INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE - IFSUL, *CAMPUS* PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

JOSÉ ÉRICO CAMERA SOARES

PROJETO E IMPLEMENTAÇÃO DE UM SISTEMA WEB VOLTADO À ÁREA DE NUTRIÇÃO UTILIZANDO AS TECNOLOGIAS *DOCTRINE* E *REST*

PASSO FUNDO
2015

JOSÉ ÉRICO CAMERA SOARES

PROJETO E IMPLEMENTAÇÃO DE UM SISTEMA WEB VOLTADO À ÁREA DE NUTRIÇÃO UTILIZANDO AS TECNOLOGIAS *DOCTRINE* E *REST*

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-Rio Grandense, *Campus* Passo Fundo, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador: Prof. Dr. Josué Toebe

PASSO FUNDO
2015

JOSÉ ÉRICO CAMERA SOARES

PROJETO E IMPLEMENTAÇÃO DE UM SISTEMA WEB VOLTADO À ÁREA DE NUTRIÇÃO UTILIZANDO AS TECNOLOGIAS *DOCTRINE* E *REST*

Trabalho de Conclusão de Curso aprovado em ____/____/____ como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet

Banca Examinadora:

_____

Prof. Dr. Josué Toebe (Orientador)

_____

Prof. Me. Rafael Marisco Bertei (Convidado)

_____

Prof. Dr. Alexandre Tagliari Lazzaretti (Convidado)

_____

Prof. Dr. Alexandre Tagliari Lazzaretti

Coordenação do Curso

PASSO FUNDO
2015

À minha família pelo apoio durante o curso.

“To strive, to seek, to find and not to yield.”
Autor: Alfred Lord Tennyson.

RESUMO

Com o constante avanço da tecnologia, um problema que surge são os diferentes tipos de interfaces gráficas de programação além das mais diversas plataformas com suas características e linguagens de programação fazendo com que os programadores tenham retrabalho de programar a todo instante o mesmo código quando uma interface muda ou quando ocorre a troca de plataforma.

Visando estas circunstancias, este trabalho apresenta a implementação de um software voltado à gestão de pacientes clinicamente atendidos por profissionais da área de nutrição, utilizando como ferramentas para tal feito a arquitetura *Representational State Transfer* (REST) como meio de comunicação entre cliente e servidor através do modelo de implementação chamado RESTfull e a ferramenta de mapeamento objeto-relacional *Doctrine*, como meio de abstração da programação da base de dados.

O trabalho também aborda pontos importantes da instalação e configuração das duas ferramentas citadas acima nos sistemas operacionais GNU/Linux e Windows além das instalações de complementos necessários para o funcionamento das mesmas. Como forma de exemplos práticos de utilização em um ambiente web de desenvolvimento, observa-se trechos de códigos fonte com suas respectivas explicações.

Por fim são apresentados considerações e apontamentos sobre os estudos realizados, bem como dificuldades encontradas e experiências obtidas durante o desenvolvimento.

Palavras-chave: rest; doctrine; mapeamento; abstração; cliente; servidor.

ABSTRACT

With the constant advancement of technology, a problem that arises are the different types of graphical programming interfaces in addition to many different platforms with its features and programming languages making programmers have rework program all the time the same code when an interface changes or when there is the exchange platform.

Targeting these circumstances, this paper presents the implementation of a software oriented to the management of patients medically attended by professionals in nutrition, using tools such Representational State Transfer (REST) as a means of communication between client and server through implementation model called RESTfull and a tool for object-relational mapping Doctrine as a means of abstraction programming database.

The work also addresses important points in the installation and configuration of the two tools above in the operating system GNU/Linux and Windows in addition to the complementary facilities necessary for their operation. As a way of practical examples of use in a developing web environment, is observed source code snippets and their respective explanations.

Finally they are presented considerations and notes on the studies, as well as difficulties and experiences obtained during development.

Key words: rest; doctrine; mapping; abstraction; client; server.

LISTA DE FIGURAS

LISTA DE ABREVIAÇÕES E SIGLAS

REST: *Representational State Transfer*.

MVC: Modelo, Visão, Controle.

HTML: *HyperText Markup Language*.

PHP: *PHP: Hypertext Preprocessor*.

HTTP: *Hypertext Transfer Protocol*.

OSI: *Open Systems Interconection*.

URI: *Uniform Resource Identifier*.

ORM: *Object Relational Mapper*.

SQL: *Structured Query Language*.

IDE: *Integrated Development Environment*.

SUMÁRIO

1. INTRODUÇÃO

Atualmente vive-se em um mundo que exige cada vez mais profissionais qualificados e ágeis. Isso não poderia ser diferente na área da informática, ramo de atuação em que é ainda mais exigida a excelência e a qualidade com reduzido espaço para erros. Para um desenvolvedor de softwares, essa exigência torna-se um dilema profissional, pois é difícil desenvolver um programa com rapidez e, ao mesmo tempo, sem erros.

Como uma tentativa de auxiliar o programador, surgem ferramentas para facilitar o trabalho do desenvolvedor de aplicações. Essas opções de trabalho visam possibilitar maior agilidade em suas produções e aproximar os erros durante o decorrer do projeto a zero. O profissional que possui o conhecimento necessário para trabalhar com tais meios de produção, certamente obterá uma vantagem extra na qualidade de seus softwares e ainda como "bônus" ganhará mais tempo, sendo este usado em correções e ajustes no próprio programa desenvolvido.

Desse modo e nesse sentido, dois métodos de desenvolvimento se destacam no ambiente de produção web: o primeiro deles é o uso do modelo arquitetônico REST, cujos benefícios são facilmente percebidos, por exemplo, usando-o como *web service*, integrando o programa desenvolvido com outros tipos de plataformas que possuam linguagens diferenciadas, possibilitando uma forma de comunicação entre os envolvidos. O segundo método é a abstração da programação SQL no gerenciador da base de dados, economizando o tempo de ter que escrever a lógica de construção de tabelas para sua utilização, realizando isso nas próprias classes do programa à ser desenvolvido. A ferramenta estudada neste caso é chamada de *Doctrine*, seu objetivo é mapear as classes do projeto com notações para posteriormente fazer a criação da base de dados automaticamente.

1.1. MOTIVAÇÃO

A principal motivação é usar ferramentas que agilizem o trabalho do desenvolvedor em trabalhos de médio e grande porte (Doctrine) além de fazer uma distinção significativa entre cliente e servidor, delegando funções para cada um destes (REST).

Com a constante evolução tecnológica, principalmente das interfaces de usuário é de extrema importância saber separar lógica de negócio da programação visual, aquela em que o usuário vai interagir com o sistema, pensando em uma possível mudança no futuro sem afetar o sistema em todo, podendo este se adaptar a diferentes realidades e mudanças. Seguindo o

modelo de implementação RESTfull, não se faz necessário a utilização do modelo MVC, o que já é uma quebra de paradigma da programação atual para a web.

Além disso, dar liberdade ao programador para trabalhar apenas com a linguagem em que está especializado traz resultados mais rápidos e satisfatórios, como por exemplo, abstrair o uso de SQL utilizando mapeamentos ORM do Doctrine, trabalhando nas próprias classes do projeto.

1.2. OBJETIVOS

1.2.1. OBJETIVO GERAL

O objetivo é construir um software usual para os profissionais e estudantes da área de nutrição que seja intuitivo e rápido utilizando as ferramentas *Doctrine* como meio de abstração da programação SQL no gerenciador de banco de dados e *REST* com o objetivo de projetar um modelo diferenciado do padrão MVC além de explorar melhor os recursos usados no protocolo HTTP.

1.2.2. OBJETIVOS ESPECÍFICOS

- Analisar outros *softwares* do mesmo segmento para compreender suas funcionalidades e pontos fracos;
- Coletar requisitos com profissional da área de nutrição;
- Estruturar o modelo relacional que ajudará no mapeamento das classes;
- Implementar um protótipo utilizando as tecnologias REST e *Doctrine*.

2. REFERENCIAL TEÓRICO

Nesta seção são apresentados conceitos de implementação referente ao uso das tecnologias *REST* e *Doctrine* e como estas ferramentas podem ser implementadas em um ambiente web utilizando linguagens tradicionais como HTML e PHP seguindo o tradicional protocolo *Hypertext Transfer Protocol* (HTTP).

2.1. REPRESENTATIONAL STATE TRANSFER (REST)

Para começar a compreender como funciona a tecnologia *REST*, deve-se ter conhecimentos no protocolo HTTP, pois seu funcionamento é todo baseado neste tipo de processamento de dados.

2.2. HTTP

Em uma breve explicação, segundo Mendes (2007, p. 23), HTTP é o responsável entre a comunicação do navegador com o servidor responsável por manter as páginas hospedadas. Seguindo o modelo *Open Systems Interconection* (OSI), o HTTP está localizado na camada de aplicação. De acordo com Gourley e Totty (2002, p. 8), esta comunicação ocorre através de requisições, podendo ser do tipo *request messages* ou *send messages*. Os autores ainda citam que toda requisição, seja ela do tipo *request* ou do tipo *send,* utiliza métodos padrões próprios deste protocolo que servem para a comunicação entre servidor e cliente, os quais são:

a) ***GET*:** Envia recurso nomeado do servidor para o cliente.

b) ***PUT*:** Armazena dados do cliente em um recurso nomeado do servidor.

c) ***DELETE*:** Exclui um recurso chamado do servidor.

d) ***POST*:** Envia os dados do cliente para o servidor de aplicação.

e) ***HEAD*:** Envia os cabeçalhos de resposta para o recurso chamado.

f) ***OPTIONS:*** Solicita os meios de comunicações disponíveis na cadeia de solicitação. (GOURLEY; TOTTY, 2002, p. 57).

Muitos profissionais no dia-a-dia usam apenas os métodos *get* e *post* deixando de lado os restantes, assim não aproveitando a total capacidade que o protocolo HTTP oferece. (TILKOV, 2007).

Outra questão importante é conhecer os possíveis status de mensagens do servidor durante a comunicação com o cliente. Eles são mostrados através de códigos numéricos e podem ser:

a) 200: A requisição ocorreu normalmente.

b) 302: Redirecionamento para obter a resposta.

c) 404: Quando o recurso ou requisição não é encontrado.

d) 500: Erro interno do servidor. (GOURLEY; TOTTY, 2002, p. 49).

Quando um erro acontece no servidor, a resposta do cliente é o código do erro com uma frase explicativa quanto ao acontecido. (GOURLEY; TOTTY, 2002, p. 8).

O protocolo HTTP pode parecer simples à primeira impressão pela sua facilidade de compreensão, já que todas as suas requisições são realizadas em textos simples, compreensível a qualquer pessoa, mas, na verdade, é uma poderosa ferramenta de comunicação a qual o modelo *REST* sabe fazer um uso excelente já que o projetista, tanto do HTTP quanto do REST é o mesmo.

2.3. REST

REST é um modelo arquitetônico de autoria de Roy Thomas Fielding, lançado em sua tese de doutorado no ano de 2000. Segundo Fielding (2000), este modelo foi desenvolvido durante seis anos com ênfase nos primeiros seis meses do ano de 1995.

> *REST* enfatiza a escalabilidade de interações de componentes, a generalidade das interfaces, implantação independente de componentes e componentes intermediários para reduzir latência de interação, reforçar a segurança e encapsular sistemas legados. (FIELDING, 2000, p. xvii, trad. nossa).

Uma das grandes vantagens em usar este modelo arquitetônico é que ele foi projetado pelo mesmo desenvolvedor do protocolo HTTP como citado anteriormente. Além disso, esse mesmo indivíduo participou da produção de um dos principais servidores *web* da

atualidade conhecido por *Apache*, ou seja, dificilmente existirá problemas de incompatibilidades entre as principais bases de desenvolvimento.

De acordo com Tilkov (2007), *REST* é um conjunto de princípios que definem como deve ser usado *Web standards* do tipo HTTP e *Uniform Resource Identifier* (URI). Seguindo sua linha de raciocínio, Tilkov ainda cita cinco princípios fundamentais do modelo de desenvolvimento usando a arquitetura *REST*:

a) Dar a todos os recursos um identificador (ID)

Este conceito visa identificar todos os recursos de um sistema desenvolvido. Por exemplo, uma loja virtual possui vários produtos, cada qual com sua URI, ou seja, seu identificador único. Existe a controvérsia em que devemos mostrar o mínimo de dados possíveis referentes ao banco de dados, por exemplo o identificador de um cliente (id) ou o identificador de um endereço, mas relacionando isso a uma URI serão obtidos resultados mais persistentes. Na idealização de uma URI, deve-se focar na facilidade de encontrar o que se deseja dentro de um sistema. (TILKOV, 2007).

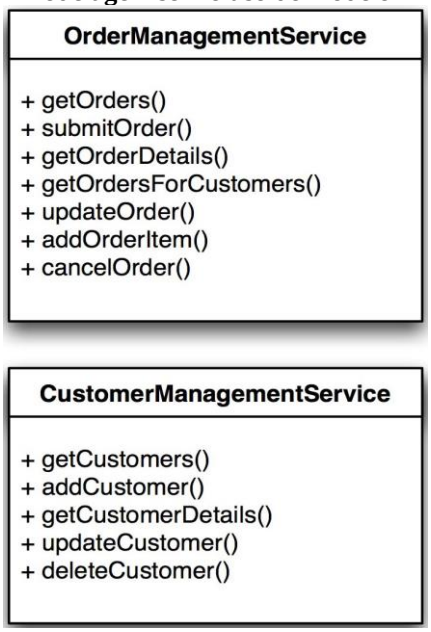
b) Associar os recursos

Associar recursos é o conceito de usar links em todas as funcionalidades que puderem ser referenciadas, abusando desse método para se obter melhores resultados dentro do sistema já que todos estão familiarizados com eles. (TILKOV, 2007).

c) Utilizar os métodos padrões

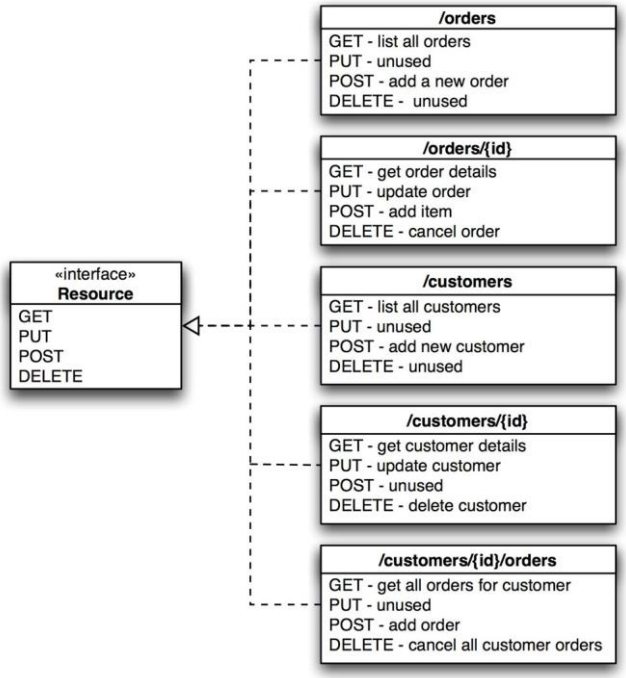
Usar os métodos padrões HTTP beneficia muito um sistema desenvolvido utilizando *RESTfull*. Praticamente qualquer aplicação ou serviço, desenvolvido ou não para ambiente *web* que queira fazer comunicação via rede utilizará este protocolo, ou seja, seu sistema atenderá os requisitos básicos para comunicação com o outro, possivelmente idealizado em uma plataforma diferente. Na figura 1, observa-se uma modelagem sem o uso do modelo *RESTfull,* consequentemente sem nenhuma integração com outros sistemas; enquanto na figura 2, pode-se visualizar como os métodos HTTP podem fazer a comunicação entre dois sistemas heterogêneos. (TILKOV, 2007).

Figura 1 – Modelagem sem o uso do modelo *RESTfull.*



Fonte: TILKOV, 2007.

Figura 2 – Modelagem implementando o modelo *RESTfull*



Fonte: TILKOV, 2007.

d) Recursos com múltiplas representações

Este conceito aborda a praticidade de se alcançar algum recurso dentro do sistema, chegando a ele por diferentes caminhos. Um dos problemas destacados por Tilkov (2007) é que não existem padrões específicos para o acesso a estes recursos, mas, em um ambiente menor, como, por exemplo, uma rede de colaboradores, pode-se facilmente criar um padrão

de acesso em que todos possam compartilhar os mesmos recursos do sistema. (TILKOV, 2007).

e) Comunicação sem estado

O conceito de não guardar estado no *REST* traz como benefício dois pontos em destaque, são eles: o primeiro é que o servidor suportará mais requisições do que em uma aplicação comum, já que, ao fazer a requisição, o cliente recebe a resposta e nada fica no servidor. O segundo benefício se dá ao fator pós requisição, sendo que depois que os dados estiverem com o cliente qualquer problema que possa ocorrer por parte do servidor não interferirá nos dados já recebidos - talvez o cliente nem perceba o ocorrido. (TILKOV, 2007).

2.4. DOCTRINE

Segundo o site do projeto *Doctrine*, a ferramenta *Doctrine* é um conjunto de bibliotecas voltadas ao armazenamento em banco de dados e mapeamentos de objetos relacionais mais conhecidos pela sigla ORM.

É recomendável que o programador que for utilizar esta ferramenta, tenha instalado PHP na versão mínima 5.4 e uma outra ferramenta chamada de *Composer*. (DUNGLAS, 2013, p. 7).

2.4.1. Composer

De acordo com Zemel (2012), *Composer* faz o gerenciamento das dependências dos projetos, ou seja, o programador especifica quais pacotes irá reutilizar em seus novos desenvolvimentos e o *Composer* automaticamente vai fazer a transferência alocando-os nos locais apropriados dentro do *workspace*. Segundo o site do *Composer* sua definição é a seguinte:

> Composer é uma ferramenta para gerenciamento de dependências em PHP. Ele permite que você declare as bibliotecas que seu projeto precisa e então ele fará a instalação delas por você. (*COMPOSER*, 2014, trad. nossa).

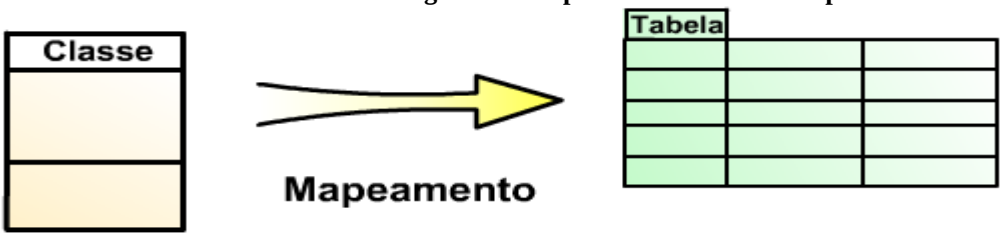
2.4.2. Object Relational Mapper (ORM)

Conforme Okano (2014), ORM é um *framework* que possui um conjunto de classes responsáveis por realizar a abstração do banco de dados, ou seja, o desenvolvedor não precisa se preocupar com códigos *Structured Query Language* (SQL) e com a criação de tabelas na

base de dados. Okano também cita algumas vantagens do uso de um ORM, como, por exemplo, aumento de produtividade, código mais legível e padronização de projeto.

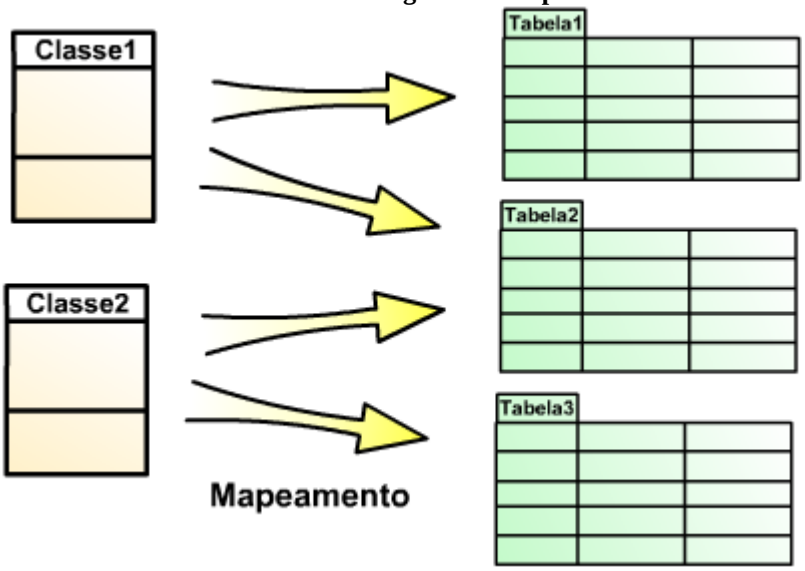
Usando um ORM, realiza-se o mapeamento das classes, basicamente para mapear, trabalha-se com dois itens principais, que são os atributos e métodos da classe. É importante ressaltar que nem todos os atributos ou métodos são mapeados, podendo estes ser usados para cálculos ou funções diversas localmente. (AMBLER, 2014). Na figura 3, observa-se um mapeamento de classes simples, enquanto a figura 4 possui um esquema mais complexo contendo um mapeamento de várias classes para uma tabela e uma tabela para várias classes. (CARVALHO, 2011).

Figura 3 – Mapeamento de classe simples.



Fonte: CARVALHO, 2011.

Figura 4 – Mapeamento de classes complexo.



Fonte: CARVALHO, 2011.

Segundo Ambler (2014), podemos ter os seguintes tipos de mapeamentos:

a) One-To-One

Relacionamento um para um, ocorre quando temos um item de memória se relacionando com outro, percorrendo-o automaticamente quando o atributo é lido, por exemplo, um funcionário ocupa um cargo, podendo somente ter um cargo. (AMBLER, 2014).

b) One-To-Many

Relacionamento um para muitos, ocorre quando um item é lido em memória, automaticamente é lido o item seguinte que possui o relacionamento com o primeiro, por exemplo, um funcionário pertence a uma repartição e uma repartição possui muitos funcionários, então quando o item funcionário é lido automaticamente é buscada sua repartição. (AMBLER, 2014).

c) Many-To-Many

Relacionamento muitos para muitos, ocorre quando temos tabelas associativas com o objetivo de manter um relacionamento com duas ou mais tabelas no banco de dados, por exemplo, um funcionário pode ter várias tarefas, deste modo surge a necessidade de uma tabela intermediária que faça a ligação entre tarefas e funcionários. (AMBLER, 2014).

2.5. DOCTRINE ORM

De acordo com Serpa (2012), *Doctrine* é um *framework* com o objetivo de fazer o mapeamento objeto-relacional desenvolvido especificamente para a linguagem PHP. Segundo o site do *Doctrine* (2014), este projeto foi inspirado no *Hibernate* (*framework* de mapeamento usado pela linguagem Java) e possui todo seu código fonte disponibilizado gratuitamente, podendo ser usado por qualquer pessoa.

Doctrine foi escrito por Konsta Vesterinen em 2006, inicialmente a equipe de desenvolvimento era formada por apenas quatro pessoas passando a ganhar contribuintes quando seus códigos passaram a ser abertos. (LEISALU, 2009, p. 11).

Como um dos principais objetivos de um ORM é abstrair a programação lógica do banco de dados, ele deve ser compatível com uma grande variedade de *data bases*. No caso do *Doctrine*, ele suporta oficialmente as principais ferramentas de guarnição de dados, por

exemplo: SQLite, MySQL, Oracle, PostegreSQL e o banco de dados da Microsoft, conhecido por Microsoft SQL Server. (DUNGLAS, 2013, p. 7).

De acordo com Leisalu (2009, p. 19), *Doctrine* se destaca dos outros modelos de mapeamento porque possui a importante função de fazer o processo reverso, ou seja, ele consegue fazer o mapeamento a partir de uma base de dados já criada.

Na figura 5, observa-se um modelo de mapeamento na classe denominada "*Bug*". A classe possui um total de quatro atributos denominados id, *description*, *created* e *status*. (DOCTRINE, 2014).

a) Id: Do tipo *Integer* e gera seus valores de forma automática;

b) *Description*: Do tipo *String*, ou seja, guardará uma sequência de caracteres no banco;

c) *Created:* Do tipo *DateTime*, irá guardar dados como data e hora;

d) *Status:* Do tipo *String* e também guardará uma sequência de caracteres.

Figura 5 – Exemplo de mapeamento em classe.

```php
<?php
// src/Bug.php
/**
 * @Entity(repositoryClass="BugRepository") @Table(name="bugs")
 */
class Bug
{
    /**
     * @Id @Column(type="integer") @GeneratedValue
     * @var int
     */
    protected $id;
    /**
     * @Column(type="string")
     * @var string
     */
    protected $description;
    /**
     * @Column(type="datetime")
     * @var DateTime
     */
    protected $created;
    /**
     * @Column(type="string")
     * @var string
     */
    protected $status;
```

Fonte: DOCTRINE, 2014.

Já a figura 6 demonstra um exemplo de como seria o controle deste mapeamento. Destacam-se os métodos *find*, *persist* e *flush*, os quais respectivamente localizam as chaves necessárias para a inserção no banco, fazem a persistência na base de dados e por último limpa a memória utilizada. (DOCTRINE, 2014).

Figura 6 – Exemplo de persistência com *Doctrine*.

```php
<?php
// create_bug.php
require_once "bootstrap.php";

$theReporterId = $argv[1];
$theDefaultEngineerId = $argv[2];
$productIds = explode(",", $argv[3]);

$reporter = $entityManager->find("User", $theReporterId);
$engineer = $entityManager->find("User", $theDefaultEngineerId);
if (!$reporter || !$engineer) {
    echo "No reporter and/or engineer found for the input.\n";
    exit(1);
}

$bug = new Bug();
$bug->setDescription("Something does not work!");
$bug->setCreated(new DateTime("now"));
$bug->setStatus("OPEN");

foreach ($productIds as $productId) {
    $product = $entityManager->find("Product", $productId);
    $bug->assignToProduct($product);
}

$bug->setReporter($reporter);
$bug->setEngineer($engineer);

$entityManager->persist($bug);
$entityManager->flush();

echo "Your new Bug Id: ".$bug->getId()."\n";
```

Fonte: DOCTRINE, 2014.

3. SOFTWARES ESTUDADOS

Alguns softwares foram estudados para melhor entendimento do assunto, por ser robusto e conter um grande número de funcionalidades o foco de análise foi direcionado ao programa DietWin versão do ano de 2011. Ao serem analisados pode-se destacar como pontos positivos funcionalidades completas que auxiliam os profissionais da área, ampla base de dados de alimentos pré-cadastrados e rapidez ao executar ações.

Já como pontos negativos observa-se uma complexidade para obter dados ou resultados rápidos, levando um usuário leigo ter uma linha de aprendizado maior do que o esperado, ou seja, softwares pouco intuitivos, interface gráfica poluída com muitas informações e poucos caminhos para chegar a funcionalidade desejada, forçando o usuário a navegar desnecessariamente entre telas.

4. NUTRILAB

Nutrilab é o nome do *software* obtido como resultado final deste trabalho de conclusão. Seu protótipo foi desenvolvido seguindo as orientações de produção do *REST* com o objetivo de, futuramente, caso exista a necessidade de integrá-lo com outras plataformas, assim fazê-lo e logicamente aproveitando todos os seus recursos disponíveis. O sistema deve

dar apoio às necessidades básicas de um nutricionista, como, por exemplo, cadastro de clientes, controle de dietas, uma base de dados pré-pronta com grupos alimentares, etc.

Como o sistema será online, vários profissionais poderão ter acessos nele, cada um com seus clientes e controles. O banco de dados optado é o Postgresql por ser mais robusto em relação às outras opções *open source*. Juntando a ferramenta *Doctrine* ao projeto, deve-se obter melhor desempenho durante a produtividade, além de deixar o código mais elegante e de fácil interpretação.

Em resumo, a intenção é obter como produto final, um programa rápido, eficiente e com fácil usabilidade. Pode-se destacar como funcionalidades:

- Cadastro de novos alimentos e grupos alimentares, além dos existentes;
- Ficha completa referente aos dados antropométricos do paciente;
- Criação de dietas personalizadas por paciente;
- Cadastros de usuário com diferentes níveis de acesso ao sistema;
- Agendamentos de pacientes;
- Armazenamento de exames e informações patológicas de acordo com o paciente;
- Informações alimentares e hábitos referentes a cada paciente;

4.1. REQUISITOS DO SISTEMA

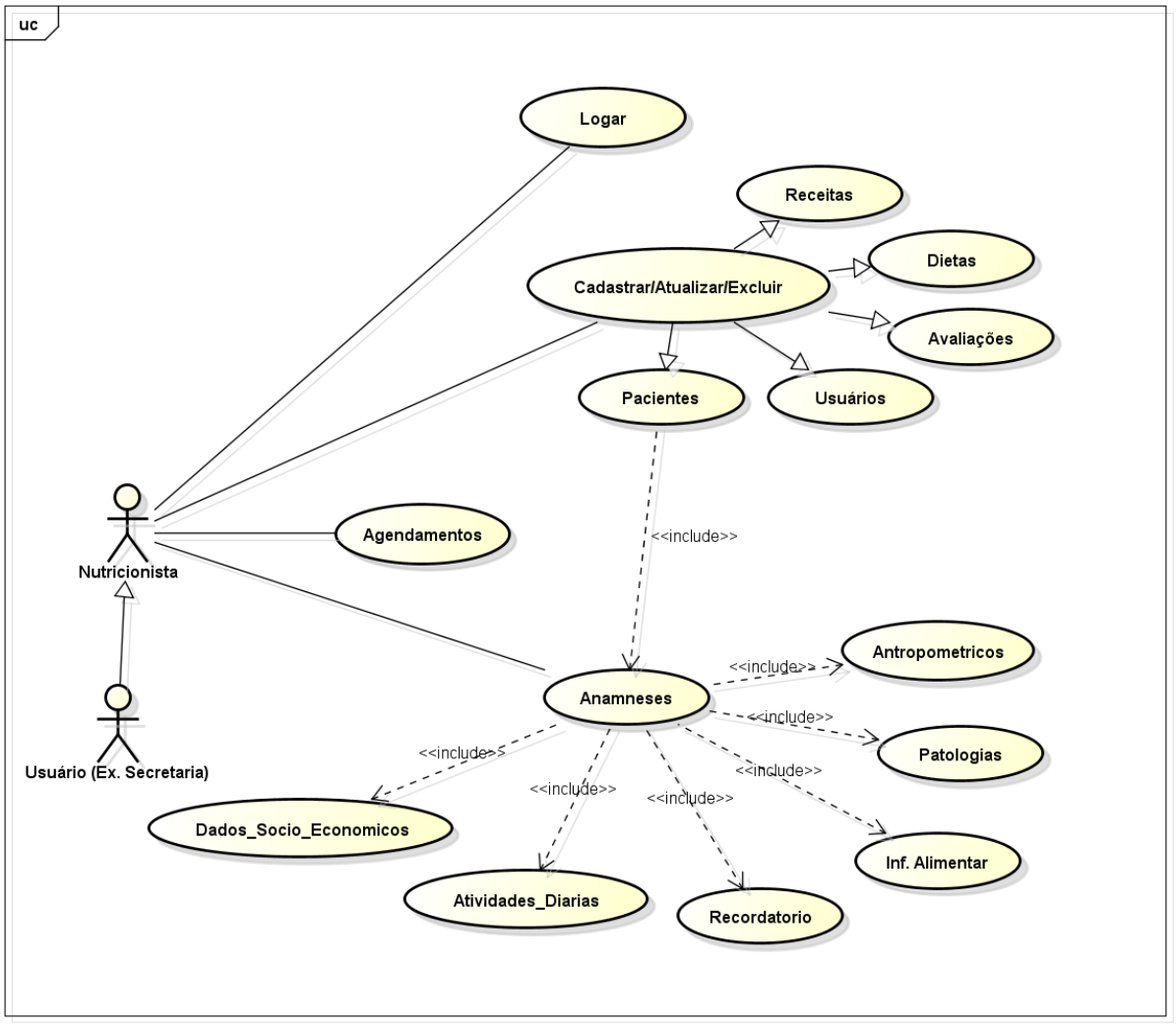
Tabela 1 - Requisitos do sistema.

F1 - Cadastro de Alimentos e Grupos Alimentares			Oculto ( )	
Descrição: O sistema deve possibilitar a inclusão de novos alimentos e grupos alimentares além dos já existentes na base de dados.				
Requisitos Não Funcionais				
Nome	Restrição	Categoria	Desejável	Permanente
NF1.1 – Controle de Acesso	A função só pode ser acessada por uma nutricionista.	Segurança	()	(X)
F2 - Cadastro de Anamnese do Paciente			Oculto ( )	
Descrição: O sistema deve possibilitar o cadastro de uma anamnese completa do paciente, incluindo dados pessoais, físicos, alimentícios e sócio econômicos.				
Nome	Restrição	Categoria	Desejável	Permanente
NF2.1 – Controle de Acesso	A função só pode ser acessada por uma Nutricionista.	Segurança	()	(X)
NF2.2 – Telas de Cadastro	As telas de cadastro devem ser separadas por suas respectivas categorias.	Interface	()	(X)
NF2.3 – Tempo de Registro	O tempo de espera para cadastrar todas as informações não pode ser superior a 3seg., contando a partir do clique no botão salvar.	Performance	(X)	()

F3 – Calcular Medidas			Oculto (X)	
Descrição: O sistema deve calcular as medidas do paciente de acordo com os dados cadastrados previamente pela nutricionista ao abrir a tela de visualização de anameses.				
Requisitos Não Funcionais				
Nome	Restrição	Categoria	Desejável	Permanente
NF3.1 – Controle de Acesso	A função só pode ser acessada por uma nutricionista.	Segurança	()	(X)
NF3.2 – Tempo de Consulta	O tempo de espera ao consultar dados deve ser no máximo 3seg.	Performance	(X)	()
F4 – Agendar Paciente			Oculto ()	
Descrição: O sistema deve permitir o agendamento de pacientes, retornando uma lista ordenada por proximidade de dia e horário.				
Requisitos Não Funcionais				
Nome	Restrição	Categoria	Desejável	Permanente
NF4.1 – Controle de Acesso	A função pode ser acessada por nutricionistas e usuários que possuam acessos para esta funcionalidade	Segurança	()	(X)
NF4.2 – Proximidade de datas	As datas mais próximas devem possuir um destaque na tela de apresentação da agenda.	Interface	(X)	()
F5 – Cadastro de Novos Usuários			Oculto ()	
Descrição: O sistema deve permitir o cadastro de novos usuários.				
Requisitos Não Funcionais				
Nome	Restrição	Categoria	Desejável	Permanente
NF5.1 – Controle de Acesso	A função só pode ser acessada por uma nutricionista.	Segurança	()	(X)
NF5.2 – Nível de usuário	O sistema deve conter níveis de usuário para acesso as funcionalidades.	Segurança	()	(X)
F6 – *Login* no Sistema			Oculto ()	
Descrição: O sistema deve realizar *login* através de pergunta e frases secretas				
Requisitos Não Funcionais				
Nome	Restrição	Categoria	Desejável	Permanente
NF5.1 – Acesso ao sistema	O acesso só pode ser realizado com pergunta e frases secretas previamente cadastradas no momento do registro da conta.	Segurança	()	(X)

Abaixo observa-se o modelo de casos de uso juntamente com sua documentação e o diagrama de classes do projeto:

Figura 7 - Modelo Casos de Uso



Fonte: Do autor.

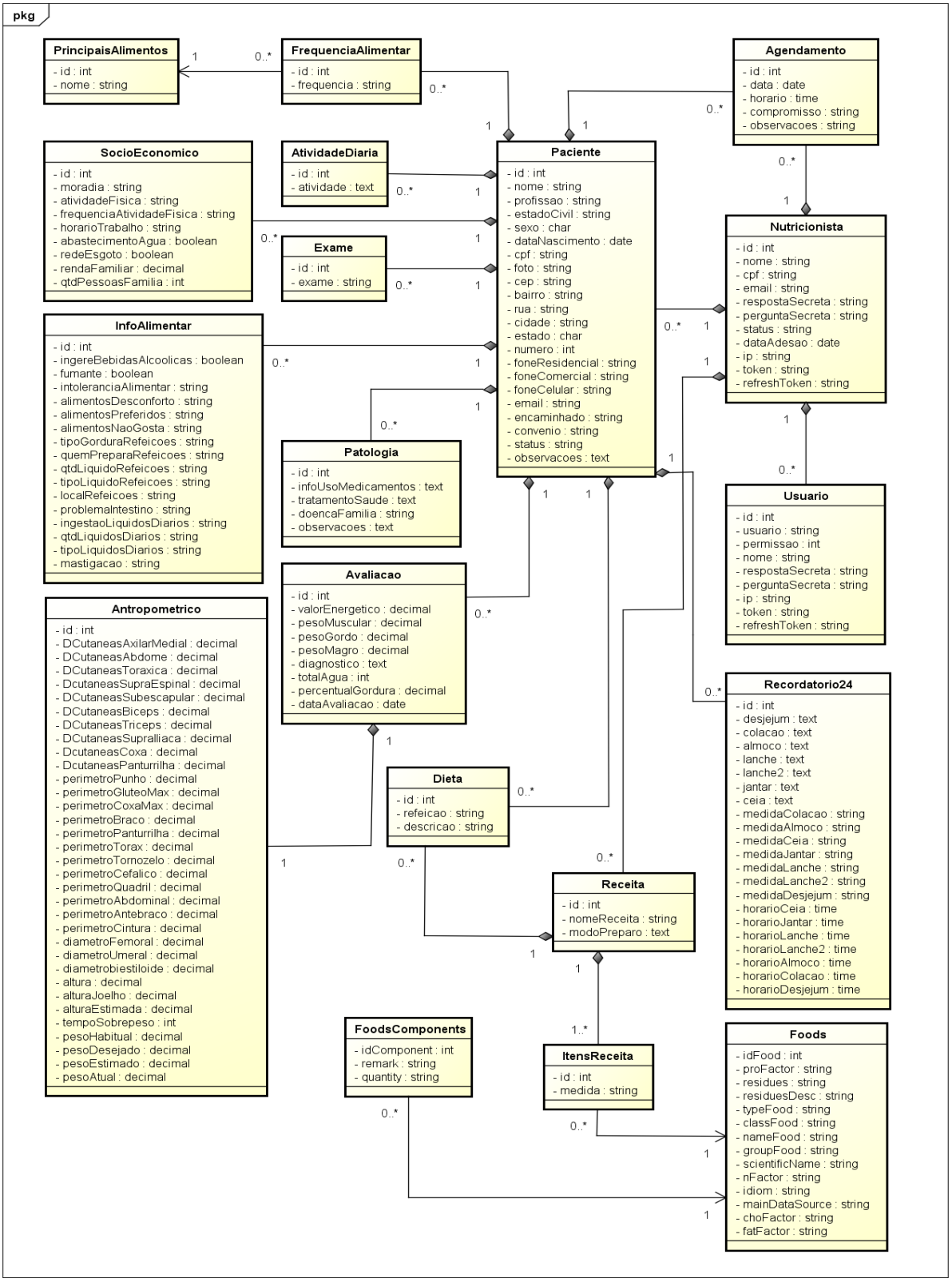
Tabela 2 - Documentação Casos de Uso

Documentação		
Atores		
Ator Principal	Nutricionista	
Ator Secundário	Usuário Comum (Secretária)	
Nome do Caso de Uso	**Descrição**	**Pré-condição**
Acessar o Sistema	O ator principal deverá entrar com cpf, pergunta secreta e frase secreta para acessar o sistema. O ator secundário deverá entrar com e-mail, pergunta secreta e frase secreta para acessar o sistema.	Possuir um cadastro no sistema.

Cadastrar/Atualizar/Excluir Receitas	Os atores poderão cadastrar, atualizar ou excluir receitas dentro do sistema.	Logado no sistema.
Cadastrar/Atualizar/Excluir Dietas	Os atores poderão cadastrar, atualizar ou excluir dietas dentro do sistema. Para poder cadastrar uma nova dieta é necessário existir pelo menos uma receita cadastrada.	Logado no sistema.
Cadastrar/Atualizar/Excluir Avaliações	Os atores poderão cadastrar, atualizar ou excluir avaliações de cada um de seus pacientes.	Logado no sistema.
Cadastrar/Atualizar/Excluir Usuários	Apenas o ator principal poderá cadastrar, atualizar ou excluir novos usuários do sistema.	Logado no sistema.
Cadastrar/Atualizar/Excluir Pacientes	Os atores poderão realizar o cadastro de novos pacientes no sistema para controlar seu estado clinico. Também podem realizar atualizações em seus cadastros ou exclusão.	Logado no sistema.
Cadastrar/Atualizar/Excluir Agendamentos	Os atores agendarão novas consultas além de realizar atualização e exclusão das mesmas.	Logado no sistema.
Anamneses		
Antropométricos	O ator cadastrará os dados físicos do paciente, como por exemplo, altura do joelho, circunferência da cintura.	Logado no sistema.
Patologias	O ator poderá ter um histórico das patologias do paciente.	Logado no sistema.
Inf. Alimentar	O ator terá um cadastro das informações alimentares de cada paciente.	Logado no sistema.
Recordatório	O ator cadastrará um recordatório de 24 horas referente os alimentos ingeridos no dia anterior.	Logado no sistema.
Atividades_Diarias	O ator cadastrará rotina de atividades dos pacientes.	Logado no sistema.
Dados Sócio Econômicos	O ator coletará dados sociais do paciente para poder passar uma receita de acordo com as condições econômicas do mesmo.	Logado no sistema.

Fonte: Do autor.

Figura 8 - Diagrama de Classes



Fonte: Do autor.

O diagrama de classes demonstra como será criada as tabelas na base de dados utilizando a ferramenta Doctrine. A partir deste diagrama é possível realizar os mapeamentos e suas notações dentro das classes correspondentes. Pode-se começar observando a tabela

nutricionistas, a qual vai armazenar diferentes tipos de nutricionistas, o que por usa vez possui uma ligação com a tabela usuários que tem por objetivo guardar os usuários de cada nutricionista em especifico, por exemplo, a nutricionista denominada “A” pode ter os usuários “X” e “F”, já a nutricionista denominada “B” possui os usuários “H” e “K”. O mesmo acontece com a tabela pacientes, ou seja, cada um destes pertence a uma nutricionista em especifico. E por fim, existe ainda uma tabela de agendamentos de horários com os pacientes de cada nutricionista.

No caso de uma nutricionista deixar de existir no sistema, tudo ligado a ela também é eliminado, como indicado na associação por composição, já que não faz sentido manter dados de uma nutricionista se a mesma foi excluída.

Observando mais atentamente a tabela pacientes, pode-se observar várias associações com outras tabelas no sentido de composição. Isso se deve ao fato de cada paciente possuir uma ficha chamada de anamnese alimentar. Nesta consta todos os tipos de dados da pessoa atendida, ou seja, dados sócios econômicos, tipo de alimentos consumidos no último dia, medidas corporais, informações alimentares, como por exemplo, alergias e alimentos preferidos e ainda a possibilidade da nutricionista associar exames a este paciente. Com estes dados é possível aplicar fórmulas dentro do sistema, para que de forma automatizada gere métricas e percentuais relacionado ao estado da composição corporal de cada paciente. Um exemplo de fórmula seria o Índice de Massa Corporal (IMC). Como citado anteriormente, se o paciente for excluído do sistema, não faz sentido manter seus dados armazenados, desta forma, a associação por composição nos indica que ao ser eliminado seus dados também devem ser descartados.

Em especial, o diagrama possui quatro tabelas que já possuem dados pré-cadastrados, estas tabelas são “foods” e “foodsComponents” além de “frequenciaAlimentar” e “principaisAlimentos”. Os relacionamentos são relativamente simples com a exceção da ligação entre as tabelas “receitas”, “itensReceita”, “dietas”, “foods” e “pacientes”. Em uma breve explicação, basicamente para a nutricionista poder indicar alimentos de uma dieta ao paciente, por exemplo, de um café da manhã, é necessário que ela tenha uma receita cadastrada primeiramente. Esta receita então, possuirá diversos itens, como por exemplo, café, açúcar, leite, pão, etc. Estes itens surgem da tabela “foods”, este sendo então um caso mais complexo de se trabalhar diferenciado dos demais.

5. INSTALAÇÕES DAS FERRAMENTAS

Neste trabalho foi utilizado o sistema operacional Debian, uma distribuição do GNU/Linux, junto com a IDE de desenvolvimento Netbeans. Base de dados Postgresql e PHP na versão 5.4 ou superior mas também demonstra como realizar os procedimentos no sistema operacional Windows.

5.1. INSTALAÇÃO COMPOSER

A instalação e configuração do *Composer* pode ser realizada seguindo dois caminhos: localmente ou globalmente. Localmente será usado somente em um projeto e globalmente pode ser usado em todos os projetos. (Composer, 2015). Abaixo segue um passo a passo de como realizar sua instalação de forma global:

- Fazer o *download* do arquivo *composer.phar*. Este arquivo pode ser encontrado no próprio site do desenvolvedor;
- Mover o arquivo para "/usr/local/bin/composer" para torna-lo global dentro do sistema operacional, é opcional manter a extensão ".phar" ao realizar a troca de diretório. (Composer, 2015);
- No *Netbeans*, localizar o botão de menu chamado "ferramentas", acessar o submenu chamado "opções" e selecionar a opção "*Composer*". Após isto deve-se clicar no botão "pesquisar", desta forma a própria IDE vai localizar o arquivo necessário para começar a utilização o *Composer*.

Um detalhe que não deve ser esquecido é o de informar a IDE um compilador PHP. Normalmente, apenas selecionando a opção "pesquisar" dentro do menu ferramentas>opções>compilador, deve ser suficiente para localiza-lo. Caso não consiga encontrar automaticamente pode-se tentar localiza-lo em "usr/bin/php". Se foi utilizado o pacote web inteiro que vem ao instalar o *Postgresql* então o compilador deverá ser encontrado em "/opt/PostgreSQL/EnterpriseDB-ApachePHP/php/bin/php", lembrando que ele é um arquivo executável e estes são caminhos de instalação padrão dos programas.

No caso de usar o sistema *Windows*, deve-se seguir os mesmos procedimentos acima com alguns detalhes diferentes. São eles:

- Ao invés de fazer o *download* do arquivo "*composer.phar*", deverá ser realizado o *download* do arquivo "*Composer-Setup*", este arquivo será responsável por realizar a instalação do C*omposer* dentro do sistema. (Composer, 2015);

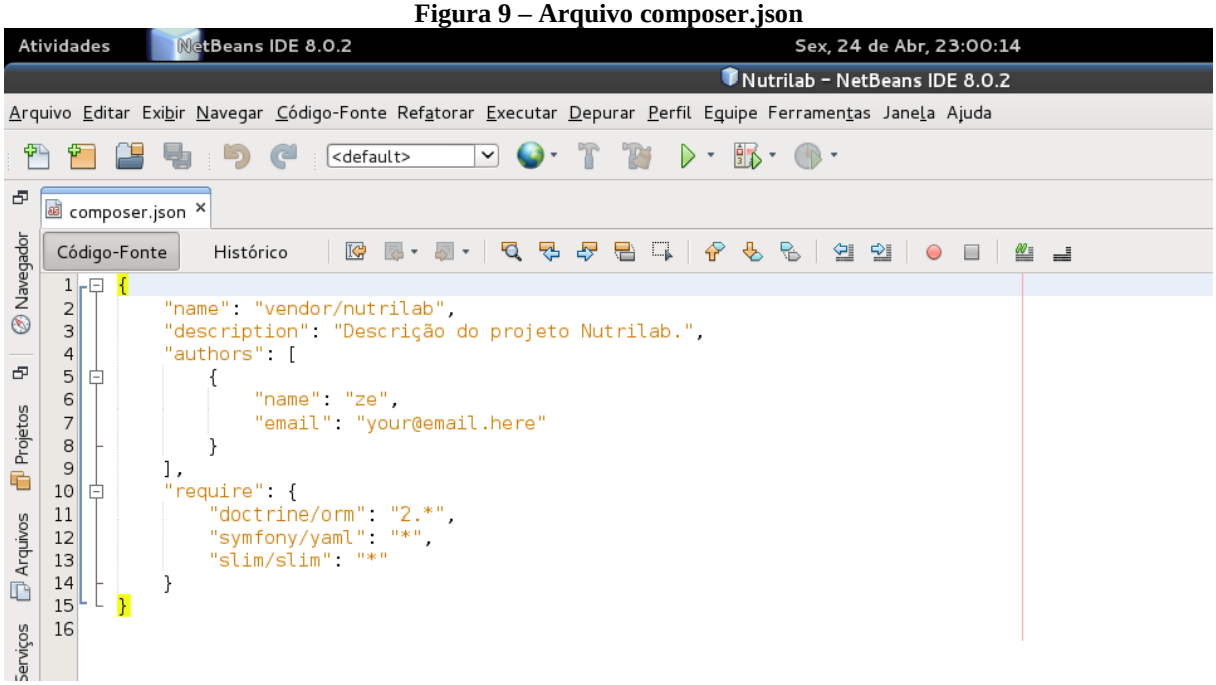
- O caminho do compilador php vai depender de qual servidor está rodando, neste caso será dado como exemplo o do Xampp. O Caminho é “C:\xampp\php\php.exe”.

Depois destes procedimentos o *Composer* está pronto para ser usado. Isto pode acontecer através de linhas do comando abrindo o *prompt* no *Windows* ou o terminal no Debian.

5.2. INSTALAÇÃO DOCTRINE

Com o *Composer* instalado, pode-se gerenciar nossas bibliotecas facilmente. Para fazer o download da ferramenta *Doctrine* deve-se abrir o arquivo “*composer.json*” e escrever dentro do espaço “*require*” o nome da biblioteca à ser feito o download e sua versão. (Doctrine, 2015). Normalmente o site do desenvolvedor fornece o nome correto para usar com o *Composer*. Quanto a versão existe uma política de versionamento a ser seguida, como o projeto usa somente versões estáveis então utilizaremos o caractere “*” que indica qualquer versão mais recente estável. Veja a figura 9 para visualizar como ficou o arquivo “*composer.json*”:

Figura 9 – Arquivo composer.json



Fonte: Do autor.

Observa-se no arquivo ainda mais uma biblioteca chamada *symfony/yaml*. Está biblioteca de arquivos na verdade é um framework de desenvolvimento que tem por objetivo organizar grandes projetos. Em nosso caso ele é uma dependência do *Doctrine*. Não é necessário colocar seu nome no arquivo de gerenciamento do *Composer* já que ele

automaticamente sabe as dependências e irá baixa-las sozinho mas neste caso especificamos a qual será utilizada no projeto.

5.3. INSTALAÇÃO SLIM

Slim é o framework que auxilia no trabalho com o REST, abstraindo grande parte da complexidade de trabalhos com os protocolos HTTP. Como pode ser visto na imagem anterior, seu download e instalação também é feito pelo *Composer* utilizando o nome slim/slim. (*Slim*, 2015).

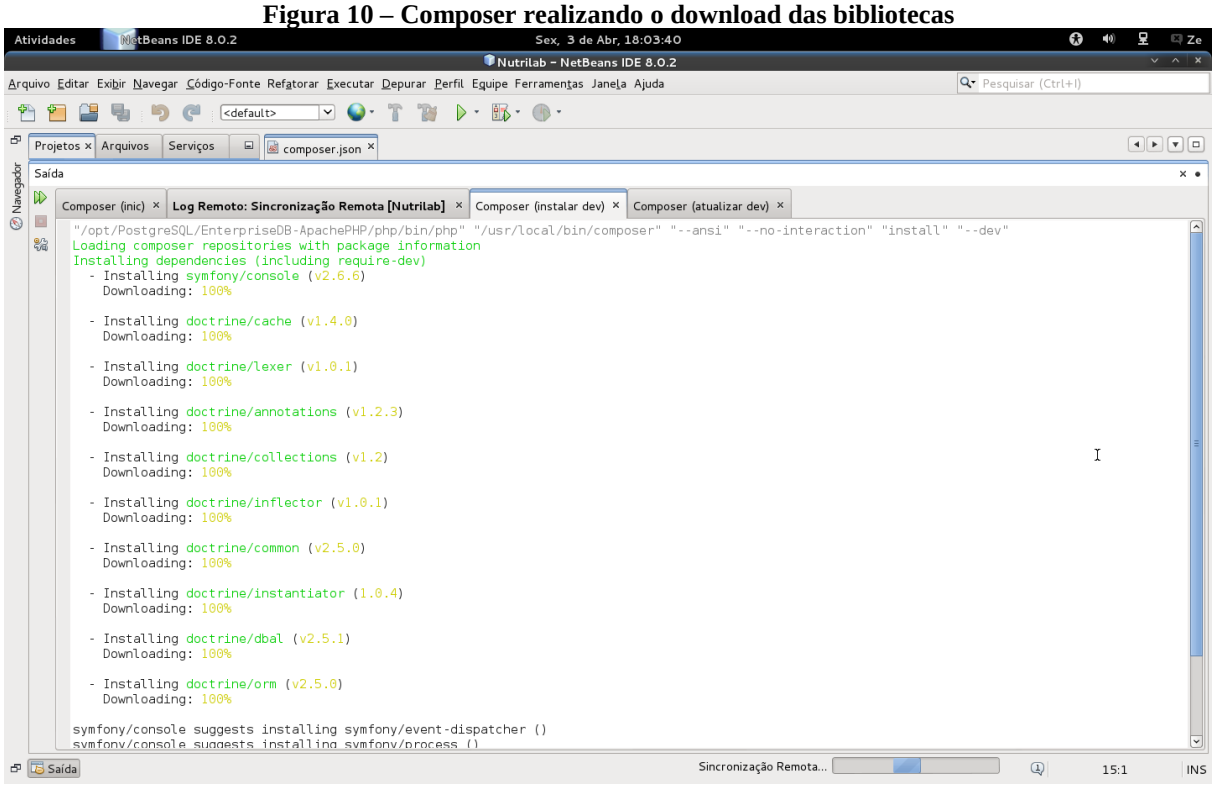
5.4. INSTALAÇÕES FINAIS

Após ter o nome e versão dos componentes necessários para o projeto no arquivo de configuração do *Composer*, para realizar seus *downloads* através da IDE basta selecionar o projeto com o botão secundário do mouse, localizar o menu *Composer* e logo após o item *Init*. Caso necessite do uso de outra biblioteca ou realizar a atualização de alguma já em uso, somente altere o arquivo *composer.json*, e realize o procedimento acima mas com a diferença de ao invés de selecionar *Init*, selecione *update*.

Desta forma todas as bibliotecas e suas dependências estão prontas para serem usadas no projeto. Um último detalhe mas não menos importante é o arquivo "*autoload.php*" localizado na pasta "*vendor*". Este arquivo deve ser incluído nos scripts que necessitarão de alguma das bibliotecas que o *Compose*r fez o *download*.

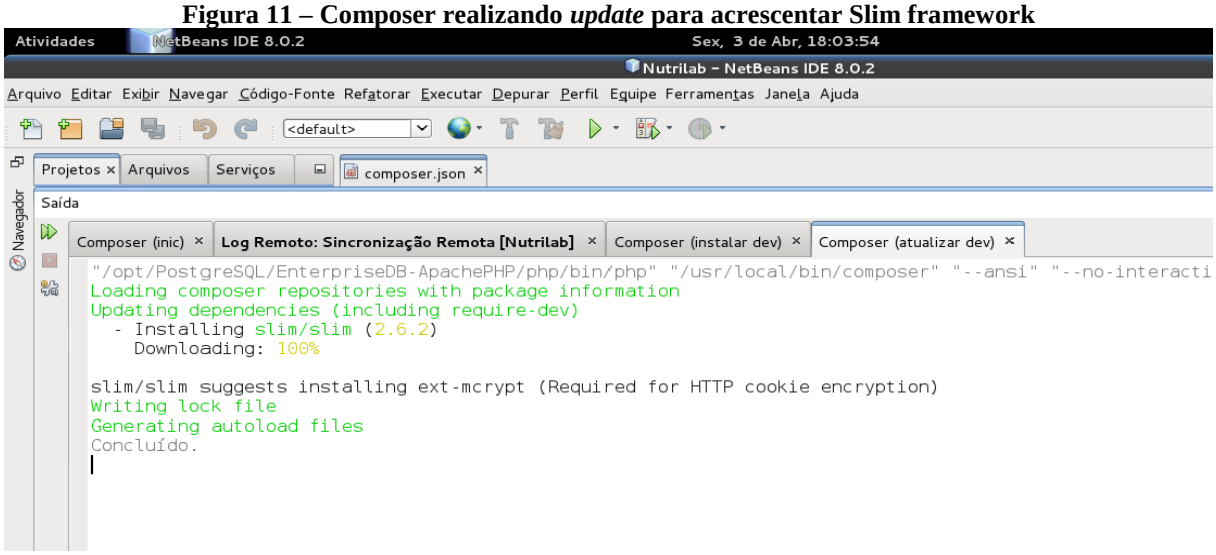
Abaixo, algumas imagens do *Composer*:

Figura 10 – Composer realizando o download das bibliotecas



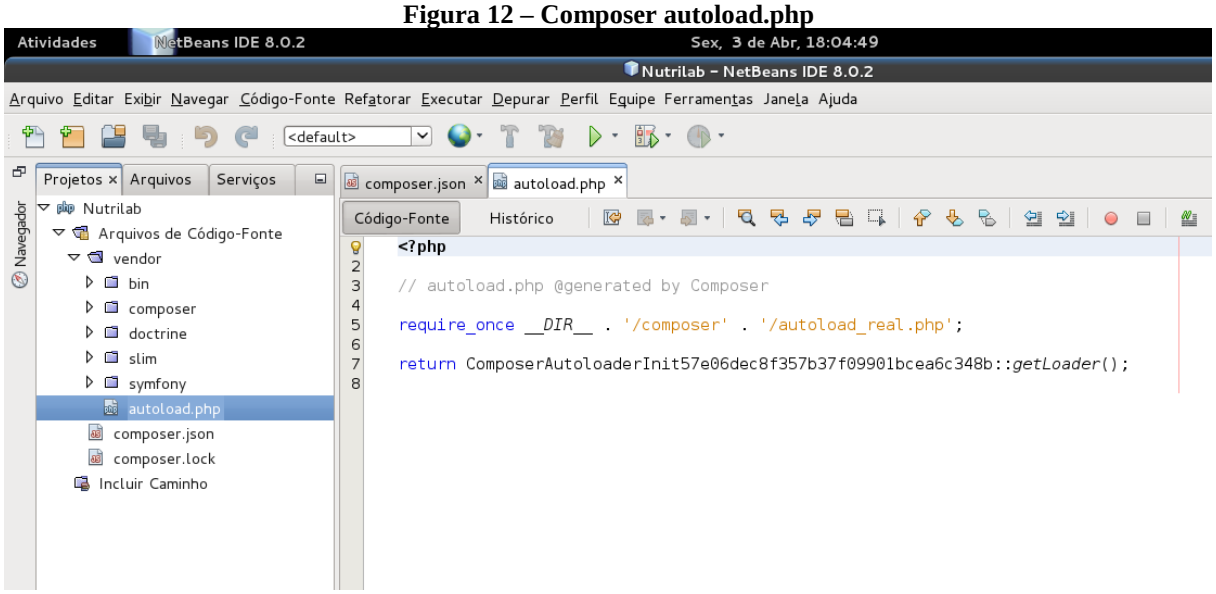
Fonte: Do autor.

Figura 11 – Composer realizando *update* para acrescentar Slim framework



Fonte: Do autor.

Figura 12 – Composer autoload.php



Fonte: Do autor.

6. IMPLEMENTAÇÃO DO SISTEMA

Este tópico descreverá como foi realizada a implementação do sistema, dificuldades e soluções obtidas no uso das ferramentas além de exemplos didáticos para o uso das mesmas.

6.1. IMPLEMENTAÇÃO *DOCTRINE*

O objetivo do *Doctrine* dentro desse projeto é agilizar o desenvolvimento evitando que o programador tenha contato com códigos SQL, trabalhando somente nas classes do projeto. Dessa forma esse ORM foi empregado nas classes modelo do *software* utilizando o conceito de mapeamentos com notações. Observe o trecho de código retirado da classe "Agendamentos" logo abaixo:

Figura 13 - Trecho de mapeamento em classe

```
/**
 * @Column(type="string", name="compromisso", nullable=false)
 */
private $compromisso;

/**
 * @Column(type="time", name="horario", nullable=false)
 */
private $horario;

/**
 * @ManyToOne(targetEntity="Nutricionista")
 * @JoinColumn(name="nutricionista", referencedColumnName="id", nullable=false)
 */
private $nutricionista;
```

Fonte: Do autor.

Observa-se na figura 13, que é passado para o *Doctrine* (em forma de comentários) as características dos atributos. Neste trecho, o atributo compromisso será no banco de dados uma coluna que guardará dados do tipo *string*, ou seja *varchar*. Já o atributo horário será do tipo *time*, armazenando um determinado tempo com horas, minutos e segundos. Já o atributo nutricionista é um pouco diferente, pois ele é uma chave estrangeira (*foreign key*) dentro da tabela agendamentos. Neste caso devemos indicar de qual entidade é esta chave, qual será o nome da coluna que guardará as chaves e logicamente qual o campo desejamos que ele busque a chave na entidade Nutricionista.

6.2. IMPLEMENTAÇÃO *RESTFULL*

Como o *REST* é um modelo arquitetônico, deve-se escolher um tipo de implementação a ser usada. Nesse projeto optou-se pelo modelo *RESTfull*, com a criação de *web services* para cada funcionalidade do programa. Desta forma o projeto ganha um nível de separação muito grande entre o *front-end* e o *back-end* além de ser possível que o sistema seja entregue em módulos separados, dividido e alterado sem influenciar outras partes especificas desde que não existe dependência entre módulos. A figura 14 demonstra um exemplo de como pode ser usado o *REST* no lado do servidor tendo como auxiliar o *Slim Framework*:

Figura 14 - *Web service REST*

```
Slim\Slim::registerAutoloader();

$app = new \Slim\Slim();
$app->response()->header('Content-Type', 'application/json; charset=utf-8');

$app->get('/','getRaiz');

$app->put('/logar/','logar');
$app->put('/logout/:token','logout');

$app->run();

function getRaiz(){
    echo 'Bem vindo ao service Login!!';
}

function logar(){
}

function logout($token){
}
```

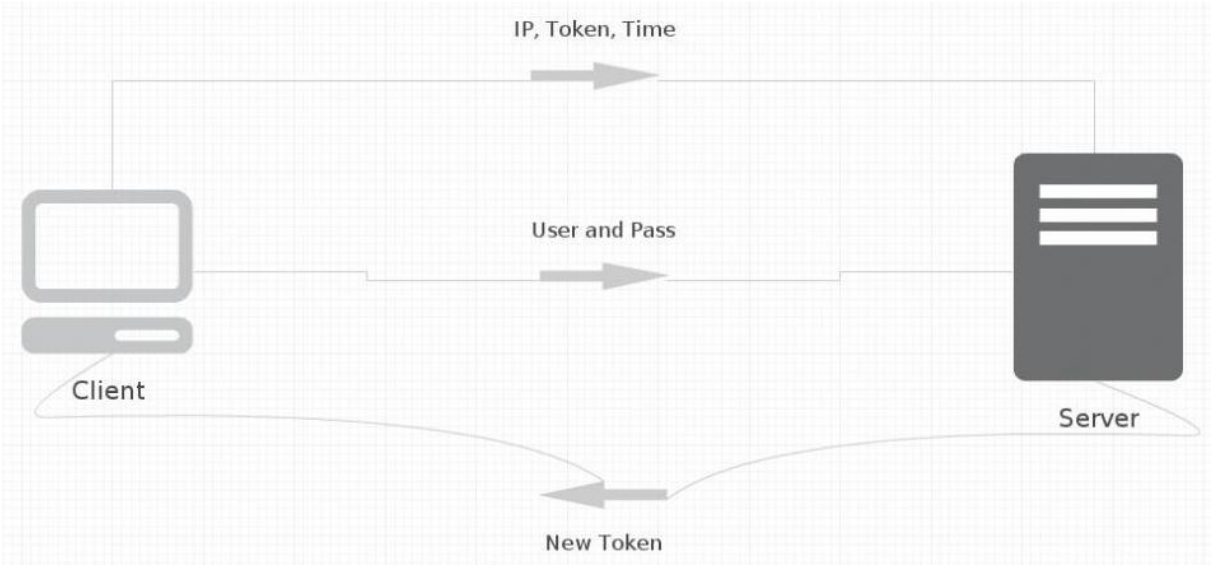
Fonte: Do autor.

No caso desse *web service* pode-se observar que é trabalhado com *Json* como forma de comunicação entre cliente e servidor. É utilizado apenas dois métodos do HTTP que são o *put e o get*. Dentro de cada um deste métodos o *Slim* irá tratar tais requisições. Os parâmetros recebidos por ele são: a URL requisitada e qual função deve ser chamada ao ser feito uma requisição ao determinado endereço.

6.2.1. SEGURANÇA COM REST

A utilização da implementação *RESTfull* possui uma falha de segurança, qualquer pessoa que saiba a URI do serviço solicitado poderia realizar uma requisição e obter os dados daquele serviço, como por exemplo, incluir, alterar ou até mesmo excluir dados de uma base de armazenamento. A solução encontrada para este problema foi a utilização de HTTPS (unanimidade) por parte dos desenvolvedores e implementação do esquema de segurança interconectado com servidores mais conhecidos como Facebook, Google, etc., chamado de "O Auth 2". Também existe a alternativa de substituir o "O Auth 2" por uma implementação própria de segurança. (InfoQ, 2015). No caso desse projeto foi optado por uma solução própria que segue no diagrama abaixo:

Figura 15 - Diagrama de segurança



Fonte: Do autor.

Compreendendo o diagrama supondo que usuário e senha estejam corretos:

- Usuário inicia comunicação com o servidor enviando usuário e senha;
- Servidor envia como resposta um *token* alfanumérico e armazena no cliente;
- A cada requisição do cliente, é enviado para o servidor o *token* juntamente com o endereço IP e a hora de acesso (opcional);

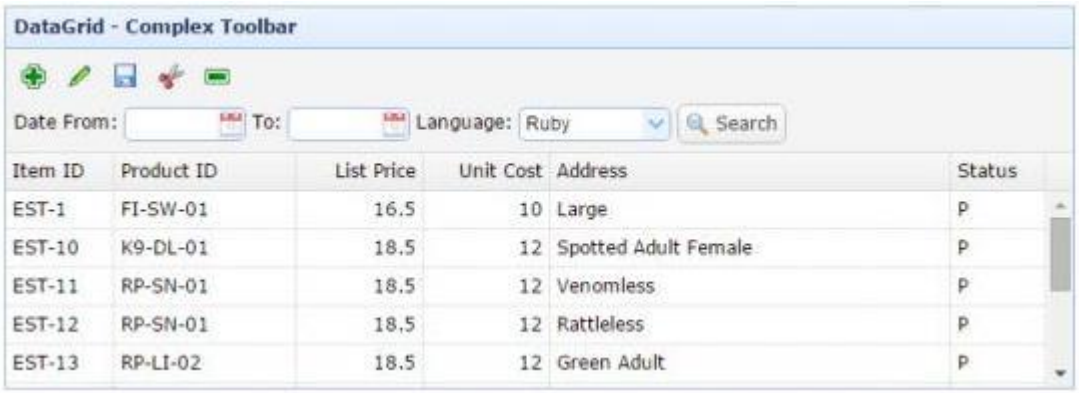
- Servidor verifica se o *token* do cliente é o mesmo gerado anteriormente e se o endereço IP coincide com o enviado ao primeiro acesso, se coincidirem, o usuário pode realizar a ação;
- Servidor então descarta *token* antigo e gera um novo, registrando ele na máquina do cliente;
- Por fim, ao sair do sistema, todos os *token* e dados como o IP são apagados, tanto do servidor quanto do cliente, reiniciando o ciclo em um novo acesso.

E no caso de dados incorretos, o servidor não autoriza a execução do comando e deleta todos os dados de acesso do cliente e no próprio servidor. Mesmo com esse método de segurança os dados ainda podem ser interceptados e adulterados, por isso a necessidade de uso do HTTPS.

6.3. INTERFACE GRÁFICA/CLIENTE

A interface de comunicação escolhida foi o *framework* chamado *jQuery EasyUi* devido ao seu modo de trabalho. Sendo composto por todos os elementos HTML além de um *layout* pré-programado nos padrões norte, sul, leste, oeste e centro. Seu uso fica ainda mais fácil devido ao fato de trabalhar somente com objetos *Json* para alimentar as interfaces e realizar comunicação com o servidor. Veja um exemplo de listagem deste *framework* na figura 15:

Figura 16 - Data Grid jQuery EasyUI



Fonte: jQuery EasyUi, 2015.

De código fonte aberto, este *framework* surgiu no ano de 2009 desenvolvido pela equipe do *jQuery EasyUi*. Tem por características usar HTML 5, ajax e permitir que o desenvolvedor trabalhe tanto na parte HTML ou Java Script para desenvolver os projetos. Existe a versão para projetos mobile apesar de conter menos funcionalidades do que a verão para Desktop. (*jQuery EasyUi*, 2015).

7. METODOLOGIA

O desenvolvimento do *software* foi realizado com embasamentos em pesquisas bibliográficas referente às tecnologias usadas. Também foi realizado um estudo prático de implementação da ferramenta *Doctrine* para obtenção de experiência com o uso da mesma, realizando pequenos mapeamentos e persistindo algumas classes em um grau de dificuldade simples.

Para trabalhar com *REST*, está sendo usado o *framework* chamado *Slim* (*Slim*, 2015) que ajudará na implementação do modelo *RESTfull*, facilitando o trabalho ao usar os métodos do protocolo HTTP.

Determinou-se a estruturação da árvore de arquivos, definindo como e onde estão localizados os diretórios bem como suas permissões de acessos.

8. CONSIDERAÇÕES FINAIS

As maiores dificuldades ao trabalhar com o *Doctrine* foi o pouco tempo para estudar a tecnologia e a falta de material de qualidade. Até mesmo a documentação do site oficial deixa a desejar. O pouco conteúdo encontrado está em língua inglesa. *Doctrine* é um sistema complexo, existe inúmeras maneiras de otimizar suas buscas, como por exemplo, sistema de cache. Alguns fóruns confundem versões anteriores da 2.x com a atual, sendo que são totalmente diferentes.

Ao contrário do *Doctrine*, *REST* possui um vasto material de excelente conteúdo. Um cuidado que deve-se ter e que pode ser uma dificuldade são modelos de implementações errôneas do modelo *RESTfull*. Este modelo de implementação segue fielmente os princípios do *REST* e muitas vezes, por facilidade de implementação desenvolvedores não o seguem, implementando variações do *RESTfull*. A maior dificuldade ao implementar *RESTfull* é quanto à segurança de trafego de dados e proteção dos *web services*.

O estudo destas duas ferramentas (*Doctrine* e *REST*) é essencial para profissionais que pretendem se manter atualizados no mercado de trabalho. Não só elas mas toda a bagagem necessária para domina-las traz um grande aprendizado, podemos citar o exemplo do gerenciador de dependências *Composer* necessário e de grande utilidade em projetos grandes. Mesmo não dominando completamente (menos de 8 meses de estudos intenso acredito que não se domine o *Doctrine*), o esforço no processo de desenvolvimento da base de

dados foi no mínimo 40% menor, incluindo as alterações necessárias durante o projeto. Isso demonstra o potencial desta ferramenta.

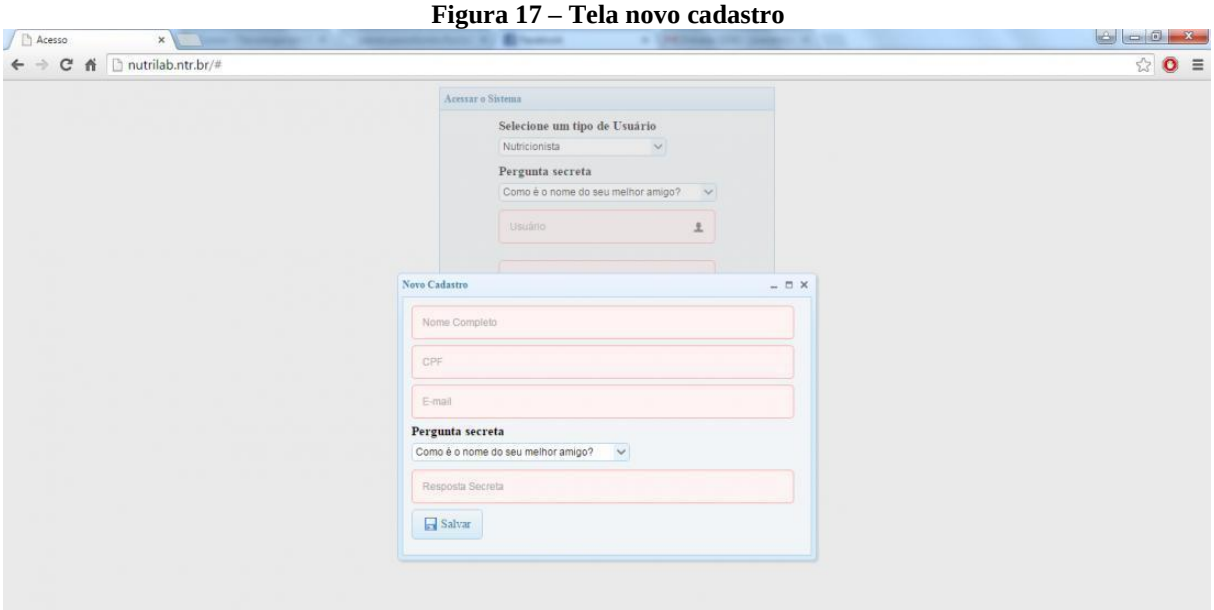
Já ao lado do *REST* as requisições ocorrem de forma mais rápida, já que as informações são apenas cadeias de caracteres transmitidas entre cliente e servidor.

O objetivo deste projeto foi alcançado, não só em conseguir finalizar o projeto mas o aprendizado adquirido ao desenvolve-lo. Pretende-se aperfeiçoá-lo para novas funcionalidades como cadastros online e liberação em tempo real de uso do sistema em questão, integração de meios de pagamentos para utilizá-lo e a otimização de código fonte.

A indicação destas duas ferramentas é para profissionais que necessitam desenvolver projetos de grande e pequeno porte de forma rápida, inovadora e organizada. A única contra indicação do uso de *REST* é em aplicações que transportam dados com cargas de valores muito baixo, por exemplo, dados binários. Para isso existem outras ferramentas especificas.

Abaixo algumas telas do sistema:

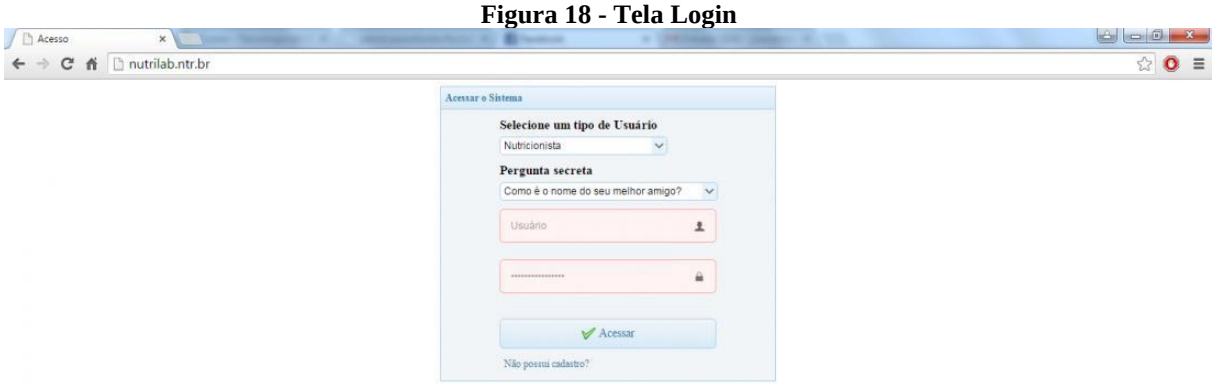
Figura 17 – Tela novo cadastro



Fonte: Do autor.

Na figura 17 observa-se a tela de um novo cadastro. Este cadastro é efetuado por uma nutricionista, que por sua vez, pode cadastrar novos usuários para a sua conta quando acessar o sistema. Neste exemplo, é utilizado o conceito de *passfrase* ao invés de *password*, tornando o sistema um pouco mais seguro que o convencional.

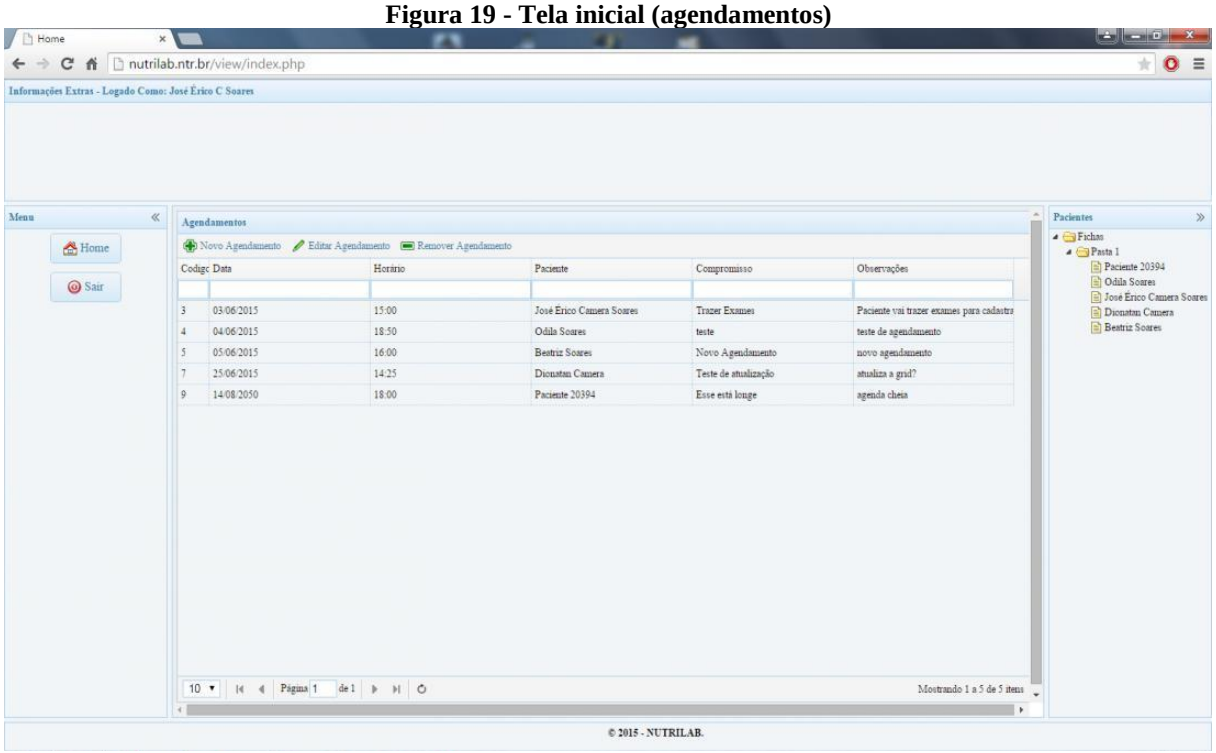
Figura 18 - Tela Login



Fonte: Do autor.

Já a figura 18 possui a interface de acesso ao sistema. Para acessa-lo deve-se escolher entre o acesso de uma nutricionista ou um usuário comum (cadastrado anteriormente por uma nutricionista). Se o *login* for efetuado com uma nutricionista é necessário inserir o cpf da mesma além da pergunta secreta e sua frase também secreta. No caso de ser um usuário comum, deve-se inserir um e-mail, sua pergunta e frase secreta.

Figura 19 - Tela inicial (agendamentos)

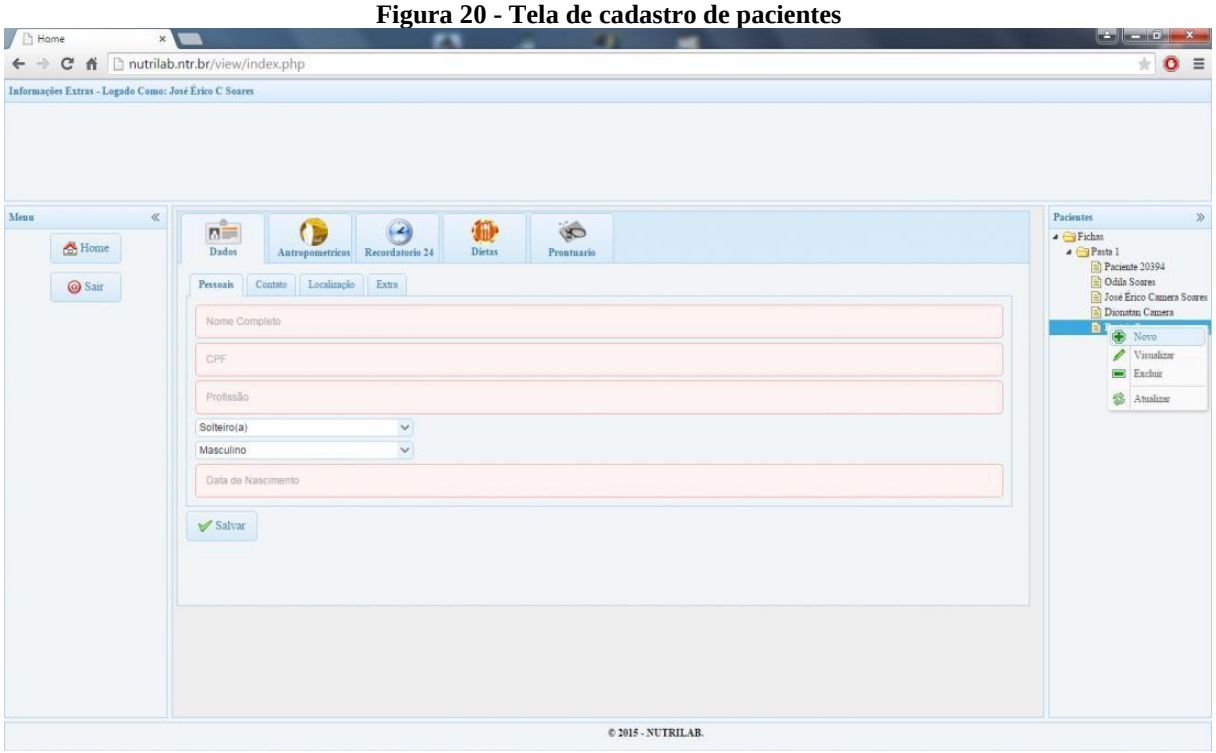


Fonte: Do autor.

Observando a figura 19, visualiza-se a tela inicial logo após o acesso ao sistema. Esta tela mostra os agendamentos da nutricionista, listando dia, hora, paciente e anotações referente a este agendamento.

Já nesta tela também é possível observar os botões de comando do sistema, como por exemplo o de *logout* ao lado esquerdo e as fichas que apresenta os dados de cada paciente cadastrado para a conta acessada ao lado direito.

Figura 20 - Tela de cadastro de pacientes



Fonte: Do autor.

A figura 20 apresenta a tela de cadastro de um novo paciente, esta é subdividida em dados pessoais, de localização, contatos, etc. Ainda possui as opções para cadastros de dados para coleta de medidas (antropométricas), recordatório de 24 horas, dietas e prontuário.

REFERÊNCIAS

AMBLER, Scott W. *Mapping Objects to Relational Databases: O/R Mapping In Detail*. (s.d.). Disponível em: <http://www.agiledata.org/essays/mappingObjects.html> Acesso em: 07 nov. 2014.

CARVALHO, Luiz. *Object-Relational Mapping(ORM) – Mapeamento de Objeto-Relacional*. 2011. Disponível em: <http://www.redrails.com.br/2011/04/13/object-relational-mappingorm-mapeamento-de-objeto-relacional> Acesso em: 07 nov. 2014.

COMPOSER. Site Get Composer. Disponível em: <https://getcomposer.org/> Acesso em: 06 nov. 2014.

DOCTRINE. Site Doctrine Docs. Disponível em: <http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/tutorials/getting-started.html> Acesso em: 06 nov. 2014.

DUNGLAS, Kévin. *Persistence in PHP with the Doctrine ORM*. Birmingham, Reino Unido: Packt Publishing Ltd, 2013.

FIELDING, Roy T. *Architectural Styles and the Design of Network-based Software Architectures*. 2000. 162f. Dissertação (Doutorado em Filosofia) – Universidade da Califórnia - IRVINE, Califórnia, 2000. Disponível em: <https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf> Acesso em: 01 nov. 2014.

GOURLEY, David; TOTTY Brian. *HTTP: The Definitive Guide*. Sebastopol, Rússia: O'Reilly Media INC., 2002.

LEISALU, Oliver. *Comparative Evaluation of Two PHP Persistence Frameworks*. 2009. 29f. Tese (Bacharelado em Tecnologia da Informação) – Universidade de TARTU, Estônia, 2009. Disponível em: <http://sep.cs.ut.ee/uploads/Main/LeisaluBachelorThesis.pdf> Acesso em: 07 nov. 2014.

MENDES, Douglas R. *Rede de Computadores: Teoria e Prática*. São Paulo: Novatec, 2007.

OKANO, Marcelo. *Análise dos melhores ORM (Object-Relational Mapping) para plataforma .NET*. (s.d.). Disponível em: <http://www.devmedia.com.br/analise-dos-melhores-orm-object-relational-mapping-para-plataforma-net/5548> Acesso em: 06 nov. 2014.

SERPA, Deivison A. L. *Doctrine – Aprenda à usar esse incrível framework em PHP de abstração de banco de dados*. 2012. Disponível em: <http://www.deivison.com.br/doctrine-aprenda-a-usar-esse-incrivel-framework-em-php-de-abstracao-de-banco-de-dados/> Acesso em: 07 nov. 2014.

TILKOV, Stefan. *A Brief Introduction to REST*. 2007. Disponível em: <http://www.infoq.com/articles/rest-introduction> Acesso em: 01 nov. 2014.

ZEMEL, Tárcio. *Composer: a evolução PHP*. 2012. Disponível em: <http://desenvolvimentoparaweb.com/php/composer-a-evolucao-php> Acesso em: 07 nov. 2014.

jQuery EasyUi. Site jQuery EasyUi. Disponível em: <http://jeasyui.com> Acesso em: 19 mai. 2015.

InfoQ. Site InfoQ Brasil. Disponível em: < http://www.infoq.com/br/presentations/oauth2-uma-bordagem-para-seguranca> Acesso em: 24 abr. 2015.

Creative Nutri. Site Creative Nutri. Disponível em: <http://www.skopein.com/creativenutri/index.php?menu=2> Acesso em: 01 mai. 2015.

Slim Framework. Site Slim Framework. Disponível em: http://www.slimframework.com Acesso em: 24 abr. 2015.