**­­­Serviço Nacional de Aprendizagem Comercial**

**Faculdade Senac Porto Alegre**

**Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas**

**MARCELO BALDISSERA CURE**

**RELATÓRIO FINAL DE PROJETO**

**SISTEMA DE COLETA DE MÉTRICAS DE PROJETOS ÁGEIS PARA APOIAR ESTIMATIVAS**

Porto Alegre

2014

**MARCELO BALDISSERA CURE**

**RELATÓRIO FINAL DE PROJETO**

**SISTEMA DE COLETA DE MÉTRICAS DE PROJETOS ÁGEIS PARA APOIAR ESTIMATIVAS**

Relatório Final de Projeto, apresentado como requisito parcial à obtenção da aprovação do projeto de TCC2 do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, pela Faculdade Senac Porto Alegre.

Orientador: Prof. Me Luciano Zanuz

Porto Alegre

2014

**LISTA DE ILUSTRAÇÕES**

[Figura 1 – Processo básico do *Scrum*. 9](#_Toc403316741)

[Figura 2 – Exemplo de quadro no Trello. 10](#_Toc403316742)

[Figura 3 - Estrutura básica do *TDD.* 15](#_Toc403316743)

[Figura 4 - Camadas e componentes do sistema. 18](#_Toc403316744)

[Figura 5 - Processo de *login*. 19](#_Toc403316745)

[Figura 6 - Gerador de Gráficos. 19](#_Toc403316746)

[Figura 7 - *Scan*. 20](#_Toc403316747)

[Figura 8 – Diagrama de casos de uso. 26](#_Toc403316748)

[Figura 9 – Diagrama de classes. 26](#_Toc403316749)

[Figura 10 – Diagrama Entidade Relacionamento. 27](#_Toc403316750)

[Figura 11 – Protótipo da tela de cadastro de *sprints*. 29](#_Toc403316751)

[Figura 12 – Protótipo da tela de Scan do projeto. 30](#_Toc403316752)

[Figura 13 – Protótipo da tela de gráficos. 31](#_Toc403316753)

[Figura 14 – Protótipo da tela de projetos do usuário. 32](#_Toc403316754)

[Figura 15 – Protótipo da tela de *sprints* dos projetos do usuário. 33](#_Toc403316755)

[Figura 16 – Protótipo da tela de *cards* de *sprints* dos projetos do usuário. 34](#_Toc403316756)

[Figura 17 – Protótipo da tela de cadastro de perfis. 35](#_Toc403316757)

[Figura 18 – Protótipo da tela de cadastro de usuários. 36](#_Toc403316758)

[Figura 19 – Protótipo da tela de cadastro de clientes. 37](#_Toc403316759)

[Figura 20 – Protótipo da tela de cadastro de projetos. 38](#_Toc403316760)

[Figura 21 – Protótipo da tela de scans pendentes. 39](#_Toc403316761)

[Figura 22 – Protótipo da tela de gráficos. 40](#_Toc403316762)

[Figura 23 – Processo de negócio do sistema. 41](#_Toc403316763)

[Figura 24 – Cobertura de testes do sistema. 42](#_Toc403316764)

[Figura 25 – Tela de cadastro de *sprints*. 43](#_Toc403316765)

[Figura 26 – Tela de Scan do projeto. 44](#_Toc403316766)

[Figura 27 – Tela de gráficos do usuário. 44](#_Toc403316767)

[Figura 28 – Tela de projetos do usuário com historico das *sprints*. 45](#_Toc403316768)

[Figura 29 – Tela de projetos do usuário com historico dos *cards* das *sprints.* 46](#_Toc403316769)

[Figura 30 – Tela de cadastro de perfis. 46](#_Toc403316770)

[Figura 31 – Tela de cadastro de usuários. 47](#_Toc403316771)

[Figura 32 – Tela de cadastro de clientes. 48](#_Toc403316772)

[Figura 33 – Tela de cadastro de projetos. 49](#_Toc403316773)

[Figura 34 – Tela de consulta de *scans* pendentes. 49](#_Toc403316774)

[Figura 35 – Tela de gráficos do administrador. 50](#_Toc403316775)

**LISTA DE TABELAS**

[Tabela 1 – Sprint 1. 23](#_Toc403316733)

[Tabela 2 – Sprint 2. 24](#_Toc403316734)

[Tabela 3 – Sprint 3. 24](#_Toc403316735)

[Tabela 4 – Sprint 4. 24](#_Toc403316736)

[Tabela 5 – Sprint 5. 25](#_Toc403316737)

[Tabela 6 – Sprint 6. 25](#_Toc403316738)

[Tabela 7 - Dicionário de dados 28](#_Toc403316739)

[Tabela 8 - Objetivos mapeados com o modelo de qualidade da ISO. 52](#_Toc403316740)

**LISTA DE SIGLAS**

UML *Unified Modeling Language*

TDD *Test Driven Development*

ER *Entity Relationship*

API *Application Programming Interface*

JSON *JavaScript Object Notation*

TI Tecnologia da Informação

PDF *Portable Document Format*

PNG *Portable Network Graphics*

**RESUMO**

Este trabalho visa melhorar as estimativas de esforço de *sprints* para projetos que utilizam metodologias ágeis. O sistema inclui cadastros de clientes, usuários, projetos, perfis e *sprints*, provendo uma visão de todos os projetos que estão acontecendo na organização. O sistema coleta a quantidade de pontos entregues ao final da sprint e saberá quantos pontos o time de desenvolvimento prometeu entregar. Com estes dados coletados, o sistema é capaz de produzir gráficos com informações históricas relevantes que servem de apoio a tomada de decisão no momento de estimar uma *sprint*. Isto acarreta em segurança para o time de desenvolvimento e para o cliente, que saberá o custo e tempo das *sprints* evitando imprevistos de tempo e financeiros.

SUMÁRIO

[1 APRESENTAÇÃO GERAL DO PROJETO 8](#_Toc403316658)

[2 DEFINIÇÃO DO PROBLEMA 9](#_Toc403316659)

[3 OBJETIVOS 12](#_Toc403316660)

[3.1 Objetivo Geral 12](#_Toc403316661)

[3.2 Objetivos Específicos 12](#_Toc403316662)

[4 ANÁLISE DE TECNOLOGIAS/FERRAMENTAS 13](#_Toc403316663)

[4.1 Linguagens de programação 13](#_Toc403316664)

[4.2 Ferramenta de Desenvolvimento 13](#_Toc403316665)

[4.3 Banco de Dados 13](#_Toc403316666)

[4.4 Controle de Atividades 13](#_Toc403316667)

[4.5 Versionamento do Código 14](#_Toc403316668)

[4.6 Qualidade do Código 14](#_Toc403316669)

[4.7 Ferramentas de Modelagem 15](#_Toc403316670)

[4.8 Protótipos de tela 15](#_Toc403316671)

[5 DESCRIÇÃO DA SOLUÇÃO 17](#_Toc403316672)

[5.1 Gerenciador de Sessões 18](#_Toc403316673)

[5.2 Gerador de Gráficos 19](#_Toc403316674)

[5.3 *Scan* 20](#_Toc403316675)

[6 ABORDAGEM DE DESENVOLVIMENTO 21](#_Toc403316676)

[6.1 Artefatos 21](#_Toc403316677)

[6.1.1 Product Backlog 21](#_Toc403316678)

[6.1.2 Sprint Backlog 21](#_Toc403316679)

[6.1.3 Histórias do Usuário 22](#_Toc403316680)

[6.1.4 Diagrama de Caso de Uso 22](#_Toc403316681)

[6.1.5 Diagrama de Classes 22](#_Toc403316682)

[6.1.6 Cobertura de testes 22](#_Toc403316683)

[7 ARQUITETURA DO SISTEMA 23](#_Toc403316684)

[7.1 Modelagem Funcional 23](#_Toc403316685)

[7.1.1 Sprint 1 23](#_Toc403316686)

[7.1.2 Sprint 2 24](#_Toc403316687)

[7.1.3 Sprint 3 24](#_Toc403316688)

[7.1.4 Sprint 4 24](#_Toc403316689)

[7.1.5 Sprint 5 25](#_Toc403316690)

[7.1.6 Sprint 6 25](#_Toc403316691)

[7.2 Modelos UML 25](#_Toc403316692)

[7.2.1 Diagrama de Casos de Uso 25](#_Toc403316693)

[7.2.2 Diagrama de Classes 26](#_Toc403316694)

[7.3 Modelagem de Dados 27](#_Toc403316695)

[7.3.1 Diagrama ER 27](#_Toc403316696)

[7.3.2 Dicionário de Dados 28](#_Toc403316697)

[7.4 Interface Gráfica 29](#_Toc403316698)

[7.4.1 Cadastro de Sprints 29](#_Toc403316699)

[7.4.2 Scan de projeto 30](#_Toc403316700)

[7.4.3 Gráficos 31](#_Toc403316701)

[7.4.4 Projetos do usuário 32](#_Toc403316702)

[7.4.5 Projetos do usuário - *Sprints* 33](#_Toc403316703)

[7.4.6 Projetos do usuário - *Sprint cards* 34](#_Toc403316704)

[7.4.7 Cadastro de perfis 35](#_Toc403316705)

[7.4.8 Cadastro de usuários 36](#_Toc403316706)

[7.4.9 Cadastro de clientes 37](#_Toc403316707)

[7.4.10 Cadastro de projetos 38](#_Toc403316708)

[7.4.11 Consulta de scans pendentes 39](#_Toc403316709)

[7.4.12 Gráficos 40](#_Toc403316710)

[7.5 Processo de Negócio 41](#_Toc403316711)

[7.6 Testes Unitários 42](#_Toc403316712)

[7.6.1 Testes criados 42](#_Toc403316713)

[7.6.2 Cobertura de testes 42](#_Toc403316714)

[8 FUNCIONAMENTO DO SISTEMA 43](#_Toc403316715)

[8.1 Membro do time 43](#_Toc403316716)

[8.1.1 Cadastro de Sprints 43](#_Toc403316717)

[8.1.2 Scan de projeto 43](#_Toc403316718)

[8.1.3 Gráficos 44](#_Toc403316719)

[8.1.4 Projetos do Usuário 45](#_Toc403316720)

[8.2 Administrador 46](#_Toc403316721)

[8.2.1 Cadastro de perfis 46](#_Toc403316722)

[8.2.2 Cadastro de usuários 47](#_Toc403316723)

[8.2.3 Cadastro de clientes 48](#_Toc403316724)

[8.2.4 Cadastro de projetos 49](#_Toc403316725)

[8.2.5 Consulta de *scans* pendentes 49](#_Toc403316726)

[8.2.6 Gráficos 50](#_Toc403316727)

[9 VALIDAÇÃO 52](#_Toc403316728)

[9.1 ESTRATÉGIA 52](#_Toc403316729)

[10 CONCLUSÕES 54](#_Toc403316730)

[REFERÊNCIAS BIBLIOGRÁFICAS 55](#_Toc403316731)

[COMPONENTES REUTILIZADOS 57](#_Toc403316732)

# 1 APRESENTAÇÃO GERAL DO PROJETO

No desenvolvimento de *software* em geral, a estimativa de esforço para implementar um requisito pode afetar o projeto em alguns fatores, como o prazo, que pode ser comprometido, podendo acarretar em atrasos de entrega e custos extras. Conforme Valente e Falbo (2002) as estimativas são peça fundamental em um projeto, pois representa um fator chave para que o mesmo tenha sucesso.

Atualmente, as metodologias ágeis estão se tornando cada vez mais populares na indústria do desenvolvimento de *software* (PLANBOX, 2013). Estas metodologias tendem a diminuir o risco do projeto fazendo entregas periódicas chamadas de iterações.

Uma das metodologias ágeis mais conhecidas e utilizadas é o Scrum. Esta metodologia consiste em fazer entregas frequentes de funcionalidades, reuniões diárias para sincronização da equipe, entre outras práticas (METHODOLOGY, 2009). Ao final de cada iteração, chamada de *sprint*, funcionalidades completas são entregues, possibilitando ao cliente prover um *feedback* e planejar as próximas *sprints* (KEN SCHWABER, 2013).

Projetos executados com *Scrum* normalmente possuem *sprints* de períodos curtos, de uma a quatro semanas, onde os requisitos são quebrados em *user stories* (histórias do usuário). Cada história do usuário é estimada, normalmente, utilizando uma técnica chamada *Planning Poker* (Molokken-Ostvold, K., Haugen, N.C. 2007), onde cada membro do time estima cada história do usuário em pontos*.*

Mesmo utilizando metodologias ágeis, onde as iterações são curtas e existam técnicas de estimativas em grupo, nem sempre as estimativas são precisas. Desta forma, projetos ágeis também carecem de técnicas para prover qualidade ao estimar histórias do usuário e *sprints.*

Equipes que utilizam métodos ágeis costumam trabalhar com Kanban. Kanban é um método que permite a uma equipe controlar cada estágio das tarefas que estão sendo executadas visualmente na forma de quadro. Este método permite que a equipe identifique gargalos no processo de desenvolvimento, e com algumas práticas, pode aumentar o fluxo de atividades e a quantidade de itens entregues. De acordo com Anderson, 2010, o Kanban é utilizado não só no setor de TI, mas também na produção de carros, como por exemplo, na fábrica da Toyota (ANDERSON, 2010). O Kanban pode ser criado de forma física ou virtual. Um dos *softwares* utilizados para trabalhar com Kanban virtual é o Trello. No Trello, as histórias do usuário são representadas por *cards* (cartões) que possuem a estimativa em pontos e outros detalhes da respectiva tarefa.

A proposta deste sistema é coletar a quantidade de pontos entregues ao final de cada *sprint* em kanbans virtuais criados no Trello, em projetos que utilizam a metodologia ágil Scrum*.* Com estes dados coletados, o sistema provê gráficos que mostram informações de apoio a decisão, como por exemplo, o histórico de pontos estimados e pontos entregues das *sprints* passadas.Isto acarreta em mais segurança para o fornecedor e para o cliente. A partir destas informações, a equipe de desenvolvimento terá uma base de dados para apoiar as estimativas, e o cliente, juntamente com a equipe, poderá priorizar os requisitos para a *sprint* evitando atrasos e custo não planejado. A proposta do trabalho é bem específica e não foram encontrados *softwares* similares, apenas sistemas que criam kanbans virtuais da mesma forma que o próprio Trello faz, sem a coleta de métricas ao final das *sprints*.

# 2 DEFINIÇÃO DO PROBLEMA

Um importante ingrediente para o sucesso de um projeto é a estimativa de esforço para a execução das tarefas, pois ela está diretamente ligada com o custo e com o prazo do projeto. De acordo com Arnaut (2008), apenas 35% dos projetos tiveram sucesso em 2006. As tarefas que serão executadas necessitam de uma estimativa. Com isto, a estimativa é um fator que delimita o prazo para a finalização do projeto, ou iteração, e o valor que o cliente deverá desembolsar.

Se os times de desenvolvimento de *software* pudessem prover estimativas de esforço precisas, este seria um cenário ideal tanto para o cliente quanto para o fornecedor do *software*. Os projetos poderiam ser vendidos com mais segurança para ambos os lados, cliente e time, estipulando o prazo e o custo sem surpresas ao final do projeto.

Projetos executados com metodologias ágeis, como o *Scrum*, tendem a ter entregas periódicas de funcionalidades específicas acordadas entre o cliente e o time de desenvolvimento.

A Figura 1 descreve o Scrum e suas fases, mostrando os *backlogs* do produto e da *sprint. Sprints* são os ciclos de desenvolvimento que possuem uma duração de uma a quatro semanas e ao seu final é entregue um pedaço funcional do sistema. A Figura 1 também mostra reuniões diárias que servem para alinhamento do time referente ao trabalho que está sendo executado. Ao final de cada *sprint* é feita uma reunião entre cliente e equipe para definir as histórias do usuário que deverão ser entregues no próximo ciclo.

Figura 1 – Processo básico do *Scrum*.

Fonte: (Eclipse Org, 2009).

Histórias do usuário são os itens que vão ficar no *backlog* do projeto. O time escreve as histórias do usuário baseando-se no ponto de vista do cliente. Pode-se considerar que a história do usuário é um requisito do sistema. (METHODOLOGY, 2009). Por exemplo, ao desenvolver uma calculadora, podemos definir as histórias do usuário como: subtração, adição, etc.

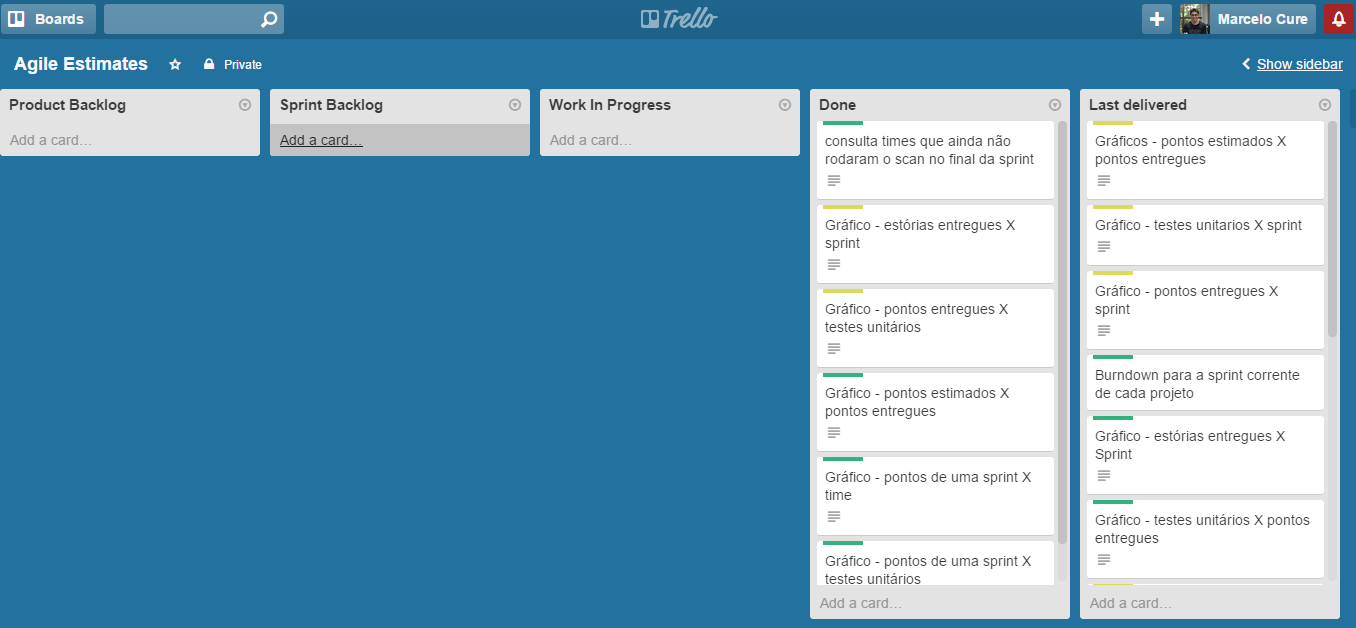
As histórias do usuário priorizadas para serem executadas na *sprint* devem ser estimadas para a equipe poder planejar de forma organizada a execução das mesmas. Normalmente as estimativas são feitas utilizando a técnica *Planning Poker*, onde cada história do usuário é estimada em pontos, que variam de 0 até 100 pontos (Molokken-Ostvold, K., Haugen, N.C., 2007).

O *Planning Poker* é uma técnica utilizada para auxiliar nas estimativas de esforço. Depois que a história do usuário é apresentada para os membros do time, estes a discutem para ter certeza de que todos entenderam. Cada membro do time escolhe uma pontuação representando o esforço que será necessário para implementar a história do usuário, e não mostra para nenhum outro membro. Após cada integrante do time ter escolhido uma pontuação, denominada *story point*, os mesmos discutem os resultados justificando-os e chegam a um denominador comum. Normalmente, diferentes membros do time provêm estimativas diferentes, por isto a discussão é importante para que todos cheguem a um consenso. Então o time passa a ter a estimativa para aquela tarefa específica.

É comum que a pontuação estimada no *Planning Poker* não seja precisa. Por exemplo, o time estimou a tarefa em 5 pontos, mas esta poderia ser estimada em 3 pontos já que o esforço gasto foi proporcional a uma tarefa de 3 pontos.

Este trabalho visa facilitar as estimativas de *sprints* baseando-se em dados históricos de *sprints* passadas. Ao planejar a *sprint*, será cadastrado a quantidade de pontos total que foi estimada, e ao final da iteração o sistema deverá coletar a quantidade total de pontos que foram entregues. Este sistema foi desenvolvido para projetos que utilizam o Trello como ferramenta de kanban virtual. O Trello possibilita a criação dos *boards* (quadros) virtuais, onde cada quadro pode ter quantas *lists* (listas) a equipe necessitar, por exemplo, um quadro específico pode ter as *lists* *backlog*, *work in progress* e *done.* As *lists* representam uma etapa do processo de desenvolvimento, e contém os *cards* (cartões) que representam uma história do usuário, onde é possível adicionar uma descrição, lista de tarefas, anexos, comentários, adicionar o usuário responsável pela tarefa, etc. A Figura 2 mostra um exemplo de quadro montado no Trello com cinco listas representando as etapas do desenvolvimento.

Figura 2 – Exemplo de quadro no Trello.



Fonte: Autoria própria.

Conforme representado na Figura 2, as histórias do usuário, representadas pelos *cards*, podem ser movidas de uma *list* para outra, conforme a necessidade, até chegar ao fim do processo.

Estes dados coletados são métricas relacionadas ao desempenho do time e a capacidade de abstrair o problema do cliente e codificar as funcionalidades de maneira eficiente. É possível extrair informações inteligentes destes dados, pois estes são o histórico das *sprints* que já passaram, isto significa que a equipe poderá extrair um aprendizado destas estatísticas que poderão ser utilizadas como suporte à tomada de decisão para futuras estimativas.

Mesmo utilizando técnicas como o *Planning Poker*, as estimativas nem sempre são precisas, ou boas o suficiente. Isto se torna um problema para o time de desenvolvimento sempre que for necessário estimar o esforço de histórias do usuário para uma *sprint*, e para o cliente que não tem segurança para prever a data de um *release* e o seu custo.

# 3 OBJETIVOS

Este capítulo descreve os objetivos gerais e específicos do projeto.

## 3.1 Objetivo Geral

O objetivo geral deste trabalho é melhorar as estimativas de *sprints*, onde as histórias são baseadas no *planning poker,* a partir de dados históricos referentes a estimativas de *sprints* anteriores.

## 3.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

1. Prover confiança para o time de desenvolvimento e para o cliente ao planejar a *sprint*, baseando-se em um histórico de estimativas e resultados obtidos;
2. Prover condições para melhorar a estimativa da *sprint*;
3. Integrar o sistema com o *software* *Trello* ([trello.com](http://www.trello.com/), 2014);
4. Coletar informações pertinentes dos *boards* do *Trello*;
5. Gerar gráficos para auxiliar nas estimativas futuras;
6. Prover, através de quantidade de testes unitários, uma idéia geral da qualidade do código.

# 4 ANÁLISE DE TECNOLOGIAS/FERRAMENTAS

Este capítulo descreve as tecnologias e ferramentas que foram utilizadas para o desenvolvimento do projeto, incluindo tecnologias de desenvolvimento, modelagem e técnicas.

## 4.1 Linguagens de programação

Este projeto foi desenvolvido utilizando a linguagem de programação *Python.* Esta linguagem provê facilidade no desenvolvimento (FONSECA e BRAGA, 2009) e possui uma variedade de bibliotecas. As APIs (Application Programming Interface) são simples e bem documentadas, assim como a própria linguagem. A facilidade na instalação de bibliotecas e na configuração do ambiente de desenvolvimento são qualidades essenciais. (LOTT, 2009).

Foi utilizado o framework *Django* para auxiliar no desenvolvimento das páginas, onde o desenvolvedor pode definir o modelo de dados utilizando a linguagem *Python*. Django é um *framework* que serve para o desenvolver aplicações web com *Python.* Este *framework* também possui uma API para integração com o banco de dados que abstrai as tabelas e através dos modelos podem ser executadas todas as operações necessárias (djangoproject.com, 2014).

## 4.2 Ferramenta de Desenvolvimento

O desenvolvimento foi feito utilizando uma ferramenta chamada *Sublime*. Esta ferramenta facilita o desenvolvimento pois ela oferece muitas funcionalidades úteis como *auto complete*, lista de arquivos do projeto, multi seleção, entre outras. Existe uma diversidade de *plug ins* que auxiliam no desenvolvimento. O *Sublime* é uma ferramenta extremamente leve. ([sublimetext.com](http://www.sublimetext.com/), 2014).

## 4.3 Banco de Dados

Os dados são armazenados no banco de dados relacional PostgreSQL. Além de possuir integração com o *Django*, o PostgreSQL é um banco de dados simples de utilizar, possibilitando buscar os dados de maneira a suprir as necessidades das funcionalidades do sistema (postgresql.org, 2014).

## 4.4 Controle de Atividades

As atividades e *sprints* foram controladas utilizando o *software* *Trello* ([trello.com](http://www.trello.com/), 2014), o qual é utilizado como um *Kanban* (ANDERSON, 2010). O *Trello* é um *software* grátis que possibilita a criação de *boards*, *lists* e *cards*. Os *boards* são a representação de um quadro do Kanban, onde é possível criar as *lists* que são cada fila do quadro representando as etapas do processo de desenvolvimento, como por exemplo, *work in progress*. Os *cards* representam cada tarefa, onde é possível adicionar uma descrição, comentários, anexos, pessoas responsáveis por aquela tarefa e outras informações. O Trello possibilita fazer o controle das *sprints* de forma organizada e visual. As tarefas das *sprints* podem ser arquivadas de forma a manter um histórico.

## 4.5 Versionamento do Código

O controle de versão de código e da própria documentação do projeto foi feito utilizando a ferramenta *GitHub* ([github.com](http://www.github.com/), 2014). Registrando um usuário, este software possibilita o cadastramento de projetos, criação de *bugs*, melhorias, etc. O *GitHub* ajudou o projeto na parte de versionamento de código e da documentação, onde tudo será salvo na nuvem. Com isto, tudo o que está versionado no *GitHub* pode ser recuperado em qualquer máquina conectada na *internet*.

## 4.6 Qualidade do Código

O sistema foi implementado utilizando um método de desenvolvimento chamado *TDD (Test Driven Development)*, que consiste em desenvolver o sistema baseando-se em testes. Antes de desenvolver uma funcionalidade, escreve-se testes unitários para esta (add a test). Estes testes, inicialmente, estarão falhando (run the tests), pois a funcionalidade ainda não foi implementada. O primeiro passo é fazer com que este passe, implementando a funcionalidadesrun the tests). Assim o desenvolvedor tem segurança em saber que aquela funcionalidade já possui testes unitários, provendo confiança também para refatorar o código (make a little change), após esta melhoria no código, os testes são executados novamente para garantir que o comportament não foi modificado. A qualidade do código pode ser medida pela cobertura de testes para cada módulo ou classe do sistema. O processo do *TDD,* explicado anteriormente, está descrito na Figura 3 (BECK, 2002).

Figura 3 - Estrutura básica do *TDD.*



Fonte: (SCOTT W. AMBLER, 2006).

## 4.7 Ferramentas de Modelagem

A criação de modelos e diagramas do sistema foi feita utilizando a ferramenta online *Cacoo*. Esta é uma ferramenta que possibilita a criação de diversos tipos de diagramas, como por exemplo diagrama ER (*Entity Relationship*) e diagramas da UML (*Unified Modeling Language*). (cacoo.com, 2014).

Foram gerados os diagramas de casos de uso, diagrama de classes e diagrama entidade relacionamento utilizando o Cacoo.

## 4.8 Protótipos de tela

Os protótipos de tela foram criados utilizando o *NinjaMock*. Esta ferramenta possibilita criar protótipos de telas de várias plataformas, como *Web*, *Android*, *iOS*, *WindowsPhone*, etc. O *NinjaMock* possibilita que os protótipos das telas sejam exportados nos formatos PDF (*Portable Document Format*) ou PNG (*Portable Network Graphics*). Para o projeto, os protótipos de tela foram exportados no formato PNG para serem adicionados no documento de projeto (ninjamock.com, 2014).

# 5 DESCRIÇÃO DA SOLUÇÃO

Esta solução baseia-se em dados históricos extraídos de estimativas de *sprints*. É um sistema específico para o auxílio às estimativas em projetos ágeis, onde as histórias do usuário são estimadas utilizando o *planning poker* e o projeto é controlado por um kanban virtual no software Trello. As estimativas das *sprints*, medidas em pontos, são extraídas e salvas em um banco de dados. Estes dados históricos são utilizados para geração de gráficos com a finalidade de prover auxílio ao time no momento de planejar uma nova *sprint*.

O sistema é restringido somente para usuários que possuem acesso. Os membros do time possuem acesso somente aos projetos que participam, onde tem as opções de cadastrar *sprints*, fazer a coleta de dados, visualizar dados referentes às *sprints* que passaram e visualizar gráficos com os dados extraídos anteriormente. O sistema inclui a área do administrador, onde os usuários com este perfil é capaz de cadastrar perfis, usuários, clientes, projetos e alocar recursos a estes projetos. Além disto, o administrador pode visualizar os gráficos referentes a todos os projetos. O usuário com perfil administrador pode consultar se existe algum projeto com o processo de *scan* pendente. Esta consulta verifica se algum dos projetos tem uma *sprint* sendo finalizada naquele dia e o time ainda não executou o processo de *scan* para coletar as métricas. Assim o administrador pode ter um controle melhor sobre o sistema e os projetos que estão em progresso.

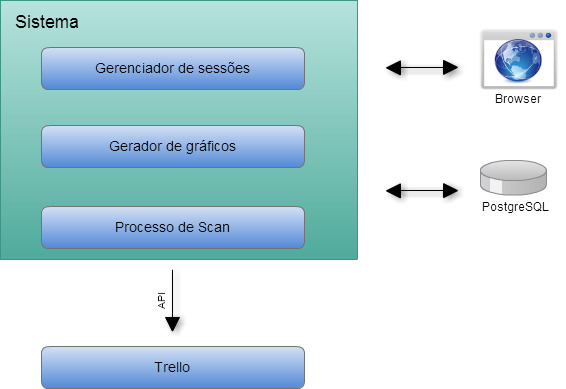
Considera-se que os projetos tenham acompanhamento na ferramenta online *Trello*. Esta ferramenta é utilizada para montar quadros Kanban virtuais, que possibilita a criação de *boards* onde o usuário pode criar *lists*, *cards*, adicionar informações, anexar arquivos, etc. Cada lista pode ser considerada uma etapa do processo de desenvolvimento, por exemplo *work in progress* seria a etapa onde a tarefa está sendo executada, logo existiria uma *list* com este título (trello.com, 2014).

O processo de coleta de métricas acessa o Trello através da API Trollop. O usuário seleciona um projeto e inicia a coleta. O sistema busca os *cards* que estão na última coluna do *board*, que é a coluna “*Done*”, onde estão as tarefas que foram concluídas na *sprint* em questão.

O sistema coleta os dados de cada *card*, como a sua identificação, data de início e fim da tarefa, estimativa em pontos e a quantidade de testes unitários criados para aquela tarefa. A última etapa da coleta de dados é salvá-los no banco de dados *PostgreSQL* e finalizar a *sprint* corrente.

O sistema disponibiliza gráficos com métricas referentes a um projeto específico ou a todos os projetos da empresa. Baseando-se nos dados coletados de *sprints* passadas, o sistema provê estes gráficos com diferentes visões destes dados, gerando informações que apoiam a tomada de decisão.

Figura 4 - Camadas e componentes do sistema.



Fonte: Autoria própria.

A Figura 4 mostra as camadas e componentes principais da aplicação, incluindo a comunicação com o *browser* (navegador web), banco de dados e com o *software* Trello. Foram criados módulos para controlar funcionalidades específicas do sistema, que são descritos nas seções a seguir.

## 5.1 Gerenciador de Sessões

O módulo Gerenciador de Sessões é responsável por controlar os acessos dos usuários. Quando o usuário faz a autenticação no sistema, os dados da sessão são armazenados no banco de dados e em *cookies* do *browser*.

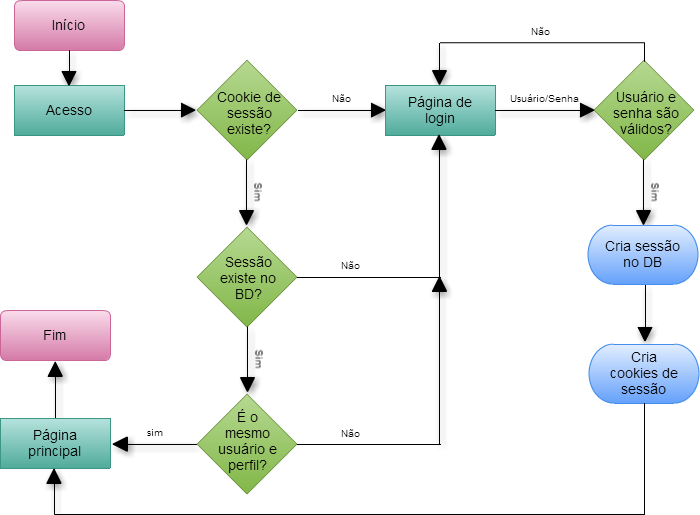
O *cookie* é utilizado para guardar informações referentes a sessão caso o usuário feche o *browser* sem clicar em *Logout*, no próximo acesso, o sistema identificará que já existe uma sessão para este usuário, não sendo necessário passar pelo processo de autenticação novamente.

As informações da sessão também são guardadas no banco de dados para evitar que usuários mal intencionados criem o *cookie* de sessão manualmente para ter acesso a informações referentes a outros perfis de usuário.

Este módulo também encaminha o usuário para a página adequada ao seu perfil.

A Figura 5 descreve com detalhes o processo de *login* de usuário.

Figura 5 - Processo de *login*.

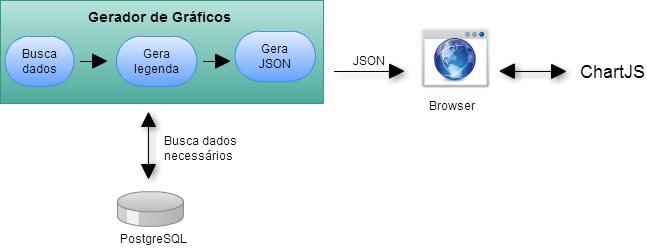


Fonte: Autoria própria.

## 5.2 Gerador de Gráficos

O módulo Gerador de Gráficos auxilia o sistema na criação dos gráficos baseados nos dados coletados. Os gráficos são gerados utilizando o ChartJS (chartjs.org, 2014). ChartJS é uma biblioteca escrita em *javascript* que gera gráficos baseando-se em um documento JSON (JavaScript Object Notation). Este módulo do sistema busca as informações do banco de dados e os transforma em um documento JSON incluindo detalhes sobre a formatação do gráfico. A biblioteca ChartJS não possui a funcionalidade de mostrar a legenda do gráfico indicando qual é o significado de cada linha. Dentro do módulo Gerador de Gráficos foi adicionada esta funcionalidade, onde a cor da linha é gerada aleatoriamente e passada para a página juntamente com a descrição. O módulo Gerador de Gráficos está melhor representado na figura 6.

Figura 6 - Gerador de Gráficos.



Fonte: Autoria própria.

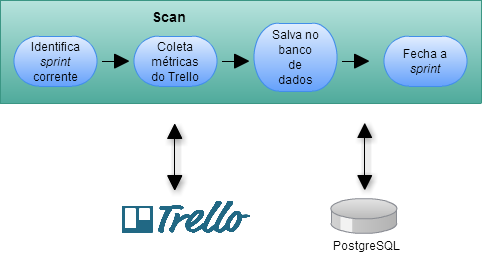
## 5.3 *Scan*

O módulo de *Scan* é responsável por fazer a leitura do Trello. Este processo acessa o Trello através da API Trollop (pypi.python.org/pypi/trollop, 2014) fazendo a busca e sumarização dos dados referentes a *sprint* corrente do projeto. No início do processo, este módulo identifica a *sprint* corrente e valida se o *scan* está sendo executado no útilimo dia da *sprint*, caso contrário o sistema aborta a execução avisando o usuário O sistema então busca todos os *cards* que estão na coluna "*done*" e coleta informações importantes, tais como o título da história do usuário, data de início e fim, estimativa em pontos e a quantidade de testes unitários criados. Ao final do processo de *scan* estes dados são salvos no banco de dados PostgreSQL e a *sprint* é finalizada.

O acesso ao Trello se dá através de um *token* de autenticação. Este *token* tem como responsabilidade controlar o acesso ao Trello para um usuário específico que tenha acesso aos *boards* da organização. O *token* de autenticação é gerado através do próprio Trello, que identifica o usuário autenticado no momento.

O módulo de *Scan* está representado na Figura 7.

Figura 7 - *Scan*.



Fonte: Autoria própria.

# 6 ABORDAGEM DE DESENVOLVIMENTO

O projeto foi desenvolvido utilizando o *Kanban* e algumas práticas do *Scrum* conforme descrito abaixo.

O *Scrum* é uma metodologia ágil de desenvolvimento de *software* que normalmente é utilizada em times pequenos, então algumas partes dele passam a ser relevantes para o cenário do projeto que possui uma equipe de um único membro. No projeto foi criado o *product backlog*, *user stories* (histórias de usuário), *sprints* e *sprint backlog*.

As tarefas são consideradas *user stories* (histórias do usuário)*,* e sua descrição apresenta uma ação a ser feita pelo usuário e o comportamento esperado. Estão sendo utilizadas as *sprints* para representar cada iteração do projeto, as *sprints* tem a duração de três semanas.

Utilizando o *Kanban* é possível ter de forma visual um acompanhamento do progresso da *sprint* corrente. O *Kanban* possibilita a identificação de gargalos no desenvolvimento, e através de algumas técnicas no gerenciamento do quadro, ele possibilita a melhoria no fluxo das atividades. O objetivo principal é finalizar as tarefas que já estão mais no lado direito do quadro, focando o trabalho nos itens que já estão sendo executados, resultando em mais itens entregues e maior fluxo de atividades.

O projeto foi dividido nas etapas descritas abaixo:

1. Criação do modelo de dados;
2. Estrutura básica do projeto com *Django*;
3. Criação de telas principais;
4. Integração com a *API* do *Trello*;
5. Processo de coleta de dados;
6. Geração dos gráficos.

## 6.1 Artefatos

A seguir, segue uma descrição dos artefatos gerados pelo projeto.

### 6.1.1 Product Backlog

O *Product Backlog* é uma lista, ordenada por prioridade, de tudo que é necessário para a construção do produto. O *Product Backlog* nunca está completo, ele é dinâmico e está mudando constantemente identificando o que o produto precisa para ser completo e apropriado.

### 6.1.2 Sprint Backlog

O *Sprint Backlog* é uma lista de itens extraídos do *Product Backlog* priorizados para serem executados na *sprint* corrente. Os itens são colocados na lista de forma ordenada, onde os itens do topo da lista são mais prioritários.

### 6.1.3 Histórias do Usuário

Histórias do usuário baseiam-se no ponto de vista do cliente. Pode-se considerar que a história do usuário é um requisito do sistema. As histórias do usuário podem ser definidas pelo time, baseando-se na perspectiva de quem vai utilizar a funcionalidade gerada por esta tarefa. Por exemplo, ao desenvolver uma calculadora, podemos definir as história como: subtração, adição, etc. (METHODOLOGY, scrum, 2009).

### 6.1.4 Diagrama de Caso de Uso

O digrama de caso de uso é um artefato da UML, que visa fazer a representação dos casos de uso do projeto, indicando os atores e as ações feitas de modo claro a identificar papéis e funcionalidades do sistema. Neste projeto, o diagrama de casos de uso é estritamente informativo, mostrando a interação dos atores com os pedaços do sistema e quais funcionalidades cada perfil de usuário acessa, assim passando uma ideia mais alto nível das ações de cada ator.

### 6.1.5 Diagrama de Classes

O digrama de classes é um artefato da UML, que visa mostrar de maneira detalhada a estrutura de classes e a comunicação entre elas. Este diagrama é baseado em um sistema escrito em uma linguagem orientada a objetos. Neste projeto, o diagrama de classes mostra as entidades principais do sistema passando uma ideia de como estas entidades relacionam-se entre si em alto nível.

### 6.1.6 Cobertura de testes

O sistema foi construído utilizando o TDD. Esta técnica visa desenvolver as unidades do programa baseando-se em testes unitários. Cria-se um teste unitário simples, inicialmente ele irá falhar, pois a funcionalidade ainda não está implementada, o próximo passo é fazer com que este teste passe, implementando o pedaço da funcionalidade esperada naquele teste. Após os teste estar passando, o desenvolvedor poderá refatorar aquele trecho de código, o teste unitário recém criado irá validar se qualquer comportamento foi modificado após as mudanças. E assim este processo se repetirá até a funcionalidade estar completa.

Os testes unitários são uma boa métrica de qualidade do código, porém, uma grande quantidade de testes unitários não garante a qualidade do sistema. Por isso, será disponibilizada a cobertura de testes no sistema que mostra a porcentagem de linhas de código que está sendo testada.

# 7 ARQUITETURA DO SISTEMA

Este capítulo demonstra a arquitetura básica do sistema, bem como os artefatos que foram gerados ajudando a descrever as funcionalidades do sistema.

## 7.1 Modelagem Funcional

Na modelagem funcional foram utilizadas as histórias do usuário, *Product Backlog* e *Sprint Backlog*. As tabelas abaixo mostram as *sprints* e suas respectivas histórias do usuário detalhadas e priorizadas.

### 7.1.1 Sprint 1

Tabela 1 – Sprint 1.

|  |  |
| --- | --- |
| Cadastros Básicos | |
| Cadastro de usuários | Eu como administrador devo ser capaz de cadastrar os  usuários com os seguintes campos: name, username, password,  email, profile. Este formulário ficará na parte esquerda da  tela. No outro lado aparecerá a lista de usuários já cadastrados. |
| Cadastro de perfis | Eu como administrador devo ser capaz de cadastrar os perfis  com os seguintes campos: descrição. Este formulário ficará na  parte esquerda da tela. No outro lado aparecerá a lista de  perfis já cadastrados. |
| Tela de login | Eu como administrador devo ser capaz de fazer login no  sistema e acessar a área do administrador. O processo de login  irá encaminhar o usuário para a tela especifica de acordo com  o seu perfil (admin ou time). Será desenvolvido um  controle de sessão utilizando uma tabela no banco de dados e  cookies do browser, criptografando a senha utilizando a  biblioteca PyCrypto. |
| Cadastro de clientes | Eu como administrador devo ser capaz de cadastrar os clientes  com os seguintes campos: name,country e Operation area.  Este formulário ficará na parte esquerda da tela. No outro  lado aparecerá a lista de clientes já cadastrados. |
| Cadastro de projetos | Eu como administrador devo ser capaz de cadastrar os projetos  com os seguintes campos:name, trello board name, customer e  listar todos os usuários com perfil “time”, possibilitando que  estes sejam alocados para o projeto. Este formulário ficará na  parte esquerda da tela. No outro lado aparecerá a lista de  projetos já cadastrados. |

### 7.1.2 Sprint 2

Tabela 2 – Sprint 2.

|  |  |
| --- | --- |
| Scan e Cadastro de sprints | |
| Cadastro de sprint | Eu como membro de um time devo ser capaz de cadastrar as  *Sprints* com os seguintes campos: Project, description, start  date, end date, estimated points. Este formulário ficará na  parte esquerda da tela. |
| Scan parte 1 | Eu como membro de um time devo ser capaz de executar o processo de scan de um projeto específico, onde o  usuário selecionará o projeto e o sistema fará o scan de acordo  com o projeto. O processo de scan inicial será simples, este  acessará o *board* do projeto no *Trello,* as informações coletadas  serão mostradas na tela de maneira simples. Ao final do processo  os dados referentes ao processo de scan serão exibidos na tela  com uma mensagem de confirmação. |

### 7.1.3 Sprint 3

Tabela 3 – Sprint 3.

|  |  |
| --- | --- |
| Scan final e gráficos do administrador | |
| Scan parte 2 | Eu como membro de um time devo ser capaz de executar o  processo de *scan*. Modificar o processo para salvar os dados no  banco de dados, este passo deve atualizar o registro da *sprint*  que foi previamente cadastrada. |
| Quantidade de pontos  entregues X *sprint* | Eu como administrador devo ser capaz de gerar um gráfico de  linhas para cada time mostrando a quantidade de pontos  entregues por *sprint*. |
| Quantidade de testes  unitários criados X *sprint* | Eu como administrador devo ser capaz de gerar um gráfico de  linhas para cada time mostrando a quantidade de testes unitários  criados por *sprint*. |

### 7.1.4 Sprint 4

Tabela 4 – Sprint 4.

|  |  |
| --- | --- |
| Gráficos do administrador final | |
| Quantidade de pontos  estimados X pontos  entregues | Eu como administrador devo ser capaz de gerar um gráfico de  linhas para cada time mostrando a quantidade de pontos  estimados por quantidade de pontos entregues. |
| Quantidade de pontos  entregues X testes unitários  para cada *sprint* | Eu como administrador devo ser capaz de gerar um gráfico de  linhas para cada time mostrando a quantidade de testes unitários  criados por pontos entregues por *sprint*. |
| Quantidade de  histórias entregues X  sprint | Eu como administrador devo ser capaz de gerar um gráfico de  linhas para cada time mostrando a quantidade de pontos  entregues por *sprint*. |

### 7.1.5 Sprint 5

Tabela 5 – Sprint 5.

|  |  |
| --- | --- |
| Gráficos do time | |
| Quantidade de pontos  entregues X *sprint* | Eu como membro de um time devo ser capaz de gerar um  gráfico de linhas mostrando a quantidade de pontos  entregues por *sprint*. |
| Quantidade de testes  unitários criados X *sprint* | Eu como membro de um time devo ser capaz de gerar um  gráfico de de linhas mostrando a quantidade de testes  unitários criados por *sprint*. |
| Quantidade de pontos  estimados X pontos  entregues | Eu como membro de um time devo ser capaz de gerar um  gráfico de de linhas mostrando a quantidade de pontos estimados  por quantidade de pontos entregues. |
| Quantidade de pontos  entregues X testes unitários  para cada *sprint* | Eu como membro de um time devo ser capaz de gerar um  gráfico de de linhas mostrando a quantidade de testes  unitários criados por pontos entregues por *sprint*. |

### 7.1.6 Sprint 6

Tabela 6 – Sprint 6.

|  |  |
| --- | --- |
| Gráficos time final e consulta scans pendentes | |
| Quantidade de pontos  entregues X testes unitários  para cada *sprint* | Eu como membro de um time devo ser capaz de gerar um  gráfico de de linhas mostrando a quantidade de testes  unitários criados por pontos entregues por *sprint*. |
| Quantidade de  histórias entregues por  *sprint* | Eu como membro de um time devo ser capaz de gerar um  gráfico de de linhas mostrando a quantidade de histórias  criadas por *sprint*. |
| Consulta times que  ainda não rodaram o  scan no final da *sprint* | Eu como administrador devo ser capaz de ver as *sprints* que  estão sendo finalizadas no dia corrente e que o time ainda  não executou o processo de *scan*. |

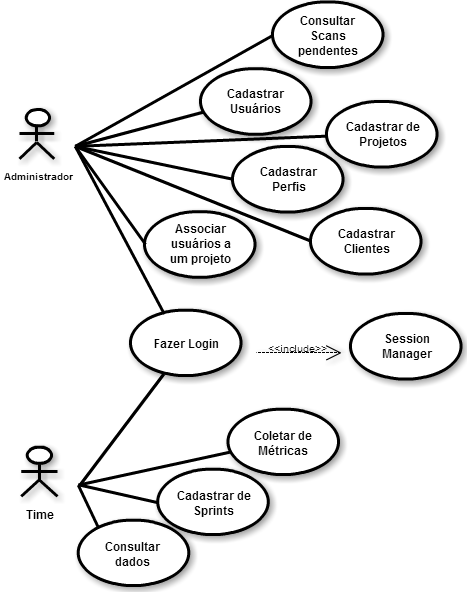
## 7.2 Modelos UML

Na modelagem UML, foram gerados os diagramas de classe e de casos de uso. Inicialmente foi criada uma versão simples, estes modelos evoluiram conforme as *sprints* foram finalizadas.

### 7.2.1 Diagrama de Casos de Uso

O diagrama de casos de uso está descrito na figura 8.

Figura 8 – Diagrama de casos de uso.

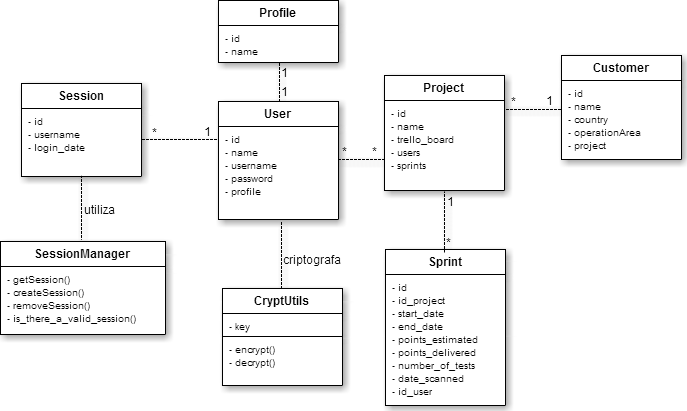


Fonte: Autoria própria.

### 7.2.2 Diagrama de Classes

O diagrama de classes está descrito na figura 9.

Figura 9 – Diagrama de classes.



Fonte: Autoria própria.

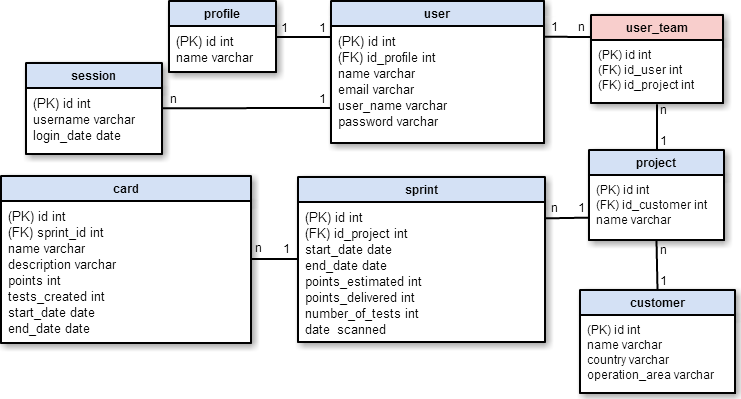
## 7.3 Modelagem de Dados

Na modelagem de dados, foram criados o diagrama ER e o dicionário de dados que estão descritos nas seções abaixo.

### 7.3.1 Diagrama ER

O modelo Entidade Relacionamento foi gerado para demonstrar a modelagem de dados do sistema. Inicialmente foi criada uma versão simples, e este diagrama evoluiu conforme as *sprints* foram finalizadas. A Figura 10 descreve o diagrama ER.

Figura 10 – Diagrama Entidade Relacionamento.



Fonte: Autoria própria.

### 7.3.2 Dicionário de Dados

O dicionário de dados está detalhado na Tabela 7, que mostra a descrição de cada tabela, dos campos e demais detalhes:

Tabela 7 - Dicionário de dados

|  |  |  |  |
| --- | --- | --- | --- |
| **1. *card* - Tabela que guarda informações dos *cards* (cartões)** | | | |
| Campo | Tipo | Nulo | Comentário |
| Id | Integer | Não | PK da tabela, *auto increment* |
| Sprint\_id | Integer | Não | FK para *sprint*, campo que representa a *sprint* |
| Name | Varchar(100) | Não | Nome do *card* |
| url | Varchar(200) | Não | URL do *card* para abrir no Trello |
| Description | Varchar(1000) | Não | Descrição detalhada do *card* |
| Start\_date | Date | Não | Data de início da tarefa |
| End\_date | Date | Não | Data do fim da tarefa |
| Points | Double | Sim | Número de pontos estimados |
| Tests\_created | Integer | Sim | Número de testes unitários criados para a tarefa |
| **2. *customer* - Tabela que guarda informações dos clientes da organização** | | | |
| Campo | Tipo | Nulo | Comentário |
| Id | Integer | Não | PK da tabela, *auto increment* |
| Name | Varchar(60) | Não | Nome do cliente |
| Country | Varchar(30) | Não | País onde o cliente reside |
| Operation\_area | Varchar(60) | Não | Área de operação que o cliente atua |
| **3. *profile* - Tabela que guarda informações dos perfis de usuário** | | | |
| Campo | Tipo | Nulo | Comentário |
| Id | Integer | Não | PK da tabela, *auto increment* |
| Name | Varchar(30) | Não | Nome do perfil |
| **4. project - Tabela que guarda informações dos projetos de um cliente** | | | |
| Campo | Tipo | Nulo | Comentário |
| Id | Integer | Não | PK da tabela, *auto increment* |
| Name | Varchar(30) | Não | Nome do projeto |
| Customer\_id | Integer | Não | FK para *customer*, campo que representa o cliente |
| Trello\_board | Varchar(100) | Não | Id do *board* usado para acessar o *board* do Trello. |
| **5. *session*- Tabela que guarda informações das sessões dos usuários** | | | |
| Campo | Tipo | Nulo | Comentário |
| Id | Integer | Não | PK da tabela, *auto increment* |
| Username | Varchar(30) | Não | Usuário autenticado nesta sessão |
| Login\_date | Date | Não | Data que a sessão iniciou |
| **6. *sprint*- Tabela que guarda informações das *sprints* dos projetos** | | | |
| Campo | Tipo | Nulo | Comentário |
| Id | Integer | Não | PK da tabela, *auto increment* |
| Description | Varchar(600) | Não | Descrição da *sprint* |
| Project\_id | Integer | Não | FK para *project*, campo que representa o projeto |
| Start\_date | Date | Não | Data de início da *sprint* |
| end\_date | Date | Não | Data final da *sprint* |
| Points\_estimated | Double | Não | Pontos estimados |
| Points\_delivered | Double | Sim | Pontos entregues, será preenchido durante o *scan* |
| Number\_of\_tests | Integer | Sim | Número de testes, será preenchido durante o *scan* |
| Date\_scanned | Date | Sim | Data do *scan*, será preenchido durante o *scan* |
| **7. *user* - Tabela que guarda informações dos usuários** | | | |
| Campo | Tipo | Nulo | Comentário |
| Id | Integer | Não | PK da tabela, *auto increment* |
| Name | Varchar(30) | Não | Nome da pessoa que possui este usuário |
| Username | Varchar(30) | Não | Nome do usuário |
| Password | Varchar(100) | Não | Senha criptografada do usuário |
| Email | Varchar(100) | Não | Email de contato do usuário |
| Profile\_id | Integer | Não | FK para *profile*, campo que representa o perfil |
| **8. *user\_project* - Tabela que guarda projetos dos usuários** | | | |
| Campo | Tipo | Nulo | Comentário |
| Id | Integer | Não | PK da tabela, *auto increment* |
| User\_id | Integer | Não | FK para *user*, campo que representa o usuário |
| Project\_id | Integer | Não | FK para *project*, campo que representa o projeto |

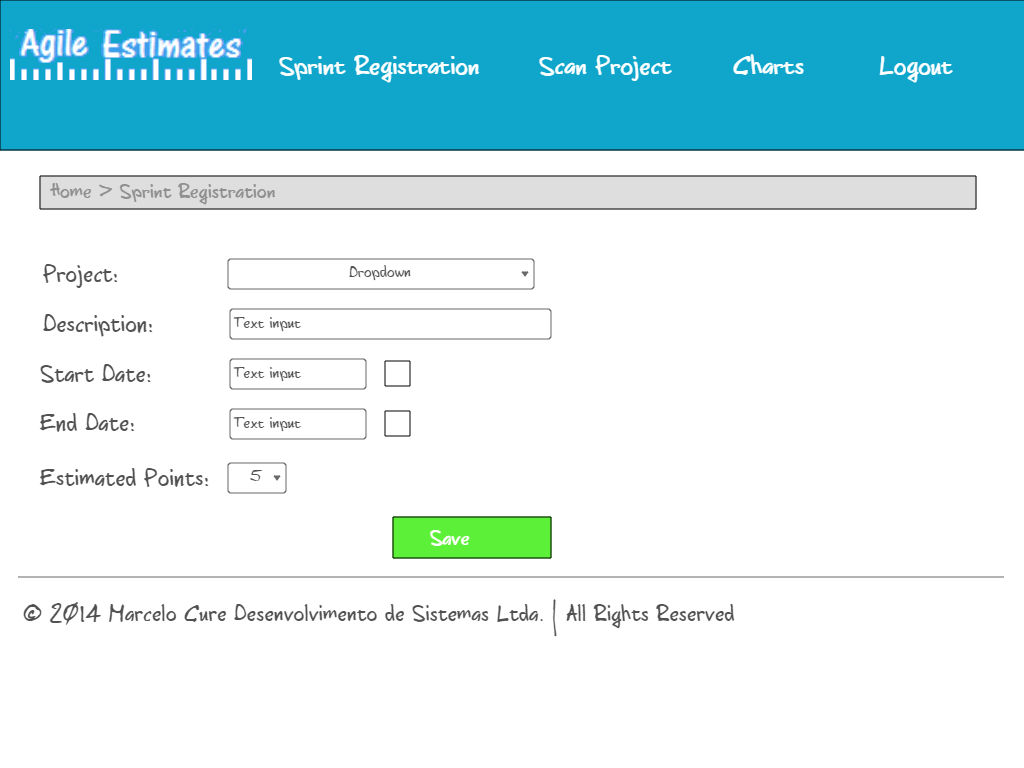
## 7.4 Interface Gráfica

Foram criados protótipos de telas do sistema para ter, de uma maneira visual, uma idéia de como as telas foram construídas. Desta maneira é possível fazer uma análise de usabilidade para deixar a navegação/execução das tarefas clara e objetiva.

### 7.4.1 Cadastro de Sprints

A figura 11 mostra o protótipo da tela de cadastro de *sprints*.

Figura 11 – Protótipo da tela de cadastro de *sprints*.

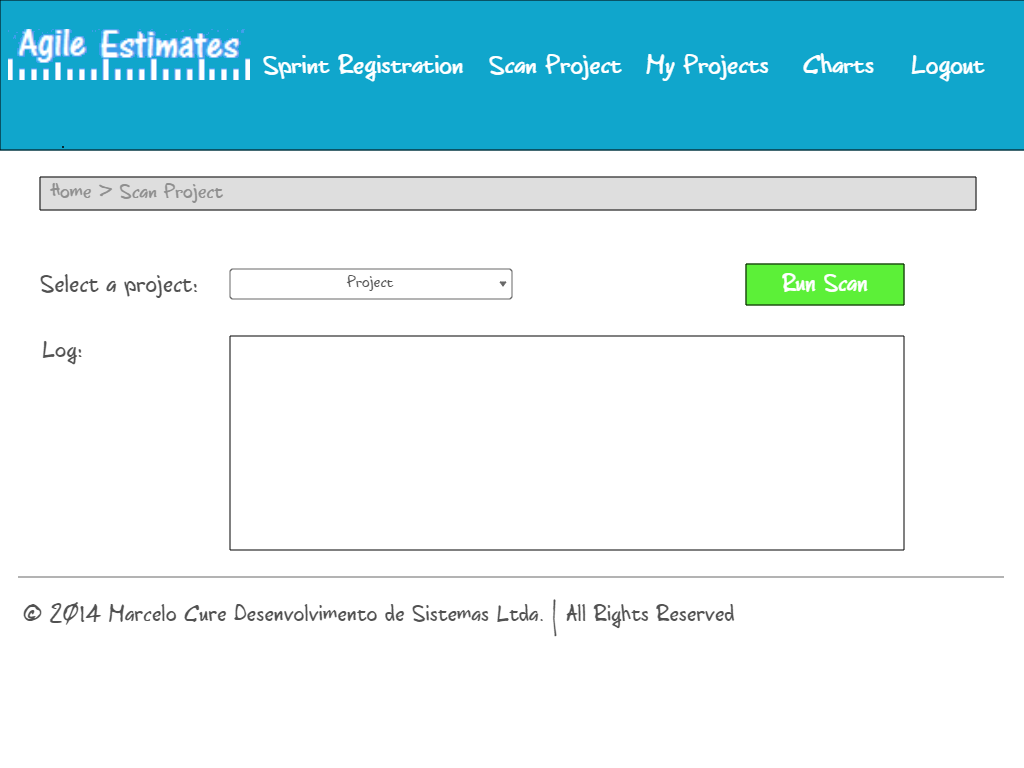


Fonte: Autoria própria.

### 7.4.2 Scan de projeto

A figura 12 mostra o protótipo da tela de scan do projeto.

Figura 12 – Protótipo da tela de Scan do projeto.

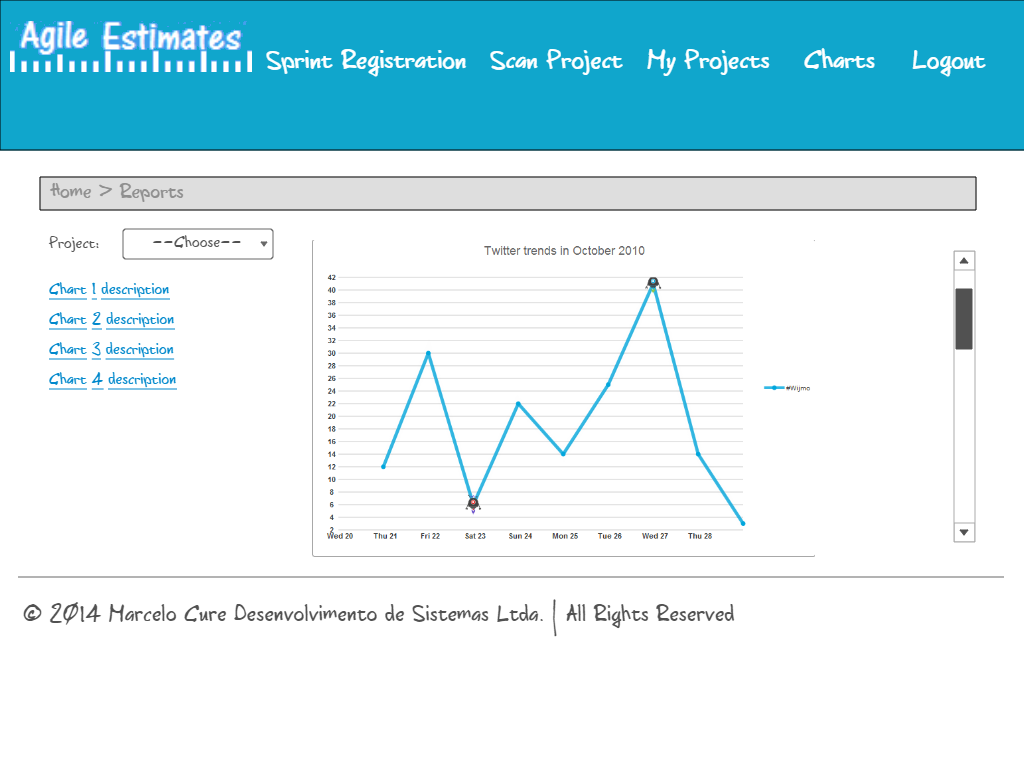


Fonte: Autoria própria.

### 7.4.3 Gráficos

A figura 13 mostra o protótipo da tela de gráficos.

Figura 13 – Protótipo da tela de gráficos.

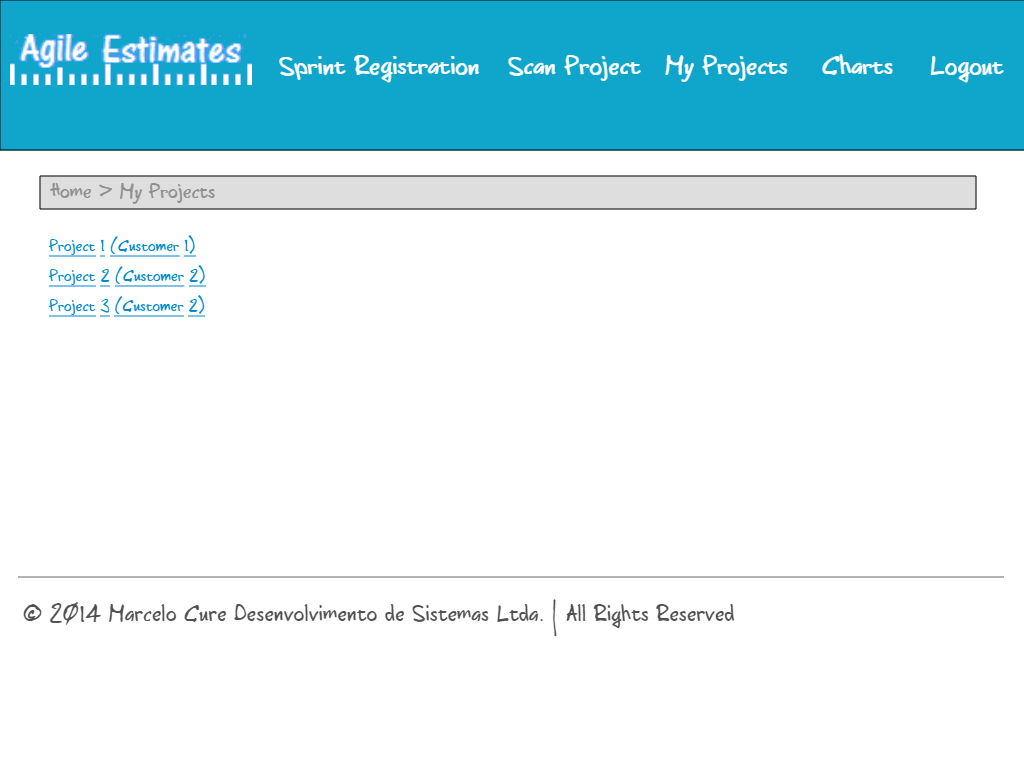


Fonte: Autoria própria.

### 7.4.4 Projetos do usuário

A figura 14 mostra o protótipo da tela de projetos do usuário.

Figura 14 – Protótipo da tela de projetos do usuário.

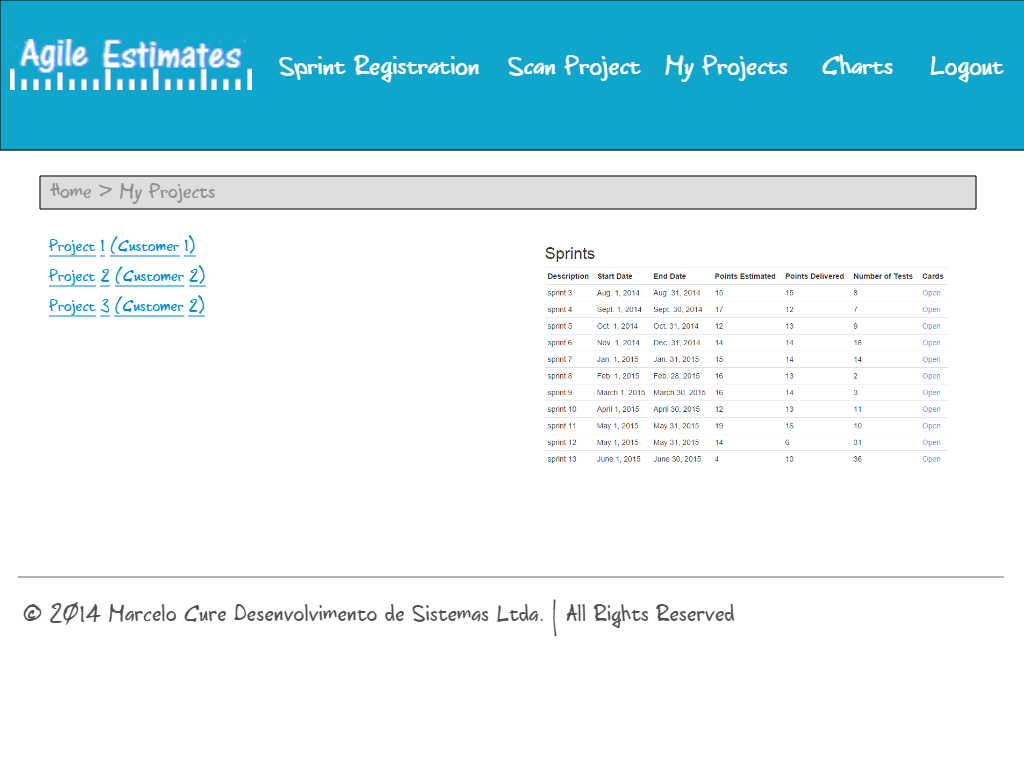


Fonte: Autoria própria.

### 7.4.5 Projetos do usuário - *Sprints*

A figura 15 mostra o protótipo da tela de *sprints* dos projetos do usuário.

Figura 15 – Protótipo da tela de *sprints* dos projetos do usuário.

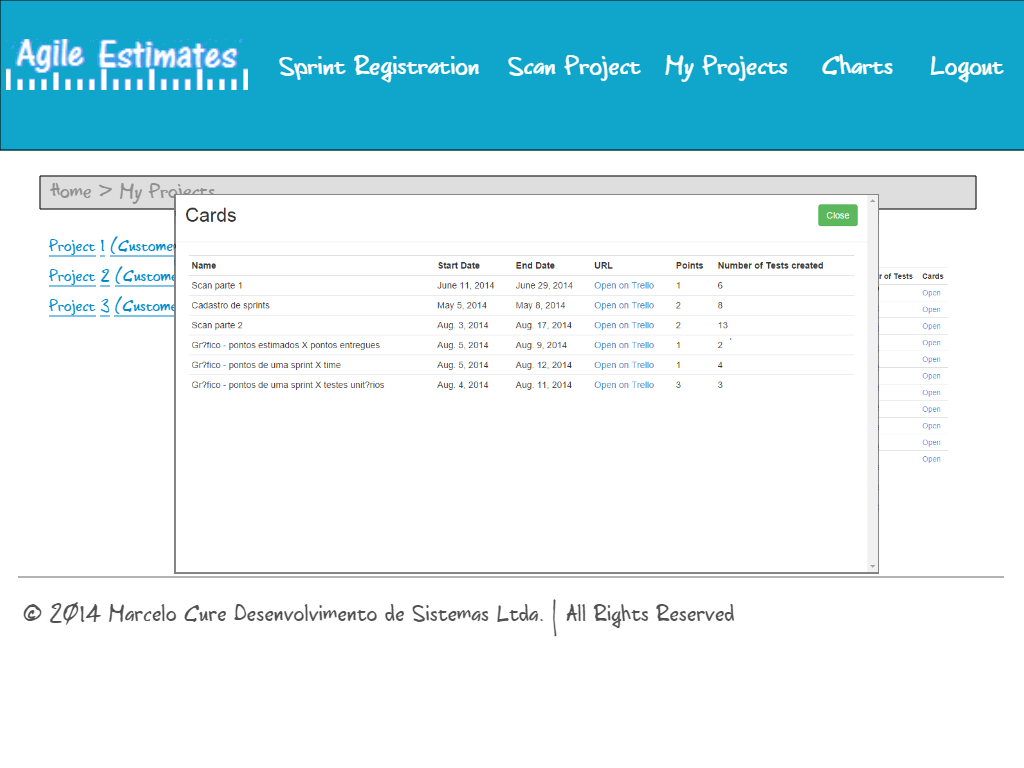


Fonte: Autoria própria.

### 7.4.6 Projetos do usuário - *Sprint cards*

A figura 16 mostra o protótipo da tela de *cards* de *sprints* dos projetos do usuário.

Figura 16 – Protótipo da tela de *cards* de *sprints* dos projetos do usuário.

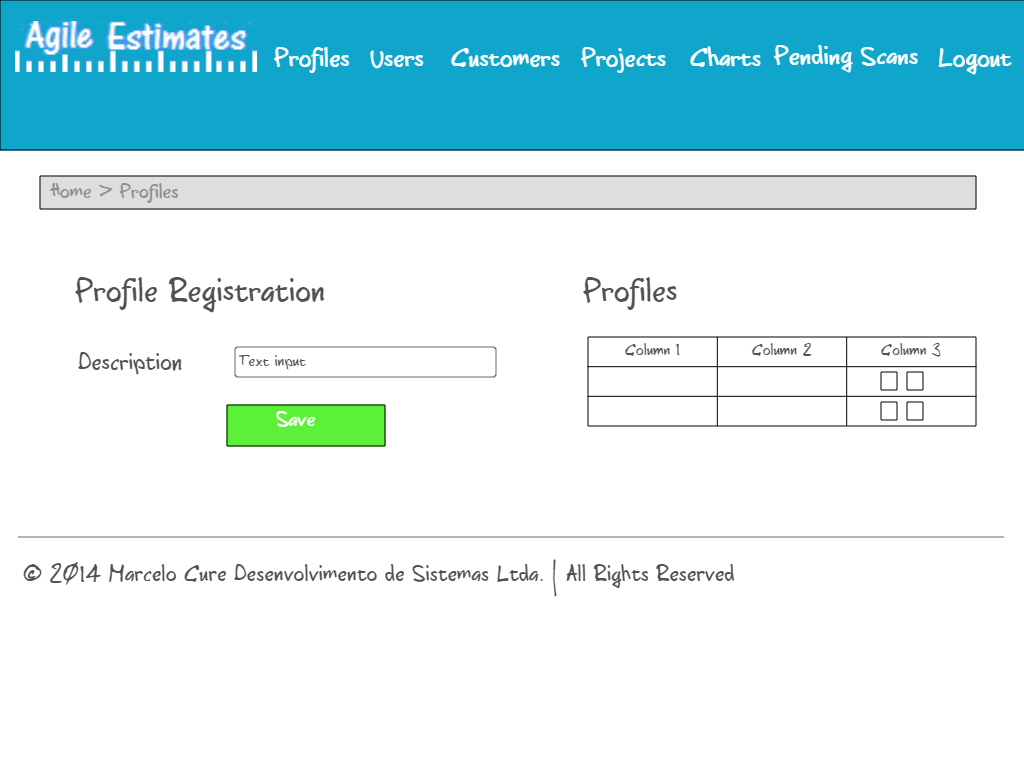


Fonte: Autoria própria.

### 7.4.7 Cadastro de perfis

A figura 17 mostra o protótipo da tela de cadastro de perfis.

Figura 17 – Protótipo da tela de cadastro de perfis.

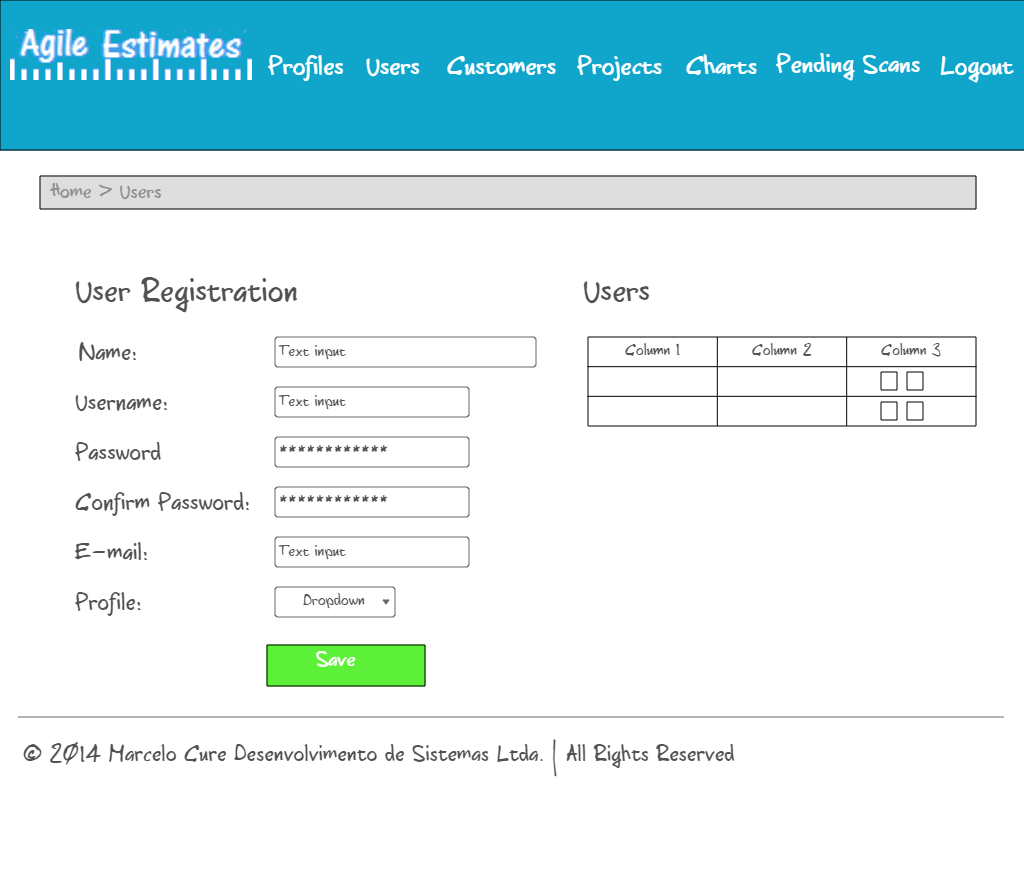


Fonte: Autoria própria.

### 7.4.8 Cadastro de usuários

A figura 18 mostra o protótipo da tela de cadastro de usuários.

Figura 18 – Protótipo da tela de cadastro de usuários.

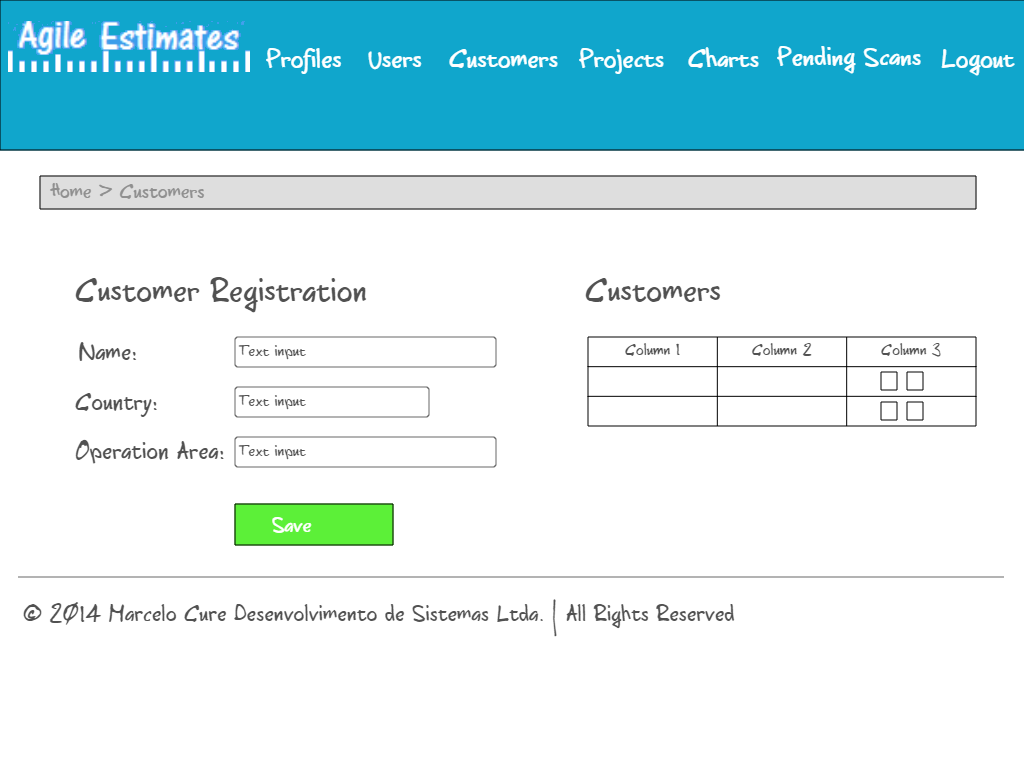


Fonte: Autoria própria.

### 7.4.9 Cadastro de clientes

A figura 19 mostra o protótipo da tela de cadastro de clientes.

Figura 19 – Protótipo da tela de cadastro de clientes.

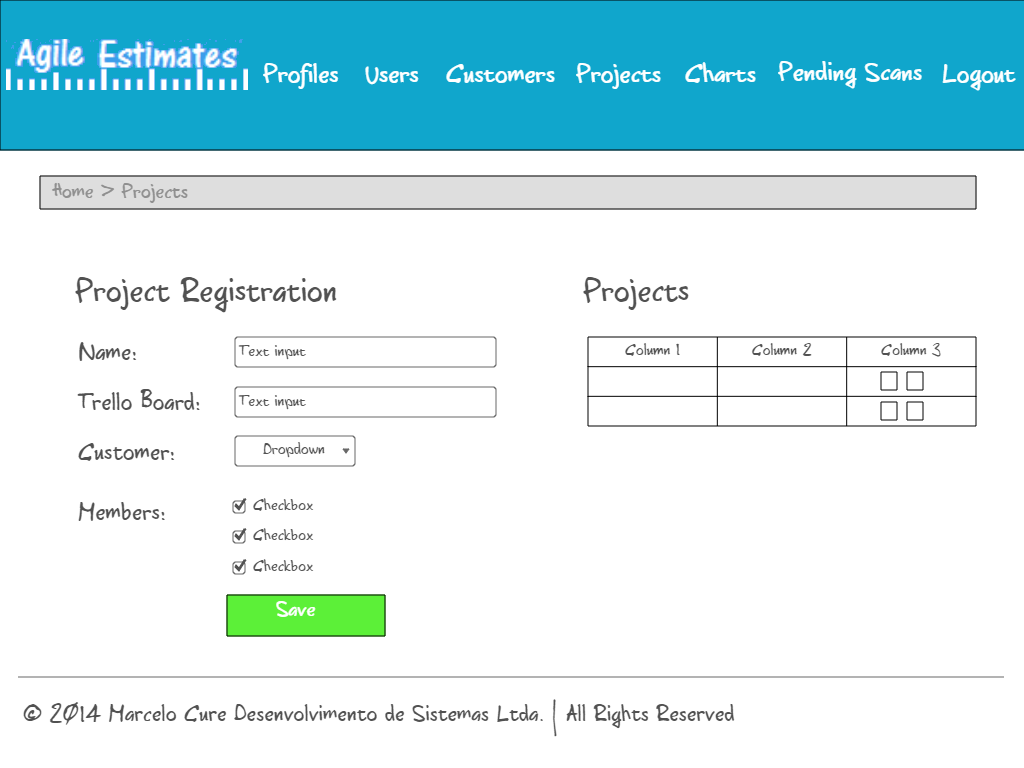


Fonte: Autoria própria.

### 7.4.10 Cadastro de projetos

A figura 20 mostra o protótipo da tela de cadastro de projetos.

Figura 20 – Protótipo da tela de cadastro de projetos.

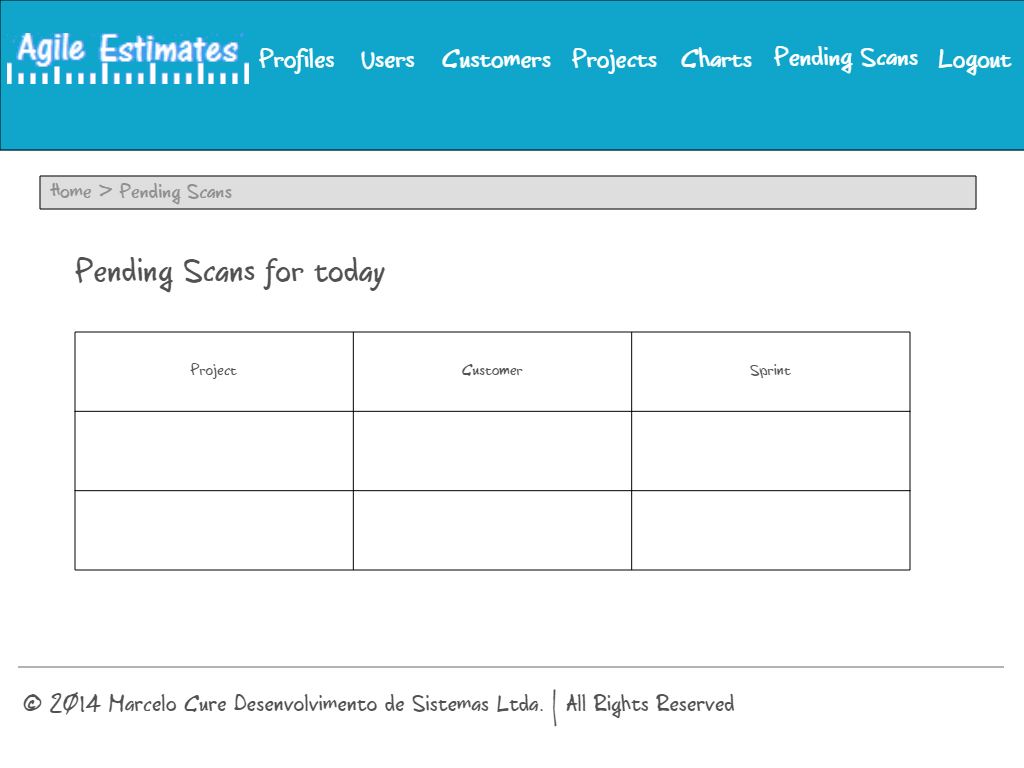


Fonte: Autoria própria.

### 7.4.11 Consulta de scans pendentes

A figura 21 mostra o protótipo da tela de scans pendentes.

Figura 21 – Protótipo da tela de scans pendentes.

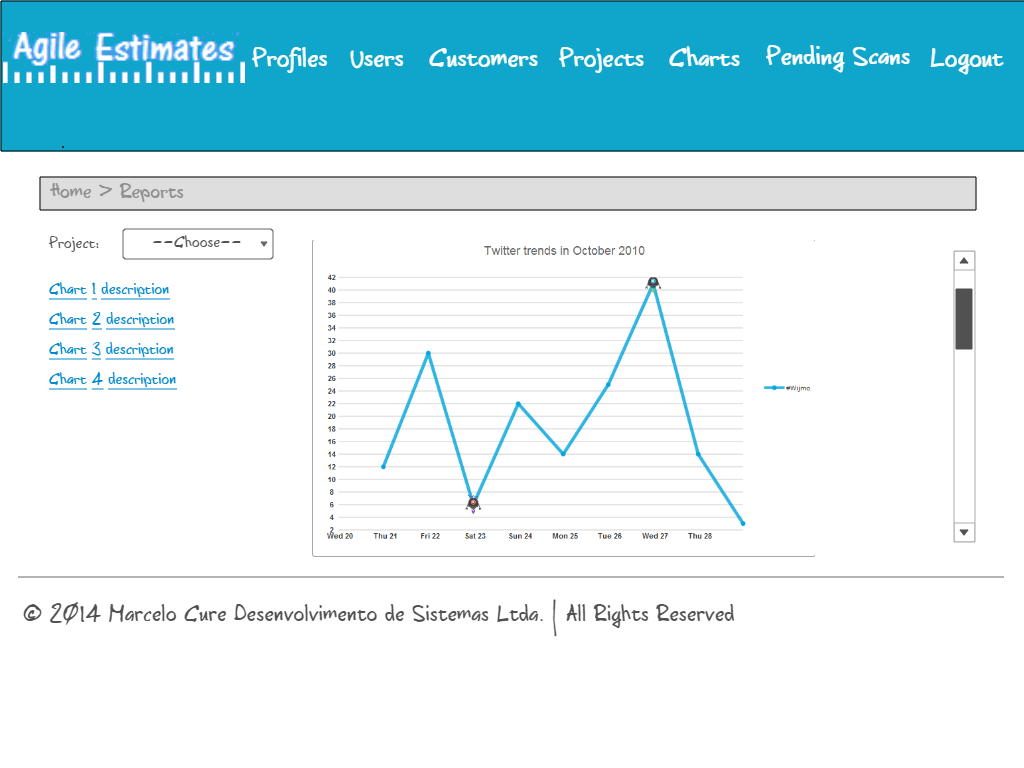


Fonte: Autoria própria.

### 7.4.12 Gráficos

A figura 22 mostra o protótipo da tela de gráficos.

Figura 22 – Protótipo da tela de gráficos.

Fonte: Autoria própria.

## 

## 7.5 Processo de Negócio

O processo de negócio está descrito na figura 23.

Figura 23 – Processo de negócio do sistema.



Fonte: Autoria própria.

## 7.6 Testes Unitários

Esta seção descreve, respectivamente, os testes unitários criados e a cobertura de testes do sistema.

### 7.6.1 Testes criados

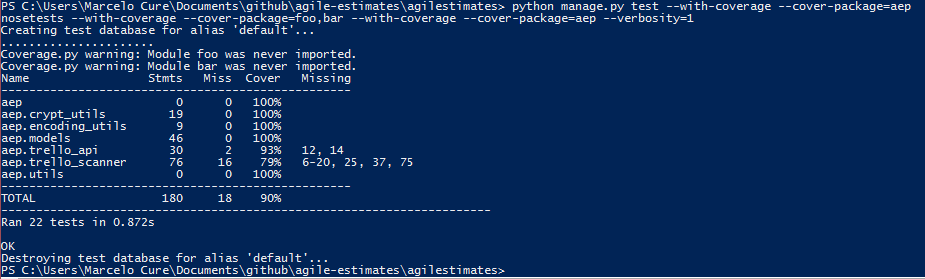
Os testes unitários estão descritos abaixo:

1. test\_get\_connection;
2. test\_remove\_label;
3. test\_remove\_label\_no\_label\_to\_remove;
4. test\_get\_field;
5. test\_get\_field\_not\_found;
6. test\_count\_points\_delivered;
7. test\_count\_unit\_tests;
8. test\_summarize\_metrics;
9. test\_build\_output\_dict;
10. test\_parse\_cards;
11. test\_normalize\_value\_unicode;
12. test\_normalize\_value\_string;
13. test\_decrypt;
14. test\_encrypt;
15. test\_get\_board;
16. test\_get\_board\_error;
17. test\_get\_done\_list;
18. test\_get\_done\_list\_not\_found;
19. test\_get\_list\_cards;
20. test\_get\_list\_cards\_error
21. test\_remove\_accents\_with\_accents;
22. test\_remove\_accents\_without\_accents.

### 7.6.2 Cobertura de testes

A figura 24 mostra a cobertura de testes unitários do sistema.

Figura 24 – Cobertura de testes do sistema.



Fonte: Autoria própria.

# 8 FUNCIONAMENTO DO SISTEMA

O sistema possui dois perfis de acesso, o membro do time e o administrador. As seções a seguir descrevem detalhadamente as funcionalidades de cada perfil de usuário mostrando as telas em questão.

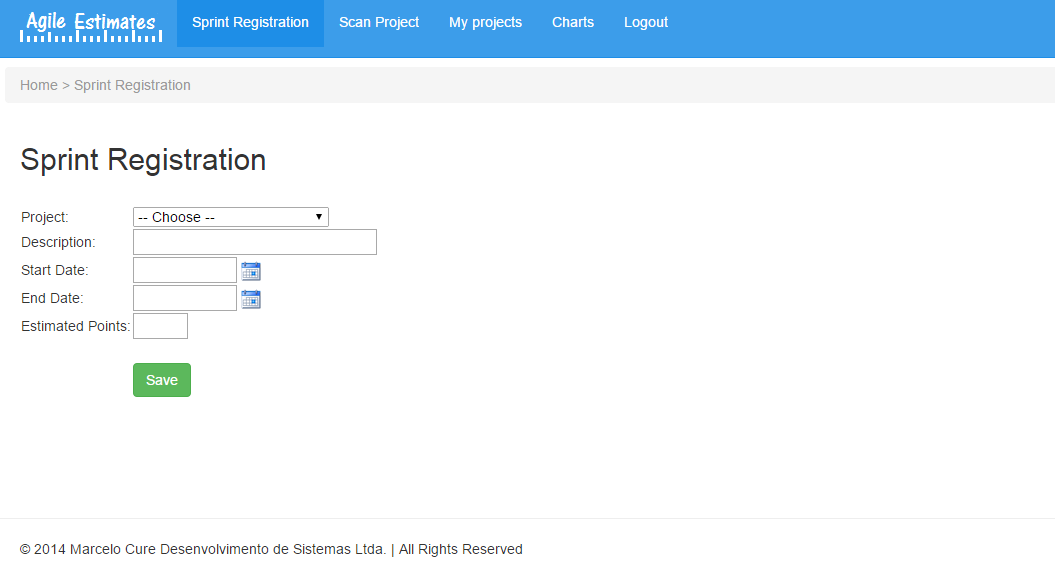
## 8.1 Membro do time

O membro do time pode participar de um ou mais projetos e suas funcionalidades estão descritas nas seções abaixo.

### 8.1.1 Cadastro de Sprints

Usuários com o perfil do time de desenvolvimento são capazes de cadastrar as *sprints* do projeto. O cadastro de *sprints* inclui o projeto em questão, o período de duração, quantidade de pontos estimados e a descrição das mesma. A Figura 25 mostra a tela de cadastro de *sprints*.

Figura 25 – Tela de cadastro de *sprints*.

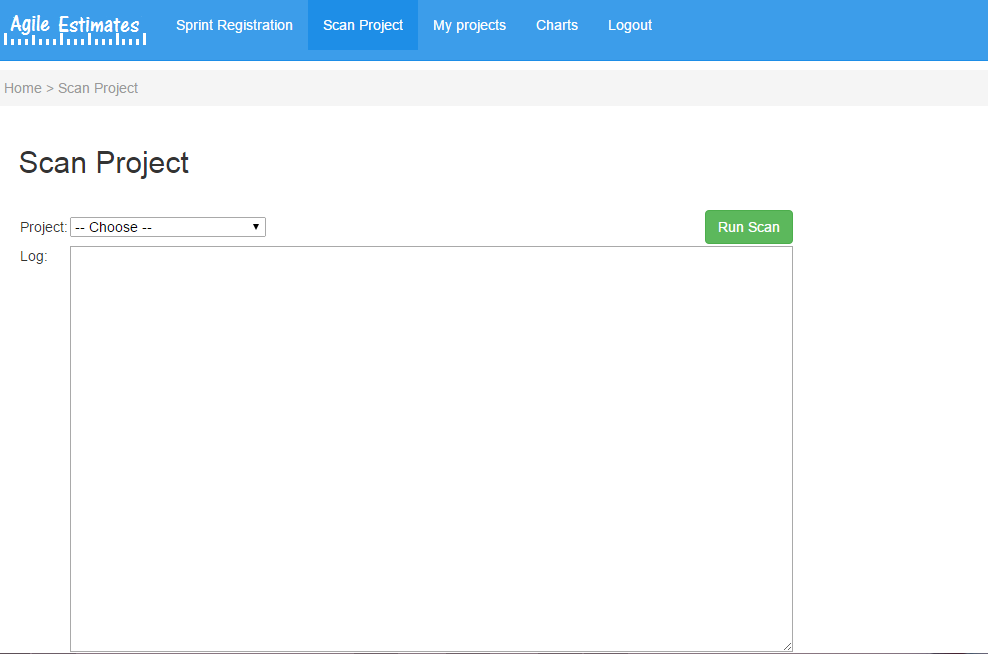


Fonte: Autoria própria.

### 8.1.2 Scan de projeto

O usuário membro do time tem a opção de fazer o scan do projeto, onde o usuário seleciona um projeto e o sistema acessa o *board* do projeto no *Trello* fazendo uma leitura de cada *card* da coluna *done*. Este processo coleta informações importantes de cada um destes *cards* como as datas de início e fim da tarefa, quantidade de testes unitários criados, número de pontos estimados e descrição. Ao final do processo o sistema irá concluir a *sprint*, colocando em seu cadastro estes dados coletados que serão utilizados para gerar os gráficos do sistema. A figura 26 mostra a tela de scan do projeto.

Figura 26 – Tela de Scan do projeto.



Fonte: Autoria própria.

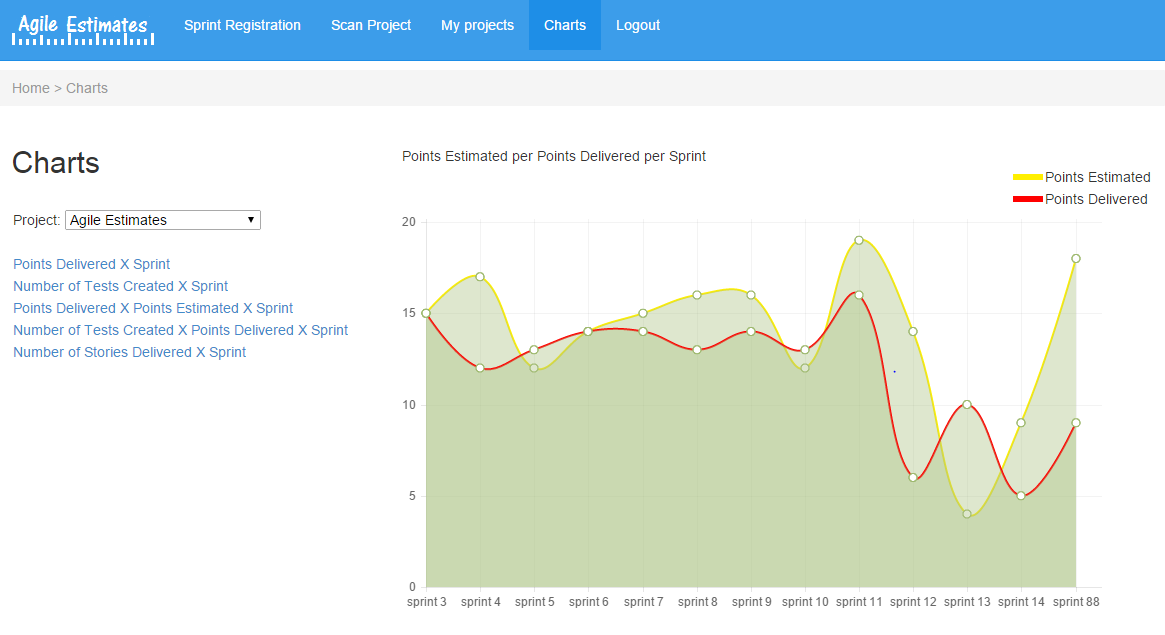
### 8.1.3 Gráficos

Com os dados coletados de *sprints* passadas, o sistema disponibiliza gráficos com diferentes visões destes dados conforme descrito abaixo:

1. Quantidade de pontos entregues X *sprint*;
2. Quantidade de testes unitários criados X *sprint*;
3. Quantidade de pontos estimados X quantidade de pontos entregues para cada *sprint*;
4. Quantidade de pontos entregues X testes unitários para cada *sprint*;
5. Quantidade de histórias entregues por *sprint* por projeto.

A Figura 27 mostra a tela de gráficos do usuário.

Figura 27 – Tela de gráficos do usuário.

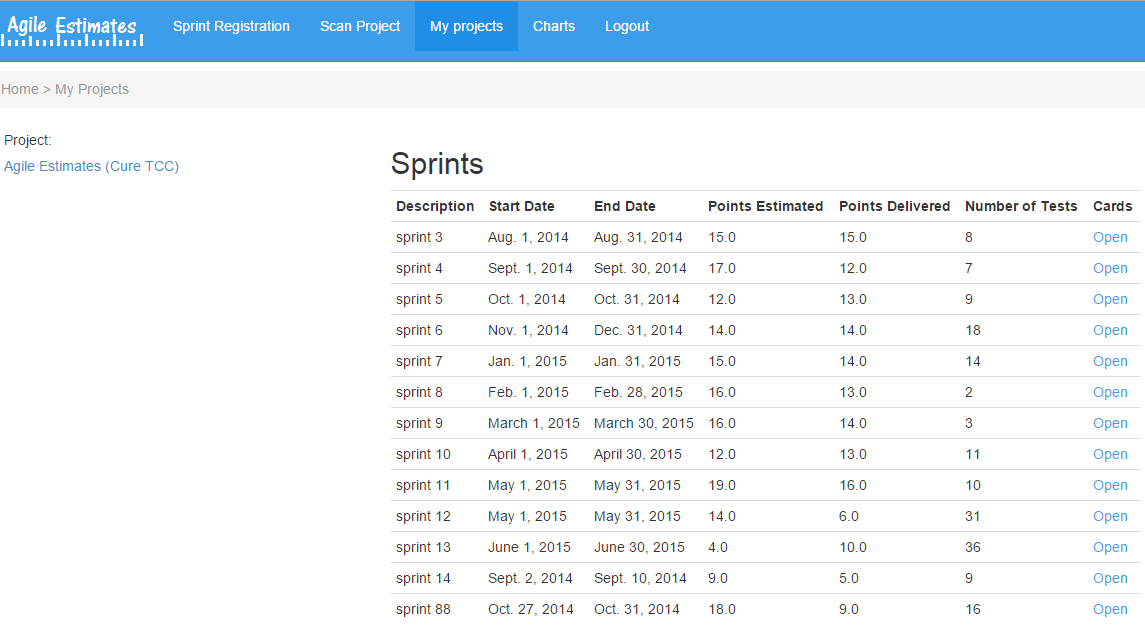


Fonte: Autoria própria.

### 8.1.4 Projetos do Usuário

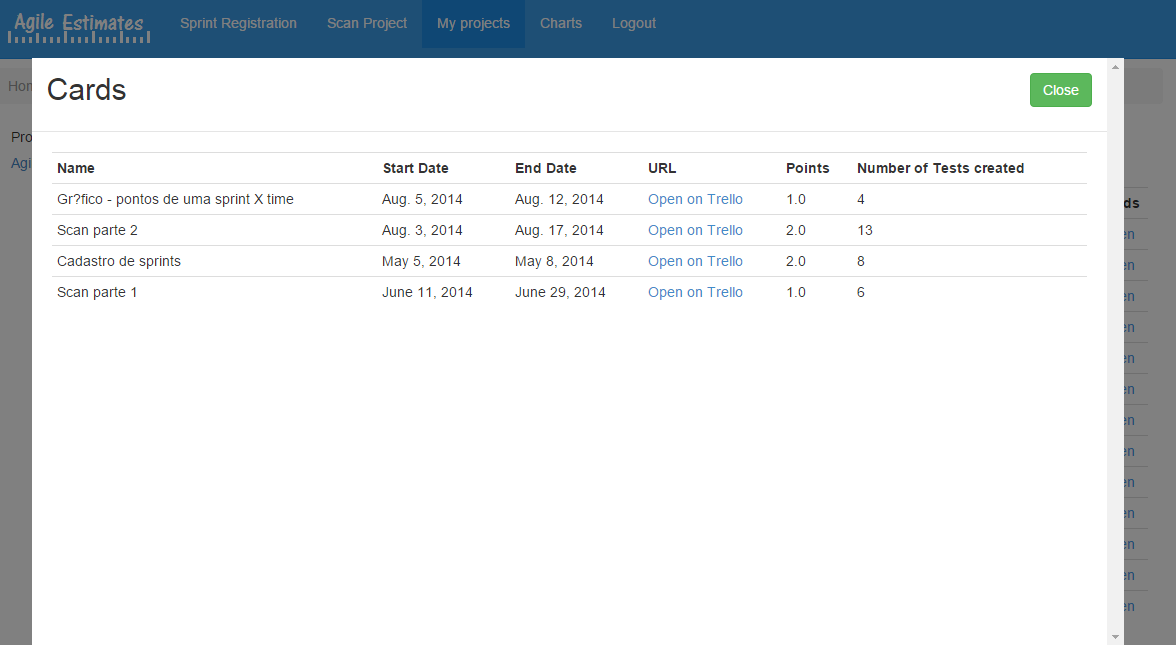
O usuário membro do time é capaz de ver o histórico de *sprints* e *cards* dos projetos que participa. As Figuras 28 e 29 mostram as tela de projetos do usuário com histórico das *sprints* e *cards*.

Figura 28 – Tela de projetos do usuário com historico das *sprints*.



Fonte: Autoria própria.

Figura 29 – Tela de projetos do usuário com historico dos *cards* das *sprints.*



Fonte: Autoria própria.

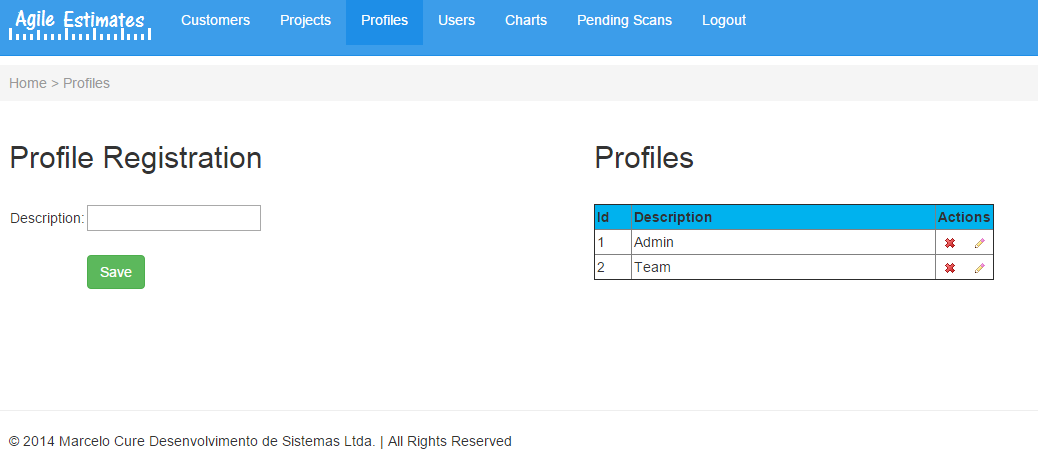
## 8.2 Administrador

O administrador possui uma visão geral de todos os projetos da organização e faz os cadastros básicos do sistema. As funcionalidades disponíveis para o administrador estão descritas nos itens abaixo.

### 8.2.1 Cadastro de perfis

Na tela de cadastro de perfis o administrador pode cadastrar os perfis de acesso do sistema. O sistema possui os perfis de administrador e membro do time pré-cadastrados. A Figura 30 mostra a tela de cadastro de perfis.

Figura 30 – Tela de cadastro de perfis.



Fonte: Autoria própria.

### 8.2.2 Cadastro de usuários

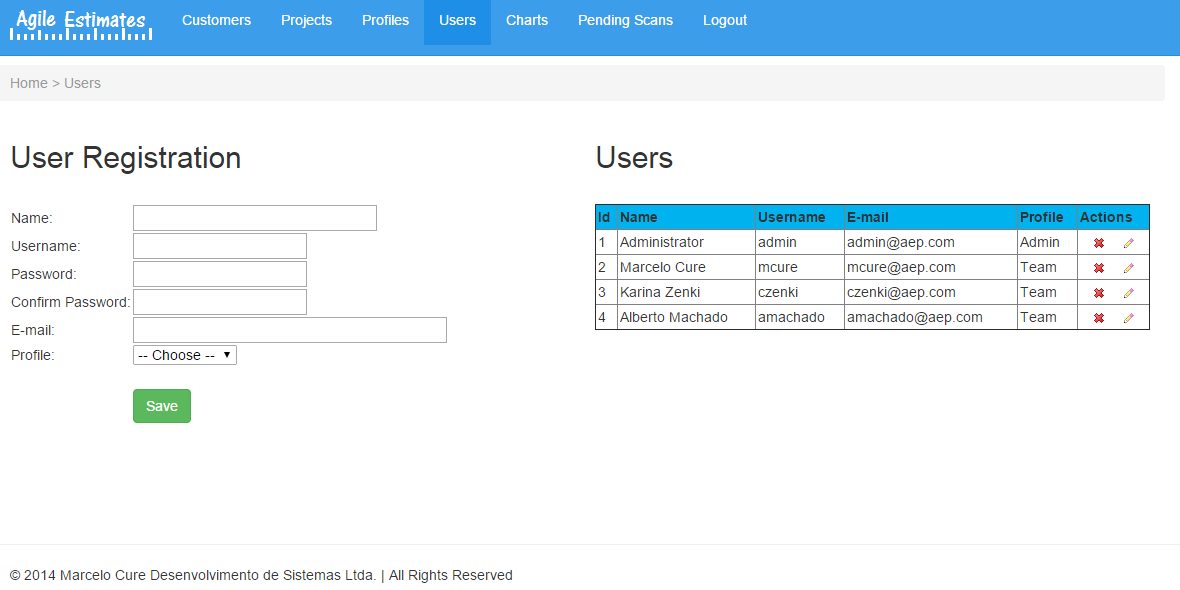
Nesta tela o administrador pode cadastrar os usuários do sistema, onde são salvos os dados básicos de um usuário, estes estão descritos abaixo:

1. Nome;
2. Username;
3. Password, onde incluirá a confirmação do mesmo;
4. E-mail;
5. Perfil de acesso.

As senhas são criptografadas e salvas no banco de dados utilizando a biblioteca PyCrypto (pypi.python.org/pypi/pycrypto/2.6.1, 2014).

A Figura 31 mostra a tela de cadastro de usuários.

Figura 31 – Tela de cadastro de usuários.



Fonte: Autoria própria.

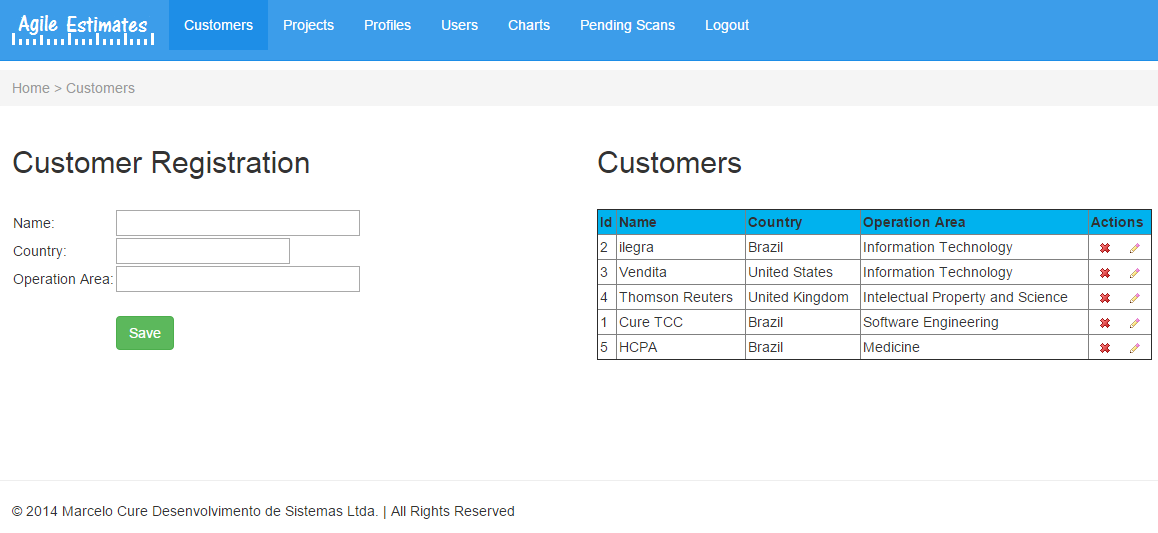
### 8.2.3 Cadastro de clientes

Nesta tela o administrador é capaz de cadastrar os clientes da organização. Os dados que são salvos estão descritos abaixo:

1. Nome;
2. País;
3. Área de operação.

A Figura 32 mostra a tela de cadastro de clientes.

Figura 32 – Tela de cadastro de clientes.



Fonte: Autoria própria.

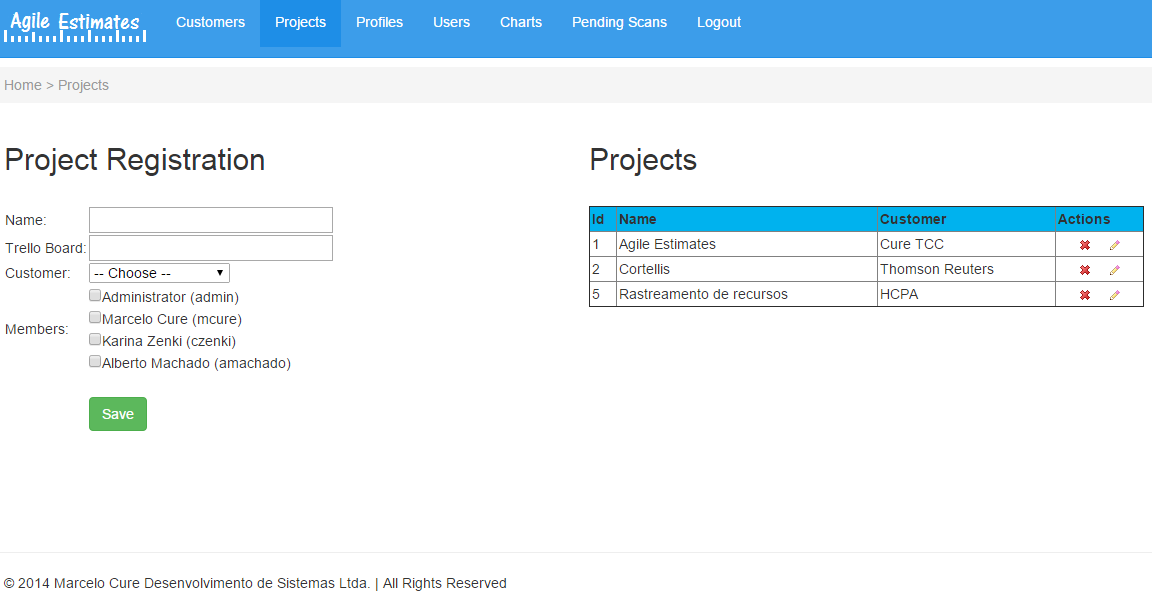
### 8.2.4 Cadastro de projetos

Nesta tela o administrador é capaz de cadastrar os projetos de um cliente e alocar recursos para o projeto. Os dados que são salvos estão descritos abaixo:

1. Descrição;
2. Cliente;
3. Membros do time alocados no projeto.

A figura 33 mostra a tela de cadastro de projetos.

Figura 33 – Tela de cadastro de projetos.



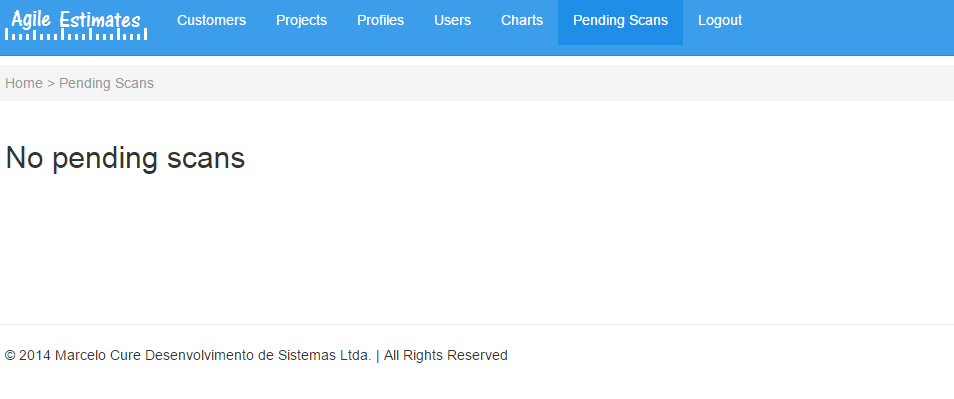
Fonte: Autoria própria.

### 8.2.5 Consulta de *scans* pendentes

Nesta tela o administrador pode consultar os projetos que possuem *sprints* sendo finalizadas no dia corrente e que o time ainda não executou o processo de *scan*.

A figura 34 mostra a tela de consulta de *scans* pendentes.

Figura 34 – Tela de consulta de *scans* pendentes.



Fonte: Autoria própria.

### 8.2.6 Gráficos

Nesta tela o administrador pode visualizar os gráficos referentes a um projeto específico ou a todos os projetos da organização.

1. Quantidade de pontos entregues por *sprint* X time;
2. Quantidade de testes unitários criados por *sprint* X time;
3. Quantidade de pontos estimados X quantidade de pontos entregues para cada *sprint*;
4. Quantidade de pontos entregues X testes unitários para cada *sprint*;
5. Quantidade de histórias entregues X *sprint* para cada projeto.

A Figura 35 mostra a tela de gráficos do administrador.

Figura 35 – Tela de gráficos do administrador.



Fonte: Autoria própria.

# 9 VALIDAÇÃO

Este capítulo apresenta a estratégia de validação deste projeto, bem como a análise dos resultados obtidos.

## 9.1 ESTRATÉGIA

Para validar se os objetivos propostos foram atingidos,   
o sistema foi implantado na empresa de desenvolvimento de *software* ilegra (ilegra.com, 2014), que utiliza o software Trello para controle do andamento das atividades. O sistema foi utilizado em um projeto e alimentado durante três *sprints*, sendo que cada *sprint* tem a duração de uma semana. Utilizaram o sistema o líder e os membros do time. Durante a quarta *sprint* foi aplicado um questionário para todos os envolvidos, para verificar se os objetivos foram atingidos.

A validação foi baseada no modelo de qualidade do produto de *software* da norma ISO/IEC 25000, também chamada SQuaRE (ISO Org., 2014). Os objetivos do projeto foram mapeados em atributos de qualidade interna, externa e qualidade do uso e foram criadas métricas para verificar a sua conformidade.

Ao final da terceira *sprint,* quando o sistema teve dados históricos suficientes para prover informações de apoio às estimativas, os membros do time receberam um questionário baseado na escala de Likert e nos seguintes atributos de qualidade:

1. Funcionalidade: Evidencia a capacidade do *software* de atender as necessidades do usuário e ter conformidade com os requisitos do sistema.
2. Confiabilidade: Evidencia a capacidade do *software* de se manter funcional com capacidade de tratar erros.
3. Usabilidade: Evidencia a capacidade do *software* de ser compreendido, atraente para o usuário e ser intuitivo.
4. Eficiência: Evidencia se o tempo de execução das tarefas do sistema são aceitaveis e compatíveis com o cenário do sistema.

descrever o que é qualidade em uso, qualidade interna e externa de modo geral, e descrever as caracteristicas usadas.... efetividade funcionalidade blablabla

A Tabela 8 apresenta o mapeamento dos objetivos com o modelo de qualidade da ISO.

Tabela 8 - Objetivos mapeados com o modelo de qualidade da ISO.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Característica | Sub-característica | Atributo/objetivo | Métrica |
| Qualidade  em uso | Efetividade | NA | Melhorar as estimativas de  *sprints*, onde as histórias  são baseadas no *planning*  *poker,* a partir de dados  históricos referentes a  estimativas de *sprints* anteriores. | MEFET |
| Prover confiança para o time de desenvolvimento e para o cliente ao planejar a sprint, baseando-se em um histórico de estimativas e resultados obtidos |
| Prover condições para melhorar a estimativa da sprint |
| Qualidade Externa | Funcionalidade | Adequabilidade | Integrar o sistema com o software Trello (trello.com, 2014); tirar um print do log do scan mostrando que conectou | MFUNC |
| Coletar informações pertinentes dos boards do Trello; tirar um print do log do scan mostrando que coletou os dados |
| Gerar gráficos para auxiliar nas estimativas futuras; |
| Qualidade Interna | Manutenibilidade | Testabilidade | Prover, através de  quantidade de testes unitários,  uma idéia geral da  qualidade do código. | CTES |
|  |  |  |  |  |

MEFET

Métrica que verifica se as estimativas melhoraram a partir do uso do software. O resultado da métrica foi obtido pelas respostas às seguintes afirmações.

1. Após o uso do sistema, a quantidade de pontos entregues foi mais próxima da estimativa da *sprint*.

2. O formato de apresentação dos dados ajudou o time a analisar o histórico de *sprints* de forma a identificar melhorias nas estimativas.

3. Após o uso do sistema, foi constatado que a quantidade de histórias do usuário não entregues ao final da *sprint* diminuiu comparado às *sprints* em que a equipe não utilizou o sistema.

4.

5.

Respostas:

1. Concordo totalmente
2. Concordo
3. Nem Concordo nem discordo
4. Discordo
5. Discordo totalmente

CFUNC

lalala

posso tentar conectar 10vezes e ver quantas conexoes foram com sucesso

o mesmo pra coletar

tentar gerar ografico varias vezes e ver se funcionou

CTES %coverage>60%

Métrica que mostra, através da cobertura de testes do sistema, uma idéia da qualidade do código. O sistema deve ter mais do que 60% de cobertura de testes.

O objetivo "Prover, através de quantidade de testes unitários, uma idéia geral da qualidade do código", foi validado através da cobertura de testes do sistema descrito na Figura 24, mostrando que 90% do código está coberto por testes.

achar uma fonte pra dizer que 60 é o normal

só explicar o que é a metrica metrica

## 9.2 CONSOLIDAÇÃO DOS RESULTADOS OBTIDOS ... ver a norma

colcoar aqui como aconteceu a validação ... resultados, conforme descrito acima

exemplo, prints, resultado do questionario, etc

a partir do resultado destas coisasblablabla com as metricas blablabla

# 10 CONCLUSÕES

Projetos executados utilizando metodologias ágeis possuem entregas frequentes de funcionalidades chamadas de iterações ou *sprints*. Cada iteração do projeto possui tarefas que devem estimadas e entregues ao final do ciclo. Porém, se as estimativas são erradas, estas tarefas podem atrasar e não serem entregues própriamente ao final da iteração, resultando em atrasos e custo extra.

Este trabalho tem por objetivo melhorar as estimativas das *sprints* de projetos que utilizam a metodologia ágil Scrum e kanban virtual na ferramenta online Trello. A solução baseia-se em projetos executados utilizando o Scrum como metodologia de desenvolvimento de software onde o projeto possua um kanban virtual utilizando a ferramenta online Trello para fazer acompanhamento das *sprints*. Com informações da *sprint* previamente cadastradas como o período de duração e sua estimativa em pontos, o sistema executa um processo de scan onde ocorre uma leitura do Trello buscando as tarefas que foram concluídas na *sprint* e salva estas informações no banco de dados. Com estes dados históricos guardados, o sistema é capaz de gerar gráficos mostrando informações relevantes de sprints passadas, servindo de apoio para a equipe estimar as histórias de usuário para planejar uma nova *sprint*. REVER PQ TEM MUITA COISA ESCRITA

A validação deste projeto aconteceu implantando o sistema na empresa de desenvolvimento de software ilegra, onde os projetos utilizam metodologias ágeis e kanban virtual do Trello para fazer o acompanhamento das *sprints*. O sistema foi utilizado por três sprints. Através dos gráficos gerados foi constatado que a estimativa das *sprints* melhorou apesar de não ser precisa. Foi constatado que 90% do código fonte do projeto está coberto por testes, o que comprova a qualidade do código do sistema.

Falar sobre os resultados obtidos e por que eles foram atingidos ... questionarios blablabla

Similar à Introdução, mas apresentando do ponto de vista do final do projeto.

Falar sobre o negócio.

Falar sobre o problema.

Falar sobre o objetivo/solução.

Falar sobre a validação e os resultados obtidos.

Falar sobre trabalhos futuros, se for o caso. nao quero falar disto

Em geral, um parágrafo para cada item acima.

# REFERÊNCIAS BIBLIOGRÁFICAS

Anderson, David J. Kanban: Successful Evolutionary Change for Your Technology Business, Blue Hole Press, 2010.

BECK, Kent. **Test-Driven Development By Example**. Three Rivers Institute, 2002.

Cacoo. Página oficial da ferramenta Cacoo. Disponível em <<https://cacoo.com/>>. Acesso em: 25 mar. 2014.

ChartJS. Página oficial da biblioteca ChartJS. Disponível em <http://chartjs.org/docs/>. Acesso em: 12 ago. 2014.

Django. Página oficial do *framework* *web* Django. Disponível em <<https://www.djangoproject.com/>>. Acesso em: 26 mar. 2014.

Eclipse Org. Disponível em: < http://epf.eclipse.org/wikis/scrumpt/Scrum/guidances/supportingmaterials/scrum\_overview\_610E45C2.html/>. Acesso em: 20 abr. 2014.

FONSECA, Vinicius Prado da, BRAGA Felipe Mendes. **Django, Desenvolvimento Ágil para a Web**, Palmas, 2009.

Github. Página oficial da ferramenta Github. Disponível em <[http://www.github.com](http://www.github.com/)/>. Acesso em: 21 mar. 2014.

ilegra. Página oficial da empresa ilegra. Disponível em <[http://www.ilegra.com](http://www.ilegra.com/)/>. Acesso em: 26 mar. 2014.

ISO Org. Disponível em: <<http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=64764>/>. Acesso em 01 jun. 2014.

jQuery. Página oficial da biblioteca JQuery. Disponível em: <http://jquery.com/>. Acesso em: 20 mar. 2014.

KEN SCHWABER. **Guia do Scrum: Um guia definitivo para o Scrum: As regras do jogo.** 2013. Disponível em: <https://www.scrum.org/Portals/0/Documents/Scrum Guides/2013/Scrum-Guide-Portuguese-BR.pdf>. Acesso em: 27 mar. 2014. **(POSTAGEM RETIRADA DE SITE)**

LOTT, Steve: Why Python is So Cool. **It May Be a Hack**. 2009. Disponível em: <http://www.itmaybeahack.com/homepage/books/nonprog/html/p01\_getting\_started/p01\_c05\_cool.html>. Acesso em: 22 out. 2014. **(POSTAGEM RETIRADA DE SITE)**

METHODOLOGY, scrum: **Scrum Effort Estimation and Story Points**. 2009. Disponível em: <<http://scrummethodology.com/scrum-effort-estimation-and-story-points/>>. Acesso em: 18 mar. 2014. **(POSTAGEM RETIRADA DE SITE)**

METHODOLOGY, scrum: **Scrum User Stories**. 2009. Disponível em: < [http://scrummethodology.com/scrum-user-stories/](%20http://scrummethodology.com/scrum-user-stories/)>. Acesso em: 19 mar. 2014. **(POSTAGEM RETIRADA DE SITE)**

Molokken-Ostvold, K., Haugen, N.C.Combining Estimates with Planning Poker--An Empirical Study. In: IEEE Proceedings of the 2007 Australian Software Engineering Conference, 2007 (ASWEC'07).

NinjaMock. Página oficial da ferramenta NinjaMock. Disponível em <<http://ninjamock.com/>>. Acesso em: 28 mar. 2014.

PLANBOX (Montreal) (Ed.). 2013. **Agile by the numbers 2013: Planbox recently sponsored a global survey by Actuation Consulting that looked into why some teams excel while others struggle**. Disponível em: <https://www.planbox.com/agile-by-the-numbers-2013-performance-study/>. Acesso em: 26 mar. 2014. **(POSTAGEM RETIRADA DE SITE)**

Postgresql. Página oficial do banco de dados Postgresql. Disponível em <<http://www.postgresql.org/>>. Acesso em: 27 mar. 2014.

PyCrypto. Página oficial da biblioteca PyCrypto. Disponível em <https://pypi.python.org/pypi/pycrypto/2.6.1>. Acesso em: 25 mai. 2014.

Python. Página oficial da linguagem de programação Python. Disponível em <<http://www.python.org.br/>>. Acesso em: 19 mar. 2014.

SCOTT W. AMBLER, 2006. **Introduction to Test Driven Development (TDD)**. Disponível em: <http://www.agiledata.org/essays/tdd.html>. Acesso em: 26 mar. 2014.

SublimeText. Página oficial do editor de texto Sublime. Disponível em <[http://www.sublimetext.com](http://www.sublimetext.com/)/>. Acesso em: 24 mar. 2014.

Trello. Página oficial do software Trello. Disponível em <[http://www.trello.com](http://www.trello.com/)/>. Acesso em: 17 mar. 2014.

Trollop. Página oficial da biblioteca Trollop. Disponível em <http://pypi.python.org/pypi/trollop>. Acesso em 28 jul. 2014.

Valente, F. F. R. e Falbo, R. A. (2002) “Uso de Gerência de Conhecimento para Apoiar a Realização de Estimativas”, In: Proceedings of the XXVIII Latin-American Conference on Informatics - CLEI'2002, Montevideo, Uruguay, November 2002.

WAGNER LINDBERG BACCARIN ARNAUT. **O Fim do Empirismo no Desenvolvimento de Software.** 2008. Disponível em: <https://www.ibm.com/developerworks/community/blogs/tlcbr/entry/fim\_do\_empirismo?lang=en>. Acesso em: 29 mar. 2014. **(POSTAGEM RETIRADA DE SITE)**

# COMPONENTES REUTILIZADOS

***Django so colocar o link poara apagina do django e o q ta escrito abaixo tem que ir pra descrição das tecnologias e ferramentas***

Será utilizado o *framework* *Django* para auxiliar no desenvolvimento das páginas e integração com banco de dados, onde o desenvolvedor pode definir o modelo de dados utilizando a linguagem *Python*. Este *framework* também provém uma *API* poderosa de integração com o banco de dados. *Django* é um framework para desenvolvimento web com *Python* e serve, principalmente, para agilizar o desenvolvimento. (djangoproject.com, 2014)

***API do Trello***

O software *Trello* oferece uma *API* chamada Trollop (pypi.python.org/pypi/trollop, 2014) para acesso dos *boards*, onde possibilita a coleta de informações onde o usuário possui acesso. Através de uma chave de segurança gerado pelo site, o desenvolvedor utiliza esta chave de segurança para autenticação e coleta de informações. (trello.com/docs, 2014)

***PyCrypto***

PyCrypto é uma biblioteca escrita em Python utilizada para criptografia de dados. Esta biblioteca está sendo utilizada para criptografar as senhas dos usuários do sistema. (pypi.python.org/pypi/pycrypto/2.6.1, 2014)

***ChartJS***

ChartJS é uma biblioteca escrita em javascript para gerar gráficos em um ambiente web. Esta ferramenta possibilida a criação de vários tipos de gráficos. O ChartJS trabalha com os dados no formato JSON, assim fica fácil de passar os conjuntos de dados e outras configurações, como cor de fundo, cor das linhas, etc. (chartjs.org/docs/, 2014)