



UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO  
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA  
ESCOLA DE INFORMÁTICA APLICADA

Marina de Lima Vinhaes

## **Sistema de Provas Online com desenvolvimento em Python com Django**

**Orientador**

Márcio de Oliveira Barros

RIO DE JANEIRO, RJ – BRASIL

2013

Marina de Lima Vinhaes

## **Sistema de Provas Online com desenvolvimento em Python com Django**

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação.

Aprovada por:

---

Prof. Márcio de Oliveira Barros, DSc. (UNIRIO)

---

Prof.<sup>a</sup> Kate Cerqueira Revoredo

---

Prof. Mariano Pimentel

RIO DE JANEIRO, RJ – BRASIL.

2013

## AGRADECIMENTOS

Agradeço a todos que tiveram alguma participação no projeto de conclusão de curso que eu sempre quis realizar.

In [1]: import this

The Zen of Python, by Tim Peters

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one-- and preferably only one --obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than *\*right\** now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea -- let's do more of those!

## **RESUMO**

Com o avanço da sociedade e da tecnologia, usuários procuram cada vez mais por sistemas que sejam integrados entre si e que ofereçam rápido acesso. A popularização de redes sociais, como por exemplo, o Facebook, torna aplicações integradas extremamente atraentes, por trazerem comodidade aos usuários e tornar desnecessário o processo de criar novas senhas para acessar aplicações. Esta monografia tem como objetivo desenvolver um sistema de provas online e apresentar a linguagem Python, uma ferramenta que vem ganhando notoriedade no mercado. Sua integração com o Facebook terá a finalidade de tornar desnecessário ao usuário realizar seu cadastro em diversas contas diferentes, quando todos os sistemas podem avaliar somente uma conta, como no caso do sistema de provas online, onde a única conta analisada será a cadastrada no Facebook.

**Palavras-chave:** Prova Online, Python, Facebook.

## **ABSTRACT**

With the advancement of society and technology, users are increasingly looking for systems integrated with each other and that offers quick access. The popularity of social networks such as Facebook, makes it extremely attractive integrated applications for bringing convenience to the users to access social networks without creating several unnecessary login accounts. This monograph aims to develop an online exam system and introduce the Python language, a programming language that is gaining notoriety in the market. The integration of the exam system with Facebook will aim to make unnecessary for the user to perform your registration in several different accounts where all systems can evaluate a single account, as in the case of online examination system, where only registered the account will be considered on Facebook.

**Keywords:** Online Tests, Python, Facebook.

## SUMÁRIO

1.Introdução .....	1
1.1 Motivação .....	1
1.2 Objetivos.....	2
1.3 Organização do texto .....	3
2. Desenvolvimento de Aplicações Web com Python e Django .....	4
2.1 Python.....	4
2.1.1 Criar um programa.....	6
2.1.2 Pacotes e módulos .....	7
2.1.3 Classes.....	8
2.1.4 Instâncias.....	9
2.1.5 Palavras reservadas em Python e suas estruturas .....	10
2.1.6 <i>For</i> em Python .....	11
2.1.7. Listas em Python .....	11
2.1.8 Conjuntos em Python.....	14
2.1.9 Dicionários.....	16
2.1.10 Built-in Functions .....	18
2.2 Django .....	19
2.2.1 Template tags e Template Filters.....	20
2.3 Considerações Finais .....	22
3. Técnicas Utilizadas no Desenvolvimento do Sistema de Provas Online.....	23
3.1 Casos de Uso da Aplicação .....	23
3.2 Modelagem Conceitual.....	25
3.3 Funcionalidades do Sistema .....	28
3.3.1 <i>Apps</i> .....	28

3.3.2 <i>Models</i> .....	30
3.3.3 <i>Views</i> .....	31
3.3.4 <i>URL 's</i> .....	33
3.3.5 <i>Templates</i> .....	34
3.3.6 <i>Página de Administração</i> .....	36
3.4 Integração com Facebook.....	37
3.5 Considerações Finais .....	39
4. Navegando no Sistema de Provas Online .....	40
4.1 Apresentação do Sistema: Administradores .....	40
4.2 Apresentação do Sistema: Professores .....	45
4.3 Apresentação do Sistema: Aluno.....	50
4.4 Considerações Finais .....	55
5. Conclusão.....	56
5.1 Contribuições.....	56
5.2 Trabalhos Futuros .....	56
5.3 Limitações do Estudo .....	57

## **Referências**

## LISTA DE CÓDIGOS

Código 1 - Estrutura de um Módulo (arquivo teste.py).....	7
Código 2 - Exemplo de Importação de Módulos com Sucesso .....	7
Código 3 – Estrutura hierárquica de um Pacote .....	8
Código 4 – Import de um Pacote dentro da aplicação .....	8
Código 5 – Exemplo de possível erro ao importar um pacote.....	8
Código 6 – Exemplo de estrutura de uma classe .....	9
Código 7 – Instanciação de objetos de uma classe .....	9
Código 8 – Exemplo de acesso ao método Calcula Subtração.....	10
Código 9 – Exemplo de For percorrendo uma lista de elementos.....	11
Código 10 – Exemplo de For com Range.....	11
Código 11 – Exemplos de listas em Python .....	11
Código 12 – Exemplo de listas com listas .....	12
Código 13 – Primeiro item de uma lista .....	12
Código 14 – Último item de uma lista.....	12
Código 15 – “Fatias” determinando o final de uma lista.....	12
Código 16 – “Fatias” determinando o início de uma lista .....	13
Código 17 – “Fatias” com números de elementos determinado.....	13
Código 18 – Exemplo de list comprehensions .....	13
Código 19 – Outra forma de ler uma list comprehension.....	14
Código 20 – Exemplos de list comprehensions.....	14
Código 21 – Criação de um conjunto .....	15
Código 22 – Comparação de Conjuntos .....	15
Código 23 – União de Conjuntos .....	15
Código 24 – Interseção de Conjuntos.....	15
Código 25 – Diferença de Conjuntos .....	16
Código 26 – Estrutura de dicionários, pares chave e valor .....	16
Código 27 – Exemplo de uma lista sendo transformada em dicionário .....	16
Código 28 – Iterações em Dicionários .....	17
Código 29 – Manuseando dicionários .....	17



Código 30 – Erro caso um valor da tabela não esteja no dicionário .....	17
Código 31 – Evitando Erros com o comando get.....	17
Código 32 – Formatação de strings através de elementos de dicionários .....	18
Código 33 – Um exemplo da função range, um Built-in function .....	19
Código 34 – Exemplo de um template filter com função date .....	20
Código 35 – Exemplo de um template filter com função length.....	21
Código 36 – Laço de repetição em Templates Tags.....	21
Código 37 – Condição em Templates Tags.....	21
Código 38 - Condição e Repetição junta na mesma lógica de desenvolvimento .....	22
Código 39 – Criação de Apps.....	28
Código 40 – Estrutura de uma App .....	29
Código 41 – Estrutura de apps do sistema de Provas Online Unirio.....	29
Código 42 – Comando para criar estrutura no banco de apps .....	30
Código 43 – Exemplos de classes presentes no arquivo models.py .....	30
Código 44 – Exemplo de uma classe meta, presente em classes do modelo.....	31
Código 45 – Código referente a um método presente em uma view.....	32
Código 46 – Exemplo de uma página URL utilizada no projeto.....	34
Código 47 – Exemplo de template tags utilizada no projeto .....	35
Código 48 – Exemplo de Template Filters.....	35
Código 49 – url.py gerado pelo sistema .....	36
Código 50 – Tela com desenvolvimento em HTML do Facebook .....	38
Código 51 – Comando para gerar o dashboard .....	42
Código 52 – Tela Admin.py .....	47

## LISTA DE ILUSTRAÇÕES

Figura 1 - Built-in Functions mais usadas .....	18
Figura 2 – Casos de Uso .....	23
Figura 3 - Modelagem Conceitual do Sistema de Provas Online.....	26
Figura 4 – Tela de Criação do Facebook App .....	37
Figura 5 – Login da Tela de Administração .....	40
Figura 6 – Tela Principal de Administração do Site pela Visão de Administradores .....	41
Figura 7 – Tela de Listagem de Centros.....	42
Figura 8 – Tela de Adição de Centros no Sistema .....	43
Figura 9 – Tela de Preenchimento de Dados de Centros.....	44
Figura 10 – Tela de Atributos de Usuários.....	45
Figura 11 - Tela Principal de Administração do Site pela Visão de Professores .....	46
Figura 12 – Tela de Lista de Provas no Banco de Dados .....	48
Figura 13 – Tela de Adição, Edição e Exclusão de Dados.....	48
Figura 14 – Tela de Correção de Provas.....	49
Figura 15 – Relatório de Notas do Sistema .....	50
Figura 16 – Tela de Login de Alunos .....	51
Figura 17 – Visualização de Alunos que Utilizam a Ferramenta no Facebook .....	511
Figura 18 - Tela de Inscrição de Disciplina.....	522
Figura 19 – Tela de Realização da Prova .....	53
Figura 20 – Mensagem de Prova Realizada com Sucesso.....	533
Figura 21 – Tela com Provas Realizadas.....	544
Figura 22 – Tela de Visualização de Provas com Nota no Sistema .....	54
Figura 23 – Tela de Gabarito de Provas Realizadas .....	55

# 1.Introdução

## 1.1 Motivação

A área de Informática é fascinante. Ao escolher esta área para seguir carreira há diversas especializações que podem ser escolhidas, como por exemplo, as áreas de redes de computadores, banco de dados, *back-end* (voltado para transformar especificações em linguagem de computadores), *front-end* (voltado para a parte visual do projeto, que ajusta um sistema para se adequar a um layout predefinido), aplicativos móveis, área nova no mercado e requisitada, análise de sistemas, teste de software e muitas outras combinações que podem ser feitas com carreiras diferentes de Informática.

No *front-end*, o responsável deve coletar informações sobre o sistema e adequá-las de acordo com o pedido pelo usuário. Sua função é de programar a parte visual do sistema, de forma que o profissional *back-end* possa receber os dados que necessita para executar suas funções. Na área de *front-end* alguns exemplos de linguagem de programação muito utilizadas são XHTML<sup>1</sup>, CSS<sup>2</sup> (incluindo CSS5), Javascript<sup>3</sup>, jQuery, entre outros. Já no *back-end*, exemplos de linguagens de programação são ASP.NET<sup>4</sup>, C (Kernighan e Richie, 1978), C# (Torgersen, 2008), C++ (Stroustrup, 1997), PHP<sup>5</sup>, Python<sup>6</sup> e Java (Gosling e McGilton, 1996).

A linguagem de programação utilizada nesse trabalho - Python - é pouco conhecida por desenvolvedores mais antigos e com mais experiência. Foi escolhida por diversas

---

<sup>1</sup> XHTML – fonte retirada do site <http://www.w3.org/TR/xhtml-modularization/>

<sup>2</sup> CSS – fonte retirada do site <http://www.w3.org/TR/CSS21/syndata.html#q10>

<sup>3</sup> JAVASCRIPT – fonte retirada do site [https://developer.mozilla.org/en-US/docs/JavaScript/New\\_in\\_JavaScript/1.8.5](https://developer.mozilla.org/en-US/docs/JavaScript/New_in_JavaScript/1.8.5)

<sup>4</sup> ASP.NET - fonte retirada do site <http://www.w3.org/TR/CSS21/syndata.html#q10>

<sup>5</sup> PHP - fonte retirada do site [http://www.php.net/manual/pt\\_BR/preface.php](http://www.php.net/manual/pt_BR/preface.php)

<sup>6</sup> PYTHON – fonte retirada do site <http://docs.python.org/tutorial/>

empresas por facilitar o desenvolvimento e ser de fácil aprendizagem, conforme citado por Lutz e Ascher (2007):

“Durante a grande explosão da internet, em meados dos anos 90, era difícil encontrar programadores para implementar projetos de software; os desenvolvedores eram solicitados a implementar sistemas com a mesma rapidez e com que a Internet evoluía. Agora, na era pós-exploração, de demissões e recessão econômica, o quadro mudou. Atualmente, as equipes de programação são obrigadas a executar as mesmas tarefas com menos pessoas.

Nesses dois cenários, o Python se distingue como uma ferramenta que permite aos programadores fazer mais com menos esforço. Ela é deliberadamente otimizada para velocidade de desenvolvimento – sua sintaxe simples, tipagem dinâmica, ausência de etapas de compilação e seu conjunto de ferramentas incorporado, permitem que os programadores desenvolvam programas em uma fração de tempo de desenvolvimento necessário para algumas outras ferramentas. O resultado é que Python aumenta a produtividade do desenvolvedor muitas vezes além do que se consegue com as linguagens tradicionais. Essa é uma boa notícia em tempos de crescimento e recessão.”

Além de mostrar a linguagem de programação Python, o projeto também tem a motivação de realizar uma aplicação que auxilie usuários do meio acadêmico na criação e realização de provas de cursos de uma faculdade. O objetivo é criar cada vez mais ferramentas de auxiliem a professores e alunos nas tarefas realizadas em um período letivo.

Como uma estratégia de visualização e divulgação da aplicação, o projeto será integrado com uma rede social que trará maior notoriedade ao sistema e o tornará cada vez popular, podendo ser visualizado por membros de diversas faculdades .

## ***1.2 Objetivos***

O objetivo do projeto é apresentar uma aplicação web que auxilie professores e alunos a realizar funções do cotidiano de salas de aula pela internet, onde pode-se manter organizado todo o histórico de alterações realizadas em um sistema. O projeto tem o intuito de auxiliar professores a criar provas, permitir que alunos à realizem e por fim dar uma nota ao aluno com um feedback de como foi sua performance na prova.

O desafio desse trabalho será integrá-lo com uma rede social, o Facebook, para que os usuários possam realizar o *login* a partir da sua conta da rede social, poupando o trabalho de criar uma conta somente para realizar as provas.

Devido a poucas pessoas do curso de Graduação de Sistemas de Informação da UNIRIO terem o conhecimento da linguagem Python, neste trabalho será apresentado um sistema que apresentará brevemente suas funcionalidades e será discutido a facilidade do desenvolvimento para aplicações web.

A linguagem será apresentada através do desenvolvimento do sistema de Provas Online, por ter páginas simples e seu fluxo ser de conhecimento de muitas pessoas. Python (Hetland, 2008) é uma ferramenta muito utilizada por empresas, sendo escolhida por diversos profissionais no mercado de trabalho na área de desenvolvimento web.

Um sistema de provas online poderia auxiliar professores de instituições de ensino a aplicar provas à distância e, assim, alunos não precisariam se deslocar de casa para realizar provas. Além disso, o sistema de correção seria rápido e objetivo, ajudando professores.

Além do desenvolvimento do sistema em Python, também será mostrado o *framework* Django (Kaplan-Moss e Holovaty, 2007), utilizado para o desenvolvimento de projetos para web. Serão discutidos os benefícios obtidos ao se utilizar o *framework* no desenvolvimento de aplicações desta natureza.

### **1.3 Organização do texto**

O presente trabalho está estruturado em capítulos e, além desta introdução, será desenvolvido da seguinte forma:

- Capítulo 2 (Desenvolvimento de Aplicações Web com Python e Django): apresenta a linguagem Python, alguns prós e contras de sua implementação, e o *framework* Django;
- Capítulo 3 (Técnicas Utilizadas no Desenvolvimento do Sistema de Provas Online): exhibe informações sobre o fluxo do sistema, através de casos de uso e modelagem conceitual;
- Capítulo 4 (Navegando no Sistema de Provas Online): apresenta detalhadamente a navegação do sistema que foi implemetado no contexto deste trabalho;
- Capítulo 5 (Conclusões): Reúne as considerações finais, assinala as contribuições da pesquisa e sugere possibilidades de aprofundamento em trabalhos futuros.

## 2.Desenvolvimento de Aplicações Web com Python e Django

Neste capítulo serão apresentados conceitos necessários para o entendimento do projeto do sistema de provas online e como ele foi construído. A linguagem de programação Python será apresentada (Seção 2.1), assim como o *framework* Django, que conta com melhorias para a linguagem (Seção 2.2).



### 2.1 Python

“Python é uma linguagem de programação de propósito geral, frequentemente aplicada em funções de script. Ela é comumente definida como uma linguagem de script orientada a objetos – uma definição que combina suporte para POO com orientação global voltada para funções de script. Na verdade as pessoas frequentemente usam o termo “script”, em vez de “programa”, para descrever um arquivo de código Python”.

[ Mark Lutz e David Ascher, 2007 ]

Python é uma linguagem orientada a objetos, onde objetos possuem dados próprios e são definidos para estruturar o programa. É uma linguagem de alto nível: isto significa que ao criar um código não será parecido com código de máquina, encontrado em todo computador e representando sequências de instruções, o código será mais parecido com a linguagem humana, onde sua abstração é relativamente elevada. É interpretado, onde o interpretador

executa o código fonte e então o sistema operacional ou processador executa o código fonte novamente.

É uma linguagem de tipificação dinâmica, ou seja, tipos não são associados a variáveis quando são declaradas, pois não são declaradas explicitamente. O tipo da variável varia de acordo com a implementação do sistema, porém sempre assume um único tipo por vez. É imperativo, devido ao uso de sub-rotinas, que permitem dividir o código para uma maior modularização do sistema. Programas imperativos são sequências de comandos para o computador executar.

Uma vantagem de desenvolver em Python é o aumento de produtividade do programador, por ter menos digitação, menos depuração e por seus comandos serem executados imediatamente, sem a necessidade de compilação ou vinculação com outras ferramentas. Além disso, para utilizar um script desenvolvido no Linux, Windows ou Mac, seria necessário somente copiar o programa para a plataforma destino.

De acordo com o site Computerwold<sup>1</sup>, o nome Python foi originado do grupo humorístico Monty Python, criador do programa *Monty Python's Flying Circus*. A linguagem Python e seu interpretador estão disponíveis para diversas plataformas, como Linux, MacOS X e Windows. De acordo com o site da Python Software Foundation<sup>2</sup>, Python foi lançada em 1991 por Guido van Rossum e é gerenciada pela *Python Software Foundation*, uma organização sem fins lucrativos.

O modelo de desenvolvimento da linguagem atualmente é comunitário. O padrão na prática de implementação é em Cpython, escrita em linguagem C, na qual os *bindings*, ligações entre bibliotecas desenvolvidas em C ou C++, podem ser escritos em qualquer outra linguagem diferente de Python. Os *bindings*, ou extensões são usados diretamente no interpretador Python e tem o objetivo de aproveitar a base de código em C/C++ existente em um ambiente, pois os dois sistemas continuarão a existir em C, porém podendo-se fazer uso deles em Python.

Python busca priorizar um código legível ao invés do desempenho. O esforço do programador tem maior enfoque que o esforço computacional, buscando reduzir o primeiro, mesmo que seja necessário aumentar o segundo. A linguagem é capaz de combinações de

---

<sup>1</sup> The A-Z of Programming Languages: Python – fonte retirada do site [http://www.computerworld.com.au/article/255835/a-z\\_programming\\_languages\\_python/?fp=4194304&fpid=1&pf=1](http://www.computerworld.com.au/article/255835/a-z_programming_languages_python/?fp=4194304&fpid=1&pf=1)

<sup>2</sup> Python Software Foundation – fonte retirada do site <http://www.python.org/psf/>

sintaxe com módulos e *frameworks* desenvolvidos por terceiros ou com os recursos de sua biblioteca padrão.

Um dos pontos fortes da linguagem é a qualidade de software. Foi desenvolvida para ter um visual agradável e ser de leitura fácil, usando mais palavras ao invés de pontuações. Em vez de usar delimitadores visuais, como chaves ou palavras reservadas, Python usa espaços em branco e indentação, que é obrigatória, para separação de blocos de códigos. Um novo bloco é identificado pelo aumento da indentação.

Erros de indentação são frequentes ao se programar em Python, sendo facilmente cometidos em editores de texto comuns. Uma forma de prevenir tais erros é configurar o editor de texto utilizado conforme a análise léxica do Python ou utilizar uma IDE que realize indentação automática. Atualmente existe um guia que analisa os arquivos à procura de infrações nas normas de desenvolvimento em Python, o PEP8 (*Python Style Guide Checker*). Utilizando em conjunto com extensões disponíveis nas IDEs, ele pode automaticamente circular todas as linhas que tenham alguma infração ou código desnecessário.

O interpretador Python encontra-se pré-instalado na maioria das versões do Linux. O Mac OS também conta com um interpretador Python pré-instalado, podendo ser atualizado no site oficial da linguagem Python. No caso do Windows, o download deve ser feito pelo site oficial, baixando o arquivo *installer*. Durante o desenvolvimento desse projeto a versão mais recente do Python é 3.2.3, que tem diversas funcionalidades muito diferentes da versão 2.7.5. A versão 2.7.5 é mais utilizada pelos desenvolvedores.

Em interpretadores iterativos, a entrada de comandos pode ser realizada a partir dos *prompts* ('>>>' e '...'). Um segundo *prompt* indica linha de continuação de um comando com múltiplas linhas. Fora esses casos, o que for gerado são saídas do interpretador. Seu uso será para problemas mais simples, sem precisar criar classes. Pode-se testar e modificar um código antes de inseri-lo no programa.

### 2.1.1 Criar um programa

Para criar um arquivo contendo um programa, qualquer editor de texto pode ser utilizado. Ao executar pelo terminal o comando `python` e o nome do arquivo criado pelo



editor de texto, o programa será executado. No caso visto no código 1, o arquivo criado foi `teste.py` e o resultado da execução será `Hello world`.

*Código 1 - Estrutura de um Módulo (arquivo teste.py)*

```
>>> a = "Hello world"
>>> print a
```

### 2.1.2 Pacotes e módulos

Em Python, um módulo é um arquivo que contém instruções da linguagem. Um módulo pode ser executado diretamente ou pode ser dividido em arquivos distintos com funções integradas.

Para criar um módulo em Python basta criar um arquivo `.py` dentro de algum diretório listado em `sys.path`. O `sys.path` contém listas com os possíveis locais onde podem ser criados módulos Python, e então os módulos são importados pelo sistema.

Os arquivos `.pyc` tornam a operação de importar um módulo um pouco mais rápida, pois o python compila o arquivo `.py` e salva em outro arquivo com a extensão `.pyc` para que possam ser referenciados nas próximas chamadas.

Um arquivo `.pyc` contém o *bytecode* de arquivos-fonte compilados, o interpretador python carrega os arquivos `.pyc` antes de arquivos `.py`, caso eles existam, pois pode poupar algum tempo, por não ter que recompilar o código fonte. Este arquivo será útil sempre que um módulo for importado, pois parte do processamento requerido para importar o módulo já foi feito, então não será necessário refazê-lo sempre. O próprio Python se encarrega de criar os arquivos `.pyc`, que são independentes de plataforma. Um exemplo de importação de módulos pode ser visto no código 2, a seguir.

*Código 2 - Exemplo de Importação de Módulos com Sucesso*

```
>>> import pacote.subpacote.modulo
>>> from outro_pacote.outro_subpacote import outro_modulo
```

Pacotes são diretórios no sistema que possuem um arquivo `__init__.py`. Esse arquivo contém o código de inicialização do pacote e a variável `__all__`, que lista os símbolos

importados no comando *from ... import \**. Todos os símbolos do pacote serão importados caso o arquivo `__init__.py` fique vazio. Considere a estrutura apresentada no código 3.

*Código 3 – Estrutura hierárquica de um Pacote*

```
meu_pacote/  
  __init__.py  -- vazio  
  subpacote/  
    __init__.py  -- vazio  
    modulo.py    -- imprime "modulo importado"
```

A partir do diretório `meu_pacote` pode ser realizada uma importação, como mostrado no código 4.

*Código 4 – Import de um Pacote dentro da aplicação*

```
>>> import pacote.subpacote.modulo  
modulo importado
```

Se os arquivos `__init__.py` não existissem, ocorreria um erro como o apresentado no código 5.

*Código 5 – Exemplo de possível erro ao importar um pacote*

```
>>> import pacote.subpacote.modulo  
Traceback (most recent call last):  
File "", line 1, in  
ImportError: No module named pacote.subpacote.modulo
```

### 2.1.3 Classes

A classe é a estrutura fundamental para definir novos objetos. Uma classe pode possuir nome, um conjunto de atributos e métodos. Por exemplo, em um programa que realiza subtrações, seria possível conceber uma classe denominada `Subtracao`, como na implementação da classe no módulo `subtracao.py` apresentado no código 6.

### *Código 6 – Exemplo de estrutura de uma classe*

```
class Subtracao:
    numero_a = None
    numero_b = None

    def __init__(self, numero_a, numero_b):
        self.numero_a = numero_a
        self.numero_b = numero_b
        print "Criando nova instancia Subtracao"

    def calcula_subtracao(self):
        return self.numero_a - self.numero_b
```

Esta classe possui dois atributos, `numero_a` e `numero_b`, que são os valores numéricos da subtração. Existem dois métodos definidos, o método `__init__` e o método `calcula_subtracao`, em ambos o primeiro argumento é uma variável *self*, que é manipulada no interior do método. A variável *self* é um ponto fundamental da sintaxe de Python para métodos: o primeiro argumento é especial e convencionou-se utilizar o nome *self* para ele. O *self* está relacionado à classe em que o método está sendo executado e pode ser utilizado como acesso aos atributos `numero_a` e `numero_b`. O método construtor é o `__init__()`. Quando a classe é instanciada, o método é invocado.

#### 2.1.4 Instâncias

Uma instância é um objeto que será determinado a partir de uma classe. Uma classe especifica objetos, porém sem poder utilizá-los diretamente. Continuando a avaliar o exemplo do código 6, a instanciação de objetos da classe Subtração poderiam ser feitos como no código 7.

### *Código 7 – Instanciação de objetos de uma classe*

```
>>> from subtracao import Subtracao
>>> s1 = Subtracao(1, 2)
Criando nova instância Subtração
```

Pode ser visto neste código que o construtor está atribuindo valores para os atributos `numero_a` e `numero_b`, na instância representada pelo argumento *self*. Agora que a subtração foi instanciada, seus métodos podem ser acessados, como pode ser visto no código 8. Ao final da execução do método uma mensagem é gerada, através do *print*.

#### *Código 8 – Exemplo de acesso ao método Calcula Subtração*

```
>>> print s1.calcula_subtracao()  
1
```

O método executado é o `calcula_subtracao` sobre a instância `s1` da classe `Subtracao`. O Python utiliza parênteses e um ponto após a variável para indicar um método.

#### 2.1.5 Palavras reservadas em Python e suas estruturas

*If*, *else* e *elif* são palavras reservadas correspondente ao grupo de estruturas de seleção, com objetivo de tratar condições de blocos de código a fim de executar expressões determinadas pelos códigos correspondentes.

Em estruturas de repetição são encontradas palavras como *for* e *while*, que percorrem blocos de códigos quantas vezes forem necessárias, realizando ações de acordo com o determinado em cada bloco. São utilizadas quando há listas com mais de um elemento ou quando cada elemento de uma lista deve ter uma ação diferenciada.

*Class* é uma palavra reservada para construção de classes e *def* é reservado para construção de métodos. Ambas palavras são utilizadas na etapa de inicialização de projetos em Python.

*With* serve para construções de escopo como, por exemplo, para adquirir um recurso. É uma estrutura de controle de fluxo, as expressões avaliadas devem resultar em um objeto que suporta o gerenciamento do contexto.

Outras palavras reservadas são *and*, *as*, *assert*, *break*, *continue*, *del*, *except*, *exec*, *finally*, *for*, *from*, *global*, *import*, *in*, *is*, *lambda*, *not*, *or*, *pass*, *print*, *raise*, *return*, *try* e *yield*, todas com uso específico para um desenvolvimento na linguagem e não podem ser usadas para dar nomes a objetos.

### 2.1.6 For em Python

Um *for* tem a função de percorrer os elementos de uma coleção, um a um. Dessa forma, monta uma lista de elementos ordenados, como visto no código 9.

*Código 9 – Exemplo de For percorrendo uma lista de elementos*

```
>>>a = [1, 2, 3, 4]
>>>for i in a:
>>>  print i
1
2
3
4
```

A função *range* tem o objetivo de limitar o número de iterações que um *for* pode realizar, em outras palavras, o número de vezes que vai percorrer a lista, como visto no código 10.

*Código 10 – Exemplo de For com Range*

```
>>>for i in range(1,4):
>>>  print "%o numero" % i
1o numero
2o numero
3o numero
```

### 2.1.7. Listas em Python

As listas são as coleções de objetos mais flexíveis do Python, pois podem conter qualquer tipo de objeto, como números ou letras. O código 11 apresenta exemplos de listas.

*Código 11 – Exemplos de listas em Python*

```
numeros = [1, 2, 3, 4, 5]
letras  = ["um", "dois", "três", "quatro", "cinco"]
```

Listas podem conter listas, como no exemplo apresentado no código 12.

*Código 12 – Exemplo de listas com listas*

```
>>> listas = [numeros, letras]
      [[1, 2, 3, 4, 5], ["um", "dois", "três", "quatro", "cinco"]]
```

Em uma lista, itens podem ser acessados em um ponto específico: basta determinar o número do item na lista. No exemplo do código 13, o elemento acessado é o zero, primeiro elemento da lista. Considerem listas em que o primeiro elemento representa o número zero e tenham n elementos.

*Código 13 – Primeiro item de uma lista*

```
>>> print numeros[0]
      1
```

Além de acessar um elemento específico em uma lista na ordem crescente, também é possível acessar um elemento em ordem decrescente. No caso representado no código 14, o elemento é o último da lista. Para isto, basta indicar que a lista está reversa usando um sinal de menos.

*Código 14 – Último item de uma lista*

```
>>> print numeros[-0]
      5
```

Listas podem acessar somente parte de seus elementos, mais conhecido como “fatias” de uma lista, apresentada no código 15. Essas fatias são fragmentos da lista original, onde dois pontos determinam o início e/ou fim de uma lista. Em casos onde dois pontos vêm antes de um número, significa que esse número será equivalente ao tamanho da lista. Por padrão, é determinado que a lista tenha ordem crescente.

*Código 15 – “Fatias” determinando o final de uma lista*

```
>>> print numeros[:4]
      [2, 3, 4, 5]
```

Quando os dois pontos estiverem após o número, significa que a lista terá início a partir do elemento que representa o número determinado. Os elementos iniciais não estarão contidos em tal lista. Um exemplo pode ser visto no código 16.

*Código 16 – “Fatias” determinando o início de uma lista*

```
>>> print letras[1:]  
["dois", "três", "quatro", "cinco"]
```

Em casos com números antes e depois dos dois pontos, o início e o fim da lista estarão determinados, como no exemplo apresentado no código 17.

*Código 17 – “Fatias” com números de elementos determinado*

```
>>> print numeros[1:3]  
[2, 3, 4]
```

Na linguagem Python há *list comprehensions*, que são caracterizadas por listas construídas de forma similar a que matemáticos costumavam fazer. *List comprehensions* são equações formadas com comandos *if* e *for*. Caso as regras de construção sejam muito complicadas ou tenham outras declarações, é melhor usar os comandos *map* ou *filter*. Alguns exemplos de *list comprehensions* podem ser vistos no código 18.

*Código 18 – Exemplo de list comprehensions*

```
>>> print [i for i in range(8)]  
[0, 1, 2, 3, 4, 5, 6, 7]  
ou  
>>> print [i for i in range(15) if i%2 == 0]  
[0, 2, 4, 6, 8, 10, 12, 14]
```

As equações são lidas de dentro para fora: primeiramente deve-se avaliar o *if*, em seguida o *for* e então o *print*, que é representado pelo *i*. Outra forma de ler a segunda equação apresentada no código 18 é vista no código 19.

### Código 19 – Outra forma de ler uma *list comprehension*

```
if i%2 ==0:
    for i in range(15):
        print i
```

No código 20 pode ser visto exemplos de *list comprehension* retirados do site oficial do Python Brasil.

### Código 20 – Exemplos de *list comprehensions*

```
>>> nums = [1,2,3,4]
>>> fruit = ["Apples", "Peaches", "Pears", "Bananas"]
>>> print [(i,f) for i in nums for f in fruit]
[(1, 'Apples'), (1, 'Peaches'), (1, 'Pears'), (1, 'Bananas'),
 (2, 'Apples'), (2, 'Peaches'), (2, 'Pears'), (2, 'Bananas'),
 (3, 'Apples'), (3, 'Peaches'), (3, 'Pears'), (3, 'Bananas'),
 (4, 'Apples'), (4, 'Peaches'), (4, 'Pears'), (4, 'Bananas')]

>>> print [(i,f) for i in nums for f in fruit if f[0] == "P"]
[(1, 'Peaches'), (1, 'Pears'),
 (2, 'Peaches'), (2, 'Pears'),
 (3, 'Peaches'), (3, 'Pears'),
 (4, 'Peaches'), (4, 'Pears')]

>>> print [(i,f) for i in nums for f in fruit if f[0] == "P" if i%2 == 1]
[(1, 'Peaches'), (1, 'Pears'), (3, 'Peaches'), (3, 'Pears')]
```

#### 2.1.8 Conjuntos em Python

Ao criar conjuntos pode-se ter certeza que não haverão dados repetidos, pois essa é uma das características de um conjunto. Outra característica é não ter ordem. Um conjunto é determinado pelo comando *set* antes dos colchetes. Ao passar uma *string*, como visto no código 21, cada número será um elemento do conjunto.



### *Código 21 – Criação de um conjunto*

```
>>> conjunto1 = set('2468')
>>> conjunto1
set(['8', '2', '4', '6'])
```

Em conjuntos podem-se usar operadores como os apresentados nos códigos 22, 23, 24 e 25 que respectivamente tem os objetivos de comparar blocos de conjuntos visando uma resposta verdadeira ou falsa; de realizar uniões de dois blocos montando um bloco com todos os números presentes em ambos os blocos; de procurar por valores comuns a dois blocos diferentes; e de criar um bloco com exceções.

### *Código 22 – Comparação de Conjuntos*

```
>>> conjunto1 <= set('123456789')
True
>>> conjunto1 >= set('123456789')
False
```

### *Código 23 – União de Conjuntos*

```
>>> conjunto2 = set([3,3,3,3,4,4,4,8,8,8,8,9])
>>> conjunto1.union(conjunto2)
set([3, 4, 8, 9, '2', '4', '6', '8'])
>>> conjunto1 | conjunto2
set([3, 4, 7, 8, 9, '2', '4', '6', '8'] )
```

### *Código 24 – Interseção de Conjuntos*

```
>>> set('789').intersection(set('567'))
set(['7'])
>>> set('345') & set(567)
set(['5'])
```

### *Código 25 – Diferença de Conjuntos*

```
>>> set('4567').difference(set('5'))  
      set(['4', '6', '7'])  
>>> set('2345') - set('34')  
      set(['2', '5'])
```

### 2.1.9 Dicionários

Dicionários são coleções sem ordenação de objetos, os itens são buscados por chaves, já que dicionários guardam os pares chave/valor como visto no código 26.

### *Código 26 – Estrutura de dicionários, pares chave e valor*

```
dicionario = {chave: valor}
```

Para gerar um dicionário pode-se determinar os valores por meio de estruturas e usar o comando *dict*. No código 27 pode ser visto um exemplo de uma lista que será transformada em dicionário, onde a variável *dicionario* é uma lista com uma estrutura chave/valor e será transformada em um dicionário por meio do comando *dict*.

### *Código 27 – Exemplo de uma lista sendo transformada em dicionário*

```
>>> dicionario = [('nome', 'marina'), ('idade', '22'), ('id', '8')]  
>>> d = dict(dicionario)  
>>> d  
      {'nome': 'marina', 'idade': 22, 'id': 8}
```

Para iterar um dicionário e ter acesso a seus elementos, deve-se relacionar o dicionário ao comando *for* e usar as funções *keys* ou *values*, como visualizado no código 28.

### *Código 28 – Iterações em Dicionários*

```
>>> dicionario = {'nome': 'marina', 'idade': 22, 'id': 8}
>>> for dict in dicionario.keys():
>>>     print "Chave: %s %dict
Chave: nome
Chave: idade
Chave: id
```

Para manusear um dicionário, basta determinar a chave da informação que deseja buscar, como apresentado no código 29.

### *Código 29 – Manuseando dicionários*

```
>>> dicionario['idade']
22
```

Caso a chave não exista, um erro será notificado, como o erro visto no código 30.

### *Código 30 – Erro caso um valor da tabela não esteja no dicionário*

```
>>> dicionario['curso']
Traceback (most recent call last):
File "", line 1, in ?
KeyError: 'profissao'
```

Para evitar erros é aconselhável usar o método *get*, que retorna 'None' em vez de um erro, isso significa que no dicionário correspondente não há uma chave com o dado informado, como apresentado no código 31.

### *Código 31 – Evitando Erros com o comando get*

```
>>> print dicionario.get('curso')
None
```

Assim como o *get*, em dicionários há outros métodos que podem ser utilizados, como *del* para apagar uma chave ou elemento, *pop* para apagar um elemento ou chave, porém

visualizando o que está sendo apagado, *clear* para limpar o conteúdo de um dicionário e *copy* para copiar o conteúdo de um dicionário. No código 32, pode ser visto um exemplo de formatação de *strings* onde uma *string* é montada com valores de um dicionário.

*Código 32 – Formatação de strings através de elementos de dicionários*

```
>>>dicionario = {'nome': 'marina', 'faculdade': 'unirio'}
>>>sentenca = 'meu nome e %(nome)s e faço faculdade na
%(faculdade)s'
>>>print sentenca % dicionario
meu nome e marina e faço faculdade na unirio
```

#### 2.1.10 Built-in Functions

As funções mais simples do Python são acessadas sem ser necessário importar módulos do programa e estão contidas na instalação da linguagem ao sistema operacional, são conhecidas como *built-in functions*. As *built-in functions* mais usadas estão listadas na figura 1.

*Figura 1 - Built-in Functions mais usadas*

Built-in Functions				
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	apply()
delattr()	help()	next()	setattr()	buffer()
dict()	hex()	object()	slice()	coerce()
dir()	id()	oct()	sorted()	intern()

Um exemplo de *built-in function* é o *range* que gera listas com progressões aritméticas, como visto no código 33. Essa função é frequentemente usada em *loops*, pois realiza iterações sobre uma sequência de números inteiros.

*Código 33 – Um exemplo da função range, um Built-in function*

```
>>>>range(10)
      [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

## 2.2 Django

Django é um *framework* Python escrito com o intuito de não repetir códigos ao desenvolver um sistema e permitir o desenvolvimento rápido de projetos, devido ao fato de algumas aplicações já terem sido previamente utilizadas.

De acordo com o site oficial do Django<sup>1</sup>, o nome django foi inspirado no músico de jazz Django Reinhardt e foi criado no Lawrence Journal-World, para gerenciar o site jornalístico na cidade de Lawrence, no Kansas.

Este *framework* tem um padrão de três camadas. A primeira são os Models, onde classes são criadas e objetos descritos para posteriormente serem guardados no banco. A segunda camada é a de Templates, onde pode ser visto toda a parte de html do projeto. A última camada é a View, onde classes serão instanciadas, lógicas serão criadas e métodos serão montados para posteriormente serem referenciados nas páginas que montam as URLs. Assim, cada método será desenvolvido para uma página especificamente. Essas camadas são chamadas de MTV: model, template e view. As principais características do Django são:

- Mapeamento Objeto-Relacional (ORM): esse mapeamento significa que a partir dos atributos determinados em cada classe Python, o banco de dados poderá ser modelado. Não será necessário utilizar o banco diretamente. As tabelas serão criadas e manipuladas pelas classes. Bancos são manipulados por prompts de comando;
- Interface Administrativa: o Django é capaz de criar uma interface administrativa padrão, montada com os objetos relacionados nas classes. Para ativá-lo, basta realizar algumas

---

<sup>1</sup> Django - <https://www.djangoproject.com/>

configurações em arquivos padrões do sistema e a interface estará montada. Para criar uma interface é extremamente fácil, porém para modificá-la existem algumas dificuldades, por seus elementos serem padrões;

- Formulários: além da interface administrativa, formulários também podem ser gerados pelos objetos do banco;
- URLs Elegantes: no Django pode-se determinar exatamente de que forma o programador deseja que as URLs sejam montadas, sem limitações. São criadas expressões regulares para inserir números, letras e até mesmo títulos relacionados às páginas URL. Tudo realizado de forma muito simples;
- Internacionalização: em uma das páginas de configuração do Django pode se encontrar suporte a diversos idiomas.

### 2.2.1 Template tags e Template Filters

Template tags e template filters são ferramentas que utilizam parte da lógica de desenvolvimento criada em métodos, na camada de views, em arquivos html ou xhtml presentes na camada de templates. Ao utilizar listas ou variáveis do desenvolvimento de métodos em templates, pode-se estender tags e filtros.

Para entender melhor, apresentaremos a forma que variáveis são utilizadas em páginas html. Para criar um template filter deve-se utilizar o formato de chaves duplas (`{{...}}`), de forma que qualquer conteúdo das variáveis passadas para os templates poderá ser utilizado nas estruturas html.

Os template filters utilizam as variáveis das views e inserem lógicas de acordo com o tipo de dado das variáveis, como por exemplo, é possível limitar o número de valores de uma lista ou determinar o formato da data a ser exibida no html, entre outros. Os códigos 34 e 35 irão apresentar exemplos práticos de template filters.

*Código 34 – Exemplo de um template filter com função date*

```
{{ publicacao|date:"d/M/Y" }}
```

Template filters usam o pipe (|) para representação de suas funções. No código 34, o elemento publicação terá a sua data formatada. Já no código 35, a lista artigo será limitada ao primeiro elemento.

### Código 35 – Exemplo de um template filter com função length

```
{% if artigo | length > 1 %}
...
{% else %}
...
{% endif %}
```

Para realizar *loops* de listas oriundas das lógicas de programação realizadas nas views, deve-se utilizar as *template tags* que iniciam e terminam com chaves e porcentagem ({%...%}). O código 36 apresenta um exemplo de um laço de repetição.

### Código 36 – Laço de repetição em Templates Tags

```
{% for artigo in latest %}
...
{% endfor %}
```

No caso acima, a variável latest pode ser considerada uma lista contendo mais de um elemento, sendo necessário então criar um *loop* para mostrar todos os valores contidos na lista. A variável artigo será criada em tempo de execução, no momento que o comando *for* for utilizado dentro do *template*. Toda utilização de *template tag* deve ser fechada ao fim da lógica, através da estrutura {% endfor %}.

*Template tags* são utilizadas para repetições, assim como podem ser utilizadas em condições, através do comando *if*, como pode ser visto no código 37.

### Código 37 – Condição em Templates Tags

```
{% if artigo %}
...
{% else %}
...
{% endif %}
```

Em templates pode-se utilizar *for* e *if* ao mesmo tempo, de acordo com a lógica determinada para o sistema. O código 38 apresenta um exemplo do uso desse conceito.

### *Código 38 - Condição e Repetição junta na mesma lógica de desenvolvimento*

```
{% if data_atual %}
    {% for artigo in latest %}
        ...
    {% endfor %}
{% endif %}
```

Existem diversas referências à *template tags* e *template filter*. Acima, foram mostrados apenas alguns exemplos mais utilizados ao desenvolver em templates, porém a abrangência de elementos em templates é muito grande. Todas as referências podem ser vistas na documentação oficial do Django<sup>1</sup>.

## **2.3 Considerações Finais**

Nesse capítulo foi mostrado um pouco sobre o Python. Alguns dos exemplos apresentados foram: como criar classes e listas. Python é uma ótima ferramenta de trabalho e é muito utilizada no ambiente empresarial voltado para aplicativos web e aplicativos móveis. Foram mostrados alguns exemplos de possíveis desenvolvimentos em Python para tais aplicativos.

A utilização do *framework* Django facilita o desenvolvimento e simplifica a criação de aplicações web. Sua manutenção, um ponto crucial para quem precisa produzir muito em pouco tempo, também é de fácil alteração por ter a doutrina de não repetir código em diversos arquivos. A manutenção em um bloco de código será facilmente dissipada a outros arquivos do sistema.

No próximo capítulo será mostrado o sistema que será montado nesse trabalho: o sistema de provas online.

---

<sup>1</sup> Templates Django – <https://docs.djangoproject.com/en/dev/topics/templates>

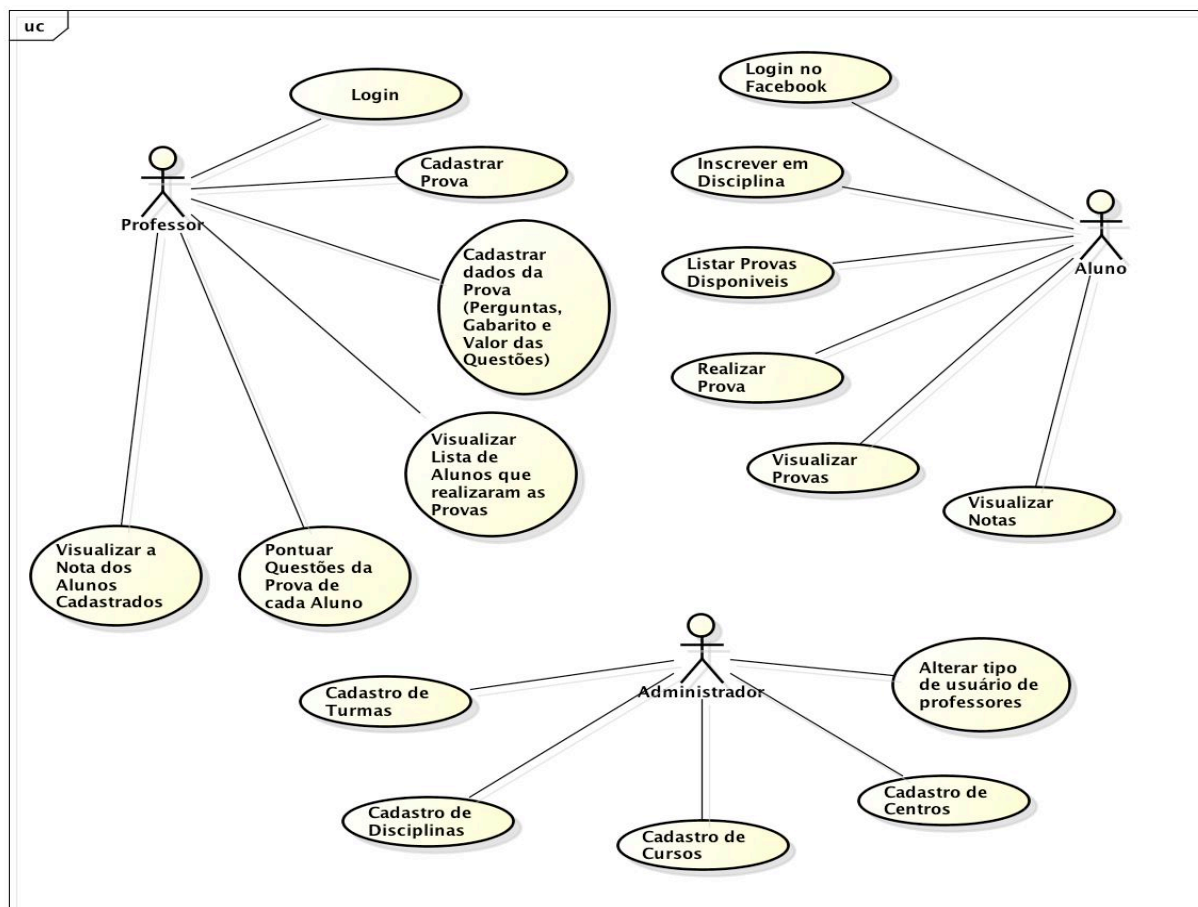


## 3. Técnicas Utilizadas no Desenvolvimento do Sistema de Provas Online

Neste capítulo serão apresentados os requisitos e as técnicas utilizadas no desenvolvimento do sistema de provas online. Serão mostrados os casos de uso (Seção 3.1), a modelagem conceitual (Seção 3.2), algumas funcionalidades do sistema (Seção 3.3) e, por fim, a integração do projeto com Facebook (Seção 3.4).

### 3.1 Casos de Uso da Aplicação

Figura 2 – Casos de Uso



No diagrama apresentado na figura 2, referente ao sistema de provas online, podem ser vistos os atores Professor, Aluno e Administrador. O primeiro terá como objetivo montar Provas com questões referentes às disciplinas que leciona e disponibilizar aos alunos até uma determinada data. Após o término do tempo para realização da prova, o ator em questão corrige a prova de cada aluno cadastrado. O ator Aluno irá realizar a prova montada pelo professor, dentro da data delimitada. Já o Administrador tem a obrigação de cadastrar dados referentes aos cursos, centros, turmas e alterar o status de professores no sistema.

Abaixo, pode ser visto detalhadamente os casos de uso do ator Professor, com uma breve explicação do que será feito em cada caso de uso.

- Login - O professor deverá ser capaz de acessar o sistema através de seu *login* e senhas previamente cadastrados;
- Cadastrar Prova – O professor terá a possibilidade de montar provas no sistema para cada disciplina que estiver lecionando. Podem ser criadas diversas provas, porém cada uma deve ter uma identificação;
- Cadastrar Dados da Prova (Perguntas, Gabarito e Valor das Questões) – Após ser cadastrada uma prova que tenha uma disciplina correspondente, o professor será capaz de inserir as perguntas, respostas e o valor de cada questão na prova;
- Visualizar Lista de Alunos que realizaram as Provas – Ao final do período para realização da prova, o professor terá acesso a listagem respostas de cada questão e dos alunos que responderam as questões;
- Pontuar Questões da Prova de cada Aluno – Após o aluno responder às questões da prova e enviá-la, o professor poderá dar uma nota a cada questão respondida;
- Visualizar Nota do Aluno Cadastrado – Por fim, o professor tem a possibilidade de visualizar as notas de provas dos alunos.

Abaixo, listamos os casos de uso do Aluno.

- Login no Facebook – O aluno deverá acessar o sistema através de seu e-mail e senha pessoal cadastrados no Facebook. Caso não tenha uma conta, poderá criá-la pelo sistema, onde deverá acessar o aplicativo aceitando os termos de utilização;

- Inscrever em Disciplina – O aluno deverá se inscrever nas disciplinas que tiver cursando para que o sistema tenha um controle de alunos cadastrados em uma disciplina;
- Listar Provas Disponíveis – Depois de inscrito em disciplinas, o aluno visualizará todas as provas que estejam dentro do prazo de realização;
- Realizar Prova – Caso a prova não tenha ultrapassado a data limite de realização, o aluno terá a possibilidade de responder as questões e enviá-la ao sistema;
- Visualizar Provas – O aluno poderá visualizar provas passadas, contanto que tais provas tenham ultrapassado a data de realização e suas notas tenham sido inseridas no sistema;
- Visualizar Notas – Aluno terá acesso a notas de provas realizadas e corrigidas.

Por fim, as ações do ator Administrador serão apresentadas:

- Cadastrar Centros – Serão cadastrados todos os cursos de uma faculdade que poderão acessar o projeto. Tais dados não poderão ser alterados por atores além do administrador;
- Cadastrar Cursos – O administrador deverá entrar com informações referentes à faculdade, como os dados dos cursos que estiverem alocados em centros do sistema;
- Cadastrar Disciplinas – Ao cadastrar os cursos, serão relacionadas as disciplinas existentes em cada centro de ensino;
- Cadastrar Turmas – As turmas serão determinadas pela diretoria do curso e em cada período tem uma gama de turmas abertas com seus respectivos professores;
- Alterar tipo de usuário de professores – Assim que um professor se cadastrar no sistema por meio do Facebook, o administrador deverá indicar que esse usuário é um professor, alterando seu tipo de usuário diretamente da listagem de usuários.

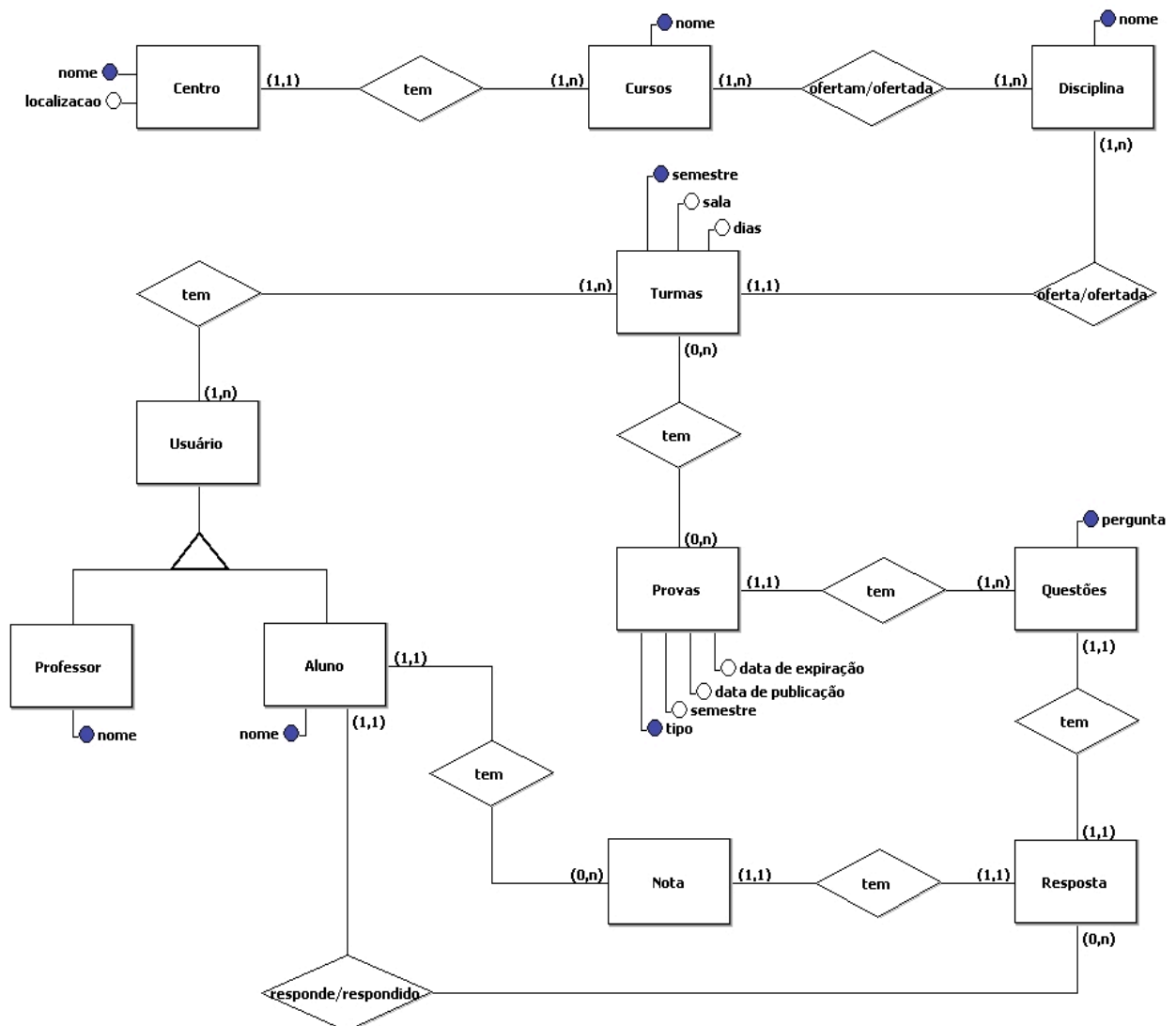
### ***3.2 Modelagem Conceitual***

Na figura 3 pode ser visualizado o diagrama referente à modelagem conceitual do projeto, onde é analisado um conjunto de conceitos integrados baseados em atividades e objetos do sistema, de acordo com as necessidades do usuário. A imagem é o produto da observação da navegação pelo sistema traduzido em uma linguagem gráfica.

Na linguagem gráfica retângulos representam entidades; losangos representam relacionamentos de entidades; atributos são representados por círculos; e linhas que ligam entidades são linhas de conexão.

Cada entidade tem sua respectiva cardinalidade em uma relação, que representa o número de possíveis relações entre as entidades. As cardinalidades podem ser de um para um, de um para muitos, de nenhum para muitos e de muitos para muitos. Um exemplo de cardinalidade seria o de disciplinas que podem ter uma ou várias turmas relacionadas em um período, porém uma turma só pode ter uma disciplina relacionada por semestre.

Figura 3 - Modelagem Conceitual do Sistema de Provas Online



Na imagem, também são apresentados os atributos de cada entidade, por exemplo, cada turma tem um semestre, sala e dias, sendo que o semestre é o atributo principal, que identifica cada turma.

A ferramenta utilizada para criar o modelo de entidades e relacionamentos é o brModelo, onde é possível montar uma imagem gráfica com entidades e atributos. Uma relação que pode ser traduzida em imagem é a especialização exclusiva, como pode ser visto na relação de usuários com alunos e professores.

Na figura 3 será apresentada as seguintes relações entre as entidades do sistema.

- Centro e Cursos – Um centro pode ter um ou mais cursos associados. Cursos, por sua vez, só podem ter relação com um centro. Centros têm nome e localidade como atributos. Nome é o atributo principal;
- Cursos e Disciplina – Cursos ofertam uma ou várias disciplinas por período, assim como disciplinas são ofertadas por um ou diversos cursos. Cursos têm um atributo identificador: o nome;
- Disciplina e Turmas – Uma disciplina pode ter diversas turmas por período. Porém uma turma teria apenas uma disciplina relacionada. Disciplinas têm um nome, que é o atributo identificador de uma entidade;
- Turmas e Usuário Professor – Um professor pode lecionar a uma ou várias turmas, assim como turmas podem ter um ou diversos professores ensinando a matéria do período. Professor tem um atributo identificador: o nome;
- Turmas e Usuário Aluno – Cada aluno pode se matricular em diversas turmas, assim como turmas podem ter vários alunos matriculados, de acordo com o limite da turma. Aluno é identificado por seu atributo identificador: o nome;
- Turmas e Prova – Uma turma pode ter diversas provas ao longo do período, de acordo com o determinado pelo professor, ou pode não ter provas criadas no sistema, tendo como nota apenas trabalhos. Uma prova está relacionada a uma turma. Semestre, sala e dias são atributos de turmas, sendo que o semestre é o atributo principal;

- Prova e Questões – Uma prova tem uma ou mais questões. Porém, cada questão deve estar relacionada a somente uma prova. Prova tem como atributos tipo, semestre, data de publicação e data de expiração, onde tipo é um atributo identificador;
- Questões e Resposta – Cada questão tem uma resposta, assim como cada resposta tem uma questão correspondente. O atributo identificador de Questões é pergunta;
- Resposta e Nota – Uma nota será dada de acordo com a resposta da questão, portanto, cada resposta terá uma nota. A relação entre respostas e notas é de um para um, onde para cada nota há uma resposta.
- Nota e Aluno – Nota se relaciona com aluno de forma que um aluno pode ter diversas notas, porém uma nota só pode estar relacionada a um aluno, independente de existirem diversas notas com o mesmo valor.
- Resposta e Aluno – A relação de resposta e aluno é feita de forma que uma resposta está relacionada a um aluno, enquanto um aluno pode criar diversas respostas para diferentes perguntas.

### ***3.3 Funcionalidades do Sistema***

#### *3.3.1 Apps*

Uma boa prática de desenvolvimento ao criar um sistema é dividir partes do código em diretórios, conhecidas como *apps*, com o intuito de manter de forma organizada as diversas áreas que um sistema pode ter.

*Código 39 – Criação de Apps*

```
python manage.py startapp prova
```

O comando do código 39 vai criar um diretório com o nome prova no sistema de Provas Online. Essa estrutura contém arquivos padrão do sistema, como pode ser visto no código 40.

Todos os *apps* criados pelo comando do código 39 terão os mesmos arquivos gerados, porém cada conjunto de arquivos terá o seu *app* correspondente, com desenvolvimento diferenciado de acordo com as necessidades do sistema.

#### Código 40 – Estrutura de uma App

```
prova/  
  __init__.py  
  models.py  
  views.py  
  url.py
```

O código 41 mostra a estrutura do sistema de Provas Online que consta de dois módulos, um para organização da parte referente à administração e outro com o diretório core, que contém todo o desenvolvimento do sistema. Dentro da pasta core serão inseridos os *apps* geral, prova, sistema, static e templates que foram criados manualmente.

#### Código 41 – Estrutura de apps do sistema de Provas Online Unirio

```
prova-unirio/  
  admin/  
    img/  
    css/  
    js/  
  core/  
    geral/  
    prova/  
    sistema/  
    static/  
    templates/
```

Na pasta core, pode ser visto o diretório geral que contém os arquivos settings.py e urls.py. No arquivo settings.py, criado pelo comando de inicialização de um projeto em django, está contida a parte do código referente ao *installed apps* utilizada para modelar o banco de dados.

Ao realizar o comando visto no código 42, o settings terá a área de *installed apps* percorrida por inteiro e serão criadas tabelas no banco de dados de acordo com cada *app* presente nos *installed apps*. Só poderão estar na área de *installed apps* os *apps* nativos do sistema ou *apps* previamente criados pelo comando do código 39. Assim todos os *apps* do sistema de Provas Online do código 41 estarão modelados no banco de dados do projeto.

#### *Código 42 – Comando para criar estrutura no banco de apps*

```
python manage.py syncdb
```

Conforme as tabelas são criadas, ao final de cada estrutura montada, é apresentada uma mensagem na tela de comando informando se a tabela foi criada com sucesso ou não. Assim que o banco for atualizado com todas as *apps* do sistema, uma mensagem final apresentará ao desenvolvedor a opção de criar uma conta de super-usuário, onde será criado o *login* e senha referente ao acesso à página de administrador.

#### *3.3.2 Models*

Ao detalhar melhor as funcionalidades do sistema, devem ser mencionados os Models, classes que mapeiam as tabelas do banco de dados contendo campos e comportamentos essenciais aos dados.

Retornando ao código 40, o arquivo models.py contém os campos essenciais aos dados que estão sendo armazenados. Um exemplo de estrutura de atributos dos *models* pode ser visto no código 43, essa estrutura corresponde à parte do código presente na classe Prova.

#### *Código 43 – Exemplos de classes presentes no arquivo models.py*

```
from django.db import models

class Prova (models.Model):
    turma = models.ForeignKey(Turma, on_delete=models.PROTECT)
    semestre = models.FloatField(max_length=5, null=True,
verbose_name="Semestre")
```

Na classe Prova, as variáveis turma e semestre são atributos do modelo. Ao criar um atributo no modelo, é obrigatório determinar qual opção de *field* esse atributo terá. O *field* é o correspondente ao tipo da variável. Os *fields* de turma e semestre são *ForeignKey* e *FloatField*, respectivamente, o que indica que turma é um atributo com relacionamento muito para muitos e requer a classe à qual o modelo está relacionado, no caso a classe turma. Já o segundo atributo representa um número de ponto flutuante com no máximo 5 dígitos.



Cada atributo da classe é mapeado em uma coluna no banco de dados. A classe Prova herda *models.Model* pois a arquitetura do django permite entender que é uma abstração do banco, que o elemento está sendo criado a partir de um modelo.

Em django, alguns exemplos de atributos são: *ImageField* que representa qualquer imagem em *JPG*, *PNG* ou *Bitmap*; um atributo *DateField* que contabiliza dia e hora; ou um *UrlField* que identifica se o campo inserido é originalmente uma URL.

Todos os *fields* têm características específicas, como o *verbose\_name*, que determina o nome que o atributo terá na página de administração; a variável *null* igual a *True*, caso o atributo possa ser salvo como nulo no banco de dados; em caso de textos e números, determinar o tamanho máximo de caracteres que podem ser inseridos ou o número máximo de dígitos que podem ser preenchidas pelo *max\_length*. Somente alguns *fields* têm especificações obrigatórias; a maioria é opcional.

Outro ponto a ser destacado é a classe meta. De acordo com as boas práticas de programação, todas classes de modelos devem ter ao menos uma classe filha: a classe meta. Essa classe tem o propósito de manter todos os programadores cientes do local exato onde devem alterar nomes de *apps* e manter o código mais organizado. Nessa classe são considerados os casos de *apps* no plural e no singular e é determinado qual atributo da classe irá manter a ordenação das listas de objetos apresentadas na página de administração em um determinado app. Um exemplo pode ser visto no código 44.

*Código 44 – Exemplo de uma classe meta, presente em classes do modelo*

```
class Meta:
    ordering = ['turma']
    verbose_name = u'Prova'
    verbose_name_plural = u'Provas'
```

### 3.3.3 Views

Os arquivos referentes às views são muito utilizados em django, por serem os arquivos responsáveis por terem as lógicas de desenvolvimento do projeto. Cada view contém métodos que são desenvolvidos com o objetivo de realizar um comportamento para uma determinada URL.

No código 45 pode ser visto o método aluno, presente no arquivo views.py. Pode ser visto na declaração do método que é passado como parâmetro a variável fb\_id, um parâmetro importado de uma URL para ser tratado na lógica de programação.

Presente no código está a declaração da variável lista, a instanciação da classe turma e da classe usuário, a iteração de uma lista filtrada pelas turmas presentes na classe Prova, um try/except para a realização de tratamento do campo prova\_aluno, caso tal variável não tenha valores e por fim a introdução de elementos em uma lista.

*Código 45 – Código referente a um método presente em uma view*

```
def aluno(request, fb_id):
    lista = []
    turma = Turma.objects.filter(aluno)
    aluno = Usuario.objects.get(fb_id=fb_id)
    for prova in Prova.objects.filter(turma=turma):
        try:
            prova_aluno = prova.aluno.get(aluno=aluno)
        except:
            prova_aluno = None

        lista.append({
            'prova': prova, 'professor': prova.professor.nome,
            'publicacao': prova.publicacao.strftime("%D:%M"),
        })

    if request.method == 'POST':
        formset = AlunoDisciplinaForm(request.POST)
    else:
        formset = AlunoDisciplinaForm()

    context = {
        "lista": lista,
        "formset": formset,
    }
    return render_to_response("aluno/adminAluno.html", context)
```

No projeto de provas online, pode ser visto o tratamento de formulários nos arquivos das views através do campo `formset`. Esse campo irá gerar um *form* do modelo `AlunoDisciplinaForm`, contendo todos os seus atributos declarados. No código 45 o *form* tratado utiliza o método *POST*, porém em outros casos pode-se utilizar o método *GET*.

No final do método da view é utilizado o `return`, que pode ser do tipo `render_to_response` ou `render` e deve conter o caminho onde o arquivo HTML está localizado e os valores que foram tratados na view e posteriormente serão utilizadas nas páginas HTML, através de um dicionário. No código 45 esse dicionário é o `context`.

### 3.3.4 URL's

O arquivo `urls.py` contém as URLs que serão utilizadas no projeto. Um esquema de URLs limpas e elegantes é importante em uma aplicação Web. Para criar URLs para um sistema, um módulo python é criado: o `URLconf`, que é um mapeamento simples entre padrões de URL.

Todas as URLs são formadas de expressões regulares e são iniciadas com `'url'`, para importar o mapeamento da `URLconf`, e entre parênteses o caminho<sup>1</sup>, a classe do arquivo de programação correspondente e um nome que poderá ser referenciado em outras partes do projeto, nessa exata ordem.

Como foi dito anteriormente, URLs são expressões regulares, portanto o caminho das URLs tem letras e interrogações, para que tenham parâmetros diferenciados ao longo do caminho. Os parâmetros podem ser opcionais ou não. O início e o fim de uma URL são marcados pelos sinais `^` e `$`, respectivamente.

No código 46 pode ser visto um arquivo de URLs do projeto apresentado. Inicialmente em um arquivo deve-se indicar de onde virão as lógicas de programação relacionadas à URL, no caso apresentado será da `app prova` e os códigos relacionados à lógica estão no arquivo `views`. Em seguida deve-se montar as URLs. As palavras `aluno` e `provas` representam o início do caminho da URL. Os parâmetros `'fb_id'` e `'id'`, que representam o `id` da pessoa que está acessando o sistema no Facebook e o `id` da prova, respectivamente, esses dados indicam que

---

<sup>1</sup> caminho: especifica o local (geralmente num sistema de arquivos) onde se encontra o recurso dentro do servidor.

na URL os parâmetros são obrigatórios. Aluno e provas, entre plicas e vírgulas, são a representação das views e os dados de 'name' representam o nome que as URLs serão tratadas dentro do desenvolvimento.

*Código 46 – Exemplo de uma página URL utilizada no projeto*

```
from django.conf.urls.defaults import patterns, url

urlpatterns = patterns('prova.views',

    url(r'^aluno/(?P<fb_id>.*)/$', 'aluno', name='aluno'),
    url(r'^provas/(?:(?P<fb_id>[\w_-]+)/(?:(?P<id>\d+)/))$',
    'provas', name='prova'),
)
```

Quando um usuário acessa uma página do projeto, o sistema utiliza um algoritmo para determinar qual código python irá executar até chegar à URL determinada. Para acessar a página a partir da URL enviada, o django percorre cada URL presente no arquivo 'urls.py', em ordem, e para no primeiro que corresponder a URL solicitada. Quando uma das expressões regulares for correspondente à URL, o python importa a view relacionada no caminho desta URL e acessa a lógica de programação da view, no final da lógica estará apresentada a estrutura HTML correspondente à URL solicitada, formando então a página solicitada.

### *3.3.5 Templates*

Os arquivos em HTML do sistema são mais conhecidos como telas de Template. Essas telas contém todo o código HTML necessário para montar a estrutura de uma página, porém são simplesmente arquivos de texto. As views determinam qual lógica será imposta em cada página HTML em seu return, geralmente a última linha em cada método da view.

Além do código em HTML, os templates contam com as *template tags* e *template filters* do Django em seus arquivos. Um template contém variáveis, que são substituídas por valores, e *tags*, que controlam a lógica do modelo.

*Template filter* é como uma função python, que recebe dois argumentos: o valor de uma variável e o valor de um argumento. Para aplicar filtros, basta utilizar o pipe (|) que separa a variável e o argumento, como no exemplo `{{ algumalista | slice: "2" }}`. O intuito é modificar as variáveis para exibição no sistema, como no exemplo acima, a lista pode ter diversos elementos, porém só os dois primeiros serão mostrados no HTML. Filtros devem retornar algo, caso contrário retornam *strings* vazias, pois não lançam exceções.

As *template tags* tem diversas funcionalidades, algumas criam texto de saída, outras controlam fluxo, executando *loops* ou lógicas. Também podem carregar informação externa para o template com o intuito de serem usadas posteriormente em variáveis. *Template tags* começam e terminam com `{% %}`. No código 47 pode ser visto um exemplo.

*Código 47 – Exemplo de template tags utilizada no projeto*

```
<div class="cadastradas">
  <p>Disciplinas Cadastradas</p>
  <ul>
    {% if turmas %}
      {% for aula in turmas %}
        <li>{{ aula.disciplina__nome }}</li>
      {% endfor %}
    {% else %}
      <li>Nenhuma aula cadastrada</li>
    {% endif %}
  </ul>
</div>
```

Todas as telas utilizadas no sistema tem um esqueleto correspondente em HTML no projeto. Um exemplo de como *template filters* é utilizado no projeto é visto na linha de código apresentada no código 48. O primeiro *template filter*, *forloop.counter*, realiza uma contagem iniciando por 1. As *template filters* *perguntas.pergunta* e *perguntas.valor*, identificam na lista pergunta os campos pergunta e valor, apresentando na tela.

*Código 48 – Exemplo de Template Filters*

```
<p>{{ forloop.counter }} - {{ perguntas.pergunta }} ( Valor: {{
perguntas.valor }} )</p>
```

### 3.3.6 Página de Administração

Um diferencial em Django é a página de Administração. No projeto de Provas Online pode ser visualizada através da página de acesso ao sistema para professores e administradores.

Por padrão não é ativada, mas é facilmente ativada com algumas alterações no código. Trata-se de uma página opcional que acessa o arquivo `settings.py` e a partir das aplicações presentes na área *installed apps* deste arquivo, percorre os *apps* do sistema, comparando os *apps* encontrados aos presentes no *installed apps*. Caso a comparação esteja correta, procura pelo arquivo `models.py` de cada app e acessa os *fields* contidos nesse arquivo, mostrando tais informações na página web de administração.

Para que seja possível ver a página de administração, primeiramente deve-se alterar a área de *installed apps* no `settings.py`, adicionando *django.contrib.admin*, e rodar o comando `python manage.py syncdb` para que seja refletida a alteração no banco. Em seguida alterar a página `url.py` do app correspondente, habilitando a linha referente a URL da página de administração, como pode ser visto no código 49, na penúltima linha do código.

*Código 49 – url.py gerado pelo sistema*

```
from django.conf.urls.defaults import *

from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    # Descomentar a próxima linha para habilitar o admin:
    #(r'^admin/(.*)', admin.site.root),)
```

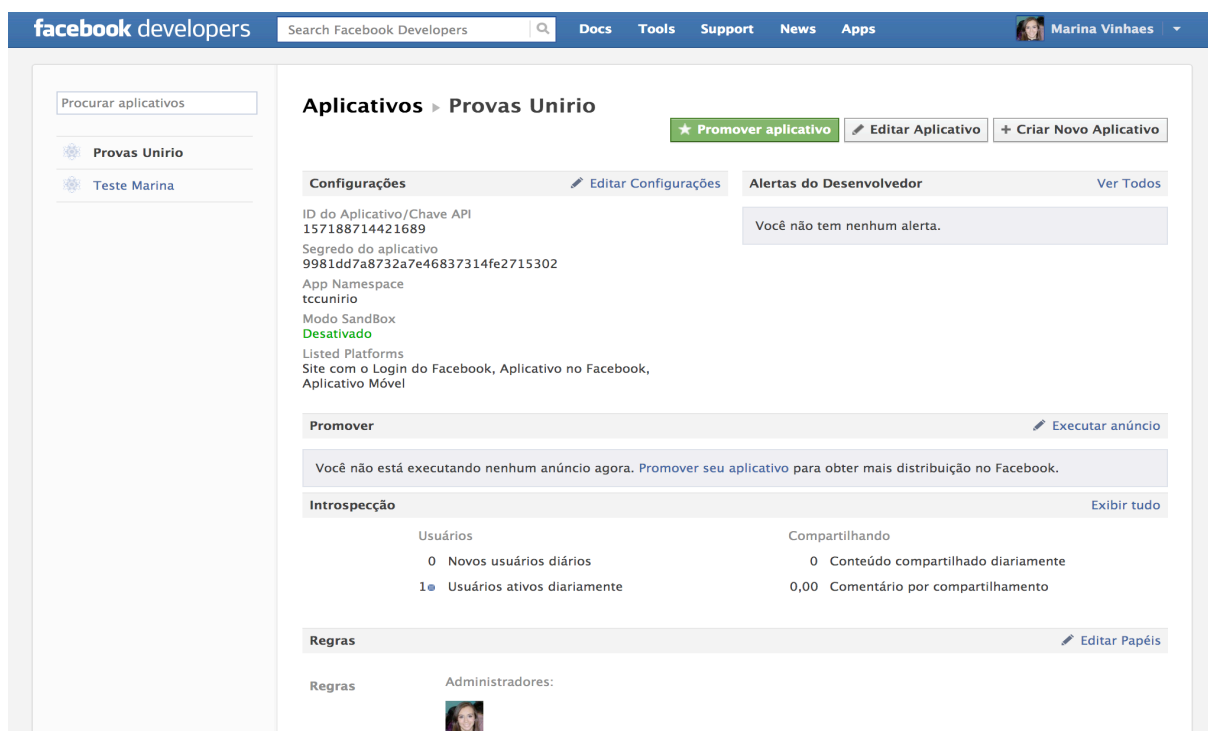
Após as alterações no arquivo `url.py` basta inicializar o servidor para visualizar a página de administração.

### 3.4 Integração com Facebook

Parte do desafio proposto neste projeto foi a integração com o Facebook, para tornar possível o acesso de alunos através da rede social. Para tal, foi necessário alterar o template da página de *login*, adicionando o JavaScript referente ao acesso do Facebook.

O benefício de utilizar o Facebook é que um aluno ou professor não tem que criar um login e senha especificamente para o sistema de provas online, podendo ser utilizados os dados criados previamente pelo usuário na rede social. É um pré-requisito do sistema é que o usuário tenha uma conta no Facebook, caso contrário, pode inserir seus dados na rede social através da aplicação.

Figura 4 – Tela de Criação do Facebook App



Para realizar a integração foi estudada a API, ou seja, a documentação do Facebook para atender da melhor forma o objetivo do desafio. A opção escolhida foi realizar a integração por JavaScript, pois atendia à meta proposta. Porém há diversas formas de realizar a integração, como por exemplo, por PHP e outras linguagens de programação como Ruby, C# e Node.js. Pode-se integrar o sistema com aplicativos iOS, Android e desktop.

Para realizar a integração primeiramente é necessário criar um aplicativo com perfil de desenvolvedor no Facebook, como visto na figura 4, para gerar um Facebook App ID que será requisitado posteriormente. O próximo passo é adicionar o JavaScript SDK, que é um pedaço de código em JavaScript criado por desenvolvedores do Facebook, na página HTML correspondente. Por fim basta adicionar o código de Login ao HTML. Ambos os códigos podem ser vistos no código 50. O código referente ao JavaScript SDK está entre tags de script e o referente ao Login dentro da div com a classe facebook.

A tela de login do Facebook é replicada no sistema de provas online, de forma que usuários não precisam ir à rede social para executar o acesso ao projeto.

*Código 50 – Tela com desenvolvimento em HTML do Facebook*

```
<script type="text/javascript">
  // Load the SDK Asynchronously
  (function(d){
    var js, id = 'facebook-jssdk', ref =
d.getElementsByTagName('script')[0];
    if (d.getElementById(id)) {return;}
    js = d.createElement('script'); js.id = id; js.async = true;
    js.src = "//connect.facebook.net/en_US/all.js";
    ref.parentNode.insertBefore(js, ref);
  })(document);
</script>

<div class="facebook">
  <h3>Sistema de Provas Online UNIRIO</h3>
  <div class="fb-login-button" data-show-faces="true" data-width="300"
data-max-rows="5" size="xlarge" read_stream="true">Login</div>
</div>
```

A maneira mais simples de utilizar o Facebook em aplicações é pelo *Social Plugins* que adicionam botões de *like*, campos de comentários e de *feed* em projetos com uma linha de código, os *iframes*, que montam códigos por meio de uma *tag*. Porém esses casos só são utilizados quando não há integração direta com o Facebook, por exemplo, em casos de divulgação de páginas web ou de discussões em sites. O Facebook, nesses casos, é somente uma ferramenta.



### ***3.5 Considerações Finais***

Nesse capítulo pôde ser visto um pouco mais sobre o projeto desenvolvido ao longo da monografia. Foram apresentadas algumas funcionalidades utilizadas no desenvolvimento do sistema e foi apresentado o desafio proposto – integração da autenticação com o Facebook.

Através dos diagramas apresentados, o fluxo do sistema ficou mais claro, sendo identificado por muitos usuários. As funcionalidades apresentadas nesse capítulo foram um detalhamento do modelo MTV, citado no capítulo 2, essenciais à criação de um programa em Python. Na próxima seção será mostrado o fluxo de navegação do sistema de provas e suas telas.

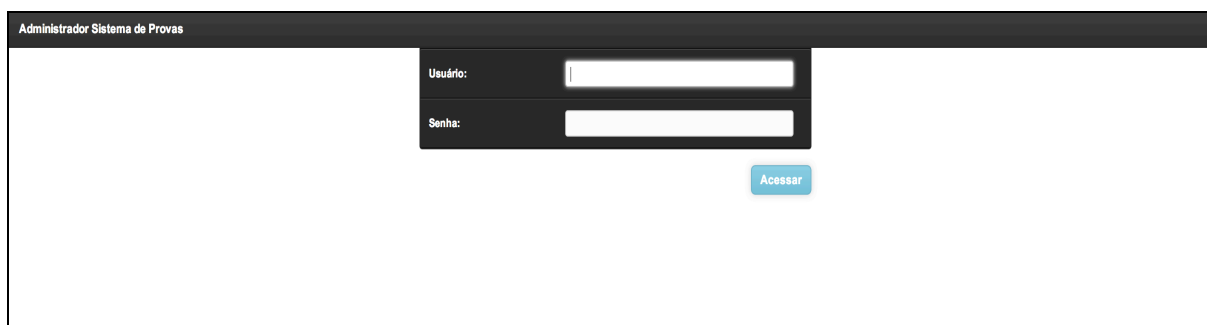
## 4. Navegando no Sistema de Provas Online

Nesse capítulo será realizada uma apresentação das telas do sistema de acordo com os principais usuários: Administradores (Seção 4.1), Professores (Seção 4.2) e Alunos (Seção 4.3). Conforme as aplicações serão apresentadas, também será explicado o fluxo de navegação e partes do sistema que foram avaliadas como um diferencial em Python em relação a outras linguagens de programação.

### 4.1 Apresentação do Sistema: Administradores

Na figura 5, pode ser vista a tela de *login* usada por Administradores e Professores. Essa página tem um pequeno diferencial, uma extensão que gera uma interface administrativa de páginas do Django, o *django-grappelli*. Seu propósito é personalizar páginas, incluir elementos, alterar suas disposições e CSS.

Figura 5 – Login da Tela de Administração



Para o primeiro acesso ao projeto foi criado um usuário e senha, através da execução do comando no código 42, no qual foi gerada uma conta de administrador para o sistema. Esse comando cria também a estrutura do banco de dados.

As telas em que pode ser visto gerenciamento de dados do banco de dados, como por exemplo telas de inserção de cursos, disciplinas e provas, são geradas automaticamente pela tela de administração do django, vista na sessão 3.3.6.

A tela principal do site, para onde o usuário é direcionado ao se logar, foi criada com o intuito de permitir que administradores insiram dados da instituição de ensino e do corpo docente, e professores tenham uma ferramenta para preencher as provas de turmas que ministram. Alunos não terão acesso a essa página, pois o propósito do projeto é tornar fácil e ágil o ingresso desses usuários ao site. Portanto, eles terão uma forma alternativa de validação, sem determinação prévia de usuário e senha, pelo Facebook.

*Figura 6 – Tela Principal de Administração do Site pela Visão de Administradores*

Administrador Sistema de Provas

Início

Administração do Site

Módulos do Sistema

Aplicações

Prova

Notas

+ Adicionar

≡ Modificar

Provas

+ Adicionar

≡ Modificar

Questões

+ Adicionar

≡ Modificar

Questões e Alunos

+ Adicionar

≡ Modificar

Sistema

Aluno Disciplinas

+ Adicionar

≡ Modificar

Centros

+ Adicionar

≡ Modificar

Cursos

+ Adicionar

≡ Modificar

Disciplinas

+ Adicionar

≡ Modificar

Turmas

+ Adicionar

≡ Modificar

Usuários

+ Adicionar

≡ Modificar

Administração

Grupos

+ Adicionar

≡ Modificar

Sites

+ Adicionar

≡ Modificar

Usuários

+ Adicionar

≡ Modificar

Provas

> Correção de Provas

> Relatório de Notas

> Sistema alunos

Ações Recentes

≡ professores

Grupo

≡ marcio

Usuário

≡ [professor] - Marcio

Usuário

≡ professores

Grupo

× [aluno] -

Usuário

Na página principal do sistema, dependendo das permissões de visão de cada usuário, certas aplicações não serão apresentadas. A figura 6 mostra a visualização de administradores com acesso a todas as funcionalidades do sistema.

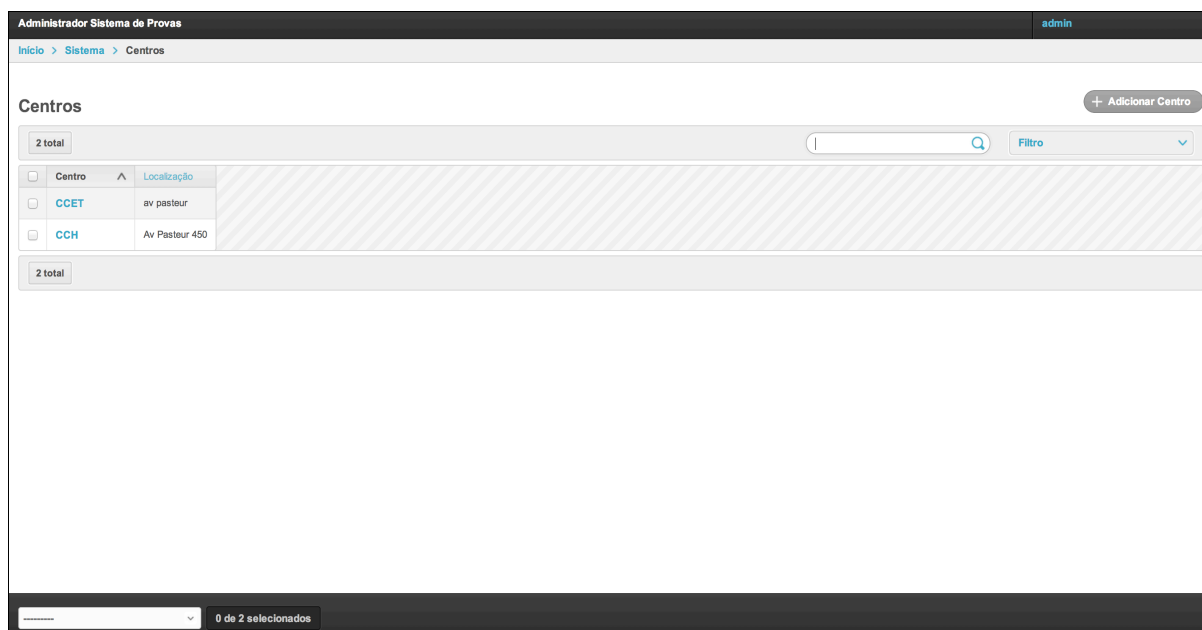
Essa tela tem um painel personalizado, o arquivo *dashboard*, que é gerado para que a tela principal possa ser customizada de acordo com as necessidades do usuário. Diferenciações apresentadas nesse arquivo são as ações recentes, que permite que alterações realizadas no sistema sejam automaticamente apresentadas na tela principal, assim como uma indicação de que ação foi realizada, como uma exclusão, adição ou modificação; a área de visualização de provas que permite acessar links de outras URLs; e a subdivisão de *apps* em blocos. No caso acima há dois blocos, um para Módulos do Sistema e outro para Administração. O código 51 apresenta o comando para gerar um arquivo *dashboard*.

*Código 51 – Comando para gerar o dashboard*

```
pip install django-admin-tools
```

Na tela principal são apresentados diversos modelos, porém as funções que um administrador deve exercer são restringidas aos modelos de centros, cursos, disciplinas, turmas e usuários. Na figura 7 é apresentada a tela de listagem do modelo de centros, onde o usuário poderá listar todos os centros previamente cadastrados no sistema.

*Figura 7 – Tela de Listagem de Centros*



Caso existam muitos centros cadastrados, o usuário pode filtrar elementos pelo nome ou pesquisar por palavras-chaves, ambas as funcionalidades são apresentadas no lado direito da página. A partir da página apresentada na figura 7 o usuário pode ser direcionado à tela de adição de centros apresentada na figura 8, ou à tela da figura 9 onde centros podem ser editados.

Em telas de listagem de elementos pode ser visto no rodapé uma referência em HTML conhecida como dropdown. Ao lado desse elemento é informado quantos elementos um app tem cadastrado. Uma das abas do dropdown dá ao usuário a possibilidade de apagar diversos elementos de uma vez, sem ter que acessar os dados do elemento, para isso basta selecionar os elementos desejados no quadrado apresentado ao lado do atributo principal e clicar na opção apagar do dropdown. Caso haja algum elemento pendente ao elemento a ser apagado, não será possível apagá-lo sem apagar primeiro a pendência.

*Figura 8 – Tela de Adição de Centros no Sistema*

A imagem mostra a interface de usuário para adicionar um novo centro. No topo, há uma barra de navegação com o título "Administrador Sistema de Provas" e o nome de usuário "admin". Abaixo, uma barra de breadcrumbs indica o caminho: "Início > Sistema > Centros > Adicionar Centro". O formulário principal, intitulado "Adicionar Centro", contém dois campos de entrada: "Centro" e "Localização". No rodapé da interface, há três botões de ação: "Salvar e continuar editando", "Salvar e adicionar outro(s)" e "Salvar".

Na figura 8 pode ser vista a tela para adição de um centro. O nome e a localização de centros são atributos criados na classe correspondente. Os campos a serem preenchidos podem ser obrigatórios ou não, no caso da figura 8, todos os campos são de preenchimento obrigatório. Caso a validação dos campos não esteja correta, não será possível salvar o objeto no banco de dados.

Em telas de adição ou de modificação há três opções de redirecionamento que o usuário pode optar. Uma das opções é salvar um dado que esteja sendo alterado e permanecer na tela atual, outra opção é salvar um dado e ser direcionado para uma tela de adição de dados, no caso de centros é representado pela figura 8. A terceira opção é salvar o dado e ser redirecionado à tela de listagem de elementos, representada pela figura 7. Em todos os casos será apresentada uma mensagem indicando a ação que foi realizada pelo usuário.

Na figura 9 pode ser vista a tela para edição dos campos nome e localização de um centro previamente cadastrado.

*Figura 9 – Tela de Preenchimento de Dados de Centros*

Administrador Sistema de Provas admin

Início > Sistema > Centros > CCET

Modificar Centro Histórico

Centro	CCET
Localização	av pasteur

Apagar Salvar e continuar editando Salvar e adicionar outro(s) Salvar

Além do modelo de centros, os modelos de cursos, disciplinas e turmas tem telas de listagem de dados, adição e edição de elementos, porém cada modelo com seus respectivos atributos. As funcionalidades são as mesmas às apresentadas pelas figuras 7, 8 e 9, a única diferença são os atributos de cada modelo.

Na figura 10 pode ser vista a tela de atributos do modelo usuário. Nesse modelo o administrador não terá a função de cadastrar usuários, pois cada usuário do sistema irá se logar através de sua conta pessoal do Facebook.

Figura 10 – Tela de Atributos de Usuários

Administrador Sistema de Provas admin

Início > Sistema > Usuários > [professor] - Marcio Barros

Modificar Usuário Histórico

Usuário [seleção] +

Tipo de Usuário Professor

Nome Marcio Barros

Email

Facebook id 100001649560759

Apagar Salvar e continuar editando Salvar e adicionar outro(a) Salvar

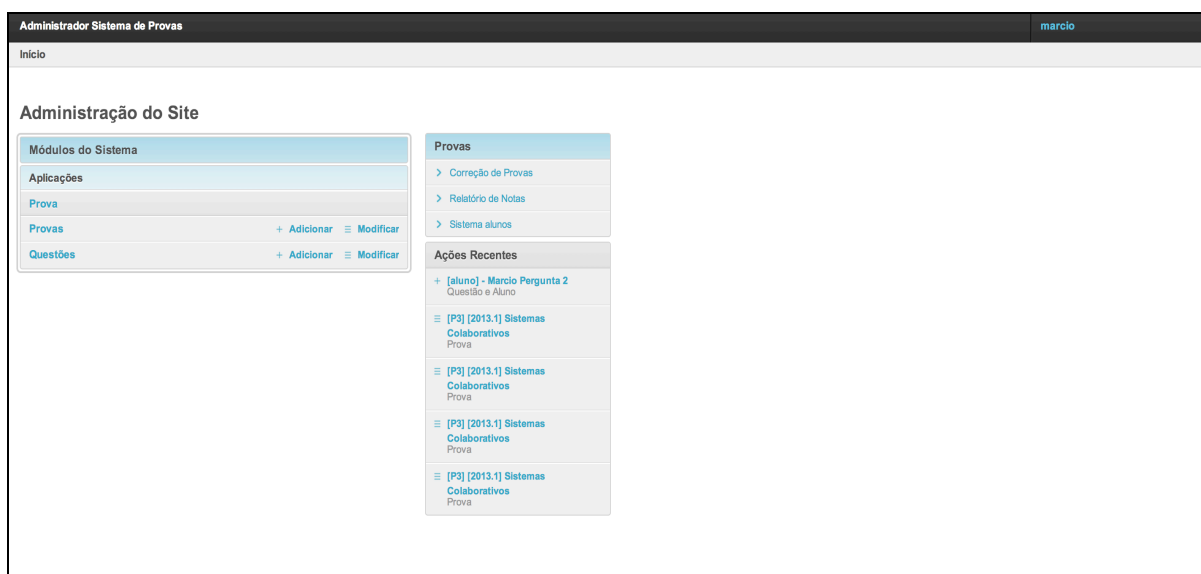
Para esse modelo, o sistema funciona da seguinte forma: quando um usuário se loga no sistema, por padrão ele será considerado um aluno. O administrador tem a função de identificar todos os professores que estão cadastrados como aluno no sistema e alterar o seu tipo de usuário para professor. Após alterar o tipo de usuário, o administrador deverá criar um *login* e senha através do atributo usuário, como apresentado na figura 10. Um usuário de sistema cadastrado que tenha acesso à tela de administração do sistema com perfil de professor poderá utilizar as ferramenta de criação de provas, correção e visualização de notas.

## 4.2 Apresentação do Sistema: Professores

Será apresentada a visão de professores ao sistema, onde somente algumas aplicações poderão ser visualizadas. A principal função dos professores será criar provas e cadastrar questões. Portanto, provas e questões serão os únicos modelos disponíveis a esse usuário.

Professores terão a função de corrigir provas e poderão visualizar notas previamente lançadas no sistema, assim como podem acessar o sistema na visão de alunos. Todas essas funções estão presentes na tela, do lado direito de módulos do sistema, como pode ser visto na figura 11.

Figura 11 - Tela Principal de Administração do Site pela Visão de Professores



As aplicações presentes na página de administração são referentes às classes criadas nos *models* de cada *app*. Para visualizar os grupos de aplicações com conteúdos editáveis é necessário criar um arquivo no diretório de *apps* do sistema, chamada de *admin.py*. Esse arquivo tem o objetivo de sinalizar para o sistema quais objetos de uma determinada *app* têm uma interface de administração.

O arquivo *admin.py* importa classes presentes no arquivo *models.py*, pois só é possível realizar a customização de classes existentes no projeto. Para que uma classe seja visualizada na tela principal de forma editável, deve-se adicionar a linha `admin.site.register(classe)` ao arquivo, sendo que a variável *classe* representa a classe correspondente no arquivo *models*.

Para uma customização maior dos campos de uma classe, basta adicionar a linha `admin.site.register(classe, classeAdmin)`, como visto na penúltima linha do código 52, referente à `admin.site.register(Prova, ProvaAdmin)`. A classe *ProvaAdmin* será uma classe de customização de atributos correspondente à classe *Prova*, presente nos *models*, com alterações e mudanças que poderão ser vistas nas telas subsequentes à página principal.

Dentro da classe de customização, diversas alterações poderão ser realizadas na disposição de dados, na visualização e no layout, como por exemplo, em *ProvaAdmin* a variável *fields* indica quais campos devem ser visíveis a usuários; *search\_field* indica quais



campos podem ser procurados na página de listagem de campos das *apps*; e *list\_editable* permite editar um campo sem ter que entrar na página de edição deste elemento, podendo alterá-lo diretamente na página de listagem, como pode ser visto na figura 12, pelo campo Data de Publicação.

*Código 52 – Tela Admin.py*

```
class ProvaInline(admin.StackedInline):
    model = Prova
    extra = 1

class ProvaAdmin(admin.ModelAdmin):
    inlines = (QuestaoInline, )
    list_display = ['turma', 'prova', 'semestre', 'publicacao']
    list_filter = ['turma', 'expiracao', 'prova', 'publicacao']
    list_editable = ['publicacao', ]
    fields = ['turma', 'prova', 'semestre', 'publicacao']
    search_fields = ['prova', 'turma', 'semestre']
    ordering = ['expiracao']

class NotaAdmin(admin.ModelAdmin):
    fields = ['prova', 'nota_final', 'aluno']
    search_fields = ['prova', 'nota_final', 'aluno']
    list_filter = ['prova', 'nota_final', 'aluno']

admin.site.register(Prova, ProvaAdmin)
admin.site.register(Nota)
```

Na figura 12 é apresentada a página que exibe a quantidade de provas criadas no banco de dados e suas chaves principais. Também é criado um *id* para cada elemento automaticamente, que só pode ser visto na URL ao editar o elemento. A forma que a classe foi customizada em *admin.py* permite visualizar alguns campos além do campo chave, na página de listagem dos elementos.

Figura 12 – Tela de Lista de Provas no Banco de Dados

Administrador Sistema de Provas admin

Início > Prova > Provas

Provas + Adicionar Prova

2 total

Turma	Prova	Semestre	Data de Publicação
Banco de Dados	Prova1	2013,1	02/04/2013 00:00:00
Banco de Dados	Prova2	2013,1	02/04/2013 16:00:00

2 total

0 de 2 selecionados Save

Um dado interessante em customização de campos da tela é a variável *inline*. Esse objeto permite que uma classe herde outra classe com atributos e customização própria. No sistema apresentado uma variável *inline* foi criada, como visto na figura 11. Quer dizer que a classe Provas terá um *inline*, da classe Questao, que terão a classe e *fields* herdados.

Figura 13 – Tela de Adição, Edição e Exclusão de Dados

Administrador Sistema de Provas admin

Início > Prova > Provas > [P1] [2013,1] Banco de Dados

Modificar Prova Histórico

Turma Banco de Dados +

Prova Prova1

Semestre 2013,1

Data de Publicação 02/04/2013 00:00:00

Data de Expiração 04/04/2013 21:00:00

Questões

Questão	Pergunta	Gabarito
Questao Teste	Questao Teste	Gabarito Teste

Valor Máximo da Questão 4.0

Apegar Salvar e continuar editando Salvar e adicionar outro(s) Salvar

A figura 13 é visualizada assim que um dos itens da página de listagem da figura 12 é clicado. Dessa forma, o usuário é direcionado para a página de edição, adição e exclusão do elemento que clicou. Na figura 12 podem ser vistos todos os atributos da classe *Prova*, sendo que alguns elementos tem preenchimento obrigatório para que o elemento seja salvo no banco de dados. Também pode ser visto o caso do *inline*, onde pode ser criado um elemento da classe *Questão* e ao mesmo tempo um elemento da classe *Prova*.

Nesse projeto foi utilizado o AJAX, um método que torna páginas web iterativas. O AJAX torna o ato de se inscrever em disciplinas e ter uma resposta, quase automático, pois atualiza partes de uma página web sem ter que recarregar a página inteira.

A tela de correção de provas foi feita inteiramente em AJAX com o intuito de tornar dinâmico o processo de correção de provas e auxiliar os professores nesse processo. Por exemplo, ao escolher uma opção de turma, a opção de prova terá seus campos preenchidos automaticamente de acordo com o campo preenchido anteriormente. Esse procedimento será realizado em cada opção disponível e, ao final, é possível salvar a nota do aluno em uma questão de acordo com o gabarito.

*Figura 14 – Tela de Correção de Provas*

A interface de correção de provas é dividida em duas colunas. A coluna da esquerda, intitulada "Correção de Provas", contém três menus suspensos para "Turma:", "Prova:" e "Questões:". Abaixo deles há uma grande área de texto para o gabarito e um campo "Nota máxima da Questão:". A coluna da direita contém um menu suspenso para "Alunos:", uma grande área de texto para a resposta do aluno e um campo "Nota:" com um botão "Salvar" ao lado.

Na figura 14, o gabarito e a resposta do aluno serão mostradas ao mesmo tempo para serem comparadas e tornar mais fácil a correção de questões.

Além de realizar a correção de provas, os professores poderão analisar o relatório de notas dos alunos que realizaram as provas no sistema e tiveram suas notas lançadas, como visto na figura 15.

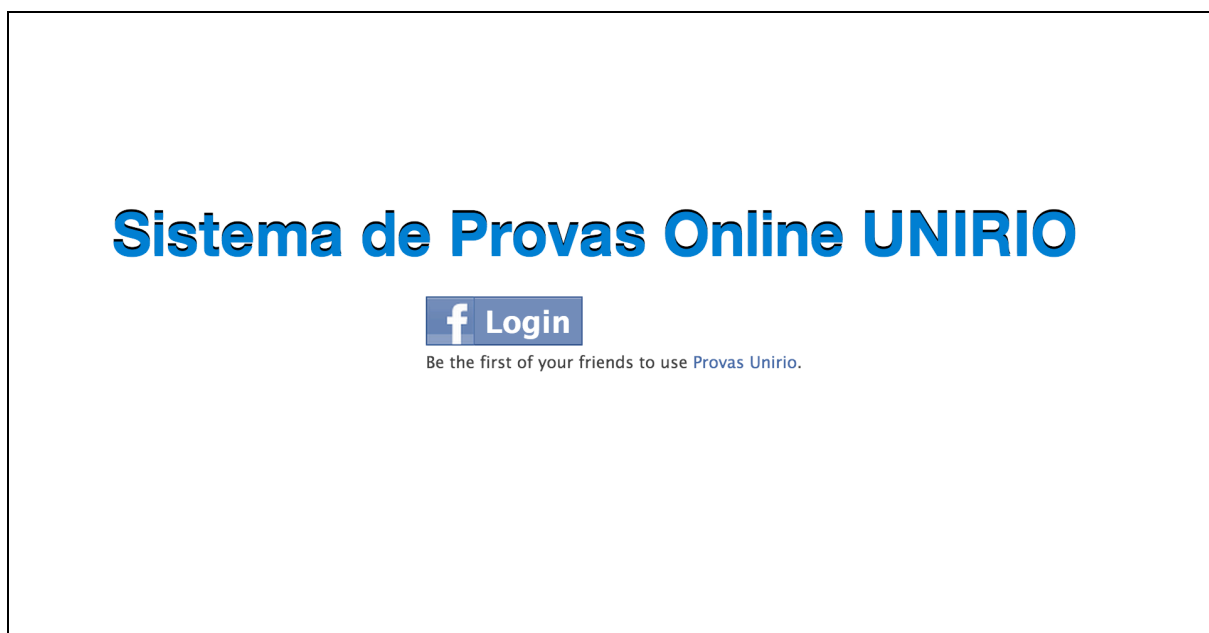
*Figura 15 – Relatório de Notas do Sistema*

Administrador Sistema de Provas			marcio
Início >			
Turma: Sistemas Colaborativos	Turma: Sistemas Colaborativos	Turma: Sistemas Colaborativos	Turma: Sistemas Colaborativos
Aluno: Marcio	Aluno: Beatriz Vinhaes	Aluno: Marina Vinhaes	Aluno: Marina Vinhaes
Prova: [P3] [2013.1] Sistemas Colaborativos	Prova: [P2] [2013.1] Sistemas Colaborativos	Prova: [P2] [2013.1] Sistemas Colaborativos	Prova: [P3] [2013.1] Sistemas Colaborativos
Nota: 3,0	Nota: 6,9	Nota: 8,0	Nota: 10,0

#### ***4.3 Apresentação do Sistema: Aluno***

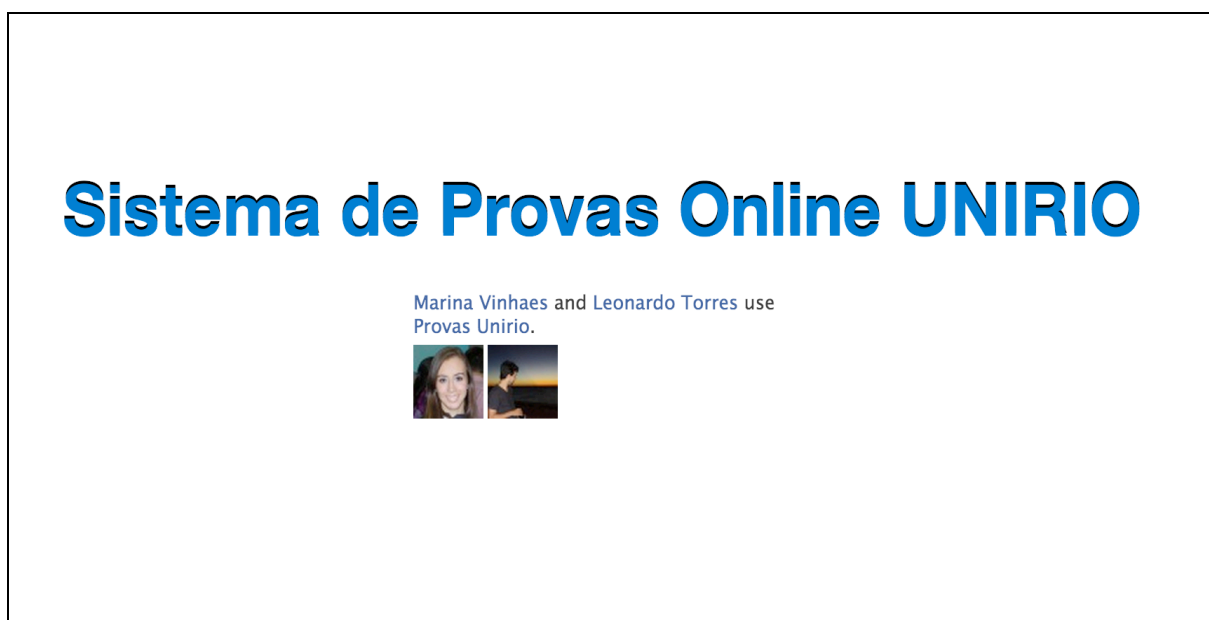
Como foi dito na especificação do projeto, o sistema é dividido em duas partes, e direcionado para dois atores principais, além do administrador: há uma parte do sistema direcionada para acesso dos professores e uma parte de acesso para os alunos. Na figura 16 pode ser visto a tela de *login* em que os alunos deverão acessar o sistema com suas senhas pessoais do Facebook e em seguida serão direcionados para a página principal do projeto na visão de alunos, representado pela figura 18.

*Figura 16 – Tela de Login de Alunos*



Por um breve momento o aluno poderá visualizar outros usuários que utilizam a ferramenta de realização de provas no Facebook, como visto na figura 17.

*Figura 17 – Visualização de Alunos que Utilizam a Ferramenta no Facebook*

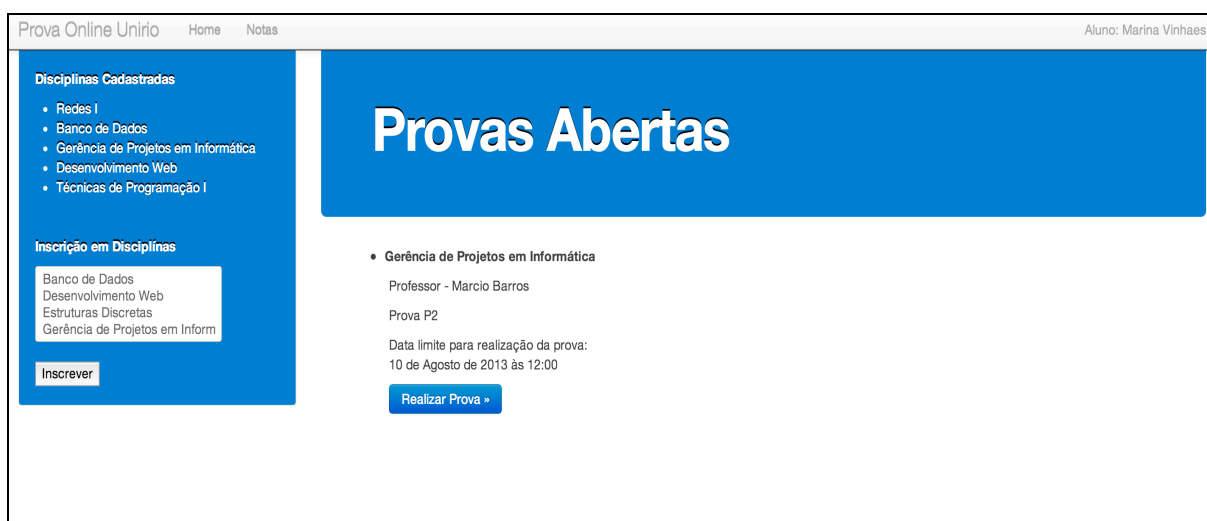


Ao se conectar no sistema, o usuário será automaticamente redirecionado para a tela principal do programa, na visão do aluno. Nessa tela, a primeira ação que o aluno deve

realizar é se cadastrar nas disciplinas que estiver cursando no período, para ter acesso a informações futuras.

A ação de se cadastrar em disciplinas foi desenvolvida em AJAX, de forma que no momento que um aluno se cadastra nas disciplinas, automaticamente serão apresentadas as provas abertas para realização desta disciplina e a disciplina solicitada será marcada como cadastrada.

*Figura 18 - Tela de Inscrição de Disciplina*



Ao se inscrever em disciplinas, será mostrado no lado direito da tela todas as provas que estão em tempo hábil de serem preenchidas, como visto na figura 18. Todas as provas que a data atual for menor que o prazo de realização da prova terão um botão que redirecionará o aluno para a página de preenchimento das questões da prova, que está localizado abaixo da descrição detalhada da turma, matéria e professor na figura 18. Caso tenha passado da data limite de realização da prova, como determinado pelo professor, a prova não estará mais visível ao aluno.

Se a prova estiver aberta para realização e o aluno ainda não tiver realizado a prova, o aluno será direcionado para página de questões da prova, onde ele poderá responder as perguntas da prova. Ao finalizar a prova, as respostas não poderão mais ser alteradas e serão enviadas ao professor para correção. A figura 19 apresenta uma tela de prova a ser respondida.

*Figura 19 – Tela de Realização da Prova*

The screenshot shows a web application interface for an online exam. At the top, there is a navigation bar with the text 'Prova Online Unirio' and links for 'Home' and 'Notas'. On the right side of the bar, it says 'Aluno: Marina Vinhaes'. The main content area contains two questions, each with a text input field. The first question is labeled '1 - Pergunta 1 (Valor: 2,0)' and the second is '2 - Pergunta 2 (Valor: 2,0)'. At the bottom left of the main area, there is a button labeled 'Finalizar'.

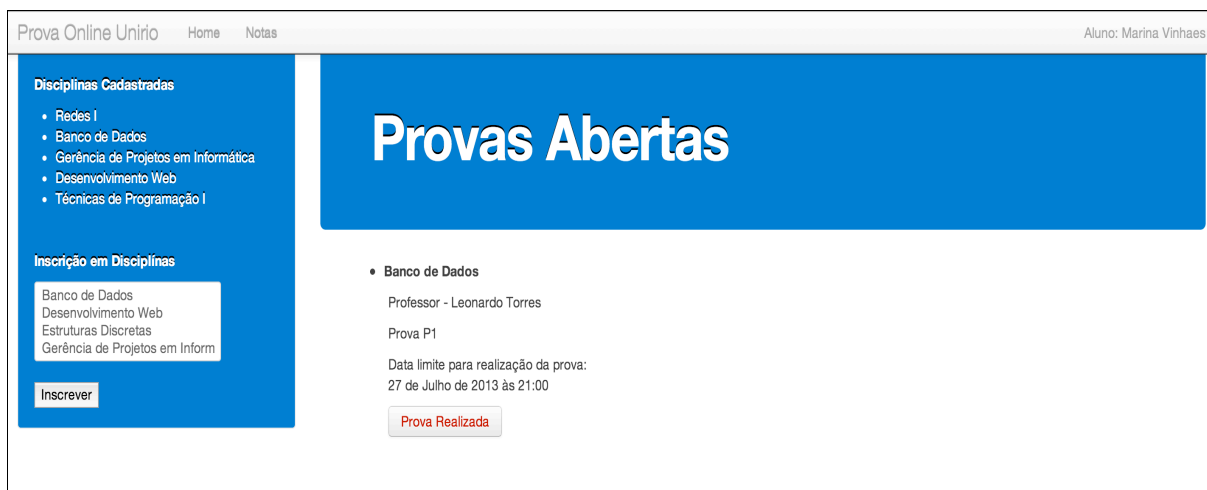
Ao finalizar a prova e enviá-la ao sistema uma confirmação de envio será apresentada na tela através de uma mensagem de sucesso, como na figura 20.

*Figura 20 – Mensagem de Prova Realizada com Sucesso*

This screenshot is similar to the previous one, showing the 'Prova Online Unirio' interface. However, a modal dialog box is overlaid in the center of the screen. The dialog box has a Chrome logo on the left and text that reads: 'The page at provaunirio.herokuapp.com says: Prova realizada com sucesso!'. At the bottom of the dialog box are two buttons: 'Cancel' and 'OK'. The background interface, including the questions and the 'Finalizar' button, is visible but slightly dimmed.

Caso a prova já tenha sido respondida, será sinalizado na tela principal do sistema, através de um botão vermelho. Será indicado que o aluno enviou suas respostas anteriormente ao sistema e não poderá realizar a prova novamente, como visto na figura 21.

*Figura 21 – Tela com Provas Realizadas*



Através da barra de menu, no topo da página, será possível acessar a área de histórico de notas. Essa área tem todas as provas que foram realizadas e tiveram sua notas lançadas, para que o aluno possa se manter atualizado de provas realizadas no período.

*Figura 22 – Tela de Visualização de Provas com Nota no Sistema*



A opção visualizar na figura 22, abaixo dos dados da prova, redireciona o aluno para a página do gabarito, representado pela figura 23. Esses dados foram inseridos pelo professor no momento em que a prova foi criada e serão apresentadas ao aluno assim que este tiver suas notas lançadas no sistema.



*Figura 23 – Tela de Gabarito de Provas Realizadas*

Prova Online Unirio Home Notas				Aluno: Marina Vinhaes
Questão	Pergunta	Resposta	Gabarito	Nota
1	Pergunta Teste 1	Resposta 1	Gabarito 1	4,0
2	Pergunta Teste 2	Resposta 2	Gabarito 2	4,0

#### ***4.4 Considerações Finais***

Para entender melhor o sistema apresentado nesse capítulo, foi recriado passo a passo da navegação com imagens para melhor entendimento de quais funções serão designadas a cada usuário. Além de criar a aplicação e apresentar o sistema, também foram apresentados detalhes de como o sistema de Provas Online foi implementado.

Por fim, pôde ser visto que o sistema apresentado é simples e de fácil entendimento.

## 5. Conclusão

Nesse capítulo, serão tratadas as contribuições realizadas pelo projeto de Provas Online, melhorias que poderão ser feitas futuramente e questões que não foram possíveis serem tratadas nesse projeto.

### 5.1 Contribuições

As contribuições desse projeto foram feitas através da criação de um projeto desenvolvido na linguagem Python na qual apresenta um sistema que realiza a criação, o preenchimento e a correção de provas online, podendo auxiliar professores e alunos a realizar testes de conhecimento de disciplinas ofertadas em um curso.

No quesito de aprendizagem e conhecimento, a linguagem de programação Python é usada há muitos anos, porém foi recentemente, em 2005, que o *framework* Django foi associado à linguagem, tornando-a propícia para desenvolvimento de aplicações web. Apesar de ter ganhado notoriedade no mercado de trabalho pela facilidade e agilidade que um sistema é criado, também pode trazer benefícios ao ambiente acadêmico, de aprendizagem, devido ao seu depurador de fácil visualização de variáveis e dos elementos do sistema.

O *framework* Django cresce cada vez mais e frequentemente é atualizado com novas versões, com mais funcionalidades e diversos tipos de desenvolvimento, sem falar das diversas extensões que são criadas por desenvolvedores com o intuito de melhorar a programação, algumas delas utilizadas no decorrer do desenvolvimento deste projeto.

### 5.2 Trabalhos Futuros

Um possível trabalho futuro para esse tema seria estudar a API do Facebook com o intuito de que a aplicação possa ser realizada inteiramente na rede social, sem demandar links

externos e URL específicas para o projeto. Neste sentido, não seria necessário criar um novo domínio para hospedar o site, ele estaria naturalmente no domínio do Facebook.

Para que esse projeto fosse desenvolvido dentro do domínio do Facebook seria necessário um estudo aprofundado sobre como inserir uma aplicação na rede social utilizando o canvas do Facebook.

### ***5.3 Limitações do Estudo***

O projeto teve o objetivo de desenvolver um sistema usando uma linguagem de programação e um *framework* utilizado no mercado que não são tipicamente apresentados durante o curso de Sistemas de Informação.

Obviamente que o trabalho poderia ser melhorado em questão de desempenho, mas neste estágio do seu desenvolvimento ele deve ser visto como um protótipo. Portanto, não foram consideradas questões como número de acessos ao site, ou como deixar o site mais rápido caso tenha um grande número de informações e dados. Essas alterações requereriam um estudo profundo e uma análise sobre o público alvo, visando identificar as melhores formas de atendê-los com o mínimo consumo de recursos computacionais.

## REFERÊNCIAS

- ASP.NET, Disponível em: <http://www.asp.net/mvc/tutorials/older-versions/overview/asp-net-mvc-overview> Acessado em: Agosto 2013
- BINDINGS, Disponível em: <http://rudamoura.com/python-bindings.html> Acessado em: Agosto 2013.
- CSS, Disponível em: <http://www.w3.org/TR/CSS21/syndata.html#q10> Acessado em: Agosto 2013.
- C, Kernighan, Brian W.; Ritchie, Dennis M, “*The C Programming Language*”, *Upper Saddle River, New Jersey: Prentice hall, 1978.*
- C#, Torgersen, Mads, "New features in C# 4.0", October 27, 2008.
- C++, Stroustrup, Bjarne, “*The C++ Programming Language* (Third ed.)”, 1997.
- DJANGO, Django Book. <http://www.djangobook.com/>. Acessado em Agosto 2013
- DJANGO, Kaplan-Moss, Jacob; Holovaty, Adrian, “The Definitive Guide to Django: Web Development Done Right (Expert's Voice in Web Development)”, Dezembro 2007.
- JAVA, Gosling, James; and McGilton, Henry, "The Java Language Environment", May 1996.
- JQUERY, Disponível em: <http://docs.jquery.com> Acessado em: Agosto 2013.
- JAVASCRIPT – Disponível em: [https://developer.mozilla.org/en-US/docs/JavaScript/New\\_in\\_JavaScript/1.8.5](https://developer.mozilla.org/en-US/docs/JavaScript/New_in_JavaScript/1.8.5) Acessado em: Agosto 2013.

PHP, Disponível em: [http://www.php.net/manual/pt\\_BR/preface.php](http://www.php.net/manual/pt_BR/preface.php) Acessado em: Agosto 2013.

PYTHON, Disponível em: <http://docs.python.org/tutorial/>. Acessado em: Agosto 2013.

PYTHON, Magnus Lie Hetland, “Begining Pyhton - From Novice to Professional”, 2008.

WIKIPEDIA, Django. [http://pt.wikipedia.org/wiki/Django\\_\(framework\\_web\)](http://pt.wikipedia.org/wiki/Django_(framework_web)). Acessado em: Agosto 2013

WIKIPEDIA, Python. <http://pt.wikipedia.org/wiki/Python>. Acessado em: Agosto 2013.

WIKIPEDIA, Nginx. <http://pt.wikipedia.org/wiki/Nginx>. Acessado em: Agosto 2013.

XHTML Disponível em: <http://www.w3.org/TR/xhtml-modularization/> Acessado em: Agosto 2013.