Aplicação do ORM Doctrine para abstração de banco de dados no desenvolvimento de software em PHP na UNA-SUS/UFMA

Lázaro Henrique C. Marques e Dilson José L. R. Júnior

Setor de Inovação Tecnológica
Universidade Aberta do SUS – São Luís – MA

{dilsonrabelo.unasus, lazaro.hcm}@gmail.com

Abstract. *PHP is a language commonly used for developing web applications because of its easy integration with HTML and various DBMSs (Database Management System). The choice of working with relational database can require extra effort from the programmer, with manipulation and safety. In this paper, we show the migration of a structured and vulnerable model to the advantages obtained by using an ORM (Object Relational Mapping) for database management and data persistence using an Object Oriented approach, allowing a big gain of time in the development and increased focus on security and analysis of business applications facing WEB. With the time saved, this approach will allow the team to work on systems documentation, thereby making the process of maintenance, easily if necessary.*

Keywords: ORM, PHP, Persistence of data, Driven-Domain Design, Doctrine

Resumo. *PHP é uma linguagem comumente utilizada para o desenvolvimento de aplicações WEB devido a sua fácil integração com HTML e os mais diversos SGBDs (Sistema de Gerenciamento de Banco de Dados). A escolha de se trabalhar com banco de dados relacional pode exigir muito tempo do programador quanto a manipulação e segurança do mesmo. Neste trabalho, mostramos a migração de um modelo de desenvolvimento estruturado e vulnerável para as vantagens obtidas com a utilização de um ORM (Object Relational Mapping) para o gerenciamento do banco e persistência de dados utilizando uma abordagem Orientada a Objetos, que permitiu um grande ganho de tempo no processo de desenvolvimento e maior foco na segurança e análise de negócio das aplicações voltadas para WEB. Com o tempo ganho, essa abordagem permitirá que a equipe possa trabalhar na documentação dos sistemas, tornando assim o processo de manutenção, se necessário, fácil nos softwares desenvolvidos.*

Palavras Chaves: ORM, PHP, Persistência de dados, Driven-Domain Design, Doctrine

1. Introdução

A UNA-SUS (Universidade Aberta do SUS) é um projeto do Ministério da Saúde, desenvolvido pela Secretária de Gestão do Trabalho e da Educação na Saúde. A UFMA (Universidade Federal do Maranhão), adere ao projeto ao final de 2009 e tem suas

atividades iniciadas em 2010. A UNA-SUS/UFMA tem como negócio a oferta de cursos de especialização e aperfeiçoamento, mediados por tutores, de forma gratuita, na modalidade de educação a distância, utilizando para isso a Plataforma Moodle. Oferece também cursos de extensão na modalidade autoinstrucional.

O Moodle é um ambiente virtual de aprendizagem que, trabalha com uma perspectiva dinâmica de aprendizagem em que a pedagogia socioconstrutivista e as ações colaborativas ocupam lugar de destaque [Silva, 2013].

O Moodle não é a única ferramenta tecnológica adotada pela a UNA-SUS/UFMA, ela também possui um leque extenso de aplicações, todas desenvolvidas utilizado a linguagem de programação PHP (PHP: Hypertext Preprocessor), no qual podemos citar, módulo de inscrição, módulo de monitoria de atividades do Moodle, módulo de ambiente de TCC (Trabalho de Conclusão de Curso), dentre outras funcionalidades. Todo esse conjunto de implementações se encontram em um processo de transição do paradigma estrutural para o orientado a objetos. Uma arquitetura em camadas foi proposta, como item de soma no processo, e percebeu-se a necessidade de se abstrair a camada de persistência de dados, já que consome grande parte do desenvolvimento. Mas qual a melhor solução a se aplicar nesse contexto? Apoiando-se em três pilares, produtividade, segurança e manutenibilidade, aplicou-se o uso de ORM (Object Relational Mapping) para este tipo implementação. Com a decisão de aplicar o uso de ORM para o desenvolvimento de novos aplicativos para a UNA-SUS/UFMA, esta decisão nos levou a um segundo questionamento. Qual o melhor ORM em PHP a ser implantado para o desenvolvimento dos softwares na UNA-SUS/UFMA?

Como roteiro, este trabalho encontra-se estruturado da seguinte maneira. Na **Seção 2** descreve-se a Fundamentação Teórica. Na **Seção 3** é descrita a metodologia de implantação do ORM Doctrine. Na **Seção 4** são apresentados os resultados obtidos após a implementação do ORM e finalizando na **Seção 5** apresentamos as conclusões obtidas deste trabalho.

2. Fundamentação Teórica

2.1 PHP: Hypertext Processor

PHP (PHP: Hypertext Preprocessor) é uma linguagem de programação WEB, criada em 1994, por Rasmus Lerdorf, seus primeiros *scripts* dinâmicos criados foram utilizados para monitorar número de acesso ao currículo do seu criador [Dall'Oglio, 2007]. A versão mais atual da linguagem se encontra na 5.6 é um software livre publicado sobre a licença GPL (General Public License). Possui um conjunto de extensões modularizadas, o que torna sua aplicabilidade adequada para servidores WEB [Suehring, Park e Morgan, 2009].

Nos dias atuais a sua usabilidade é voltada para o desenvolvimento WEB, esse uso ganhou mais força a partir da versão 5 com a implementação dos conceitos da Orientação a Objetos na linguagem. Isso permitiu o surgimento de ferramentas e frameworks que facilitaram o desenvolvimento de novas aplicações.

2.2 Object Relational Mapping

Quando se retrata o cenário de desenvolvimento de software com banco de dados relacional existe uma preocupação que envolve a manipulação de SQL (Structured

Query Language – linguagem padrão para se manipular banco de dados relacionais [Date, 2003]) e segurança de banco de dados. Essa preocupação tende a consumir do programador grande parte do seu tempo de desenvolvimento.

Persistência de dados é o ato de armazenar dados em uma mídia com a capacidade de escrita, que neste caso refere-se ao banco de dados relacional. De acordo com Bauer e King (2004), é de pouca utilidade um sistema que não consegue armazenar informações para serem manipuladas posteriormente.

Com o advento da Orientação a Objetos surge uma técnica de programação denominada ORM (Object Relational Mapping), que tem como proposta abstrair toda essa camada de persistência de dados. O ORM representa em objetos as tabelas do banco de dados relacional, utilizando metadados que descrevem o mapeamento entre os objetos e banco dados [Bauer e King, 2004].

Ainda com Bauer e King (2004), o uso do ORM pode ser considerado como vantajoso sob os seguintes critérios, produtividade, segurança, manutenibilidade e independência do fornecedor do banco de dados. No que diz respeito ao critério de produtividade eles defendem que o ORM deve possuir a capacidade de abstrair os métodos de manipulação com o banco de dados relacional. No critério manutenibilidade, permite uma redução de linhas de código, isso torna o código mais compreensível e facilita a manutenção. Em segurança, deve garantir a aplicação de filtro de dados e validações, antes de encaminhar para o banco relacional. No último critério, independência do fornecedor do banco de dados, um ORM deve ter a capacidade de se adaptar a diferentes tipos de SGBD (Sistema de Gerenciamento de Banco de Dados).

Hoje no mercado existem inúmeros frameworks, em diversas linguagens de programação, que adotam a técnica de programação. Em cada framework temos implementações e configurações divergentes, mas que em essência seguem os pontos definidos por Bauer e King.

2.3 ORM Doctrine

O Doctrine é um framework de código fonte aberto, no qual tem como objetivo criar uma série de bibliotecas PHP para tratar das funcionalidades de persistência de dados e funções relacionadas a isto [Minetto 2014]. O ORM é um dos componentes do framework, sua implementação sustenta os critérios definidos por Bauer e King, isso graças a forma a sua arquitetura foi desenvolvida.

Sua arquitetura foi modelada seguindo o padrão de projeto Unit of Work. Este padrão controla a persistência de dados no banco relacional. Em um cenário onde pode ocorrer inúmeras mudanças, inclusões ou exclusões de dados, pode existir o uso excessivo do controle de transações. A proposta do padrão é gerenciar esse contexto de uso das transações e persistência de dados auxiliando na utilização adequada no controle das transações [Fowler, Rice, Foemmel, Hieatt, Mee, Stafford (2002)].

Ainda segundo Minetto (2014), o Doctrine tornou-se uma ferramenta para solucionar o problema de objeto relacional no ambiente PHP e vem sendo usado por projetos de diversos tamanhos e frameworks.

3. Metodologia de Implantação

A implantação do framework ORM Doctrine apresentado neste trabalho, foi motivado pelo cenário de desenvolvimento em que os softwares da UNA-SUS/UFMA se encontravam.

Esse cenário tinha uma abordagem prática e funcional, mas não adequada nos aspectos de segurança, manutenibilidade e produtividade. Todo novo software desenvolvido era estruturalmente criado de forma inadequada para os preceitos encontrados atualmente. Esse tipo de estrutura mantinha o software extremamente acoplado e de difícil manutenção.

No critério de segurança, um ponto forte para mudança, foi a abertura que o código permitia para injeção de código SQL através da URL (Uniform Resource Locator), ou comumente conhecido como *SQL Injection*. Essa falha foi identificada ao ser observado trechos de SQL com concatenação de dados e sem parametrização das informações. O reforço desta primícia se dá também, no uso da extensão PHP denominada MYSQL, na qual não oferece suporte a parametrização de SQL e foi descontinuada do PHP desde a versão 5.

A inexistência da separação entre negócio, visão e domínio afetavam diretamente a produtividade e a manutenção. Estas camadas eram representadas em um só contexto, marcações HTML (Hypertext Markup Language) misturadas com SQL e negócio.

Diante desses pontos, uma solução precisava ser apresentada e implementada para solucionar o problema.

3.1 Uma arquitetura de software

No cenário encontrado a melhor solução para este tipo de abordagem é bem nítida, onde a busca seja a organização de software e definição de camadas claras e consistentes, a aplicação de uma arquitetura de software baseada no paradigma da Orientação a Objetos com associação de DDD (Domain-Driven Design). Evans (2009), define DDD como um conjunto de conceitos, para a elaboração de software com foco no domínio, considerando-o como o coração da implementação.

Na definição dessa arquitetura, camadas seriam criadas e cada uma seria responsável por implementar somente aquilo que foram designadas. A arquitetura seria uma variação do padrão de projeto MVC (Modelo, Visão e Controle), mas com um conjunto de pequenas adições. Nessa variação, em uma das camadas adicionadas atribuiu-se a responsabilidade para persistência dos dados.

3.2 A definição de uma camada de persistência

Na camada de persistência de dados encontramos duas abordagens que poderiam satisfazer a necessidade da arquitetura, a primeira delas é o uso da extensão PDO (PHP Data Object), uma interface padrão para manipulação de banco de dados, e a segunda a de um framework ORM, mas qual abordagem adotar? A definição da melhor abordagem a ser utilizada foi caracterizada pelo levantamento de critérios definidos em discussão com a equipe de desenvolvimento da UNA-SUS/UFMA e fundamentada em autores especialistas na área, como exemplo Bauer e King. Na tabela abaixo são

destacados os critérios, as justificativas e a melhor aplicabilidade para uso da camada de persistência de dados.

Tabela 1. Quadro de critério para definição da camada de persistência

Critérios	PDO	ORM	Melhor Aplicabilidade
Complexidade de persistência	Implementação de todos as Instruções (INSERT, UPDATE e DELETE)	Abstração em objetos	ORM
Desempenho	Manipulação direta ao Banco de Dados	Implementação de cache para manipulação e consulta de dados	PDO
Integração com o domínio	De difícil integração, retorno de dados efetuado em array ou objetos. Escrita de código-fonte que efetuasse essa integração.	Integração total ao Domínio	ORM
Consulta de dados	Manipulação direta com a camada de persistencia de dados.	Integrada ao domínio	ORM
Segurança de dados	Implementada durante a persistência	Abstração no modelo	ORM
Alteração de campos no banco de dados	Grande impacto na aplicação	Adequação de campos no domínio	ORM
Mudança de banco de dados	De fácil implementação	De fácil implementação	PDO/ORM

Como demonstrado na tabela o uso de ORM é mais adequado ao tipo de abordagem que os softwares da UNA-SUS/UFMA buscam. Mas isso nos leva a um novo questionamento, qual ORM adotar para a arquitetura?

3.3 A escolha do ORM Doctrine para o desenvolvimento de softwares na UNA-SUS/UFMA

Na busca de um ORM compatível com a linguagem PHP que adequasse a arquitetura dos softwares da UNA-SUS/UFMA, dois frameworks se destacaram, são eles: o Doctrine e o Propel. Ambos foram avaliados sob os seguintes critérios: modelos que refletissem a estrutura de dados relacional, abstração de instruções básicas (CRUD - *create, read, update and delete*), linguagem abstrata de consulta e desempenho. Nesses critérios destacamos características de ambos frameworks, já definindo pontos positivos e negativos com intuito de atingir a necessidade da arquitetura.

Modelos que refletissem a estrutura de dados relacional: No Propel o mapeamento dos modelos são implementados através da escrita de um arquivo XML e a partir de uma ferramenta geradora de código e da leitura do arquivo é que os modelos são criados. No Doctrine esse mapeamento é efetuado através de notações, aplicadas diretamente em classes escritas pelo programador. Neste item, para atender os critérios da arquitetura em questão, o Doctrine leva vantagem sobre o Propel, pois não necessita da manipulação com XML e nem de uma ferramenta geradora de código, isso pode gerar impacto durante possíveis mudanças na estrutura de dados relacional. As figuras 1 e 2 a seguir demonstram respectivamente o mapeamento do Propel e Doctrine.

```
<?xml version="1.0" encoding="UTF-8"?>
<database name="bookstore" defaultIdMethod="native">
  <table name="book" phpName="Book">
    <column name="id" type="integer" required="true" primaryKey="true" autoIncrement="true"/>
    <column name="title" type="varchar" size="255" required="true" />
    <column name="isbn" type="varchar" size="24" required="true" phpName="ISBN"/>
    <column name="publisher_id" type="integer" required="true"/>
    <column name="author_id" type="integer" required="true"/>
  </table>
  <table name="author" phpName="Author">
    <column name="id" type="integer" required="true" primaryKey="true" autoIncrement="true"/>
    <column name="first_name" type="varchar" size="128" required="true"/>
    <column name="last_name" type="varchar" size="128" required="true"/>
  </table>
  <table name="publisher" phpName="Publisher">
   <column name="id" type="integer" required="true" primaryKey="true" autoIncrement="true" />
   <column name="name" type="varchar" size="128" required="true" />
  </table>
</database>
```

Figura 1. Mapeamento dos modelos no Propel ORM

```php
<?php
/** @Entity */
class Message
{
    /** @Column(type="integer") */
    private $id;
    /** @Column(length=140) */
    private $text;
    /** @Column(type="datetime", name="posted_at") */
    private $postedAt;
}
```

Figura 2. Mapeamento dos modelos no ORM Doctrine

Abstração de instruções básicas: Em ambos os frameworks, o uso de instruções básicas para criação de um CRUD é relativamente parecida. Métodos como *save()*, *persist()* e *delete()* são implementados nos dois frameworks. As definições de consulta (*read*), são detalhadas no item a seguir.

Linguagem abstrata de consulta: Em ambos frameworks, implementam esse conceito de forma bem estruturada. No Propel, a consulta de dados pode ser implementada de duas formas, SQL Nativo e manipulação de classes. O mesmo se aplica para o Doctrine com um adicional na implementação de uma pseudo-linguagem, denominada DQL (Doctrine Query Language). As figuras 3 e 4, exemplificam o modelo de consulta abstrata no Propel e Doctrine respectivamente.

```php
<?php
$books = BookQuery::create()
  ->filterByISBN('0140444173')
  ->useAuthorQuery() // returns a new AuthorQuery instance
    ->filterByFirstName('Leo') // this is an AuthorQuery method
  ->endUse() // merges the Authorquery in the main Bookquery and returns the BookQuery
  ->orderByTitle()
  ->limit(10)
  ->find();
```

Figura 3. Pseudo-linguagem de consulta no Propel ORM

```php
<?php
$query = $em->createQuery("SELECT u.id FROM CmsUser u WHERE CONCAT(u.name, 's') = ?1");
$query->setParameter(1, 'Jess');
$ids = $query->getResult();

$query = $em->createQuery('SELECT CONCAT(u.id, u.name) FROM CmsUser u WHERE u.id = ?1');
$query->setParameter(1, 321);
$idUsernames = $query->getResult();
```

Figura 4. Pseudo-linguagem de consulta no ORM Doctrine

Desempenho: Neste último item um teste de stress foi aplicado sobre ambos frameworks, para avaliar tempo de resposta e número de registros afetados. Como ferramentas de inferência, foram usados os seguintes fatores e artefatos:

Fatores: Registros por Minutos.

Artefatos: MacBook Pro. Processador 2.5 GHZ. 8 GB Memória RAM DDR3 1.6 GHZ, Framework PHPUnit para testes unitários e NetBeans PHP 8.1 como ambiente integrado de desenvolvimento

Testes aplicados: Inserção de dados e consulta de informações.

Como amostragem foram efetuados 3(três) testes. O resultado médio obtido foi:

Tabela 2. Quadro de resultados da amostragem de desempenho

	Quantidade registros afetados	Tempo (minutos)
Doctrine	100	65
Propel	100	90

Inserção de dados

	Quantidade registros afetados	Tempo (minutos)
Propel	100	0,5
Doctrine	100	0,4166

Consulta de dados

Tomando esses critérios como pontos para decisão o mais sensato para o tipo de aplicação será utilizar o ORM Doctrine como ferramenta para persistência de dados.

4. Resultados

Com a adoção de uma nova arquitetura em que escolheu-se utilizar o framework ORM Doctrine para implementação dos sistemas desenvolvidos na UNA-SUS/UFMA o processo de persistência de dados se tornou eficiente e eficaz, o que permitiu se ter mais foco em outros pontos críticos do processo de criação de softwares, como validação de domínio e interoperabilidade.

O modelo de dados e forma como se trabalhava tornou-se desacoplada do restante da aplicação, totalmente gerenciada pelo Doctrine não mais pensado sobre demanda, muita vezes escrita de formas diferentes para finalidades semelhantes e de difícil reutilização.

Preocupações quanto a consistência dos dados trabalhados no banco de dados relacional passaram a ser bem menores, uma vez que, após mapeadas as tabelas, o próprio Doctrine passa a gerenciá-las como objetos, permitindo tanto aqueles que desenvolveram quanto os que venham a desenvolver a aplicação não demandem muito tempo pensando em como serão as estruturas com as quais precisam trabalhar. Tudo isso só foi possível por ter-se adotado uma aplicação orientada a objetos em que utilizou-se o conceito de modelo de domínio. Assim, se mostrou bastante acolhedora a utilização de classes para representar tabelas e atributos representando as colunas do Banco de Dados da aplicação.

A simplicidade ou até mesmo a não necessidade de *querys* foi mais um ponto positivo da adoção do Doctrine ORM para o desenvolvimento na UNA-SUS/UFMA. Não houve mais a necessidade de se escrever a mão as primitivas de CRUD para cada entidade encontrada nos sistemas, um processo demorado e repetitivo. A automatização desse ponto, proporcionada pelo framework, foi bem vantajosa para a equipe e foi onde se poupou bastante tempo no processo de desenvolvimento pela quantidade de código que pôde ser reaproveitada durante a concepção do CRUD de entidades.

Consultas utilizando *Joins,* que para muitos programadores são complicadas foram facilmente contornadas com o mapeamento objeto-relacional. Os *Joins* foram tratados nas associações que objetos possuem uns com os outros, pois ao acessar um atributo que representa uma outra classe o próprio ORM entende a ação como um *Join*. Ao adentrar em vários níveis de associações é como se estivéssemos fazendo múltiplos *Joins,* porém de uma forma muito mais simples e intuitiva. Poucas foram as vezes em que necessitou-se trabalhar com *querys* que utilizaram *Join* e ainda assim o framework apresenta recursos como DQL para manipulação de *querys* desacopladas do SGBD.

A blindagem contra *SQL Injection* foi outro um bom resultado obtido do uso de ORM. Ele próprio utiliza um SQL parametrizado e fornece uma série de filtros para tipos de dados permitindo um controle das entradas de usuário.

Alterações ao longo e depois da conclusão de projetos, como inclusão de novas entidades, alteração de tipos de dados, adicionar novos atributos ou novas associações, não se mostraram uma grande preocupação, com o gerenciamento das entidades sendo realizado pelo ORM Doctrine, o impacto de mudanças nesse âmbito se mostraram mínimos.

5. Conclusão

A escolha do ORM Doctrine pela UNA-SUS/UFMA no desenvolvimento de seus projetos em PHP se mostrou bastante proveitosa. Obteve-se ganhos na organização do projeto, que passou a ter uma camada de persistência de dados totalmente independente do restante da aplicação o que deu uma maior flexibilidade no processo de desenvolvimento.

O tempo necessário para produção de um sistema completo foi drasticamente reduzido com a utilização de ORM, prazo para conclusão de sistemas deixou de ser uma das principais preocupações e empecilho, permitindo que os desenvolvedores se foquem em questões mais críticas do sistema, como segurança e lógica de negócio e cumprimento de requisitos, permitindo que os softwares sejam entregues completos e com qualidade, algo que não era garantido com o modelo anterior de desenvolvimento encontrado na UNA-SUS/UFMA.

Também com essa fatia do tempo de desenvolvimento ganha, pretende-se finalmente realizar a documentação rica dos sistemas e do modelo de desenvolvimento e arquitetural utilizados.

Referências

Dall'Oglio, Pablo. (2007) "PHP: Programando com orientação a objetos", Novatec Editora.

Suehring, Steve. Converse, Tim. Park, Joyce. (2009) "PHP 6 MySQL 6 Bible", Wiley Publish Inc.

Bauer, Christian. King, Gavin. (2005) "Hibernate in Action", Manning Publications.

Minetto, Elton. (2014) "Doctrine na prática", Lean Publish.

Fowler, Martin. Rice, David. Foemmel, Matthew. Hieatt, Edward. Mee, Robert. Stafford Randy. (2002) "Patterns of Enterprise Application Architecture", Addison Wesley.

Evans, Eric. (2010) "Domain-Driven Design – Atacando as complexidades no coração do software", Alta Books.

Silva, Robson S. da. (2013) "Moodle para autores e tutores", Novatec Editora.

Date, C. J. (2003) "Introdução a sistemas de banco de dados", Elsevier.

"Propel ORM", http://propelorm.org/, Outubro.

"Doctrine", http://www.doctrine-project.org/, Outubro.