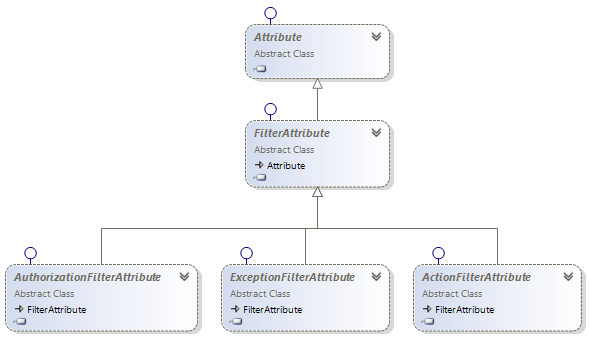


Figura 3 - Hierarquia das classes dos filtros existentes dentro do framework

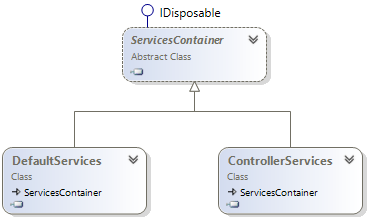


Figura 4 - Classes para a customização das configurações do serviço.

**Front End (Client)**

O cliente será desenvolvido utilizando AngularJS.

Para detalhes da Arquitetura, visualizar a apresentação abaixo:

Apresentação Angular JS <https://inside-docupedia.bosch.com/confluence/display/websec/Released+Web+Development+Guidelines>

O cliente foi escolhido, por ser responsivo e atender os requisitos proposto no escopo dessa iniciativa, onde o mesmo deverá rodar em Celulares, Tablets e Desktop, sem custo para desenvolvimento de aplicativos específicos.

Para o desenvolvimento, estamos utilizando o Node.JS (Criação de Infra e Servidor Web), Bower (Controle de Pacotes e biblioteca javascript) e por último o Grunt (Ferramenta para entrega contínua).

O front End, implementa o padrão de mercado MVVM (Model View View Model).

Esse padrão foi escolhido, devido a sua facilitação em trabalhar com APIs. Mantendo o código extensível.

**Web API Arquitetura**

A Arquitetura é escalável e utiliza diversos padrões de mercado, como MVC, Repository.

A Arquitetura Geral da API está segmentada da seguinte forma:

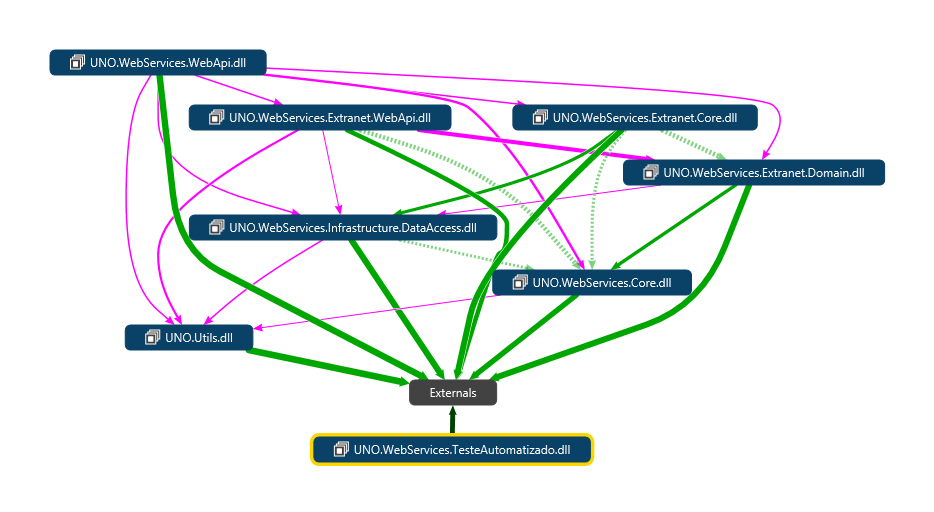


Figura 5 – Arquitetura – Aplicação – Web API - Geral

Vamos analisar os Itens:

1. UNO.WebServices.TesteAutomatizado.dll

Responsável pelos Testes automatizados da Aplicação

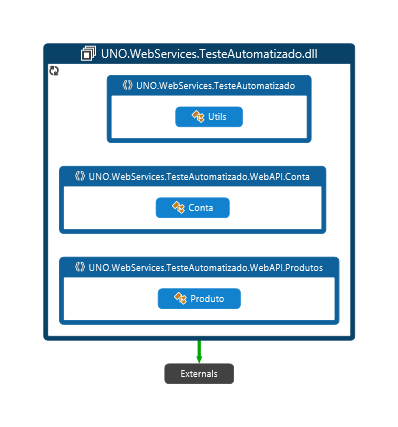


Figura 6 – Arquitetura – Aplicação – Web API – Testes Automatizados

1. UNO.WebServices.WebApi.dll

Aplicação Web, onde ficam hospedados os Serviços.

Implementa o padrão de mercado MVC (Model View Controller).

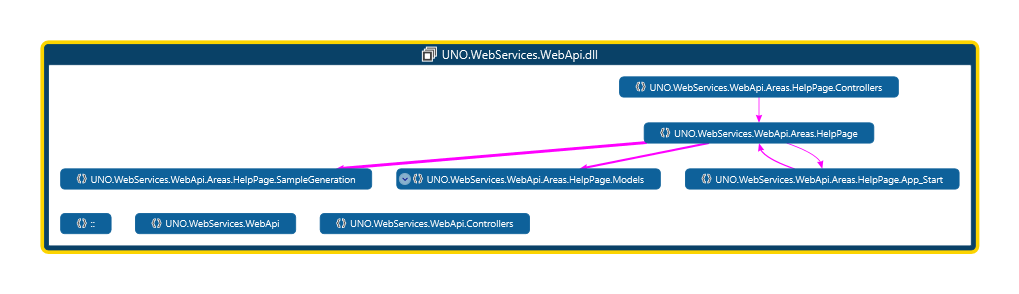


Figura 7 – Arquitetura – Aplicação – Web API - MVC

1. UNO.WebServices.Infrastructure.DataAccess.dll

Contém a extensão da ferramenta de ORM EntityFramework utilizada para simplificar o acesso à base de dados.

O ORM foi escolhido devido sua facilidade em manusear as informações na base de dados.

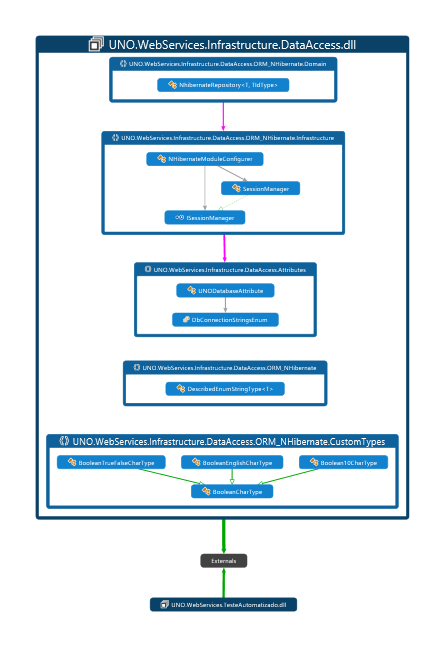


Figura 8 – Arquitetura – Aplicação – Web API - ORM

1. UNO.WebServices.Core.dll

O Core da API possui as classes necessárias para reutilização e extensão dos serviços da API.

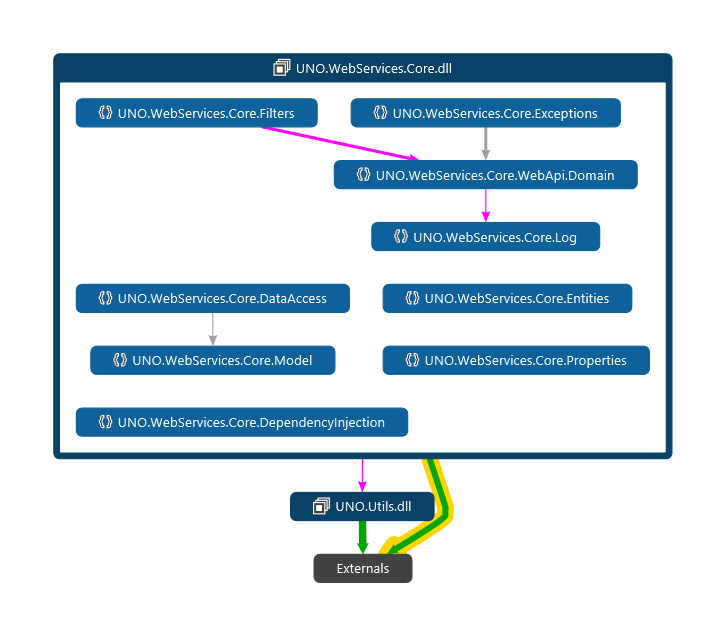


Figura 9 – Arquitetura – Aplicação – Web API - Core

1. UNO.Utils.dll

O Utils da API possui os serviços de suporte à operação da aplicação API.

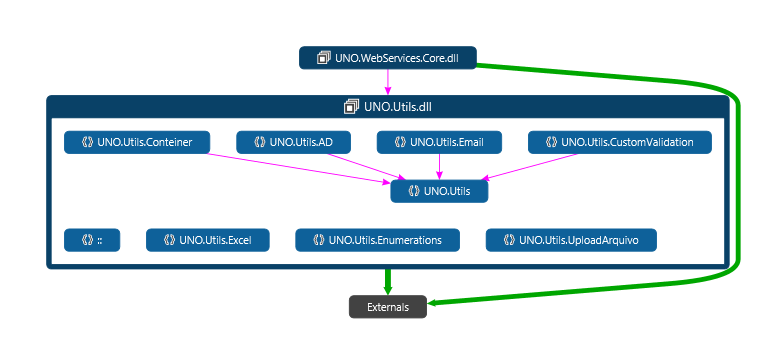


Figura 10 – Arquitetura – Aplicação – Web API - Utils

1. UNO.WebServices.Extranet.WebApi.dll

Chegamos nos métodos de Serviço principal.

Nessa iniciativa é que implementaremos toda a camada de negócio da aplicação.

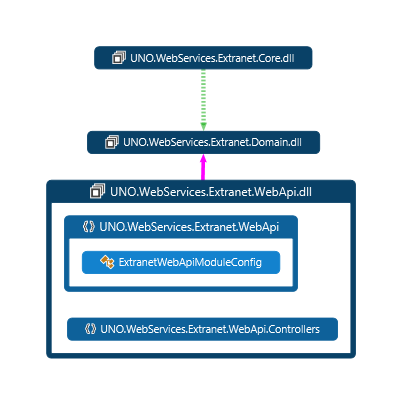


Figura 11 – Arquitetura – Aplicação – Web API – Extranet - Controle

1. UNO.WebServices.Extranet.Domain.dll

Nessa iniciativa é que implementaremos os itens referentes ao domínio dos serviços da aplicação.

Podemos ver abaixo, os Models e as interfaces de serviços a serem implementos pelo Core.

O Model reflete nossa entidade (Tabela DB) mapeada pelo ORM com todos os seus relacionamentos.

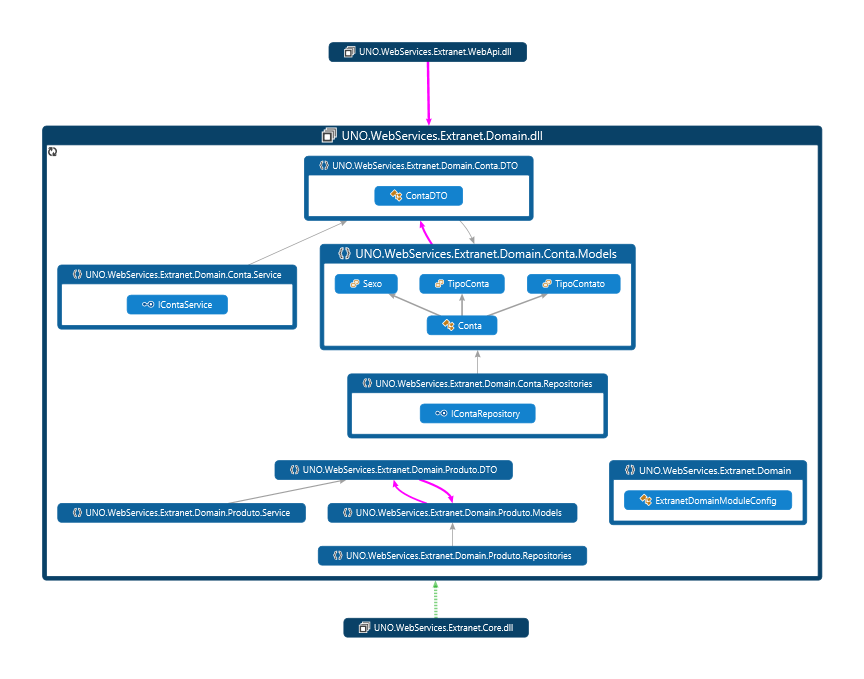


Figura 12 – Arquitetura – Aplicação – Web API – Extranet - Domínio

1. UNO.WebServices.Extranet.Core.dll

Chegamos nos métodos de Serviço Core.

Nessa iniciativa é que implementaremos os métodos contidos nas interfaces criadas na iniciativa domínio.

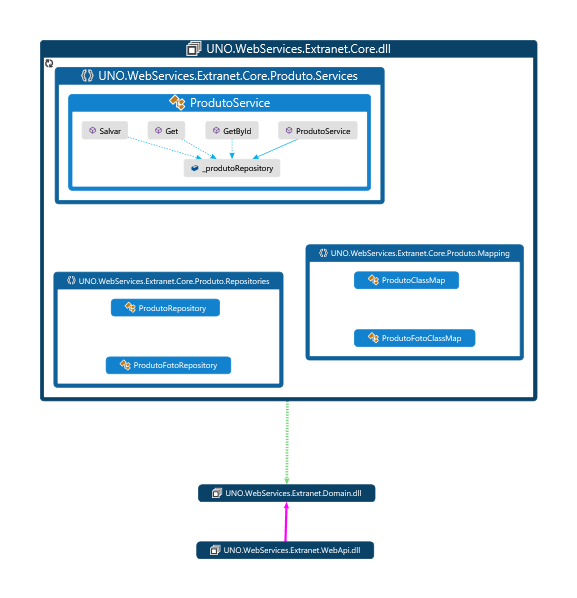


Figura 13 – Arquitetura – Aplicação – Web API – Extranet – Core

# ESPECIFICAÇÃO DA INFRAESTRUTURA

* Usuários

Esta seção lista os requisitos que estão relacionados com a quantidade de usuários concorrentes/total do sistema, bem como a infraestrutura utilizada para seu acesso.

|  |  |  |
| --- | --- | --- |
| Tipo de usuário | Descrição | Valor estimado |
| Rede interna | Usuários que utilizarão o sistema na rede interna. | ~20 |

* Períodos de utilização intensa

Esta seção lista de horários prováveis de utilização do sistema pelos usuários. O objetivo é determinar os dias e horário de utilização mais intensa do sistema.

- Turno administrativo: 08:00am – 17:00pm

* Armazenamento

Esta seção lista os requisitos de armazenamento de dados do sistema.

|  |  |  |  |
| --- | --- | --- | --- |
| Requisito | Objetivo | Indicador | Valor estimado |
| Carga inicial | Tamanho inicial da base de dados para utilização do sistema | Espaço estimado em MB | 20 |
| Crescimento mensal | Crescimento da base de dados mensal | Crescimento estimado em MB | 50 |
| Tempo de armazenamento | Tempo em que as informações devem estar disponíveis | Quantidade de anos | Minimo de 1 ano |

# GERÊNCIA DE CONFIGURAÇÃO

A gerência de configuração da aplicação será constituída da seguinte forma:

A Infraestrutura será gerenciada pelo Visual Studio Team Foundation Server.

A aplicação está organizada em **Branchs** conforme estrutura abaixo:

* **Branch: Production** 
  + Versão de Produção – Versão Estável do Produto

Caso tenhamos mais times trabalhando em releases diferentes, teremos uma segunda branch de desenvolvimento.

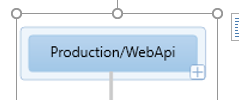


Figura 14 – Gerencia de Configuração – Manutenção

# Referências

# Histórico de alterações

|  |  |  |  |
| --- | --- | --- | --- |
| **Issue** | **Date** | **Editor** | **Description of amendment** |
| 1.0 | 06/12/2021 | Luis Teodoro | Preenchimento inicial |