RELATÓRIO DETALHADO DO PROJETO CONTACTS.COM ®

1. TESTES UNITÁRIOS

Testes unitários são basicamente testes realizados nas menores partes testáveis de um software, dependendo do caso essas partes testáveis podem ser métodos, classes, funcionalidades ou módulos.

Por exemplo se o programa for desenvolvido em uma linguagem que suporte paradigma funcional por exemplo, a menor parte testável do código será uma função. No caso de orientação a objetos o teste será em algum método de alguma classe/objeto do software.

A função de um teste unitário é verificar se várias ‘pequenas partes’ do projeto estão funcionando como foram projetados e se vão continuar funcionando, mesmo que outras ‘pequenas partes’ do sistema seja alterado. Geralmente os testes são feitos em métodos.

No projeto Contacts.com ® foi utilizado o framework SimpleTest para auxiliar na criação dos testes.

Para utilizar o framework é necessário apenas descompactar o arquivo (disponível em LINK), dentro da pasta do projeto, e incluir o arquivo “autorun.php” no arquivo de teste. Para verificar o retorno do teste basta abrir o arquivo no navegador e ele já mostra quantos testes foram concluídos e quais deles tiveram erros.

No sistema Contacts.com ® foram feitos dois testes unitários. No primeiro teste foi verificado se a função que faz a busca dos usuários cadastrados está retornando true.

A verificação desse retorno é importante dentro do sistema, pois se este método não retornar true isso significa que existe algum erro no comando SQL ou na base de dados. E se hipoteticamente por algum desses motivos a funcionalidade não estiver funcionando todo o sistema seria afetado.

Implementação do primeiro teste:

```
//teste que verifica se a busca dos usuarios não falhou
class TestLogin extends UnitTestCase{
    function testbuscaUsuario(){
        $teste = new pessoa();

        $this->assertTrue($teste->buscaUsuario());
    }
}
```

A ausência de uma gestão dos níveis de acesso coloca riscos significativos dentro dos ambientes empresariais, sobretudo no que diz respeito aos softwares utilizados. Antecipando e precavendo-se de eventuais riscos sejam eles internos ou externos, a equipe de desenvolvimento do projeto Contacts.com ®, realizou um teste para verificar se assim que o usuário fazer o login as permissões suas sejam verificadas. Esse teste verifica qual tipo de permissão foi encontrada e se o método retornou Administrador.

Implementação do segundo teste.

```
//Verifica se foi possivel localizar a permissao e se ela for
de Administrador
class TestPermissao extends UnitTestCase{
    function testretornaPermissao(){
        $re = new Factory();
        $pe = $re->retornaPermissao();
        $resultado = $pe -> permissao();

        $this->assertEqual($resultado,'Administrador');

    }
}
?>
```

2. USER STORY CARDS

Dentro da Engenharia de Software existe um subcampo que chamamos de engenharia de requisitos, essa por sua vez, estuda as formas de analisar, validar, especificar e documentar requisitos. E é nesta fase do projeto que exploramos o quê, como e para quem iremos desenvolver as próximas funcionalidades do sistema, haja visto que se o software não for impactar na vida de ninguém, ele não deveria nem ser desenvolvido.

Para isso é que são feitos os User Story Cards, que podem ser pequenos cartões ou mesmo um texto curto com sentenças claras. O objetivo dos story cards é o de facilitar a compreensão do desenvolvedor e da equipe modelando situações e requisitos do programa.

Não existe um modelo padrão a se seguir na hora de confeccionar esses cartões, com algumas equipes preferindo faze-los de forma manuscrita e outras com o auxílio de softwares e/ou serviços on-line. A única regra a se respeitar na hora fazer um User Story Cards, é a de seguir o seguinte formato:

"Como um <ator>, quero <meta/desejo> de modo que <benefício/resultado>"

No sistema Contacts.com ® foram implementados dois User Story Cards, cada um deles direcionados para cada um dos dois tipos de usuários que o software terá.

Dentro de organizações empresariais existem setores que pouco ou nada dependem de ligações telefônicas, e outros que as ligações são indispensáveis.

Para os setores onde as ligações não são pré-requisitos para que os colaboradores executarem suas tarefas, foi implementado o tipo de usuário ‘Usuário, que apenas tem a opção de visualizar os contatos já existentes. Por exemplo funções dentro de setores da produção, que apenas fará uso da lista esporadicamente e quando o fizer será apenas para encontrar algum contato já existente.

Já para os que trabalham e fazem uso de ligações foi implementado o tipo de usuário ‘Administrador’, esse por sua vez tem as opções de criação / edição / deleção de contatos, haja vista a necessidade. Por exemplo funções como telefonista e vendedor/comprador, que estão constantemente necessitando fazer inserções ou ajustes em suas listas de contatos.

Esse tipo de estrutura organizacional facilita a gestão dos usuários que utilizaram o sistema, pois com essa organização a TI consegue conceder privilégios de acesso aos usuários de acordo com a necessidade de cada um e levando em conta as políticas de segurança da empresa que o usar. A ausência de uma gestão da identidade de acesso coloca riscos significativos na conformidade e também na segurança total da empresa, dando margem a ameaças tanto externas como internas.

Link para os User Story Cards [LINK].

3. DIAGRAMAS

Diagramas são representações gráficas usadas para demonstrar de forma simplificada esquemas ou resumos sobre determinados assuntos. Dentro da área de computação essas representações são usadas para especificar, visualizar e documentar a estrutura e funcionamento de um software ou parte dele.

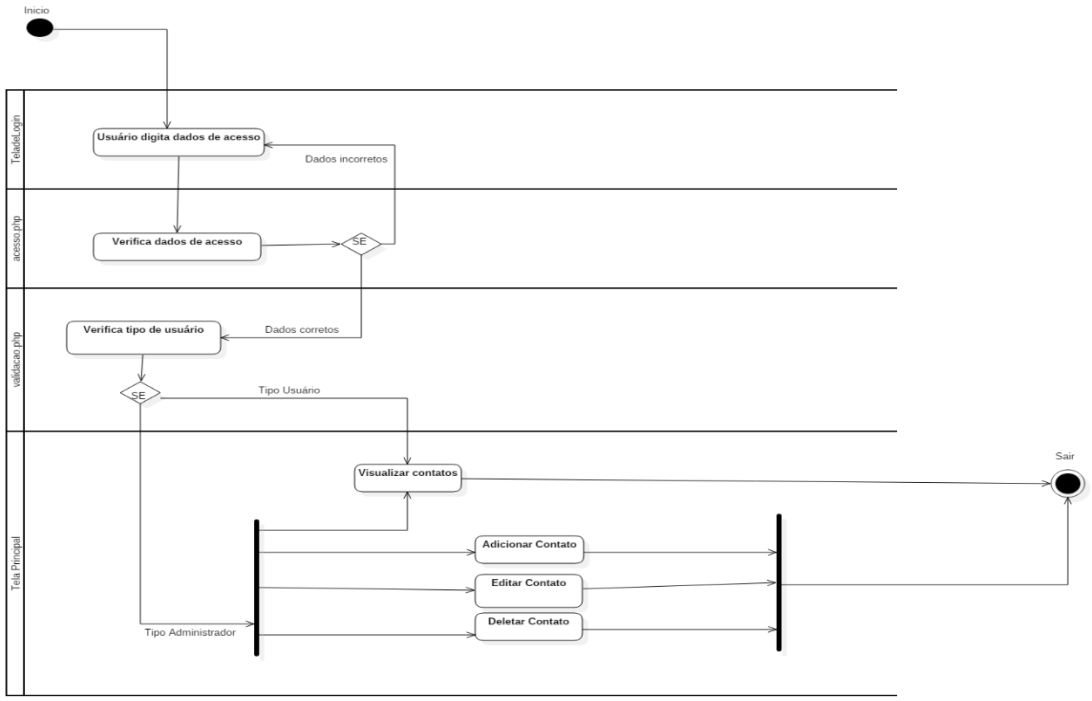
Para se ter uma padronização foi criada a Linguagem de Modelagem Unificada, comumente chamada de UML. Os diagramas desenvolvidos usando essa linguagem de modelagem são divididos em vários tipos, cada um com um proposito especifico.

Por se tratar de um software relativamente pequeno, e de fácil entendimento foi escolhido dois modelos dentre os inúmeros modelos existentes, para fazer a representação gráfica do projeto Contacts.com ®. O primeiro deles é o diagrama de atividades e o segundo o chamado diagrama de caso de uso.

Diagramas de atividades, são usados para representar graficamente atividades de qualquer parte ou componente do sistema. Os diagramas de atividade não são importantes somente para a modelagem de aspectos dinâmicos de um sistema ou um fluxograma, mas também para a construção de sistemas executáveis por meio de engenharia de produção reversa.

No projeto o diagrama de atividades foi usado para detalhar algumas das funcionalidades do software, graças ao uso desse modelo foi possível explicitar o funcionamento dos mecanismos de controle de acesso usados (login/senha/permissões) e como eles fazem o controle de acesso e de permissões de manipulação.

Link para Diagrama de Atividades [LINK].



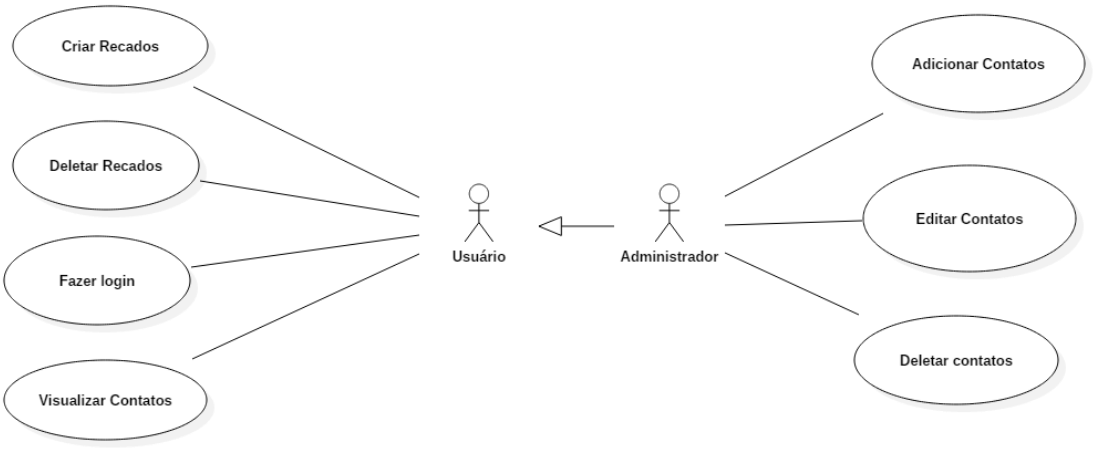
Diagramas de casos de uso, são utilizados para fazer a representação do conjunto de comportamentos de alto nível que o sistema deve executar para um determinado ator. Esse tipo de diagrama é um dos mais simples, e não há necessidade de grandes detalhamentos.

O diagrama de caso de uso confeccionado ilustra de forma clara quais as atribuições para cada tipo de usuários que o sistema possui.

Na representação fica explicito que o tipo de usuário ‘Usuário’ deverá fazer seu login para ter acesso ao sistema, e poderá apenas visualizar contatos e criar / deletar recados.

Já o tipo de usuário ‘Administrador’ além de poder realizar as mesmas funções do tipo ‘Usuário’, poderá também gerenciar a lista de contatos, adicionando, editando e deletando registros.

Link para Diagrama de Caso de Uso [LINK].



4. DESIGN PATTERN

Design patterns, ou padrões de desenvolvimento de sistemas, são modelos que auxiliam o desenvolvedor nortear a criação da sua aplicação.

Existem diferentes modelos de design patterns, e que são usados em diferentes etapas do desenvolvimento, alguns deles por exemplo são usados durante a criação, e ajudam a abstrair a construção dos objetos, permitindo a flexibilidade através da herança de classes. Outros exemplos são os padrões estruturais, que tem por objetivo a realizar o relacionamento entre as entidades para facilitar o design do sistema/aplicação e os comportamentais que são usados para facilitar a comunicação entre os objetos criados.

Com o uso de Design Patterns o programador terá vários benefícios dentre eles um código mais enxuto, limpo, organizado, aumento da qualidade e redução da complexidade do código. Mas a maior vantagem de utilizar esses padrões é a facilidade na manutenção do código.

No desenvolvimento do projeto Contacts.com ®, foram aplicados três designs paterns, o Singleton, Factory e o Strategy.

O Singleton é utilizado quando queremos que só tenha uma instancia da classe em todo o projeto. Este método foi utilizado na classe conexão onde somente a própria classe pode chamar o construtor. Este padrão de projeto vai garantir que o objeto só seja instanciado uma vez em todo o projeto.

Implementação do Design Pattern Singleton.

```php
//design pattern Singleton
class conexao
{
    // Instância da classe
    private static $instance = null;
    private $conn;

    // Construtor privado: só a própria classe pode invocá-lo
    private function __construct()
    {
        $host = "localhost";
        $user = "root";
        $pswd = "";
        $db = "projeto";

        try {
            $this->conn = mysqli_connect($host, $user, $pswd, $db);
            $this->conn->set_charset('utf8');
        } catch (Exception $e) {
            die("Erro na conexão com MySQL! " . $e->getMessage());
        }
    }
    // método estático
    static function getInstance()
    {
        // Já existe uma instância?
        if (self::$instance == NULL)
            self::$instance = new conexao();        // Não existe, cria a instância
        return self::$instance;                     // Já existe, simplesmente retorna
    }

    // Previne o uso de clone
    private function __clone() {}
     public function getCon(){
        return $this->conn;
    }
}
```

O padrão Strategy serve para “definir uma família de algoritmos, encapsular cada uma delas e torná-las intercambiáveis. O Strategy permite que o algoritmo varie independentemente dos clientes que o utilizam” (como definido no livro do GoF). O padrão é aplicado em situações em que muitas classes se relacionam e diferem apenas no modo de atuação. No projeto foi usado este padrão no arquivo permissões, onde várias classes implementam a interface, com o mesmo método, mas com ações diferentes. Assim pode-se criar várias classes implementando a interface e ter a garantia que todas elas irão conter os mesmos métodos, mas com ações diferentes.

Implementação do Design Pattern Strategy.

```php
//desgin pattern Strategy
interface tipoPermissao{
    public function permissao();
}

class Administrador implements tipoPermissao{
    public function permissao(){
        $_SESSION['variavel'] = '0';
        return 'Administrador';
    }
}

class Usuario implements tipoPermissao{
    public function permissao(){
        $_SESSION['variavel'] = '1';
        return 'Usuario';
    }
}
```

Por fim o Factory, que funciona como uma fábrica de instancias, onde uma classe busca várias outras classes. No projeto foi utilizado no arquivo permissões, onde a função após uma verificação do nível de acesso do usuário retorna a classe necessária.

Implementação do Design Pattern Factory.

```php
//desgin pattern Factory
class Factory{
    public function retornaPermissao(){
        $permissao = $_SESSION['variavel'];
        if($permissao == '0'){
            $pe = new Administrador();
            return $pe;
        }elseif($permissao == '1'){
            $pe = new Usuario();
            return $pe;
        }
    }

}
```

5. ESPECIFICAÇÃO DA ARQUITETURA DE SOFTWARE

Arquitetura do sistema

O tipo de arquitetura de sistema definido para o projeto Contacts.com ®, foi o de Cliente x Servidor, onde tem-se:

• Cliente: as estações clientes, por meio do acesso com o navegador web (browser).

• Servidor: O SGBD – Sistema Gerenciador de Banco de Dados, - no caso o MYSQL, juntamente com os aplicativos desenvolvidos em PHP v5.5.12 lotados em um Servidor Web Apache v2.4.9 (Win32).

Assim o projeto tem duas camadas: uma de apresentação – lado Cliente e outra contendo as regras de negócio e a de banco de dados - no lado Servidor.

Ambiente de Produção / Linguagem de Programação

O projeto Contacts.com ® foi desenvolvido praticamente sem o uso de frameworks ou IDEs sendo que em sua maior parte foi codificado a partir do editor de código Sublime na versão 3.

A linguagem de programação usada foi o PHP v5.5.12, em complemento também foi usado a linguagem de marcação HTML5 e o CSS3 para sua estilização.

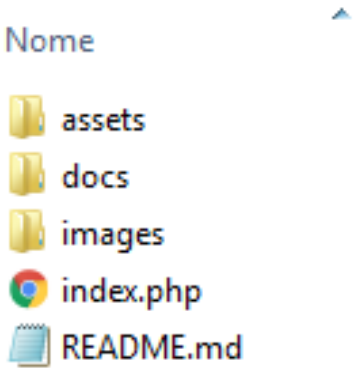
Apenas foi feito o uso de framework em sua fase de testes, com o framework SimpleTest sendo escolhido para completar essa etapa.

Tipo de Arquitetura

Como já mencionado o Contacts.com ® foi baseado numa arquitetura cliente x servidor, aonde estão separados os clientes dos servidores, ocorrendo a troca de informações por meio de uma rede de computadores (Intranet ou Internet).

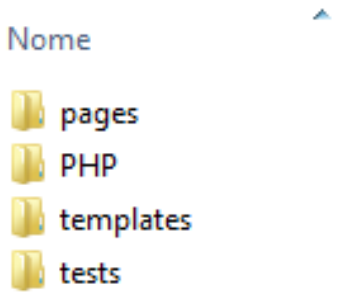
Estruturas de pastas

Como não foi utilizado Framework para o desenvolvimento, a estrutura principal do projeto, foi simplificada ao máximo para facilitar a compreensão e futuras modificações/manutenções.



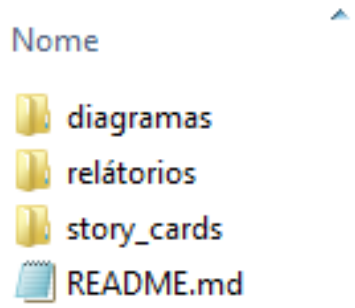
Onde index.php, é o arquivo principal da aplicação.

Na pasta ‘assets’ encontram-se todos os outros arquivos necessários para a criação e funcionamento do sistema (.php), e também os arquivos de estilização (.css), subdivididos conforme imagem abaixo.

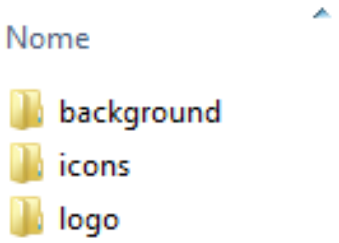


A pasta ‘pages’ armazenas os arquivos responsáveis por realizar a criação das páginas do projeto (telas Principal, Edição e Deleção). Já a pasta ‘PHP’ armazena arquivos que realizam as conexões e tarefas da aplicação. O diretório ‘templates’ guarda os arquivos de estilização, e a pasta ‘testes’ contém além dos arquivos do tipo php que foram implementados para a criação dos testes unitários, como os arquivos necessários para o framework SimpleTest funcionar.

O diretório docs é responsável pelo armazenamento dos arquivos que documentam a aplicação, como os diagramas, relatórios e algumas imagens.



O arquivo imagens armazena as imagens do projeto, como backgrounds, ícones e logo.



Como mencionado a estrutura de pastas foi simplificada ao máximo para manter um alto nível de manutenibilidade.

Sistema Operacional / Versões dos Browsers

Para garantir que sua aparência / layout, e todas as suas funcionalidades estejam garantidas, o projeto deve ser acessado a partir de sistemas operacionais iguais ou superiores ao Windows 8 (32 / 64 bits).

Para a realização do acesso ao sistema Contacts.com ®, o usuário deve possuir, no mínimo, uma das seguintes versões de Browser:

•Microsoft Internet Explorer 8 ou superior;

•Mozilla Firefox 14 ou superior;

•Safari 3 para Windows ou superior;

•Google Chrome 21 ou superior;

•Opera 11 ou superior.