

CENTRO UNIVERSITÁRIO MONTE SERRAT

EDSON ROBERTO BORGE JUNIOR

GREGORY VILAR DE SOUZA PASSOS

LEONARDO ESPOSITO SANTÓ LOPES

MARCUS VINICIUS DE S. Q. LOZANO

RODRIGO FERNANDES ANTUNES

WILLIAN GONÇALVES RESILLE

**SISTEMA PARA GESTÃO DE UMA EMPRESA DO RAMO DE ARMAZENAMENTO DE CONTÊINERES**

Santos

2015



CENTRO UNIVERSITÁRIO MONTE SERRAT

EDSON ROBERTO BORGE JUNIOR

GREGORY VILAR DE SOUZA PASSOS

LEONARDO ESPOSITO SANTÓ LOPES

MARCUS VINICIUS DE S. Q. LOZANO

RODRIGO FERNANDES ANTUNES

WILLIAN GONÇALVES RESILLE

**SISTEMA PARA GESTÃO DE UMA EMPRESA DO RAMO DE ARMAZENAMENTO DE CONTÊINERES**

Trabalho Interdisciplinar (Projeto Integrador) apresentado ao Centro Universitário Monte Serrat como exigência parcial para a aprovação na disciplina Projeto Integrador IV do Curso de Analise e Desenvolvimento de Sistemas.

**Orientador(a): Me. Helio Augusto de Lima Rangel**

Santos

2015

EDSON ROBERTO BORGE JUNIOR

GREGORY VILAR DE SOUZA PASSOS

LEONARDO ESPOSITO SANTÓ LOPES

MARCUS VINICIUS DE S. Q. LOZANO

RODRIGO FERNANDES ANTUNES

WILLIAN GONÇALVES RESILLE

**SISTEMA PARA GESTÃO DE UMA EMPRESA DO RAMO DE ARMAZENAMENTO DE CONTÊINERES**

Trabalho Interdisciplinar (Projeto Integrador) apresentado ao Centro Universitário Monte Serrat como exigência parcial para a aprovação na disciplina Projeto Integrador IV do Curso de Analise e Desenvolvimento de Sistemas.

**Orientador(a): Me. Helio Augusto de Lima Rangel**

**BANCA EXAMINADORA**

Nome do examinador:

Titulação:

Instituição:

Nome do examinador:

Titulação:

Instituição:

**Local:** Centro Universitário Monte Serrat – UNIMONTE

**Data da aprovação**: / /

Sumário

[RESUMO 7](#_Toc435541068)

[INTRODUÇÃO 8](#_Toc435541069)

[1 METODOLOGIA AGIL 9](#_Toc435541070)

[2 DESENVOLVIMENTO MOBILE 13](#_Toc435541071)

[2.1 DESENVOLVIMENTO 15](#_Toc435541072)

[2.1.1 FERRAMENTAS 15](#_Toc435541073)

[2.1.2 ARMAZENAMENTO 16](#_Toc435541074)

[3 TESTE DE SOFTWARE 17](#_Toc435541075)

[3.1 CONCEITOS BASICOS 17](#_Toc435541076)

[3.2 TIPOS DE TESTES 18](#_Toc435541077)

[3.3 FERRAMENTAS DE TESTE 19](#_Toc435541078)

[3.4 AUTOMAÇÃO DE TESTES 20](#_Toc435541079)

[4 CONCEITOS DE AUDITORIA 21](#_Toc435541080)

[5 CONCEITO DE INTEGRAÇÃO CONTINUA 25](#_Toc435541081)

[6 CONCEITOS DE PADRÕES DE PROJETO 28](#_Toc435541082)

[7 EMPREENDEDORISMO SOCIAL 30](#_Toc435541083)

[8 RELATÓRIO TÉCNICO 32](#_Toc435541084)

[8.1 DESCRIÇÃO DA EMPRESA 32](#_Toc435541085)

[8.2 OBJETIVO DO SISTEMA 32](#_Toc435541086)

[8.3 CENÁRIO ATUAL 33](#_Toc435541087)

[8.4 CENÁRIO PROPOSTO 33](#_Toc435541088)

[8.5 MER DO BANCO DE DADOS 35](#_Toc435541089)

[8.6 CASOS DE USO 36](#_Toc435541090)

[8.7 TELAS DO SISTEMA 41](#_Toc435541091)

[8.8 CRONOGRAMA 46](#_Toc435541092)

[8.9 DIAGRAMA DE CASOS DE USO 47](#_Toc435541093)

[8.10 POLÍTICA DE SEGURANÇA 47](#_Toc435541094)

[8.11 PLANO DE CONTINGÊNCIA 48](#_Toc435541095)

[8.12 RELATÓRIO DE AUDITORIA 48](#_Toc435541096)

[8.13 METODOLOGIA 49](#_Toc435541097)

[8.14 RELATÓRIO DE DESENVOLVIMENTO 49](#_Toc435541098)

[8.14.1 EXEMPLO 49](#_Toc435541099)

[8.15 RELATÓRIO DE TESTE 49](#_Toc435541100)

[8.15.1 EXEMPLO 49](#_Toc435541101)

[8.16 MELHORIAS 50](#_Toc435541102)

[8.16.1 IMPLANTAÇÃO MODELO MVC 50](#_Toc435541103)

[8.16.2 IMPLANTAÇÃO DE LOG 50](#_Toc435541104)

[8.16.3 IMPLANTAÇÃO DE INTEGRAÇÃO CONTÍNUA 51](#_Toc435541105)

[8.16.3.1 FERAMENTAS 51](#_Toc435541106)

[8.16.3.2 ESTUTURA 51](#_Toc435541107)

[CONCLUSÃO 52](#_Toc435541108)

[ANEXO 53](#_Toc435541109)

[REFERÊNCIAS BIBLIOGRÁFICAS 66](#_Toc435541110)

Lista de Figuras

Figura 1 - Projeto XP. 12

Figura 2 - Nativo x Hibrido x Html5 15

Figura 3 - Defeito x Erro x Falha 18

Figura 4 - Esquema de Integração Continua 27

Figura 5 - Empreendedorismo empresarial x Empreendedorismo social. 31

Figura 6 - Logotipo da Empresa. 32

Figura 7 - MER do Banco de Dados. 35

Figura 8 - Tela de Login. 41

Figura 9 - Tela Principal. 41

Figura 10 – Tela de Fluxo de Entrada. 42

Figura 11 – Tela de Fluxo de Saída. 42

Figura 12 - Menu do Sistema. 43

Figura 13 - Menu de Configurações. 43

Figura 14 - Tela de Cadastro de Motorista. 44

Figura 15 - Tela de Cadastro de Caminhão. 44

Figura 16 - Tela de Cadastro de Container. 45

Figura 17 - Tela de Consulta. 45

Figura 18 - Lista de Tarefas 46

Figura 19 - Gráfico de Grant. 46

Figura 20 - Diagrama de Casos de Uso. 47

Figura 21 - Integração Continua. 51

# 

**Lista de Tabelas**

Tabela 1 - Cadastrar Entrada......................................................................................36

Tabela 2 - Cadastrar Saída..........................................................................................38

Tabela 3 - Cadastrar Usuário......................................................................................40

# RESUMO

Uma das grandes necessidades na área portuária atualmente é a otimização da logística presente, com o intuito de oferecer aos seus respectivos clientes um serviço efetivo, combinando agilidade, qualidade e custos reduzidos. Utilizando-se do método dedutivo e da pesquisa bibliográfica para fundamentar os assuntos aqui abordados, foi desenvolvido um *software* para um terminal de contêineres de pequeno porte onde o principal objetivo baseia-se em terminais de grande porte, onde as rotinas de serviço são quase que sempre automatizadas, contando sempre com o foco no baixo custo e no retorno satisfatório, tanto financeiramente quanto empresarialmente. Foram utilizados fundamentos e metodologias reconhecidas pelo mercado para desenvolver uma solução informatizada, onde o principal objetivo é a facilidade na organização do terminal e auxiliando efetivamente no suporte aos funcionários no sentido de familiarização com o *software*.

Palavras-chave: Logística Portuária, Terminal de container, *Software*.

# INTRODUÇÃO

Cada vez mais é necessária a utilização de ferramentas para otimizar o tempo e aperfeiçoar as rotinas de trabalho em um terminal portuário. A área de Tecnologia da Informação vem se aproximando da gestão logística nos terminais portuários, tornando-os mais fluidos e informatizados.

A gestão logística nos terminais portuários no futuro será realizada basicamente através do gerenciamento da informação, ressaltando que para isso as empresas no setor devem investir em informação e informatização de última geração, para poder gerenciar e oferecer serviços de qualidade.

Percebendo a necessidade de uma economia de tempo em cada tarefa burocrática e agilidade na liberação de entrada e saída de caminhões no pátio, propomos uma solução onde pensamos na melhoria de todas as rotinas operacionais de um terminal de contêineres. Inicialmente desenvolvido em uma versão para *desktop*, o ProPátio trazia como diferencial o sistema gráfico de exibição de quadras, que oferece condições para o controle, a localização e a organização mais efetiva do pátio. Todas as consultas e reposicionamentos são feitos remotamente, evitando remoções desnecessárias dos contêineres, com maior eficiência do processo e, consequentemente, custo reduzido.

Em busca de ampliar a capacidade e usabilidade do *software*, objetiva-se nesse trabalho o desenvolvimento da plataforma *web* do sistema RetroPátio, registrado sob o domínio *http://www.retropatio.com.br*, se tornando uma forma mais versátil de ser operada nos terminais portuários, fazendo o gerenciamento, organização e visualização dos contêineres.

O trabalho estrutura-se da seguinte forma: Na primeira parte, mostra a base de desenvolvimento da aplicação seguindo os conceitos de Análise e Desenvolvimento de Sistemas; prosseguindo com os detalhes de toda a implementação do sistema, com descrição detalhada de telas e todas as respectivas funcionalidades.

# Metodologia agil

Na década de 80, existia um pensamento de que o melhor jeito de se fazer um software de qualidade era realizar um planejamento minucioso, com uma documentação pesada, uma análise rigorosa e sem espaço para mudanças de última hora. Esse pensamento veio da engenharia de *software* acostumada a trabalhar em projetos grandes e de longa duração, como por exemplo, sistemas aeroespaciais, que exigem esse tipo de abordagem pelo risco de um erro pequeno causar um problema gigante.

Mas quando começaram a trabalhar com a ideia de sistemas menores, corporativos, viu-se que por mais que esse tipo de pensamento era bom para casos extremos, se tornava ineficiente para projetos mais simples, onde se passava mais tempo com a análise, planejamento e documentação do que com o desenvolvimento do software em si, causando um transtorno quanto à demora na entrega de um software que supostamente poderia ser feito de modo mais simples. Isso gerou muita insatisfação dos desenvolvedores que passavam mais tempo documentando do que gerando código e levou muitos deles a proporem ’métodos ágeis’, onde a preocupação maior está no desenvolvimento de um software funcional em vez de sua documentação. (SOMMERVILLE, 2011).

Foi através dos principais desenvolvedores desses ‘métodos’ que anos depois (mais precisamente 2001) acordaram entre si e criaram o manifesto ágil. Que afirma:

(BECK, 2001).

Estamos descobrindo maneiras melhores de desenvolver software fazendo-o nós mesmos e ajudando outros a fazê-lo. Através deste trabalho, passamos a valorizar:

**Indivíduos e interação entre eles** mais que processos e ferramentas

**Software em funcionamento** mais que documentação abrangente

**Colaboração com o cliente** mais que negociação de contratos

**Responder a mudanças** mais que seguir um plano

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.

O manifesto como o mesmo fala, valoriza mais os itens citados a esquerda, mas sem tirar o mérito dos itens à direita. Com certeza um software tem que possuir documentação, processos, ferramentas, planos e etc. Mas a metodologia ágil se preocupa mais em desenvolver um software funcional que responde a mudanças do que possuir uma burocracia para cada fase de desenvolvimento e implementação. Pensando nisso foram criados doze princípios básicos que suportam o manifesto, são eles:

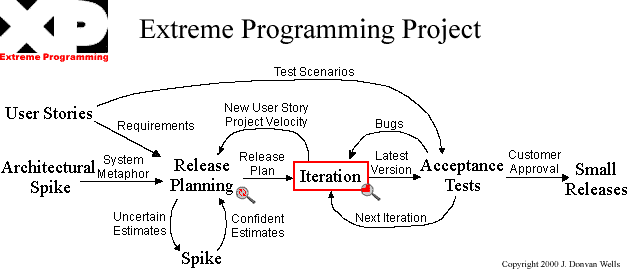
(BECK, 2001).

1. Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor.
2. Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.
3. Entregar software funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos.
4. Pessoas relacionadas à negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto.
5. Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho.
6. O Método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara.
7. Software funcional é a medida primária de progresso.
8. Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes.
9. Contínua atenção à excelência técnica e bom design, aumenta a agilidade.
10. Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.
11. As melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis.
12. Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.

A metodologia ágil vem com o seu maior objetivo que é trabalhar com pessoas, sejam essas pessoas a equipe ou clientes, o modo de como ela trabalha o seu desenvolvimento torna mais comum essas interações tendo em vista que *softwares* são desenvolvidos em partes, sempre sendo enviadas ao cliente à medida que forem sendo finalizadas. Assim o cliente da um *feedback* mais rápido e preciso do que esta certo ou errado, ajudando o desenvolvedor a fazer um software mais próximo a realidade do cliente.

Uma das metodologias ágeis mais conhecidas é a *Extreme Programming* (XP), que teve seu primeiro projeto iniciado em seis de março de 1996, o seu sucesso vem da possibilidade de entregar um *software* à medida que ele for sendo necessário, ao invés de se fazer todo um planejamento e entrega-lo de uma vez só. Ela enfatiza as pessoas que trabalham no projeto e o trabalho em equipe dessas pessoas (gerentes, clientes, desenvolvedores etc.). Não existe um “chefe” nessa metodologia, todos possuem seus papeis e autoridade para falar e propor soluções quando necessário, talvez esse seja um dos pontos ruins da XP, pois possuir um time focado que tem uma excelente comunicação e um senso de colaboração não é uma tarefa fácil. Desenvolver um software de maneira ágil é se livrar de tudo que prende o desenvolvedor, fazendo com que ele não apenas escreva um código porque o foi mandado, mas porque ele acha que essa é a maneira mais viável. Um dos papeis mais importantes nesse desenvolvimento é o do cliente, a pessoa que esta comprando esse software também é parte dessa equipe. Ninguém melhor que ele vai saber como tem que ser e quais funcionalidades são necessárias, outra tarefa difícil para a XP, fazer com que o cliente trabalhe junto com a equipe de desenvolvimento, que cliente iria querer isso? Mais trabalho? Pra que? Mas à medida que a XP entra em ação com sua velocidade no desenvolvimento, primeiro separando as partes importantes e trabalhando nelas para o primeiro release, fazendo o cliente ver realmente do que o seu *software* é capaz e quais atributos ele pode querer atribuir ou retirar, pois afinal de conta o produto final é dele e ele mais do que ninguém tem que falar como tem que ser feito. Mas como isso é feito? A figura 1 mostra o processo de desenvolvimento da XP. (WELLS, 2013).

Figura 1 - Projeto XP.

 Fonte: http://www.extremeprogramming.org/map/images/project.gif.

Os requisitos do usuário são primeiramente planejados, estimados e testados antes do desenvolvimento do código, quando tudo esta aceito é codificado e testado novamente, caso existam *bugs* são consertados. Após a aprovação do cliente de que tudo está 100% é liberado um pequeno *release* e se começa a pensar na próxima história do usuário.

A *Extreme Programming* vem com soluções ágeis para tarefas que deveriam ser muito estudadas e pensadas em outras metodologias, com uma única coisa em mente “pessoas” e seu sucesso em várias empresas independente de seus tamanhos e regiões, provam que é possível desenvolver de uma maneira voltada para o cliente e com foco no que importa, mas não é uma maneira fácil, obter confiança e conceder liberdade para uma equipe decidir como levar o projeto é algo que poucas empresas estão dispostas a arriscar. (WELLS, 2013).

Em contrapartida existem frameworks como o RUP (*Rational Unified Process*) que agrega atividades, fluxos, artefatos, papeis capazes de se adaptarem a vários tipos de projetos, podendo ser utilizado em projetos ágeis, como é afirmado por Pinto (2012) “o RUP não é uma metodologia. É um arcabouço de boas práticas que devem ser customizadas para aplicação específica em projetos de software, inclusive projetos ágeis, que possuem reduzidos prazos de entrega.*”*, porém ao contrario da XP o RUP tende a ser mais burocrático por possuir essa vasta coleção de boas práticas, podendo assim causar transtornos se utilizado de forma errada. (PINTO, 2012).

# Desenvolvimento mobile

O avanço da tecnologia nos permite cada vez mais estar em total conexão com qualquer pessoa que quisermos, e isso traz inúmeras melhorias em questão de poder se aproximar de pessoas que não podemos ter um contato pessoal, e cabe aos desenvolvedores tornar isso possível. A invenção do telefone celular vem de muitos anos atrás (1947 para ser mais preciso) e desde então a reinvenção dessa tecnologia continua até hoje e por muitos anos após esta presente data.

Claro que quando foi inventado o telefone celular não fazia nem 50% do que hoje é capaz, mas o conceito de poder ligar e conversar com qualquer pessoa em qualquer lugar é extraordinário, coisa que não é mais assunto de mesa de bar. Hoje em dia se tornou tão comum o seu uso, que em muitas dessas mesas de bar nem conversa mais acontece, pois a imersão que esses aparelhos nos permitem, acaba tirando nós mesmos da realidade.

Atualmente é indispensável a utilização de um celular (ou *smartphone* como agora é chamado), seja para mandar mensagens, ligar, jogar, navegar na internet, conectar-se a redes sociais etc. Isso traz várias vantagens, pois todas as respostas estão no seu bolso, mas ao mesmo tempo torna pessoas viciadas nessa tecnologia, por exatamente possuir tudo que é necessário no dia-dia, seja pra saber das ultimas noticias, ou utilizar de mapas para dar a localização e até a rota de onde se precisa ir. (JORDÃO, 2009).

Essa evolução abriu muitas portas para um novo mercado, os aplicativos para celular (Apps) que mudaram completamente o conceito do celular. Anteriormente o celular possuía uma predefinição que permitia ao usuário ter acesso a relógio, mensagens, ligações, alguns jogos pré-instalados, mas hoje em dia você compra um celular que vem com essas mesmas predefinições e tem a “liberdade” de entrar na “loja de aplicativos” e baixar o que você bem entender, e isso trouxe uma revolução no mercado de desenvolvimento, porque antes desse modelo ser apresentado, eram desenvolvidos sistemas para *desktop* e ou para internet, agora você pode desenvolver um aplicativo que é exposto nessa loja virtual, sendo que todas as pessoas que possuem um *smartphone* tem acesso a essa loja e até se pode cobrar por esse aplicativo que foi desenvolvido.

É um mercado que está em alta, pois os aplicativos permitem que problemas que antes nem pareciam problemas sejam solucionados. Como por exemplo, procurar pelo restaurante mais perto onde quer que você esteja. De acordo com Mirelle Pinheiros: “O mercado nacional ajuda investidores e desenvolvedores: atualmente, ele movimenta US$ 25 bilhões anuais, com expectativa de alcançar US$ 70 bilhões em 2017, de acordo com projeção do Ministério da Ciência, Tecnologia e Inovação (MCTI).”. Esses números são a prova de que o mercado é grande e difundido, dando oportunidade para qualquer pessoa que possuir uma ideia. (PINHEIRO, 2015).

Mas não adianta apenas fazer qualquer aplicativo, Segundo o consultor Antonio Carlos Soares: “é preciso oferecer alguma coisa que seja fundamental na vida do usuário ou que vá mudá-la. Por isso, é preciso pensar no aplicativo como um produto tradicional: ele tem que ser lançado com o menor número possível de falhas.”. Estudos dizem que a maioria dos usuários que baixam um aplicativo novo não permanece com ele por muito tempo, se o mesmo não atingir suas expectativas e necessidades ou possuir muitos erros. Por isso que ter sucesso nesse mercado é uma tarefa que existe um extremo esforço, mas se tiver a ideia certa e a persistência nela pode ser extremamente recompensadora.

Como é o caso do aplicativo “99 Táxis”, um aplicativo que busca solucionar um problema já antigo, de conseguir um taxi seguro de maneira fácil e rápida. A empresa teve um começo complicado pela dificuldade de fazer um aplicativo que atendesse as necessidades tanto dos taxistas quanto dos clientes, ainda possuindo uma barreira tecnológica entre os taxistas e *smartphones*. Mas tiveram persistência na ideia e foram às ruas conversar com taxistas para convencê-los a utilizar o aplicativo, e aos poucos foram atraindo o que hoje são 30 mil taxistas cadastrados na cidade de São Paulo que possui uma frota de 34 mil carros. (PEGN, 2015).

Outro mercado que está crescendo muito com a chegada dos aplicativos é a indústria de jogos, de acordo com o relatório Global Games Investment Review, divulgado pela Digi-Capital. “Até 2017, os jogos *mobile* e *online* devem representar 60% do faturamento da indústria de jogos. E segundo o estudo, a indústria de jogos deve chegar ainda a um faturamento de U$$ 100 bilhões nos próximos três anos”. (LANDIM, 2014).

O mercado de games esta a todo vapor, jogar é uma das atividades mais frequentes no dia-dia das pessoas, segundo psicólogo M.Farouk Radwan videogames são jogados para: aliviar o estresse; aumentar a autoestima; fugir da realidade; redenção, entre outros. Por isso é um mercado que está sempre à procura de coisas novas e vem crescendo significativamente. (RADWAN, 201?).

Um dos jogos desenvolvidos para mobile mais famosos é o “Angry Birds” lançado em dezembro de 2009, que atualmente está na sua segunda versão o “Angry Birds 2” lançado em julho desse ano, este que já atingiu mais de 10 milhões de *downloads*, sendo que outros jogos da franquia agregam mais de 3 bilhões. A empresa Rovio dona dos famosos pássaros, não poupou esforços para transformar o jogo em uma marca, que hoje possui uma série animada através de seu aplicativo “ToonsTV”, possui diversos produtos como cadernos, bonecos, roupas etc. e previsto para 12 de maio de 2016 (no Brasil) lançará seu primeiro filme animado estrelando seus amados protagonistas. (G1, 2015).

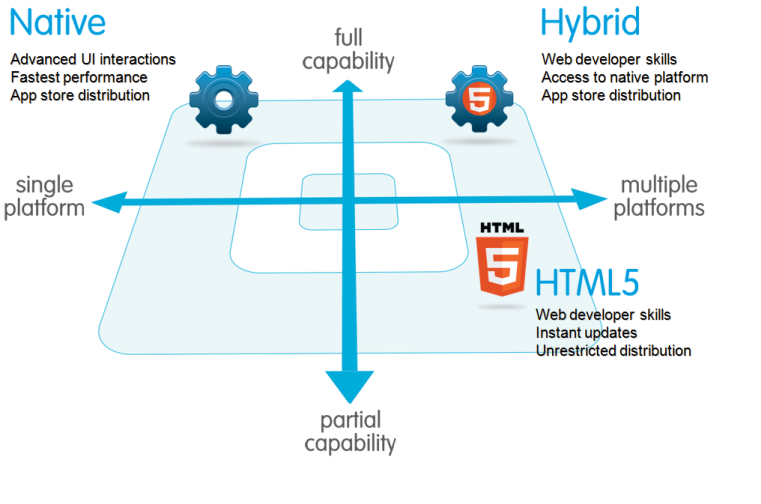
## DESENVOLVIMENTO

O mercado mobile vem crescendo dia após dia e não é exclusivo apenas para grandes empresas ou para um tipo de produto. Existem inúmeras possibilidades de sucesso para qualquer um que possui interesse e vontade de tentar algo novo (sabendo-se que esse investimento tem seus riscos).

### FERRAMENTAS

Atualmente existem diversas ferramentas para desenvolvimento mobile e elas compartilham um principio básico, de que aplicativos podem ser criados de três formas:

Figura 2 - Nativo x Hibrido x Html5



Fonte - https://s3.amazonaws.com/dfc-wiki/en/images/c/c2/Native\_html5\_hybrid.png

Nativo – É Desenvolvido através de ferramentas específicas de um determinado sistema (exemplo Xcode ferramenta da Apple), esse tipo possui uma capacidade maior de utilizar as funcionalidades do dispositivo, obtendo uma melhor performance do aplicativo. Mas como ele é desenvolvido em uma linguagem nativa, ele apenas irá funcionar em seus respectivos aparelhos. (KORF, 2015).

Web – É desenvolvido através de linguagens padrões como HTML5, JavaScript e CSS. Isso gera uma portabilidade maior podendo funcionar em qualquer aparelho, é mais fácil de ser desenvolvido, pois as linguagens são mais simples do que as nativas, porém não exibe um grande desempenho. Como foi o caso do Facebook, em 2012 em uma conferência ([TechCrunch Disrupt San Francisco](http://techcrunch.com/events/disrupt-sf-2012/)) Mark Zuckerberg afirmou: “O maior erro que cometemos como empresa foi apostar muito no HTML5 em vez de [código] nativo...”. Isso gerou ao facebook um aplicativo lento e pouco confiável. (KORF, 2015; GHEDIN, 2012).

Híbrido – Combina a codificação nativa e web, esse tipo de desenvolvimento permite aproveitar as funcionalidades do aparelho (GPS, Câmera etc) utilizando HTML5. Através de frameworks como PhoneGap permite usar esse mesmo código HTML5 para gerar aplicativos multiplataformas. (KORF, 2015).

### ARMAZENAMENTO

O armazenamento pode ocorrer das seguintes formas:

(KORF, 2015).

Local: Onde os arquivos são guardados no próprio dispositivo, isso gera uma resposta mais rápida do aplicativo, porém os dados podem ser apagados e ou perdidos se trocar de aparelho.

Banco de dados: Onde os dados são armazenados fora do dispositivo, garantindo uma disponibilidade maior para o usuário, porém para obter esses dados é necessário uma conexão a internet.

# Teste de software

Teste de software é um processo de execução do *software* no qual está sendo desenvolvido a fim de verificar se atende as necessidades dos requisitos e se funcionou corretamente no ambiente para o qual a aplicação foi desenvolvida.

Os testes têm como seu principal objetivo encontrar possíveis falhas para que as mesmas sejam corrigidas pela equipe de desenvolvimento antes de ser disponibilizada para entrega final, visando assim uma maior confiança do produto.

Testar um *software* nada mais é do que verificar através de uma execução controlada se o comportamento está de acordo com o que foi especificado na documentação, cujo seu objetivo é descobrir o máximo de falhas com pouco esforço e passar o resultado para os desenvolvedores a fim de melhorarem seu desenvolvimento para alcançar os padrões estabelecidos pela empresa. (DIAS, 201?).

## CONCEITOS BASICOS

Precisamos definir primeiramente o conceito sobre Defeitos, Erros e Falhas. São eles:

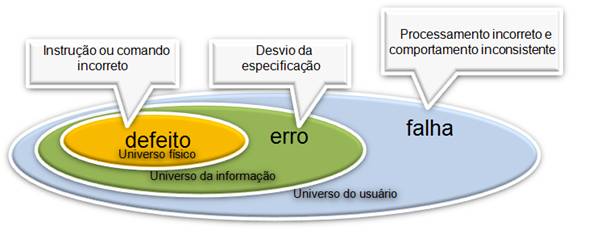
(DIAS, 201?).

**Defeito:** Ocorre quando existe uma instrução ou comando utilizado de forma incorreta pelo programador.

**Erro:** Ocorre quando a regra de negócio espera um retorno e o sistema não tem o retorno esperado.

**Falha:** Ocorre quando um cliente espera um comportamento do sistema e o mesmo não ocorre ou então se comporta de forma diferente.

Figura 3 - Defeito x Erro x Falha



Fonte - http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-teste-de-software/8035

## TIPOS DE TESTES

Testar a aplicação é o único jeito de ter certeza que ela está funcionando de forma apropriada, sem riscos de entregar um *software* defeituoso e cheio de falhas e para isso foram desenvolvidos vários tipos de testes:

(COSTA, 2014)

* Teste de Configuração – verifica se o *software* é compatível com o *hardware*;
* Teste de Instalação – verifica se a instalação é bem sucedida em diferentes configurações e circunstâncias;
* Teste de Integridade – verifica a robustez do *software* (o quanto ele é a prova de falhas);
* Teste de Segurança – verifica se o usuário acessa os dados de maneira segura;
* Teste Funcional – verifica se o *software* faz o prometido (se os requisitos funcionam de acordo com o planejado);
* Teste de Unidade – testa um componente separadamente (Exemplo: uma classe);
* Teste de Integração – verifica se os componentes estão funcionando de maneira conjunta;
* Teste de Volume – verifica o volume de dados durante um longo período de tempo;
* Teste de Performance – verifica a performance do *software* sob condições normais de uso (Teste de Carga), condições extremas (Teste de Stress) e capacidade de se manter funcional (Teste de Estabilidade);
* Teste de Usabilidade – verifica o manuseamento do *software* através do usuário final (funcionalidades, *layout*, etc.);
* Teste de Regressão – verifica se a atualização recente causou algum problema nas funcionalidades da aplicação;
* Teste de Manutenção – verifica se a mudança no hardware ou sistema operacional causou algum problema no *software*.

## FERRAMENTAS DE TESTE

Em alguns testes como o teste unitário, é viável que se use ferramentas para acelerar e melhorar o processo de teste, uma das ferramentas mais utilizadas é o Mock. Através de *frameworks* são criados Mock Objects, desenvolvidos para testar o código, podendo simular comportamento de objetos complexos, fazendo uma chamada remota. Porem o Mock possui limitações, se utilizado em excesso geram muita manutenção caso o código seja modificado.

Foi desenvolvido para várias linguagens e possui várias bibliotecas como:

* Java – JMockit, Mockito, EasyMock, JMock, MockCreator, MockLib e HibernateMock;
* .NET – NMockLib, Rhino Mocks, NMock e NMock2 TypeMock;
* PHP – SimpleTest, Yay! Mock, SnapTest e PHPUnit.

O Mock não é exclusivo apenas para essas linguagens, ele existe em muitas outras como: Ruby, Delphi, entre outros. (MEDEIROS, 201?).

Outra ferramenta de teste que podemos explorar é o Selenium, utilizado para testes em aplicações web, com a possibilidade de ser executado em qualquer navegador que suporta javascript. Atualmente é composto por algumas ferramentas como:

(SANTOS, 201?).

* Selenium-IDE – Essa ferramenta re-executa ações previamente feitas pelo tester;
* Selenium-RC – Utiliza linguagens de programação para testes mais complexos;
* Selenium-GRID – Permite a utilização de mais de uma máquina para a realização do teste, aumentando assim sua escalabilidade.

## AUTOMAÇÃO DE TESTES

A automação de testes possui a vantagem de executar vários testes de uma só vez, gerando documentação apropriada. Assim, focando a atenção dos profissionais de teste para tarefas mais importantes. Além disso, o custo atribuído a um teste já automatizado é muito mais baixo comparado ao manual, pois não é necessária a utilização de pessoas, sem contar que o tempo de execução é reduzido. Collins ainda aponta: “Alguns outros benefícios para o projeto incluem: melhorias no ciclo de vida de desenvolvimento de software , a confiabilidade do software , execução de testes de regressão, geração de documentos  e reprodução e monitoramento de falhas.”

Porém a automação de testes possui um investimento inicial alto, necessitando de profissionais capacitados, ferramentas específicas e tempo para a configuração do teste.

Alguns dos testes que podem ser automatizados são:

* Testes Unitários;
* Testes de Aceitação;
* Testes de Integração.

A automação é melhor aconselhada quando a equipe está disposta a faze-la, pois requer tempo e muito planejamento para ser efetuada. Ainda sim existem riscos como: mudanças de requisitos no sistema, que torna a manutenção cara, dificuldade na implementação dependendo da complexidade do sistema. Sem contar que não é possível automatizar totalmente os testes.

Essas dificuldades acabam que afastando gestores, pois requer muito esforço, dinheiro e possui benefícios a longo prazo. Porém se a empresa procura qualidade eu seus serviços, não se pode fugir da automação de testes. (COLLIS, 2013).

# CONCEITOS DE AUDITORIA

A auditoria de sistemas tem como principais funções adequar, revisar, avaliar e recomendar o aprimoramento dos controles internos utilizados nos sistemas de informação das empresas, além de avaliar a qualidade dos recursos humanos e tecnológicos que são envolvidos nos mesmos processos. (LYRA, 2008, *apud* SCHMIDT, 2006).

O processo de auditoria pode atuar em todos os níveis da empresa, sendo altamente recomendado participação nos níveis operacionais, táticos e estratégicos.

Atualmente, pode-se definir como objetivos globais da auditoria de sistemas da informação:

* **Integridade:** Quando há consistência nas transações de informações dentro da empresa. Os usuários têm confiança nas tomadas de decisões baseados nas informações fornecidas pelo sistema;
* **Confidencialidade:** As informações são filtradas por níveis, sendo que apenas o usuário autorizado pode ter acesso a determinado tipo de informação;
* **Privacidade:** O sistema restringe determinadas funções de acordo com o nível hierárquico de cada funcionário, buscando evitar o vazamento de informações;
* **Acuidade:** Busca a validação da informação, atestando a sua veracidade e evitando inserção de dados inconsistentes no banco de dados, gerando transações inválidas;
* **Disponibilidade:** Impedir que o sistema esteja fora do ar nos momentos em que for necessário, podendo gerar grandes prejuízos para a empresa;
* **Auditabilidade:** Disponibilizar *logs* de operação que permitam futuras trilhas de auditoria;
* **Versatilidade:** Fácil adaptação do usuário, interface amigável e de fácil entendimento, agilidade nos recursos de importação e exportação de dados, etc;
* **Manutenibilidade:** Plano de testes bem elaborado, código escrito de maneira coesa e limpa, de forma a evitar possíveis “remendos”, separação total dos ambientes de testes e de produção, de modo a evitar contaminação que possa comprometer o desempenho do sistema.

As empresas de auditoria de sistemas atualmente adotam a automatização dos processos, buscando grandes vantagens como a redução de custos, diminuição dos riscos através da melhora na qualidade dos processos, prospecção de novos clientes, utilizando tecnologias de automação disponíveis na própria empresa, a fim de buscar a satisfação plena do cliente e um diferencial competitivo no mercado. (LYRA, 2008).

O auditor de sistemas precisa ter conhecimentos em várias áreas diferentes, como contabilidade e tecnologia da informação, além de ter um sério embasamento cultural, pois além de ter de lidar com sistemas e processos, o auditor lida também com o lado humano da organização, já que ele é o responsável pelo levantamento de todos os processos e serviços executados por todos os funcionários da empresa, sem exceções.

Segundo o COBIT, a auditoria de sistemas da informação deve alcançar a excelência nos seguintes itens:

(ATTIE, 2011).

* Estrutura de Gerenciamento de Programa;
* Estrutura de Gerenciamento de Projeto;
* Abordagem de Gerenciamento de Projeto;
* Comprometimento dos Participantes;
* Escopo do Projeto;
* Fase de Início do Projeto;
* Planejamento do Projeto Integrado;
* Recursos do Projeto;
* Gerenciamento de Riscos do Projeto;
* Planejamento da Qualidade do Projeto;
* Controle de Mudanças no Projeto;
* Métodos de Planejamento de Garantia do Projeto;
* Avaliação, Relatórios e Monitoramento do Desempenho do Projeto;
* Conclusão do Projeto.

Nas auditorias feitas em um ambiente de tecnologia de informação, é comum o auditor trabalhar com três tipos comuns de abordagens: Abordagem ao redor do computador, através do computador e com o computador. (IMONIANA, 2008).

**ABORDAGEM AO REDOR DO COMPUTADOR**

A abordagem ao redor do computador se dá quando o auditor tem conhecimento de todos os controles operacionais e funções executados pelo sistema, sem necessariamente se atentar ao desempenho do sistema em si, como nível de processamento de dados, entre outros. A auditoria é feita através de documentos-fonte, onde são analisados todos os fluxos de entrada e saída de dados e processos de controle, sempre em formatos de linguagem legíveis por leigos em informática. Essa abordagem era muito utilizada antigamente, quando os processos empresariais não envolviam tanta tecnologia como nos dias atuais. Isso possuía vantagens, como não exigir conhecimento extenso de tecnologia de informação por parte do auditor, além de baixos custos. Como desvantagens, destacam-se a falta de informação de como os dados são processados e atualizados no sistema, tornando a auditoria incompleta e inconsistente; e a falta de conhecimento tecnológico do auditor, podendo perder o controle dos documentos-fonte em caso de futuras atualizações do sistema. (IMONIANA, 2008).

**ABORDAGEM ATRAVÉS DO COMPUTADOR**

Esse tipo de abordagem serve como um complemento à abordagem ao redor do computador, pois se aprofunda mais no controle de transações e manuseio de dados, transações comerciais, entre outros. Esses processos devem ser acompanhados pelo auditor através de um computador, uma vez que todos os tipos de entrada e saída de dados do sistema podem ser monitorados, gerando relatórios contundentes. Como vantagens, pode-se destacar o ganho de capacitação do auditor, que passa a verificar com mais frequência as áreas mais delicadas do sistema, que precisam de revisão constante. Por outro lado, as desvantagens são o custo de manutenção elevado e o risco de perdas muitas vezes incalculáveis, em caso de erro na operação. (IMONIANA, 2008).

**ABORDAGEM COM O COMPUTADOR**

Apesar da abordagem através do computador complementar a abordagem ao redor do computador, ainda há a chance de se produzirem relatórios incompletos. Pensando nisso, algumas empresas de auditoria e pesquisadores da área de contabilidade propuseram um método de auditar as tecnologias usando a abordagem com o computador, buscando a maior perfeição possível. O processo é compilado em partes, sendo que os objetivos são alcançados um a um:

(IMONIANA, 2008).

* Utilizar a capacidade lógica e aritmética do computador para verificar se cálculos de transações econômicas e financeiras são feitos corretamente;
* Verificar a capacidade dos cálculos estatísticos, a fim de propor integridade de dados de contas a receber, saldos disponíveis e indisponíveis de estoque, entre outros;
* Utilizar as capacidades de edição e classificação do sistema, a fim de ordenar e organizar registros contábeis. Como, por exemplo, um controle de estoque, onde podem ser feitas varreduras no sistema para separar produtos obsoletos no estoque de produtos com maior índice de vendas, a fim de gerar relatórios com dados mais contundentes e de fácil análise;
* A utilização das capacidades matemáticas do computador para fornecer listas de amostras de auditoria, assim como incluir resultados de auditorias feitas manualmente, para futuros relatórios de análise.

# CONCEITO DE INTEGRAÇÃO CONTINUA

O termo Integração Contínua tem origem na metodologia ágil XP (*Extreme Programing*), e consiste basicamente em uma integração de um código desenvolvido e/ou alterado ao projeto principal quase que ao mesmo tempo em que as funcionalidades são criadas. Essa integração pode ocorrer inúmeras vezes em um único dia, e auxilia na manutenção do código final, já que sempre é verificado se as alterações e funcionalidades não geraram defeitos no projeto final durante suas integrações. Essa integração pode ser feita manualmente ou com ferramentas automatizadas, como o *Jenkins*, *Hudson*, entre outros. (FOWLER, 2007).

Segundo Martin Fowler (2007):

“Integração Contínua é uma pratica de desenvolvimento de software onde os membros de um time integram seu trabalho frequentemente, geralmente cada pessoa integra pelo menos diariamente – podendo haver múltiplas integrações por dia. Cada integração é verificada por um build automatizado (incluindo testes) para detectar erros de integração o mais rápido possível. Muitos times acham que essa abordagem leva a uma significante redução nos problemas de integração e permite que um time desenvolva um software coeso mais rapidamente.”.

Para que a integração seja estruturada entre a equipe de trabalho, é necessário um controle centralizado de versão a fim de agilizar a identificação de possíveis erros de integração, citados acima por Fowler (2007). O controle de versão se faz bastante necessário em metodologias ágeis, uma vez que a equipe trabalha em conjunto durante todo o tempo, compartilhando dados e informações, e sempre tendo acesso as últimas versões validadas, aos usuários que efetuaram as atualizações e evitando retrabalho em códigos já validados. O controle de versão tem ferramentas que auxiliam o trabalho conjunto, como o CVS, *Subversion* e GIT. Elas trabalham com um servidor central encarregado pelo versionamento do sistema. Assim, clientes têm acesso ao código, podendo visualizar, alterar e fazer o *upload* de novos códigos e/ou alterações, sempre que necessário.

Basicamente, o controle de versão funciona da seguinte forma: O desenvolvedor faz um código e efetua um *build* antes de integrar seu código com a base principal. Em seguida, o sistema de versão sincroniza o código com a base principal do projeto. Algumas metodologias ditam que o *build* deve ser feito constantemente, com a justificativa de que é mais difícil localizar erros em uma grande integração do que em uma integração pequena. É recomendado que o processo de *build* seja feito de forma automatizado para evitar problemas futuros na integração. (SOMMERVILLE, 2011).

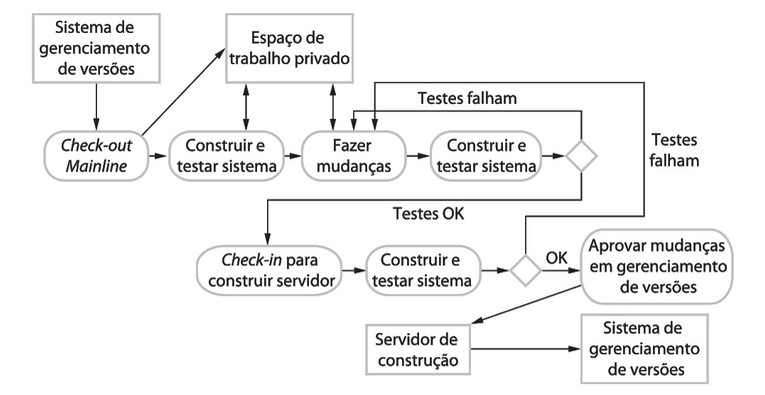
Segundo Sommerville (2011):

“As etapas na integração contínua são:

1. Realizar *check-out* do sistema de *mainline* do sistema de gerenciamento de versões no espaço de trabalho privado do desenvolvedor;
2. Construir o sistema e executar testes automatizados para assegurar que o sistema construído passe em todos os testes. Caso contrário, a construção é interrompida e você deve informar quem realizou o último *check-in* de sistema de *baseline*. Eles são responsáveis por reparar o problema;
3. Fazer as mudanças para os componentes de sistema;
4. Construir o sistema no espaço de trabalho privado e executar novamente os testes de sistema. Se os testes falharem, continue a editar;
5. Uma vez que o sistema tenha passado nos testes, verificar isso no sistema construído, mas não aprovar como uma nova *baseline* de sistema;
6. Construir o sistema no servidor de construção e executar os testes. Você precisará fazer isso para o caso de outras pessoas terem modificados os componentes depois que você tenha feito o *check-out* do sistema. Se esse for o caso, conferir os componentes que falharam e editá-los, de maneira que os testes passem em seu espaço de trabalho privado;
7. Se o sistema passar em seus testes sobre o sistema de construção, então, aprovar as mudanças feitas como uma nova *baseline* no *mainline* de sistema.”.

A automatização das rotinas de integração tem vários benefícios, como menos erros por parte da equipe de trabalho, redução de riscos devido à facilidade na localização de erros, uma vez que não se é integrado tudo no final do projeto, e sim várias pequenas integrações no decorrer dele, evitando acúmulo de bugs e contribuindo para um sistema bem desenvolvido e coeso. (SOMMERVILLE, 2011).

Figura 4 - Esquema de Integração Continua



Fonte: Sommerville, 2011.

# CONCEITOS DE PADRÕES DE PROJETO

Atualmente para se desenvolver um sistema que seja bem sucedido naquilo que foi proposto, é preciso criar as melhores soluções através de processos detalhados de levantamento de requisitos funcionais e não funcionais, além de executar inúmeros testes, sempre buscando identificar o problema o mais cedo possível para não comprometer a entrega final do projeto, ou até mesmo entregar um projeto falho, que comprometa sua execução e prejudique a rotina do usuário. Um sistema forte também precisa ser bem documentado, a fim de futuras consultas, pois embora vários projetos se diferenciem entre si, as soluções para os eventuais problemas encontrados podem ser as mesmas, economizando tempo e até mesmo possíveis custos.

Os padrões de projetos surgiram em 1977 quando Christopher Alexander sugeriu projetos semelhantes de prédios que eram ao mesmo tempo agradáveis e eficazes. Servia para descrever um problema e apresentar uma solução que poderia ser utilizada em diferentes contextos, gerando assim um ganho de experiência e conhecimento. (GAMMA, 2000).

Atualmente os padrões de projetos têm chamado a atenção dos projetistas de *software*, pois permitem o reaproveitamento de soluções para os projetos, além da reutilização do código. Usando a orientação a objetos, os padrões de projetos reduzem a complexidade do processo de projetar um *software*. Os padrões geralmente atendem a quatro características obrigatórias:

1. Devem possuir um nome, que descreva o problema, as soluções e consequências. Um nome permiti definir o vocabulário a ser utilizado pelos projetistas e desenvolvedores em um nível mais alto de abstração.
2. Todo padrão deve relatar de maneira clara a qual(is) problema(s) ele deve ser aplicado, ou seja, quais são os problemas que quando inserido em um determinado contexto o padrão conseguirá resolve-lo. Alguns podendo exigir pré-condições.
3. Solução descreve os elementos que compõem o projeto, seus relacionamentos, responsabilidades e colaborações. Um padrão deve ser uma solução concreta, ele deve ser exprimido em forma de gabarito (algoritmo) que, no entanto pode ser aplicado de maneiras diferentes.
4. Todo padrão deve relatar quais são as suas consequências para que possa ser analisada a solução alternativa de projetos e para a compreensão dos benefícios da aplicação do projeto.

O mais importante sobre os padrões é que eles são soluções aprovadas. Cada catálogo inclui apenas padrões que foram considerados úteis por diversos desenvolvedores em vários projetos. Os padrões catalogados também são bem definidos; os autores descrevem cada padrão com muito cuidado e em seu próprio contexto, portanto será fácil aplicar o padrão em suas próprias circunstâncias. Eles também formam um vocabulário comum entre os desenvolvedores.

Os padrões devem ser usados como um guia, e não como estratégica específica. Os catálogos geralmente trazem um código-fonte como exemplo, mas não devem ser considerados como estratégias definitivas. Basicamente, os padrões ajudam com o que e como, mas não com por que ou quando.

Uma das tarefas mais difíceis é escolher um padrão que possui a melhor solução para um projeto, sem comprometer o resultado final. A escolha deve ser bastante minuciosa, e deve levar em consideração como determinado padrão vai solucionar os problemas existentes, além de examinar os princípios do padrão, como ele opera e o que ele faz de fato. (LEITE, 2014).

São exemplos de padrões de projetos:

* *Abstract Factory* – Fornece uma interface para criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas. Deve ser aplicado quando se deseja separar a aplicação da classe concreta, seja ele um componente ou até mesmo um *framework* específico;
* *Factory Method –* Define uma interface para criar objetos deixando as subclasses decidirem quais classes instanciar. É bastante usado por aqueles que trabalham no desenvolvimento de *frameworks*, que utilizam classes abstratas para manter relacionamentos entre os objetos;
* *Singleton* – Esse padrão garante que um objeto sempre terá uma única instância para todo o escopo de uma aplicação. É utilizado quando se é necessário manter em um sistema apenas uma instância de objeto com um ponto de acesso conhecido;
* *Adapter* – Converte a interface de uma classe por outra desejada pelos clientes, permitindo o trabalho em conjunto de interfaces antes consideradas incompatíveis. É utilizado quando se usa uma classe existente, mas que não tenha a interface requerida, e quando classes reutilizáveis são criadas, cooperando com classes não-relacionadas ou não previstas, com interface que são inicialmente compatíveis.

# EMPREENDEDORISMO SOCIAL

No meio de tantas formas de empreendedorismo existentes e de tantos perfis de empreendedores atualmente, o perfil do empreendedor social é interessante. Ele se diferencia dos demais por não ter como foco principal apenas o capital, e sim com o comprometimento de tornar o mundo em que vivemos um melhor lugar. São pessoas acima de tudo sonhadoras, que buscam extinguir o desequilíbrio entre a sociedade como um todo causado pelo capitalismo predominante do século XXI. O empreendedorismo social pode ser considerado um derivado do empreendedorismo inovador e criativo, que também é importante na geração de valor dentro de uma organização. O empreendedor social dentro de uma organização contribui para a sua responsabilidade social, o que hoje em dia pode ser traduzido como alta competitividade no mercado. É necessário quase sempre ser criativo, altamente inovador e buscar sempre soluções autossustentáveis, buscando envolver várias pessoas dentro de uma sociedade. (COSTA, 2015).

Atualmente são frequentes o surgimento de empresas criadas por empreendedores socialmente responsáveis. Isso ocorre devido ao aumento da demanda dessas organizações, pois cada vez mais os clientes estão deixando de optar por empresas administradas por executivos tradicionais, que tem como único foco e objetivo a geração de lucro do seu capital. Os empreendedores socialmente responsáveis têm papel extremamente ativo dentro de uma sociedade, especialmente em países subdesenvolvidos, onde são frequentemente admirados e servem como inspiração para as comunidades em que atuam, sustentando a ideia de que o estado de riqueza pessoal vai muito além do que apenas lucros financeiros, e sim com o envolvimento em causas nobres, se comprometendo com o próximo. São pessoas visionárias, que conseguem visualizar oportunidades mesmo na mais extrema dificuldade e que são movidos, acima de tudo, pelo sentimento de realização ao ajudar o próximo e sentir que a sua contribuição está melhorando o mundo de alguma forma. (ARANTES, 2014).

Figura 5 - Empreendedorismo empresarial x Empreendedorismo social.



Fonte - Costa, Cunha, Dias e Souza, 2002.

A figura mostra que há importantes diferenças entre os dois tipos de empreendedorismo. Nota-se que o empreendedorismo social visa o coletivo, sempre produzindo serviços benéficos à comunidade, exemplo das ONGs (Organizações Não Governamentais), que dependem exclusivamente da filantropia e de doações para se manter. As ONGs são responsáveis por criação de projetos socioeducativos, muitas vezes visando o combate à pobreza e ao analfabetismo.

Além das ONGs, atualmente nota-se que muitas organizações têm recorrido a ajuda de outras instituições para serviços como consultoria e treinamentos de executivos para trabalhar com questões sociais, fazendo desde trabalho voluntário até doações e patrocínios a ONGs e causas sociais. (ARANTES, 2014).

# Relatório técnico

## DESCRIÇÃO DA EMPRESA

Figura 6 - Logotipo da Empresa.



A iCoding é uma empresa desenvolvedora de soluções ágeis para pequenas, médias e grandes empresas. Nossa missão é analisar e identificar as dificuldades encontradas nos processos de gerenciamento e desenvolvimento, e implementar novas soluções em busca de agilidade nos processos internos e externos, assim como as rotinas administrativas e operacionais, buscando eliminar possíveis riscos e obstáculos no gerenciamento da empresa através do gerenciamento de custos, utilização de ferramentas e tecnologia apropriadas e com uma equipe de desenvolvedores, analistas de requisitos e gerentes de projetos altamente qualificada, buscando uma efetiva integração no planejamento estratégico das organizações.

## OBJETIVO DO SISTEMA

Diminuir o tempo da rotina de entrada e saída de caminhões e contêineres do terminal através de um cadastro informatizado de motoristas, carretas e contêineres. Assim, funcionários terão maior agilidade na localização de veículos e contêineres dentro do pátio, assim como facilitará a busca por movimentações anteriores através de registros salvos no banco de dados do *software*.

## CENÁRIO ATUAL

Em um terminal de armazenagem de contêineres de pequeno porte, foi analisado o fluxo de rotinas implementadas pelos funcionários. No total, cinco (5) funcionários são encarregados de verificar a entrada e saída de caminhões no *gate* (portão), cadastrar a entrada dos caminhões, motoristas e contêineres, vistoriar e procurar por possíveis avarias em todos os contêineres, localizar a quadra onde se encontra o contêiner que será removido ou onde será alocado um contêiner inserido no pátio, um funcionário atuando como líder, supervisionando os demais funcionários e efetuando a alteração de contêineres cadastrados incorretamente. Foi identificado que os funcionários fazem o controle de entrada e saída e cadastro por meio de planilhas simples e, como a tarefa de atualização ou alteração de dados é delegada a apenas uma pessoa, no caso o líder da equipe, a produtividade do terminal é comprometida, causando atrasos na entrada de caminhões no pátio.

## CENÁRIO PROPOSTO

Um sistema totalmente reformulado, onde haverá um funcionário para verificação de avarias, que encaminhará a minuta para o operador do *gate* que, conforme seu nível de acesso ao sistema, delegado pelo líder, possuirá privilégios de adição e edição de registros e transferirá as informações da minuta para o sistema ProPátio. Contará também com quatro servidores, sendo dois pra o Banco de Dados e dois para o Sistema, a necessidade de se ter dois servidores para cada um é ter o balanceamento de carga.

Utilizando 50% de cada um, com isso em caso de queda do servidor o um assume a carga do outro e visse versa, com isso garantimos maior disponibilidade do sistema, evitando fila de caminhões em ambos os lados do terminal, tanto o de dentro quanto o de fora.

A Empresa contratante fica responsável pela conexão de internet, sugerimos no mínimo dois *links* trabalhando em *Load-Balanced* (Caso um *link* fique fora do ar, o outro assume, ou divide a carga de navegação da empresa em quanto os dois estão ativos).

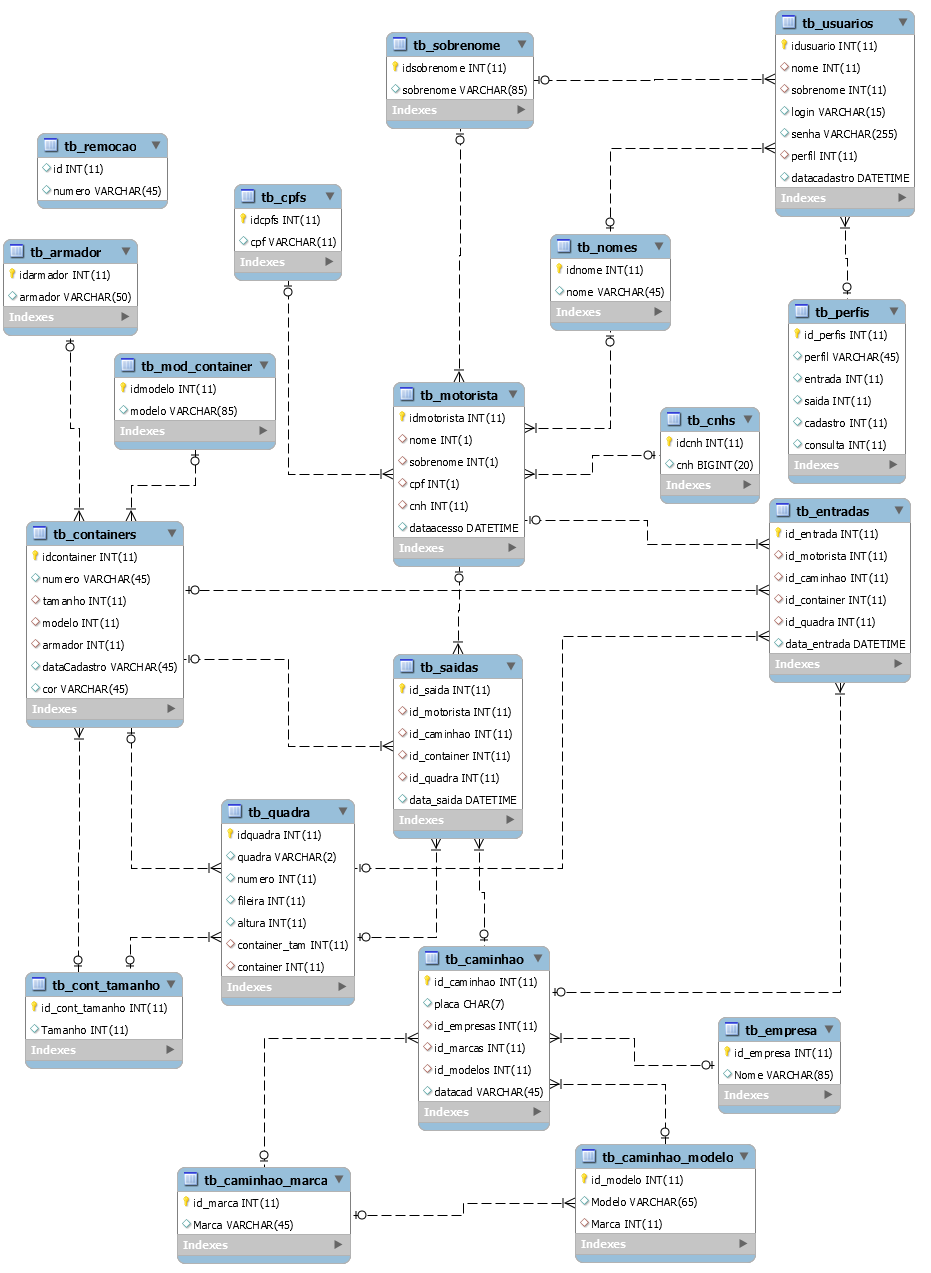
Todos os computadores que contiverem acesso á *internet* poderá acessar o sistema desde que os usuários tenham acesso ao sistema, dessa forma poderá obter informações registradas na base de dados a fim de consultar, editar, adicionar ou até mesmo excluir informações.

Nosso software será desenvolvido em JAVA 8, com o framework MVC VRaptor 4 de alta produtividade para um desenvolvimento Java Web, com *Design Patterns*CDI (Context Dependency Injection/ Injeção de Dependência e Contextos) junto com Banco de Dados MySQL e servidor de aplicação Wildfly8.

O *Design Patterns* CDI visa remover dependências desnecessárias entre as classes, ou torna-las mais leves e ter um *design* de *software* que seja fácil de manter e evoluir.

## MER DO BANCO DE DADOS

Figura 7 - MER do Banco de Dados.



|  |  |
| --- | --- |
| **UC1 – CADASTRAR ENTRADA** | |
| **OBJETIVOS:** | Permitir que o ator cadastre a entrada do caminhão, motorista e contêiner. |
| **ATORES:** | Operador de Gate |
| **PRÉ-CONDIÇÕES:** | O Ator seleciona a opção CADASTRAR ENTRADA no sistema. |
| **FLUXO PRINCIPAL:** | 1. O Sistema apresenta a tela de CADASTRO DE ENTRADA com os seguintes campos:    1. **Dados do Contêiner**  * Número; **[A1]** * Armador; * Cores; * Modelo; * Tamanho; * Quadra; * Posições;   1. **Dados do Motorista** * CPF; **[A2]** * Nome; * Sobrenome; * CNH;   1. **Dados do Caminhão** * Placa; [A3] * Empresa; * Marca; * Modelo;   **O Sistema ainda apresenta os seguintes botões:**   * Limpar; [A4] * Gravar; [A5]  1. O ator preenche os campos apresentados na tela; 2. O ator seleciona a opção GRAVAR; **[A5]** 3. O sistema verifica se os dados informados são válidos; **[RN1]** 4. O sistema apresenta a MESMA TELA, contendo o aviso de “ENTRADA CADASTRADA COM SUCESSO!”, com todos os campos em forma de “SOMENTE LEITURA” para conferencia dos dados; 5. O sistema grava os dados da entrada; 6. O sistema apresenta o botão OK; 7. O ator seleciona o botão OK; 8. O sistema retorna para o passo **1** do fluxo principal; 9. Caso de uso encerrado; |

## CASOS de USO

Tabela 1 – Cadastrar Entrada.

|  |  |
| --- | --- |
| **FLUXO ALTERNATIVO:** | **[A1]** – Numero   1. O ator digita o NUMERO DO CONTÊINER e pressiona a tecla ENTER no teclado; 2. O sistema busca o NUMERO digitado no banco de dados; 3. Caso o sistema encontre o NUMERO, preenche automaticamente os dados: TAMANHO, MODELO, ARMADOR e QUADRA; 4. Caso não encontre, o sistema apresenta uma mensagem de alerta informando que o contêiner não se encontra no pátio. 5. O sistema retorna ao passo **2** do Fluxo Principal;   **[A2]** – CPF   1. O ator digita o CPF DO MOTORISTA e pressiona a tecla ENTER no teclado; 2. O sistema busca o CPF digitado no banco de dados; 3. Caso o sistema encontre o CPF, preenche automaticamente os campos: NOME, SOBRENOME e CNH; 4. Caso no sistema não encontre o CPF, ele troca a seleção para o campo NOME, para ser preenchido pelo ator. 5. O sistema retorna ao passo **2** do Fluxo Principal;   **[A3]** – Placa   1. O ator digita a PLACA DO CAMINHÃO e pressiona a tecla ENTER no teclado; 2. O sistema busca a placa digitada no banco de dados; 3. Caso o sistema encontre a placa, preenche automaticamente os campos: EMPRESA, MARCA e MODELO; 4. Caso no sistema não encontre a PLACA, ele troca a seleção para o campo EMPRESA, para ser preenchido pelo ator. 5. O sistema retorna ao passo **2** do Fluxo Principal;   **[A4]** - LIMPAR   1. O ator seleciona o botão LIMPAR; 2. O sistema limpa todos os campos editáveis; 3. O sistema retorna ao passo **1** do Fluxo Principal;   **[A5]** - GRAVAR   1. O ator seleciona o botão GRAVAR; 2. O sistema verifica se os dados informados são válidos; **[RN1]** 3. Caso os campos não estejam preenchidos corretamente, o sistema apresenta uma mensagem de alerta, mostrando os campos que não foram preenchidos ou estão preenchidos incorretamente. 4. O sistema retorna ao passo **2** do Fluxo Principal. |
| **PÓS-CONDIÇÕES** | **[RN1]** TODOS os campos são obrigatórios. |

|  |  |
| --- | --- |
| **UC2 – CADASTRAR SAÍDA** | |
| **OBJETIVOS:** | Permitir o ator cadastrar a saída do contêiner do pátio. |
| **ATORES:** | Operador de Gate |
| **PRÉ-CONDIÇÕES:** | O Ator seleciona a opção CADASTRAR SAÍDA no sistema. |
| **FLUXO PRINCIPAL:** | 1. O Sistema apresenta a tela de CADASTRO DE SAÍDA com os seguintes campos:    1. **Dados do Contêiner**  * Número; **[A1]**   1. **Dados do Motorista** * CPF; **[A2]** * Nome; * Sobrenome; * CNH;   1. **Dados do Caminhão** * Placa; **[A3]** * Empresa; * Marca; * Modelo;   **O Sistema ainda apresenta os seguintes botões**:   * Limpar; **[A4]** * Gravar; **[A5]**  1. O ator preenche os campos apresentados na tela; 2. O ator seleciona a opção GRAVAR; **[A5]** 3. O sistema verifica se os dados informados são válidos; **[RN1]** 4. O sistema apresenta a MESMA TELA, contendo o aviso de “SAÍDA CADASTRADA COM SUCESSO!”, com todos os campos em forma de “SOMENTE LEITURA” para conferencia dos dados; 5. O sistema grava os dados da saída; 6. O sistema apresenta o botão OK; 7. O ator seleciona o botão OK; 8. O sistema retorna para o passo **1** do fluxo principal; 9. Caso de uso encerrado; |

Tabela 2 – Cadastrar Saída.

|  |  |
| --- | --- |
| **FLUXO ALTERNATIVO:** | **[A1] – Numero**   1. O ator digita o NUMERO DO CONTÊINER e pressiona a tecla ENTER no teclado; 2. O sistema busca o NUMERO digitado no banco de dados; 3. Caso o sistema encontre o NUMERO, preenche automaticamente os dados: TAMANHO, MODELO, ARMADOR e QUADRA; 4. Caso não encontre, o sistema apresenta uma mensagem de alerta informando que o contêiner não se encontra no pátio. 5. O sistema retorna ao passo **2** do Fluxo Principal;   **[A2] – CPF**   1. O ator digita o CPF DO MOTORISTA e pressiona a tecla ENTER no teclado; 2. O sistema busca o CPF digitado no banco de dados; 3. Caso o sistema encontre o CPF, preenche automaticamente os campos: NOME, SOBRENOME e CNH; 4. Caso no sistema não encontre o CPF, ele troca a seleção para o campo NOME, para ser preenchido pelo ator. 5. O sistema retorna ao passo **2** do Fluxo Principal;   **[A3] – PLACA**   1. O ator digita a PLACA DO CAMINHÃO e pressiona a tecla ENTER no teclado; 2. O sistema busca a placa digitada no banco de dados; 3. Caso o sistema encontre a placa, preenche automaticamente os campos: EMPRESA, MARCA e MODELO; 4. Caso o sistema não encontre a PLACA, ele troca a seleção para o campo EMPRESA, para ser preenchido pelo ator. 5. O sistema retorna ao passo **2** do Fluxo Principal;   **[A4] - LIMPAR**   1. O ator seleciona o botão LIMPAR; 2. O sistema limpa todos os campos editáveis; 3. O sistema retorna ao passo **1** do Fluxo Principal;   **[A5] - GRAVAR**   1. O ator seleciona o botão GRAVAR; 2. O sistema verifica se os dados informados são válidos; **[RN1]** 3. Caso os campos não estejam preenchidos corretamente, o sistema apresenta uma mensagem de alerta, mostrando os campos que não foram preenchidos ou estão preenchidos incorretamente. 4. O sistema retorna ao passo **2** do Fluxo Principal. |
| **PÓS-CONDIÇÕES:** | **[RN1]** Todos os campos são obrigatórios. |

|  |  |
| --- | --- |
| **UC3 – CADASTRAR USUÁRIO** | |
| **OBJETIVOS:** | Permitir o ator cadastrar usuário para acesso a opções do sistema. |
| **ATORES:** | Gerente |
| **PRÉ-CONDIÇÕES:** | O Ator seleciona a opção ADCIONAR USUÁRIO no link MENU dentro do sistema. |
| **FLUXO PRINCIPAL:** | 1. O Sistema apresenta a tela de CADASTRO DE USUÁRIO com os seguintes campos:  * Nome; * Perfil; * Data de Cadastro(Somente Leitura); * Login; * Senha; * Confirmar Senha;   O Sistema ainda apresenta os seguintes botões:   * Limpar; **[A1]** * Gravar; **[A2]**  1. O ator preenche os campos apresentados na tela; 2. O ator seleciona a opção GRAVAR; **[A2]** 3. O sistema verifica se os dados informados são válidos; **[RN1]** 4. O sistema apresenta a MESMA TELA, contendo o aviso de “USUÁRIO CADASTRADO COM SUCESSO!”, com todos os campos em forma de “SOMENTE LEITURA” para conferencia dos dados; 5. O sistema grava os dados Do usuário; 6. O sistema apresenta o botão OK; 7. O ator seleciona o botão OK; 8. O sistema apresenta a tela de listagem dos usuários; 9. Caso de uso encerrado; |
| **FLUXO ALTERNATIVO:** | **[A1]** - LIMPAR   1. O ator seleciona o botão LIMPAR; 2. O sistema limpa todos os campos editáveis; 3. O sistema retorna ao passo **1** do Fluxo Principal;   **[A2]** - GRAVAR   1. O ator seleciona o botão GRAVAR; 2. O sistema verifica se os dados informados são válidos; **[RN1]** 3. Caso os campos não estejam preenchidos corretamente, o sistema apresenta uma mensagem de alerta, mostrando os campos que não foram preenchidos ou estão preenchidos incorretamente. 4. O sistema retorna ao passo **2** do Fluxo Principal. |
| **PÓS-CONDIÇÕES:** | **[RN1]** Todos os campos são obrigatórios. |

Tabela 3 – Cadastrar Usuário.

## TELAS DO SISTEMA

Figura 8 - Tela de Login.

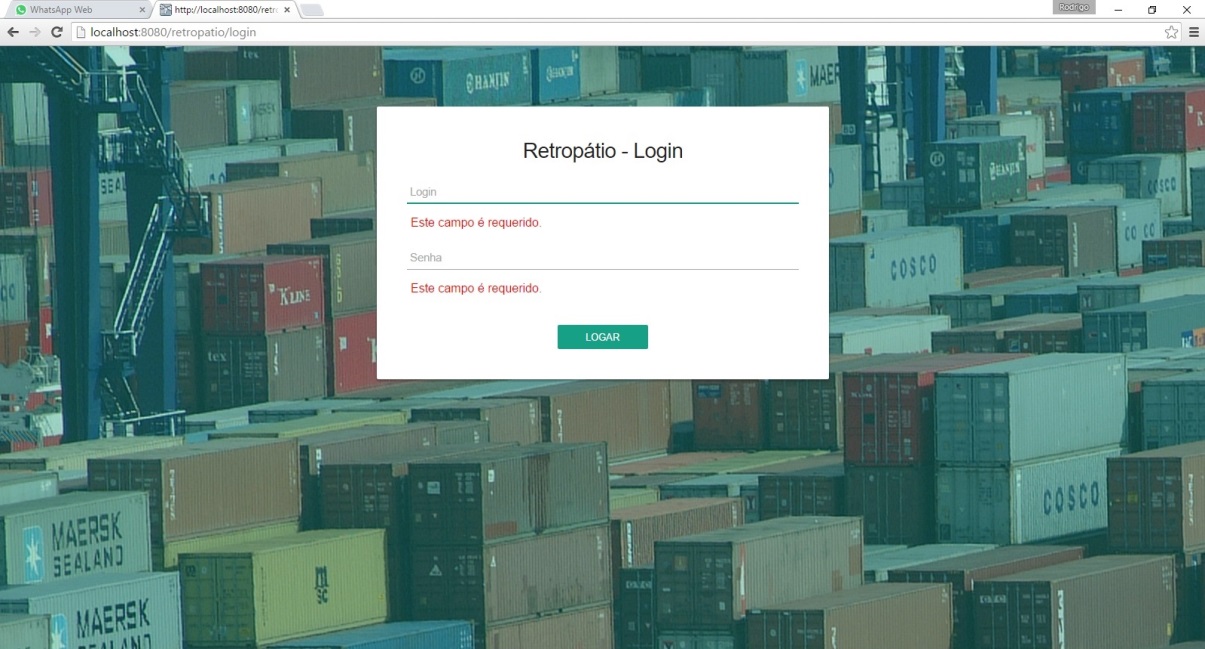


Figura 9 - Tela Principal.

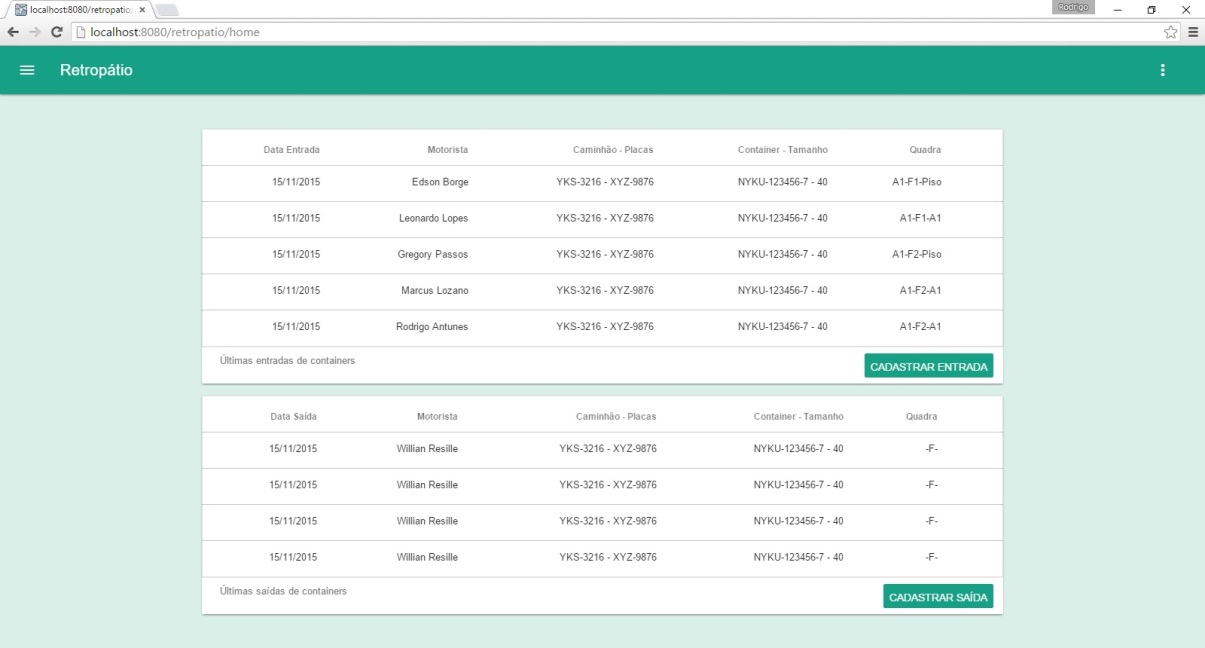


Figura 10 – Tela de Fluxo de Entrada.

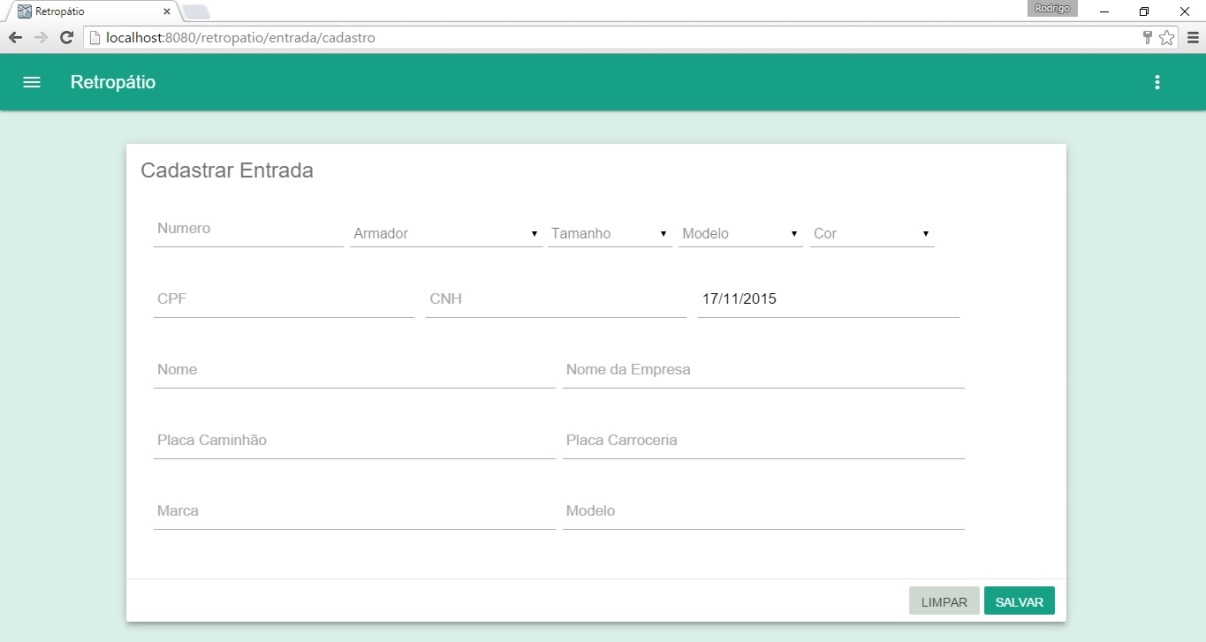


Figura 11 – Tela de Fluxo de Saída.

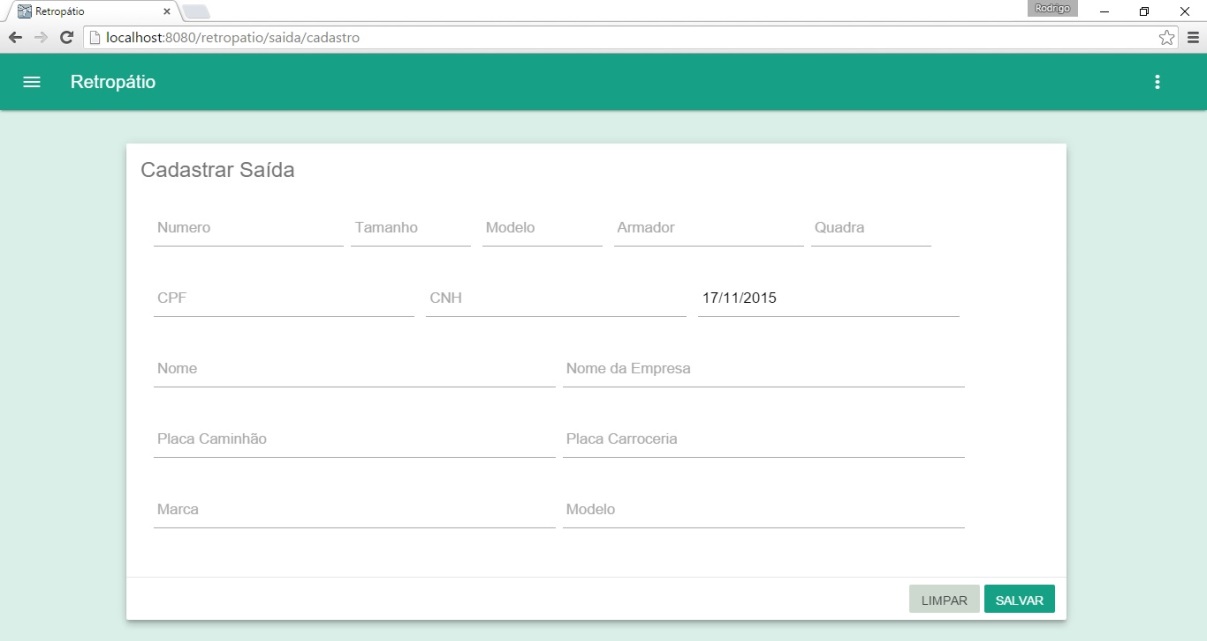


Figura 12 - Menu do Sistema.

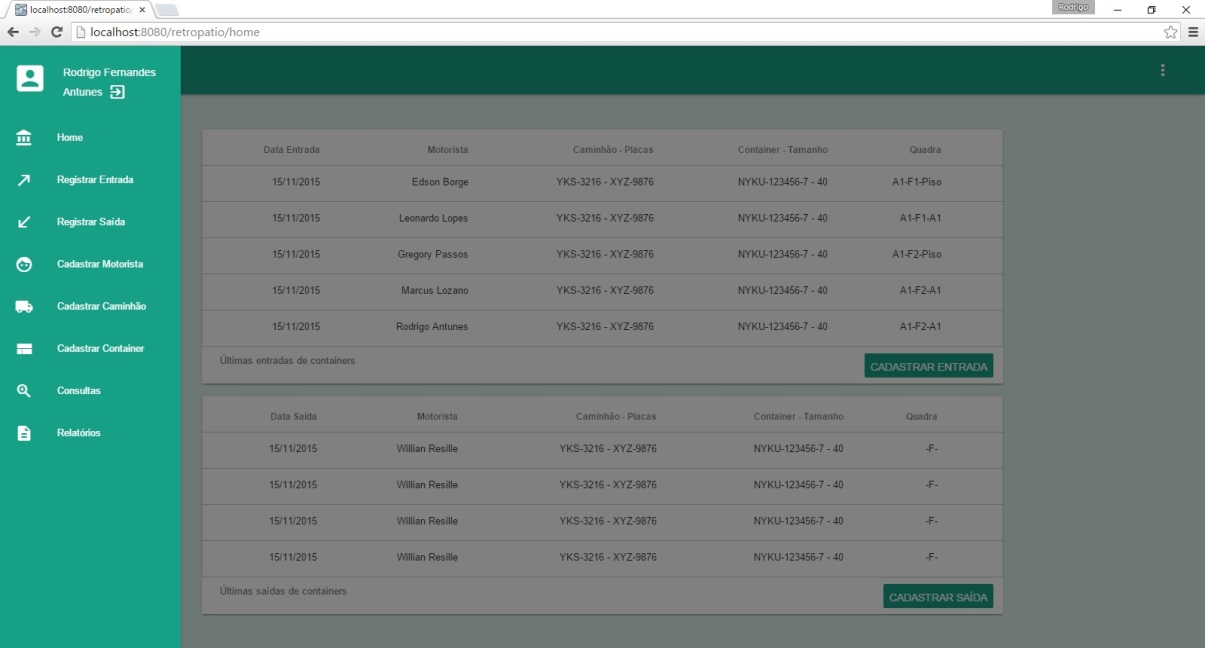


Figura 13 - Menu de Configurações.

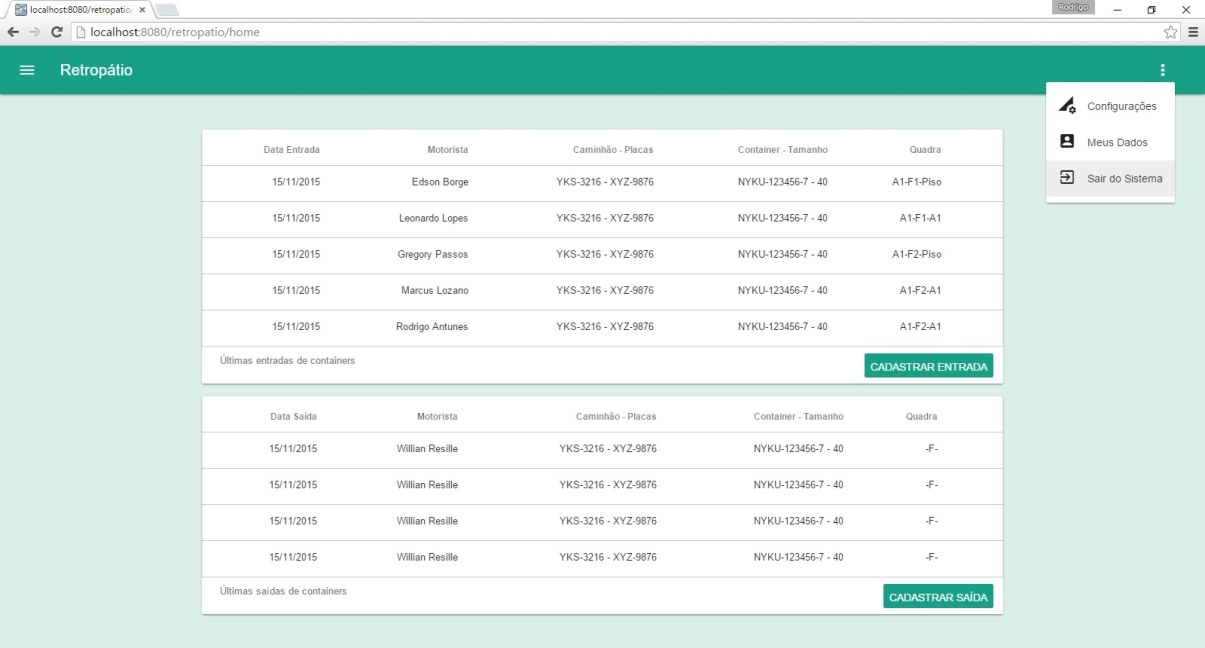


Figura 14 - Tela de Cadastro de Motorista.

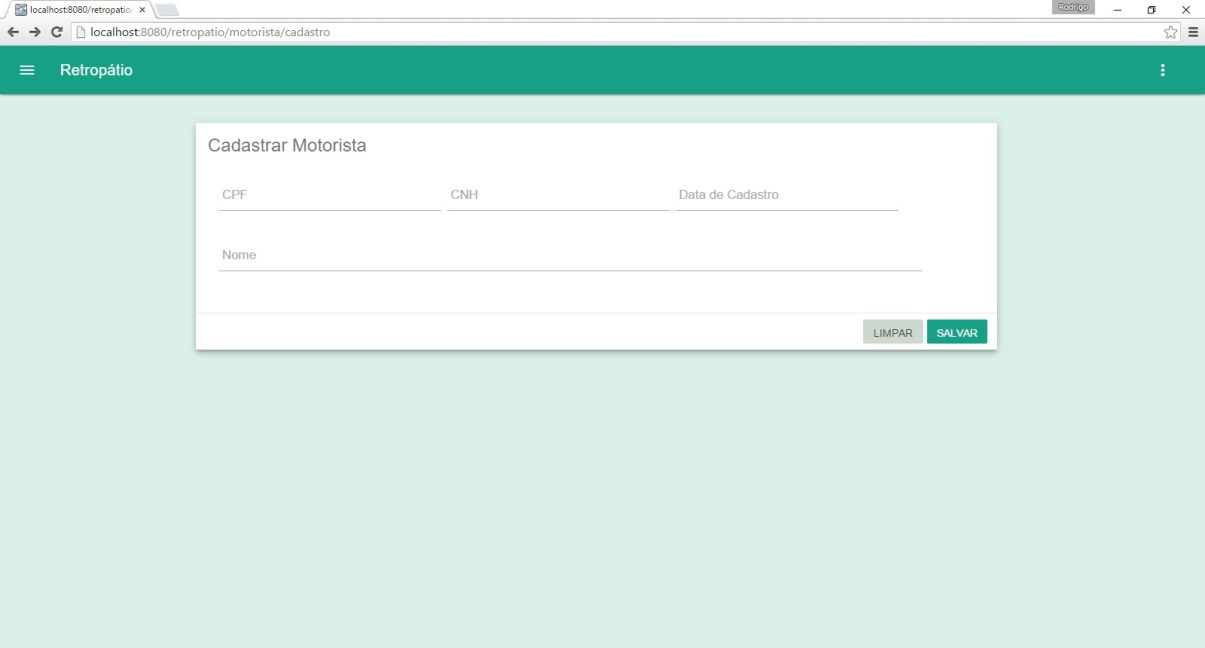


Figura 15 - Tela de Cadastro de Caminhão.

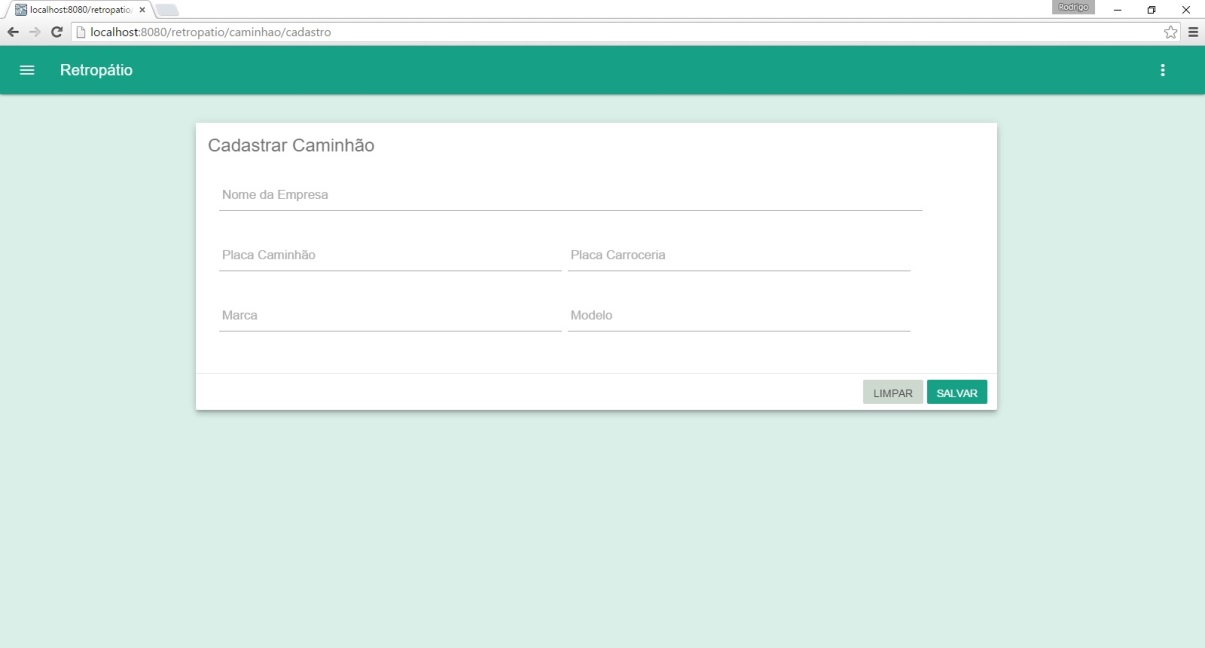


Figura 16 - Tela de Cadastro de Container.

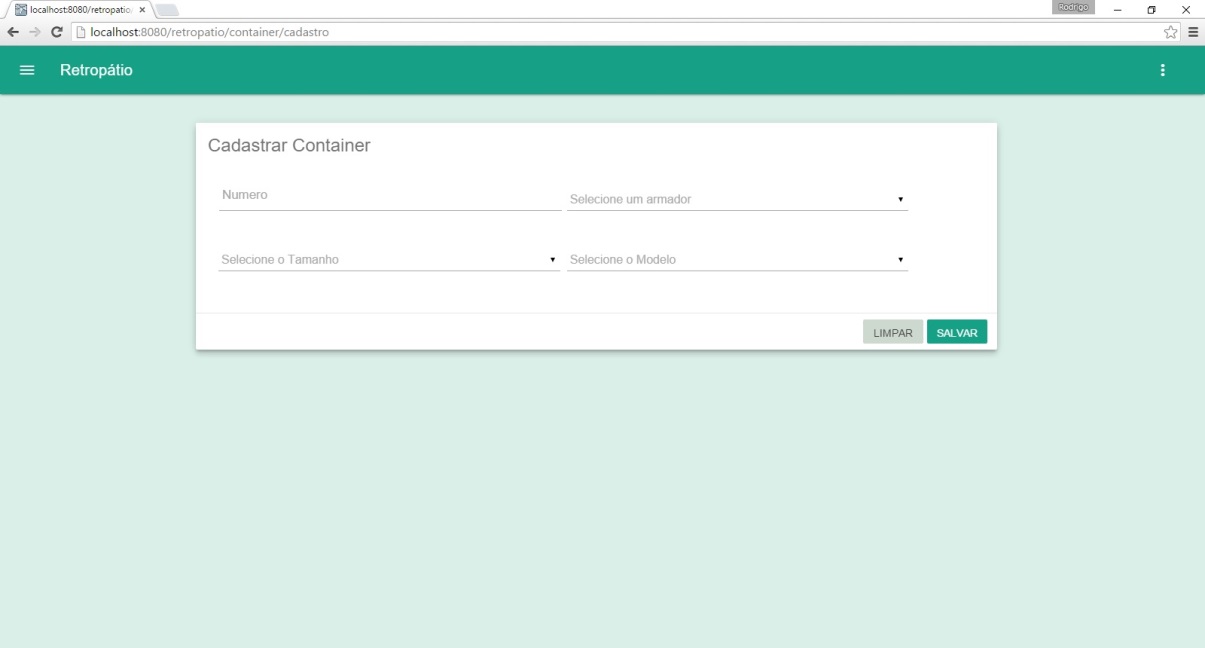
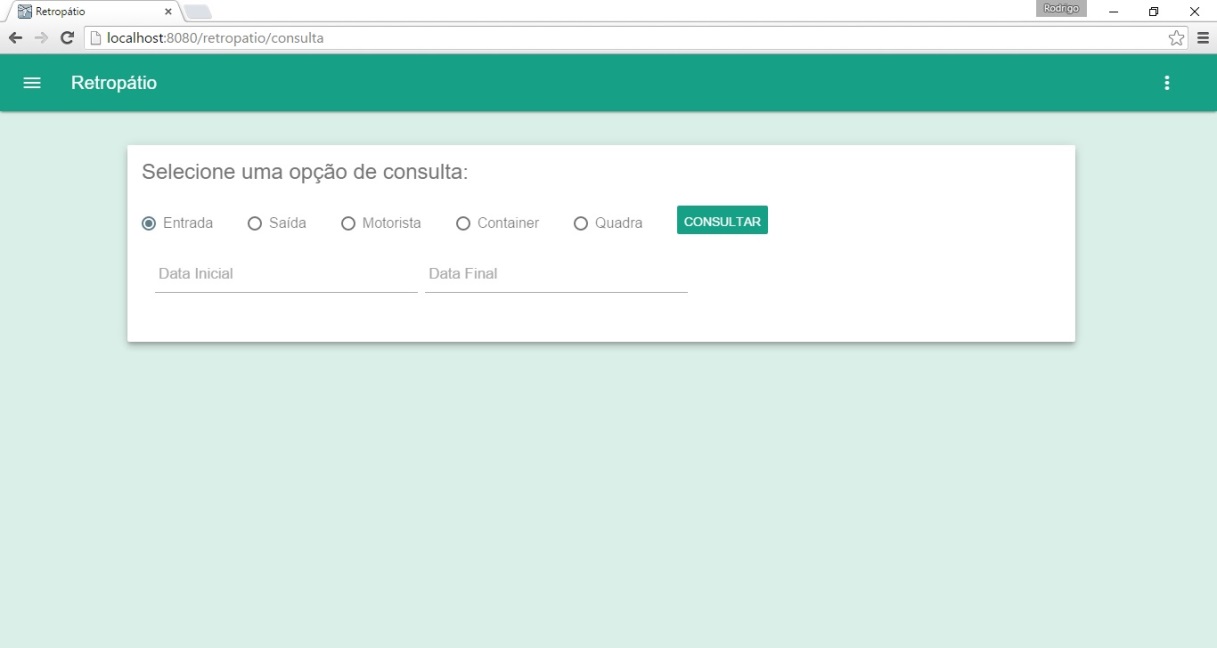


Figura 17 - Tela de Consulta.



## CRONOGRAMA

Figura 18 - Lista de Tarefas

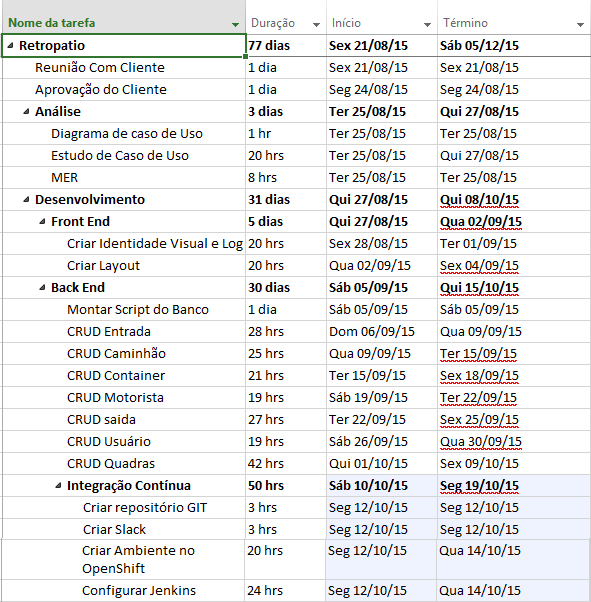
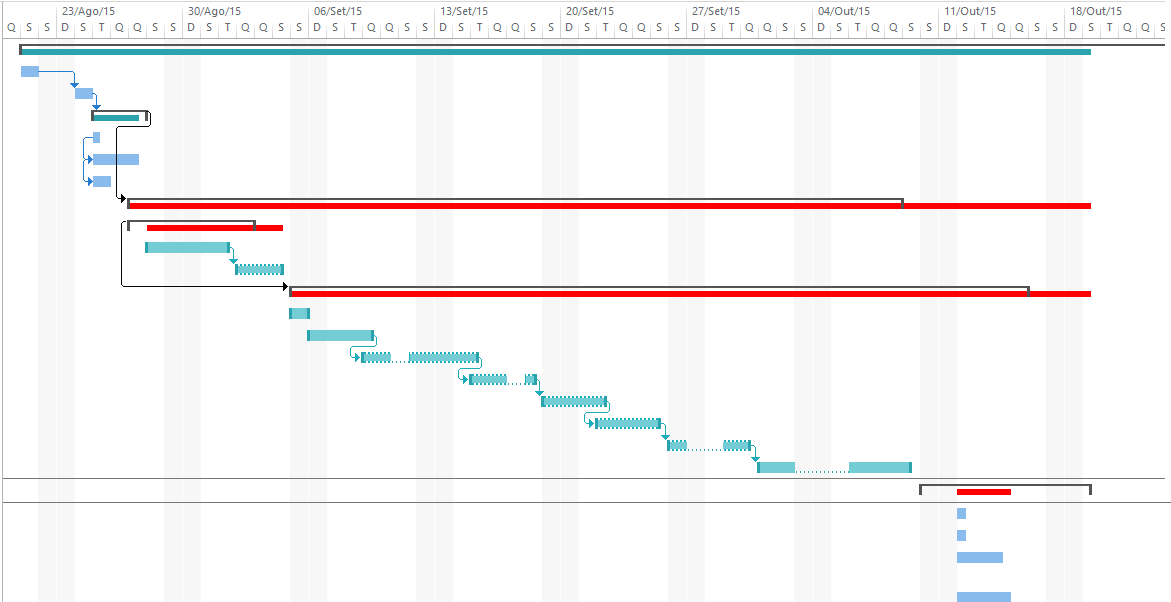
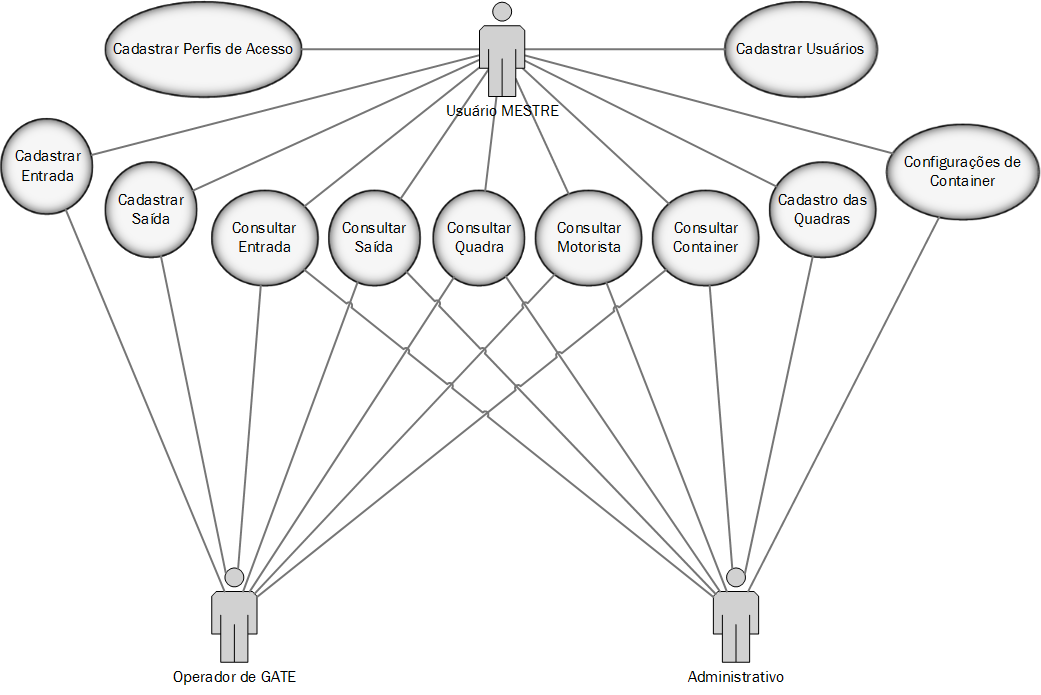


Figura 19 - Gráfico de Grant.



## DIAGRAMA DE CASOS DE USO

Figura 20 - Diagrama de Casos de Uso.

****

## POLÍTICA DE SEGURANÇA

* Todos os funcionários terão usuário e senha para autenticação, nas máquinas, pastas e acesso a internet;
* É proibido o uso de Pen-drive, assim como recarregar celular pela porta USB;
* Todos os acessos externos terão que ser realizadas conexão criptografada por VPN;
* Os servidores de aplicação terão o certificado HTTPS;
* Os *backups* serão criptografados em fita.
* O *backup* ocorrerá de duas em duas horas, de forma incremental e ao final do dia um b*ackup* completo;
* Nossos servidores estarão em um *Data Center* com autenticação Biométrica para poder acessar a sala;
* Nosso *Data Center* irá conter duas (2) câmeras de segurança;
* Nossos servidores serão monitorados para identificarmos possíveis problemas, ou prever problemas que possam ocorrer futuramente.

## PLANO DE CONTINGÊNCIA

* Teremos duas máquinas de aplicação e duas máquinas de banco de dados, sendo divididas as cargas entre elas, ambas irão poder utilizar até 50% de sua capacidade, pois se uma der problema a outra assume a carga que estava sendo depositada no outro servidor, até solucionarmos o problema;
* Caso ocorra inconsistência nos dados, teremos o *backup* para ser restaurado;
* Teremos dois (2) *nobreaks* com carga suficiente para deixar os equipamentos ligados por até duas horas.

## RELATÓRIO DE AUDITORIA

Durante o processo de auditoria serão verificados os seguintes itens: *backup* e *logs*.

Para auditoria do *backup* será criado um ambiente separado e testado para verificar a integridade do mesmo, sendo o backup feito em mídia física ficando em posse do administrador de sistemas da empresa.

Os logs serão criados com as informações de ação dos usuários, serão testadas se estão sendo salvas de acordo com o que foi executado no sistema.

## METODOLOGIA

Utilizamos a metodologia Scrum para um desenvolvimento ágil afim de diminuir gastos, fazendo entregas pequenas periodicamente, caso haja falhas a correção é mais rápida e o custo consequentemente é menor, e a satisfação do cliente é maior, pois está envolvido no projeto.

## RELATÓRIO DE DESENVOLVIMENTO

Utilizamos como forma de documentação um relatório onde contém a descrição da atividade e seus requisitos, a data de entrega e um resumo do que foi desenvolvido e caso haja impacto em outras telas será descrito.

### EXEMPLO

**Atividade:** Refatoração da arquitetura do sistema para o modelo MVC, afim de melhorar a manutenção do sistema.

**Data da entrega:** 15/11/2015.

**Desenvolvedor:** Rodrigo Antunes.

**Resumo:** Feita a refatoração em todo o projeto, deixando todo orientado a objeto e no modelo MVC, ficando mais fácil a manutenção do projeto.

**Impacto:** Essa atividade possui grande impacto em todo o sistema, sendo necessário o teste do sistema atual para verificar todas as funções das telas e comparar com a nova versão refatorada, para verificar se todas as funções estão funcionando de acordo.

## RELATÓRIO DE TESTE

Utilizamos um documento que complementa o relatório de desenvolvimento onde são descritos os testes realizados e os problemas encontrados para que o desenvolvedor tenha acesso e corrigir os problemas encontrados.

### EXEMPLO

**Atividade:** Refatoração da arquitetura do sistema para o modelo MVC, afim de melhorar a manutenção do sistema.

**Data da entrega:** 30/11/2015.

**Tester:** Willian Resille

**Inconformidades:** Na tela de entrada, não está realizando o filtro pela placa, sendo necessária a digitação de todos os campos manualmente.

**Criticidade:** Alta

## MELHORIAS

### IMPLANTAÇÃO MODELO MVC

Aplicação de uma estrutura MVC (*Model*, *View*, *Controler*):

Foi aplicado padrão MVC, as classes *Model* contém todos os atributos dos objetos e seus respectivos métodos *getters* e *setters* e seu construtor.

As camadas *Controler* contêm as chamadas dos métodos da camada *Business* onde contém as regras de negócio e que por sua vez contém chamadas para a camada de persistência, onde são realizadas consultas, inserções, deleções e alterações no banco de dados.

As camadas *View* contém apenas a exibição dos dados processados pelas demais camadas.

### IMPLANTAÇÃO DE LOG

O Sistema possui um sistema de Log de ações dos usuários que são salvas diretamente

no banco em uma tabela de log que terá uma duração de 6 meses, após 6 meses será deletada.

* **Os dados gravados destas operações são:**
* Código de Identificação do Usuário;
* Data e Hora;
* Ação executada;
* Dados alterados na tela.

### IMPLANTAÇÃO DE INTEGRAÇÃO CONTÍNUA

### FERAMENTAS

**Jenkins**: É uma ferramenta para automatização de *deploy* da aplicação.

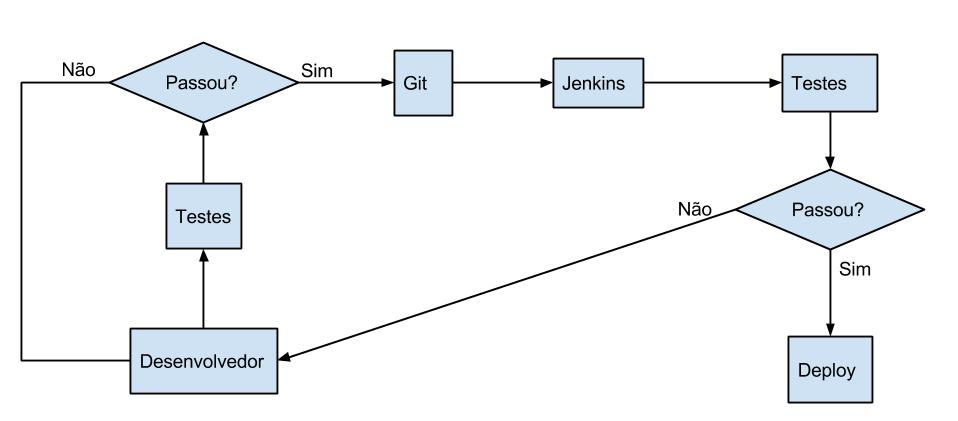
**GIT**: É um repositório de arquivos onde possui todo o histórico de modificações do código podendo assim ver quem fez alteração em determinada linha do código e ver também como era antes de determinada alteração.

**Slack**: É uma ferramenta de comunicação no qual se pode integrar ferramentas como Jenkins, GIT, e outros.

### ESTUTURA

Foi realizada a implantação do Jenkins para realização de *deploy* da aplicação, ela está ligada diretamente com o GIT onde fica monitorando o repositório e faz o *Build* automaticamente, o *Deploy* é feito manualmente após os testes estarem validados. Tanto o Git quanto o Jenkins estão sincronizados com o SLACK sendo assim qualquer ação tomada dentro de uma das ferramentas, automaticamente é enviada uma mensagem para o grupo onde está configurada a sincronização como visto na figura 21.

Figura 21 - Integração Continua.



# CONCLUSÃO

Com base nos estudos feitos durante esse trabalho, analisando toda a rotina de trabalho de terminais de pequeno porte, identificando possíveis problemas e desenvolvendo soluções inteligentes e com custo atraente, concluiu-se que a plataforma web do sistema RetroPátio terá um papel significativo no desenvolvimento de pequenos terminais de contêineres, oferecendo segurança nas informações e agilidade nos desdobramentos dentro do terminal, a um custo compatível com o mercado. Reduzindo custos e o tempo de cadastros de entrada e saída de contêineres, caminhões e motoristas, a lucratividade do terminal terá uma melhora significativa, podendo inclusive almejar uma ampliação de instalações e, consequentemente, da capacidade de armazenamento, fazendo assim a prospecção e a fidelização de novos clientes.

Com tudo aprendemos também importantes assuntos referentes à área de TI, principalmente sobre o desenvolvimento de sistemas, como o apresentado por nós, e através desses ensinamentos esperamos estar mais aptos e preparados para novos desafios.

Em projetos futuros o sistema contará com acesso web para que as transportadoras possam registrar os containers que enviarão para o terminal. Após, emitindo um controle com os dados do container e um QRCODE, no qual, assim que o motorista chegar com a portaria do terminal, o conferente do gate, usará um leitor de QRCODE (smartphone), confirmando os dados e enviando diretamente para a tela do operador do gate, que o mesmo irá selecionar o melhor local para alocar o container que acabará de chegar, confirmando a quadra e enviando o motorista com o caminhão para o conferente de pátio, que junto com o operador de empilhadeira finalizem o processo de descarga. Assim reduzindo mais o tempo e agilizando de tramite entre caminhão e portaria.

# ANEXO

**Codigo Fonte do Sistema RetroPátio**

**Método gravaEntrada da Classe de Persistencia EntradasDao.java**

**Public** String gravaEntrada(Entrada entrada) **throws** Exception{

String RetornoQuadra = **null**;

Connection dbConnection = **null**;

CallableStatement callableStatement = **null**;

String getDBUSERByUserIdSql = "{call gravarentrada(?,?,?,?,?)}";

**try** {

dbConnection = ConnectionFactory.*getConnection*();

callableStatement = (CallableStatement) dbConnection.prepareCall(getDBUSERByUserIdSql);

callableStatement.setInt(1, Integer.*parseInt*(entrada.getIdMotorista()));

callableStatement.setInt(2, Integer.*parseInt*(entrada.getIdCaminhao()));

callableStatement.setInt(3, Integer.*parseInt*(entrada.getIdContainer()));

callableStatement.setInt(4, Integer.*parseInt*(entrada.getIdQuadra()));

callableStatement.setString(5, entrada.getDataHora());

callableStatement.execute();

ResultSet rs = (ResultSet) callableStatement.getResultSet();

**while**(rs.next())

{

RetornoQuadra = rs.getString("IDENTRADA");

}

} **catch** (SQLException e) {

throw new SqlException(E.getMessage());

} **finally** {

**if** (callableStatement != **null**) {

callableStatement.close();

}

**if** (dbConnection != **null**) {

dbConnection.close();

}

}

**return** RetornoQuadra;

}

**Class Model – Entrada.java**

**package** retropatio.model;

**public** **class** Entrada {

**private** String nomeMotorista;

**private** String placaCaminhao;

**private** String numeroContainer;

**private** String posicaoQuadra;

**private** String dataSaida;

**private** String idMotorista;

**private** String idContainer;

**private** String idCaminhao;

**private** String idQuadra;

**private** String dataHora;

**private** String idSaida;

**private** String nome;

**private** String sobrenome;

**private** String nomeCompleto;

**private** String motoristaCPF;

**private** String motoristaCNH;

**private** String placa;

**private** String dataAcesso;

**private** String Empresa;

**private** String Marca;

**private** String modelo;

**private** String Erro;

**public** String getNomeMotorista() {

**return** nomeMotorista;

}

**public** **void** setNomeMotorista(String nomeMotorista) {

**this**.nomeMotorista = nomeMotorista;

}

**public** String getPlacaCaminhao() {

**return** placaCaminhao;

}

**public** **void** setPlacaCaminhao(String placaCaminhao) {

**this**.placaCaminhao = placaCaminhao;

}

**public** String getNumeroContainer() {

**return** numeroContainer;

}

**public** **void** setNumeroContainer(String numeroContainer) {

**this**.numeroContainer = numeroContainer;

}

**public** String getPosicaoQuadra() {

**return** posicaoQuadra;

}

**public** **void** setPosicaoQuadra(String posicaoQuadra) {

**this**.posicaoQuadra = posicaoQuadra;

}

**public** String getDataSaida() {

**return** dataSaida;

}

**public** **void** setDataSaida(String dataSaida) {

**this**.dataSaida = dataSaida;

}

**public** String getIdMotorista() {

**return** idMotorista;

}

**public** **void** setIdMotorista(String idMotorista) {

**this**.idMotorista = idMotorista;

}

**public** String getIdContainer() {

**return** idContainer;

}

**public** **void** setIdContainer(String idContainer) {

**this**.idContainer = idContainer;

}

**public** String getIdCaminhao() {

**return** idCaminhao;

}

**public** **void** setIdCaminhao(String idCaminhao) {

**this**.idCaminhao = idCaminhao;

}

**public** String getIdQuadra() {

**return** idQuadra;

}

**public** **void** setIdQuadra(String idQuadra) {

**this**.idQuadra = idQuadra;

}

**public** **void** setTamanho(String tamanho) {

Tamanho = tamanho;

}

**public** String getDataEntrada() {

**return** dataEntrada;

}

**public** **void** setDataEntrada(String dataEntrada) {

**this**.dataEntrada = dataEntrada;

}

**public** String getIdQuadraContainer() {

**return** idQuadraContainer;

}

**public** **void** setIdQuadraContainer(String idQuadraContainer) {

**this**.idQuadraContainer = idQuadraContainer;

}

**public** String getQuadra() {

**return** quadra;

}

**public** **void** setQuadra(String quadra) {

**this**.quadra = quadra;

}

**public** String getAltura() {

**return** Altura;

}

**public** **void** setAltura(String altura) {

Altura = altura;

}

**public** **void** setNumeroQuadra(String numeroQuadra) {

**this**.numeroQuadra = numeroQuadra;

}

**public** **void** setTamanhoContainerQuadra(String tamanhoContainerQuadra) {

**this**.tamanhoContainerQuadra = tamanhoContainerQuadra;

}

}

**Trecho do Codigo HTML + JQuery da Tela cadastrar\_entradas.jsp**

<div class="ui large modal" id="modalLarge">

<i class="close icon" id="closer"></i>

<div class="header" id="titulo">Cadastro de Entrada</div>

<div class="content" id="content">

<ul id="error" class="ui red message" style="display:none"><li><span id="msgError"></span></li></ul>

<div class="ui form">

<div id="dadosContainer">

<h4 class="ui header">

<i class="cube icon"></i>

<span class="content">Dados do container</span>

</h4>

<div class="three fields">

<div class="field">

<label for="nome">Número</label>

<s:textfield name="numeroContainer" cssClass="upcase" placeholder='digite o nº' id="buscaContainer" key="label.nome" size="20" ></s:textfield>

</div>

<div class="field">

<label for="sobrenome">Armador</label>

<s:select label="Armadores" key="listaArmadores" headerKey="0" id="selectArmadores" headerValue="Selecione um armador" list="listaArmadores" cssClass="ui dropdown" ></s:select>

<s:hidden name="armadoresSel" id="armadoresSel" ></s:hidden>

</div>

<div class="field">

<label for="cor">Cores</label>

<div class="ui dropdown selection" id="testeCor" tabindex="0">

<input type="hidden" name="gender" value="0">

<div class="text">Selecione uma cor</div>

<i class="dropdown icon" tabindex="0"></i>

<div class="menu transition hidden" tabindex="-1">

<div class="item" data-value="Selecione uma cor">Selecione uma cor</div>

<div class="item" data-value="amarelo">

<img src="/RetroPatio/assets/img/amarelo.jpg" alt="">Amarelo</div>

<div class="item active selected" data-value="azul" ><img src="/RetroPatio/assets/img/azul.jpg" alt="">Azul</div>

<div class="item active selected" data-value="branco" ><img src="/RetroPatio/assets/img/branco.jpg" alt="">Branco</div>

<div class="item active selected" data-value="cinza"><img src="/RetroPatio/assets/img/cinza.jpg" alt="">Cinza</div>

<div class="item active selected" data-value="laranja"><img src="/RetroPatio/assets/img/laranja.jpg" alt="">Laranja</div>

<div class="item active selected" data-value="lilas"><img src="/RetroPatio/assets/img/lilas.jpg" alt="">Lilas</div>

<div class="item active selected" data-value="marrom"><img src="/RetroPatio/assets/img/marrom.jpg" alt="">Marrom</div>

<div class="item active selected" data-value="preto"><img src="/RetroPatio/assets/img/preto.jpg" alt="">Preto</div>

<div class="item active selected" data-value="rosa"><img src="/RetroPatio/assets/img/rosa.jpg" alt="">Rosa</div>

<div class="item active selected" data-value="roxo"><img src="/RetroPatio/assets/img/roxo.jpg" alt="">Roxo</div>

<div class="item active selected" data-value="verde"><img src="/RetroPatio/assets/img/verde.jpg" alt="">Verde</div>

<div class="item active selected" data-value="vermelho"><img src="/RetroPatio/assets/img/vermelho.jpg" alt="">Vermelho</div>

</div>

</div>

</div>

</div>

<div class="four fields">

<div class="field">

<label for="modelo">Modelo</label>

<select id="selectModelos" class="ui dropdown" >

<option value="0">Selecione um modelo</option>

</select>

<s:hidden name="modeloSel" id="modeloSel" ></s:hidden>

</div>

<jsp:include page="/assets/javascripts.jsp" />

<jsp:include page="/assets/funcoesjs.jsp" />

<script>

$(document).ready(function(){

carregaSelectModelos();

carregaSelectTamanhos();

$("#testeCor").dropdown({

on: "hover",

onChange: function (val) {

$("#corCont").val(val);

},

"set text" : "Selecione uma cor"

});

$("#selectArmadores").dropdown({

on: "hover",

"set text" : "Selecione um armador",

onChange: function (val) {

$("#armadoresSel").val(val);

},

});

$("#selectQuadras").dropdown({

on: "hover",

onChange: function (val) {

$("#quadraPosi").val(val);

carregaPosicoes();

}});

$("#selectMarcas").dropdown({

on: "hover",

"set text" : "Selecione uma marca",

onChange: function (val) {

$("#marcaCaminhao").val(val);

}

});

$("#selectPosicoes").dropdown({on: "hover"});

$("#selectArmadores").dropdown("set text", "Selecione um armador");

$("#selectMarcas").dropdown("set text" , "Selecione uma marca");

})

function carregaSelectTamanhos(){

$.getJSON("carregaTabelaTamanhos",

function(response){

var option = "<option value='0'>Selecione um tamanho</option>";

$("#selectTamanhos").empty();

$.each(response,function(key,item){

option+= "<option value='" + item.tamanho + "'>" + item.tamanho + "</option>";

});

$("#selectTamanhos").append(option);

$("#selectTamanhos").dropdown({

on: "hover",

onChange: function (val) {

$("#tamanhoSel").val(val);

carregaQuadras(val);

}});

});

}

</script>

**Script MySql – Banco de Dados do Sistema RetroPátio**

-- MySQL dump 10.13 Distrib 5.6.17, for Win32 (x86)

-- Host: localhost Database: icod\_propatio

-- Server version 5.6.22-log

-- Table structure for table `tb\_armador`

DROP TABLE IF EXISTS `tb\_armador`;

CREATE TABLE `tb\_armador` (

`idarmador` int(11) NOT NULL AUTO\_INCREMENT,

`armador` varchar(50) DEFAULT NULL,

PRIMARY KEY (`idarmador`)

) ENGINE=InnoDB AUTO\_INCREMENT=101 DEFAULT CHARSET=utf8;

--

-- Table structure for table `tb\_caminhao`

--

DROP TABLE IF EXISTS `tb\_caminhao`;

CREATE TABLE `tb\_caminhao` (

`id\_caminhao` int(11) NOT NULL AUTO\_INCREMENT,

`placa` char(7) DEFAULT NULL,

`id\_empresas` int(11) DEFAULT NULL,

`id\_marcas` int(11) DEFAULT NULL,

`id\_modelos` int(11) DEFAULT NULL,

`datacad` varchar(45) DEFAULT NULL,

PRIMARY KEY (`id\_caminhao`),

UNIQUE KEY `id\_caminhao\_UNIQUE` (`id\_caminhao`),

KEY `empresas\_idx` (`id\_empresas`),

KEY `marcas\_idx` (`id\_marcas`),

KEY `modelos\_idx` (`id\_modelos`),

CONSTRAINT `empresas` FOREIGN KEY (`id\_empresas`) REFERENCES `tb\_empresa` (`id\_empresa`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `idmarcas` FOREIGN KEY (`id\_marcas`) REFERENCES `tb\_caminhao\_marca` (`id\_marca`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `idmodelos` FOREIGN KEY (`id\_modelos`) REFERENCES `tb\_caminhao\_modelo` (`id\_modelo`) ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE=InnoDB AUTO\_INCREMENT=34 DEFAULT CHARSET=utf8;

--

-- Table structure for table `tb\_caminhao\_marca`

--

DROP TABLE IF EXISTS `tb\_caminhao\_marca`;

CREATE TABLE `tb\_caminhao\_marca` (

`id\_marca` int(11) NOT NULL AUTO\_INCREMENT,

`Marca` varchar(45) DEFAULT NULL,

PRIMARY KEY (`id\_marca`),

UNIQUE KEY `idtb\_marca\_caminhao\_UNIQUE` (`id\_marca`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8;

--

-- Table structure for table `tb\_caminhao\_modelo`

--

DROP TABLE IF EXISTS `tb\_caminhao\_modelo`;

CREATE TABLE `tb\_caminhao\_modelo` (

`id\_modelo` int(11) NOT NULL AUTO\_INCREMENT,

`Modelo` varchar(65) DEFAULT NULL,

`Marca` int(11) DEFAULT NULL,

PRIMARY KEY (`id\_modelo`),

UNIQUE KEY `idtb\_caminhao\_modelo\_UNIQUE` (`id\_modelo`),

KEY `Marcas\_idx` (`Marca`),

CONSTRAINT `Marcas` FOREIGN KEY (`Marca`) REFERENCES `tb\_caminhao\_marca` (`id\_marca`) ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE=InnoDB AUTO\_INCREMENT=23 DEFAULT CHARSET=utf8;

--

-- Table structure for table `tb\_cnhs`

--

DROP TABLE IF EXISTS `tb\_cnhs`;

CREATE TABLE `tb\_cnhs` (

`idcnh` int(11) NOT NULL AUTO\_INCREMENT,

`cnh` bigint(20) DEFAULT NULL,

PRIMARY KEY (`idcnh`),

UNIQUE KEY `idcnh\_UNIQUE` (`idcnh`)

) ENGINE=InnoDB AUTO\_INCREMENT=19 DEFAULT CHARSET=utf8;

--

-- Table structure for table `tb\_cont\_tamanho`

--

DROP TABLE IF EXISTS `tb\_cont\_tamanho`;

CREATE TABLE `tb\_cont\_tamanho` (

`id\_cont\_tamanho` int(11) NOT NULL AUTO\_INCREMENT,

`Tamanho` int(11) DEFAULT NULL,

PRIMARY KEY (`id\_cont\_tamanho`),

UNIQUE KEY `idtb\_cont\_tamanho\_UNIQUE` (`id\_cont\_tamanho`)

) ENGINE=InnoDB AUTO\_INCREMENT=5 DEFAULT CHARSET=utf8;

--

-- Table structure for table `tb\_containers`

--

DROP TABLE IF EXISTS `tb\_containers`;

CREATE TABLE `tb\_containers` (

`idcontainer` int(11) NOT NULL AUTO\_INCREMENT,

`numero` varchar(45) DEFAULT NULL,

`tamanho` int(11) DEFAULT NULL,

`modelo` int(11) DEFAULT NULL,

`armador` int(11) DEFAULT NULL,

`dataCadastro` varchar(45) DEFAULT NULL,

`cor` varchar(45) DEFAULT NULL,

PRIMARY KEY (`idcontainer`),

KEY `modelo\_idx` (`modelo`),

KEY `armador\_idx` (`armador`),

KEY `tamanhos\_idx` (`tamanho`),

CONSTRAINT `armador` FOREIGN KEY (`armador`) REFERENCES `tb\_armador` (`idarmador`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `modelo` FOREIGN KEY (`modelo`) REFERENCES `tb\_mod\_container` (`idmodelo`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `tamanhos` FOREIGN KEY (`tamanho`) REFERENCES `tb\_cont\_tamanho` (`id\_cont\_tamanho`) ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE=InnoDB AUTO\_INCREMENT=242 DEFAULT CHARSET=utf8;

--

-- Table structure for table `tb\_cpfs`

--

DROP TABLE IF EXISTS `tb\_cpfs`;

CREATE TABLE `tb\_cpfs` (

`idcpfs` int(11) NOT NULL AUTO\_INCREMENT,

`cpf` varchar(11) DEFAULT NULL,

PRIMARY KEY (`idcpfs`),

UNIQUE KEY `idcpfs\_unique` (`idcpfs`),

UNIQUE KEY `cpf\_unique` (`cpf`)

) ENGINE=InnoDB AUTO\_INCREMENT=18 DEFAULT CHARSET=utf8;

--

-- Table structure for table `tb\_empresa`

--

DROP TABLE IF EXISTS `tb\_empresa`;

CREATE TABLE `tb\_empresa` (

`id\_empresa` int(11) NOT NULL AUTO\_INCREMENT,

`Nome` varchar(85) DEFAULT NULL,

PRIMARY KEY (`id\_empresa`),

UNIQUE KEY `id\_empresa\_UNIQUE` (`id\_empresa`)

) ENGINE=InnoDB AUTO\_INCREMENT=34 DEFAULT CHARSET=utf8;

--

-- Table structure for table `tb\_entradas`

--

DROP TABLE IF EXISTS `tb\_entradas`;

CREATE TABLE `tb\_entradas` (

`id\_entrada` int(11) NOT NULL AUTO\_INCREMENT,

`id\_motorista` int(11) DEFAULT NULL,

`id\_caminhao` int(11) DEFAULT NULL,

`id\_container` int(11) DEFAULT NULL,

`id\_quadra` int(11) DEFAULT NULL,

`data\_entrada` datetime DEFAULT NULL,

PRIMARY KEY (`id\_entrada`),

UNIQUE KEY `id\_entrada\_UNIQUE` (`id\_entrada`),

KEY `idMotorista\_idx` (`id\_motorista`),

KEY `idCaminhao\_idx` (`id\_caminhao`),

KEY `idContainer\_idx` (`id\_container`),

KEY `idQuadra\_idx` (`id\_quadra`),

CONSTRAINT `idCaminhao` FOREIGN KEY (`id\_caminhao`) REFERENCES `tb\_caminhao` (`id\_caminhao`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `idContainer` FOREIGN KEY (`id\_container`) REFERENCES `tb\_containers` (`idcontainer`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `idMotorista` FOREIGN KEY (`id\_motorista`) REFERENCES `tb\_motorista` (`idmotorista`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `idQuadras` FOREIGN KEY (`id\_quadra`) REFERENCES `tb\_quadra` (`idquadra`) ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE=InnoDB AUTO\_INCREMENT=184 DEFAULT CHARSET=utf8;

--

-- Table structure for table `tb\_mod\_container`

-

DROP TABLE IF EXISTS `tb\_mod\_container`;

CREATE TABLE `tb\_mod\_container` (

`idmodelo` int(11) NOT NULL AUTO\_INCREMENT,

`modelo` varchar(85) DEFAULT NULL,

PRIMARY KEY (`idmodelo`)

) ENGINE=InnoDB AUTO\_INCREMENT=9 DEFAULT CHARSET=utf8;

--

-- Table structure for table `tb\_motorista`

--

DROP TABLE IF EXISTS `tb\_motorista`;

CREATE TABLE `tb\_motorista` (

`idmotorista` int(11) NOT NULL AUTO\_INCREMENT,

`nome` int(1) DEFAULT NULL,

`sobrenome` int(1) DEFAULT NULL,

`cpf` int(1) DEFAULT NULL,

`cnh` int(11) DEFAULT NULL,

`dataacesso` datetime DEFAULT NULL,

PRIMARY KEY (`idmotorista`),

UNIQUE KEY `idmotorista\_unique` (`idmotorista`),

UNIQUE KEY `cpf\_UNIQUE` (`cpf`),

KEY `nome\_idx` (`nome`),

KEY `sobrenome\_idx` (`sobrenome`),

KEY `cpf\_idx` (`cpf`),

KEY `cnh\_idx` (`cnh`),

CONSTRAINT `cnh` FOREIGN KEY (`cnh`) REFERENCES `tb\_cnhs` (`idcnh`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `cpf` FOREIGN KEY (`cpf`) REFERENCES `tb\_cpfs` (`idcpfs`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `nome` FOREIGN KEY (`nome`) REFERENCES `tb\_nomes` (`idnome`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `sobrenome` FOREIGN KEY (`sobrenome`) REFERENCES `tb\_sobrenome` (`idsobrenome`) ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE=InnoDB AUTO\_INCREMENT=69 DEFAULT CHARSET=utf8;

--

-- Table structure for table `tb\_nomes`

--

DROP TABLE IF EXISTS `tb\_nomes`;

CREATE TABLE `tb\_nomes` (

`idnome` int(11) NOT NULL AUTO\_INCREMENT,

`nome` varchar(45) DEFAULT NULL,

PRIMARY KEY (`idnome`)

) ENGINE=InnoDB AUTO\_INCREMENT=32 DEFAULT CHARSET=utf8;

--

-- Table structure for table `tb\_perfis`

--

DROP TABLE IF EXISTS `tb\_perfis`;

CREATE TABLE `tb\_perfis` (

`id\_perfis` int(11) NOT NULL AUTO\_INCREMENT,

`perfil` varchar(45) DEFAULT NULL,

`entrada` int(11) DEFAULT NULL,

`saida` int(11) DEFAULT NULL,

`cadastro` int(11) DEFAULT NULL,

`consulta` int(11) DEFAULT NULL,

PRIMARY KEY (`id\_perfis`),

UNIQUE KEY `idcargos\_unique` (`id\_perfis`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8;

--

-- Table structure for table `tb\_quadra`

--

DROP TABLE IF EXISTS `tb\_quadra`;

CREATE TABLE `tb\_quadra` (

`idquadra` int(11) NOT NULL AUTO\_INCREMENT,

`quadra` varchar(2) DEFAULT NULL,

`numero` int(11) DEFAULT NULL,

`fileira` int(11) DEFAULT NULL,

`altura` int(11) DEFAULT NULL,

`container\_tam` int(11) DEFAULT NULL,

`container` int(11) DEFAULT NULL,

PRIMARY KEY (`idquadra`),

UNIQUE KEY `container\_UNIQUE` (`container`),

KEY `Tamanho\_Cont\_idx` (`container\_tam`),

KEY `ID\_Container\_idx` (`container`),

CONSTRAINT `ID\_Container` FOREIGN KEY (`container`) REFERENCES `tb\_containers` (`idcontainer`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `Tamanho\_Cont` FOREIGN KEY (`container\_tam`) REFERENCES `tb\_cont\_tamanho` (`id\_cont\_tamanho`) ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE=InnoDB AUTO\_INCREMENT=3413 DEFAULT CHARSET=utf8;

--

-- Table structure for table `tb\_remocao`

--

DROP TABLE IF EXISTS `tb\_remocao`;

CREATE TABLE `tb\_remocao` (

`id` int(11) DEFAULT NULL,

`numero` varchar(45) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--

-- Table structure for table `tb\_saidas`

--

DROP TABLE IF EXISTS `tb\_saidas`;

CREATE TABLE `tb\_saidas` (

`id\_saida` int(11) NOT NULL AUTO\_INCREMENT,

`id\_motorista` int(11) DEFAULT NULL,

`id\_caminhao` int(11) DEFAULT NULL,

`id\_container` int(11) DEFAULT NULL,

`id\_quadra` int(11) DEFAULT NULL,

`data\_saida` datetime DEFAULT NULL,

PRIMARY KEY (`id\_saida`),

UNIQUE KEY `id\_saida\_UNIQUE` (`id\_saida`),

KEY `Caminhao\_idx` (`id\_caminhao`),

KEY `Container\_idx` (`id\_container`),

KEY `Quadra\_idx` (`id\_quadra`),

KEY `Cod\_Motorista\_idx` (`id\_motorista`),

CONSTRAINT `Cod\_Caminhao` FOREIGN KEY (`id\_caminhao`) REFERENCES `tb\_caminhao` (`id\_caminhao`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `Cod\_Container` FOREIGN KEY (`id\_container`) REFERENCES `tb\_containers` (`idcontainer`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `Cod\_Quadra` FOREIGN KEY (`id\_quadra`) REFERENCES `tb\_quadra` (`idquadra`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `Codigo\_Motor` FOREIGN KEY (`id\_motorista`) REFERENCES `tb\_motorista` (`idmotorista`) ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE=InnoDB AUTO\_INCREMENT=39 DEFAULT CHARSET=utf8;

--

-- Table structure for table `tb\_sobrenome`

--

DROP TABLE IF EXISTS `tb\_sobrenome`;

CREATE TABLE `tb\_sobrenome` (

`idsobrenome` int(11) NOT NULL AUTO\_INCREMENT,

`sobrenome` varchar(85) DEFAULT NULL,

PRIMARY KEY (`idsobrenome`)

) ENGINE=InnoDB AUTO\_INCREMENT=23 DEFAULT CHARSET=utf8;

--

-- Table structure for table `tb\_usuarios`

--

DROP TABLE IF EXISTS `tb\_usuarios`;

CREATE TABLE `tb\_usuarios` (

`idusuario` int(11) NOT NULL AUTO\_INCREMENT,

`nome` int(11) DEFAULT NULL,

`sobrenome` int(11) DEFAULT NULL,

`login` varchar(15) DEFAULT NULL,

`senha` varchar(255) DEFAULT NULL,

`perfil` int(11) DEFAULT NULL,

`datacadastro` datetime DEFAULT NULL,

PRIMARY KEY (`idusuario`),

KEY `nomes\_idx` (`nome`),

KEY `sobrenomes\_idx` (`sobrenome`),

KEY `cargos\_idx` (`perfil`),

CONSTRAINT `nomes` FOREIGN KEY (`nome`) REFERENCES `tb\_nomes` (`idnome`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `perfis` FOREIGN KEY (`perfil`) REFERENCES `tb\_perfis` (`id\_perfis`) ON DELETE NO ACTION ON UPDATE NO ACTION,

CONSTRAINT `sobrenomes` FOREIGN KEY (`sobrenome`) REFERENCES `tb\_sobrenome` (`idsobrenome`) ON DELETE NO ACTION ON UPDATE NO ACTION

) ENGINE=InnoDB AUTO\_INCREMENT=18 DEFAULT CHARSET=utf8;

-- Dump completed on 2015-10-18 17:31:25

# Referências Bibliográficas

ARANTES, Elaine Cristina. **Empreendedorismo e responsabilidade social**. 2 Ed. rev. Curitiba: InterSaberes, 2014.

BECK, Kent. Disponível em: <http://www.manifestoagil.com.br/> Acesso em: 12 de outubro 2015.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML: guia do usuário**. 2. Ed. rev. e atual. Rio de Janeiro: Elsevier Science Ltda, 2006.

COLLINS, Elaine. **Automação de testes é...** . Disponível em: <http://www.divus.com.br/sobre/blog/item/14-automacao-de-testes-e>. Acesso em 28 de outubro 2015.

COSTA, Pedro. **Os tipos de teste de software**. Disponível em: <http://crowdtest.me/tipos-teste-software/>. Acesso em: 28 de outubro 2015.

COSTA, Wander Moreira da; CUNHA, Dênio Magno da; DIAS, Igor Augusto de Melo; SOUZA, Erika Rúbia de. **Empreendedorismo**. Minas Gerais, Grupo Anima Educação, 2015.

DIAS, Arilo Claudio. **Artigo Engenharia de Software - Introdução a Teste de Software**. Disponível em: <http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-teste-de-software/8035>. Acesso em: 28 de outubro 2015.

DIAS, Sergio Vidal Dos Santos. **Auditoria de processos organizacionais: Teoria, Finalidade, Metodologia de Trabalho e Resultados Esperados**. 3. Ed. São Paulo: Atlas, 2011.

FOWLER, Martin. **Continuous Integration: Improving Software Quality and Reducing Risk**. 1 Ed. USA: Addison-Wesley Professional, 2007.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Padrões de Projeto: soluções reutilizáveis de software orientado a objetos**. 1. Ed. Porto Alegre: Bookman, 2000.

GHEDIN, Rodrigo. **Mark Zuckerberg fala sobre HTML5, apps móveis e as ações “decepcionantes” do Facebook**. Disponível em: <http://gizmodo.uol.com.br/mark-zuckerberg-fala-html5-apps-moveis-bolsa-facebook/> Acesso em: 16 de outubro 2015.

G1. **'Angry birds': Longa de animação baseado no game ganha 1º trailer; veja**. Disponível em: <http://g1.globo.com/pop-arte/cinema/noticia/2015/09/angry-birds-longa-de-animacao-baseado-no-game-ganha-1-trailer-veja.html> Acesso em: 16 de outubro 2015.

G1. **'Angry Birds 2' é lançado mais de 5 anos após 1º game; veja trailer**. Disponível em: <http://g1.globo.com/tecnologia/games/noticia/2015/07/angry-birds-2-e-lancado-mais-de-5-anos-apos-1-game-veja-trailer.html> Acesso em: 16 de outubro 2015.

IMONIANA, Joshua Onome. **Auditoria de sistemas de informação**. São Paulo: 2ª ed. Ed. Atlas, 2008.

JORDÃO, Fabio. **História: a evolução do celular**. Disponível em: <http://www.tecmundo.com.br/celular/2140-historia-a-evolucao-do-celular.htm>. Acesso em: 16 de outubro 2015.

KORF, Mario; OKSMAN Eugene. **Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options**. Disponível em: <https://developer.salesforce.com/page/Native,\_HTML5,\_or\_Hybrid:\_Understanding\_Your\_Mobile\_Application\_Development\_Options> Acesso em 16 de outubro 2015.

LANDIM, Wikerson. **Com celulares, indústria de games pode chegar a US$ 100 bilhões em 2017**. Disponível em: <http://www.tecmundo.com.br/video-game-e-jogos/58852-celulares-industria-games-chegar-us-100-bilhoes-2017.htm> Acesso em: 16 de outubro 2015.

LEITE, Alexandre Ferreira. **Conheça os padrões de Projeto**. 2014. Disponível em: http://www.devmedia.com.br/conheca-os-padroes-de-projeto/957#ixzz3lgY9jesY. Acesso em 03/10/2015.

LYRA, Mauricio Rocha. **Segurança e auditoria em sistemas de informação**. Rio de Janeiro: Ciência Moderna, 2008. 253 p.

MEDEIROS, Higor. **Mocks:** **Introdução a Automatização de Testes com Mock Object**. Disponível em: <http://www.devmedia.com.br/mocks-introducao-a-automatizacao-de-testes-com-mock-object/30641>. Acesso em 28 de outubro 2015.

PAGE-JONES, Meilir. **Fundamentos do desenho orientado a objeto com UML**. 2. Ed. São Paulo: Makron Books, 2001.

PAULA FILHO, Wilson de Pádua. **Engenharia de software: fundamentos, métodos e padrões**. 2. Ed. Rio de Janeiro: LTC, 2005.

PFLEEGER, Shari Lawrence. **Engenharia de software: teoria e prática**. São Paulo: Pearson Prentice Hall, 2004.

PEGN TV. **Mercado de aplicativos é promissor mas requer cuidado**; veja dicas. Disponível em: <http://g1.globo.com/economia/pme/noticia/2015/06/mercado-de-aplicativos-e-promissor-mas-requer-cuidado-veja-dicas.html>. Acesso em: 16 de outubro 2015.

PINHEIRO, Mirelle. **Indústria de aplicativos decola no Brasil**. Disponível em: <http://www.em.com.br/app/noticia/tecnologia/2015/05/18/interna\_tecnologia,648580/industria-de-aplicativos-decola-no-brasil.shtml>. Acesso em 16 de outubro 2015.

PINTO, Evandro. **O RUP não é uma metodologia**. Disponível em: <http://wm2info.com.br/blog/2012/01/17/o-rup-nao-e-uma-metodologia/> Acesso em: 28 de outubro 2015.

PRESSMAN, Roger S. **Engenharia de Software**. 6. Ed. São Paulo: McGraw-Hill Interamericana, 2006.

RADWAN, M.Farouck. **Why do people, especially men, like video games**. Disponível em: <http://www.2knowmyself.com/why\_do\_people\_especially\_men\_like\_video\_games> Acesso em 16 de outubro 2015.

ROCHA, Fábio Gomes. **Integração contínua: uma introdução ao assunto**. 2014. Disponível em: http://www.devmedia.com.br/integracao-continua-uma-introducao-ao-assunto/28002. Acesso em 14/09/2015.

SANTOS, Ismayle; ALCANTARA, Pedro. **Automação de testes funcionais com o Selenium**. Disponível em: <http://www.ufpi.br/subsiteFiles/ercemapi/arquivos/files/minicurso/mc2.pdf>. Acesso em: 28 de outubro 2015.

SOMMERVILLE, Ian. **Engenharia de Software** – 9. Ed. – São Paulo: Pearson Prentice Hall, 2011.

SOMMERVILLE, Ian. **Engenharia de software**. 8. Ed. São Paulo: Pearson, 2007.

WELLS, Don. Disponível em: <http://www.extremeprogramming.org/index.html> Acesso em: 12 de outubro 2015.