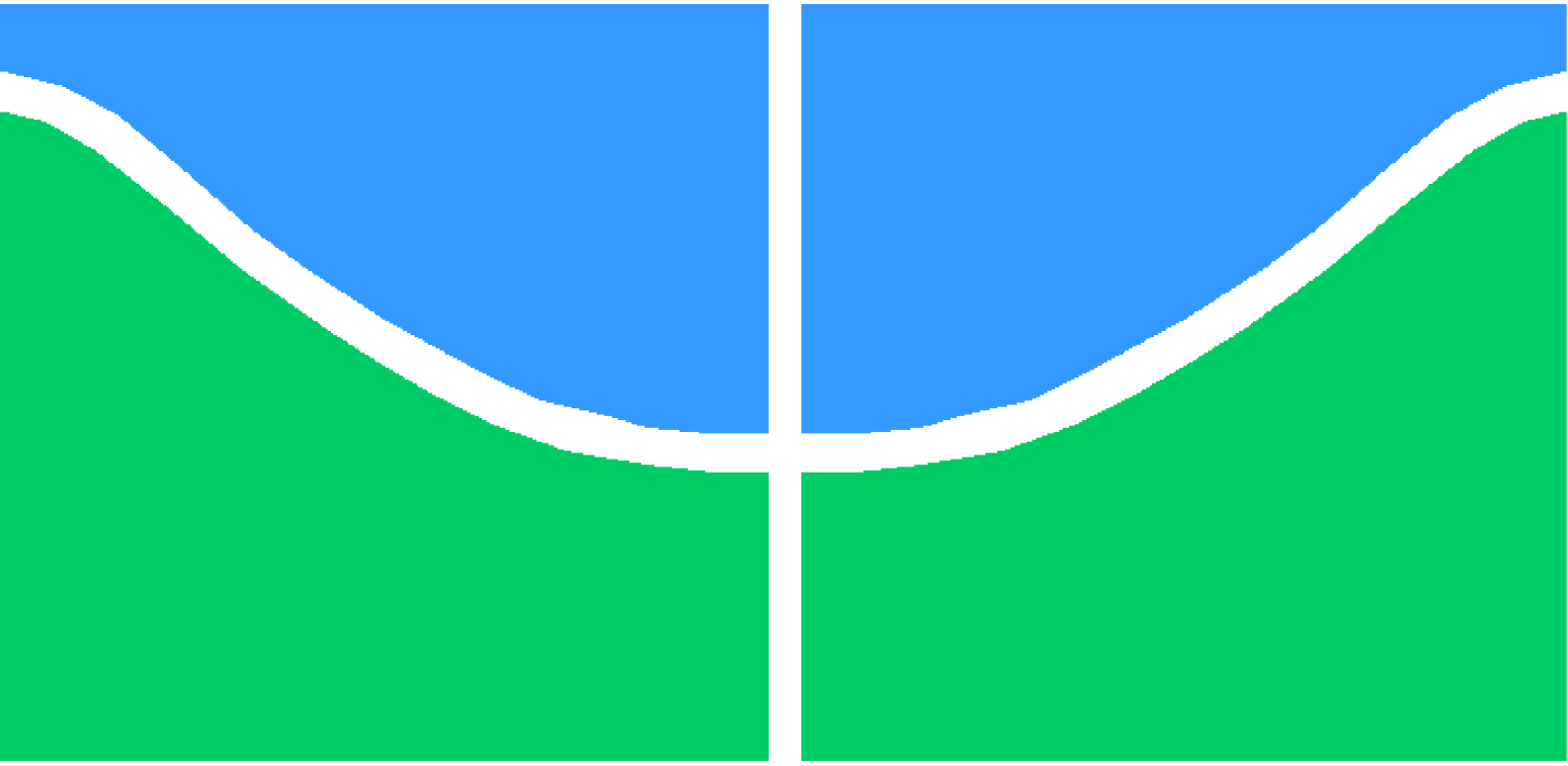

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de *Software*

Sistema de Monitoramento de Insumos - Universidade de Brasília (SMI-UnB)

Autor: Brenddon Gontijo Furtado
Orientador: Prof. Dr. Fábio Macedo Mendes

Brasília, DF
2017

Brenddon Gontijo Furtado

Sistema de Monitoramento de Insumos - Universidade de Brasília (SMI-UnB)

Monografia submetida ao curso de graduação em (Engenharia de *Software*) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de *Software*).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Dr. Fábio Macedo Mendes

Coorientador: Prof. Dr. Paulo Roberto Miranda Meirelles

Brasília, DF

2017

Brenddon Gontijo Furtado

Sistema de Monitoramento de Insumos - Universidade de Brasília (SMI-UnB)/ Brenddon Gontijo Furtado. – Brasília, DF, 2017-

91 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Fábio Macedo Mendes

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2017.

1. Engenharia de *Software*. 2. Gerência de Configuração de *Software*. 3. Desenvolvimento Ágil de *Software*. 4. *Software* Livre. 5. Monitoramento Energético. I. Prof. Dr. Fábio Macedo Mendes. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Sistema de Monitoramento de Insumos - Universidade de Brasília (SMI-UnB)

CDU 02:141:005.6

Brenddon Gontijo Furtado

Sistema de Monitoramento de Insumos - Universidade de Brasília (SMI-UnB)

Monografia submetida ao curso de graduação em (Engenharia de *Software*) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de *Software*).

Trabalho aprovado. Brasília, DF, 12 de julho de 2017:

Prof. Dr. Fábio Macedo Mendes
Orientador

Prof. Dr. Carla Silva Rocha Aguiar
Convidado 1

Prof. Dr. Tiago Alves da Fonseca
Convidado 2

Brasília, DF
2017

Resumo

O consumo energético vem crescendo com o passar dos anos, o que implica na necessidade de criação de novas infraestruturas com diversos impactos sociais, ambientais e econômicos. O uso racional de energia é capaz de diminuir esses impactos, proporcionando um desenvolvimento mais sustentável. Tendo em vista essas questões, a Universidade de Brasília criou uma iniciativa de monitoramento energético para seus *campi* com a idealização de um sistema para tais fins. O objetivo deste trabalho é desenvolver esse sistema de monitoramento e, a partir desse ponto, fomentar políticas de uso mais racional de energia dentro da Universidade.

O desenvolvimento do sistema será baseado nos requisitos definidos junto à Prefeitura de Campus e utilizará os conhecimentos e métodos da Engenharia de *Software*, aplicando-os no ciclo de vida do sistema. O presente trabalho realiza a coleta e apresentação dos dados energéticos, mas pode ser estendido para outros tipos de insumos. Os conceitos utilizados abordam práticas de desenvolvimento ágeis, *software* livre, gerência de configuração de *software* e tecnologias *web*. Este trabalho apresenta como foi o ciclo de vida do sistema e detalhadamente todas as suas características importantes.

Palavras-chaves: Engenharia de *Software*. Gerência de Configuração de *Software*. Desenvolvimento Ágil de *Software*. *Software* Livre. Monitoramento Energético.

Abstract

Energy consumption has been growing over the years, which implies the need to create new infrastructures with diverse social, environmental and economic impacts. The rational use of energy is able to reduce these impacts, providing a more sustainable development. In view of these issues, the University of Brasilia has created an energy monitoring initiative for its campus with the design of a system for such purposes. The objective of this work is to develop this monitoring system and, from that point on, to promote more rational energy use policies within the University.

The development of the system will be based on the requirements established with Campus City Hall and will use the knowledge and methods of *Software* Engineering, applying them in the system life cycle. The present work performs the collection and presentation of energy data, but can be extended to other types of inputs. The concepts used address agile development practices, free *software*, *software* configuration management, and web technologies. This paper presents how the system life cycle and all its important characteristics were detailed.

Key-words: *Software* Engineering. *Software* Configuration Management. Agile Methods. Free *Software*. Energy Monitoring.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	*Application Programming Interface*
GPP	Gestão de Projetos e Portfólio de *Software*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
MDS	Métodos de Desenvolvimento de *Software*
MVC	*Model View Controller*
MTV	*Model Template View*
UnB	Universidade de Brasília
SMI-UnB	Sistema de Monitoramento de Insumos - Universidade de Brasília
UDP	*User Datagram Protocol*
XP	Extreme Programming

Sumário

1 Introdução

O consumo de energia elétrica vem aumentando com o passar dos anos. No ano de 2014, o Brasil consumiu 531.1 TWh, o que corresponde a um aumento de 2.9% comparado a 2013. Cerca de 2.4% deste total é consumido por prédios públicos (Ministério de Minas e Energia, 2015). No ano de 2015, foi realizado um reajuste tarifário da energia elétrica correspondente a 33% para residências e 32,5% para empresas, indústrias e comércios (AES Eletropaulo, 2015). Tendo em vista esse cenário, é interessante a administração pública viabilizar políticas para o uso mais racional da energia elétrica.

Em Maio de 2016, a Prefeitura de Campus da Universidade de Brasília (UnB) aprovou um projeto de monitoramento energético da própria Universidade, objetivando que cada campus da instituição (Asa Norte, Ceilândia, Gama e Planaltina) seja responsável por realizar seu próprio monitoramento de energia e enviar essas informações para a administração central.

O presente trabalho consiste no desenvolvimento de um sistema web capaz de unificar, de forma setorial, os insumos utilizados pelos *campi* da UnB, com o intuito de reduzir gastos e promover a sustentabilidade. Inicialmente foi realizado a coleta de grandezas energéticas, porém, o projeto possui a possibilidade de extensão para outros insumos. Os monitoramentos realizados foram efetivados por meio de equipamentos eletrônicos denominados transdutores, que foram instalados nos quadros de energia para transmitir suas informações para o sistema através da rede de um campus.

A apresentação das grandezas energéticas monitoradas por um transdutor foi realizada por meio de um gráfico de linhas, tornando possível que usuários do sistema consigam visualizar de maneira simples e fácil as medições de um período de tempo específico.

O desenvolvimento do sistema teve como base fundamentos e metodologias presentes na Engenharia de *Software*, com o objetivo de se obter um *software* confiável e eficiente de maneira econômica e sustentável.

Procedimentos denominados métodos ágeis guiaram o ciclo de vida do sistema de maneira iterativa e incremental, procurando trazer *software* funcional e com qualidade em curtos períodos de tempo. Atrelado a essa abordagem, adotou-se boas práticas de desenvolvimento, referentes ao Kanban e *Extreme Programming*. Essas práticas se adequavam ao contexto vigente e auxiliaram na entrega de um produto que atendeu as expectativas do cliente.

Foi necessário utilizar os conhecimentos da Gerência de Configuração de *Software* para que o sistema respondesse às volatilidades presentes no decorrer do desenvolvimento

de *software*, garantindo, assim, sua manutenibilidade, confiabilidade e qualidade.

Algumas filosofias do processo de desenvolvimento de *software* Devops foram utilizadas no decorrer do projeto, buscando facilitar sua automatização e distribuição.

1.1 Objetivo Geral

Desenvolver parte inicial de um sistema *web* capaz de monitorar temporalmente insumos da Universidade de Brasília, com auxílio de conhecimentos e métodos da Engenharia de *Software*.

1.1.1 Objetivos Específicos

- Coleta, armazenamento e análise de informações energéticas;
- API de comunicação com sistemas externos;
- Realizar protótipo para comunicação entre uma administração central e os *campi* da UnB.

1.2 Metodologia Utilizada

A metodologia abordada neste trabalho teve como base o desenvolvimento de *software* livre, utilizando-se os princípios empíricos de desenvolvimento ágil de *software* juntamente com o Kanban e algumas práticas do *Extreme Programming*.

1.3 Estruturação do Trabalho

O trabalho se encontra estruturado da seguinte maneira:

- Capítulo 2: métodos empíricos da Engenharia de *Software* utilizados para auxiliar no desenvolvimento do sistema.
- Capítulo 3: ciclo de vida.
- Capítulo 4: características e tecnologias adotadas no sistema.
- Capitulo 5: conclusão e trabalhos futuros.

2 Métodos Empíricos em Engenharia de *Software*

2.1 Métodos Ágeis

Os negócios atualmente operam em um ambiente global sujeito a rápidas mudanças, sendo crucial estar preparado para novas oportunidades de mercado, mudanças de condições econômicas e surgimento de produtos e serviços concorrentes. O *software* é um dos componentes cruciais para a realização de várias operações de negócio e realizá-lo de forma rápida também é uma maneira de aproveitar novas oportunidades e responder às pressões competitivas (SOMMERVILLE, 2006).

O manifesto ágil (BECK et al., 2001) consiste na base que fundamenta o desenvolvimento ágil de *software* sendo composto de quatro valores fundamentais:

- **Indivíduos e interações** 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

"Enquanto há valor nos itens à direita, valorizamos mais os itens à esquerda" (BECK et al., 2001).

Além desses valores são definidos 12 princípios de agilidade:

- Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de *software* de valor;
- Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam às mudanças, para que o cliente possa tirar vantagens competitivas;
- Entregar *software* funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos;
- Pessoas relacionadas a negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto;
- Construir projetos ao redor de indivíduos motivados, dando a eles o ambiente e suporte necessário e confiar que farão seu trabalho;

- 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;
- *Software* funcional é a medida primária de progresso;
- Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter, indefinidamente, passos constantes;
- A contínua atenção à excelência técnica e o bom *design* aumentam a agilidade;
- Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito;
- As melhores arquiteturas, requisitos e *designs* emergem de times auto-organizáveis;
- Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajusta e otimiza seu comportamento de acordo.

Nem todos os processos ágeis aplicam tais princípios de maneira igualitária e alguns modelos escolhem ignorar (ou ao menos minimizar) a importância de um ou mais princípios (PRESSMAN, 2010).

Processos de desenvolvimento ágeis geralmente são iterativos onde a especificação, projeto, desenvolvimento e teste são intercalados. O *software* é desenvolvido em uma série de incrementos e cada incremento fornece uma nova funcionalidade ao sistema (SOMMERVILLE, 2006). As duas principais vantagens de se adotar uma abordagem incremental para o desenvolvimento de *software* são:

- Entrega acelerada dos serviços ao cliente. Os clientes poderão obter valor do sistema já nos incrementos iniciais;
- Engajamento do usuário com o sistema. Os usuários do sistema devem estar envolvidos no processo de desenvolvimento dando *feedback* à equipe de desenvolvimento sobre os incrementos entregues.

2.1.1 *Extreme Programming*

Extreme Programming (XP) é uma filosofia de desenvolvimento de *software* baseada nos valores de comunicação, simplicidade, *feedback* e coragem. É designado para se trabalhar com projetos que podem ser construídos por equipes de dois a dez programadores, os quais não devem possuir limitações por causa do ambiente computacional e devem ser capazes de realizar testes de *software* em uma fração de um dia (BECK; ANDRES, 2004).

No XP, são definidas 12 importantes práticas que auxiliam no desenvolvimento de *software* e em suas *releases*. Uma *release* consiste no lançamento, parcial ou não, de um *software*, sendo que em muitas das vezes são lançadas versões beta[1] dele, proporcionando, assim, uma possível fase de *debugging* ou *feedback*, provinda do cliente ou até mesmo da equipe de desenvolvimento. As práticas do XP, por sua vez, alinham-se bem no decorrer do ciclo de desenvolvimento de *software* e são includentes, ou seja, onde uma possui determinada desvantagem, outra irá compensar com seus pontos positivos (BECK; ANDRES, 2004). As práticas são:

- Cliente Presente: clientes devem sempre estar presentes para auxiliarem a equipe com seus *feedbacks*;
- *Design* Simples: o sistema deve sempre ser desenhado de forma mais simples possível, sendo que complexidades encontradas devem ser removidas rapidamente.
- Integração Contínua: integração e construção do sistema muitas vezes ao dia, sempre que uma tarefa for concluída;
- Jogo do Planejamento (do inglês *Planning Game*): determina de maneira rápida o escopo de uma *release* combinando prioridades de negócio e estimativas técnicas.
- Metáfora (do inglês *Metaphor*): linguagem comum que os membros devem possuir para explicar facilmente como o sistema funciona como um todo;
- Padrões de Codificação: escrita do código de acordo com boas práticas de programação, enfatizando uma comunicação através dele;
- Programação em Pares: realizar a codificação do sistema sempre com dois programadores que compartilhem a mesma máquina;
- Propriedade Coletiva: qualquer pessoa pode modificar, desde que esteja realizando de maneira correta, qualquer parte do código a qualquer momento;
- Refatoração: reestruturação do sistema sem alteração de seu comportamento, objetivando remover duplicidade, simplificação de código, aumento de flexibilidade e melhoramento da comunicação;
- Semana de 40 Horas: o trabalho semanal realizado não pode ultrapassar o limite de 40 horas;
- Teste: são escritos testes unitários, que devem funcionar de maneira adequada para que o desenvolvimento do sistema continue;

[1] Versão de um *software* que ainda se encontra em desenvolvimento. Essa versão é disponibilizada para que os usuários realizem testes e reportem qualquer tipo de problema encontrado aos desenvolvedores

- Versões Pequenas: colocar rapidamente um sistema simples e funcional em produção, para que posteriormente sejam lançadas novas versões em um curto período de tempo.

2.1.2 Kanban

O Kanban (RADIGAN, 2015) é um *framework* que visa auxiliar equipes a organizarem de maneira mais prática suas tarefas. Utiliza um quadro de produção, Figura 1, que irá conter todo o fluxo de trabalho, possuindo três principais colunas:

- *To Do*: tarefas que serão realizadas em breve;
- *In Progress*: tarefas que atualmente estão sendo realizadas pela equipe;
- *Done*: tarefas que foram finalizadas.

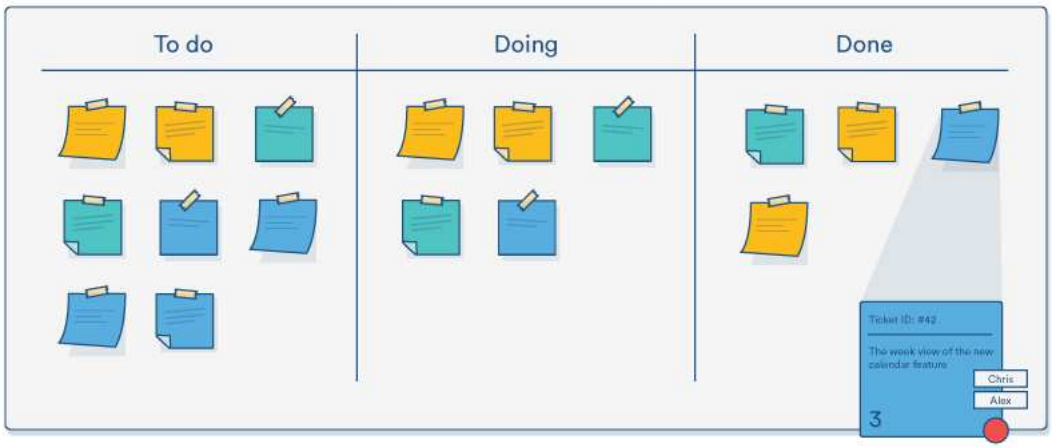

Figura 1 – Quadro Kanban de produção. Fonte: (RADIGAN, 2015)

Cada vez que uma tarefa passar por mudanças que alterem seu estado, o quadro deve ser atualizado, trazendo mais transparência para a equipe. Além disso, tarefas que são planejadas e não se encaixam no contexto vigente do quadro são armazenadas no *Backlog*.

2.2 Requisitos de *Software*

Entender os requisitos de um problema é uma das difíceis missões que são encaradas na Engenharia de *Software*. Em muitos casos, o cliente não sabe o que realmente quer ou não consegue expressar em termos inteligíveis para o Engenheiro de *Software*. É comum, também, que as necessidades do cliente mudem com o decorrer do projeto. Sommerville (SOMMERVILLE, 2006) define os requisitos de um projeto como “requisitos do

usuário" e "requisitos do sistema". Os requisitos de usuários consistem de declarações, em linguagem natural, que o sistema deve fornecer e restrições que deve operar. Os requisitos de sistema estabelecem de maneira detalhada as funções e restrições do sistema. Além disso, os requisitos de sistema classificam-se em funcionais e não funcionais:

- Requisitos Funcionais: declaração de funções que o sistema deve fornecer, como irá reagir com certas entradas e como deve se comportar para certas situaçoes;
- Requisitos Não Funcionais: restrições sobre serviços ou funções oferecidas pelo sistema.

O levantamento adequado de requisitos de *software* é crucial para que possa ser feito um mapeamento das reais necessidades do cliente com funcionalidades que um sistema deve atender. Thayer (THAYER; BAILIN; DORFMAN, 1997) define a Engenharia de Requisitos como um processo que provê os mecanismos apropriados para o entendimento do que o cliente quer, analisando a necessidade, avaliando a viabilidade, negociando uma solução inteligente, especificando a solução de maneira não ambígua, validando a especificação e gerenciando os requisitos conforme são transformados em um sistema operacional.

Dentre as atividades da Engenharia de Requisitos, encontra-se a elicitação de requisitos, a qual, para Leite (LEITE; SANT'ANNA; FREITAS, 1994), é reponsável por identificar todos os fatos que irão fazer parte dos requisitos do sistema, buscando fornecer todo o entendimento que o *software* deve possuir.

Com os requisitos do sistema em mãos, é possível identificar os seus casos de usos, que são narrativas ou modelos de texto que descrevem uma função ou recurso do sistema do ponto de vista do usuário (PRESSMAN, 2010). Em um contexto ágil, como por exemplo o XP, o caso de uso é mapeado na estória de usuário. Estórias de usuário são escritas pelos clientes como operações que o sistema precisa fazer por eles. Servem para propor estimativas a uma *release* e seus tempos de realização devem ser curtos, entre uma e três semanas de desenvolvimento (BECK; ANDRES, 2004).

2.3 Teste de *Software*

Teste de *software* é o processo de execução de um produto para averiguar se ele atingiu suas especificações e funciona corretamente em seu ambiente alvo (NETO; DIAS, 2007).

De acordo com Luo (2001), um bom teste é o que possui uma alta probabilidade de encontrar um erro ainda não descoberto e um teste bem sucedido é o que de fato descobre erros desconhecidos.

Esses testes são estruturados em níveis, cada um com um determinado objetivo dentro do conjunto de testes, de modo a garantir a qualidade do produto em desenvolvimento (LUO, 2001).

2.3.0.1 Testes Unitários

Testes unitários possuem como objetivo verificar a existência de defeitos em cada módulo do projeto. Seu alvo são os métodos desenvolvidos ou pequenos trechos específicos de código (NETO; DIAS, 2007).

É realizado durante o desenvolvimento, pelo próprio desenvolvedor, pois testa a unidade básica de *software*, que é o menor "pedaço" testável, por sua vez chamado de unidade, dando origem ao nome deste tipo de teste (LUO, 2001).

Um exemplo de objetivo do teste unitário é a procura pela identificação de erros de lógica e de implementação (MALDONADO et al., 2004).

2.4 Gerência de Configuração de *Software*

Um sistema pode ser definido como a combinação de elementos que interagem entre si e estão organizados para alcançar um ou mais objetivos previamente declarados, onde suas características físicas e funcionais de *hardware* ou *software* representam sua configuração (BOURQUE; FAIRLEY, 2014).

Em uma definição mais formal, a ISO 24765 (ISO/IEC/IEEE, 2010) define Gerência de Configuração como uma disciplina responsável por: "identificar e documentar as características funcionais e físicas de um item de configuração, controlar as alterações dessas características, registar e reportar o processamento de alterações e o *status* de implementação e verificar a conformidade com os requisitos especificados".

A Gerência de Configuração de *Software* (SCM, do inglês *Software Configuration Management*) é um processo que beneficia o gerenciamento de projeto, assim como o seu desenvolvimento, manutenção e atividades referentes à garantia de qualidade (BOURQUE; FAIRLEY, 2014).

O CMMI-DEV (CMMI Product Team, 2010) define três objetivos para a Gerência de Configuração de *Software*:

- Estabelecimento de *Baselines*: para cada nova mudança implementada um incremento na evolução do projeto é gerado. Essas mudanças devem possuir um histórico bem definido. As ferramentas de controle de versão facilitam esse trabalho, além de possibilitarem uma programação concorrente;

- Rastreamento e Controle de Mudanças: durante o desenvolvimento de *software*, mudanças ocorrem com frequência. É necessário, portanto, que tais mudanças sejam armazenadas, analisadas e agrupadas de acordo com o histórico e suas prioridades;

- Estabelecimento de Integridade: verificar se a construção de um sistema, atendendo suas configurações pré-estabelecidas, é bem sucedida a cada nova mudança registrada.

2.5 *Software* Livre

" O *Software* Livre é aquele que permite aos usuários usá-lo, estudá-lo, modificá-lo e redistribui-lo, em geral, sem restrições para tal e prevenindo que não sejam impostas restrições aos futuros usuários " (MEIRELLES, 2013).

Em comparação ao *software* restrito, o *software* livre apresenta algumas vantagens devido ao fato de seu código-fonte estar disponível para qualquer usuário, partindo do princípio que seu licenciamento esteja de acordo com as definições da *Free Software Foundation*[2] ou da *Open Source Initiative*[3]. Essa disponibilidade auxilia no desenvolvimento de aplicações personalizadas, uma vez que que é possível partir de uma solução já existente ao invés de desenvolver tudo partindo do zero. Tal abordagem possui um impacto significativo na redução de custos e diminuição na duplicação de esforço (MEIRELLES, 2013).

Raymond (RAYMOND, 1999), observando o modelo de desenvolvimento do Linux, percebeu que o compartilhamento de código possivelmente melhoraria a qualidade final de uma aplicação, uma vez que uma quantidade grande de desenvolvedores, com diferentes habilidades e conhecimentos, conseguem propor melhorias e consertar *bugs* em uma pequena quantidade de tempo (MEIRELLES, 2013).

2.6 Usabilidade de *Software*

A usabilidade é geralmente considerada como o fator que assegura que os produtos sejam fáceis de usar, eficientes e agradáveis, sob a perspectiva do usuário, através da otimização das interações estabelecidas pelas pessoas com produtos interativos (ROGERS; SHARP; PREECE, 2013).

Segundo a ISO 9126 (ISO/IEC, 2001), o conceito de usabilidade, em um contexto de *software*, é definido como "capacidade do produto de *software* de ser entendido, aprendido, usado e atraente para o usuário, quando usado em condições especificadas". Nielsen

[2] <http://www.gnu.org/philosophy/free-sw.pt-br.html>
[3] <https://opensource.org/docs/definition.html>

(NIELSEN, 1993) propõe que a usabilidade está distribuida em diversos elementos e define alguns fatores que estão associados a ela, sendo estes:

- Eficiência: proporcionar ao usuário o cumprimento do seu objetivo de forma rápida e fácil. Para isso, uma aplicação deve ser feita de forma simples, com o mínimo de poluição visual possível, intuitiva e bem estruturada, reduzindo o esforço e a dificuldade em se realizar um determinada tarefa;
- Facilidade de Aprendizagem: capacidade do cliente aprender rapidamente a utilizar o sistema de maneira intuitiva;
- Facilidade de Memorização: o sistema deve ser capaz de ser facilmente memorizado, ou seja, por mais que um usuário fique um tempo sem utilizá-lo, irá se recordar facilmente de como se usa;
- Satisfação: a utilização do sistema deve providenciar ao usuário uma experiência agradável;
- Seguraça: refere-se a prevenção para que o usuário não cometa erros. Uma aplicação deve conter botões de desfazer, botões de edição, pedidos de confirmação de ação antes de ser realizada e dicas de como proceder em determinadas situações.

2.7 Métricas de *Software*

Desenvolver ou selecionar produtos de *software* de alta qualidade é de primordial importância. A especificação abrangente e a avaliação da qualidade do produto do *software* são um fator chave para garantir uma qualidade adequada. Isso pode ser alcançado definindo características de qualidade apropriadas, levando em consideração a finalidade do uso do produto de *software*. É importante que todas as características relevantes da qualidade do produto de *software* sejam especificadas e avaliadas, sempre que possível usando métricas validadas ou amplamente aceitas (ISO/IEC, 2001).

Dentre os inúmeros tipos de métricas existentes, algumas comumente presentes em aplicações que utilizam uma abordagem orientada a objetos, segundo Pressman(PRESSMAN, 2010), são:

- Tamanho: número de classes ou operações, linhas presentes em um método e afins;
- Complexidade: número de operações e caminhos distintos para o fluxo de código em cada método;
- Acoplamento: quão uma classe depende de outra para realizar suas funções;

- Coesão: quão uma classe realiza apenas suas responsabilidades específicas;
- Similiridade: quão duas ou mais classes são similares com suas estruturas, funções, comportamentos ou propósitos.

3 Ciclo de Vida do Sistema

3.1 Definição da Abordagem e Metodologia

Foi definida a utilização de uma abordagem de desenvolvimento ágil (BECK et al., 2001) para o contexto inicial do projeto, levando-se em conta que o processo de desenvolvimento seria realizado pelo próprio autor, a disponibilidade do cliente seria mais abrangente e o projeto evoluiria aos poucos, com entregas contínuas de pequenos incrementos de *software*.

A metodologia utilizada se baseou no Kanban (RADIGAN, 2015), com algumas práticas do *Extreme Programming*, visto que ambos atendiam de maneira adequada ao fluxo de trabalho contínuo, previsto pela abordagem escolhida.

Para auxiliar nas *releases* do projeto, foi utilizado o conceito de *sprints*, provido pelo *Scrum* (SCHWABER; SUTHERLAND, 2001). As *sprints* são pequenos intervalos de tempo, de um mês ou menos, durante o qual uma versão incremental potencialmente utilizável do produto é criada. Uma nova Sprint se inicia imediatamente após a conclusão da Sprint anterior. O projeto adotou *sprints* de 2 semanas.

Ressalta-se que os próximos tópicos do ciclo de vida não foram realizados completamente em sequência, visto que esses acabavam se intercalando em determinados momentos, como previsto em uma abordagem ágil.

3.2 Contexto e Necessidades

A Prefeitura de Campus designou dois professores doutores da Faculdade UnB Gama, Alex Reis e Loana Nunes Valesco, para darem suporte às necessidades energéticas que o sistema deveria atender. Por questões de disponibilidade e quantidade de equipamentos de medição adquiridos, o campus UnB Gama foi escolhido como ambiente de teste para a primeira parte do projeto.

Tendo como problema a falta de monitoramento energético adequado na Universidade de Brasília, foi realizado um estudo mais aprofundado sobre o contexto para identificar as necessidades do cliente. Tal estudo abordou fatores energéticos cruciais, como por exemplo, sistemas trifásicos, horários de ponta e fora de ponta, tensão, corrente, resistência e afins.

A energia elétrica em corrente alternada e em sistema trifásico é um dos métodos mais comuns para se gerar, transmitir e distribuir corrente elétrica alternada. Baseia-se

em um sistema denominado polifásico e empresas elétricas de todo o mundo o utilizam como meio de transmissão de energia (STEVENSON, 1962).

A Tarifa Branca sinaliza a variação do valor da energia conforme o dia e o horário do consumo. Nos dias úteis, o valor da Tarifa Branca varia em três horários: ponta, intermediário e fora de ponta. Na ponta e no intermediário, a energia é mais cara. Fora de ponta, é mais barata. Durante feriados nacionais e fins de semana, o valor é sempre fora de ponta (ANEEL, 1996).

Com um entendimento melhor do contexto e reuniões com os professores, as seguintes necessidades foram encontradas:

- Cada campus da Universidade de Brasília deve ser responsável por realizar seu próprio monitoramento energético;
- O campus Darcy Ribeiro deve funcionar como uma espécie de administração central, capaz de reunir os dados de todos os outros campus;
- As medições devem ser armazenadas e analisadas.

Inicialmente, o nome do projeto foi definido como Sistema de Monitoramento Energético - Universidade de Brasília (SME-UnB), porém, após algumas reuniões realizadas, percebeu-se a possiblidade de monitoramento, no futuro, de medições de água. Tendo em vista os recursos que seriam monitorados, o nome do projeto foi modificado para Sistema de Monitoramento de Insumos - Universidade de Brasília (SMI-UnB).

Foi definido que, para o contexto inicial do projeto, seriam enfatizados os dados de energia e que seriam entregues duas *releases*, de aproximadamente 80 dias, sendo estas:

- Release 1: coleta dos dados de energia;
- Release 2: apresentação dos dados e protótipo de comunicação inter-campi.

Foi escolhido o *software* livre GitLab CE[1] para realizar a hospedagem do código/documentação do projeto[2] e o controle de mudanças. Já para o controle de versão, definiu-se a utilização da ferramenta Git[3].

A escolha do GitLab CE se deve pela cultura de *software* livre, por se tratar de um projeto de âmbito acadêmico e público. Assim, o compartilhamento de conhecimento pode ser realizado de maneira mais fácil e a qualidade final não seria comprometida (RAYMOND, 1999).

[1] <https://gitlab.com/gitlab-org/gitlab-ce>
[2] <https://gitlab.com/brenddongontijo/SMI-UnB>
[3] <https://git-scm.com/>

A licença escolhida para o projeto foi a MIT[4]. Criada pelo *Massachusetts Institute of Technology*, a licença MIT é bastante utilizada pelo meio acadêmico por sua facilidade de implantação e permissividade, autorizando o uso comercial, modificação, distribuição e sublicenciamento (New Media Rights, 2008).

A UnB não possui um modelo de referência para o licenciamento de *software* livre. Assim, a licença escolhida foi adequada, pois prevê a possibilidade de sublicenciar o projeto no futuro, caso seja necessário.

3.3 Requisitos

Os requisitos do projeto foram adquiridos através das reuniões e mapeados, de maneira mais simplificada, em *milestones* (DMITRIY; VALERY, 2013) no repositório. Cada *milestone* possuía um conjunto de *issues* associadas, as quais apresentariam uma terminologia mais técnica da solução em si.

Uma *milestone* funciona como um quadro Kanban, tendo as colunas *To Do*, *Doing* e *Done*. Pode possuir uma data início/fim e é composta por um aglomerado de problemas, comumente chamados de *issues*, os quais ficam transitando pelas colunas, conforme necessário.

As *milestones* e *issues* foram criadas no decorrer do projeto, ou seja, a cada *sprint* foram avaliadas as mudanças de escopo ocorridas, quais seriam as *issues* da próxima *sprint*, se era necessário criar ou atualizar uma determinada *milestone*, etc. As *milestones* do projeto, Figura 2, foram divididas da seguinte maneira:

- *Bugfixes* GPP/MDS: refatorações sobre o incremento de *software* adquirido por uma equipe externa de desenvolvedores e gerentes, esta equipe será explicada na próxima sub-sessão. A refatoração abordava arrumar todo o sistema de login e de usuários, criando diferentes tipos de acesso e algumas permissões importantes;

- Coleta de dados transdutor de energia: gerência de transdutores de energia e coleta de suas informações, interfaciando seus protocolos de transporte e serial. Definição de validações sobre a gerência, armazenamento de transdutores, criação/leitura das mensagens seriais e envio/recebimento dos pacotes presentes na camada de transporte;

- Sistema de Gráficos: gráfico de linhas dinâmico para apresentação das medições realizadas por um transdutor, contendo pontos de máximo/mínimo, opções para geração do gráfico, marcações para períodos com ausência de medições. Além do gráfico, definiu-se uma refatoração sobre todo o *layout* do sistema;

[4] <https://opensource.org/licenses/MIT>

- Campus e Edifícios: mapeamento de como a UnB seria estruturada na aplicação, definição de toda a gerência e validação dos *campi*, edifícios e integração com a gerência de transdutores;
- Transmissão de Dados Entre Servidores: definição de servidores com papéis de mestre e escravo, API *web* capaz de se comunicar com sistemas externos, construção de um ambiente propício a ser de produção, protótipo de comunicação entre administração central e *campi* da UnB.

Figura 2 – *Milestones* realizadas durante as duas *releases* do projeto.

3.4 Visão Geral do Desenvolvimento

O desenvolvimento da parte inicial do projeto foi marcado por duas *releases*, sendo estas:

- *Release* 1: 20/07/2016 a 25/11/2016;
- *Release* 2: 06/03/2017 a 23/06/2017.

3.4.1 *Release* 1

A primeira *release* teve início com um estudo aprofundado sobre o equipamento de medição de energia que seria utilizado. Após o estudo, foi definida a arquitetura inicial do projeto e realizado um pequeno protótipo funcional, Figuras 3 e 4, que conseguia efetivamente coletar os dados de energia.

Foi realizado um guia de instalação para o ambiente de desenvolvimento no repositório. Esse guia objetiva tornar mais acessível o projeto a programadores que estivessem dispostos a realizar novas contribuições.

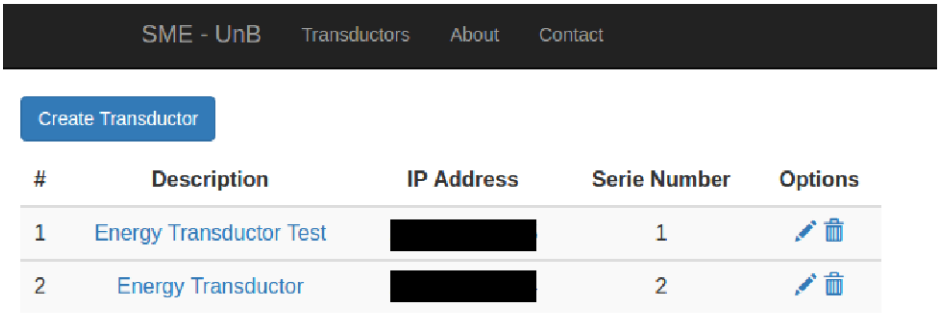

Figura 3 – Protótipo para apresentação dos transdutores *release* 1.

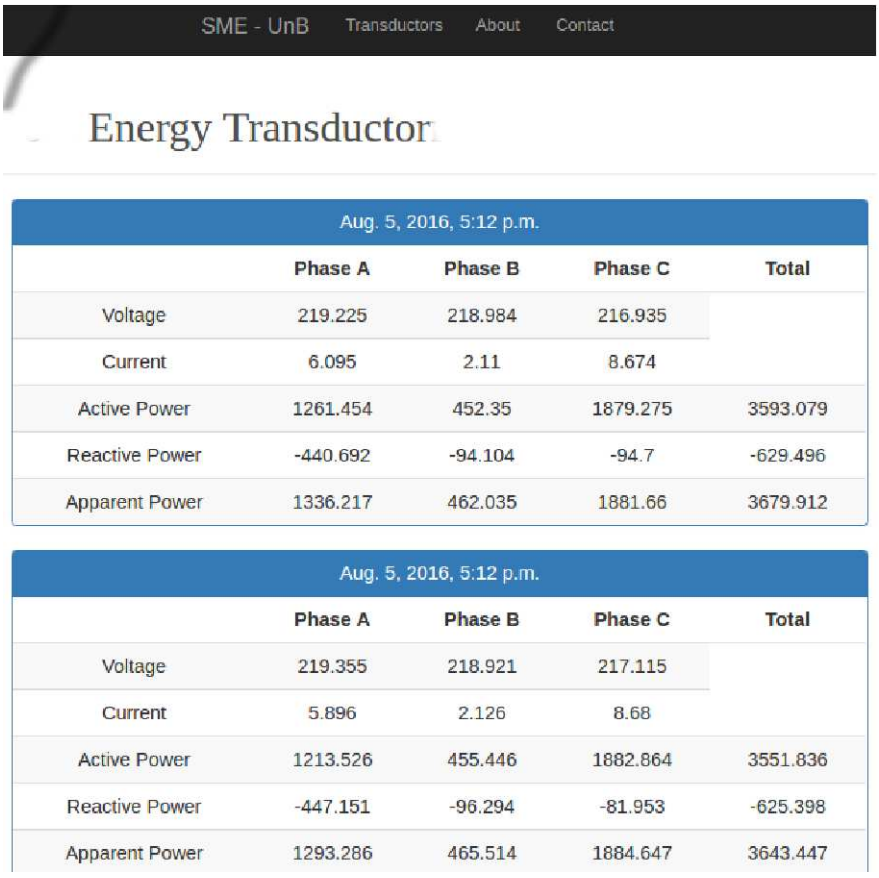

Figura 4 – Protótipo para apresentação de medições de energia *release* 1.

Uma vez que o código estava sendo escrito na linguagem Python 2.7, o desenvolvimento do projeto foi guiado pela utilização de suas boas práticas de programação, definidas pela PEP 8[5]. Essas práticas buscam deixar o código mais legível e proporcionar uma maior eficiência, pois identificam problemas de indentação de código, tamanho de

[5] <https://www.python.org/dev/peps/pep-0008/>

caracteres em uma linha, linhas em banco desnecessárias, bibliotecas importadas múltiplas vezes e outros padrões de estilo. A ferramenta utilizada para realizar a verificação dessas normas foi o flake8[6].

Diversos testes unitários foram realizados a cada término de uma funcionalidade, objetivando uma cobertura de no mínimo 90%. Os testes foram escritos utilizando o módulo *unittest* do próprio Python. Esse ciclo de desenvolvimento permitiu identificar e corrigir pequenos erros de funcionalidades.

Utilizou-se a biblioteca *mock*[7], do *unittest*[8], para que fosse possível definir determinados comportamentos em métodos que utilizassem um outro método durante sua execução, objetivando um maior desempenho na hora de executar a suíte de testes e unitariedade dos métodos. A cobertura total de código obtida ao fim da *release* 1 foi de 95%, com 86 casos de teste.

Com o auxílio do serviço Gitlab CI, foi possível realizar de maneira fácil a integração contínua do projeto. Esse serviço utiliza como princípio um *Runner*, que se baseia em uma máquina virtual isolada, responsável por realizar um *Job*. Um *Job* consiste na execução de comandos pré-definidos no arquivo padrão *.gitlab-ci.yml*.

Diversas decisões arquiteturais e tecnológicas foram realizadas no decorrer da *release* conforme as necessidades de implementação iam surgindo, objetivando modularizar e trazer maior robustez ao sistema. Uma delas foi a utilização da ferramenta `cron`, objetivando realizar uma coleta de dados temporizada e autônoma das medições de energia.

Um ponto importante a se destacar, referente à primeira *release*, é o da colaboração provinda por alunos de duas disciplinas aplicadas pelo curso de Engenharia de *Software* na Faculdade UnB Gama: Gestão de Projetos e Portfólio de *Software* e Métodos de Desenvolvimento de *Software*. As disciplinas em questão eram interdependentes e foram realizadas em conjunto.

O autor do trabalho atuou como *Product Owner* (SCHWABER; SUTHERLAND, 2001) para as equipes, definindo os requisitos que deveriam ser feitos por elas. O *Product Owner* (PO), ou dono do produto, é o responsável por maximizar o valor do produto e do trabalho do Time de Desenvolvimento realizando o gerenciamento do *Backlog* do Produto[9]. Os requisitos acordados com as equipes foram refentes à gerência de usuários, autenticação, geração de relatórios, *log* para o sistema, sistema de alarme para eventos indesejados e autenticação na aplicação. Nem todos os requisitos foram realizados e os incrementos de *software* providenciados ao projeto foram, em sua grande parte, referentes ao gerenciamento de usuários e autenticação.

6 <https://pypi.python.org/pypi/flake8>
7 <https://pypi.python.org/pypi/mock>
8 <https://docs.python.org/3/library/unittest.html>
9 Lista com todas as funcionalidades desejadas para um produto.

3.4.2 *Release* 2

A segunda *release* teve início com uma forte refatoração sobre os incrementos de *software* provindos das equipes de GPP e MDS. Em conjunto, visto que o projeto estava em Python 2.7, foi realizada uma mudança para Python 3.5. Junto a essa mudança, algumas modificações se concretizaram na autenticação do sistema, buscando que essa fosse realizada por meio de e-mail, em vez de nome de usuário.

Foi realizado um novo *layout* para o sistema, visto que seria mais agradável ao usuário acessar páginas com cores agradáveis, botões significativos, perguntas de confirmação para ações importantes, *links* para páginas anteriores e afins.

A apresentação periódica das informações energéticas se deve através de um gráfico de linhas dinâmico, com opções de aproximação e movimentação, para que o usuário pudesse visualizar pequenas ou longas faixas de medição. A biblioteca utilizada para realizar o gráfico foi a Matplotlib[10]. Foram realizadas 3 opções para geração de gráfico, sendo estas:

- Medições de Hoje: gera um gráfico de 00:00 até o horário atual;
- Medições de Dias Anteriores: é escolhido um, entre os 7 dias anteriores, para que seu gráfico possa ser gerado;
- Inserir Data Manualmente: um gráfico é gerado com base em uma data final e inicial. O período máximo definido entre as datas foi de 1 semana.

Com a expansão da aplicação, foi necessário realizar algumas mudanças para os usuários. A primeira mudança foi referente aos diferentes níveis de acesso, sendo permitido a criação de usuários administradores e normais. A criação dos usuários da aplicação só pode ser realizada por administradores, que possuem acesso sobre essa. A segunda mudança foi relacionada às permissões dos usuários normais, referentes à gerência de prédios ou transdutores. Essa gerências garantem os direitos de incluir, alterar, habilitar e desabilitar suas entidades.

Foi realizada uma API aberta que se comunica com algum sistema externo e expõe as informações de prédios, trandutores e medições de energia. Essa API é fundamental para o funcionamento do projeto, pois foi definido que toda gerência das entidades da aplicação será realizada na administração central.

O protótipo para comunicação entre uma administração central e os *campi* da UnB foi realizado e utiliza a API anteriomente citada. Nesse protótipo dois tipos de sincronização são realizadas: sincronia de entidades e de medições. A sincronia de entidades é

[10] <https://matplotlib.org/>

realizada quando se tenta cadastrar, por exemplo, um transdutor de um prédio presente em um determinado campus. Esse cadastro é concretizado apenas se houver uma comunicação entre o servidor da administração central e o servidor desse prédio. A sincronia de medições é feita quando a administração central extrai as últimas medições realizadas por cada prédio cadastrado.

Muitos testes unitários foram realizados nessa *release*, objetivando cobrir as funcionalidades implementadas. Não foi possível realizar os testes para a API, pelo período de desenvolvimento ter chegado ao fim. A cobertura total de código obtida foi de 93% e foram realizados 106 casos de teste a mais para a aplicação.

3.5 Implantação

Não foi possível colocar o SMI-UnB em produção, visto que muitas funcionalidades importantes ainda precisam ser desenvolvidas, porém, algumas de suas funções importantes foram implementadas com sucesso.

Alguns conhecimento provindos pelo *Devops* foram utilizados para auxiliar na distribuição do SMI-UnB. O *Devops* é um processo de desenvolvimento de *software* que valoriza comunicação e colaboração entre gerentes de produto, desenvolvedores de *software* e outros profissionais. O *DevOps* também automatiza os processos de integração de *software*, testes, implantação e mudanças de infraestrutura (LOUKIDES, 2012).

Utilizou-se o Docker[11] para que fosse possível criar um ambiente unificado para o sistema, visando evitar futuros problemas de implantação.

O Docker é uma plataforma de contêiners de *software*. Um contêiner, Figura 5, possui empacotado tudo que é necessário para se executar um *software* completo ou parte dele. Diferente das máquinas virtuais, os contêiners são executados em uma mesma máquina, compartilhando o *kernel* do seu sistema operacional, sendo que cada um terá seu processo isolado no espaço de usuário.

Para Tanenbaum (TANENBAUM, 2007), o sistema operacional é a peça mais básica de *software* e opera em modo núcleo, possuindo acesso completo a todo o *hardware* e ao conjunto de instruções oferecidos pela máquina. O resto do *software* opera em modo usuário, onde é disponível apenas um conjunto de instruções da máquina para execução.

Realizou-se um ambiente para testes no campus UnB-Gama, que possuía o SMI-UnB instalado como um serviço, objetivando averiguar o que havia sido implementado.

[11] <https://www.docker.com/>

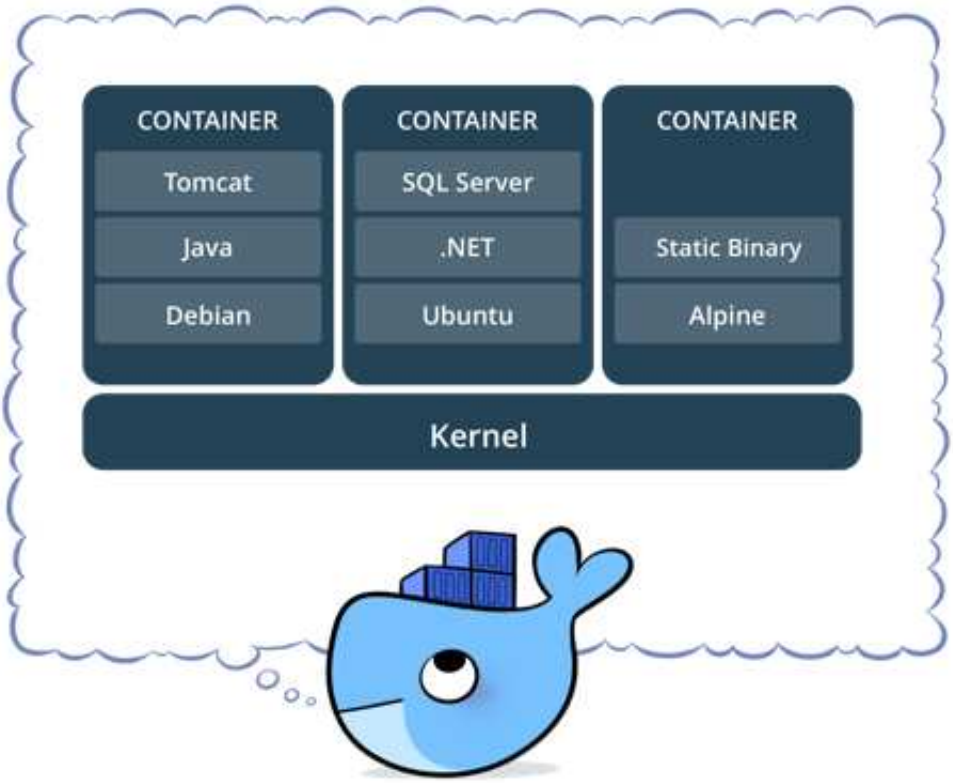

Figura 5 – Exemplo de contêiners providos pelo Docker. Fonte: (DOCKER, 2017)

4 Características do SMI-UnB

4.1 Visão Geral

O SMI-UnB se baseia um sistema *web* para armazenamento e coleta, setorial e temporal, dos insumos da Universidade de Brasília. Tem como base a existência de uma administração central e prédios contidos nos *campi* da UnB. A administração central é expressa por um servidor mestre e realiza a gerência de todo o sistema, ou seja, todos os prédios, usuários e afins são cadastrados nela. Os prédios dos *campi* são representados por servidores escravos e são responsáveis por coletar e armazenar seus insumos consumidos. O insumo escolhido para monitoramento, em um primeiro momento, foi o de energia.

De tempos em tempos, o servidor mestre realiza uma sincronização de informações com todos os servidores escravos, objetivando unificar os insumos da Universidade na administração central.

4.2 Tecnologias Escolhidas

Definiu-se a utilização do *framework* Django (Django Software Foundation, 2005) como estrutura base do SMI-UnB, visto que o sistema consistiria em uma aplicação *web* e não haveria nenhuma limitação que impedisse a utilização desse.

4.2.1 Django

O Django é um *framework* para desenvolvimento *web* implementado na linguagem Python[1]. Sua arquitetura se inspira no modelo tradicional MVC (*Model View Controller*), porém, com algumas especificidades. A comunidade *Django* adota o acrônimo MTV (*Model Template View*), onde os papéis de *model*, *view* e *controller* são redefinidos como:

- *Model*: corresponde à *model* do MVC tradicional e representa as classes que popularão as tabelas do banco de dados. O *Django* possui um *Object-Relational Mapping* (ORM), para realizar a manipulação dessas tabelas, não sendo necessária a escrita de consultas em SQL para a persistência das informações;
- *Template*: corresponde aproximadamente à *view* do MVC tradicional e descreve como as informações serão apresentadas para o usuário;
- *View*: representada por uma função *callback* referente a uma classe de URLs, descrevendo quais informações serão apresentadas e como elas serão enviadas para o

[1] <https://www.python.org/>

template. Alguns autores defendem que a view corresponde ao *controller* do MVC tradicional, mas para os próprios desenvolvedores do *Django*, a *view* deve ser minimalista e boa parte do papel do *controller* deve ser implementado nas próprias classes dos modelos.

Na nomenclatura do *Django*, um conjunto de funcionalidades pode ser agrupado em uma aplicação, chamada de *app*. Cada aplicação possui suas próprias *models*, *views* e *templates*.

A Figura 6 mostra a arquitetura do *Django*, apresentando as camadas do MTV durante a comunicação com o navegador até o acesso ao banco de dados. O *URL dispatcher* identifica endereços requisitados pelo usuário e realiza o redirecionamento da requisição para a aplicação correta. A coordenação de requisições entre o *URL dispatcher* e a *view* e da *view* até o *template* é feita pelos chamados *Middlewares*. Os *Middlewares* realizam a persistência de informações entre as diferentes camadas.

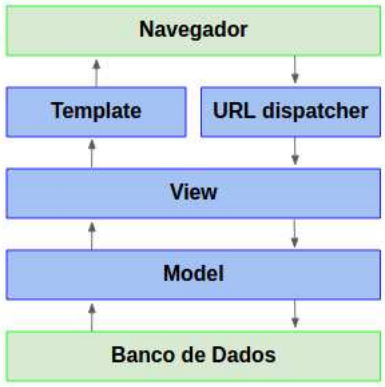

Figura 6 – Arquitetura MTV *Django*.

4.3 Protocolos Utilizados

Antes de começar o desenvolvimento, foi necessário realizar um profundo estudo sobre o equipamento para medição de dados de energia, comumente chamado de transdutor, visto que esse já havia sido pré-designado para utilização, devido a contratos realizados anteriormente pela Universidade de Brasília.

O equipamento em questão foi o TR 4020, Figura 7, disponibilizado pela empresa Embrasul (EMBRASUL, 2017). Segundo seu manual, possui período de amostragem a cada 50 ms, comunica-se utilizando o protocolo de comunicação Modbus, no modo RTU, com velocidades de 10M/100Mbps em sistemas Ethernet, utilizando o protocolo UDP como transporte. No datagrama UDP, no campo de dados, o protocolo ModBUs-RTU é encapsulado, sendo que a porta de comunicação padrão é a 1001. O endereço ModBus dos

equipamentos por padrão é 1, onde a diferenciação entre equipamentos se dá pelo número de IP.

Figura 7 – Transdutor TR4020. Fonte: (EMBRASUL, 2017)

4.3.1 *Modbus-RTU*

Modbus (MODICON, Inc., 1996) é um protocolo serial utilizado para transmitir informações entre dispositivos eletrônicos. Suas mensagens utilizam a arquitetura de mestre-escravo, como mostra a Figura 8. Nesta arquitetura, o papel de mestre é designado ao dispositivo que envia as requisições e de escravo ao que responde passivamente a elas.

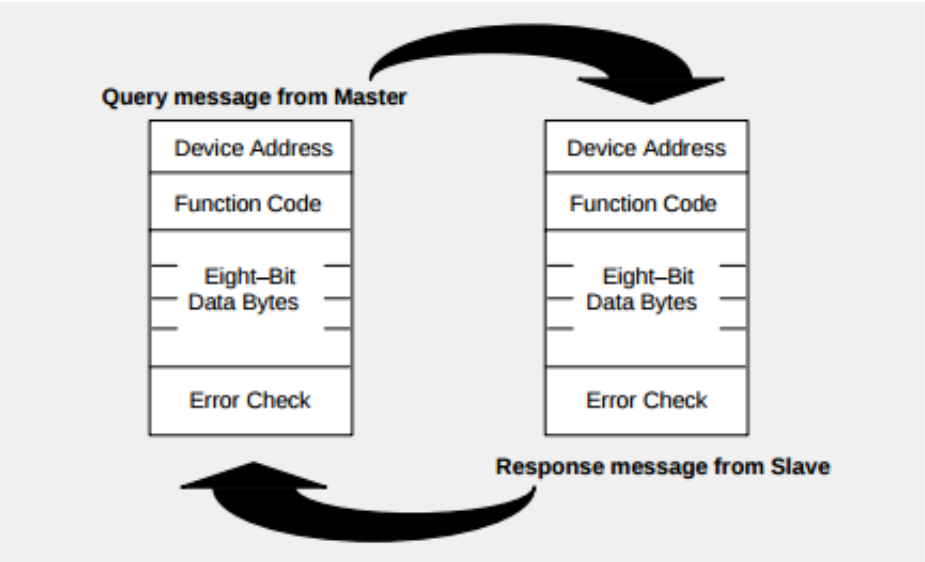

Figura 8 – Comunicação Mestre-Escravo *Modbus*. Fonte: (MODICON, Inc., 1996)

Quando controladores são configurados para se comunicarem em uma rede Modbus, usando o modo *Remote Terminal Unit* (RTU), cada *byte* contem duplas hexadecimais de 4 *bits*. A maior vantagem em se utilizar esse modo é que sua grande densidade de caracteres permite uma maior taxa de transferência comparado ao modo ASCII, em uma mesma taxa de transmissão (MODICON, Inc., 1996).

Uma mensagem em Modbus RTU possui 16 *bytes* e é definida da seguinte maneira:

- Identificador do Aparelho: 2 *bytes*;
- Código de Função: 2 *bytes*, define qual tipo de operação o equipamento irá realizar;
- Campo de Dados: 8 *bytes*, sendo 4 *bytes* para indicar o endereço do primeiro registrador requisitado e 4 *bytes* para indicar a quantidade de registradores que serão lidos;
- *Cyclic Redundancy Check* (CRC): 4 *bytes* para verificação de erros.

A resposta do escravo possui a seguinte estrutura:

- Identificador do Aparelho: 2 *bytes*;
- Código de Função: 2 *bytes*, define qual tipo de operação o equipamento irá realizar;
- Tamanho do *Payload*: 2 *bytes*, define o tamanho do campo de dados em *bytes*;
- Campo de Dados: possui tamanho variável, de acordo com o valor do campo anterior;
- *Cyclic Redundancy Check* (CRC): 4 *bytes* para verificação de erros.

4.3.1.1 Leitura utilizando equipamento TR4020

O transdutor TR4020 possui valores em inteiro ou ponto flutuante para os registros armazenados em seus registradores. Para ler e compor um valor em inteiro, deve-se realizar a leitura de 1 registro Modbus (16 *bits*), sendo que esse registro possui 2 *bytes* concatenados em sequência como *Byte*-A e *Byte*-B, sendo o *Byte*-A mais significativo. Para valores em ponto flutuante (IEEE Task P754, 1985), é realizada a leitura de 2 registros Modbus (16 *bits* cada). Esse registro possui 4 *bytes* concatenados em sequência como *Byte*-A, *Byte*-B, *Byte*-C e *Byte*-D. Por exemplo, para ler um valor em ponto flutuante que expressa a tensão na fase A (endereços 68 e 69), escreve-se na comunicação a sequência de *bytes*, conforme a Figura 10.

```
01h # Identificador do aparelho
03h # Código de função (Leitura)
00h # Primeiro byte do endereço 68
44h # Segundo byte do endereço 68
00h # Bytes a serem lidos
02h # Bytes a serem lidos (total: 2 bytes)
84h # CRC
1Eh # CRC
```

Figura 9 – Requisição para leitura de tensão na fase A, utilizando TR4020.

Tem-se como resultado, por exemplo, a resposta contida na Figura 10.

```
01h # Identificador do Aparelho
03h # Código de função (Leitura)
04h # Tamanho do payload (4 bytes)
38h # Primeiro byte do valor para tensão na fase A
88h # Segundo byte do valor para tensão na fase A
43h # Terceiro byte do valor para tensão na fase A
59h # Quarto byte do valor para tensão na fase A
C7h # CRC
87h # CRC
```

Figura 10 – Resposta para leitura de tensão na fase A, utilizando TR4020. O exemplo representa um valor de 217.220 para a leitura da tensão.

O procolo utiliza o padrão *big endian* (TANENBAUM; GOODMAN, 1998) para representação de números inteiros, enquanto a maioria das máquinas atualmente utiliza o padrão *little endian*, assim, é necessário inverter a ordem dos *bytes* para transmitir os valores de tensão. O padrão *little endian* define que a organização dos *bytes* de uma palavra deve começar dos *bits* menos significativos para os mais significativos, enquanto o *big endian* utiliza a ordem reversa.

4.3.2 UDP

O protocolo *User Datagram Protocol* (UDP) é um protocolo da camada de transporte, não orientado a conexões. Seu cabeçalho, Figura 11, possui 8 *bytes*, seguido de uma carga útil. As portas apresentadas no cabeçalho representam as máquinas de origem e destino (TANENBAUM, 2002).

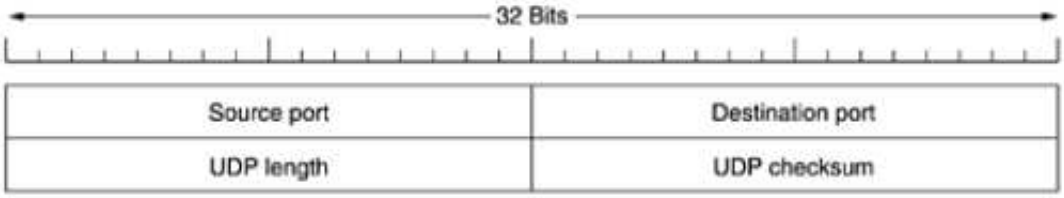

Figura 11 – Cabeçalho do UDP. Fonte: (TANENBAUM, 2002)

Para Tanenbaum, a camada de transporte é o núcleo de toda a hierarquia de protocolos. Sua função é promover uma transferência de dados confiável e econômica entre a máquina de origem e a máquina de destino, independente das redes físicas em uso no momento.

Nas redes de acesso empresarial, uma rede de área local (LAN) é usada para conectar um sistema final a um roteador de borda. Existem muitos tipos diferentes de tecnologias LAN. No entanto, a *Ethernet* é a tecnologia de acesso mais prevalente nas redes corporativas. A *Ethernet* opera 10 Mbps ou 100Mbps e utiliza cabos par trançado

para conectar uma série de sistemas finais uns com os outros e com um roteador de borda. O roteador de borda é responsável por rotear pacotes que tenham destinos fora dessa LAN. A *Ethernet* usa um meio compartilhado para que os usuários finais compartilhem a taxa de transmissão da LAN (KUROSE; ROSS, 2002).

O protocolo de camada de rede da Internet se chama "Protocolo da Internet", ou IP. O IP fornece comunicação lógica entre hosts, possuindo um modelo de entrega de melhor esforço. Isso significa que o IP realiza seu "melhor esforço" para fornecer segmentos entre hosts comunicantes, mas não oferece garantias. Em particular, não garante a entrega do segmento, não garante a entrega ordenada de segmentos e garante a integridade dos dados nos segmentos (KUROSE; ROSS, 2002).

4.4 Armazenamento das Informações

Definiu-se que os *campi* da UnB seriam divididos por região administrativa, cada campus teria um conjunto de edifícios, cada edifício poderia possuir diversos transdutores e os transdutores realizariam diversas medições. Para que isso fosse possível, criou-se os *apps campuses*, *buildings* e *transductor*.

A ferramenta utilizada para realizar os diagramas de classes dos *apps* do SMI-UnB foi a StarUML[2]. Além disso, não foram expressas nos diagramas as classes referentes as *views* e formulários (*forms*), objetivando uma menor poluição visual.

No *app campuses*, Figura 12, são definidos os modelos para uma região administrativa e os *campi* em si.

Seguindo o mesmo princípio, o *app buildings*, Figura 13, define um modelo para os edifícios e possui um *manager*, buscando auxiliar a manipulação de seus *querysets*. Um *queryset* representa uma coleção de objetos do banco de dados (Django Software Foundation, 2005).

As medições de energia iriam seguir o esquema de um sistema trifásico, onde suas fases foram representadas como A, B e C. Os parâmetros afetos ao suprimento de energia elétrica definidos para serem coletados foram:

- Tensão;
- Corrente;
- Potência Ativa;
- Potência Reativa;

[2] <http://staruml.io/>

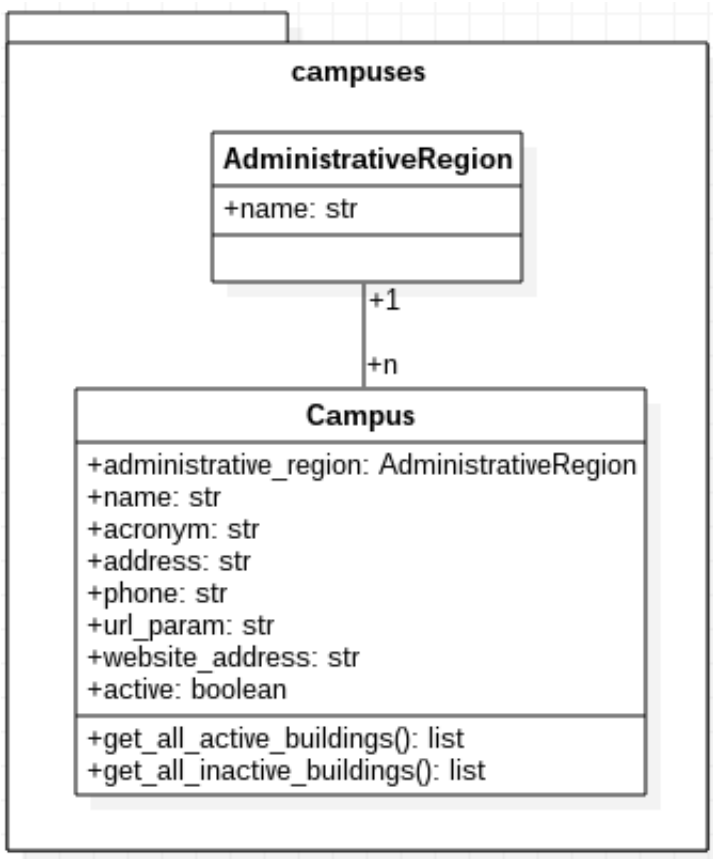

Figura 12 – Diagrama de Classes para *app campuses*.

- Potência Aparente.

Os transdutores e suas medições são definidos no *app transductor*, Figura 14. Uma das pecularidades presente nos transdutores é a necessidade de um modelo de transdutor. Esses modelos, definidos pela classe *TransductorModel*, possuem toda a informação necessária para que seja possível realizar a coleta de dados, definindo o endereço e o tipo (*int* ou *float*) dos registradores que serão lidos, além dos protocolos seriais e de transporte utilizados.

As classes *Transductor* e *Measurements* são *Polymorphic Models*[3] e possuem alguns métodos base. Com essas classes é possível extrair com mais facilidade e rapidez todas as suas classes filhas, que no caso seriam as representações dos recursos monitorados pela UnB. As especializações criadas foram referentes aos transdutores de energia e medições de energia, representados pelas classes *EnergyTransductor* e *Energymeasurements*. Além disso, existe uma classe auxiliar, chamada *EnergyOperations*, responsável por realizar cálculos matemáticos com os dados de energia coletados.

[3] <https://django-polymorphic.readthedocs.io/en/stable/>

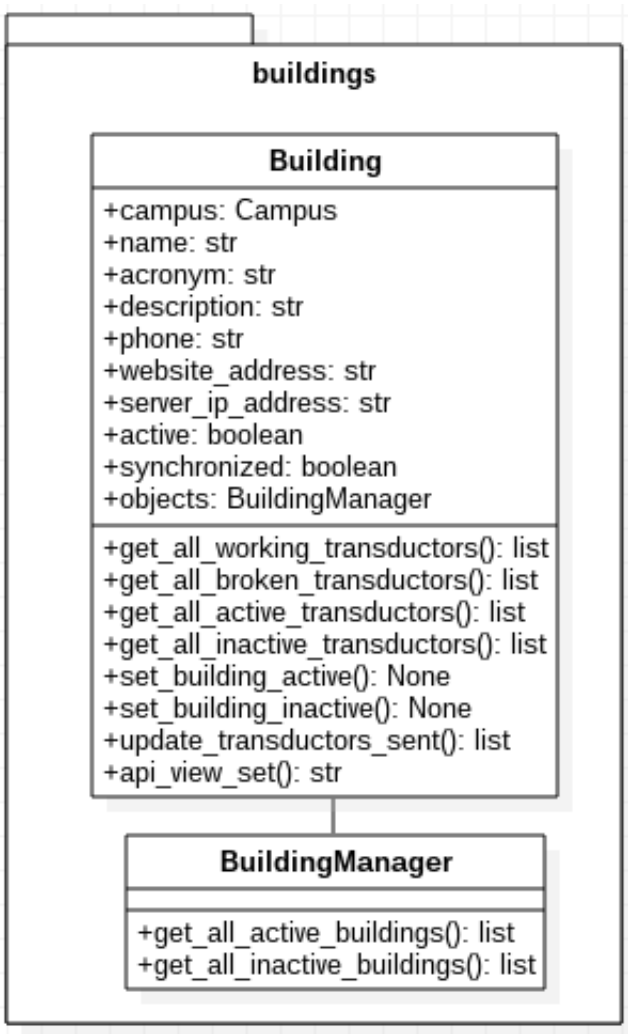

Figura 13 – Diagrama de Classes para *app buildings*.

4.5 Coleta de dados

Definiu-se que existirão dois tipos de servidores para ser possível realizar a coleta de dados inter-campi: mestre e escravo. O servidor mestre seria a representação da administração central e os escravos seriam prédios, espalhados pelos *campi* da UnB, que realizariam a coleta de dados de seus transdutores, presentes na sua mesma rede.

O tempo para coleta das informações foi definido da seguinte maneira:

- A cada 1 hora, o servidor mestre seria responsável por realizar uma sincronia com todas as medições realizadas pelos escravos;

- A cada 1 minuto, os servidores escravos seriam responsáveis por realizarem suas coletas de dados.

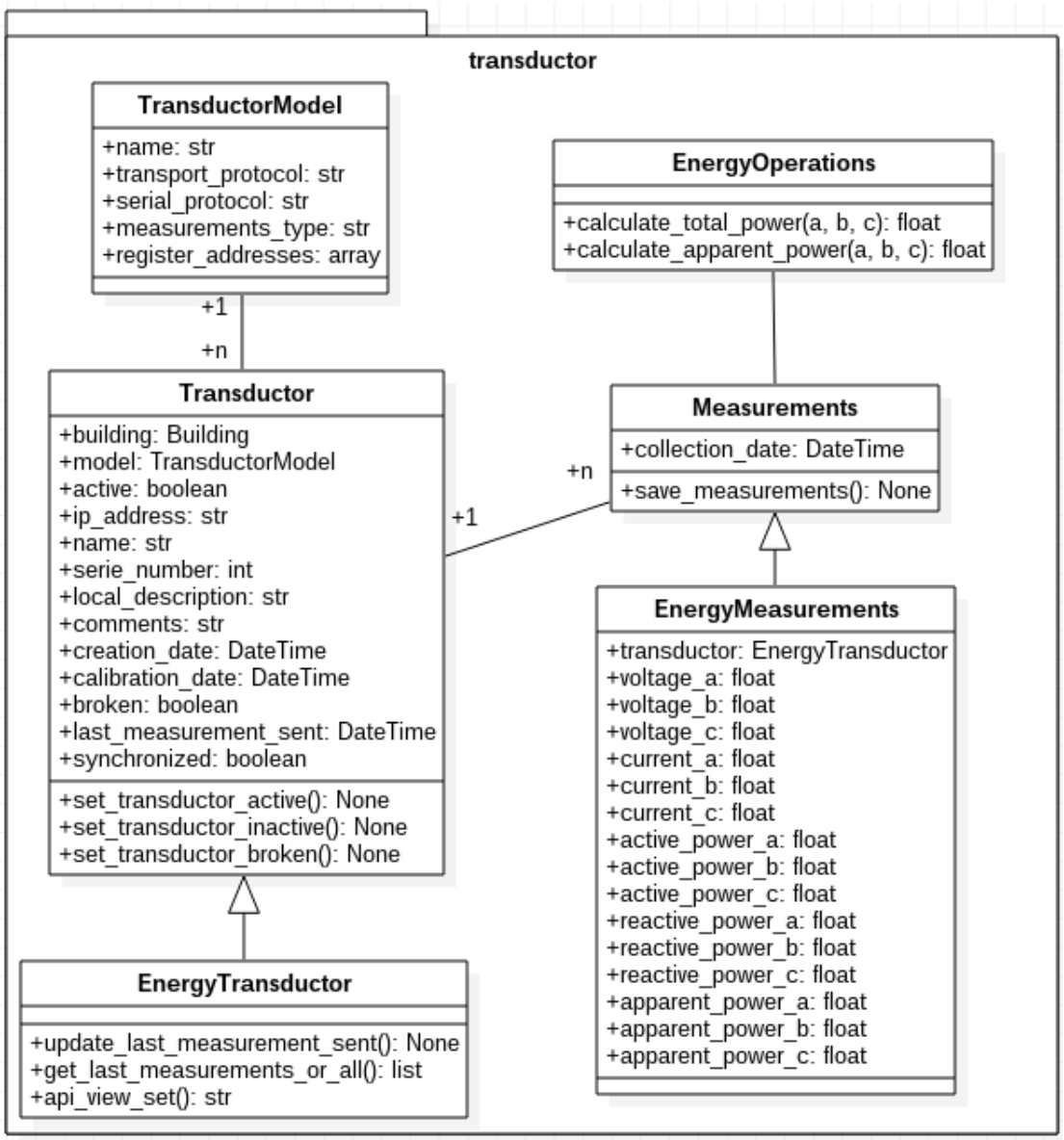

Figura 14 – Diagrama de Classes para *app transductor*.

4.5.1 Servidor Escravo

A coleta de dados, realizada por um prédio (servidor escravo), é feita com o auxílio do app *data_reader*, Figura 15.

A classe *SerialProtocol* e *TransportProtocol* são abstratas e possuem alguns métodos base, para que possa ser possível criar diferentes tipos de especializações, conforme a aplicação necessite. As duas inicialmente criadas, foram referentes aos protocolos Modbus-RTU e UDP.

A coleta de dados para cada prédio, com base no equipamento TR4020, é ilustrada pela Figura 16. Foi utilizada a ferramenta Bizagi[4] para a realização da modelagem.

[4] <https://www.bizagi.com/>

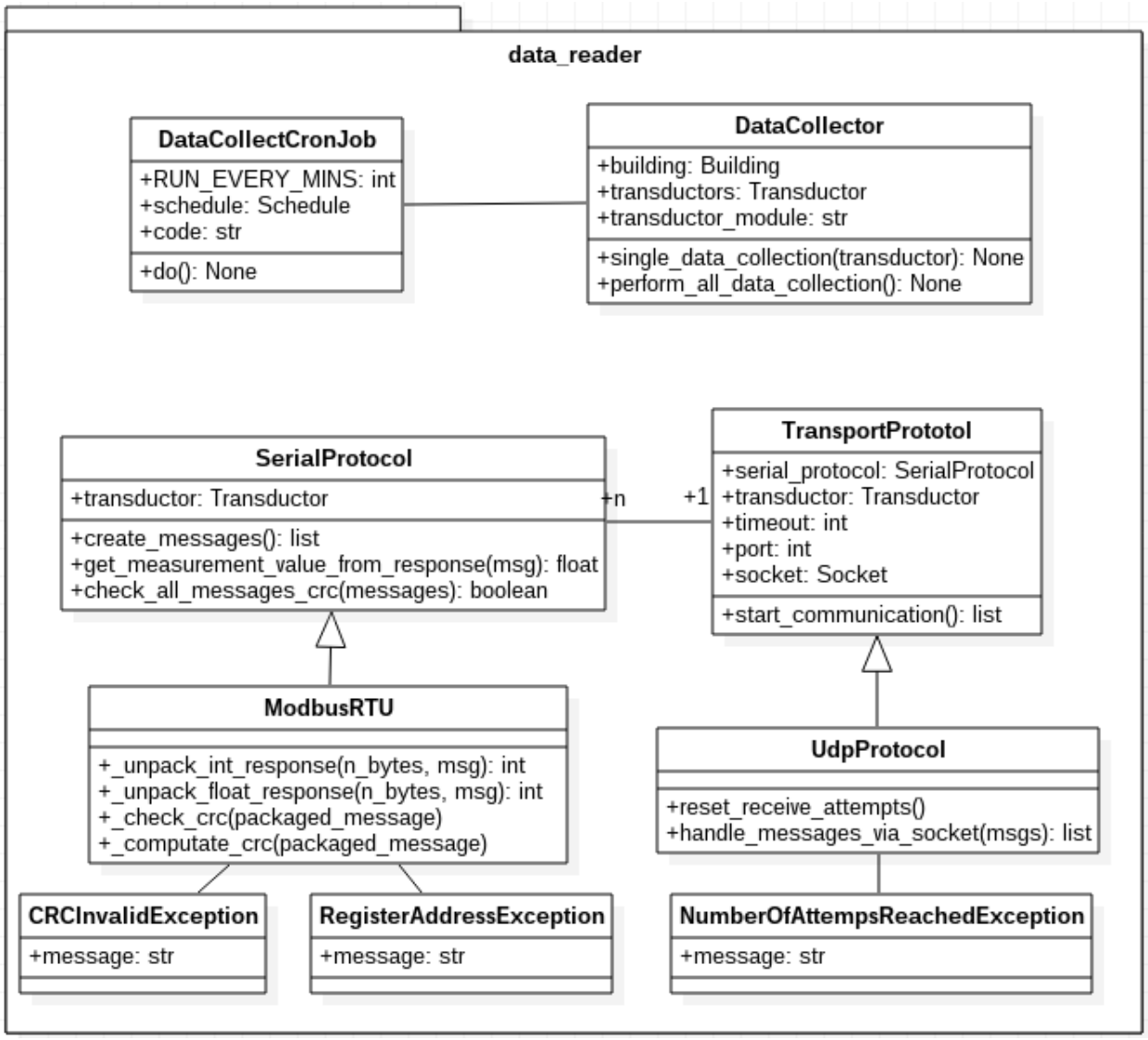

Figura 15 – Diagrama de Classes para *app data_reader*.

Inicialmente, a classe *DataCollector* identifica todos os transdutores que estão no prédio e cria uma *thread* para cada um, objetivando paralelismo na coleta de dados. Segundo Tanembaum (TANENBAUM, 2007), as *threads* são entidades escalonadas para a execução sobre a CPU, permitindo que múltiplas execuções ocorram no mesmo ambiente de um processo, com um grande grau de independência uma da outra.

Em cada *thread* é identificado o modelo TR4020 para o transdutor, o qual possui as informações sobre os protocolos Modbus-RTU e UDP. A classe *ModbusRTU* prepara todas as mensagens que deverão ser lidas pelo equipamento, uma para cada grandeza energética e para cada fase. As mensagens seriais criadas são recebidas pela classe *UdpProtocol*, que tentará realizar a comunicação com o aparelho. Os pacotes são enviados e recebidos, um por um, até que todos sejam recebidos corretamente. Com os pacotes recebidos, são extraídas suas cargas úteis, que basicamente correspondem às medições de cada grandeza, logo após, elas são lidas, utilizando a classe *ModbusRTU*. Por fim, a classe *DataCollector*

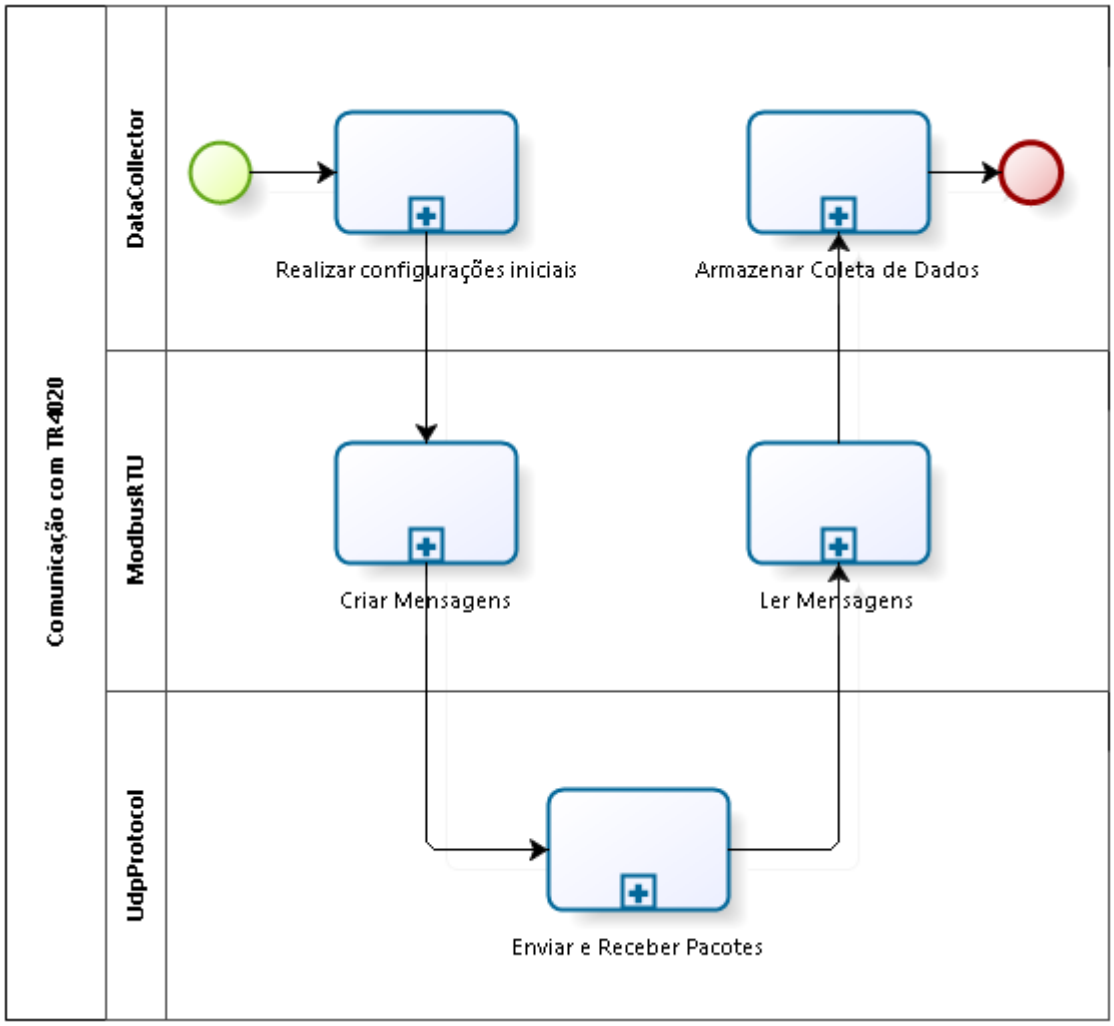

Figura 16 – Coleta de dados energéticos utilizando TR4020.

recebe as medições e as salva.

A coleta temporizada, a cada 1 minuto, das medições energéticas realizadas pelos prédios utiliza a ferramenta `cron` (VIXIE, 1987), presente em sistemas UNIX. O `cron` é um *daemon* para executar comandos de agendamento. Um *daemon* é um programa executado em segundo plano e visa estar sempre em execução, caso seja iniciado. Além disso, o `cron` utiliza o `crontab` para manter os arquivos, que possuem instruções a serem executadas periodicamente, de cada usuário. Um arquivo mantido pelo `crontab` deve seguir a estrutura da Figura 17.

```
1  minuto hora dia_mes mes dia_semana comando_para_execucao
2  # Linha em banco para deixar cron valido
```

Figura 17 – Estrutura de um arquivo mantido pelo *crontab*.

Adicionou-se um novo comando para a aplicação, chamado *runcrons*. Esse co-

mando é definido pela ferramenta django-cron[5] e basicamente executa um código que possua como base a classe *CronJobBase*. Esse código, após ser executado, é bloqueado até que o tempo de espera para outra execução seja atingido, como uma espécie de cronômetro.

Criou-se a classe *DataCollectCronJob*, Algoritmo 4.1, objetivando invocar o *DataCollector* para realizar a coleta de dados. O arquivo mantido pelo crontab para realizar a coleta, periodicamente a cada 1 minuto, foi definido conforme o Algoritmo 4.2.

```
class DataCollectCronJob(CronJobBase):
    RUN_EVERY_MINS = 0
    schedule = Schedule(run_every_mins=RUN_EVERY_MINS)
    code = 'smi_unb.data_reader.cronjob.DataCollectCronJob'

    def do(self):
        data_collector = DataCollector()
        data_collector.perform_all_data_collection()
```

Algoritmo 4.1 – Classe *DataCollectCronJob*.

```
* * * * * python3 /SMI-UnB/manage.py runcrons \
smi_unb.data_reader.cronjob.DataCollectCronJob
# Necessary line at end of file to make cron valid
```

Algoritmo 4.2 – *Cron* para execução da coleta dos dados de energia.

4.5.2 Servidor Mestre

O servidor mestre é responsável por realizar uma sincronização de entidades e medições de energias utilizando uma API *web*. Ressalta-se que a sincronização de entidades será explicada na próxima sessão.

Application Programming Interface (API) é um conjunto de requisitos que regem como um aplicativo pode conversar com outro. As APIs realizam isso expondo algumas das funções internas de um programa para o mundo exterior de forma limitada, possibilitando que os aplicativos compartilhem dados e tomem ações em nome do outro, sem exigir que os desenvolvedores compartilhem todo o código do *software* (PROFFITT, 2013).

Quando usado no contexto do desenvolvimento *web*, uma API é tipicamente definida como um conjunto de requisições do protocolo HTTP, juntamente com uma definição da estrutura de mensagens de resposta, geralmente utilizando as linguagens *Extensible Markup Language* (XML) ou *JavaScript Object Notation* (JSON) (BENSLIMANE; DUSTDAR; SHETH, 2008).

[5] <http://django-cron.readthedocs.io/en/latest/>

O *Hypertext Transfer Protocol* (HTTP), é um protocolo *web* presente na camada de aplicação do modelo OSI e é implementado por dois programas: um cliente e outro servidor. Os programas cliente e servidor conversam entre si, trocando mensagens HTTP, sendo que o protocolo define como o cliente (por exemplo, um navegador) solicitará páginas *web* de um servidor (por exemplo, o Django) e como o servidor irá transferir essas páginas para o cliente (KUROSE; ROSS, 2002).

A API utilizada no projeto baseou-se no Django REST Framework (CHRISTIE, 2011). O *Representational State Transfer* (REST) (FIELDING, 2000) é um estilo arquitetural para projetar sistemas distribuídos e apresenta as seguintes características:

- Estado e funcionalidade são divididos em recursos distribuídos;
- Todo recurso é exclusivamente endereçável, usando um conjunto uniforme e mínimo de comandos;
- O protocolo é cliente/servidor, sem estado, em camadas e suporta armazenamento em *cache*.

O Django REST Framework utiliza alguns métodos HTTP para mapear as operações CRUD (criar, resgatar, atualizar e deletar) nas requisições HTTP, sendo estes:

- GET: recuperar informações de uma entidade;
- POST: criar ou atualizar uma entidade;
- PUT: criar ou atualizar uma entidade. O método PUT é idempotente, ou seja, se uma operação for realizada duas vezes sobre o mesmo objetivo, não haverá efeito;
- PATCH: modificar parcialmente uma entidade;
- DELETE: deletar uma entidade.

O conceito de *endpoints* é utilizado pelo Django REST Framework, objetivando a interação com a API do lado do servidor, pois especificam onde os recursos podem ser acessados. Uma das classes padrão utilizada no projeto foi a *ModelViewSet*. Além disso, o Django REST Framework utiliza serializadores. Os serializadores permitem que dados complexos, como *querysets* e instâncias de modelos, sejam convertidos em tipos de dados Python nativos que podem ser facilmente processados em JSON, XML ou outros tipos de conteúdo. Os serializadores também fornecem desserialização, permitindo que os dados analisados sejam convertidos novamente em tipos complexos, após realizada uma validação dos dados recebidos (CHRISTIE, 2011).

Realiza-se a sincronia de dados por meio da classe *EnergyMeasurementSynchronizer*, presente no *app api*, Figura 18. Em linhas gerais, essa classe realiza duas requisições HTTP para cada servidor escravo. Na primeira requisição são consumidas as medições de energia mais recentes de cada transdutor, via API. Com as medições resgatadas, atualiza-se o horário da última coleta de dados de cada transdutor, no servidor mestre. Após atualizados os horários no mestre, realiza-se a segunda requisição, que atualiza o horário da última coleta de dados de todos os transdutores presentes no escravo.

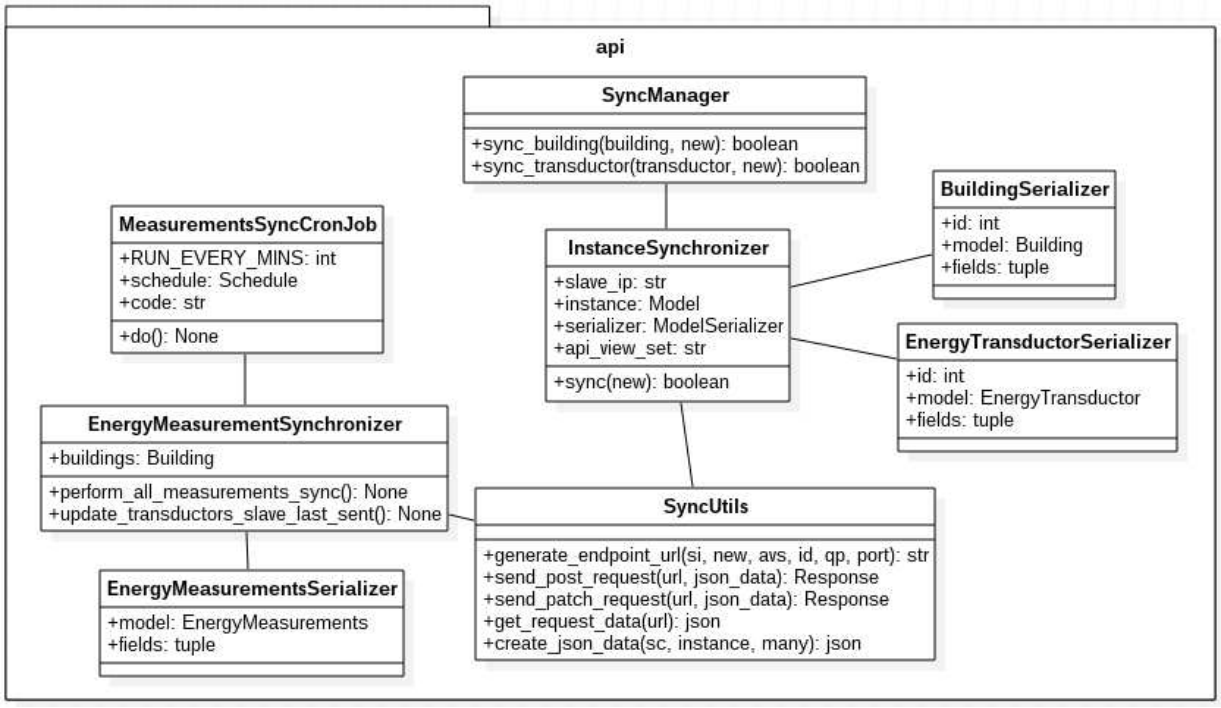

Figura 18 – Diagrama de Classes para *app api*.

O `crontab` para se realizar a sincronia das medições, a cada 1 hora, é expresso pelo Algoritmo 4.3 e utiliza a classe *MeasurementsSyncCronJob*, Algoritmo 4.4.

A base da API foi realizada com sucesso, para que no futuro haja apenas aprimoramentos, como a sua autenticação e hiperlincamento, conforme recomendado pelo Django REST Framework.

```
0 * * * * python3 /SMI-UnB/manage.py runcrons \
smi_unb.api.cronjob.MeasurementsSyncCronJob
# Necessary line at end of file to make cron valid
```

Algoritmo 4.3 – *Cron* para execução da sincronia dos dados de energia.

```
class MeasurementsSyncCronJob(CronJobBase):
    RUN_EVERY_MINS = 59
    schedule = Schedule(run_every_mins=RUN_EVERY_MINS)
    code = 'smi_unb.api.cronjob.MeasurementsSyncCronJob'
```

```
def do(self):
    e_synchronizer = EnergyMeasurementSynchronizer()
    e_synchronizer.perform_all_measurements_sync()
```

Algoritmo 4.4 – Classe MeasurementsSyncCronJob.

4.6 Segurança em Geral

Realizou-se um sistema de *login* por meio de e-mail, para facilitar os usuários a se autenticarem no sistema. O Django já possui um módulo de autenticação bem definido, que realiza tanto a autenticação quanto a autorização de um usuário. Como esse módulo realizava *login* por meio do nome de usuário, algumas mudanças foram implementadas para ser possível o *login* por meio de e-mail. Para isso, criou-se o *app authentication*, Figura 19, o qual define a classe *EmailBackend*, que por sua vez realiza a autenticação por e-mail.

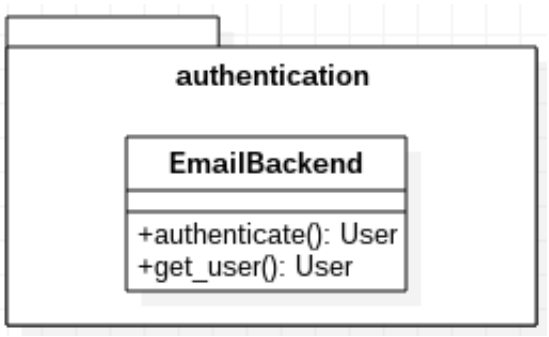

Figura 19 – *App* authentication

Os transdutores e prédios presentes no sistema não podem ser excluídos, tendo em vista a importância de se deixar registrado suas medições realizadas. Para isso acontecer, alguns botões de habilitar e desabilitar foram adicionados à aplicação e um atributo referente a ativo foi adicionado aos seus modelos.

O servidor mestre é responsável por realizar todo o registro, edição, ativação e desativação de transdutores e prédios. Assim, sempre que se tenta realizar uma operação desse tipo, o mestre realiza uma verificação, a nível de formulário, com o escravo. Essa verificação é feita por meio de uma requisição HTTP, utilizando o método GET e possui um *timeout* configurável. Caso o código de resposta recebido seja válido, o mestre tenta realizar uma sincronização com escravo via API, utilizando a classe *SyncManager*, Figura 18. Ressalta-se que esta solução é um protótipo e deve ser aperfeiçoada, buscando aceitar requisições mais seguras, por meio do protocolo HTTPS.

Definiu-se, inicialmente, duas permissões para os usuários: gerência de prédios e de transdutores. Cada uma dessas fornece ao usuário os direitos de incluir, modificar e habilitar/desabilitar. A gerência de usuários e definição das permissões são realizadas pelo *app users*, Figura 20.

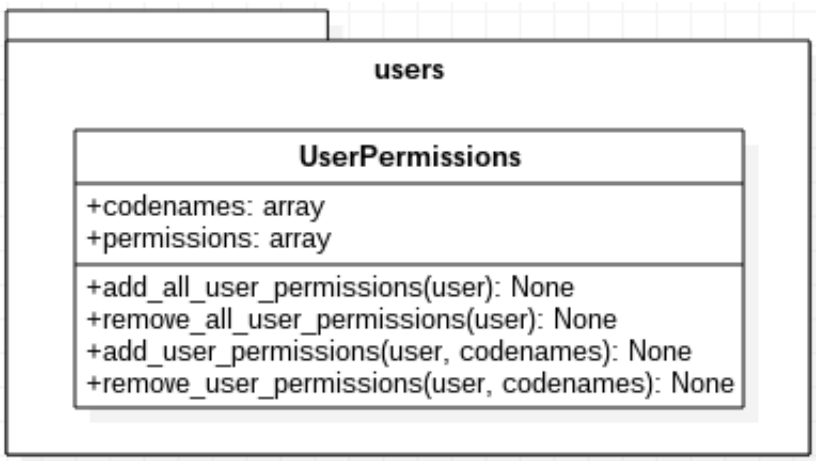

Figura 20 – Diagrama de Classes para *App users*.

4.7 Gerência de Configuração

4.7.1 Docker

Criaram-se duas configurações para os ambientes referentes ao mestre e escravo, onde suas diferenciações baseavam-se no `crontab` que seria executado em cada um. Para realizar essas configurações, utilizou-se a ferramenta Docker Compose, tornando mais fácil a definição e execução de múltiplos contêiners, com uma possibilidade de ligação entre eles. Os serviços definidos para a aplicação foram:

- nginx: fornecimento dos arquivos estáticos;
- web: fornecimento da aplicação em Django;
- postgres: armazenamento das informações;
- redis: *cache* da aplicação;
- data: armazenamento do banco de dados.

A configuração para o servidor mestre é representada pelo Algoritmo 4.5.

```
version: '2'
services:
  nginx:
```

```
    restart: always
    build: ./docker/nginx/
    ports:
      - "80:80"
    volumes:
      - ./docker/nginx/collect/static/:/var/www/static/
    volumes_from:
      - web
    links:
      - web:web

  web:
    build:
      context: .
      dockerfile: ./docker/Dockerfile.master
    expose:
      - "8000"
    links:
      - postgres:postgres
      - redis:redis
    env_file: ./docker/env
    volumes:
      - ./src/:/app/src/

  postgres:
    restart: always
    image: postgres:latest
    volumes_from:
      - data
    volumes:
      - ./docker/postgres/docker-entrypoint-initdb.d: \
        /docker-entrypoint-initdb.d
    env_file:
      - ./docker/env
    expose:
      - "5432"

  redis:
    restart: always
```

```
    image: redis:latest
    expose:
      - "6379"

  data:
    restart: always
    image: alpine
    volumes:
      - /var/lib/postgresql
    command: "true"
```

Algoritmo 4.5 – Serviços providos pelo Docker Compose.

O serviço nginx utiliza um servidor nginx[6] para lidar primeiramente com requisições externas e fornecer os arquivos estáticos da aplicação. O nginx consiste em um servidor HTTP e *proxy* reverso. Servidores *proxy* reverso utilizam um servidor *proxy* para atuar como um intermediadores entre uma requisição realizada por um cliente e o servidor que a atenderá. Um servidor *proxy* é uma entidade de rede que satisfaz solicitações HTTP em nome de um cliente, possuindo seu próprio armazenamento em disco e mantendo cópias de objetos recentemente solicitados (KUROSE; ROSS, 2002).

Requisições que precisam ser geradas dinamicamente são tratadas pelo serviço *web*, que são enviadas para um servidor Gunicorn[7]. O Gunicorn é um servidor Python *Web Server Gateway Interface* (WSGI) HTTP. O WSGI é uma especificação para uma interface simples e universal entre servidores *web* e aplicações *web* ou *frameworks* para a linguagem de programação Python (PEP 3333, 2010).

Quando o serviço *web* é iniciado, executa-se um *script*, Algoritmo 4.6, responsável por criar um arquivo contendo as variáveis de ambiente, popular o banco de dados com os *campi* da UnB e os modelos de transdutor base, inicializar o *daemon* `cron` e o servidor Gunicorn.

```
#!/bin/bash

# Creating file with environment variables to use in crontab
env | sed 's/^\(.*\)$/ \1/g' > /root/env

# Populating database and initializing cron and gunicorn
python3 manage.py makemigrations && \
python3 manage.py migrate && \
```

[6] <https://nginx.org/en>
[7] <http://gunicorn.org/>

```
python3 manage.py loaddata src/smi_unb/fixtures/initial_data.json && \
cron && \
gunicorn smi_unb.wsgi -b 0.0.0.0:8000
```

Algoritmo 4.6 – *Script start.sh*.

A criação do arquivo contendo as variáveis de ambiente é necessária, pois o `crontab` de um contêiner não possui acesso a elas. Os *crontabs* dos servidores mestres e escravo realizam uma exportação das variáveis antes de executarem seus devidos comandos. O Algoritmo 4.7 ilustra o `crontab` do servidor mestre.

```
0 * * * * export $(cat /root/env | xargs) && \
python3 /app/manage.py runcrons \
smi_unb.api.cronjob.MeasurementsSyncCronJob
# Necessary line at end of file to make cron valid
```

Algoritmo 4.7 – *Crontab* do servidor mestre.

O serviço *postgres* utiliza uma imagem banco de dados PostgreSQL para realizar o armazenamento das informações.

Para acelerar as requisições de páginas *web* do SMI-UnB, utilizou-se o serviço *redis*, responsável por executar um servidor Redis[8]. O *Cache* é um componente de *hardware* ou *software* que armazena dados, objetivando uma maior rapidez para pedidos futuros. Os dados armazenados em um *cache* podem ser o resultado de uma computação executada anteriormente ou da duplicação de dados presentes em um outro lugar (HENNESSY; PATTERSON, 2011).

O serviço *data* foi utilizado para gerenciar a persistência de dados, objetivando não arriscar qualquer exclusão acidental durante, por exemplo, uma atualização no contêiner do PostgreSQL.

Criou-se um *script*, Algoritmo 4.8, responsável por criar um arquivo contendo as variáveis de ambiente referentes ao banco de dados.

```
#!/bin/bash
echo "Collecting data to create the database..."

read -p "Enter the database name: " db_name

read -p "Enter the database user: " db_user

while true; do
```

[8] <https://redis.io/>

```
    read -s -p "Enter the database password: " db_pass
    echo
    read -s -p "Confirm the database password: " db_pass_confirm
    [ "$db_pass" = "$db_pass_confirm" ] && break
    echo -e "\nPasswords don't match. Please try again...\n"
done

file="./docker/env"

if [ -f "$file" ]
then
    rm "$file"
fi

printf "DB_NAME=$db_name\nDB_USER=$db_user\nDB_PASS=$db_pass
        \nDB_SERVICE=postgres\nDB_PORT=5432" >> "$file"

echo -e "\nDatabase configuration created succefully!"
```

Algoritmo 4.8 – *Script db_initial_data.sh*.

Algumas *tasks* foram adicionadas ao projeto, com auxílio da ferramenta Invoke[9], para facilitarem na execução de comandos longos e serviços do SMI-UnB. Por exemplo, para criar um ambiente de produção do servidor mestre seria necessário seguir os seguintes passos:

- Executar *script* para variáveis de ambiente do banco de dados;
- Montar os contêiners do SMI-UnB;
- Iniciar os contêiners.

Em vez de executar os comandos individualmente, esses são agrupados em uma *task* chamada *install_master*, Algoritmo 4.9, responsável por executá-los sequêncialmente.

```
@task
def install_master(ctx):
    print('Installing Master Docker')

    # Database configuration
    subprocess.call(['./scripts/db_initial_data.sh'])
```

[9] <http://www.pyinvoke.org/>

```
    # Building docker containers
    run('docker-compose -f docker-compose-master.yml build
         --force-rm')

    # Initializing containers
    run('docker-compose -f docker-compose-master.yml up -d')
```

Algoritmo 4.9 – *Task install_master*, presente no arquivo *tasks.py*.

Além da *task* explicada anteriormente, outras *tasks* referentes à inicialização/encerramento do Docker Compose e do ambiente de desenvolvimento também foram realizadas.

4.7.2 Integração Contínua

O script de integração contínua do projeto, Algoritmo 4.10, utiliza imagens oficiais do Python 3.5 e do PostgreSQL, presentes no Docker-Hub[10], que é repositório oficial de imagens do Docker. Todas as imagens do docker-hub já estão prontas para serem executadas em contêiners. Após os contêiners serem iniciados e vinculados, é realizada a instalação dos pacotes utilizados pelo SMI-UnB. Com a instalação dos pacotes, inicia-se verificação das normas da PEP8, com a ferramenta flake8 e por fim, são executados todos os testes do sistema e exibida a cobertura total, por meio da ferramenta Coverage[11]. O *job* da integração contínua irá falhar caso ocorra qualquer erro nos procedimentos informados.

```
image: "python:3.5"

services:
  - postgres:latest

variables:
  POSTGRES_DB: smiunbtest
  POSTGRES_USER: postgres
  POSTGRES_PASSWORD: ""

test:
    script:
        - apt-get update -qq
        - apt-get install python3-pip -y -qq
        - pip3 install -e .[dev]
```

[10] <https://hub.docker.com/>
[11] <https://pypi.python.org/pypi/coverage/>

```
- flake8 src/ --exclude migrations
- coverage run manage.py test smi_unb \
  --settings=smi_unb.settings_runner
- coverage report
```

Algoritmo 4.10 – *Script* para integração contínua do projeto.

4.8 Apresentação das Informações

O *framework* Bootstrap foi utilizado como base para se realizar o *layout* da aplicação, por possuir estruturas bem definidas e fáceis de serem utilizadas. Utilizou-se o jQuery[12] para facilitar o uso de *javascripts* na aplicação. Os ícones presentes foram provenientes do Font Awesome[13]. As três bibliotecas são disponilizados sobre uma licença livre e podem ser utilizados em qualquer projeto, contanto que haja uma referência sobre elas.

Ressalta-se que não foi realizado nenhum teste de usabilidade para a aplicação. Os princípios de usabilidade utilizados partiram de conhecimentos do próprio autor, adquiridos através de conceitos da literatura e experiências pessoais.

A página inicial do SMI-UnB, Figura 21, utiliza uma imagem gratuita do site *FreeImages*[14], disponibilizada sob sua licença[15]. O *site* e o criador, Friedrich Plechschmidt, possuem todos os direitos de propriedade intelectual sobre a imagem.

Figura 21 – Página inicial do SMI-UnB.

12 <https://jquery.com/>
13 <http://fontawesome.io/>
14 <http://pt.freeimages.com>
15 <http://pt.freeimages.com/license>

Foram definidas duas páginas principais para o SMI-UnB: painel de controle e "minha conta". O painel de controle, Figura 22, possui como intuito unir os principais recursos do sistema de uma maneira fácil e rápida, para que o usuário não perca muito tempo até encontrá-los. A página "minha conta", Figura 23, possui as opções para o gerenciamento das informações de um usuário específico, como a alteração das informações básicas e mudança de senha.

Figura 22 – Página do painel de controle do SMI-UnB.

Figura 23 – Página para gerenciamento de informações de um usuário.

Os botões da aplicação, Figuras 24 e 25, possuem cores e nomes bem significativos, auxiliando o usuário a realizar uma operação com maior segurança. Por exemplo, a cor verde para um botão indica que um registro/ativação será realizado, a cor amarela indica uma alteração e a vermelha uma desativação. Além disso, as operações de criar, atualizar, habilitar ou desabilitar necessitam de uma confirmação provinda do usuário, objetivando essas não sejam realizadas de maneira indevida.

Figura 24 – Botões presentes na página de um campus.

Figura 25 – Botões presentes na página de um transdutor.

As páginas do sistema possuem *breadcrumbs*[16], Figura 26, que mapeiam onde o usuário se encontra e quais são as páginas anteriores a essa. Assim, dificilmente o usuário irá se perder na aplicação.

Figura 26 – *Breadcrumbs* para página de um transdutor específico.

As demais imagens do SMI-UnB encontram-se no apêndice.

4.8.1 Gráfico de Linhas

Realizou-se um gráfico de linhas com auxílio do app *report*, Figura 27, para que fosse possível apresentar as informações energéticas ao usuário de maneira prática e in-

[16] Elemento de controle gráfico usado como ajuda de navegação em interfaces de usuário.

tuitiva. A biblioteca utilizada para criar o gráfico foi a *matplotlib*[17]. Utilizou-se o *plugin mpld3*[18] para apresentação e dinamicidade do gráfico.

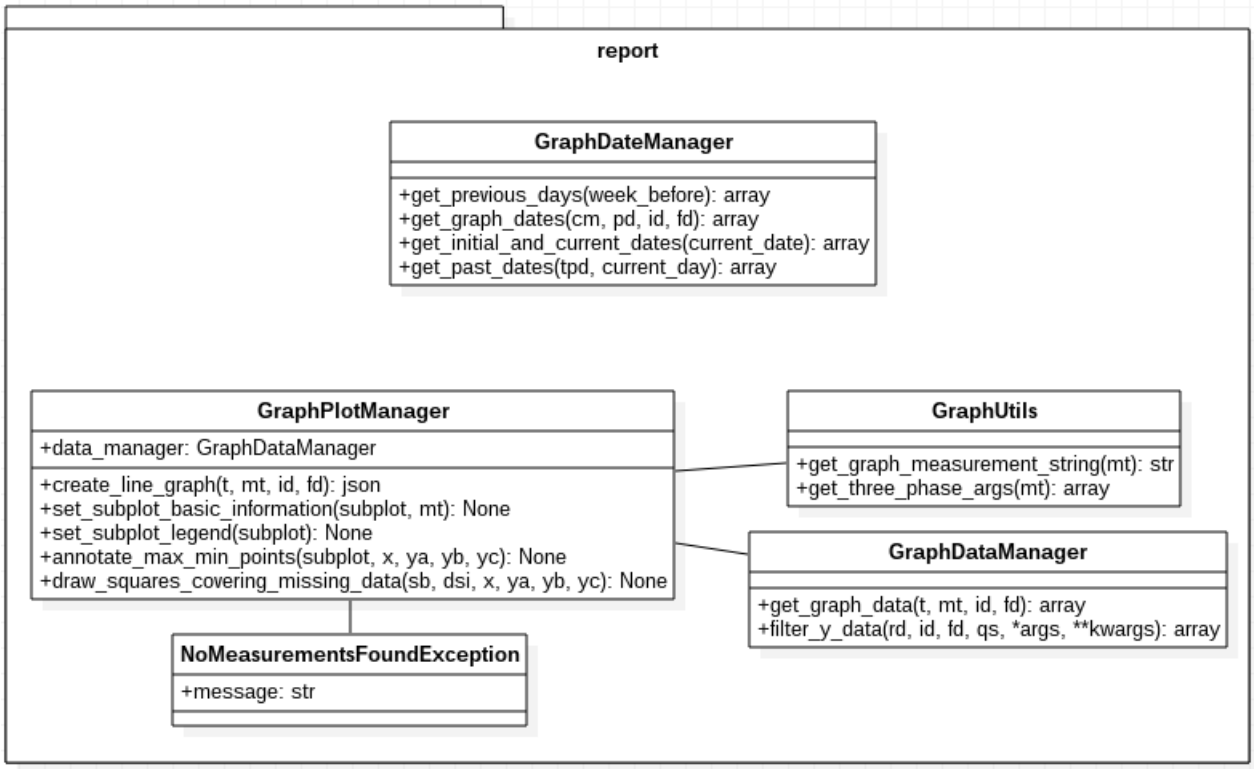

Figura 27 – Diagrama de Classes para *App report*.

Foram definidas quatro importantes classes no *app report*, sendo estas:

- *GraphDateManager*: realiza o tratamento e manipulação de datas, utilizadas para indicar um determinado período de medição do gráfico;
- *GraphPlotManager*: realiza a plotagem de todos os elementos do gráfico;
- *GraphDataManager*: realiza cálculos com os valores que serão apresentados no gráfico;
- *GraphUtils*: indica quais serão os tipos de medições apresentados no gráfico.

Foi definido que o máximo de pontos apresentados em um gráfico seria de 200, objetivando uma geração rápida e pouca espera do usuário. Para isso, definiu-se que o período máximo para apresentação de informações seria de 1 semana e suas médias seriam realizadas da seguinte maneira:

- Até 2 horas: gráfico com médias de 1 em 1 minuto;

[17] <https://matplotlib.org/>
[18] <http://mpld3.github.io/>

- Entre 2 e 6 horas: gráfico com médias de 5 em 5 minutos;
- Entre 6 e 24 horas: gráfico com médias de 10 em 10 minutos;
- Entre 1 dia e 7 dias: gráfico com médias de 1 em 1 hora.

Para se gerar um gráfico basta selecionar um transdutor, o tipo de medição e uma opção para o período de coleta, Figura 28

Figura 28 – Opções apresentadas para se gerar um gráfico de linhas.

A Figura 29 ilustra o gráfico de linhas com medições reais de tensão realizadas em um ambiente de teste no campus UnB-Gama, refentes ao dia 25/06/2017. É perceptível que durante a madrugada ocorrem os menores índices para tensão e os maiores durante o horário de ponta.

Outro exemplo, Figura 30, ilustra o gráfico de linhas gerado com medições de corrente, referentes ao dia 19/06/2017. Percebe-se nesse gráfico que houve períodos sem medições, representados pelos retângulos amarelos. Outro ponto importante a se destacar refere-se à capacidade da aplicação de continuar realizando a coleta de dados do transdutor, mesmo possuindo períodos sem realizar efetivamente uma medição.

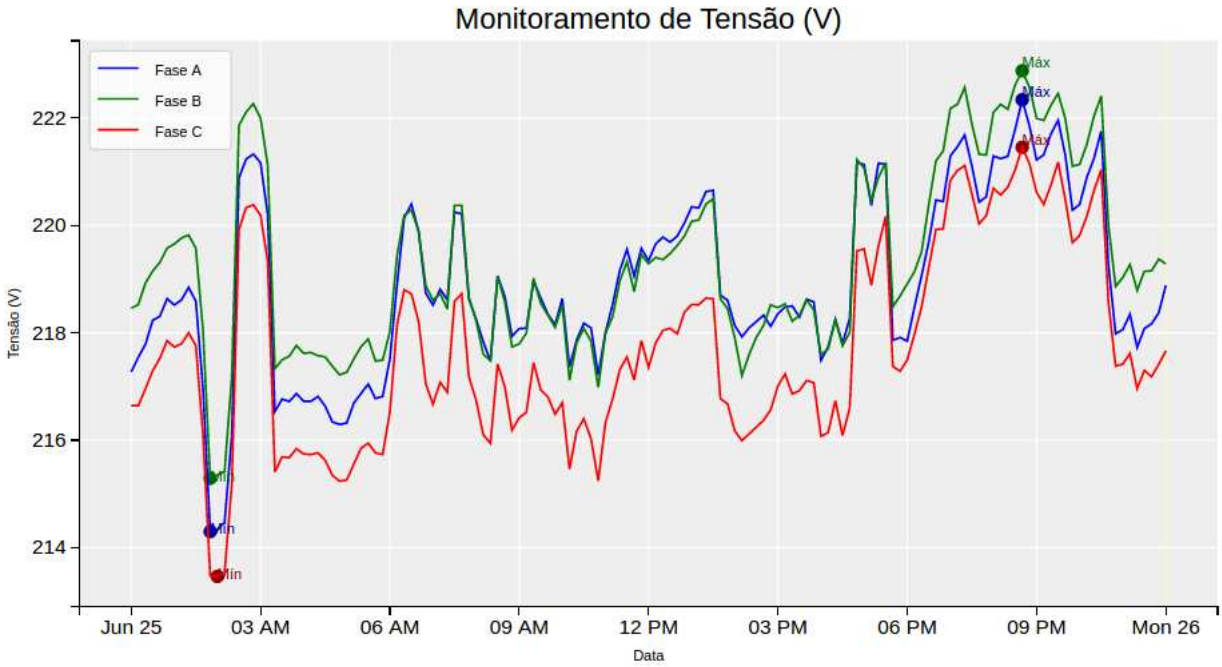

Figura 29 – Medições de tensão realizadas em ambiente de testes, referentes ao dia 25/06/2017.

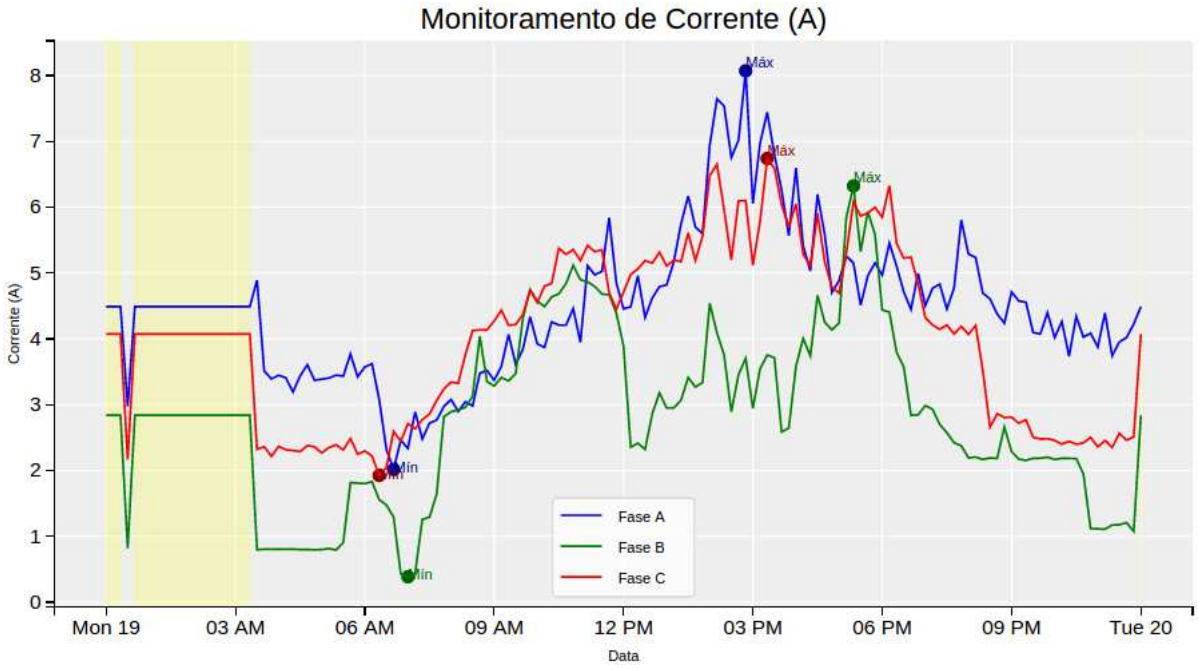

Figura 30 – Medições de corrente realizadas em ambiente de testes, referentes ao dia 19/06/2017.

4.9 Métricas

4.9.1 *SLOCCount*

O *SLOCCount*(WHEELER, 2009) realiza a contagem das linhas de código fonte físicas (SLOC) de um projeto. Ele determina automaticamente quais arquivos são de código-fonte suas linguagens de programação utilizadas. A ferramenta não realizada a contagem de códigos em *javascript* ou HTML. Com os resultados obtidos, são apresentas estimativas de esforço, tempo de desenvolvimento, quantidade de desenvolvedores e custo. A Figura 31 apresenta o resultado obtido pelo SMI-UnB.

```
Total Physical Source Lines of Code (SLOC)                = 6,521
Development Effort Estimate, Person-Years (Person-Months) = 1.43 (17.19)
 (Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months)                         = 0.61 (7.37)
 (Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule)  = 2.33
Total Estimated Cost to Develop                           = $ 193,496
 (average salary = $56,286/year, overhead = 2.40).
SLOCCount, Copyright (C) 2001-2004 David A. Wheeler
SLOCCount is Open Source Software/Free Software, licensed under the GNU GPL.
SLOCCount comes with ABSOLUTELY NO WARRANTY, and you are welcome to
redistribute it under certain conditions as specified by the GNU GPL license;
see the documentation for details.
Please credit this data as "generated using David A. Wheeler's 'SLOCCount'."
```

Figura 31 – Análise do SMI-UnB utilizando *SLOCCount*.

Após realizada uma análise com a ferramenta, foi possível observar que o projeto teria um custo de 193,496 dólares e seria realizado em um período de aproximadamente 7 meses, com o apoio de 2 desenvolvedores. Além disso, a estimativa de tempo se aproximou ao período de 5.7 meses, refente às duas *releases* do projeto.

4.10 Requisitos para Utilizar o SMI-UnB

Para utilizar o SMI-UnB é necessário possuir, em cada servidor, uma configuração que possa executar o sistema operacional debian, sendo esta:

- Processador de 1GHz *Pentium*;
- Memória RAM de 512MB;

Como o projeto utiliza imagens do docker, Figura 32, é necessário ter um espaço em disco de 2.6 GB, além de um espaço adicional para armazenar os registros realizados na aplicação.

```
root@debian:~/SMI-UnB# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
smiunb_web          latest              c341d35f4185        7 days ago          1.13 GB
smiunb_nginx        latest              b9910dc33304        12 days ago         212 MB
python              3.5                 9613c8434d7b        2 weeks ago         684 MB
redis               latest              83744227b191        3 weeks ago         98.9 MB
postgres            latest              02d6fa85db71        3 weeks ago         269 MB
alpine              latest              a41a7446062d        5 weeks ago         3.97 MB
tutum/nginx         latest              a2e9b71ed366        15 months ago       206 MB
```

Figura 32 – Tamanho dos contêiners do Docker.

É necessária uma conexão à *internet* para que seja possível realizar uma coleta de dados (protocolo UDP), sincronia de entidades (protocolo HTTP) ou prover páginas *web* da aplicação.

5 Conclusão

O trabalho atingiu os objetivos definidos para criação de uma aplicação básica para monitoramento energético na Universidade de Brasília. A aplicação está em um estado funcional, apesar de não ter sido implantada de forma definitiva. Existe uma instância em homologação, sendo executada na infraestrutura do Laboratório Avançado de Pesquisa e Produção de *Software*[1] (LAPPIS), presente na Faculdade UnB Gama.

Foi realizada uma apresentação do sistema para os professores de Engenharia de Energia, citados anteriormente no trabalho. Boas críticas foram obtidas para o projeto em si, com ênfase nas medições realizadas e como essas foram apresentadas. Além disso, os professores realizaram comentários para futuras propostas de projetos de eficiência energética em outros órgãos públicos que pudessem utilizar como base o SMI-UnB.

5.1 Trabalhos Futuros

Melhoramentos futuros estão abertos para serem realizados sobre o trabalho, estes são:

- Aperfeiçoar *layout da aplicação*
- Aprimorar segurança e sincronização da API;
- Automatizar ainda mais as tasks para o Docker;
- Criar receitas para a aplicação;
- Monitoramento de outros recursos (por exemplo, água)
- Realizar testes funcionais e de integração;
- Realizar testes de usabilidade com usuários finais;
- Realizar sistema de *log* que informe as ações realizadas pelos usuários;
- Realizar um sistema de *backup*;
- Realizar a funcionalidade para esquecimento de senhas, via e-mail;
- Separar os transdutores numa VLAN segregada para evitar interferências em suas operações;

[1] <http://lappis.rocks/>

- Subir a imagem do serviço *web* no Docker Hub;
- Otimizar as imagens do Docker para ocuparem menos espaço.

5.2 Cronograma

Para a realização do cronograma do projeto, Figuras 33 e 34, utilizou-se a ferramenta *Gantter*. Os cronogramas abordam como se dividiram as sprints e atividades realizadas no decorrer do desenvolvimento do SMI-UnB.

Nome	Duração	Início	Fim	Predecessores
Release 1	**93d?**	**20/07/2016**	**25/11/2016**	
Sprint 0	**10d?**	**20/07/2016**	**02/08/2016**	
Entender contexto	3d?	20/07/2016	22/07/2016	
Identificar requisitos do projeto	7d?	25/07/2016	02/08/2016	
Sprint 1	**10d?**	**08/08/2016**	**19/08/2016**	**2**
Realizar coleta inicial de dados do transdutor de energia	10d?	08/08/2016	19/08/2016	
Realizar páginas para apresentação das leituras realizad	10d?	08/08/2016	19/08/2016	
Sprint 2	**10d?**	**22/08/2016**	**02/09/2016**	**5**
Refatorar urls e templates	10d?	22/08/2016	02/09/2016	
Adicionar guia de instalação do projeto (Vagrant e Chef S	10d?	22/08/2016	02/09/2016	
Sprint 3	**10d?**	**05/09/2016**	**16/09/2016**	**8**
Refatorar arquitetura	10d?	05/09/2016	16/09/2016	
Realizar testes para model de transdutor	10d?	05/09/2016	16/09/2016	
Criar modelos de transdutores	10d?	05/09/2016	16/09/2016	
Sprint 4	**10d?**	**19/09/2016**	**30/09/2016**	**11**
Documentação do código utilizando Google Python Style	10d?	19/09/2016	30/09/2016	
Adicionar estrutura Boilerplate ao projeto	10d?	19/09/2016	30/09/2016	
Testes excluir e editar Transdutor	10d?	19/09/2016	30/09/2016	
Trocar classe abstrata de Measurements por Polymorphi	10d?	19/09/2016	30/09/2016	
Sprint 5	**10d?**	**03/10/2016**	**14/10/2016**	**15**
Refatoração arquitetura	10d?	03/10/2016	14/10/2016	
Implementação/teste protocolo serial	10d?	03/10/2016	14/10/2016	
Criar PolymorphicManager para EnergyMeasurements	10d?	03/10/2016	14/10/2016	
Sprint 6	**10d?**	**17/10/2016**	**28/10/2016**	**20**
Implementação/teste protocolo de transporte	10d?	17/10/2016	28/10/2016	
Utilizar Mock para testes	10d?	17/10/2016	28/10/2016	
Sprint 7	**10d?**	**31/10/2016**	**11/11/2016**	**24**
Implementação/teste Data collector	10d?	31/10/2016	11/11/2016	
Sprint 8	**10d?**	**14/11/2016**	**25/11/2016**	**27**
Adicionar integração contínua ao projeto	10d?	14/11/2016	25/11/2016	
Realizar coleta temporizada dos dados (django_cron)	10d?	14/11/2016	25/11/2016	

Figura 33 – Cronograma da *release* 1.

Nome	Duração	Ínicio	Fim	Predecessores
Release 2	78d?	**06/03/2017**	**21/06/2017**	**1**
Sprint 1	10d?	**06/03/2017**	**17/03/2017**	
Refatorar código de GPP/MDS	10d	06/03/2017	17/03/2017	
Corrigir ambientação e testes	10d?	06/03/2017	17/03/2017	
Mudar projeto para python3	10d?	06/03/2017	17/03/2017	
Sprint 2	10d?	**20/03/2017**	**31/03/2017**	**33**
Corrigir sistema de autenticação	10d?	20/03/2017	31/03/2017	
Corrigir sistema de usuários	10d?	20/03/2017	31/03/2017	
Sprint 3	10d?	**03/04/2017**	**14/04/2017**	**37**
Melhorar layout ao sistema	10d?	03/04/2017	14/04/2017	
Sprint 4	10d?	**17/04/2017**	**28/04/2017**	**40**
Realizar gráfico de linhas para medições	8d?	17/04/2017	26/04/2017	
Adicionar pontos de mínimo e máximo ao gráfico	2d?	27/04/2017	28/04/2017	
Sprint 5	10d?	**01/05/2017**	**12/05/2017**	**42**
Acrescentar opções adicionais para gerar gráfico de linhas	5d?	01/05/2017	05/05/2017	
Realizar testes do gráfico	5d?	08/05/2017	12/05/2017	
Sprint 6	10d?	**15/05/2017**	**26/05/2017**	**45**
Mapear campus e prédios na aplicação	7d?	15/05/2017	23/05/2017	
Adicionar permissões aos usuários	3d?	24/05/2017	26/05/2017	
Sprint 7	10d?	**29/05/2017**	**09/06/2017**	**48**
Adicionar Docker ao projeto	10d?	29/05/2017	09/06/2017	
Criar api REST	10d?	29/05/2017	09/06/2017	
Sprint 8	8d?	**12/06/2017**	**21/06/2017**	**51**
Finalizar api para transmissão de medições	5d?	12/06/2017	16/06/2017	
Arrumar cron do docker	3d?	19/06/2017	21/06/2017	

Figura 34 – Cronograma da *release* 2.

Referências

AES Eletropaulo. *Entenda o aumento na conta de energia*. 2015. Disponível em: <https://www.aeseletropaulo.com.br/para-sua-casa/prazos-e-tarifas/conteudo/entenda-o-aumento-na-conta-de-energia-(mar%C3%A7o2015)>. Citado na página 17.

ANEEL. *Tarifa Branca*. 1996. Disponível em: <http://www.aneel.gov.br/tarifa-branca>. Citado na página 30.

BECK, K.; ANDRES, C. *Extreme Programming Explained: Embrace Change (2Nd Edition)*. [S.l.]: Addison-Wesley Professional, 2004. ISBN 0321278658. Citado 3 vezes nas páginas 20, 21 e 23.

BECK, K. et al. *The Agile Manifesto*. [S.l.], 2001. Citado 2 vezes nas páginas 19 e 29.

BENSLIMANE, D.; DUSTDAR, S.; SHETH, A. P. Services mashups: The new generation of web applications. *IEEE Internet Computing*, v. 12, n. 5, p. 13–15, 2008. Disponível em: <http://dblp.uni-trier.de/db/journals/internet/internet12.html#BenslimaneDS08>. Citado na página 50.

BOURQUE, P.; FAIRLEY, R. E. (Ed.). *SWEBOK: Guide to the Software Engineering Body of Knowledge*. Version 3.0. Los Alamitos, CA: IEEE Computer Society, 2014. ISBN 978-0-7695-5166-1. Disponível em: <http://www.swebok.org/>. Citado na página 24.

CHRISTIE, T. 2011. Disponível em: <http://www.django-rest-framework.org>. Citado na página 51.

CMMI Product Team. *CMMI® for Development, Version 1.3*. [S.l.], 2010. Disponível em: <http://www.sei.cmu.edu/library/abstracts/reports/10tr033.cfm>. Citado na página 24.

Django Software Foundation. *Django Project*. 2005. Disponível em: <https://www.djangoproject.com/>. Citado 2 vezes nas páginas 39 e 44.

DMITRIY, Z.; VALERY, S. *Gitlab*. 2013. Disponível em: <https://about.gitlab.com/>. Citado na página 31.

DOCKER, I. *What is Docker*. 2017. Disponível em: <https://www.docker.com/what-docker>. Citado 2 vezes nas páginas 9 e 37.

EMBRASUL. 2017. Disponível em: <http://www.embrasul.com.br>. Citado 3 vezes nas páginas 9, 40 e 41.

FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. Tese (Doutorado), 2000. AAI9980887. Citado na página 51.

HENNESSY, J. L.; PATTERSON, D. A. *Computer Architecture, Fifth Edition: A Quantitative Approach*. 5th. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011. ISBN 012383872X, 9780123838728. Citado na página 57.

IEEE Task P754. *IEEE standard for binary floating-point arithmetic*. New York: Institute of Electrical and Electronics Engineers, 1985. Citado na página 42.

ISO/IEC. *ISO/IEC 9126. Software engineering – Product quality*. [S.l.]: ISO/IEC, 2001. Citado 2 vezes nas páginas 25 e 26.

ISO/IEC/IEEE. *ISO/IEC/IEEE 24765 - Systems and software engineering - Vocabulary*. [S.l.], 2010. Citado na página 24.

KUROSE, J. F.; ROSS, K. *Computer Networking: A Top-Down Approach Featuring the Internet*. 2nd. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0201976994. Citado 3 vezes nas páginas 44, 51 e 56.

LEITE, J. C. S. do P.; SANT'ANNA, M.; FREITAS, F. G. de. *Third International Conference on Software Reuse*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994. Citado na página 23.

LOUKIDES, M. *What is DevOps?* 2012. Disponível em: <http://radar.oreilly.com/2012/06/what-is-devops.html>. Citado na página 36.

LUO, L. Software testing techniques. *Institute for software research international Carnegie mellon university Pittsburgh, PA*, v. 15232, n. 1-19, p. 19, 2001. Citado 2 vezes nas páginas 23 e 24.

MALDONADO, J. C. et al. Introduçao ao teste de software. *São Carlos*, 2004. Citado na página 24.

MEIRELLES, P. R. M. *Monitoramento de métricas de código-fonte em projetos de software livre*. Tese (Doutorado) — Universidade de São Paulo, 2013. Citado na página 25.

Ministério de Minas e Energia. *Balanço Energético Nacional*. 2015. Disponível em: <https://ben.epe.gov.br/downloads/Relatorio_Final_BEN_2015.pdf>. Citado na página 17.

MODICON, Inc. *Modicon MODBUS Protocol Reference Guide*. 1996. Disponível em: <http://modbus.org/docs/PI_MBUS_300.pdf>. Citado 2 vezes nas páginas 9 e 41.

NETO, A.; DIAS, C. Introdução a teste de software. *Engenharia de Software Magazine*, v. 1, 2007. Citado 2 vezes nas páginas 23 e 24.

New Media Rights. *Open Source Licensing Guide*. 2008. Disponível em: <http://www.newmediarights.org/open_source/new_media_rights_open_source_licensing_guide>. Citado na página 31.

NIELSEN, J. *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993. ISBN 9780080520292. Citado na página 26.

PEP 3333. *Python Web Server Gateway Interface v1.0.1*. 2010. Disponível em: <https://www.python.org/dev/peps/pep-3333/>. Citado na página 56.

PRESSMAN, R. *Software Engineering: A Practitioner's Approach*. 7. ed. New York, NY, USA: McGraw-Hill, Inc., 2010. ISBN 0073375977, 9780073375977. Citado 3 vezes nas páginas 20, 23 e 26.

PROFFITT, B. *What APIs Are And Why They're Important*. 2013. Disponível em: <http://readwrite.com/2013/09/19/api-defined>. Citado na página 50.

RADIGAN, D. *A brief introduction to kanban*. 2015. Disponível em: <https://www.atlassian.com/agile/kanban>. Citado 3 vezes nas páginas 9, 22 e 29.

RAYMOND, E. The cathedral and the bazaar. *Knowledge, Technology & Policy*, Springer, v. 12, n. 3, p. 23–49, 1999. Citado 2 vezes nas páginas 25 e 30.

ROGERS, Y.; SHARP, H.; PREECE, J. *Design de interação: além da interação humano-computador*. 3nd. ed. Porto Alegre: [s.n.], 2013. Citado na página 25.

SCHWABER, K.; SUTHERLAND, J. *The Scrum Guide*. 2001. Citado 2 vezes nas páginas 29 e 34.

SOMMERVILLE, I. *Software Engineering: (Update) (8th Edition) (International Computer Science)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2006. ISBN 0321313798. Citado 3 vezes nas páginas 19, 20 e 22.

STEVENSON, W. *Elements of Power System Analysis*. McGraw-Hill, 1962. (International student edition). Disponível em: <https://books.google.com.br/books?id=cOdSAAAAMAAJ>. Citado na página 30.

TANENBAUM, A. *Computer Networks*. 4th. ed. [S.l.]: Prentice Hall Professional Technical Reference, 2002. ISBN 0130661023. Citado 2 vezes nas páginas 9 e 43.

TANENBAUM, A. S. *Modern Operating Systems*. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007. ISBN 9780136006633. Citado 2 vezes nas páginas 36 e 48.

TANENBAUM, A. S.; GOODMAN, J. R. *Structured Computer Organization*. 4th. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998. ISBN 0130959901. Citado na página 43.

THAYER, R. H.; BAILIN, S. C.; DORFMAN, M. *Software Requirements Engineerings, 2Nd Edition*. 2nd. ed. Los Alamitos, CA, USA: IEEE Computer Society Press, 1997. ISBN 0818677384. Citado na página 23.

VIXIE, P. 1987. Disponível em: <https://linux.die.net/man/8/cron>. Citado na página 49.

WHEELER, D. *SLOCcount*. 2009. Disponível em: <http://www.dwheeler.com/sloccount/>. Citado na página 66.

Apêndices

APÊNDICE A – Integração Contínua

```
Running with gitlab-ci-multi-runner 9.3.0-rc.2 (110d530)
  on docker-auto-scale (e11ae361)
Using Docker executor with image python:3.5 ...
Starting service postgres:latest ...
Pulling docker image postgres:latest ...
Using docker image postgres:latest for postgres service...
Waiting for services to be up and running...
Using docker image sha256 for predefined container...
Pulling docker image python:3.5 ...
Using docker image python:3.5 for build container...
Running on runner-e11ae361-project-1216906...
Cloning repository...
Cloning into '/builds/brenddongontijo/SMI-UnB'...
Checking out e70a256f as master...
Skipping Git submodules setup
$ apt-get update -qq
$ apt-get install python3-pip -y -qq
# Installing packages...
$ flake8 src/ --exclude migrations
$ coverage run manage.py test \
smi_unb --settings=smi_unb.settings_runner
# Running tests...
----------------------------------------------------------------------
Ran 192 tests in 17.081s
OK
Creating a new SECRET_KEY at security/secret_key.dat
Running server in DEBUG mode. Plese do *not* go to production!
Documentation files not found: disabling tests!
Creating test database for alias 'default'...
Destroying test database for alias 'default'...
$ coverage report
# Coverage report...
Job succeeded
```

Algoritmo A.1 – *Log* de integração contínua do SMI-UnB

APÊNDICE B – Cobertura Total de Código

Module	*statements*	*missing*	*excluded*	*branches*	*partial*	*coverage*
src/smi_unb/__init__	0	0	0	0	0	100%
src/smi_unb/api/__init__	0	0	0	0	0	100%
src/smi_unb/api/models	0	0	0	0	0	100%
src/smi_unb/api/serializers	23	4	0	0	0	83%
src/smi_unb/api/synchronization	84	59	0	18	0	25%
src/smi_unb/api/views	42	24	0	8	0	36%
src/smi_unb/authentication/__init__	0	0	0	0	0	100%
src/smi_unb/authentication/auth	18	0	0	3	0	100%
src/smi_unb/authentication/forms	22	0	0	4	0	100%
src/smi_unb/authentication/migrations/__init__	0	0	0	0	0	100%
src/smi_unb/authentication/tests/__init__	0	0	0	0	0	100%
src/smi_unb/authentication/tests/test_auth	31	0	0	0	0	100%
src/smi_unb/authentication/tests/test_forms	27	0	0	0	0	100%
src/smi_unb/authentication/tests/test_views	65	0	0	0	0	100%
src/smi_unb/authentication/urls	4	0	0	0	0	100%
src/smi_unb/authentication/views	30	0	0	12	0	100%
src/smi_unb/buildings/__init__	0	0	0	0	0	100%
src/smi_unb/buildings/admin	0	0	0	0	0	100%
src/smi_unb/buildings/forms	66	4	0	14	2	92%
src/smi_unb/buildings/migrations/0001_initial	8	0	0	0	0	100%
src/smi_unb/buildings/migrations/0002_building_synchronized	5	0	0	0	0	100%
src/smi_unb/buildings/migrations/__init__	0	0	0	0	0	100%
src/smi_unb/buildings/models	42	7	0	2	0	80%
src/smi_unb/buildings/tests/__init__	0	0	0	0	0	100%
src/smi_unb/buildings/tests/test_forms	65	1	0	0	0	98%
src/smi_unb/buildings/tests/test_models	41	0	0	0	0	100%
src/smi_unb/buildings/tests/test_views	246	1	0	0	0	99%
src/smi_unb/buildings/urls	4	0	0	0	0	100%
src/smi_unb/buildings/views	79	0	0	32	0	100%
src/smi_unb/campuses/__init__	0	0	0	0	0	100%
src/smi_unb/campuses/migrations/0001_initial	7	0	0	0	0	100%
src/smi_unb/campuses/migrations/__init__	0	0	0	0	0	100%
src/smi_unb/campuses/models	18	0	0	0	0	100%
src/smi_unb/campuses/tests/__init__	0	0	0	0	0	100%
src/smi_unb/campuses/tests/test_views	56	0	0	0	0	100%
src/smi_unb/campuses/urls	4	0	0	0	0	100%
src/smi_unb/campuses/views	19	0	0	2	0	100%
src/smi_unb/data_reader/__init__	0	0	0	0	0	100%
src/smi_unb/data_reader/exceptions	12	0	0	0	0	100%
src/smi_unb/data_reader/migrations/__init__	0	0	0	0	0	100%
src/smi_unb/data_reader/models	0	0	0	0	0	100%
src/smi_unb/data_reader/tests/__init__	0	0	0	0	0	100%
src/smi_unb/data_reader/tests/test_utils	179	0	0	0	0	100%
src/smi_unb/data_reader/utils	164	0	0	47	4	98%
src/smi_unb/my_account/__init__	0	0	0	0	0	100%
src/smi_unb/my_account/forms	13	0	0	0	0	100%

Figura 35 – Primeira parte da cobertura do SMI-UnB.

src/smi_unb/my_account/migrations/__init__	0	0	0	0	0	100%
src/smi_unb/my_account/models	0	0	0	0	0	100%
src/smi_unb/my_account/tests/__init__	0	0	0	0	0	100%
src/smi_unb/my_account/tests/test_forms	16	0	0	0	0	100%
src/smi_unb/my_account/tests/test_views	70	0	0	0	0	100%
src/smi_unb/my_account/urls	4	0	0	0	0	100%
src/smi_unb/my_account/views	32	0	0	10	0	100%
src/smi_unb/report/__init__	0	0	0	0	0	100%
src/smi_unb/report/exceptions	4	0	0	0	0	100%
src/smi_unb/report/forms	46	0	0	20	0	100%
src/smi_unb/report/models	0	0	0	0	0	100%
src/smi_unb/report/tests/__init__	0	0	0	0	0	100%
src/smi_unb/report/tests/test_forms	81	0	0	0	0	100%
src/smi_unb/report/tests/test_utils	213	0	0	2	0	100%
src/smi_unb/report/tests/test_views	78	0	0	2	0	100%
src/smi_unb/report/urls	4	0	0	0	0	100%
src/smi_unb/report/utils	166	0	0	52	0	100%
src/smi_unb/report/views	28	0	0	4	0	100%
src/smi_unb/retrieve_password/__init__	0	0	0	0	0	100%
src/smi_unb/retrieve_password/migrations/__init__	0	0	0	0	0	100%
src/smi_unb/retrieve_password/models	0	0	0	0	0	100%
src/smi_unb/retrieve_password/tests	18	0	0	0	0	100%
src/smi_unb/retrieve_password/urls	4	0	0	0	0	100%
src/smi_unb/retrieve_password/views	65	54	0	10	0	15%
src/smi_unb/settings	72	15	0	12	4	73%
src/smi_unb/tests/__init__	0	0	0	0	0	100%
src/smi_unb/tests/test_documentation	31	8	0	6	0	73%
src/smi_unb/tests/test_smi_unb	4	2	0	0	0	50%
src/smi_unb/tests/test_views	18	0	0	0	0	100%
src/smi_unb/transductor/__init__	0	0	0	0	0	100%
src/smi_unb/transductor/forms	91	19	0	24	9	76%
src/smi_unb/transductor/migrations/0001_initial	10	0	0	0	0	100%
src/smi_unb/transductor/migrations/0002_transductor_last_measurements_sent	5	0	0	0	0	100%
src/smi_unb/transductor/migrations/0003_auto_20170607_0122	5	0	0	0	0	100%
src/smi_unb/transductor/migrations/0004_transductor_synchronized	5	0	0	0	0	100%
src/smi_unb/transductor/migrations/__init__	0	0	0	0	0	100%
src/smi_unb/transductor/models	96	25	0	2	0	72%
src/smi_unb/transductor/tests/__init__	0	0	0	0	0	100%
src/smi_unb/transductor/tests/test_forms	48	1	0	0	0	98%
src/smi_unb/transductor/tests/test_models	82	0	0	2	0	100%
src/smi_unb/transductor/tests/test_views	221	1	0	0	0	99%
src/smi_unb/transductor/urls	4	0	0	0	0	100%
src/smi_unb/transductor/views	62	0	0	28	0	100%
src/smi_unb/urls	10	0	0	0	0	100%
src/smi_unb/users/__init__	0	0	0	0	0	100%
src/smi_unb/users/forms	58	0	0	10	0	100%
src/smi_unb/users/migrations/0001_initial	5	0	0	0	0	100%
src/smi_unb/users/migrations/0002_auto_20170522_2136	5	0	0	0	0	100%
src/smi_unb/users/migrations/0003_auto_20170524_1837	5	0	0	0	0	100%
src/smi_unb/users/migrations/__init__	0	0	0	0	0	100%
src/smi_unb/users/models	24	0	0	8	0	100%
src/smi_unb/users/tests/__init__	0	0	0	0	0	100%
src/smi_unb/users/tests/test_forms	39	0	0	0	0	100%
src/smi_unb/users/tests/test_models	65	0	0	2	0	100%
src/smi_unb/users/tests/test_views	170	0	0	10	0	100%
src/smi_unb/users/urls	4	0	0	0	0	100%
src/smi_unb/users/views	80	0	0	36	1	99%
src/smi_unb/views	6	0	0	0	0	100%
Total	**3428**	**225**	**0**	**382**	**20**	**92%**

Figura 36 – Segunda parte da cobertura do SMI-UnB.

APÊNDICE C – Imagens da Aplicação

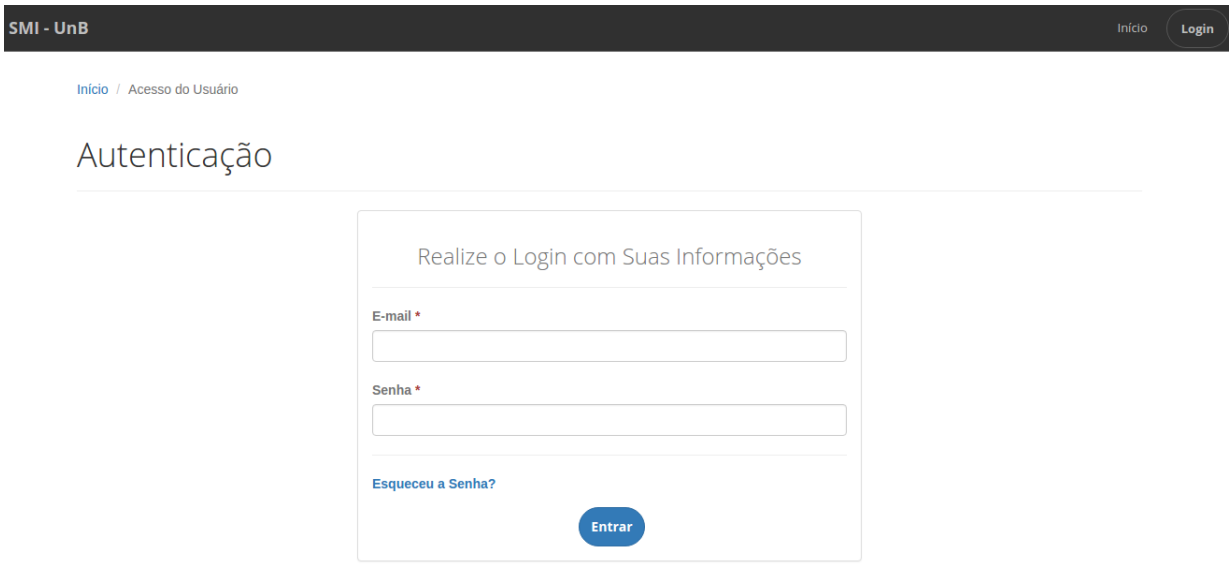

Figura 37 – Página de autenticação.

Figura 38 – Página de usuários.

Figura 39 – Página de edição das informações básicas da conta.

Figura 40 – Página de alteração de senha.

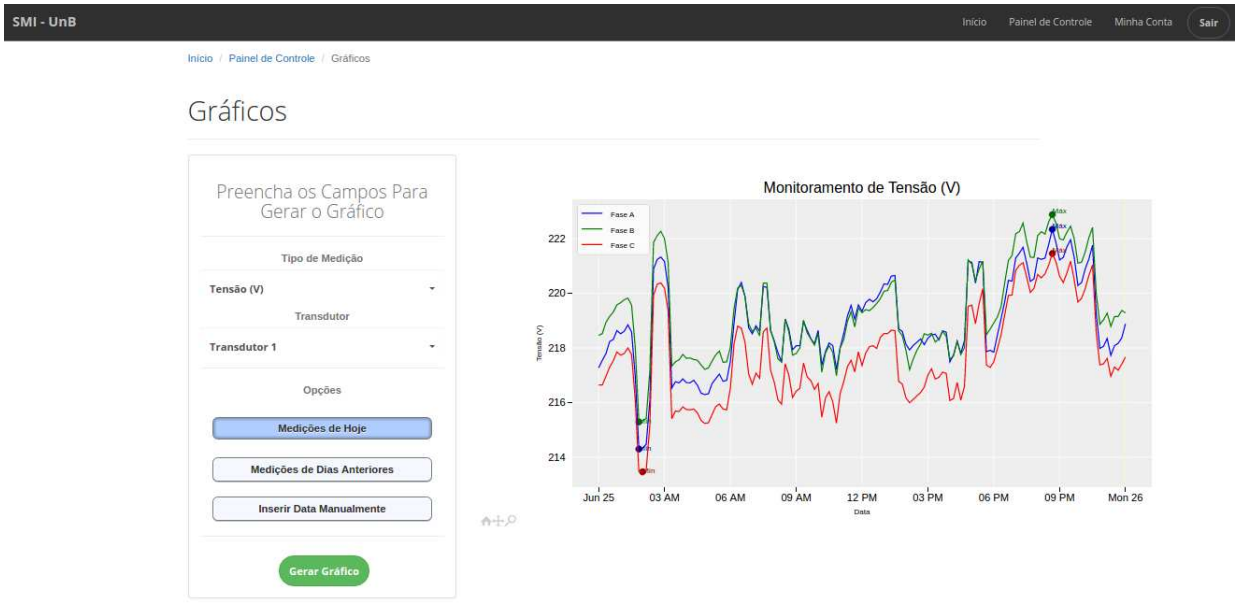

Figura 41 – Página de gráficos.

Figura 42 – Página dos *campi* da UnB.

Figura 43 – Página de edifícios desativados em um campus.

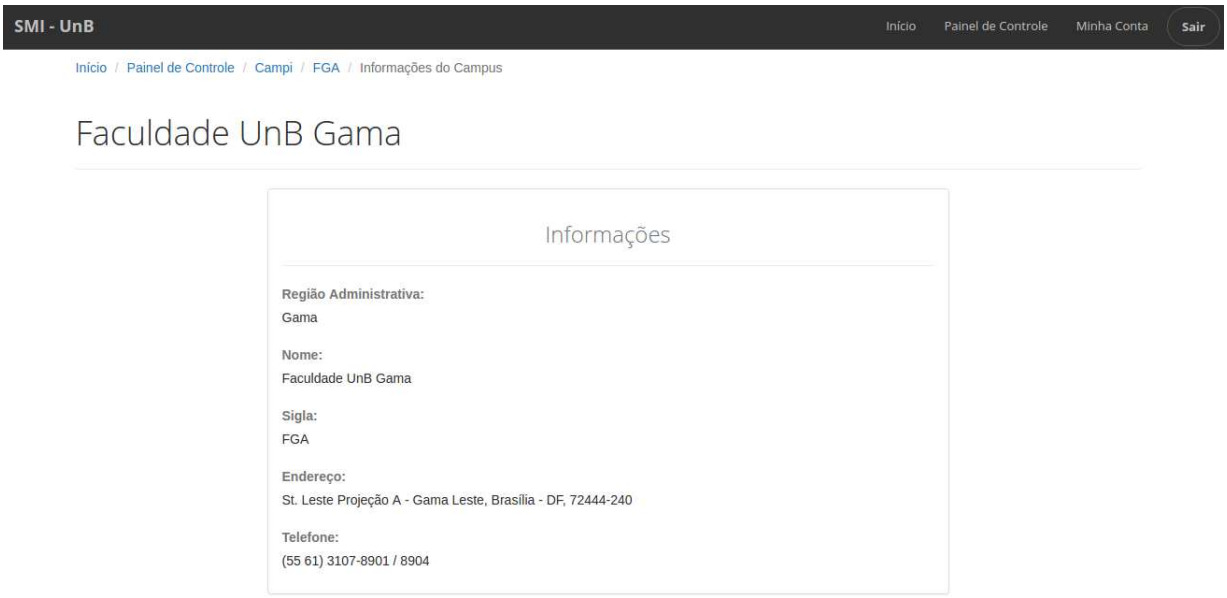

Figura 44 – Página de informações de em um campus.

Figura 45 – Página principal de um edifício.

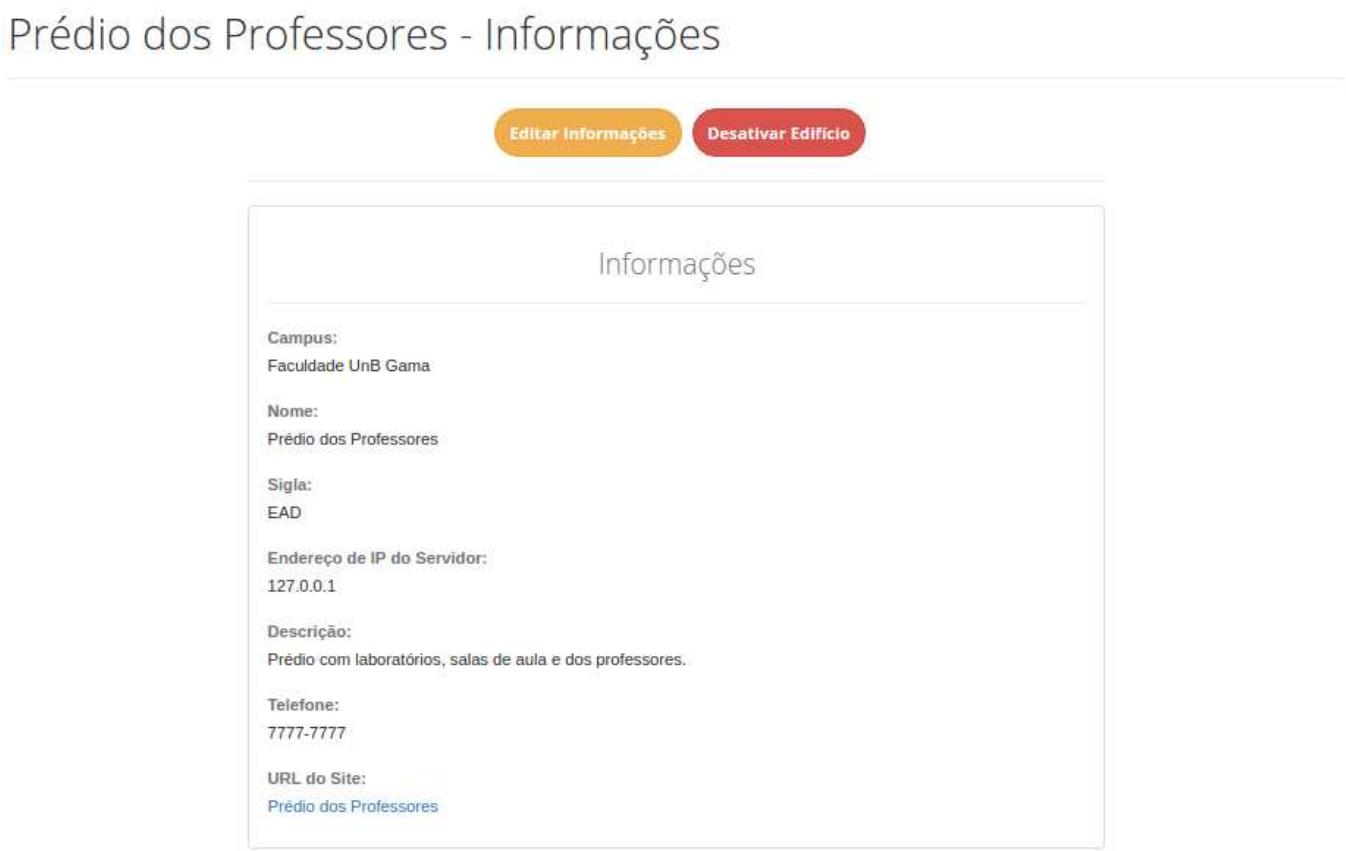

Figura 46 – Informações de um edifício.

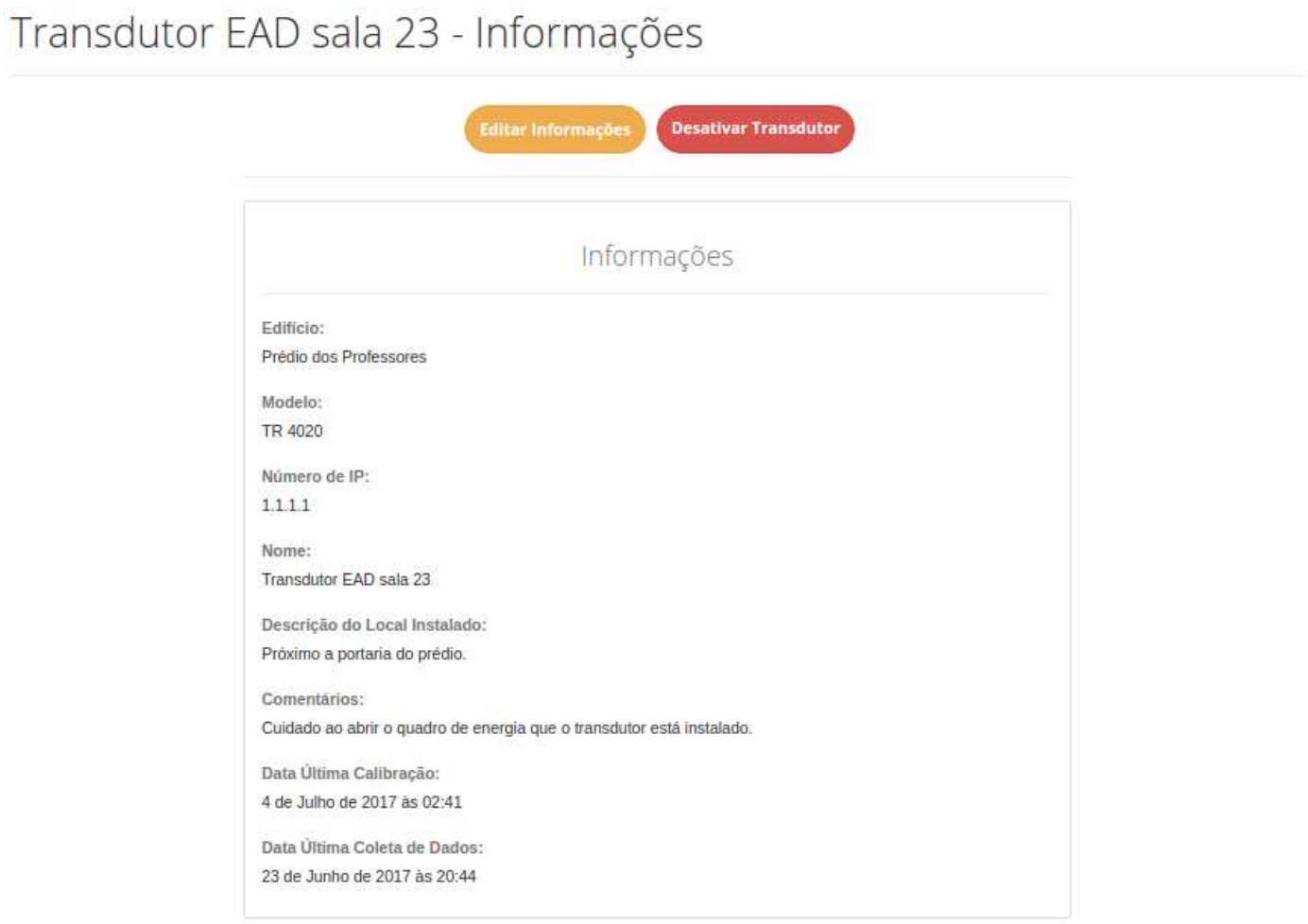

Figura 47 – Informações de um transdutor.

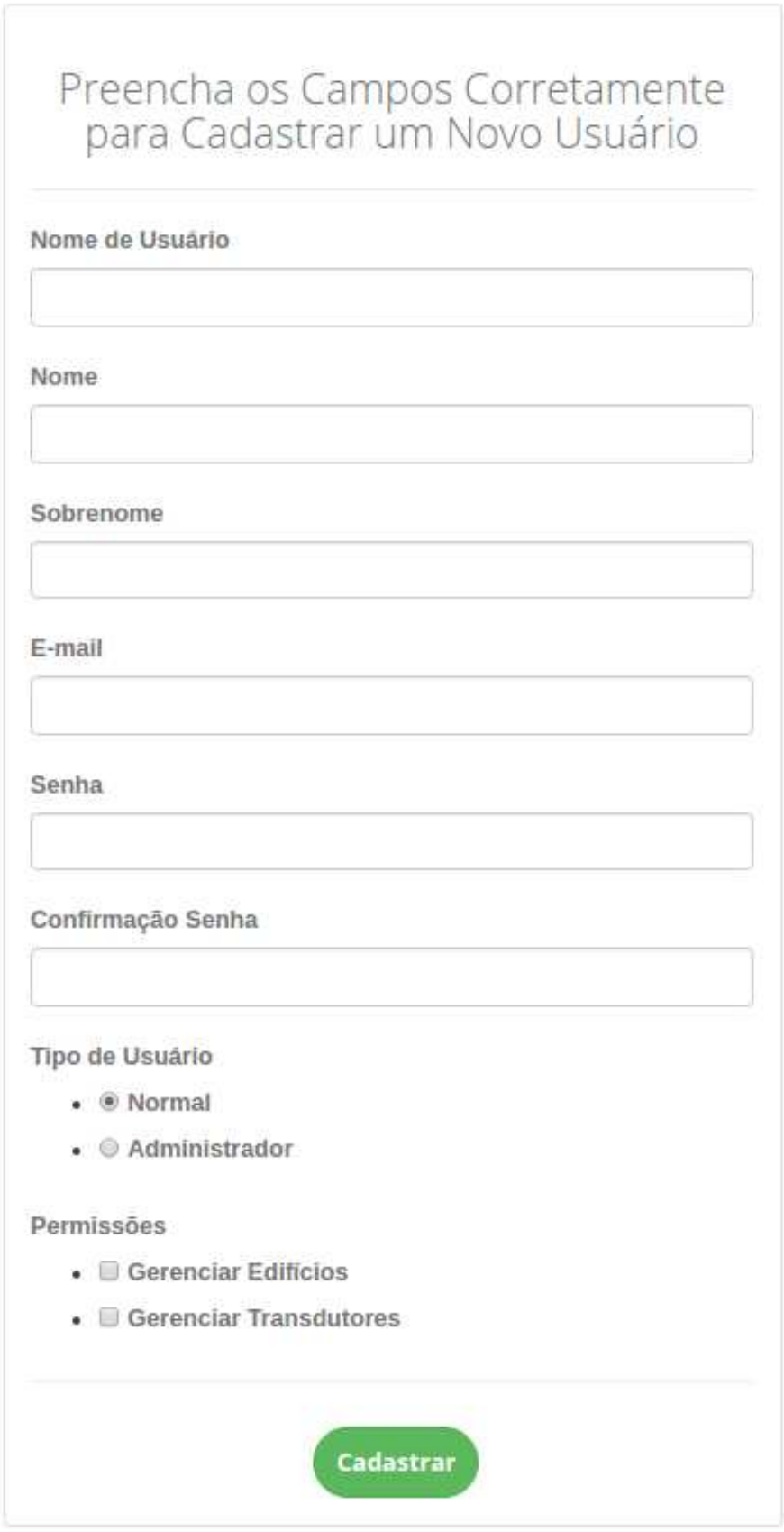

Figura 48 – Formulário para cadastro de um usuário.

Figura 49 – Formulário para cadastro de um edifício.

Figura 50 – Formulário para cadastro de um transdutor.