**Título**: Persistência com NHibernate

**Descrição**: Veja neste artigo como podemos facilitar as operações de persistência com o NHibernate através de um exemplo de mundo real com uso de princípios S.O.L.I.D., Componentização de nossos assemblies e como bônus vamos ensinar o nosso leitor, ao compilar diretamente seus componentes, os mesmos serem registrados automaticamente na GAC – Global Assembly Cache de forma que o mesmo fique disponível par ser compartilhado para quaisquer de suas aplicações .NET

**Autor**: gabrielsimas



A persistência sempre foi, e pelo visto sempre será, um grande desafio para qualquer desenvolvedor que se encontre entre uma modelagem e uma implementação, precisamos unificar padrões de projeto e melhores práticas de programação sempre que precisarmos operar para fora do domínio de nossa aplicação de forma segura, viável e que não consuma muitos recursos, e ao fazermos comunicação com o meio externo, ou seja, quando fazemos persistência com um banco de dados, arquivo binário ou outro tipo de fonte de dados precisamos quase que da perfeição e fazer com que nosso código fique o mais reutilizável possível para que outras equipes possam utilizá-lo e, com seu uso, ganhar tempo com suas implementações.

Diante desta necessidade, utilizamos frameworks já consolidados no mercado que no início de grandes projetos fazem com que tenhamos grande economia de tempo e recursos posto que apenas temos um esforço administrativo maior em sua configuração, ou treinamento dos desenvolvedores, no mais o tempo que seria gasto criando grandes estruturas para persistência cai menos da metade de todo o esforço. Como grande prova disso, temos o NHibernate, já tão mencionado em artigos anteriores deste autor e suas publicações na Revista .NET Magazine, como uma grande solução para persistência, com grande facilidade no que tange o mapeamento das classes, seja por XML, e agora também como um novo Projeto, o mapeamento por attribute para decorar as Classes.

Não fugindo muito deste assunto com relação a qual destas abordagens é a melhor: por mapeamento estático, ou seja, o uso do XML ou o mapeamento dinâmico por meio dos attributes. Por convenção e didática, vamos utilizar neste artigo o mapeamento estático, deixando o mapeamento dinâmico para artigos posteriores como forma de complementar esta nossa série de artigo sobre NHibernate e Spring.NET.

**Estudo de caso: Persistência para um Sistema de Controle de Artigos.**

Sabemos bem que existe uma forma de conseguir uma renda extra e esta forma é fazer artigos para a DevMedia e com isto existe a necessidade de se criar um Sistema, além de controlar os artigos que já forma feitos de uma forma inicial, tambem dá para termos a noção de quanto arrecadamos com a devMedia nesta belíssima troca profissional.

Utilizamos, nesta etapa inicial, o controle de nosso cronograma através do Scrum, porém não falaremos de scrum após esta primeira parte, posto que não faz parte do escopo de nosso artigo.

Nossas necessidades são expressas no Épico “Artigos” abaixo:

* Como Autor, eu preciso cadastrar todos os meus artigos;
* Como Autor, eu preciso ter um controle dos meus artigos quanto ao tema;
* Como Autor, eu preciso controlar os cronograma do artigo com datas de inicío,fim e publicação no site da devmedia;
* Como Autor, eu preciso controlar as tecnologias presentes em cada artigo. Exemplo: se são .NET, Java e etc.
* Como Autor, eu preciso manter um cadastro de Temas

Com isso, o Item Artigo se tornou um Épico com as Estórias de Usuário descritas anteriormente.

Já temos então a solicitação do que precisamos, agora, o nosso Project Owner, separou o épico em Backlog de Produto e o mesmo ficou com apenas um item: Controle de Artigos.

Após a nossa Sprint Planning Meeting decidimos quebrar este Item do Backlog de Produto (PBI) da seguinte maneira, como mostra a Figura 1.

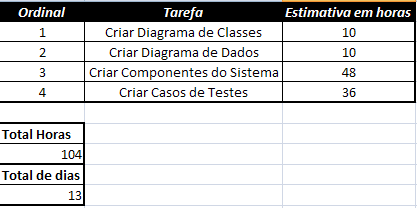


Figura 1 - Tarefas estimadas

Pelo que pudemos ver na Figura 1, teremos 13 longos dias de projeto. Não vamos mostrar aqui o quadro branco do Scrum até porque ele foge do escopo de nosso artigo, mas já deu para entender que temos uma Sprint bem curtinha até porque o nosso artigo não durou 13 dias para ser confeccionado, porém, só para o leitor ter uma amostra em como as metodologias ágeis facilitam o nosso dia-a-dia mesmo com poucas funcionalidades a serem efetuadas.

Diante disto, vamos criar o nosso Modelo de Dados inicialmente com o nosso Diagrama de Classes, afinal de contas, nosso Modelo é Object centric, ou seja, primeiro o Diagrama de Classes para depois criar o Diagrama de Entidade Relacionamento do Banco de dados refletindo o nosso diagrama de classes.

**Diagrama de Classes**

Na Figura 2, temos justamente o nosso Diagrama de Classes de acordo com o levantamento anterior.

**Modelo de dados**

**Descrição da Estrutura da Solução**

**Domínio**

**Mapeamento com NHibernate**

**Mapeamento 1-N**

**Mapeamento 1-1**

**Mapeamento reverso: uma polêmica.**

**Antes de criar a Persistência**

**TDD com NUnit: encontrando erros inesperados antes de implementar o seu Sistema**

**Testando o nosso Modelo: Teste de Integração**

**Criando o componente para reuso eterno: Genericos com o NHibernate**

**Ordem correta de criação e desenvolvimento**

**Interfaces**

**Um aparte: Usando Singleton para criar a Fábrica de Sessões**

**Classe Genérica**

**Implementação no Componente de DAL**

**Segundo Caso de Testes: Teste de Regressão**

**Componente de Negócios: usando o DTO**

**Criando uma classe de cópia entre a DAL e o DTO: Encapsulamento no Buraco Negro**

**Utilizando fachadas para a Regra de Negócios: Uma proteção.**

**Após o desenvolvimento: a GAC**

**Para a GAC, vamos assinar o nosso Componente?**

**Conclusão**