Projeto WTE 2.0

Requisitos não-funcionais e padrões de desenvolvimento

É de fundamental importância para o bom desenvolvimento de software entender a arquitetura do sistema, comportamento e padronização do mesmo.  
  
A predefinição das funcionalidades que nada tem a ver com o objetivo do sistema guia os desenvolvedores para um software mais consistente, de fácil manutenção e segurança com os dados.

May 12, 2016

F3M Soluções em Tecnologia

# Sumário

[Sumário 1](#_Toc453588560)

[Histórico de Revisões 2](#_Toc453588561)

[Introdução 2](#_Toc453588562)

[Propósito 2](#_Toc453588563)

[Escopo 2](#_Toc453588564)

[Definições, Acrônimos e Abreviações 2](#_Toc453588565)

[Referências 2](#_Toc453588566)

[Comunicação 3](#_Toc453588567)

[Histórico 3](#_Toc453588568)

[Fluxo de Requisição 3](#_Toc453588569)

[Definições de Camadas 3](#_Toc453588570)

[API 3](#_Toc453588571)

[Núcleo 3](#_Toc453588572)

[Domínio 4](#_Toc453588573)

[Interface de Núcleo/Domínio 4](#_Toc453588574)

[Padronização de Código 5](#_Toc453588575)

[Módulos 5](#_Toc453588576)

[Projetos 5](#_Toc453588577)

[Interfaces 5](#_Toc453588578)

[Classes 5](#_Toc453588579)

[Propriedades 5](#_Toc453588580)

[Instâncias ou declarações de variáveis 5](#_Toc453588581)

[Variáveis 6](#_Toc453588582)

[Contexto de Banco de Dados 6](#_Toc453588583)

# Histórico de Revisões

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Versão | Descrição | Autor |
| 12/05/2016 | 1.0.0 | Versão inicial | Renato Moraes |

# Introdução

## Propósito

Criar regras para que o desenvolvimento da aplicação WTE seja realizado com maior segurança e padronização. Serão definidos formatos de arquivos, classes de html, segurança, fluxo de negócio dentre outras informações importantes

## Escopo

Neste documento serão apresentados os Requisitos não-funcionais para o desenvolvimento da aplicação WTE

## Definições, Acrônimos e Abreviações

|  |  |
| --- | --- |
|  |  |

## Referências

* --

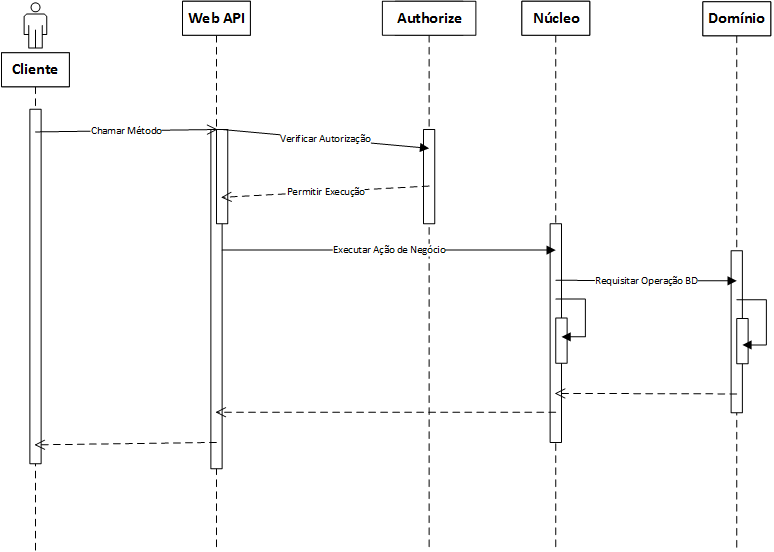
## 

# Comunicação

## Histórico

Por conta da vasta interferência externa no Sistema WTE e na possibilidade de adesão de outras aplicações e várias plataformas (mobile, desktop) é imprescindível que os processos sejam disponibilizados de forma flexível e compatível a todos os meios. Por este motivo a comunicação entre a interface gráfica e o negócio será por meio de API.

## Fluxo de Requisição



## Definições de Camadas

### API

Projeto Web API deverá ser a porta de entrada para o negócio. Este serviço deve ser entendido como controladora da interface o que não permitiria que o mesmo fosse utilizado como classe negócio.

### Núcleo

Também conhecida como negócio, as classes núcleo concentram as regras de negócio, devem ser bem dividas em suas ações, cada micro ação deve ter seu método interno tornando o código mais granulado e fácil de estender ações a outras regras de negócio que necessitarem.

Deve herdar de uma interface[INFachada] e não deverá ser acessada, nem mesmo pela API, sem que seja obtida pela classe “Fabrica”, sendo assim ela deve ter sua privacidade definida como “internal” podendo ser acessada diretamente apenas no mesmo projeto.

### Domínio

As classes de domínio devem definir padrões para executar ações no banco de dados, nenhuma ação de banco de dados (select, insert, update, delete, exec...) deve ser executadas fora das classes de domínio. O objetivo é centralizar estas execuções, dando assim mais especialidade e especificidade a cada camada.

Deve herdar de uma interface[IDFachada], acessada apenas pelas classes de “negócio”, mas é necessário que seja obtida pela classe “Fabrica”, sendo assim ela deve ter sua privacidade definida como “internal” podendo ser acessada diretamente apenas no mesmo projeto.

### Interface de Núcleo/Domínio

Deverá conter apenas as assinaturas dos métodos que serão “externalizados”. Logo o ambiente externo só terá acesso ao que for definido nas interfaces.

# Padronização de Código

## Módulos

São agrupamentos de rotinas que solucionam um “pedaço” do sistema, Ex.: financeiro, contábil, expedição, faturamento...

### Estrutura de Pastas

* Raiz
  + Dominio
  + Modelo
  + Fachada
  + Nucleo
  + ModeloContexto.cs

## Projetos

* F3M.WTE.**<NomeModulo>**
  + **Exemplo:** F3M.WTE.**Financeiro**, F3M.WTE.**Contabilidade**
* F3M.WTE.<NomeModulo>**.Teste**
  + **Exemplo:** F3M.WTE.Financeiro**.Teste**, F3M.WTE.Contabilidade**.Teste**

## Interfaces

Sempre iniciar com a letra **I**

* **I**<NomeInterface>: Para interfaces fora de padrões de camada
  + **IRepository,**
* **IN**<NomeInterface>: Para interfaces de camada de Núcleo
  + **IN**ContasReceber
* **ID**<NomeInterface>: Para interfaces de camada de domínio
  + **ID**ContasReceber

## Classes

Sempre iniciar com a letra maiúscula, no padrão **C**amel**C**ase.

* ClassePadrao
* **N**<NomeClasse>: Para classes de núcleo
  + **N**NucleoClasse, **N**ContasReceber, **N**Cliente
* **D**<NomeClasse>: Para classe de Dominio
  + **D**DominioClasse, **D**ContasReceber, **D**Cliente

## Propriedades

Sempre iniciar com letras maiúscula, no padrão **C**amel**C**ase. **Evite utilizar o moderador private em propriedades, prefira utilizar variáveis.**

* <moderador> <tipo> <NomePropriedade>{ get; set; } ou completa;

## Instâncias ou declarações de variáveis

Devem **sempre** iniciar por letras minúsculas, no padrão **C**amel**C**ase. Não utilize prefixos (obj, int, str), seja direto e claro quanto ao que será feito daquela variável, coloque um nome de fácil leitura e entendimento. Sempre que possível utilize português para identificar variáveis, evite pronomes e verbos de ligação (De, Se).

Exemplos

Cliente clienteMatriz = new Cliente();

int soma~~Dos~~ValoresNegativos = 0 🡺 int somaValoresNegativos = 0;

### Variáveis

**Evite utilizar variáveis públicas quando forem variáveis de classe, prefira utilizar propriedades**.

int somaObjetos = 0;

private int usuarioId = 0;

## Contexto de Banco de Dados

Deve ser instanciada fora das camadas. Para que possa ser passada através de diferentes objetos. **Não deve ser iniciado o contexto de conexão nas classes de domínio ou negócio.**