UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

UM SISTEMA *WEB* PARA EXECUÇÃO REMOTA DE APLICAÇÕES DE ALTO DESEMPENHO

TRABALHO DE GRADUAÇÃO

Otávio Migliavacca Madalosso

Santa Maria, RS, Brasil

2015

UM SISTEMA *WEB* PARA EXECUÇÃO REMOTA DE APLICAÇÕES DE ALTO DESEMPENHO

Otávio Migliavacca Madalosso

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de

Bacharel em Ciência da Computação

Orientadora: Profª. Drª. Andrea Schwertner Charão

Santa Maria, RS, Brasil

2015

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

UM SISTEMA *WEB* PARA EXECUÇÃO REMOTA DE APLICAÇÕES DE ALTO DESEMPENHO

elaborado por
Otávio Migliavacca Madalosso

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Andrea Schwertner Charão, Dr[a].
(Presidente/Orientadora)

Benhur De Oliveira Stein, Prof. Dr. (UFSM)

Henrique Pereira, MSc. (CPD - UFSM)

Santa Maria, 30 de Novembro de 2015.

AGRADECIMENTOS

À minha família, que sempre me proporcionou tudo que precisei durante todos os momentos da minha vida e que me direcionaram para onde cheguei hoje.

Aos meus grandes amigos, que também fizeram o papel de família e que devem e serão tratados como tal enquanto eu viver.

À banca, pelos comentários construtivos e incentivadores.

À minha coordenadora, que desde meu terceiro semestre no curso me incentivou e ofereceu oportunidades em diferentes projetos de pesquisa.

À comunidade de desenvolvedores de todas as ferramentas utilizadas nesse projeto, por seus ensinamentos, explicações, e boa vontade de difundir conhecimento.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

UM SISTEMA *WEB* PARA EXECUÇÃO REMOTA DE APLICAÇÕES DE ALTO DESEMPENHO

AUTOR: OTÁVIO MIGLIAVACCA MADALOSSO
ORIENTADORA: ANDREA SCHWERTNER CHARÃO
Local da Defesa e Data: Santa Maria, 30 de Novembro de 2015.

Algumas áreas de pesquisa utilizam constantemente algoritmos que demandam alto desempenho dos seus ambientes de execução. Ocasionalmente, surgem algoritmos novos, com diferentes propriedades, que se propõem a resolver um problema de forma mais eficiente e/ou completa. Esses algoritmos demandam uma plataforma prática para serem disponibilizados ao público de pesquisa sem fiquem restritos a ambientes institucionais. Este trabalho tem como objetivo criar um modelo de portal *web* que permita ao administrador gerenciar algoritmos que serão disponibilizados para os usuários, e que permita o cadastro de usuários para fazer uso desses algoritmos. Esse modelo de portal foi desenvolvido seguindo os princípios do desenvolvimento ágil e utilizando linguagens de programação e *frameworks* que possuem uma boa sinergia com o uso desses princípios.

Palavras-chave: Computação de alto desempenho. Programação Web. Framework Django. Execução Remota. Python.

SUMÁRIO

LISTA DE FIGURAS

1 INTRODUÇÃO

Algoritmos com grande custo computacional são facilmente encontrados em áreas como meteorologia, biologia e astronomia. Esses algoritmos possuem a característica de exigir um nível elevado de processamento, e consequentemente, os tempos necessários para suas conclusões tendem a ser longos e variam dependendo do ambiente aonde são executados.

Pesquisadores destas e de outras áreas podem vir a desenvolver novas implementações de algoritmos utilizados pela comunidade de pesquisa. Essas implementações podem trazer muitos benefícios para outros pesquisadores que necessitam deste tipo de solução. Infelizmente, é comum essas implementações ficarem restritas a ambientes privados, não por questões de licença, mas simplesmente pela ausência de um método prático para disponibilizá-la ao público.

Baseado nessa situação, surge a ideia de desenvolver um portal *web* que permita a execução de algoritmos remotamente, de acordo com as configurações feitas pelo administrador. Desta forma, o usuário seria capaz de utilizar dados próprios para que sejam processados pelos algoritmos, e consiga obter os resultados quando a tarefa for concluída.

Portais como o descrito acima já existem, porém são desenvolvidos de acordo com a estrutura aonde serão mantidos. A vantagem deste projeto é que será mais genérico, prevendo a sua utilização por diferentes públicos e em diferentes ambientes.

Um algoritmo que se enquadra no propósito do portal e que será utilizado durante o desenvolvimento do mesmo, é a uma versão do algoritmo *Friends-of-Friends* (HUCHRA J. P. E GELLER, 1982) de complexidade n*log(n) paralelizado através do *framework* OpenMP. Essa variação do algoritmo foi desenvolvida em um projeto de pesquisa vinculado ao INPE (Instituto Nacional de Pesquisas Espaciais) pelo autor deste trabalho.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo deste trabalho é criar um modelo de portal *web* que possibilite aos usuários cadastrados no portal executar algoritmos utilizando diferentes dados e disponibilizar o resultado da execução após sua conclusão.

1.2 Justificativa

O projeto é capaz de gerar benefícios significativos para a comunidade de pesquisa de diversas áreas, criando um modelo de ambiente que facilite a divulgação e teste de resultados de algoritmos alternativos para resolução de problemas comuns.

Além de servir como modelo, o projeto disponibilizará um algoritmo que se enquadra na categoria alvo do projeto: a versão de complexidade n*log(n) e paralela do *Friends-of-Friends*.

2 FUNDAMENTOS E REVISÃO DE LITERATURA

Para a produção deste projeto foi necessário fazer um estudo de ferramentas que iriam ser utilizadas para auxiliar na execução do trabalho. A natureza de um portal web, por exemplo, já apresenta um grande número possibilidades de *frameworks web* com o propósito de auxiliarem no desenvolvimento deste portal. Priorizando a velocidade de desenvolvimento e afinidade do autor com a linguagem, foi escolhido o *framework* Django (DJANGO, 2015), para ser utilizado na implementação.

Também é necessário o uso de um gerenciador de tarefas para lidar com as tarefas que serão geradas através do portal. Como os algoritmos que serão utilizados neste portal podem demandar alto custo computacional, é necessário que o ambiente aonde o portal será mantido não seja o mesmo aonde as tarefas serão executadas. Assim, impossibilitando a concorrência entre a execução algoritmo e o processo que mantém o portal. Seria inapropriado o portal aguardar a conclusão de uma tarefa ser concluída para só então retornar uma resposta ao usuário que a solicitou, para resolver isso, foi escolhido o gerenciador Celery (CELERY, 2015) para realizar o controle dessas execuções.

2.1 Django

Django é um *framework* escrito na linguagem *Python* para a criação de aplicações *web*que encoraja o desenvolvimento ágil, em alto nível e com design pragmático. Desde julho de 2005 é um projeto de código aberto publicado sob a licença BSD.

O principal objetivo do *framework* é facilitar a criação de *websites* dirigidos a banco de dados e se relaciona muito com a política de *DRY (Don't Repeat Yourself)* e com o conceito de aplicações modulares (ou plugáveis). Dentre as características desejadas para realização do projeto, o Django pareceu adequado por apresentar suporte a tecnologias que favorecem o desenvolvimento ágil do projeto, como mapeamento objeto-relacional e um modelo de desenvolvimento em camadas *(Model-View-Template)*.

Definição de filosofias de projeto Django segundo a documentação do *framework*:

- Baixo acoplamento: Uma camada do *framework* não precisa conhecer as demais camadas que o compõe a menos que seja absolutamente necessário.
- Menos código: Aplicações Django deve usar o mínimo de código possível. Tirando o

máximo de vantagem das capacidade da linguagem Python.

Algumas definições com relação a nomenclatura de módulos do *framework* segundo o livro *Two Scoopes of Django* (GREENFELD; ROY, 2015):

- Projeto Django : Portal/Aplicação Web mantida pelo *framework* Django
- Django *apps*: bibliotecas representando um único aspecto do projeto. Um projeto Django contém varios apps.
- *INSTALLED_APPS*: Lista de Django apps utilizando por um projeto Django.
- *Third-Party-packages* : são apps modulares que são obtidos com a ferramenta de pacotes Python.

2.2 Celery Task Queue

Celery é um gerenciador de tarefas assíncronas baseado em troca de mensagens entre a aplicação que irá criar tarefas, e os *workers* (processos que irão executar as tarefas). O fluxo de trabalho do Celery pode ser dividido nos estágios de requisição, execução e retorno de tarefas, a requisição é feita através do portal, pelo usuário, e enfileirada na lista de tarefas pendentes. Enquanto houverem tarefas pendentes, elas serão atribuídas aos *workers* que fazem parte do sistema, que irão executar e retornar ao portal o resultado da execução.

Além de resolver o problema de controle de tarefas, essa aplicação também traz um outro benefício ao projeto que é a possibilidade de expandir o número de *workers* que irão tratar das tarefas, ou seja, conforme a demanda de execuções aumente, será simples configurar outra máquina para fazer o papel de mais um *worker*.

2.3 Redis

Redis (SANFILIPPO, 2015) é uma ferramenta de armazenamento e comunicação de dados mantida sob a licença BSD (*open-source*). Ele será utilizado neste projeto atuando como *broker* entre as máquinas que vão agir como *workers* e a máquina que irá manter a aplicação disponível aos usuários (Django *server*).

Além do Redis, há somente uma outra opção estável de *broker* para ser utilizado em conjunto com o Celery: o "RabbitMQ". Dentre essas duas opções, o Redis foi escolhido por

apresentar mais informação na documentação e mais participação da comunidade que utiliza Celery.

2.4 Friends-of-Friends

O *Friends-of-Friends* (HUCHRA J. P. E GELLER, 1982) é um algoritmo utilizado para manipular e analisar grandes quantidades de dados produzidos por simulações da área da astronomia, mais especificamente em tópicos como a distribuição de matéria escura em grande escala, a formação de halos de matéria escura, e a formação e evolução de galáxias e aglomerados. Essas simulações tem um papel fundamental no estudo desses assuntos (BERTSCHINGER, 1998; G. EFSTATHIOU M. DAVIS, 1985).

Na 15ª edição do ERAD-RS[1], foram publicados resultados de execuções de uma nova implementação do *Friends-of-Friends* (OTAVIO M. MADALOSSO, 2015) originados de um projeto de pesquisa cujo objetivo era reduzir a complexidade do algoritmo e paralelizá-lo para que obtivesse uma diminuição de seu tempo de execução.

O algoritmo funciona utilizando uma entrada de dados composta por posições de n corpos celestes que devem ser agrupados de acordo com um dado raio. Quando dois corpos estão posicionados a uma distância menor do que a do raio informado, eles pertencem a um mesmo grupo e qualquer outro corpo que estiver a uma distância menor ou igual ao raio de qualquer integrante de um grupo, também pertence ao grupo. O resultado esperado do algoritmo é informar quais corpos estão relacionados entre sí seguindo essa regra.

Uma característica significativa desse algoritmo é o grande volume de dados que compõem o arquivo de entrada, durante o desenvolvimento do algoritmo, foi utilizado um arquivo de entrada com informação de 317 mil corpos celestes. Como esse arquivo precisa ser enviado a partir do usuário para o sistema, é necessário que seja determinado um limite do tamanho do arquivo e um prazo de validade para o qual esse arquivo continuará disponível no sistema após sua utilização. Caso contrário é inevitável que, conforme o portal seja utilizado, os recursos de memória do ambiente destinados a manter os arquivos sejam comprometidos.

[1] Escola Regional de Alto Desempenho do Rio Grande do Sul

2.5 Trabalhos Relacionados

Como citado anteriormente, existem alguns projetos que compartilham algumas características com a proposta do trabalho. Os portais de *eResearch* por exemplo, focam em usuários pesquisadores, que demandam alta capacidade computacional para realizarem suas pesquisas, além de acesso a certos conjuntos de dados e aplicações ligadas a suas áreas de pesquisa. Todos esses recursos são disponibilizados para o usuário através de complexas uniões de diferentes recursos, gerenciados pelas instituições que mantém o serviço e entregues para o usuário final de forma transparente, facilitando a utilização pelo mesmo.

- New Zealand eScience - NeSI (NEW ZEALAND ESCIENCE INFRASTRUCTURE, 2015) - O NeSI é o serviço de infraestrutura de pesquisa da Nova Zelândia, oferece vários serviços para a comunidade de pesquisa do país. Para ter acesso ao sistema é necessário possuir um usuário que esteja vinculado com algum projeto já existente, ou por meio de um formulário para aplicação de um novo projeto, que será avaliado e se aprovado, cadastrado no sistema (junto com os dados que permitem acesso ao usuário).

- The National e-Science Centre - NeSC (NATIONAL E-SCIENCE CENTRE, 2015) - A universidade de Glasgow, no Reino Unido, mantém esse projeto que tem foco em tecnologia de *grid* combinada com aplicações de pesquisa. O domínio das aplicações suportadas abrangem áreas como bio-informática, testes clínicos, epidemiologia, entre outras. O principal objetivo desse projeto é dispor ao usuário um ambiente de *HPC (High-Performance Computing)* completamente transparente ao usuário final. Assim como o NeSI, também é um projeto desenvolvido para ser executado e mantido em um conjunto de máquinas específicas, que dispõem dos requisitos necessários.

Existem também portais que a interação que o usuário faz com o sistema é a produção de código fonte em determinada linguagem. O objetivo desses portais varia entre o ensino de novas técnicas de programação, ensino das linguagens e desafios de desempenho de algoritmos. Para as situações de ensino, o sistema pode instruir o usuário como resolver determinado problema e solicitar que o mesmo resolva algum problema semelhante, provando que aprendeu o conteúdo proposto. Para desafios de algoritmos, o usuário envia a sua solução proposta para o sistema executar, e após a execução, recebe uma avaliação que é publicada em um rank, promovendo competição entre os usuários. Exemplos de portais que seguem essa metodologia:

- HackerRank (HACKERRANK, 2015) - Este site desafia os usuários a encontrem as melhores soluções para desafios computacionais que podem ser resolvidos em várias linguagens de programação. Cada solução proposta por um usuário é avaliada de acordo com o resultado do algoritmo enviado e seu tempo de execução, essa avaliação é publica em um *rank* contendo as demais avaliações da comunidade, criando uma competição associada ao desafio proposto.

- Codecademy (CODECADEMY, 2015) - Diferente do exemplo anterior, a proposta do Codecademy é o ensino de linguagens e paradigmas de programação para seus usuários. Através da plataforma, o usuário pode selecionar, a partir de uma lista de linguagens, qual linguagem de programação deseja aprender, e seguir um tutorial contendo exemplos e desafios de programação da opção escolhida. Os códigos gerados são testados na plataforma e se os resultados forem conforme os esperados de cada exercício, é liberado um novo tópico de aprendizagem até que o curso seja finalizado.

- CodeinGame (CODEINGAME, 2015) - Essa plataforma pode ser avaliada como um intermediário entre os outros dois sites citados acima. Assim como o Codecademy, o propósito do CodeinGame é o ensino de técnicas e linguagens de programação, porém, ele tem possui as características competitivas do HackerRank pois também oferece placares de liderança e "batalhas"de códigos (principalmente ligados a área de inteligência artificial), aonde dois participantes cadastram seus algoritmos que serão avaliados para resolução de um mesmo problema.

Outro modelo de portal pode ser observado no Algorithmia (ALGORITHMIA, 2015), uma plataforma que serve de intermediário entre desenvolvedores e clientes do mercado de algoritmos. O desenvolvedor pode disponibilizar um algoritmo no portal, e cobrar royalties por sua utilização, enquanto o portal faz a execução do algoritmo solicitado pelo cliente, e cobra uma taxa pelo serviço por execução.

3 DESENVOLVIMENTO

Aqui serão descritas as atividades realizadas para atingir os objetivos propostos. Serão apresentados tópicos fundamentais para o desenvolvimento do projeto como tipos de usuários, modelos de dados e fluxograma, e também detalhes de implementação a respeito das principais funcionalidades desenvolvidas.

A metodologia do projeto concentrou-se em dar prioridade aos requisitos que apresentavam maior risco, priorizando a funcionalidade de executar tarefas remotamente. Para agilizar a organização e desenvolvimento, os requisitos foram organizados em um *Product Backlog* no qual foram listados segundo critérios de risco e de dependências. Esse *Product Backlog* foi definido em conjunto entre o autor, a orientadora do trabalho e o orientador do projeto de pesquisa que originou o algoritmo utilizado no projeto (*Friends-of-Friends*, versão n*log(n)), e mantido em uma planilha.

A técnica utilizada para criação dos requisitos foi a chamada "histórias de usuário".

Histórias de usuário (*user stories*) descrevem seus requisitos em uma forma ágil, fazendo uso de regras para escrita de situações de onde serão extraídos requisitos para o sistema. Essas regras podem variar, a utilizada nesse projeto está representada na Figura 3.1a.

Como a maioria dos requisitos extraídos do projeto são dependentes entre sí, não foram atribuídos pesos aos itens do *backlog* e a sequência de requisitos implementados seguiu o ritmo natural do desenvolvimento:

- Desenvolvimento de telas.
- Login no sistema.
- Registro de novo usuário.
- Registro de novo algoritmo pelo administrador.
- Usuário solicitar nova execução.
- Sistema processar execução de usuário remotamente.
- Disponibilização de resultados e arquivos da execução solicitada.

Foram utilizadas alguns Third-Party-Packages que poderiam trazer ganhos ao projeto em questão de agilidade de desenvolvimento. Foram elas a *django-registration-redux* [2], para

[2] http://django-registration-redux.readthedocs.org/

```
Como um: indivíduo que participa do sistema.
Eu quero: necessidade do usuário descrita por ele.
Para que: Razão que justifique o requisito.
```

(a) *História de usuário - cartão modelo*

```
Como um desenvolvedor de algoritmos, eu quero manter uma plataforma para
  expor meus algoritmos para serem utilizados por outras pessoas.
```

(b) *Cartão pelo administrador do sistema.*

```
Como um usuário do sistema, quero executar os algoritmos oferecidos
  utilizando meu próprio conjunto de dados para entrada do algoritmos
  selecionado, e apurar o tempo de execução e o resultado da execução.
```

(c) *Cartão pelo usuário cadastrado.*

```
Como um usuário não cadastrado, quero requisitar acesso ao portal para
  poder executar meus experimentos.
```

(d) *Cartão pelo usuário não cadastrado.*

Figura 3.1: Cartões de histórias

registro de usuário, e a *django-crispy-forms*[3], que efetua validação e renderização de formulários.

3.1 Funcionalidades e tipos de usuário

As funcionalidades do sistema podem ser divididas em 3 grupos distintos (Figura 3.2).

3.1.1 Usuário Anônimo

Um usuário anônimo possui a permissão de acessar áreas de informação a respeito do sistema, de contato com o administrador do sistema, e da área de registro, onde pode solicitar o registro e, seguindo as orientações apresentadas, fazer login como um usuário registrado. (Figura 3.1d)

3.1.2 Usuário registrado

O usuário registrado tem permissão de criar execuções no portal, para isso ele faz o *upload* de um arquivo que será utilizado como entrada no algoritmo selecionado. O usuário

[3] http://django-crispy-forms.readthedocs.org/en/latest/

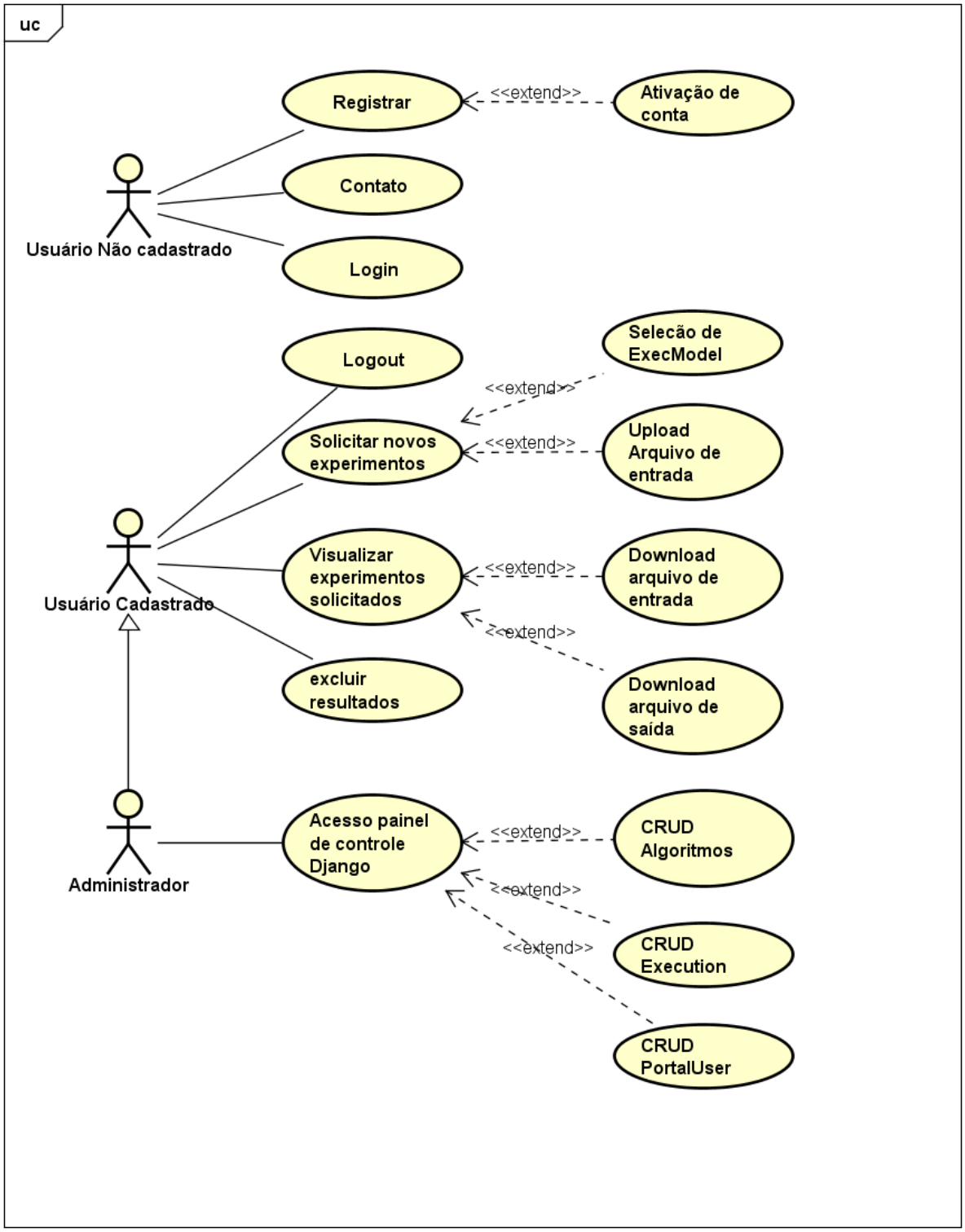

Figura 3.2: Diagrama de casos de uso do projeto.

também pode monitorar o estado dos execuções que ele requisitou e fazer o *download* dos arquivos de cada experimento, tanto o arquivo de entrada, como a saída (se houver) do algoritmo. O usuário pode selecionar execuções realizadas por ele e as excluir. (Figura 3.1c)

3.1.3 Administrador

O administrador tem as mesmas capacidades do que um usuário registrado e detém privilégios de acesso ao painel de administração do Django. Isso permite que ele faça o cadastro de novos algoritmos no sistema e a editar qualquer informação que o sistema detenha no banco de dados. (Figura 3.1b)

3.2 Modelos de dados

Definidas as funcionalidades a serem desenvolvidas, foram criados modelos de dados para manter as informações dos usuários, de algoritmos disponíveis pelo sistema e execuções requisitadas pelos usuários.

Essas 3 classes compõem o modelo de dados (Figura 3.4) que integram todas as funcionalidades do sistema, todos herdam a base de modelo do Django. Uma visão mais detalhada dos modelos pode ser observada na Figura 3.3, o arquivo desenvolvido para definir os modelos do sistema.

3.2.1 Algorithm

Essa classe é responsável por manter os dados referentes ao(s) algoritmo(s) que o sistema disponibiliza, é composto por 3 atributos que representem o nome do algoritmo, a descrição de seu propósito, e o comando que deverá ser utilizado para executá-lo.

3.2.2 Execution

A classe *Execution* mantém todas as informações referentes a uma execução. Ela inclui uma chave estrangeira que referencia o usuário que requisitou-a, a data na qual a requisição foi feita, o status da execução, outra chave estrangeira que referencia qual algoritmo será utilizado, dois campos para os endereços nos quais devem ser mantidos os dados de entrada e saída à serem usados, e o tempo gasto na execução.

```python
from django.db import models
from django.contrib.auth.models import User


class Algorithms(models.Model):
  idAlg = models.AutoField(primary_key=True)
  nameAlg = models.CharField(null=False, blank=False, max_length=100)
  desc = models.CharField(null=True, blank=False, max_length=500)
  command = models.CharField(null=False, blank=False, max_length=100)

  def __unicode__(self):
    return self.nameAlg


class UsuarioPortal(models.Model):
  nickname = models.CharField(
    default='default', max_length=30, blank=False, null=True)
  company = models.CharField(
    default='default', max_length=50, blank=False, null=True)
  usuario = models.OneToOneField(User)
  date_register = models.DateTimeField('date_register', auto_now_add=True)
  last_access = models.DateTimeField('last_access', auto_now=True)
  resultsPerPage = models.IntegerField(default=10)

  def __unicode__(self):
    return self.nickname


def user_directory_path_in(instance, filename):
  return './users/user_{0}/{1}/input'.format(instance.request_by.usuario.id
    , instance.id)


def user_directory_path_out(instance, filename):
  return './users/user_{0}/{1}/output'.format(instance.request_by.usuario.
    id, instance.id)


class Execution(models.Model):
  request_by = models.ForeignKey(UsuarioPortal)
  date_requisition = models.DateTimeField(
    'date_requisition', auto_now_add=True)
  status = models.IntegerField(default=1)
  algorithm = models.ForeignKey(Algorithms, null=True, blank=False)
  inputFile = models.FileField(upload_to=user_directory_path_in, null=True)
  outputFile = models.FileField(upload_to=user_directory_path_out, null=
    True)
  time = models.FloatField(default=-1)

  def __unicode__(self):
    return self.request_by.id

```

Figura 3.3: Models.py

3.2.3 PortalUser

O PortalUser é a classe que mantem os dados de cada usuário do portal. Ela se relaciona com a aplicação nativa do Django para autenticação de usuários, a "*Auth*", que continua tratando das tarefas relacionadas a acesso ao sistema. Essa aplicação mantém os dados de usuário, senha, *e-mail* e informações de data de registro e último acesso.

Além disso, a classe portal mantém outros dados:

- Nickname (Utilizado para mensagens de contato e alertas pelo sistema).
- Company (Para manter os dados de qual instituição o usuário pertence).
- ResultsPerPage (Preferência de quantos resultados por página o usuário deseja quando for visualizar sua lista de execuções).

3.3 Fluxo de execução

O fluxo de execução do portal segue conforme a Figura 3.5.

3.4 Etapas de implementação

Essa seção irá explorar problemas e as soluções utilizadas no desenvolvimento de algumas funcionalidades do portal.

3.4.1 Formulário de Contato

A página de contato do sistema foi desenvolvida utilizando um formulário que exige informações necessárias para identificação e endereço de resposta de quem efetuou o contato. Para isso foi utilizada a aplicação *django-crispy-forms* para obtenção e validação de dados.

3.4.1.1 Django Crispy Forms

Durante a implementação dessa funcionalidade foi instalada a aplicação Crispy. Ela trata da criação de formulários e executa a validação e renderização desses formulários. Quando um usuário preenche incorretamente um formulário, ela trata de renderizar um novo fazendo alterações nos campos preenchidos incorretamente, tornando evidente aonde se encontram os erros de preenchimento.

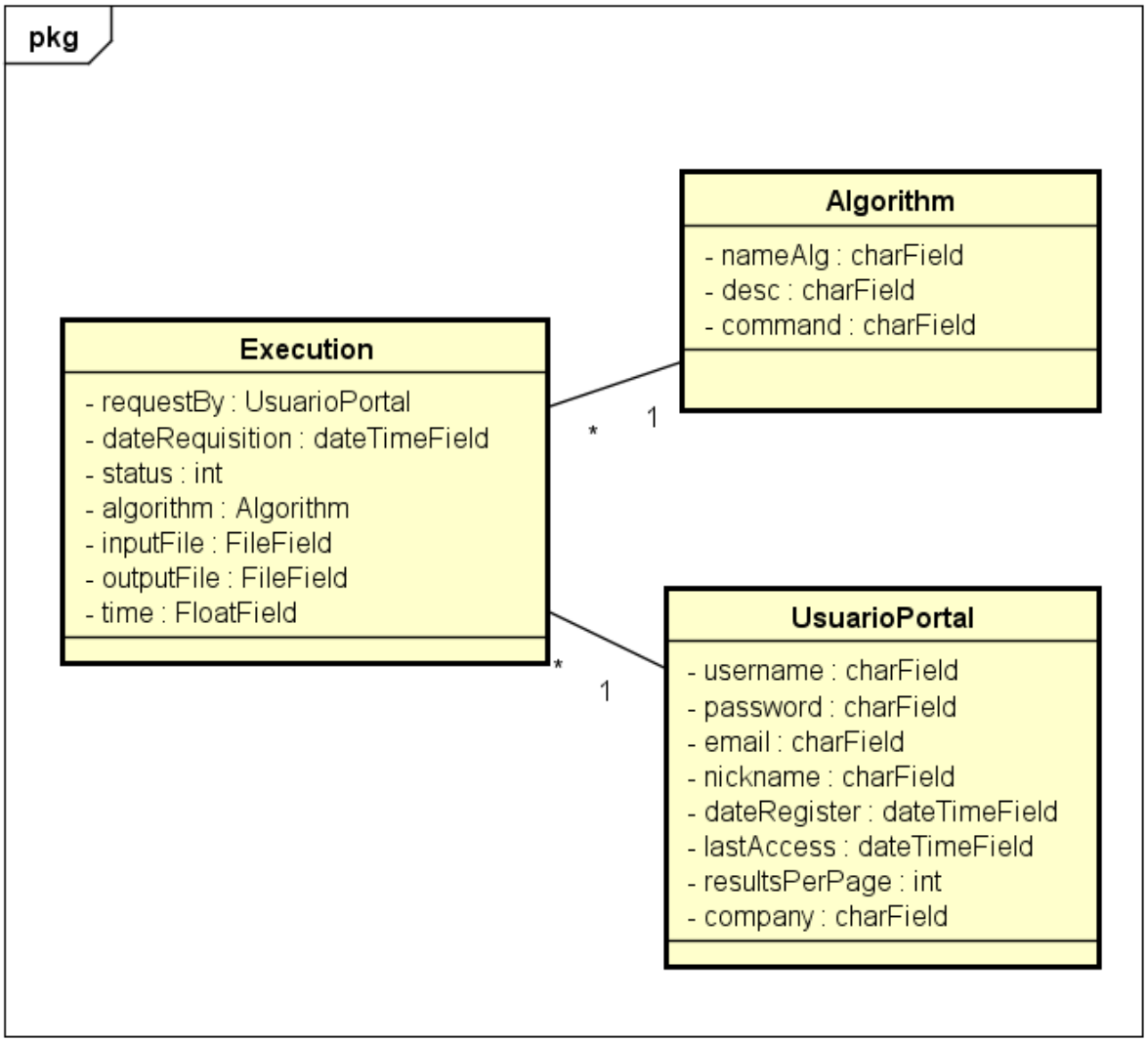

Figura 3.4: Diagrama de classes da aplicação.

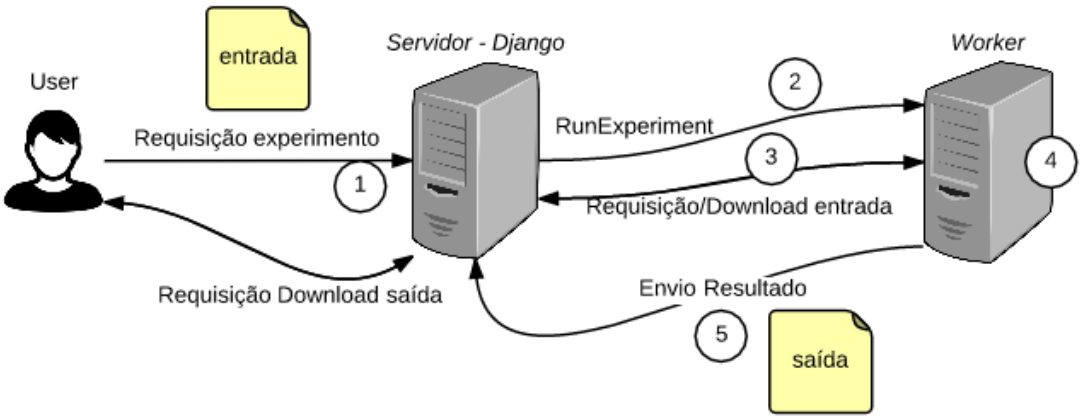

Figura 3.5: Fluxograma.

Esse *third-party-package* foi utilizado não somente na tarefa de contato como também em todos os demais formulários do projeto.

3.4.2 Registro de Usuário

Como qualquer pessoa deve poder utilizar o portal, é imprescindível que haja uma forma autônoma de registro de novos usuários. No primeiro momento tentou-se implementar a funcionalidade utilizando formulários e verificações desenvolvidas no próprio projeto, mas durante o desenvolvimento foi encontrada uma solução mais rápida e mais adequada, o uso do *third-party-package "Django-Registration-Redux"*.

O *Django-Registration-Redux* provê as funcionalidades de registro de usuários para projetos Django. Essa aplicação já dispõe de templates e formulários para desempenhar a sua função, também contém um sistema de ativação de contas no qual o usuário que solicitou registro recebe um e-mail no endereço usado no cadastro que contém um link para ativar sua conta, forçando uma verificação de que o e-mail utilizado existe e pertence mesmo ao usuário.

Por padrão ela mantém apenas os dados de *e-mail*, nome e senha do usuário que solicitou o registro, o que satisfaz os requisitos mínimos para executar sua funcionalidade, porém para o sistema em desenvolvimento, pareceu interessante a possibilidade de obter mais dados do usuário em seu cadastro, tais como instituição a qual o usuário pertence (Universidade, empresa, etc.), apelido pelo qual o usuário opta por ser tratado, e outras informações que podem ser úteis posteriormente ao administrador do sistema.

Além disso, essa aplicação não tem por padrão, nenhuma forma de filtro de domínios de endereços de e-mail, o que também pode ser útil ao sistema, caso o administrador queira limitar o acesso do portal aos usuários que detenham um e-mail de um domínio específico.

Para desenvolver essas ideias, foi criado um formulário com os campos extras desejados que estende o formulário padrão do *Django-Registration-Redux*, assim, a tela de registro de usuário(Figura 3.7) exibe um formulário que solicita os dados tanto da aplicação, quanto do modelo de usuário. O código para criação desse formulário pode ser observado na figura(Figura 3.6a) Além disso, foi criada uma função para sobrescrever a função de registro no sistema, essa função continua executando a operação padrão de registro do *Django-Registration-Redux*, mas também associa os dados desse registro ao dados de usuário do PortalUser. O código que realiza essa operação pode ser observado na Figura 3.6b.

```
from django import forms
from registration.forms import RegistrationFormUniqueEmail
from .models import Algorithms


class UsuarioPortalForm(RegistrationFormUniqueEmail):
  nickname = forms.CharField(required=False)
  company = forms.CharField(required=False)

```

(a) forms.py

```
from registration.backends.default.views import RegistrationView
from experiment.forms import UsuarioPortalForm
from experiment.models import UsuarioPortal

class MyRegistrationView(RegistrationView):

  form_class = UsuarioPortalForm

  def register(self,request,form_class):
    new_user = super(MyRegistrationView,self).register(request, form_class)
    user_profile = UsuarioPortal()
    user_profile.usuario = new_user
    user_profile.nickname = form_class.cleaned_data['nickname']
    user_profile.company = form_class.cleaned_data['company']
    user_profile.save()
    return user_profile

```

(b) regbackend.py

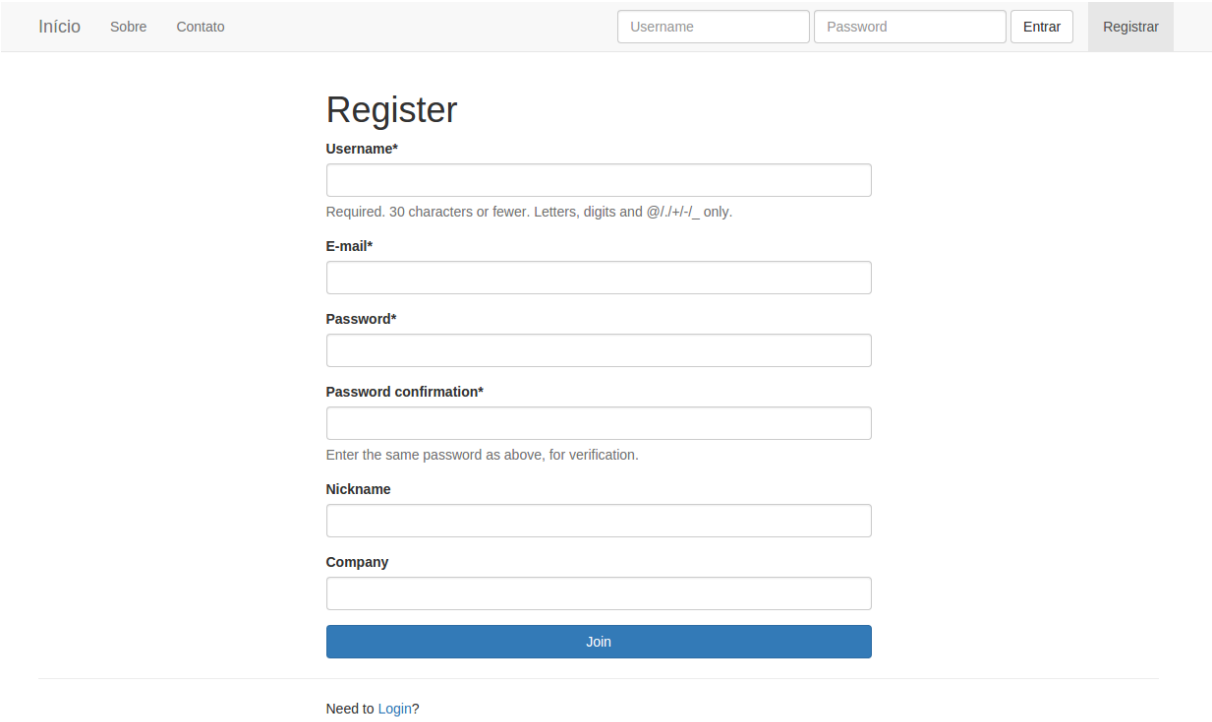

Figura 3.7: Formulário de Registro de Usuário.

3.4.3 Execução remota de tarefas

A execução remota de tarefas foi um requisito que exigiu busca de ferramentas e técnicas para sua realização. É necessário esclarecer que não é viável que o próprio processo que mantém a aplicação no ar e trata de todos os *requests* realizados por usuários também lide com as execuções solicitadas pelos mesmos, pois isso causaria uma lentidão muito grande no sistema.

Para contornar esse problema foram verificadas duas técnicas: a criação de um novo processo que faria a execução do experimento, ou então o uso de alguma aplicação que gerencie filas de tarefas e distribuição das mesmas para demais processos e/ou máquinas.

Compreendida a dimensão e complexidade de criar um sistema para execução de tarefas a partir do zero, optou-se por buscar por aplicações compatíveis com as tecnologias utilizadas do projeto e que pudessem satisfazer a necessidade identificada.

A aplicação escolhida foi o Celery, ele gera filas de execução de tarefas por meio de troca de mensagens. A máquina que mantém o portal, também mantém um processo de execução do broker Redis, que faz o envio e recebimento de mensagens entre o processo que cria novas tarefas, e os *workers* disponíveis para receber tarefas.

```
from models import UsuarioPortal, Execution
from celery.utils.log import get_task_logger
from celery.decorators import task
import requests
import os
import time

logger = get_task_logger(__name__)

@task(name="RunExperiment")
def RunExperiment(execution, ide):
  os.system("mkdir " + str(ide))
  os.system("wget http://10.1.4.28:8000/experiments/downloadInputFile?id="
    +
  str(ide) + " -O ./" + str(ide) + "/input")
  start = time.time()
  os.system(execution + " " + str(ide) + "/input >" + str(ide) + "/output")
  dur = time.time() - start
  files={'file': str("/"+str(ide) + "/output")}
  path = str(str(ide)+"/output")
  files = {'file': open(path, 'rb')}
  data = {'id':str(ide),'time':dur}
  r = requests.post('http://10.1.4.28:8000/experiments/result', files=files
    ,data=data)
  print (r.status_code, r.reason)

```

Figura 3.8: tasks.py

A execução de qualquer algoritmo é feita através da mesma função "RunExperiment"(Figura 3.8). Essa função recebe como argumentos o comando que o *worker* deve executar e o identificador do arquivo de entrada que deve ser utilizado. Em seguida, o *worker* faz o download do arquivo de entrada para seu sistema de arquivos local e realiza a execução do algoritmo selecionado com esse arquivo de entrada recém adquirido. Entre o comando para o sistema realizar a execução, foram criadas duas variáveis para realizarem a logística de cronometrar o tempo de execução. Por fim, cria um formulário POST contendo o arquivo de saída da execução e o tempo de execução do experimento, e o envia junto com um request para a máquina que gerou a tarefa. A view para qual o request será mapeado através da url irá salvar o arquivo recebido como a saída da execução e atualizar os dados relativos a aquele experimento.

O fluxograma apresentado na Figura 3.5 ilustra as operações comentadas acima, e que foram implementadas conforme mostra a código da Figura 3.8.

Essa implementação permite ao portal criar tarefas que serão executadas pelos chamados *workers*. Os *workers* são processos independentes que devem ser iniciados nas máquinas que irão executar o processamento dos algoritmos e que trocam mensagens com o sistema que

Figura 3.9: Tela de acompanhamento das execuções.

solicitou a execução.

3.4.4 Acompanhamento de execuções

Após a requisição de uma execução, os dados a respeito da mesma são adicionados a uma tabela (Figura 3.9) contendo todas as execuções solicitadas por aquele usuário, e ficam disponíveis na tela inicial do portal. É através dessa tabela que o usuário pode interagir com as execuções requisitadas. Através da tabela, é possível efetuar o *download* do arquivo de entrada e de saída (se houver) da execução. Além disso o usuário tem a opção de selecionar e excluir os dados referentes a execuções que ele solicitou.

Além dos dados gerados na requisição da execução, são apresentados outros 2 dados que informam a situação e o tempo da execução, são eles:

- Tempo: Representando quanto tempo foi gasto na execução.
- Status: Informa ao usuário se a execução já foi processada ou se ainda está aguardando um *worker* assumir a tarefa.

3.4.5 Sistema de arquivos

Como as tarefas realizadas pelo portal exigem dados para serem processados pelo algoritmo, e gera novos dados, foi necessário criar um sistema de arquivos para manter uma ordem na qual seja possível recuperar esses arquivos após o seu processamento. O sistema foi implementado conforme a Figura 3.10 ilustra. Nele, é escolhido um diretório como sendo a raiz de todos os arquivos que o portal irá manter referente a dados de entrada e saída dos algoritmos, e esses dados serão mantidos em diretórios nomeados de acordo com o identificador do expe-

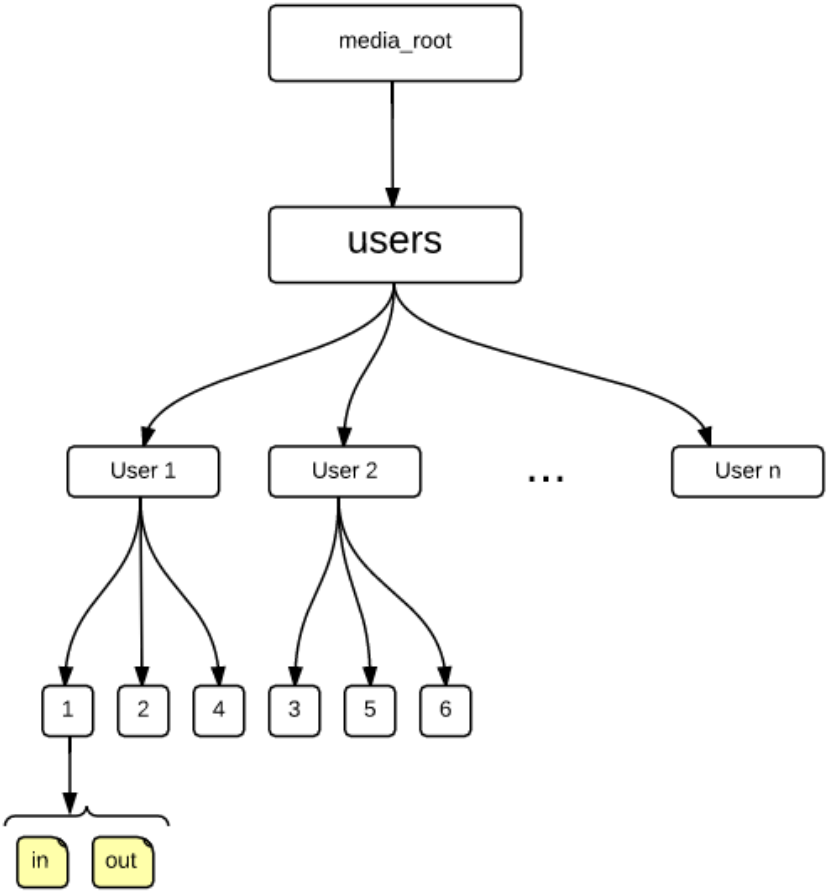

Figura 3.10: Sistema de Arquivos.

rimento ao qual pertencem, e esse diretório, por sua vez, será mantido em um outro diretório nomeado de acordo com o identificador do usuário que requisitou o experimento.

4 RESULTADOS

Ao término do trabalho, foi criado um modelo que satisfaz os requisitos levantados durante a fase de planejamento do portal. Através do modelo gerado, a comunidade de pesquisadores/desenvolvedores de novos algoritmos pode disponibilizar a utilização dos mesmos para qualquer pessoa.

Além disso, o modelo gerado foi utilizado para a criação de um portal que realiza a execução da versão n*log(n) e paralela do *Friends-of-Friends*, tornando agora possível que pesquisadores utilizem o algoritmo para encontrar soluções baseadas em seus próprios conjuntos de dados e verifiquem a diferença de desempenho entre execuções utilizando diferentes algoritmos e/ou entradas. O portal em questão foi executado utilizando duas máquinas disponíveis no Laboratório de Sistemas de Computação (LSC) da UFSM, uma delas mantém o sistema Django disponibilizando o acesso e principais funcionalidades do portal, e a outra age como o único *worker* do sistema, executando as tarefas que o portal gera.

Os teste submetido ao portal para verificar a exatidão e consistência das operações consistiu em executar o algoritmo selecionado utilizando um certo conjunto de dados de entrada, e comparar seu resultado com uma execução feita manualmente utilizando o mesmo algoritmo e o mesmo conjunto de dado. O conjunto de dados de saída processados por ambas as execuções (manual e intermediada pelo portal) resultaram em arquivos completamente iguais, satisfazendo o teste. Nesse teste, foi feito um registro de um usuário com o *username* "NovoUsuario", utilizando dados de *e-mail* válidos para efetuar o registro da conta (Figura 4.1). Após o registro e a validação do e-mail utilizado pelo usuário, ele tem acesso ao ambiente restrito do portal, aonde pode visualizar suas execuções (Figura 4.2) ou acessar o formulário para requisição de uma nova execução (Figura 4.3), selecionando o algoritmo desejado e enviando um arquivo com um conjunto de dados de entrada. Finalmente, o usuário pode observar a tabela contendo os dados das execuções solicitadas e por meio dela, obter o arquivo de entrada ou saída (Figura 4.4).

Quanto a parte de administração, a interação que o administrador faz no sistema é feita através do painel de administração (Figura 4.5) aonde pode cadastrar um novo algoritmo preenchendo os campos do formulário mostrado na Figura 4.6.

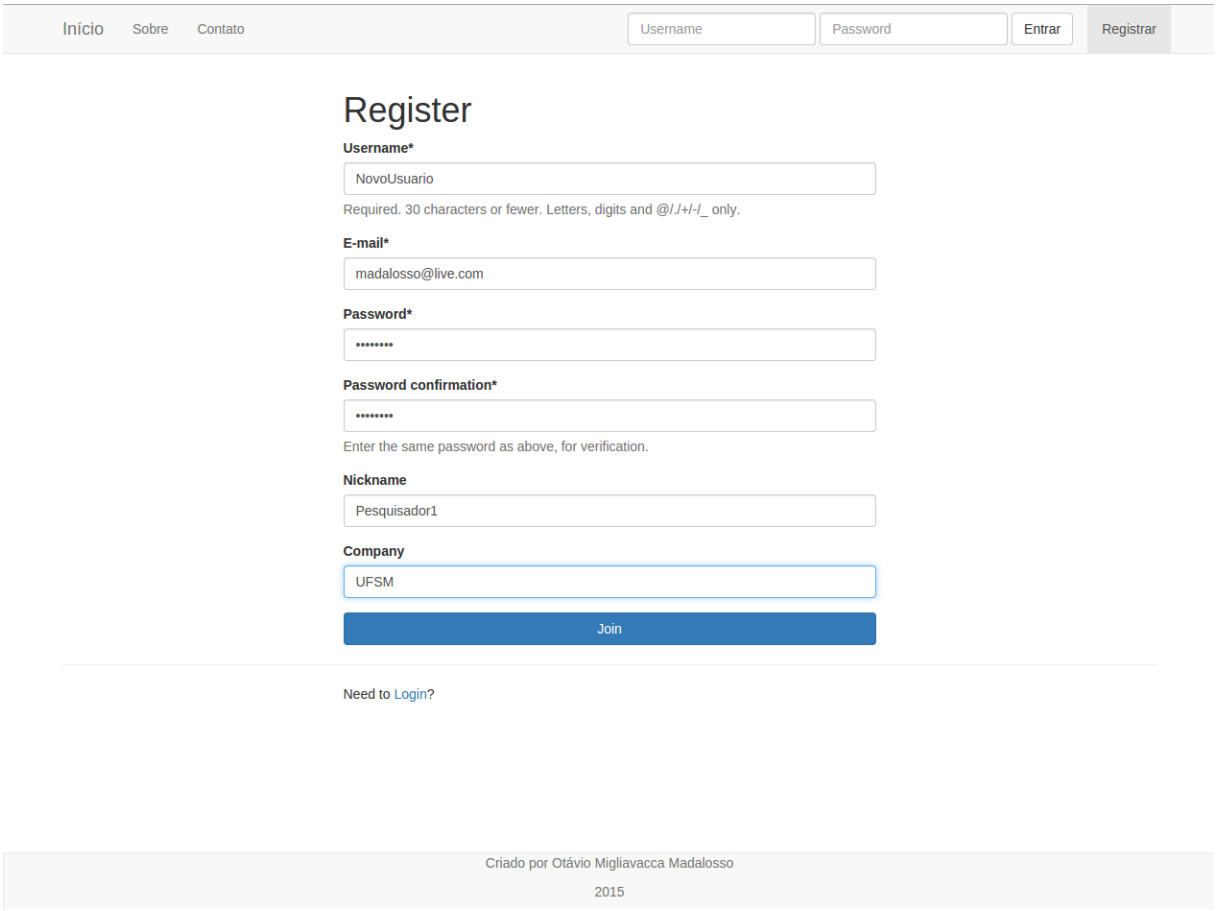

Figura 4.1: Registro de Usuário

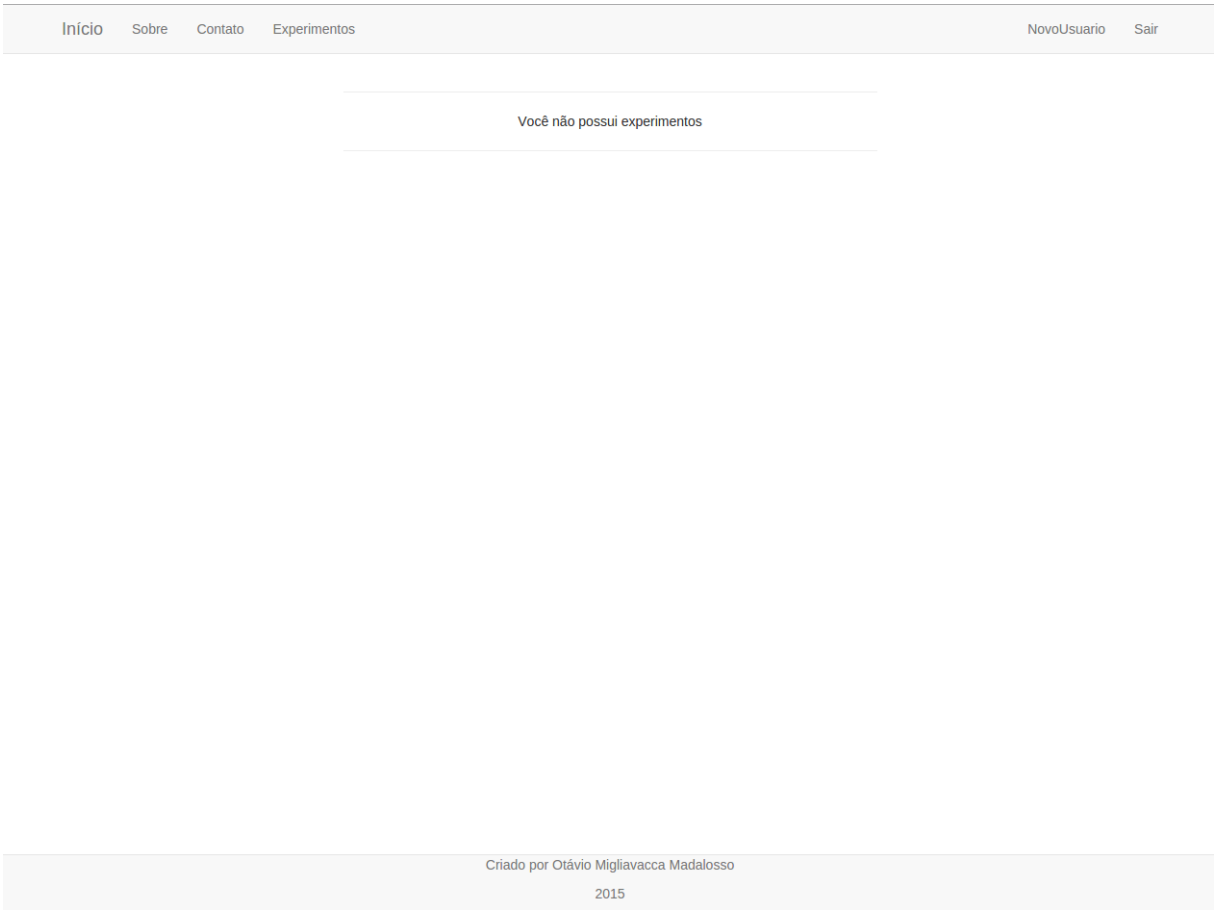

Figura 4.2: Visualização de execuções

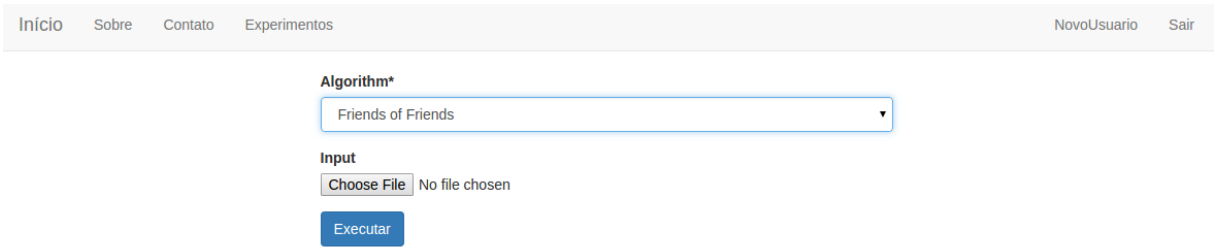

Figura 4.3: Requisição de nova execução

Figura 4.4: Tabela de execuções

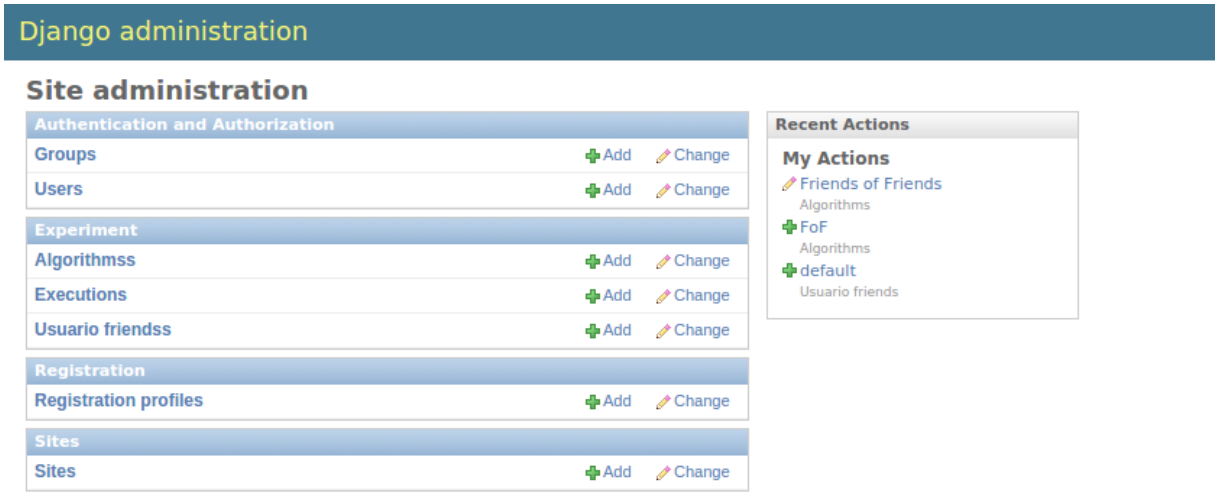

Figura 4.5: Painel de Administração

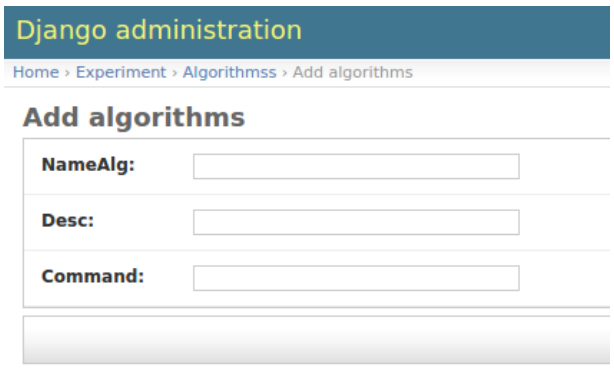

Figura 4.6: Cadastro de novo algorítmo

5 CONCLUSÃO

Este trabalho teve o propósito de criar um modelo de portal capaz de satisfazer as necessidades exploradas na sessão de objetivos: Execução remota de algoritmos de alto desempenho.

A etapa de desenvolvimento iniciou-se através de um levantamento de requisitos e funcionalidades do sistema, e a seguir, a implementação, feita em paralela com o estudo das melhores técnicas e ferramentas para a implementação dessas funcionalidades.

Ao término das implementações, o projeto desenvolvido conseguiu satisfazer os requisitos propostos, gerando um modelo de projeto Django que permite que seus usuários executem algoritmos de alto desempenho produzindo resultados disponibilizados no próprio projeto, além de uma medição do tempo de execução que não sofre interferência de disputa por recursos da máquina que mantém o projeto.

REFERÊNCIAS

ALGORITHMIA. https://algorithmia.com/, acessado em Novembro de 2015.

BERTSCHINGER, E. Simulations of structure formation in the universe. **Annu. Rev. Astron. Astrophys 36**, [S.l.], 1998.

CELERY. **Celery**: distributed task queue. http://celery.readthedocs.org/en/latest/, acessado em Outubro de 2015.

CODECADEMY. https://www.codecademy.com, acessado em Novembro de 2015.

CODEINGAME. https://www.codingame.com/, acessado em Novembro de 2015.

DJANGO. **Django Framework**. https://www.djangoproject.com/, acessado em Outubro de 2015.

G. EFSTATHIOU M. DAVIS, S. D. M. W. C. S. F. Numerical techniques for large cosmological N-body simulations. **Astrophysical Journal Supplement Series (ISSN 0067-0049), vol. 57**, [S.l.], 1985.

GREENFELD, D.; ROY, A. **Two Scoops of Django**: best practices for django 1.8. [S.l.: s.n.], 2015.

HACKERRANK. https://www.hackerrank.com/, acessado em Novembro de 2015.

HUCHRA J. P. E GELLER, M. J. Groups of Galaxies Nearby groups. **The astrophysical**, [S.l.], 1982.

NATIONAL e-Science Centre. http://www.nesc.ac.uk/, acessado em Novembro de 2015.

NEW Zealand eScience Infrastructure. https://www.nesi.org.nz/, acessado em Novembro de 2015.

OTAVIO M. MADALOSSO, A. S. C. Implementação do algoritmo Friends of Friends de complexidade n*logn(n) para classificação de objetos astronômicos. **ERAD-RS XV**, [S.l.], 2015.

SANFILIPPO, S. **Redis**. http://redis.io/, acessado em Novembro de 2015.