

Centro Universitário Barão de Mauá
Curso de Bacharelado em Ciência da Computação

**FRAMEWORK DE DESENVOLVIMENTO WEB DJANGO E SUAS
PRINCIPAIS CARACTERÍSTICAS**

Willem Allan Rigo Ferreira

Ribeirão Preto

2011

Willem Allan Rigo Ferreira

**FRAMEWORK DE DESENVOLVIMENTO WEB DJANGO E SUAS
PRINCIPAIS CARACTERÍSTICAS**

Trabalho de Conclusão de Curso apresentado ao término do Curso de Ciência da Computação do Centro Universitário Barão de Mauá, como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Lucas Baggio Figueira

Ribeirão Preto

2011

Dedicatória

Dedico este projeto ao meu pai
Carlos, minha mãe Regina, meus
irmãos Wendell e Karen, colegas e
amigos que me ajudaram e acreditaram
no desenvolvimento deste trabalho.

Agradecimentos

*Agradeço a todos que me
ajudaram a desenvolver este trabalho
e aos meus familiares que me
“aturaram” nestes dias estressantes.*

*Agradecimento especial ao
Prof. Dr. Lucas Baggio, que
embarcou neste trabalho auxiliando-
me permanentemente.*

RESUMO

Este trabalho tem o objetivo de mostrar as facilidades de se utilizar o Framework Django para desenvolvimento de web sites simples ou complexos. Para testar o Framework foi desenvolvido um portal educacional como modelo para maximizar o aproveitamento das suas vantagens. O sistema permite que o Professor possa publicar materiais, divulgação de notas e dados relativos à frequência. O Professor pode reservar equipamentos para utilização em sala, evitando o incômodo de deslocamento até a organização de ensino, e possibilita que os alunos tenham acesso às notas e materiais disponibilizados pelo professor. O aluno necessita apenas de um computador, tablet ou celular com acesso a internet com os quais pode retirar proveito do conhecimento disponível no portal. O sistema também contém uma área para que o administrador do sistema possa inserir as informações mais importantes, que só ele poderá criar ou alterar, como por exemplo, adicionar alunos e professores no sistema, também cadastrará as disciplinas, turmas, equipamentos disponíveis etc. Utilizou-se a linguagem Python junto com o Framework Django e o banco SQLite. Observou-se que o Framework facilita bastante o desenvolvimento, gerando o administrativo completo do sistema, precisando fazer apenas algumas customizações, além de gerar um banco de dados da aplicação a partir do modelo de dados.

Palavras-chave: Python, Django, SQLite, Framework.

LISTA DE FIGURAS

Figura 1 Desativando o USE_I18N	21
Figura 2 Iniciando um projeto Django.....	23
Figura 3 Arquivo de configuração do Django parte 1	24
Figura 4 Arquivo de configuração do Django parte 2	26
Figura 5 Arquivo de configuração do Django parte 3	27
Figura 6 Criação das apps do portal.....	28
Figura 7 Visualizando os arquivos da app aluno.....	28
Figura 8 Apps instaladas no settings.....	29
Figura 9 Model aluno	30
Figura 10 Código SQL gerado a partir do modelo aluno	31
Figura 11 Exemplo adquirido no site Django Brasil	33
Figura 12 Modelo curso	34
Figura 13 Modelo professor	34
Figura 14 Modelo turma	35
Figura 15 Modelo disciplina	35
Figura 16 Modelo professorEquipamento	35
Figura 17 Modelo material	36
Figura 18 Modelo equipamento	37
Figura 19 Modelo disciplinaAluno	37
Figura 20 Diretório raiz do projeto	38
Figura 21 Configurando o settings para o uso de templates parte 1.....	38
Figura 22 Configurando o settings para o uso de templates parte 2.....	39
Figura 23 Template base.html.....	39
Figura 24 Template index.html.....	40
Figura 25 Configurando o settings para o uso de templates parte 3.....	40
Figura 26 Folha de estilos do portal.....	41
Figura 27 Views home.....	42
Figura 28 Arquivo urls.py	42
Figura 29 Visualização da tela principal do site no navegador.....	44
Figura 30 Formulário encarregado de agendar os equipamentos	45
Figura 31 Forms.py arquivo que define a estrutura do formulário	46
Figura 32 View responsável por tratar os dados recebidos do formulário	47
Figura 33 Template agendado.html.....	48
Figura 34 Arquivo urls.py completo do portal.....	48
Figura 35 Professor escolhe o equipamento para agendar reserva	49
Figura 36 Formulário de reserva do equipamento escolhido na tela anterior	49
Figura 37 Exibindo os erros de validação do formulário.....	50
Figura 38 Campos preenchidos	50
Figura 39 Formulário submetido com sucesso.....	51
Figura 40 Template adiciona.html parte 1	51
Figura 41 Template adiciona.html parte 2	52
Figura 42 Tela de cadastro do material.....	52

Figura 43 View da app material	53
Figura 44 Configurações para o administrativo do Django no arquivo settings.py	54
Figura 45 Configurando a URL /admin/ no arquivo urls.py	55
Figura 46 Arquivo admin.py da app Aluno	55
Figura 47 Arquivo admin.py da app Turma	55
Figura 48 Arquivo admin.py da app ProfessorEquipamento	56
Figura 49 Arquivo admin.py da app Professor	56
Figura 50 Arquivo admin.py da app Material	56
Figura 51 Arquivo admin.py da app Equipamento	56
Figura 52 Arquivo admin.py da app DisciplinaAluno	56
Figura 53 Arquivo admin.py da app Disciplina	56
Figura 54 Arquivo admin.py da app Curso	57
Figura 55 Tela do login do admin do Django	57
Figura 56 Tela principal do administrativo	58
Figura 57 Tela das disciplinas cadastradas	58
Figura 58 Tela para cadastrar uma nova disciplina	59
Figura 59 Tela para agendamento de equipamento	59

LISTA DE TABELAS

Tabela 1 – Expressões Regulares.....	43
--------------------------------------	----

LISTA DE SIGLAS

MVC: *Model-view-controller*

RoR: *Ruby on Rails*

BSD: *Berkeley Software Distribution*

DRY: *Don't repeat yourself*

MTV: *Model-Template-View*

ORM: *Object-Relacional Mapping*

HTML: *HyperText Markup Language*

XML: *Extensible Markup Language*

JSON: *JavaScript Object Notation*

URL: *Uniform Resource Locator*

RSS: *Really Simple Syndication*

SQL: *Structured Query Language*

GNU: *GNU's Not Unix*

Grails: *Groovy on Rails*

SUMÁRIO

INTRODUÇÃO.....	12
1 APRESENTAÇÃO	13
1.1 FRAMEWORK	13
1.2 PARADIGMA MVC.....	14
1.3 DJANGO	15
1.3.1 MAPEAMENTO OBJETO-RELACIONAL.....	17
1.3.2 INTERFACE ADMINISTRATIVA.....	17
1.3.3 FORMULÁRIOS.....	18
1.3.4 URL DISPATCHER	19
1.3.5 TEMPLATES.....	19
1.3.6 CACHE	20
1.3.7 INTERNACIONALIZAÇÃO	20
2 DESENVOLVIMENTO DO PORTAL EDUCACIONAL	22
2.1 CONFIGURANDO O AMBIENTE DE TRABALHO.....	22
2.2 INICIANDO O PROJETO	22
2.3 CRIANDO UMA APP	27
2.4 MODELS	29
2.4.1 CAMPOS	31
2.4.2 TIPOS DE CAMPOS	31
2.4.3 OPÇÕES DOS CAMPOS	31
2.4.4 RELACIONAMENTOS	32
2.5 TEMPLATES	38
2.6 VIEWS	41
2.7 FORMULÁRIOS	44
2.8 ADMIN	54
CONCLUSÃO	60
REFERÊNCIAS BIBLIOGRÁFICAS.....	61
APÊNDICE A – INSTALAÇÃO DO PYTHON E DO DJANGO.....	63

INTRODUÇÃO

O Django é um Framework similar ao Ruby on Rails, diferenciando-se dele por utilizar a linguagem Python no desenvolvimento de sites. Desse modo, pressentiu-se que o desenvolvimento do portal educacional seria uma oportunidade para aprender a teoria e, com a prática, desfrutar dos recursos oferecidos pelo Django, a despeito da sua complexidade.

Presentemente, tornou-se vital que empresas e até pessoas disponham de um site para divulgação de sua marca, produtos e serviços. O custo de divulgação através da mídia: rádio, televisão e jornal é mais elevado em relação a um site.

Observa-se, hoje, um grande crescimento no número de usuários da internet nos países emergentes, especialmente no Brasil, onde as operadoras de internet têm reduzido os preços cobrados em decorrência da sua popularização. Além da ampliação do número de internautas, a revolução tecnológica vem diversificando e aumentando a quantidade de aparelhos portáteis de funcionamento cada vez mais simples, a exemplo de: tablet, celulares, televisores, netbooks etc, facilitando grandemente a circulação e geração de informações na web, criando novas necessidades e estimulando o consumo.

O desenvolvimento web tornou-se uma área fundamental em programação, combinada com os processos que visam a melhorar a administração e a manutenção da gestão de informações, dispositivos e equipamentos que operam eficazmente os dados armazenados de maneira a otimizar a tomada de decisão.

Este trabalho será composto por dois capítulos. O primeiro discorrerá sobre os conceitos teóricos e a apresentação geral do Framework, enquanto o segundo capítulo demonstrará os passos dados na instalação e desenvolvimento da aplicação do portal a ser criado, explicando os trechos de códigos mais importantes.

A base teórica do trabalho partirá da pesquisa documental, através de consulta a livros, revistas, artigos e sites. Em termos práticos, construirá um portal educacional, que servirá de projeto piloto para a aprendizagem e teste do Framework.

1 APRESENTAÇÃO

O Django é um Framework que vem ampliando os seus domínios, de forma crescente, dentro do cenário de desenvolvimento web. Após algumas dúvidas entre outros Frameworks, como o Ruby on Rails (RoR), que se assemelha ao Django por utilizar o paradigma MVC, que visa a separar o código em três camadas diferentes, que consistem no Model-View-Controller.

1.1 FRAMEWORK

Um Framework é uma aplicação que agrupa códigos comuns entre diversos projetos de software provendo uma funcionalidade genética. Também possui um conjunto de componentes definidos para ajudar o desenvolvedor a criar um projeto com o menor esforço possível. Framework é uma aplicação quase completa, mas que possui partes que precisam ser escritas pelo desenvolvedor.¹

Os diversos Frameworks possuem uma biblioteca de ORM (Object-Relational Mapping), que é responsável pela conexão com o banco de dados, e também encarregada de converter as tabelas do banco de dados em classes. Com isso, o programador não precisa se preocupar com a linguagem SQL (Structured Query Language). A biblioteca responsável pelo ORM pode gerar todas as classes na aplicação a partir de um modelo relacional ou vice-versa, facilitando o trabalho do programador. Outros componentes interessantes que os Frameworks utilizam são: criação de templates, integração de diversos idiomas, sistemas administrativos entre outros.

Muitos programadores confundem Framework com biblioteca, pois, ambos possuem diversas classes para auxiliar o desenvolvimento. Na biblioteca as classes não são interligadas, enquanto, no Framework as dependências e colaborações estão embutidas. É um Framework quem dita o fluxo de controle da aplicação, então, conclui-se que o programador não precisa se preocupar com as comunicações entre os objetos ao contrário das bibliotecas.²

¹ OFICINA DA NET. Framework, o que é e para que serve? 2011. Disponível em: <http://www.oficinadanet.com.br/artigo/1294/framework_o_que_e_e_para_que_serve>. Acesso em: 22 out. 2011.

² WIKIPEDIA. Framework. 2011. Disponível em: <<http://pt.wikipedia.org/wiki/Framework>>. Acesso em: 22 out. 2011.

As principais características que um Framework possui são: fácil de usar, reutilizável, bem documentado, deve ser extensível, contendo funcionalidades abstratas onde o desenvolvedor as implementa de acordo com a sua necessidade, além disso, deve ser seguro, eficiente e completo para atender todos os problemas propostos.

Alguns Frameworks que estão ganhando espaço no desenvolvimento web são: RoR (Ruby On Rails), Grails (Groovy on Rails), Django e CakePHP, todos eles utilizam o paradigma MVC (Model-View-Controller).

1.2 PARADIGMA MVC

MVC é um padrão de arquitetura de software que tende a separar a lógica de negócio da lógica de apresentação dos dados (design), permitindo um desenvolvimento ágil, isolando os testes e manutenção. Em algumas referências, é possível encontrar a sigla MTV (Model-Template-View), que representa basicamente a mesma coisa.³

“Embora não seja obrigatório o desenvolvimento usando esse paradigma, é extremamente recomendado, pois, dessa forma, você estará modularizando seu código e desacoplando a camada de negócio da aplicação das camadas de interface com o usuário. Além disso, o Django automaticamente organiza o esqueleto de seu projeto nesse formato e se baseia nesse paradigma para suas convenções (conforme foi explicado anteriormente ao falarmos sobre DRY).”⁴

O MVC é uma arquitetura ou padrão que permite dividir as funcionalidades de seu sistema em camadas. Essa divisão é feita para facilitar a solução de problemas. Ele possui três principais camadas, cada uma com suas características e funções bem definidas para facilitar a vida do desenvolvedor.

O Modelo (Model) é a camada responsável por manipular as informações dos dados, onde esses dados e informações são provenientes de um banco de dados ou de arquivo XML.

³ SANTANA, Osvaldo; GALESI, Thiago. PYTHON e DJANGO – Desenvolvimento ágil de aplicações web. Novatec Editora Ltda. São Paulo, 2010, p. 156.

⁴ SANTANA, Osvaldo; GALESI, Thiago. PYTHON e DJANGO – Desenvolvimento ágil de aplicações web. Novatec Editora Ltda. São Paulo, 2010, p. 156.

A Visão (View) é a camada responsável por todo conteúdo da aplicação visualizado pelo usuário.

A Controladora (Controller) é a camada responsável pelo controle do fluxo de informação que passa pelo sistema. É a controladora que executa a regra de negócio do Modelo (Model) e repassa a informação para a visualização (View).

Devido o aumento da complexidade das aplicações, essa arquitetura tem como foco separar um problema complexo em vários problemas menores. Quando os problemas são quebrados em vários problemas menores possibilita resolvê-los paralelamente sem interferir nas demais camadas, facilitando o desenvolvimento do layout, regras de negócio e criação de novos recursos. O paradigma MVC facilita muito o trabalho em equipe devido a sua divisão de tarefas. As principais características do paradigma MVC são: facilita reutilização de código; facilidade na manutenção; facilidade na inserção de novos recursos; maior integração no trabalho em equipe; e, facilidade em manter o código limpo e legível.⁵

1.3 DJANGO

O Django é um Framework MVC de desenvolvimento web, escrito na linguagem Python, que foi desenvolvido pelo grupo editorial "The World Company" no intuito de disponibilizar uma versão web dos seus jornais. Posteriormente, em 2005, foi liberado sob a licença Berkeley Software Distribution (BSD), tornando-se assim um software de código aberto. O Framework foi nomeado Django em homenagem ao famoso músico de jazz Django Reinhardt.

O Django atualmente está disponível na versão 1.3. O Framework foi desenvolvido no objetivo de resolver problemas complexos em prazos curtos, com isso prevenindo os desenvolvedores a não ultrapassarem os prazos estipulados.

⁵ OFICINA DA NET. O que é Model-view-controller (MVC). 2011. Disponível em: <http://www.oficinadanet.com.br/artigo/desenvolvimento/o_que_e_model-view-controller_mvc/>. Acesso em: 04 set. 2011.

Assim como diversos Frameworks ágeis de desenvolvimento web, o Django utiliza o conceito DRY - Don't repeat yourself (Não se repita), que trabalha com a ideia de reutilização de código, evitando a sua repetição.⁶

Django é considerado um superframework, pois, é composto de vários menores Frameworks (componentes) como: Mapeamento do Objeto Relacional, Admin, Formulários, URL Dispatcher, Templates, Cache, Internacionalização, serialização de dados, sistema de testes automatizado, serviço de mensagens (e-mail e mensagens entre usuários do sistema), geração de feeds (RSS/Atom), paginação de resultados etc.

Com Framework Django é possível gerar o administrativo do site automaticamente com uma interface agradável a partir dos modelos de dados utilizando o componente de ORM.

Por meio do componente de formulários é possível aproveitar as validações que o Django cria a partir do modelo de dados escritos pelo desenvolvedor, diminuindo gradativamente o tempo gasto na modelagem dos formulários do sistema.

O componente URL Dispatcher contribui com a segurança dos arquivos do site, pois, ele cuida da parte de mascarar as urls, com isso o desenvolvedor ganha em termos de flexibilidade e segurança, escondendo realmente onde estão os arquivos do servidor. O Framework possui um arquivo de configuração, onde todas as urls ficam e o desenvolvedor pode customizá-las.

O Django possui um sistema de Templates muito ágil, que utiliza uma linguagem de Templates exclusiva, extensível e amigável. Com esse poderoso componente que o Framework disponibiliza, o desenvolvedor separa o código Python do design, que serão HTML e CSS, o que, desse modo facilita a manutenção do código fonte em futuras atualizações.⁷

Além de todos esses componentes que beneficiam o desenvolvimento, ele dispõe do componente de internacionalização que dá suporte à aplicação em diversos idiomas

⁶ SANTANA, Osvaldo; GALESI, Thiago. PYTHON e DJANGO – Desenvolvimento ágil de aplicações web. Novatec Editora Ltda. São Paulo, 2010, p. 155.

⁷ DJANGO BRASIL. O framework Django. 2011. Disponível em: < <http://www.djangobrasil.org/>>. Acesso em: 03 set. 2011.

(multi-idíomas). O desenvolvedor pode especificar as *strings* de tradução do sistema em cada arquivo e cada arquivo estará representado em um idioma diferente. O Django exibe todas as línguas especificadas na aplicação.

1.3.1 MAPEAMENTO OBJETO-RELACIONAL

ORM - Object-Relational Mapping (Mapeamento Objecto-Relacional) é um componente disponível no Framework, que permite que o programador crie sua aplicação, utilizando objetos sem se preocupar com a persistência desses dados no seu banco de dados. Essa camada de software abstrai toda a comunicação com seu banco de dados, permitindo que o desenvolvedor manipule seus objetos sem o uso da linguagem SQL.

O Django permite o desenvolvedor acessar diretamente o banco de dados em uma aplicação, mas esse tipo de uso só se torna interessante na otimização de determinadas *queries*. Então, o programador define a modelagem de dados através de classes em Python, e, partindo destas classes será possível gerar suas tabelas no banco de dados e manipulá-las sem necessidade de utilizar código SQL. Este componente permite que o desenvolvedor faça o passo inverso também, basta ele configurar a aplicação com o modelo relacional existente, o qual deve possuir as tabelas já criadas, a partir destas tabelas o componente gera todas as classes para a aplicação.

1.3.2 INTERFACE ADMINISTRATIVA

O Django disponibiliza uma interface administrativa automática com um design pronto. O Framework lê no seu modelo de metadados para fornecer uma interface agradável, onde o usuário já pode manipular o conteúdo do site no mesmo instante adicionando ao site novas informações. O desenvolvedor pode customizar o administrativo do site, alterando os modelos de dados, adicionando filtros, buscando ou até mesmo alterando campos que serão visualizados ou editados.

No componente de admin do Django é possível criar várias instâncias do administrativo do site, basta criar diversas chamadas do admin, alterando a raiz no URLConf. A opção de criar diversos administrativos pode ser interessante caso o sistema possua particularidades que só alguns administradores possam alterá-las ou alguns

recursos, como no caso de super administradores e administradores básicos com permissões específicas.

1.3.3 FORMULÁRIOS

Por meio do componente de formulários do Django é possível gerar formulários automaticamente, através dos modelos de dados. A biblioteca de formulários do Django: gera automaticamente um formulário HTML com os campos do formulário; verifica os dados do formulário com um conjunto de regras de validação; no caso de erros de validação nos campos, reexibe o formulário com os alertas de erros, que também converte os dados do formulário submetido para os tipos de dados que o Python reconhece. A biblioteca de formulários possui subclasses que são responsáveis por uma parte do formulário, como: widget, field, form e form media.

A classe widget é uma classe responsável pelos elementos HTML apresentados no formulário, por exemplo `<input type="text">` ou `<textarea>`, que são os campos do formulário.

A classe field já cuida da parte de validação dos campos, por exemplo, EmailField, que é uma validação disponível no Framework Django que verifica se o dado informado é um e-mail válido.

O form é uma classe em que o desenvolvedor pode criar uma coleção de campos que será exibido e validado no formulário.

“A Form object encapsulates a sequence of form fields and a collection of validation rules that must be fulfilled in order for the form to be accepted. Form classes are created as subclasses of `Django.forms.Form` and make use of a declarative style that you’ll be familiar with if you’ve used Django’s database models.”⁸

⁸ DJANGO PROJECT. Working with forms. 2011. Disponível em: <<https://docs.djangoproject.com/en/1.3/topics/forms/>>. Acesso em: 03 set. 2011.

Finalmente, form media que é a classe que trata o CSS e Javascript que serão utilizados no formulário.

Como foi explicado anteriormente, um Framework possui diversos componentes integrados e isso é o que o difere de uma biblioteca. A biblioteca do formulário é integrada com outros componentes como a camada de banco de dados e a internacionalização do Django, mas o desenvolvedor não é obrigado a utilizar este recurso de internacionalização.

1.3.4 URL DISPATCHER

No Django não há limitações para criação de urls elegantes e, de maneira simples, mostrando que é uma aplicação Web de alta qualidade. URL dispatcher cuida do processamento das urls do sistema, executando funções especificadas pelo desenvolvedor e possibilitando o uso de urls amigáveis ao usuário. Para escrever as urls da aplicação, o desenvolvedor precisa apenas alterar o arquivo de URLConf, onde todas as urls da aplicação permanecem. O módulo é escrito com um código Python puro que é um mapeamento simples e utiliza-se muito de expressões regulares. Este mapeamento pode ser longo ou curto e pode ter referências a outros mapeamentos. “A clean, elegant URL scheme is an important detail in a high-quality Web application. Django lets you design URLs however you want, with no Framework limitations.”⁹

1.3.5 TEMPLATES

O sistema de Templates do Django foi projetado para alcançar um equilíbrio entre o poder e a facilidade, onde fornece uma linguagem para a criação de templates usados na geração de páginas dinâmicas. Os programadores da linguagem PHP, que misturam o código PHP com o código HTML, precisam considerar que o sistema de templates do Django não é simplesmente um código Python embutido em HTML. Esse sistema, basicamente, recebe variáveis, onde estão os dados que serão exibidos, dados estes que serão tratados na View e depois demonstrados na página. O sistema de templates é usado

⁹ DJANGO PROJECT. URL dispatcher. 2011. Disponível em: <<https://docs.djangoproject.com/en/1.3/topics/http/urls/>>. Acesso em: 25 out. 2011.

para exibir a apresentação dos dados, não a lógica do programa, onde pode gerar qualquer formato baseado em texto (HTML, XML, JSON etc.).

1.3.6 CACHE

O Django apresenta um componente de cache que cuida das requisições de um usuário ao site, isto é muito importante, pois, cada vez que um usuário acessa uma página, o servidor web faz todo o tipo de cálculo, como consultas a banco de dados, renderização de Templates e lógica de negócio para criação da página que o seu visitante precisa ver. Este procedimento tem um custo significativo de processamento. Qual seria o cenário se mais de cem usuários tentassem acessar simultaneamente a página? Para não precisar ficar carregando toda hora a mesma página, o Framework de cache salva algumas informações estáticas na máquina do usuário, com isso não fica requisitando o site inteiro ao servidor toda vez em que ele tenta acessar uma página. O Django oferece vários níveis de granularidade de cache: o desenvolvedor pode escolher algumas Views que deseja salvar no cache ou de todo o site.

“Django comes with a robust cache system that lets you save dynamic pages so they don't have to be calculated for each request. For convenience, Django offers different levels of cache granularity: You can cache the output of specific views, you can cache only the pieces that are difficult to produce, or you can cache your entire site.”¹⁰

1.3.7 INTERNACIONALIZAÇÃO

O sistema de internacionalização do Django tem total suporte para aplicações multi-idioma, deixando o desenvolvedor especificar strings de tradução e fornecendo ganchos para funcionalidades específicas do idioma. O componente de internacionalização permite que uma única aplicação possa disponibilizar diversos conteúdos e funcionalidades em múltiplas linguagens.

¹⁰ DJANGO PROJECT. Django's cache framework. 2011. Disponível em: <<https://docs.djangoproject.com/en/1.3/topics/cache/>>. Acesso em: 25 out. 2011.

O desenvolvedor pode adicionar diversos hooks em seu código Python e templates, hooks estes chamados translation strings (strings de tradução). Elas dizem ao Django que a aplicação deve ser traduzida na linguagem que o usuário escolher. Claro que o usuário só poderá escolher as linguagens que o sistema disponibiliza. O sistema de internacionalização traduz as aplicações web em tempo de execução, conforme o idioma escolhido. O Django, basicamente, realiza as duas seguintes funções no sistema de internacionalização: a primeira delas vem a ser aquilo que o programador seleciona das partes específicas, que será traduzido no Template; a segunda aplicação utiliza os hooks para a tradução e o usuário pode escolher o idioma de sua preferência.¹¹

Caso o desenvolvedor não queira utilizar o componente de internacionalização, é importante desativá-lo, pois, os hooks são habilitados de padrão e para desativá-los basta editar o arquivo de configuração settings.py e alterar *USE_I18N* para *false*. Devido esta modificação a aplicação terá menos processamento e ganhará mais desempenho, como pode ser visto na Figura 1.¹²

```
27 LOGIN_URL = "/login/"
28 LOGOUT_URL = "/logout/"
29 LOGIN_REDIRECT_URL = "/"
30
31 TIME_ZONE = 'America/Sao_Paulo'
32 LANGUAGE_CODE = 'pt-br'
33
34 SITE_ID = 1
35
36 USE_I18N = False
37 USE_L10N = True
38
39 MEDIA_ROOT = os.path.join(ROOTDIR, 'media')
40 MEDIA_URL = '/media/'
```

Figura 1 Desativando o USE_I18N

¹¹ DJANGO PROJECT. Internationalization and localization. 2011. Disponível em: <<https://docs.djangoproject.com/en/1.3/topics/i18n/>>. Acesso em: 27 out. 2011.

¹² DJANGO BRASIL. Internacionalização. 2011. Disponível em: <<http://docs.djangobrasil.org/topics/i18n.html#topics-i18n>>. Acesso em: 25 out. 2011.

2 DESENVOLVIMENTO DO PORTAL EDUCACIONAL

Portal Educacional é um projeto voltado para universidades, pretendendo ser inteiramente dinâmico. O propósito é que alunos e professores possam visualizar as respectivas notas e aulas de qualquer computador ligado à internet. Com a criação de um portal Educacional, é possível desenvolver uma aplicação web com alto nível de complexidade, utilizando diversos recursos interessantes do Framework. Um projeto de Portal Educacional tem uma lógica relacional complexa, desenvolvendo-o com o Django não é necessário se importar com a modelagem em SQL, pois, ele já utiliza o Mapeamento Objeto-Relacional. Com ele é possível criar a modelagem de dados através de classes em Python, gerando as tabelas necessárias para sua aplicação e manipulá-las sem necessidade de utilizar o SQL.

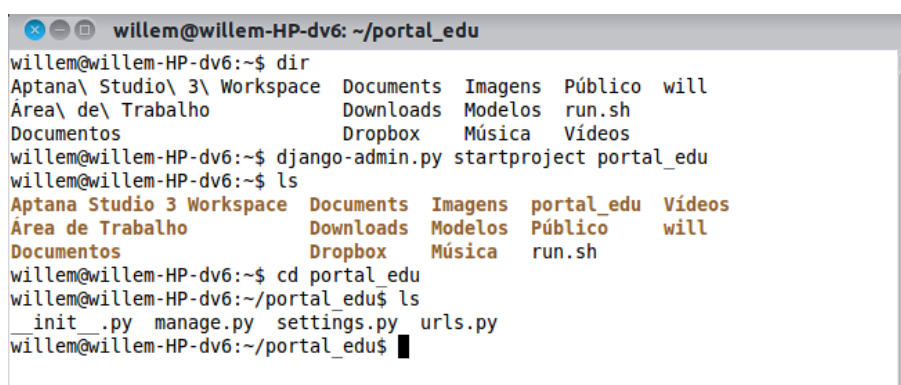
2.1 CONFIGURANDO O AMBIENTE DE TRABALHO

Para configurar o ambiente de trabalho, o primeiro passo é baixar e instalar o Python, caso o sistema operacional seja GNU/Linux ou Mac Os X, que provavelmente já possui o Python instalado. Verifique a versão do Python e, caso esteja na versão 2.6, atualize-o para versão 2.7. Siga o Apêndice A referente à instalação do Python e do Django.

2.2 INICIANDO O PROJETO

Antes de mais nada, é preciso configurar a variável ambiente em seu sistema operacional. Provavelmente, se o programador utilizar Linux ou Mac não precisará desta configuração ao contrário do usuário do Windows. Para configurar a variável ambiente no Windows basta seguir a figura que encontra-se no Apêndice M. Depois, é preciso copiar o arquivo Django-admin.py no diretório path do Python para que possa utilizar os comandos do Framework no terminal com mais facilidade, pois, com o arquivo na raiz do Python não será necessário localizá-lo.

O primeiro passo para iniciar o projeto, é executar o comando para criação da aplicação. Assim, utiliza-se o comando `Django-admin.py startproject portal_edu`. O arquivo `Django-admin.py` é o responsável por toda área administrativa do Django, via linha de comando, que descreve tudo o que se pode fazer com o Framework. Após executar este comando é criada uma pasta `portal_edu` com arquivos padrões de configuração do Framework. O diretório terá três arquivos, eles são: `manage.py`, `settings.py` e `urls.py`, como pode ser vista na Figura 2.



```

willem@willem-HP-dv6: ~/portal_edu
willem@willem-HP-dv6:~$ dir
Aptana\ Studio\ 3\ Workspace  Documents  Imagens  Público  will
Área\ de\ Trabalho           Downloads  Modelos  run.sh
Documentos                   Dropbox   Música   Vídeos
willem@willem-HP-dv6:~$ django-admin.py startproject portal_edu
willem@willem-HP-dv6:~$ ls
Aptana Studio 3 Workspace  Documents  Imagens  portal_edu  Vídeos
Área de Trabalho          Downloads  Modelos  Público    will
Documentos                Dropbox    Música    run.sh
willem@willem-HP-dv6:~$ cd portal_edu
willem@willem-HP-dv6:~/portal_edu$ ls
__init__.py  manage.py  settings.py  urls.py
willem@willem-HP-dv6:~/portal_edu$

```

Figura 2 Iniciando um projeto Django

O arquivo `manage.py` cuida de duas partes importantes no projeto Django, ele é responsável por ajustar o pacote do seu projeto no `sys.path` e também define a variável de ambiente `DJANGO_SETTING_MODULE` para que aponte para o arquivo `settings.py` do seu projeto. Geralmente trabalha-se apenas com um projeto, então utiliza-se o `manage.py`, mas se precisar trabalhar com mais de um projeto utiliza-se o `Django-admin.py`. O Framework Django utiliza muito a linha de comando com códigos para gerar aplicações dentro do projeto.

O arquivo `settings.py` cuida de toda configuração do projeto Django. Neste arquivo é possível fazer configurações básicas e avançadas. É possível ativar o debug no servidor responsável por exibir todos os possíveis erros, facilitando assim a manutenção do código fonte, que é muito útil no decorrer do desenvolvimento da aplicação. Quando o site é divulgado, é importante estar completamente sem erros e depois de confirmado, pode desabilitar esta opção para que não exiba nenhum erro no site para os usuários. Se esses erros forem exibidos no site em produção, algum usuário mal intencionado pode usufruir das falhas do sistema e buscar brechas para uma invasão. Por isso, é sempre bom

configurá-lo o melhor possível. No settings é possível configurar o e-mail do administrador, diretórios de Templates, internacionalização da aplicação e outros diversos recursos.

Na Figura 3 podem ser visualizadas as configurações do banco de dados, internacionalização, data e hora, idioma, configurações do administrador e url do admin de login e logout. A aplicação utilizar-se o SQLite3 como banco de dados, gerando um arquivo no formato db.

```

1  import os
2
3  ROOTDIR = os.path.dirname( file )
4
5  DEBUG = True
6  TEMPLATE_DEBUG = DEBUG
7
8  ADMINS = (
9      ....('Willem', 'willemarf@gmail.com'),
10  )
11
12  MANAGERS = ADMINS
13
14  DATABASES = {
15      ---- ....'default': {
16          ---- ..... 'ENGINE': 'django.db.backends.sqlite3',
17          ---- ..... 'NAME': 'portaledu.db',
18          ---- ..... 'USER': '',
19          ---- ..... 'PASSWORD': '',
20          ---- ..... 'HOST': '',
21          ---- ..... 'PORT': '',
22          ---- .....}
23  }
24
25  LOGIN_URL = "/login/"
26  LOGOUT_URL = "/logout/"
27  LOGIN_REDIRECT_URL = "/"
28
29  TIME_ZONE = 'America/Sao_Paulo'
30  LANGUAGE_CODE = 'pt-br'
31
32  SITE_ID = 1
33
34  USE_I18N = True
35  USE_L10N = True

```

Figura 3 Arquivo de configuração do Django parte 1

Na linha 1 da Figura 3, foi utilizado o comando *import os*, que é o responsável por incluir uma biblioteca com diversos comandos referentes ao sistema operacional. Logo na linha 3, utilizando a biblioteca *os* com o comando *os.path.dirname* que salva o caminho do diretório *root* na variável *ROOTDIR*. Na linha 29, o comando *TIME_ZONE* recebe um valor referente à região da aplicação. Assim, o desenvolvedor poderá utilizar as funções data/hora, sendo o horário exibido de acordo com a região determinada pelo comando *TIME_ZONE*. A seguir, na linha 30 o comando *LANGUAGE_CODE* armazena o idioma da aplicação. Por sua vez, os comandos das linhas 34 e 35 referem-se à tradução e à formatação, que envolvem a internacionalização e localização. As linhas 5 e 6 são utilizadas para habilitar o *debug* da aplicação e as linhas 25, 26 e 27 são relativas aos links do administrativo.

Por sua vez, a linha 14 inicia o código de configuração da aplicação com o banco de dados, porém, o desenvolvedor pode ou não criar outras configurações para variados bancos de dados, como por exemplo, de uma base para testes e outra para produção. Para configurar o banco SQLite3 é necessário configurar o tipo de banco de dados em *ENGINE* e configurar o nome do arquivo em *NAME*.

Na Figura 4, podem-se visualizar as seguintes configurações: diretórios dos Templates, diretórios do administrativo e arquivos estáticos.

No arquivo settings podem ser observadas as configurações dos pacotes criados pelo desenvolvedor como: curso, turma, professor, aluno, dentre outros, conforme a Figura 5 abaixo.

Dessa forma, o autor optou por configurar apenas os comandos assinalados, onde serão explicados no decorrer do trabalho.

```

37 MEDIA_ROOT = os.path.join(ROOTDIR, 'media')
38 MEDIA_URL = '/media/'
39
40 STATIC_ROOT = os.path.join(ROOTDIR, 'public')
41 STATIC_URL = '/static/'
42
43 ADMIN_MEDIA_PREFIX = STATIC_URL + 'admin/'
44
45 STATICFILES_DIRS = (
46     — os.path.join(ROOTDIR, 'static'),
47 )
48
49 STATICFILES_FINDERS = (
50     ....'django.contrib.staticfiles.finders.FileSystemFinder',
51     ....'django.contrib.staticfiles.finders.AppDirectoriesFinder',
52     #     'django.contrib.staticfiles.finders.DefaultStorageFinder',
53 )
54
55 SECRET_KEY = 'r839bo)q^ksn2%c_9uh)ta+6mj1$&-0rk&mpm5sv*kfc)ec2u'
56
57 TEMPLATE_LOADERS = (
58     ....'django.template.loaders.filesystem.Loader',
59     ....'django.template.loaders.app_directories.Loader',
60     #     'django.template.loaders.eggs.Loader',
61 )
62
63 TEMPLATE_DIRS = (
64     — os.path.join(ROOTDIR, 'templates'),
65 )
66
67 MIDDLEWARE_CLASSES = (
68     ....'django.middleware.common.CommonMiddleware',
69     ....'django.contrib.sessions.middleware.SessionMiddleware',
70     ....'django.middleware.csrf.CsrfViewMiddleware',
71     ....'django.contrib.auth.middleware.AuthenticationMiddleware',
72     ....'django.contrib.messages.middleware.MessageMiddleware',
73 )
74
75 ROOT_URLCONF = 'portal_edu.urls'

```

Figura 4 Arquivo de configuração do Django parte 2

```
78 INSTALLED_APPS = (  
79     .... 'django.contrib.auth',  
80     .... 'django.contrib.contenttypes',  
81     .... 'django.contrib.sessions',  
82     .... 'django.contrib.admin',  
83     .... 'django.contrib.sites',  
84     .... 'django.contrib.messages',  
85     .... 'django.contrib.staticfiles',  
86     .... 'home',  
87     .... 'curso',  
88     .... 'turma',  
89     .... 'aluno',  
90     .... 'professor',  
91     .... 'disciplina',  
92     .... 'equipamento',  
93     .... 'material',  
94     .... 'disciplinaAluno',  
95     .... 'professorEquipamento',  
96 )
```

Figura 5 Arquivo de configuração do Django parte 3

Após a criação do projeto e configurado o settings.py com as opções desejadas, é chegado o momento de colocar mão na massa. Próximo passo é criar as apps do site. A app é uma aplicação menor dentro do site. Exemplos de app no site em processo de desenvolvimento são: aluno, professor, disciplina, material dentre outras como pode ser visualizado na Figura 5 acima. O Framework utiliza essa filosofia de separar o site em várias apps para facilitar o desenvolvimento, todos os códigos referentes aos alunos estarão na pasta da app do aluno e assim em diante para as outras apps.

2.3 CRIANDO UMA APP

O Framework possui um código pré-definido na criação da app. Novamente, via terminal, o desenvolvedor está na pasta raiz do seu projeto, onde possui os três arquivos gerados pelo Django. Eles são: settings.py, manage.py e urls.py. O comando que deve ser

executado é o *python manage.py startapp nome_da_aplicação*, um exemplo na utilização seria *python manage.py startapp aluno*.

```
MacBook-de-Willem:portal_edu willem$ python manage.py startapp aluno
MacBook-de-Willem:portal_edu willem$ python manage.py startapp professor
MacBook-de-Willem:portal_edu willem$ python manage.py startapp curso
MacBook-de-Willem:portal_edu willem$ python manage.py startapp disciplina
MacBook-de-Willem:portal_edu willem$ python manage.py startapp disciplinaAluno
MacBook-de-Willem:portal_edu willem$ python manage.py startapp equipamento
MacBook-de-Willem:portal_edu willem$ python manage.py startapp home
MacBook-de-Willem:portal_edu willem$ python manage.py startapp material
MacBook-de-Willem:portal_edu willem$ python manage.py startapp professorEquipamento
MacBook-de-Willem:portal_edu willem$ python manage.py startapp turma
MacBook-de-Willem:portal_edu willem$ ls
__init__.py          disciplina            manage.py            settings.py
__init__.pyc         disciplinaAluno      material             settings.pyc
aluno                equipamento          professor            turma
curso                home                 professorEquipamento  urls.py
MacBook-de-Willem:portal_edu willem$ █
```

Figura 6 Criação das apps do portal

Como pode-se visualizar na Figura 6 acima, todas as apps criadas. Na Figura 7, mostra que cada app terá quatro arquivos.

```
MacBook-de-Willem:portal_edu willem$ cd aluno
MacBook-de-Willem:aluno willem$ ls
__init__.py  models.py  tests.py  views.py
MacBook-de-Willem:aluno willem$ █
```

Figura 7 Visualizando os arquivos da app aluno

Após a criação das apps do projeto é preciso adicionar aos pacotes instalados dentro do settings.py, conforme a Figura 8. As aplicações marcadas são os pacotes criados e adicionados pelo desenvolvedor.

```

INSTALLED_APPS = (
    ....'django.contrib.auth',
    ....'django.contrib.contenttypes',
    ....'django.contrib.sessions',
    ....'django.contrib.admin',
    ....'django.contrib.sites',
    ....'django.contrib.messages',
    ....'django.contrib.staticfiles',
    ....'home',
    ....'curso',
    ....'turma',
    ....'aluno',
    ....'professor',
    ....'disciplina',
    ....'equipamento',
    ....'material',
    ....'disciplinaAluno',
    ....'professorEquipamento',
)

```

Figura 8 Apps instaladas no settings

2.4 MODELS

O modelo é onde o programador cria o esquema de dados visando à modelagem do banco de dados. Basicamente, o desenvolvedor cria uma classe no model. O nome da classe fica sendo o nome da tabela e os atributos da classe que são os campos do banco.

“**Model:** contém a representação dos dados do projeto. Normalmente, é a parte que conversa com o banco de dados e que persistem seus dados. No caso do Django, o sistema de ORM frequentemente é usado na criação de nossas classes de Model.”¹³

O Framework disponibiliza uma API de acesso ao banco de dados, que possibilita manipular informações do banco de dados sem precisar utilizar código SQL. O Framework adapta o código SQL gerado, dependendo do banco de dados que o desenvolvedor define. O Django gera as tabelas a partir dos Models da aplicação, onde o nome da tabela é o nome da classe junto ao nome da app, isto sem que o desenvolvedor precise modelar o

¹³ SANTANA, Osvaldo; GALESI, Thiago. PYTHON e DJANGO – Desenvolvimento ágil de aplicações web. Novatec Editora Ltda. São Paulo, 2010, p. 156.

banco. Com o Framework é possível forçar um nome para a tabela sem que o Framework atribua um apelido para a tabela. Para tanto, basta adicionar a *class Meta* e o comando *db_table* então o carácter “u” na frente do nome da tabela indica que a codificação é do tipo utf8. O desenvolvedor não pode esquecer de adicionar o nome das apps criadas no *settings.py* na tupla *INSTALLED_APP*. Após a criação de todos Models e adicionados ao *settings.py* é preciso rodar o comando *manage.py syncdb*, este comando, que se encarrega de criar as tabelas no banco de dados. Pode-se visualizar o exemplo do portal, analisando o modelo aluno na Figura 9.

```

1  from django.db import models
2
3  class Aluno(models.Model):
4      nome = models.CharField(max_length=100,null=False)
5      login = models.CharField(max_length=45,null=False,unique=True)
6      senha = models.CharField(max_length=45,null=False)
7
8      class Meta:
9          db_table = u'aluno'
10
11     def __unicode__(self):
12         return str(self.id) + ' - ' + self.nome

```

Figura 9 Model aluno

Os atributos nome, login e senha são campos do tipo *varchar(45)* e *not null* como foi especificado através do objeto Models. O Framework cria a chave primária automaticamente para o modelo. Caso o programador não queira criar essa chave é necessário demarcar sua opção no modelo.

O Django gera automaticamente as telas administrativas, como já foi falado anteriormente. Quando o usuário acessa uma tela que possui relacionamentos com outra tabela, no caso a tabela aluno, é exibido o *fieldsets* do aluno para selecionar e aparece apenas o id do aluno. O administrador do sistema não sabe apenas pelo código qual aluno selecionar para melhorar a viabilidade do administrativo, devendo alterar no Model a função *__unicode__*, que é a responsável por exibir o id do aluno no *fieldsets*, e exibirá o id junto ao nome do aluno.

```
CREATE aluno (
  ..id int(11) NOT NULL AUTO_INCREMENT,
  ..nome varchar(100) NOT NULL,
  ..login varchar(45) NOT NULL,
  ..senha varchar(45) NOT NULL,
  ..PRIMARY KEY (id),
  ..UNIQUE KEY login (login)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Figura 10 Código SQL gerado a partir do modelo aluno

2.4.1 CAMPOS

Os campos que são definidos no modelo são os das colunas do banco de dados, nos quais eles são especificados por meio de atributos de classe.

2.4.2 TIPOS DE CAMPOS

O programador deve especificar qual instância da classe `Field` apropriada para cada campo. O Framework possui os tipos das classes definidas que podem ser utilizadas nos campos. Algumas coisas que as classes determinam: o tipo da coluna do banco de dados (ex: `INTEGER`, `VARCHAR`); o *widget*, que pode ser usado na interface administrativa do Django (ex: `<input type="text">`, `<select>`); e, os requisitos mínimos para validação, que são usados no admin do Django e nos formulários automaticamente gerados.

O Django disponibiliza diversos tipos de campos definidos, e, caso o programador não esteja satisfeito com os tipos disponíveis, pode criar o seu próprio tipo de campo.

2.4.3 OPÇÕES DOS CAMPOS

Essas opções são argumentos específicos recebidos dependendo do tipo do campo, por exemplo, o `CharField` obriga o programador a especificar o *max_lenght*, que é o tamanho do campo, o qual é também obrigatório ao `VARCHAR` no banco de dados. Existem argumentos que são comuns aos diversos tipos de campos, alguns deles são: *null*, *blank*, *choices*, *default*, *help_text*, *primary_key* e *unique*.

No argumento *null*, caso seja atribuído *True*, será salvo *NULL*. Se o desenvolvedor não passar o argumento *null*, o campo será marcado como *False* automaticamente.

Também o argumento *blank*, se não passado como argumento, é marcado *False* como padrão. Caso o argumento *blank* tenha passado como *True*, o campo pode ser vazio. Na eventualidade do *blank* ser marcado como *False* é obrigatório ser preenchido. Vale lembrar que há uma grande diferença entre o argumento *null* e o argumento *blank*. O argumento *blank* é usado para validação dos campos, enquanto o *null* é usado para salvar os dados no banco de dados com o valor *NULL*, caso os dados estejam nulos.

O argumento *choices* pode ser uma lista ou tupla, ele é utilizado para criar uma caixa de seleção no lugar de um campo de texto comum. Ele tem a intenção de limitar as opções, obrigando o usuário a escolher uma daquelas definidas pelo desenvolvedor. O primeiro elemento passado da tupla é o valor que será salvo e o segundo elemento será exibido na interface na caixa de seleção para o usuário.

Caso o programador queira colocar um valor padrão para o campo é utilizado o *default*.

O parâmetro *help_text* é usado para colocar um texto de ajuda no campo, assim o programador pode apresentar um texto que auxilia o usuário a preencher o campo.

Para utilizar o parâmetro *primary_key* basta adicioná-lo com o valor *True* no argumento do campo desejado. Caso não seja usado nenhum *primary_key*, o Django, automaticamente cria um campo chave para o modelo.

Caso não queira que os campos tenham dados repetidos, basta passar o argumento *unique=True*.

2.4.4 RELACIONAMENTOS

O Framework possibilita ao programador utilizar os três tipos de relacionamentos comuns: um-para-um, muitos-para-um e muitos-para-muitos.

Para utilizar o relacionamento de um-para-um é preciso criar um campo do tipo *OneToOneField* e também é necessário passar um argumento dizendo qual o Model relacionado.

No relacionamento de muitos-para-um também é preciso criar um campo e utilizar o objeto Models para passar um argumento, o qual diz a que classe corresponde, o que é praticamente idêntico ao *OneToOneField*, diferindo apenas no que concerne ao tipo de campo, o *ForeignKey*.

Enfim, o relacionamento de muitos-para-muitos. Para usá-lo é preciso criar um campo e atribuir o tipo *ManyToManyField* para o atributo de classe do seu Model. Depois, é necessário passar como argumento, definindo a classe do relacionamento. Neste tipo de relacionamento é possível associar outros dados com o relacionamento entre dois Models.

Neste exemplo, no relacionamento de muitos-para-muitos pode visualizar-se que foram criadas três classes: Person, Group e Membership. A classe Group possui um atributo relacionado com a classe Membership. Esta possui todos os dados do relacionamento: id da primeira classe, id da segunda classe e os campos extras. O atributo members utiliza o tipo de campo *ManyToManyField* e passa como primeiro argumento a classe Person, que é o objeto onde concebem os dados da pessoa e o segundo argumento *through='Membership'* onde constam as informações restantes do relacionamento, como pode ser visto na Figura 11.

```

1 class Person(models.Model):
2     name = models.CharField(max_length=128)
3
4     def __unicode__(self):
5         return self.name
6
7 class Group(models.Model):
8     name = models.CharField(max_length=128)
9     members = models.ManyToManyField(Person, through='Membership')
10
11    def __unicode__(self):
12        return self.name
13
14 class Membership(models.Model):
15     person = models.ForeignKey(Person)
16     group = models.ForeignKey(Group)
17     date_joined = models.DateField()
18     invite_reason = models.CharField(max_length=64)

```

Figura 11 Exemplo adquirido no site Django Brasil

Abaixo segue a Figura 12, referente ao código do modelo curso. Este possui apenas um atributo chamado *nome* em sua classe. O desenvolvedor criou um apelido para a tabela na linha 7, através do comando `db_table = u'curso'` e também alterou o método `__unicode__`, que por padrão exibe apenas o valor do id. Após o programador alterar o método `__unicode__` será exibido o id e o nome.

```

1  from django.db import models
2
3  class Curso(models.Model):
4      nome = models.CharField(max_length=150,null=False)
5
6      class Meta:
7          db_table = u'curso'
8
9      def __unicode__(self):
10         return str(self.id) + ' - ' + self.nome

```

Figura 12 Modelo curso

As Figuras 12 e 13 são praticamente idênticas, porém, a classe professor tem o apelido diferente e possui dois campos adicionais, login e senha.

```

1  from django.db import models
2
3  class Professor(models.Model):
4      nome = models.CharField(max_length=100,null=False)
5      login = models.CharField(max_length=45,null=False,unique=True)
6      senha = models.CharField(max_length=45,null=False)
7
8      class Meta:
9          db_table = u'professor'
10
11     def __unicode__(self):
12         return str(self.id) + ' - ' + self.nome

```

Figura 13 Modelo professor

A Figura 14 possui o código igual ao da Figura 12, a única diferença é o nome da tabela.

```

1 from django.db import models
2
3 class Turma(models.Model):
4     nome = models.CharField(max_length=150, null=False)
5
6     class Meta:
7         db_table = u'turma'
8
9     def __unicode__(self):
10         return str(self.id) + ' - ' + self.nome

```

Figura 14 Modelo turma

O Model disciplina possui duas chaves estrangeiras, professor e curso. Para o relacionamento é utilizado o comando *models.ForeignKey*, tendo como parâmetro o nome da app, seguido de um ponto e depois o nome da classe, de acordo com as Figuras 15 e 16.

```

1 from django.db import models
2
3 class Disciplina(models.Model):
4     nome = models.CharField(max_length=100, null=False)
5     professor_id = models.ForeignKey('professor.Professor')
6     curso_id = models.ForeignKey('curso.Curso')
7
8     class Meta:
9         db_table = u'disciplina'
10
11     def __unicode__(self):
12         return str(self.id) + ' - ' + self.nome

```

Figura 15 Modelo disciplina

```

1 from django.db import models
2
3 class ProfessorEquipamento(models.Model):
4     professor_id = models.ForeignKey('professor.Professor', null=False)
5     equipamento_id = models.ForeignKey('equipamento.Equipamento', null=False)
6     data_inicial = models.DateTimeField(null=False)
7     data_final = models.DateTimeField(null=False)
8     sala = models.CharField(max_length=3, null=False)
9
10     class Meta:
11         db_table = u'professor_equipamento'
12
13     def __unicode__(self):
14         return str(self.professor_id.nome) + ' - ' + self.equipamento_id.nome

```

Figura 16 Modelo professorEquipamento

O modelo do material possui algumas configurações extras em relação aos modelos anteriores. Neste modelo temos o *ordering*, que especifica qual a ordem em que serão exibidos os dados. O comando *verbose_name* é usado para exibir o nome da app no singular, enquanto o comando *verbose_name_plural* é utilizado para mostrar o texto no plural, como pode ser visto na Figura 17.

```

1  # -*- coding: utf-8 -*-
2  from django.db import models
3
4  class Material(models.Model):
5      nome = models.CharField(max_length=100, null=False)
6      descricao = models.TextField()
7      data = models.DateTimeField(auto_now_add=True)
8      arquivo = models.FileField(upload_to='material')
9      disciplina_id = models.ForeignKey('disciplina.Disciplina')
10
11      class Meta:
12          db_table = u'material'
13          ordering = ["data"]
14          verbose_name = u"Material"
15          verbose_name_plural = u"Materiais"
16
17      def __unicode__(self):
18          return str(self.id) + ' - ' + self.nome

```

Figura 17 Modelo material

Na Figura 18 foi usado o argumento *choices* no campo *status*. Esse argumento recebe uma lista de opções estáticas a serem exibidas em futuros formulários.

```

1  # -*- coding: utf-8 -*-
2  from django.db import models
3
4  class Equipamento(models.Model):
5      STATUS_CHOICES = (
6          ('D', 'Disponível'),
7          ('I', 'Indisponível'),
8          ('M', 'Manutenção'),
9      )
10     nome = models.CharField(max_length=100, null=False)
11     cod_patrimonio = models.CharField(max_length=100, null=False)
12     status = models.CharField(max_length=1, choices=STATUS_CHOICES)
13
14     class Meta:
15         db_table = u'equipamento'
16
17     def __unicode__(self):
18         return str(self.id) + ' - ' + self.nome

```

Figura 18 Modelo equipamento

Nos atributos *nota1*, *nota2*, *exame* e *falta* foram utilizados o tipo `DecimalField`, que possui o argumento *decimal_places* usado para definir o número de casas decimais, de acordo com a Figura 19.

```

1  # -*- coding: utf-8 -*-
2  from django.db import models
3
4  class DisciplinaAluno(models.Model):
5      disciplina_id = models.ForeignKey('disciplina.Disciplina', null=False)
6      aluno_id = models.ForeignKey('aluno.Aluno', null=False)
7      turma_id = models.ForeignKey('turma.Turma', null=False)
8      nota1 = models.DecimalField(max_digits=4, decimal_places=2, default=0, null=False)
9      nota2 = models.DecimalField(max_digits=4, decimal_places=2, default=0, null=False)
10     exame = models.DecimalField(max_digits=4, decimal_places=2, default=0, null=False)
11     falta = models.DecimalField(max_digits=4, decimal_places=2, default=0, null=False)
12
13     class Meta:
14         db_table = 'disciplina_aluno'
15
16     def __unicode__(self):
17         return str(self.disciplina_id.nome) + ' - ' + self.aluno_id.nome + ' - ' + self.turma_id.nome

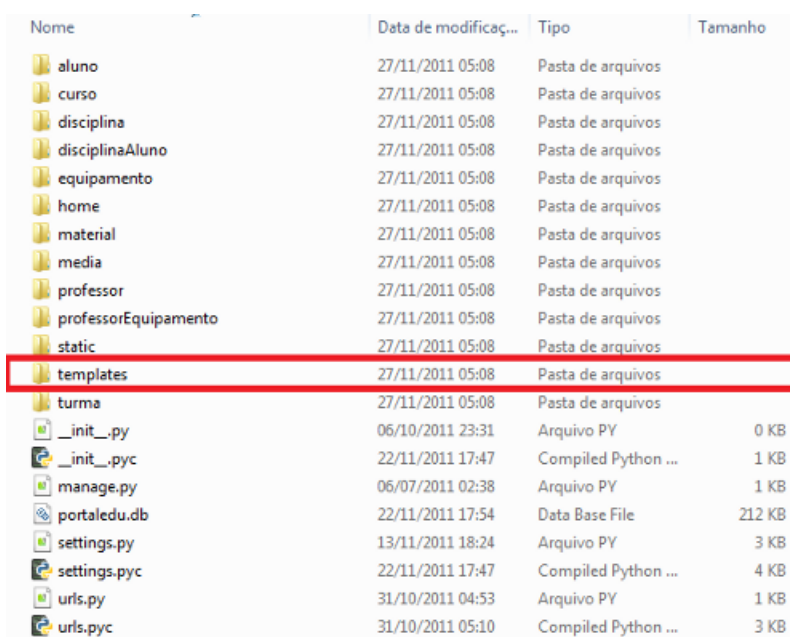
```

Figura 19 Modelo disciplinaAluno

2.5 TEMPLATES

O Framework Django oferece um sistema de Templates para criação de páginas. Este sistema é encarregado de transformar as variáveis do arquivo em valores que são obtidos pela View. O sistema de Templates possui diversas estruturas de controle que o transformam em um tipo de linguagem de programação.

O primeiro passo é criar o diretório na pasta raiz do projeto *portal_edu*, como pode ser visualizado na Figura 20.



Nome	Data de modificaç...	Tipo	Tamanho
aluno	27/11/2011 05:08	Pasta de arquivos	
curso	27/11/2011 05:08	Pasta de arquivos	
disciplina	27/11/2011 05:08	Pasta de arquivos	
disciplinaAluno	27/11/2011 05:08	Pasta de arquivos	
equipamento	27/11/2011 05:08	Pasta de arquivos	
home	27/11/2011 05:08	Pasta de arquivos	
material	27/11/2011 05:08	Pasta de arquivos	
media	27/11/2011 05:08	Pasta de arquivos	
professor	27/11/2011 05:08	Pasta de arquivos	
professorEquipamento	27/11/2011 05:08	Pasta de arquivos	
static	27/11/2011 05:08	Pasta de arquivos	
templates	27/11/2011 05:08	Pasta de arquivos	
turma	27/11/2011 05:08	Pasta de arquivos	
__init__.py	06/10/2011 23:31	Arquivo PY	0 KB
__init__.pyc	22/11/2011 17:47	Compiled Python ...	1 KB
manage.py	06/07/2011 02:38	Arquivo PY	1 KB
portaledu.db	22/11/2011 17:54	Data Base File	212 KB
settings.py	13/11/2011 18:24	Arquivo PY	3 KB
settings.pyc	22/11/2011 17:47	Compiled Python ...	4 KB
urls.py	31/10/2011 04:53	Arquivo PY	1 KB
urls.pyc	31/10/2011 05:10	Compiled Python ...	3 KB

Figura 20 Diretório raiz do projeto

Após a criação do diretório, o desenvolvedor precisa configurar o arquivo *settings.py* e definir o diretório de Templates. Veja a Figura 21 abaixo.

```
1 import os
2
3 ROOTDIR = os.path.dirname(__file__)
```

Figura 21 Configurando o settings para o uso de templates parte 1

O comando acima *os.path.dirname(__file__)* aponta o diretório raiz do projeto para não ter futuros problemas, independentemente do sistema operacional. Antes de

começar os Templates é preciso apontar no settings também o diretório de Templates, Figura 22.

```

63 TEMPLATE_DIRS = (
64     — os.path.join(ROOTDIR, 'templates'),
65 )

```

Figura 22 Configurando o settings para o uso de templates parte 2

O desenvolvedor criou uma página padrão em HTML, que será exibida em todas outras páginas do site, evitando a repetição de código. A página terá o título do portal e os arquivos de CSS para design.

```

1 <html>
2 ..<head>
3 ....{% block cabecalho %}
4 ....<title>Portal EDU</title>
5 ....<link rel="stylesheet" type="text/css" href="{% STATIC_URL %}css/estilo.css" />
6 ....{% endblock %}
7 ..</head>
8 ..<body>
9 ....<h1>Portal EDU</h1>
10 ....{% block corpo %}
11 ....{% endblock %}
12 ..</body>
13 </html>

```

Figura 23 Template base.html

O arquivo acima tem o nome de base.html e foi salvo dentro do diretório Templates. É possível observar a sintaxe da linguagem de Templates do Django, que é muito parecida com a linguagem Python. Pode ser visto que dentro das tags HTML encontram-se estes comandos “{% %}” e “{{ }}” e dentro destes comandos encontram-se códigos conhecidos do Template do Django, que são encarregados de convertê-los para o HTML. É fácil perceber que o autor separou em dois blocos de comandos, o cabeçalho e o corpo. O cabeçalho contém todos os códigos que estiverem dentro da tag head. O bloco chamando corpo não possui nenhum comando no seu interior, pois, todos os códigos deste bloco estão incluídos na página que se estende a esta página. Exemplificando, todo código

da página `base.html` aparecerá dentro da página `index.html` como pode ser visto na Figura 24.

```

1  {% extends 'base.html' %}
2
3  {% block corpo %}
4
5  <div class="opcao"><a href="/aluno/">Aluno</a></div>
6  <div class="opcao"><a href="/professor/">Professor</a></div>
7  <div class="opcao"><a href="/admin/">Administrador</a></div>
8
9  {% endblock %}

```

Figura 24 Template `index.html`

Foi criado um diretório chamado *static* para colocar as imagens, CSS e outros arquivos estáticos. Outro detalhe importante é a configuração do arquivo “`settings.py`”, que define o caminho dos diretórios de arquivos estáticos denominado *static* e o diretório responsável por salvar arquivos enviados pelos formulários denominados *media*.

```

37  MEDIA_ROOT = os.path.join(ROOTDIR, 'media')
38  MEDIA_URL = '/media/'
39
40  STATIC_ROOT = os.path.join(ROOTDIR, 'public')
41  STATIC_URL = '/static/'
42
43  ADMIN_MEDIA_PREFIX = STATIC_URL + 'admin/'
44
45  STATICFILES_DIRS = (
46  — os.path.join(ROOTDIR, 'static'),
47  )

```

Figura 25 Configurando o settings para o uso de templates parte 3

Foi criado um arquivo de folhas de estilos com as cores, fontes e outras informações de layout, que deve ser salvo dentro do caminho “`static/css/`” especificado na Figura 25.


```

1  body,div,p,td {
2  —font-family: Verdana, Arial, sans-serif;
3  —font-size: 12px;
4  }
5
6  .opcao{
7  ++border: #CCCCCC solid 1px;
8  ++background: #F2F2F2;
9  ++color:#000000;
10 ++margin:5px;
11 ++padding:5px;
12 }
13
14 a{
15 ++color: black;
16 ++text-decoration: none;
17 ++font-weight:bold;
18 }

```

Figura 26 Folha de estilos do portal

2.6 VIEWS

O código da Figura 24 é da página index.html. O primeiro comando estende a página base.html, onde contém a estrutura padrão do layout do site e todos os códigos deste arquivo será exibido dentro do bloco do corpo. Foi criado o esqueleto do portal, demarcando-o em duas partes, cabeçalho e corpo.

O próximo passo é criar a View, que se encarregará de enviar as informações para o Template poder exibi-los. Neste caso, foi criado uma app para index com o nome de home, e o comando usado para criar a app foi “*python manage.py startapp home*”. Para a visualização da página principal do portal é necessário alterar a views.py, que se encontra dentro do diretório home. Foi criado a função index que será chamada no arquivo urls.py, onde cuida de todas as urls do portal. Veja como ficou o arquivo views.py referente à app home na Figura 27.

```

1 from django.shortcuts import render_to_response
2 from django.template import RequestContext
3
4 def index(request):
5     return render_to_response('index.html', {}, context_instance=RequestContext(request))

```

Figura 27 Views home

A função `render_to_response()` visualizada na Figura 27 é responsável pelo carregamento do Template e de enviar os dados para o Template `index.html`. Para que os usuários possam visualizar esta página, o programador precisa adicionar uma linha no arquivo `urls.py`, que fica localizado na raiz do projeto. Veja como ficou o arquivos `urls.py` após a modificação do programador na Figura 28.

```

1 from django.conf.urls.defaults import *
2
3 urlpatterns = patterns('',
4     (r'^$', 'home.views.index'),
5 )

```

Figura 28 Arquivo `urls.py`

A linha 4 da Figura acima é a que mapeia a primeira url na aplicação. O primeiro argumento é uma expressão regular e o segundo argumento é o nome da app, seguido da View, e, por fim, a função `index` é a responsável pela transferência para o Template das informações da página.

A letra 'r' colocada à frente do inicio da expressão regular indica que não se pode descartar nenhum caractere dentro da string a seguir. Veja a seguir a tabela de expressões regulares com os caracteres mais utilizadas no Django.

Tabela 1 – Expressões Regulares ¹⁴

Caractere	Significado
\$	Casa com fim da string
^	Casa com início da string
.	Casa com qualquer coisa.
(e)	Marcam um grupo. Um grupo é um conjunto de caracteres casado em conjunto. Grupos podem ganhar nomes, o que ajuda no processamento das expressões regulares. Pode-se dar um nome para um grupo colocando-se ? P<nome> logo depois do abre parênteses, seguido pela expressão regular do grupo.
+, *, ?, { e }	Marcam quantidade: “+” indica um ou mais caracteres, “*” indica zero ou mais caracteres, “?” indica zero ou um caractere, e “{ }” indicam uma quantidade específica de caracteres. Sempre se referem ao grupo, classe ou elemento à esquerda na expressão regular.
[e]	Marcam uma classe. Exemplo, [0-9] casa com os números entre 0 a 9, [aeiou] casa com vogais.
\d, \D	Casam com dígitos (\d) ou com o que não for dígito (\D).
\s, \S	Casam com caracteres de espaço em branco (\s) ou com o que não for espaço em branco (\S).
\w, \W	Casam com caracteres alfanuméricos (\w) ou com o que não for alfanumérico (\W).
	Operador OU. Exemplo: (a b) casa com 'a' ou 'b'.
\	Caractere de escape. Por exemplo, se quisermos casar algum dos caracteres especiais. Exemplo: \+ casa o caractere '+’.

¹⁴ SANTANA, Osvaldo; GALESI, Thiago. PYTHON e DJANGO – Desenvolvimento ágil de aplicações web. Novatec Editora Ltda. São Paulo, 2010, p. 182.

Para iniciar o servidor e testar a aplicação em questão, basta rodar o comando `python manage.py runserver`, depois abrir o navegador e digitar o endereço. Observe o resultado na Figura 29.

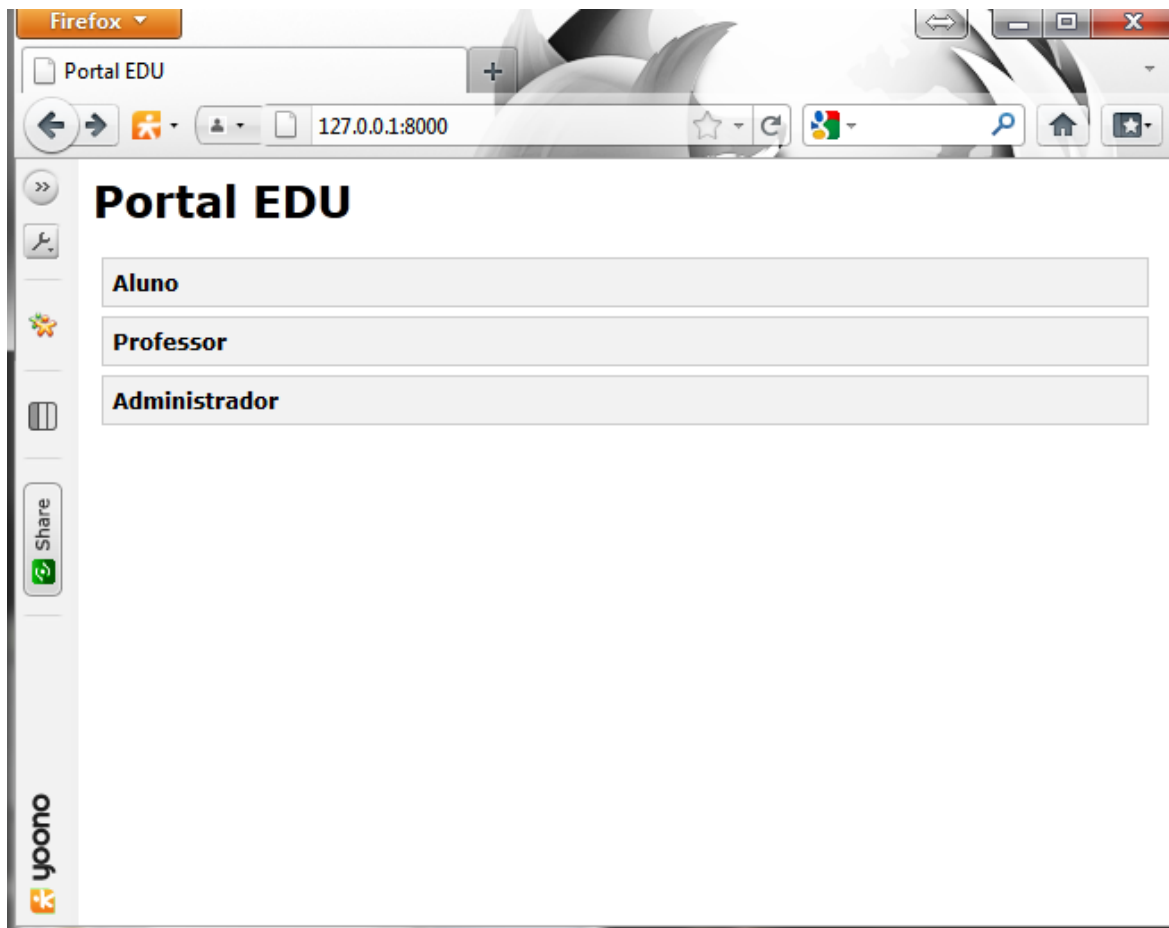


Figura 29 Visualização da tela principal do site no navegador

2.7 FORMULÁRIOS

A aplicação do portal possui dois formulários complexos, o primeiro possibilita ao professor agendar os equipamentos para o uso em sala de aula e o segundo permite ao professor enviar os materiais ou conteúdos para a disciplina desejada.

O formulário de equipamento recebe um valor do equipamento selecionado na tela anterior, como pode ser visto no Template do formulário.

```

1  {% extends 'base.html' %}
2  {% block corpo %}
3  <p>{{ equipamento.nome }}</p>
4  <form action="" method="post">
5  ——— {{ csrf_token %}}
6  ..<input type="hidden" id="equipamento_id" name="equipamento_id" value="{{ equipamento.id }}" >
7  ..<input type="hidden" id="professor_id" name="professor_id" value="{{ professor.id }}" >
8  ..<table>
9  ....<tr>
10 .....<td>{{ form.data_inicial.label_tag }}</td>
11 ....</tr>
12 ....<tr>
13 .....<td>{{ form.data_inicial.errors }}</td>
14 ....</tr>
15 ....<tr>
16 .....<td>{{ form.data_inicial }} Ex: 30/01/2011 19:30:00</td>
17 ....</tr>
18 ....<tr>
19 .....<td>{{ form.data_final.label_tag }}</td>
20 ....</tr>
21 ....<tr>
22 .....<td>{{ form.data_final.errors }}</td>
23 ....</tr>
24 ....<tr>
25 .....<td>{{ form.data_final }} Ex: 30/01/2011 21:05:00</td>
26 ....</tr>
27 ....<tr>
28 .....<td>{{ form.sala.label_tag }}</td>
29 ....</tr>
30 ....<tr>
31 .....<td>{{ form.sala.errors }}</td>
32 ....</tr>
33 ....<tr>
34 .....<td>{{ form.sala }}</td>
35 ....</tr>
36 ....<tr>
37 .....<td>&nbsp;</td>
38 ....</tr>
39 ..</table>
40 ..<button type="submit">Agendar</button>
41 </form>
42 <p><a href="..">Voltar</a></p>
43 {% endblock %}

```

Figura 30 Formulário encarregado de agendar os equipamentos

Foi aproveitado o id do professor da sessão e o id do equipamento selecionado na tela anterior que completam os dados pertinentes ao formulário. “Entre essas informações está um código de segurança usado pelo sistema de proteção contra ataques do tipo cross-site request forgery. Esse código será adicionado ao nosso formulário pela tag {% csrf_token %} no template.”¹⁵

¹⁵ SANTANA, Osvaldo; GALESI, Thiago. PYTHON e DJANGO – Desenvolvimento ágil de aplicações web. Novatec Editora Ltda. São Paulo, 2010, p. 177.

Os comandos que terminam com “.errors” exibem mensagens de erros, de acordo com as validações descritas no Model. Os comandos com o final “.label_tag” exibirão o texto do nome do campo, enquanto os que terminam com o nome do field serão exibidos em uma caixa de texto, dependendo daquela que foi definida no modelo.

Por exemplo, no campo sala foram definidos o CharField e o tamanho máximo dos três caracteres, então será exibido um campo de texto com uma validação até três caracteres. Se após submeter o formulário, a regra não for obedecida, será exibida uma mensagem de erro.

O autor deve escrever a View para tratar os dados recebidos do formulário e também definir o arquivo forms.py com a estrutura herdada do Model, como pode ser visto na Figura 31.

```
1 from django import forms
2 from professorEquipamento.models import ProfessorEquipamento
3
4 class FormProfessorEquipamento(forms.ModelForm):
5     class Meta:
6         model = ProfessorEquipamento
```

Figura 31 Forms.py arquivo que define a estrutura do formulário

Na Figura 31, foi utilizada uma estrutura idêntica ao modelo da app ProfessorEquipamento, pois, o arquivo forms.py está utilizando o código `model = ProfessorEquipamento`. Com esta linha definiu-se o forms.py como igual ao objeto do Model. O programador poderia ter definido alguns campos e deixado outros de fora, mas não é o caso.

```

1 from django.shortcuts import render_to_response
2 from django.template import RequestContext
3 from equipamento.models import Equipamento
4 from forms import FormProfessorEquipamento
5 from models import ProfessorEquipamento
6
7 DIR = 'equipamento/'
8
9 def reservarEquipamento(request, id_equipamento):
10
11     ..if request.method == 'POST':
12     ....form = FormProfessorEquipamento(request.POST)
13     ....if form.is_valid():
14     .....form.enviado = True
15     .....professorEquipamento = form.save()
16     .....return render_to_response(DIR + 'agendado.html', {'professorEquipamento': professorEquipamento})
17
18     ..else:
19     ....form = FormProfessorEquipamento()
20
21     ..VARS = {
22     ....'form':form,
23     ..}
24     —
25     ..try:
26     ....equipamento = Equipamento.objects.get(id=id_equipamento)
27     ....return render_to_response(DIR + 'reservar.html', .
28     ....{'form': form, 'professor': request.session['professor'], 'equipamento': equipamento}, .
29     ....context_instance=RequestContext(request))
30     ..except:
31     ....return render_to_response(DIR + 'erro.html', {'erro': 'Equipamento não encontrado.'})

```

Figura 32 View responsável por tratar os dados recebidos do formulário

Foi desenvolvida uma função chamada `reservarEquipamento`, que, como o nome já o diz, terá a encargo de reservar equipamentos. Ela será carregada duas vezes. A primeira vez que carregar a View do formulário, o objeto `form` estará em branco e, no segundo processamento, o objeto receberá os dados do formulário. Como pode ser visto na Figura 32 acima, foi criado um `if`, que verifica se a requisição proveio do método `POST`. Caso isso não se concretize, significa que a primeira requisição foi recebida através do método `GET`. O `POST` ocorre quando o usuário submete a página, portanto, é preciso instanciar o objeto `form`, uma primeira vez para evitar erros, pois, o referido objeto estará vazio. Após o formulário ser submetido e a View recebido a requisição via `POST`, serão armazenados todos os dados do formulário no objeto `form`. O próximo passo será verificar se o objeto é válido, por exemplo, se a sala foi preenchida e se possui pelo menos três caracteres. Caso a validação seja aceita, o próximo passo será colocar o formulário como enviado, salvar os dados e redirecionar o processo para a página subsequente, que mostrará a mensagem: Agendado com sucesso. Na linha 15 da Figura 32, o código `form.save()` mascarará o código SQL `“INSERT INTO professor_equipamento (...) VALUES (...)”`.

Foi criada uma verificação que impede o usuário de passar o id de um equipamento inexistente. Caso tente alterar a url, passando um código inexistente, essa ação será redirecionada para uma página, que exibirá a mensagem de erro, com o respectivo argumento. Na linha 26 utiliza o componente encarregado de camuflar o banco de dados, o ORM, equivalente ao comando SQL “*SELECT * FROM equipamento WHERE id = id_equipamento*”. O comando SQL está dentro de um try e caso não exista o equipamento solicitado ocorrerá erro, caindo no except, onde será redirecionado para a página erro.html. Havendo sucesso será encaminhado à página reservar.html, que exibirá o formulário. Quando o formulário for validado, aparece na tela a Figura 33, Template agendado.html.

```

1  {% extends 'base.html' %}
2
3  {% block corpo %}
4
5  <ul>
6  ..<li>
7  ....<b>{{ professorEquipamento.equipamento_id.nome }}</b>.
8  ....agendado com sucesso para <b>{{ professorEquipamento.data_inicial }}</b>..
9  ..</li>
10 </ul>
11
12 <p><a href="..">Voltar</a></p>
13 {% endblock %}

```

Figura 33 Template agendado.html

O autor precisou adicionar ao URL Dispatcher as novas urls. Veja o exemplo abaixo, Figura 34.

```

1  from django.conf.urls.defaults import *
2  from django.contrib import admin
3
4  admin.autodiscover()
5
6  urlpatterns = patterns('',
7  — (r'^$', 'home.views.index'),
8  — (r'^aluno/$', 'aluno.views.login'),
9  — (r'^aluno/logout/$', 'aluno.views.logout'),
10 — (r'^aluno/disciplina/$', 'disciplinaAluno.views.disciplinasAluno'),
11 — (r'^professor/$', 'professor.views.login'),
12 — (r'^professor/logout/$', 'professor.views.logout'),
13 — (r'^professor/material/$', 'material.views.material'),
14 — (r'^professor/material/adiciona/$', 'material.views.adiciona'),
15 — (r'^professor/equipamento/$', 'equipamento.views.equipamento'),
16 — (r'^professor/disciplina/$', 'disciplina.views.disciplinasProfessor'),
17 — (r'^professor/disciplina/(?P<id_disciplina>d+)/$', 'disciplinaAluno.views.disciplinasAlunos'),
18 — (r'^professor/nota/aluno/(?P<id_disciplinaAluno>d+)/$', 'disciplinaAluno.views.alterarNotaAluno'),
19 — (r'^professor/equipamento/(?P<id_equipamento>d+)/$', 'professorEquipamento.views.reservarEquipamento'),
20 ....(r'^admin/', include(admin.site.urls)),
21 )

```

Figura 34 Arquivo urls.py completo do portal

Abaixo as figuras referentes à tela de agendamento de equipamento.

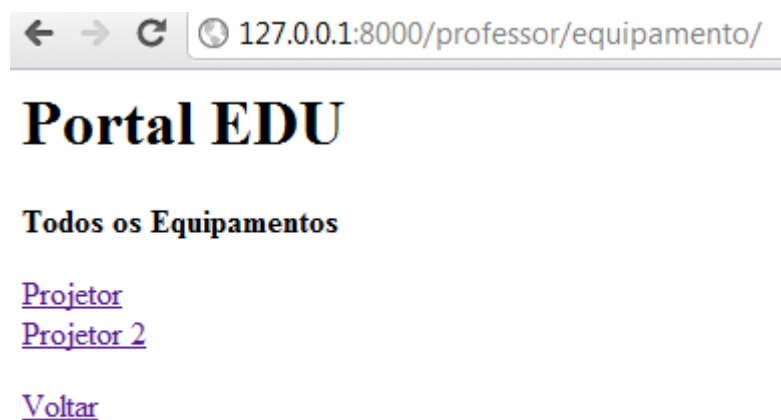
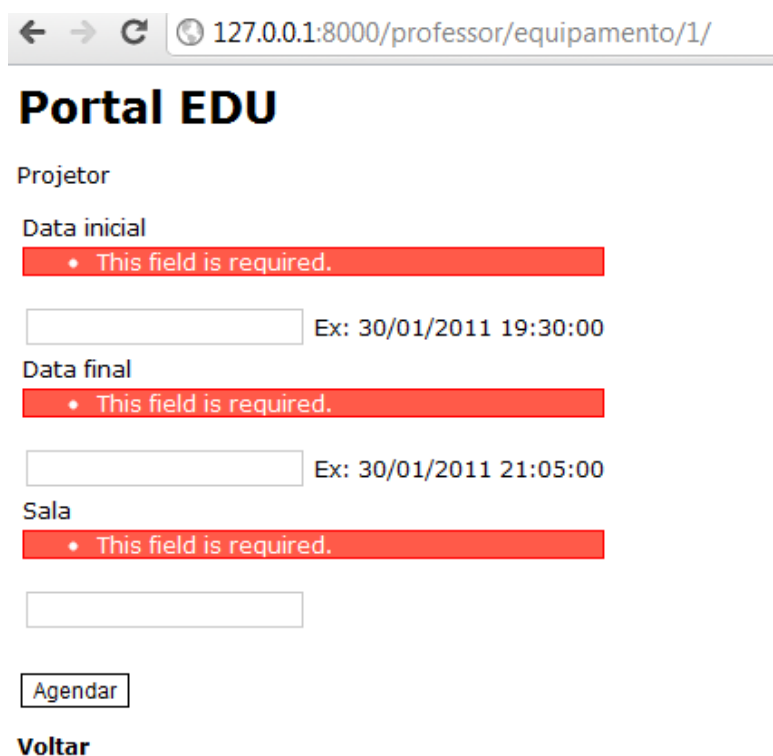


Figura 35 Professor escolhe o equipamento para agendar reserva

A screenshot of a web browser showing the 'Portal EDU' reservation form. The address bar displays '127.0.0.1:8000/professor/equipamento/1/'. The page title is 'Portal EDU'. Below the title, the text 'Projektor' is displayed. There are two input fields for dates: 'Data inicial' and 'Data final'. The 'Data inicial' field has an example value 'Ex: 30/01/2011 19:30:00'. The 'Data final' field has an example value 'Ex: 30/01/2011 21:05:00'. There is an input field for 'Sala'. Below the input fields, there is a button labeled 'Agendar'. At the bottom, there is a link labeled 'Voltar'.

Figura 36 Formulário de reserva do equipamento escolhido na tela anterior



A screenshot of a web browser showing the 'Portal EDU' page. The browser's address bar displays '127.0.0.1:8000/professor/equipamento/1/'. The page title is 'Portal EDU'. Below the title, the section 'Projektor' is visible. It contains three input fields: 'Data inicial', 'Data final', and 'Sala'. Each of these fields has a red error message below it that reads '• This field is required.' The 'Data inicial' field has an example value 'Ex: 30/01/2011 19:30:00' and the 'Data final' field has an example value 'Ex: 30/01/2011 21:05:00'. At the bottom of the form, there is an 'Agendar' button and a 'Voltar' link.

← → ↻ 127.0.0.1:8000/professor/equipamento/1/

Portal EDU

Projektor

Data inicial

- This field is required.

Ex: 30/01/2011 19:30:00

Data final

- This field is required.

Ex: 30/01/2011 21:05:00

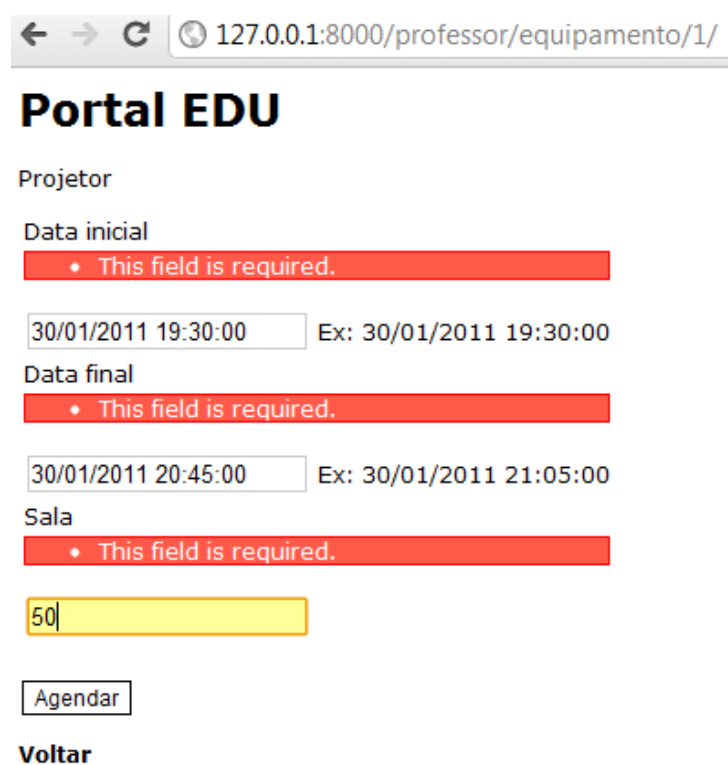
Sala

- This field is required.

Agendar

Voltar

Figura 37 Exibindo os erros de validação do formulário



A screenshot of the same 'Portal EDU' page, but with the form fields filled in. The 'Data inicial' field now contains '30/01/2011 19:30:00' and the 'Data final' field contains '30/01/2011 20:45:00'. The 'Sala' field contains the number '50'. The red error messages are still present. The 'Agendar' button and 'Voltar' link are also visible.

← → ↻ 127.0.0.1:8000/professor/equipamento/1/

Portal EDU

Projektor

Data inicial

- This field is required.

30/01/2011 19:30:00 Ex: 30/01/2011 19:30:00

Data final

- This field is required.

30/01/2011 20:45:00 Ex: 30/01/2011 21:05:00

Sala

- This field is required.

50

Agendar

Voltar

Figura 38 Campos preenchidos

Como pode ser visto na Figura 38, os erros estão em inglês, pois, o autor não traduziu as mensagens padrões de validação do componente de Models e também não foi utilizada a internacionalização de idiomas.

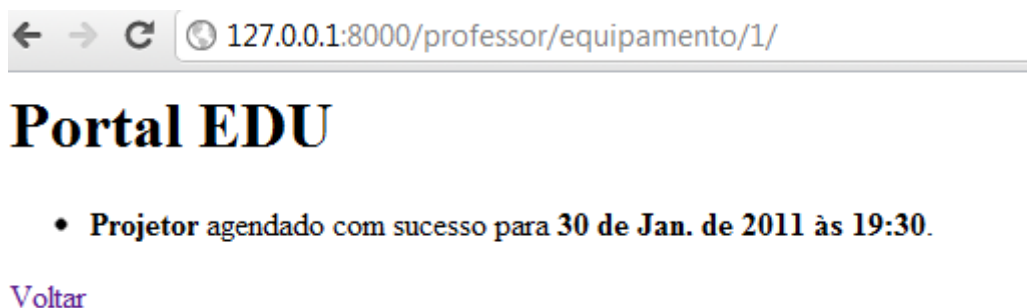


Figura 39 Formulário submetido com sucesso

Outro formulário do portal é a tela de cadastro de materiais do professor. Veja o arquivo do template abaixo na Figura 40 e Figura 41.

```

1  {% extends 'base.html' %}
2  {% block corpo %}
3  <form action="" method="post" enctype="multipart/form-data">
4  ——— {{ csrf_token }}
5  ——— <table>
6  ——— <tr>
7  ——— <td>{{ form.nome.label_tag }}</td>
8  ——— </tr>
9  ——— <tr>
10 ——— <td>{{ form.nome.errors }}</td>
11 ——— </tr>
12 ——— <tr>
13 ——— <td>{{ form.nome }}</td>
14 ——— </tr>
15 ——— <tr>
16 ——— <td>{{ form.descricao.label_tag }}</td>
17 ——— </tr>
18 ——— <tr>
19 ——— <td>{{ form.descricao.errors }}</td>
20 ——— </tr>
21 ——— <tr>
22 ——— <td>{{ form.descricao }}</td>
23 ——— </tr>
24 ——— <tr>
25 ——— <td>{{ form.arquivo.label_tag }}</td>
26 ——— </tr>
27 ——— <tr>
28 ——— <td>{{ form.arquivo.errors }}</td>
29 ——— </tr>
30 ——— <tr>
31 ——— <td>{{ form.arquivo }}</td>
32 ——— </tr>

```

Figura 40 Template adiciona.html parte 1

```

33 -----<tr>
34 -----<td>{{ form.disciplina_id.label_tag }}</td>
35 -----</tr>
36 -----<tr>
37 -----<td>{{ form.disciplina_id.errors }}</td>
38 -----</tr>
39 -----<tr>
40 -----<td>
41 -----<select name="disciplina_id" id="id_disciplina_id">
42 -----<option></option>
43 -----{% for d in disciplina %}
44 -----<option value="{{ d.id }}" >{{ d.nome }}</option>
45 -----{% endfor %}
46 -----</select>
47 -----</td>
48 -----</tr>
49 -----<tr>
50 -----<td>&nbsp;</td>
51 -----</tr>
52 -----</table>
53 -----<button type="submit">Adicionar</button>
54 -----</form>
55 -----<p><a href="..">Voltar</a></p>
56 -----{% endblock %}

```

Figura 41 Template adiciona.html parte 2

A única diferença deste Template em relação ao anterior é que carrega todas as disciplinas diretamente na tela do formulário em uma caixa na qual o professor escolhe a disciplina e também seleciona um arquivo como material para upload. Veja no exemplo a seguir como foi apresentada a tela de cadastro.

Portal EDU

Nome

Descricao

Arquivo

Escolher arquivo Nenhum a...cionado

Disciplina id

Métodos Numéricos

Voltar

Figura 42 Tela de cadastro do material

```

1 | from django.shortcuts import render_to_response
2 | from models import Material
3 | from django.template import RequestContext
4 | from forms import FormMaterial
5 | from disciplina.models import Disciplina
6 | import datetime
7 |
8 | DIR = 'material/'
9 |
10 | def material(request):
11 |     — lista_materiais = Material.objects.all()
12 |     — return render_to_response(DIR + 'material.html', {'lista_materiais' : lista_materiais})
13 |
14 | def adiciona(request):
15 |
16 |     — # lista todas disciplinas do professor
17 |     — id_professor = request.session['professor'].id
18 |     — disciplina = Disciplina.objects.filter(professor_id=id_professor)
19 |
20 |     — if request.method == 'POST':
21 |         — form = FormMaterial(request.POST, request.FILES)
22 |         — if form.is_valid():
23 |             — form.enviado = True
24 |             — material = form.save()
25 |             — return render_to_response(DIR + 'salvo.html', {})
26 |
27 |     — else:
28 |         — form = FormMaterial()
29 |
30 |     — VARS = {
31 |         — 'form':form,
32 |     — }
33 |
34 |     — return render_to_response(DIR + "adiciona.html", ,
35 |     — {'form': form, 'disciplina': disciplina}, ,
36 |     — context_instance=RequestContext(request))

```

Figura 43 View da app material

A Figura 43 acima se assemelha à do professorEquipamento. O autor procedeu consulta à classe Disciplina para filtrar as disciplinas referentes ao professor e enviá-las para o Template. Na linha 17 atribui o valor do id do professor da sessão para a variável auxiliar logo abaixo na linha 18, onde é realizado o filtro que retorna o objeto com todas as disciplinas referentes ao id passado como argumento, que é o do professor logado no sistema. Quando carregar a tela pela primeira vez ele irá carregar as últimas linhas 34, 35 e 36, que consistem apenas num comando, o qual se encarrega de enviar os dados. O segundo parâmetro consiste nos dados colocados entre {}, que serão enviados para a página adicional.html.

2.8 ADMIN

“Um dos principais diferenciais do Django em relação a outros frameworks para desenvolvimento web é a sua interface automática de administração, que oferece um mecanismo extremamente poderoso para gerenciar os dados de sua aplicação sem a necessidade de ficar desenvolvendo por conta própria interfaces para cadastro de determinados tipos de objetos.” ¹⁶

Para utilizar o administrativo do Django é necessário fazer algumas configurações no settings.py, adicionando as quatro apps selecionadas, que estão dispostas na Figura 44 abaixo dentro das opções da INSTALLED_APPS.

```
78 INSTALLED_APPS = (  
79     ....'django.contrib.auth',  
80     ....'django.contrib.contenttypes',  
81     ....'django.contrib.sessions',  
82     ....'django.contrib.admin',  
83     ....'django.contrib.sites',  
84     ....'django.contrib.messages',  
85     ....'django.contrib.staticfiles',  
86     ....'home',  
87     ....'curso',  
88     ....'turma',  
89     ....'aluno',  
90     ....'professor',  
91     ....'disciplina',  
92     ....'equipamento',  
93     ....'material',  
94     ....'disciplinaAluno',  
95     ....'professorEquipamento',  
96 )
```

Figura 44 Configurações para o administrativo do Django no arquivo settings.py

¹⁶ SANTANA, Osvaldo; GALESI, Thiago. PYTHON e DJANGO – Desenvolvimento ágil de aplicações web. Novatec Editora Ltda. São Paulo, 2010, p. 197.

Após habilitar as apps, é preciso sincronizar o banco de dados a fim de que as tabelas usadas pelo administrativo sejam criadas. As mais importantes tabelas do Django são: as dos usuários e a dos grupos de usuários. Utilize o comando para sincronizar o banco `python manage.py syncdb`. Para terminar as configurações do administrativo é necessário configurar o arquivo de `urls.py` do Django, permitindo o acesso à URL `/admin/`, apontando para a interface de administração do Framework. Ao arquivo `urls.py` é preciso adicionar as três linhas selecionadas abaixo.

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

admin.autodiscover()

urlpatterns = patterns('',
    (r'^$', 'home.views.index'),
    (r'^aluno/$', 'aluno.views.login'),
    (r'^aluno/logout/$', 'aluno.views.logout'),
    (r'^aluno/disciplina/$', 'disciplinaAluno.views.disciplinasAluno'),
    (r'^professor/$', 'professor.views.login'),
    (r'^professor/logout/$', 'professor.views.logout'),
    (r'^professor/material/$', 'material.views.material'),
    (r'^professor/material/adiciona/$', 'material.views.adiciona'),
    (r'^professor/equipamento/$', 'equipamento.views.equipamento'),
    (r'^professor/disciplina/$', 'disciplina.views.disciplinasProfessor'),
    (r'^professor/disciplina/(?P<id_disciplina>\d+)/$', 'disciplinaAluno.views.disciplinasAlunos'),
    (r'^professor/nota/aluno/(?P<id_disciplinaAluno>\d+)/$', 'disciplinaAluno.views.alterarNotaAluno'),
    (r'^professor/equipamento/(?P<id_equipamento>\d+)/$', 'professorEquipamento.views.reservarEquipamento'),
    (r'^admin/', include(admin.site.urls)),
)
```

Figura 45 Configurando a URL `/admin/` no arquivo `urls.py`

Após as configurações, o administrativo estará funcionando, porém, não há razão para produzir o administrativo sem os correspondentes Models gerados pelo desenvolvedor. Para cada app tem-se um Model, o qual requer um arquivo `admin.py`. Veja como ficaram os arquivos `admin.py` das apps.

```
1 from aluno.models import Aluno
2 from django.contrib import admin
3
4 admin.site.register(Aluno)
```

Figura 46 Arquivo `admin.py` da app Aluno

```
1 from turma.models import Turma
2 from django.contrib import admin
3
4 admin.site.register(Turma)
```

Figura 47 Arquivo `admin.py` da app Turma

```

1 from professorEquipamento.models import ProfessorEquipamento
2 from django.contrib import admin
3
4 admin.site.register(ProfessorEquipamento)

```

Figura 48 Arquivo admin.py da app ProfessorEquipamento

```

1 from professor.models import Professor
2 from django.contrib import admin
3
4 admin.site.register(Professor)

```

Figura 49 Arquivo admin.py da app Professor

```

1 from material.models import Material
2 from django.contrib import admin
3
4 admin.site.register(Material)

```

Figura 50 Arquivo admin.py da app Material

```

1 from equipamento.models import Equipamento
2 from django.contrib import admin
3
4 admin.site.register(Equipamento)

```

Figura 51 Arquivo admin.py da app Equipamento

```

1 from disciplinaAluno.models import DisciplinaAluno
2 from django.contrib import admin
3
4 admin.site.register(DisciplinaAluno)

```

Figura 52 Arquivo admin.py da app DisciplinaAluno

```

1 from disciplina.models import Disciplina
2 from django.contrib import admin
3
4 admin.site.register(Disciplina)

```

Figura 53 Arquivo admin.py da app Disciplina


```
1 from curso.models import Curso
2 from django.contrib import admin
3
4 admin.site.register(Curso)
```

Figura 54 Arquivo admin.py da app Curso

É bastante simples criar o administrativo do Django. Para acessá-lo basta abrir o navegador e digitar o endereço <http://127.0.0.1:8000/>. A figura abaixo ilustra a referida operação.

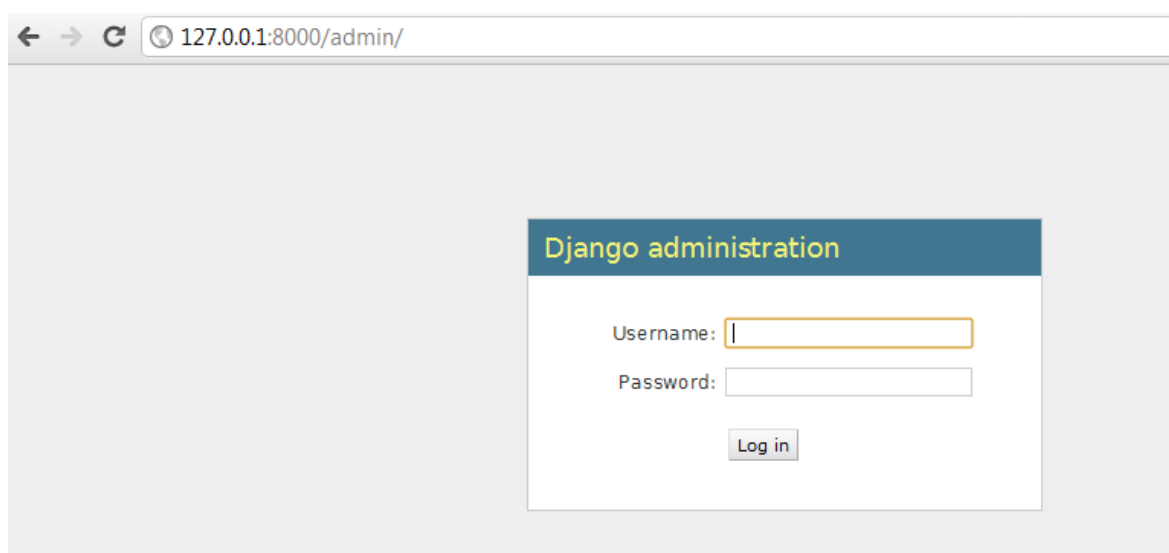


Figura 55 Tela do login do admin do Django

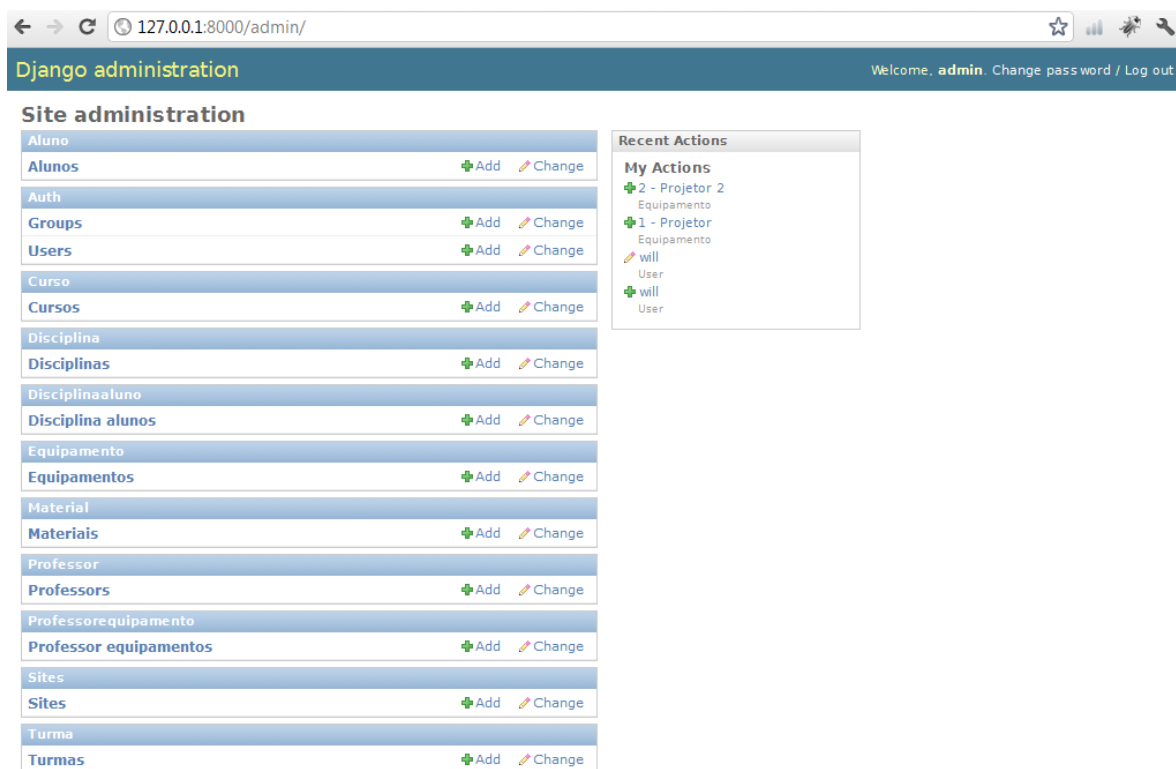


Figura 56 Tela principal do administrativo

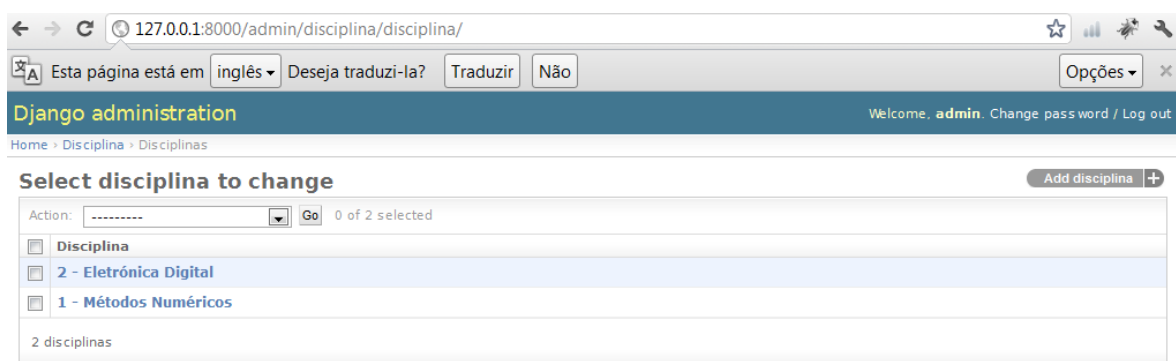


Figura 57 Tela das disciplinas cadastradas

The screenshot shows a web browser window with the URL `127.0.0.1:8000/admin/disciplina/disciplina/add/`. The page title is "Django administration" and the user is logged in as "admin". The breadcrumb trail is "Home > Disciplina > Disciplinas > Add disciplina".

The form "Add disciplina" contains the following fields:

- Nome:** A text input field.
- Professor id:** A dropdown menu with a plus icon to add a new professor.
- Curso id:** A dropdown menu with a plus icon to add a new course. The dropdown is open, showing a list of courses:
 - 1 - Ciência da Computação
 - 2 - Farmácia
 - 3 - Direito
 - 4 - Medicina

At the bottom of the form are three buttons: "Save and continue editing", "Save and add another", and "Save".

Figura 58 Tela para cadastrar uma nova disciplina

The screenshot shows a web browser window with the URL `127.0.0.1:8000/admin/professorEquipamento/professorequipamento/add/`. The page title is "Django administration" and the user is logged in as "admin". The breadcrumb trail is "Home > ProfessorEquipamento > Professor equipamentos > Add professor equipamento".

The form "Add professor equipamento" contains the following fields:

- Professor id:** A dropdown menu with a plus icon to add a new professor.
- Equipamento id:** A dropdown menu with a plus icon to add a new equipment.
- Data inicial:** A date and time selection field. The date is set to "Today" and the time is set to "Now".
- Data final:** A date and time selection field. The date is set to "Today" and the time is set to "Now".
- Sala:** A text input field.

A calendar widget is displayed, showing the month of November 2011. The calendar has a grid with days of the week (S, M, T, W, T, F, S) and dates (1 through 30). The date "1" is highlighted in yellow. Below the calendar are links for "Yesterday", "Today", and "Tomorrow", and a "Cancel" button.

At the bottom of the form are three buttons: "Save and continue editing", "Save and add another", and "Save".

Figura 59 Tela para agendamento de equipamento

CONCLUSÃO

Em princípio, o Python não foi programado para uso na web, mas o Framework Django, a exemplo dos similares, possibilitou-lhe criar aplicações na internet.

Outros aspectos que favorecem o uso do Framework são os sistemas administrativos muito práticos, que possibilitam a geração de telas completas, claras e objetivas, e a questão da segurança do componente URL Dispatcher, que apresenta as urls especificadas pelo programador, dificultando que os intrusos acessem os dados do site. Em função dos seus benefícios, os Frameworks, entre os quais o do Django, estão ampliando crescentemente a participação no mercado de trabalho.

Uma vez aprendido, o Framework mostra-se rápido, dinâmico, eficiente e gera grande economia de tempo, qualidades que valorizam o Django e os demais Frameworks. Atualmente, muitos deles são adotados por empresas de desenvolvimento de software, movimento que tende a crescer. Nestes termos, sem sombra de dúvida, pode-se sugerir o seu uso para os programadores em geral.

Com efeito, há a pretensão de continuar utilizando o Django tanto nos presentes como nos futuros projetos pessoais. Vale destacar que maiores disponibilidades de espaço e tempo seriam necessárias para, neste trabalho, explorar adequadamente os recursos oferecidos pelo Framework Django, dentre os quais podem ser mencionados os de internacionalização, paginação, comentários etc.

REFERÊNCIAS BIBLIOGRÁFICAS

SANTANA, Osvaldo; GALESI, Thiago. **PYTHON e DJANGO – Desenvolvimento ágil de aplicações web**. Novatec Editora Ltda. São Paulo, 2010.

CODE IGNITER BRASIL. MVC (Model – View – Controller). Disponível em: <<http://codeigniterbrasil.com/passos-iniciais/mvc-model-view-controller/>>. Acesso em: 07 set. 2011.

DJANGO. Quick install guide. 2011. Disponível em: <<https://docs.djangoproject.com/en/1.3/intro/install/>>. Acesso em: 03 ago. 2011.

DJANGO. The Django admin site. 2011. Disponível em: <<https://docs.djangoproject.com/en/1.3/ref/contrib/admin/>>. Acesso em: 03 set. 2011.

DJANGO. File Uploads. 2011. Disponível em: <<https://docs.djangoproject.com/en/1.3/topics/http/file-uploads/>>. Acesso em: 03 set. 2011.

DJANGO BRASIL. Models. 2011. Disponível em: <<http://docs.djangoproject.org/topics/db/models.html>>. Acesso em: 03 set. 2011.

DJANGO BRASIL. Trabalhando com formulários. 2011. Disponível em: <<http://docs.djangoproject.org/topics/forms/index.html#topics-forms-index>>. Acesso em: 03 set. 2011.

DJANGO BRASIL. O framework Django. 2011. Disponível em: <<http://www.djangoproject.org/>>. Acesso em: 03 set. 2011.

DJANGO. Django-admin.py and manage.py. 2011. Disponível em: <<https://docs.djangoproject.com/en/1.3/ref/Django-admin/>>. Acesso em: 06 set. 2011.

DJANGO. Django settings. 2011. Disponível em: <<https://docs.djangoproject.com/en/1.3/topics/settings/>>. Acesso em: 06 set. 2011.

DJANGO. Django's cache framework. 2011. Disponível em: <<https://docs.djangoproject.com/en/1.3/topics/cache/>>. Acesso em: 07 set. 2011.

DJANGO. Models. 2011. Disponível em: <<https://docs.djangoproject.com/en/1.3/topics/db/models/>>. Acesso em: 12 set. 2011.

DJANGO. Model field reference. 2011. Disponível em: <<https://docs.djangoproject.com/en/1.3/ref/models/fields/>>. Acesso em: 12 set. 2011.

DJANGO. Model Meta options. 2011. Disponível em: <<https://docs.djangoproject.com/en/1.3/ref/models/options/>>. Acesso em: 12 set. 2011.

DJANGO. Working with forms. 2011. Disponível em: <<https://docs.djangoproject.com/en/1.3/topics/forms/>>. Acesso em: 14 set. 2011.

DJANGO. URL dispatcher. 2011. Disponível em:
<<https://docs.djangoproject.com/en/1.3/topics/http/urls/>>. Acesso em: 25 out. 2011.

DJANGO BRASIL. Internacionalização. 2011. Disponível em:
<<http://docs.djangobrasil.org/topics/i18n.html#topics-i18n>>. Acesso em: 25 out. 2011.

OFICINA DA NET. O que é Model-view-controller (MVC). 2011. Disponível em:
<http://www.oficinadanet.com.br/artigo/desenvolvimento/o_que_e_model-view-controller_mvc/>. Acesso em: 04 set. 2011.

OFICINA DA NET. MVC – O padrão de arquitetura de software. 2011. Disponível em:
<http://www.oficinadanet.com.br/artigo/1687/mvc_-_o_padrao_de_arquitetura_de_software>. Acesso em: 09 set. 2011.

OFICINA DA NET. Framework, o que é e pra que server? 2011. Disponível em:
<http://www.oficinadanet.com.br/artigo/1294/framework_o_que_e_e_para_que_serve>. Acesso em: 22 out. 2011.

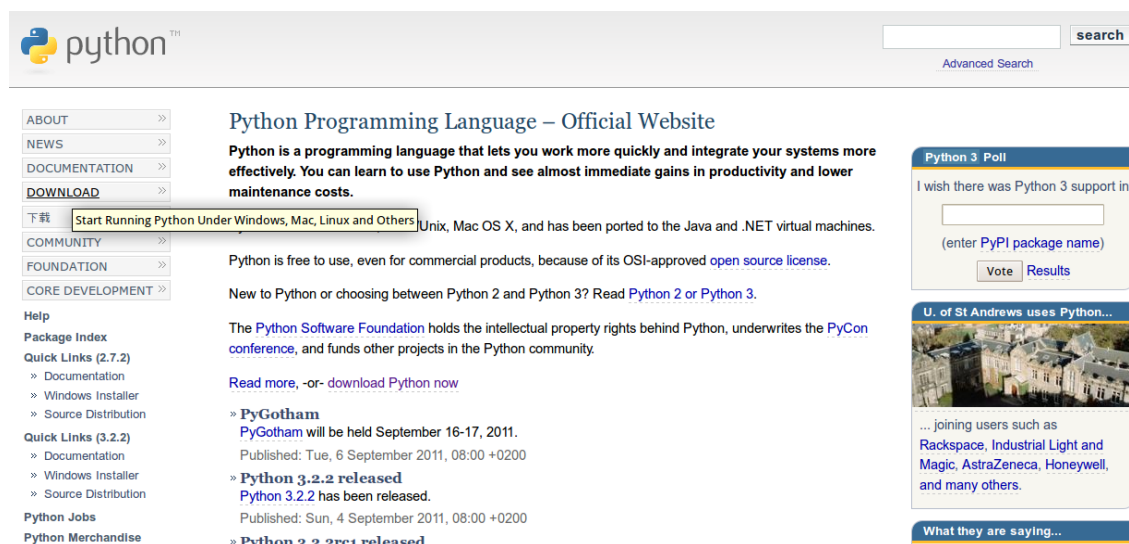
PENSO TI. O que é MVC. 2011. Disponível em:
<<http://www.pensoti.com.br/2011/desenvolvimento/o-que-e-mvc/>>. Acesso em: 10 ago. 2011.

UFCG TI. O que é um framework. Disponível em:
<<http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/oque.htm>>. Acesso em: 05 ago. 2011.

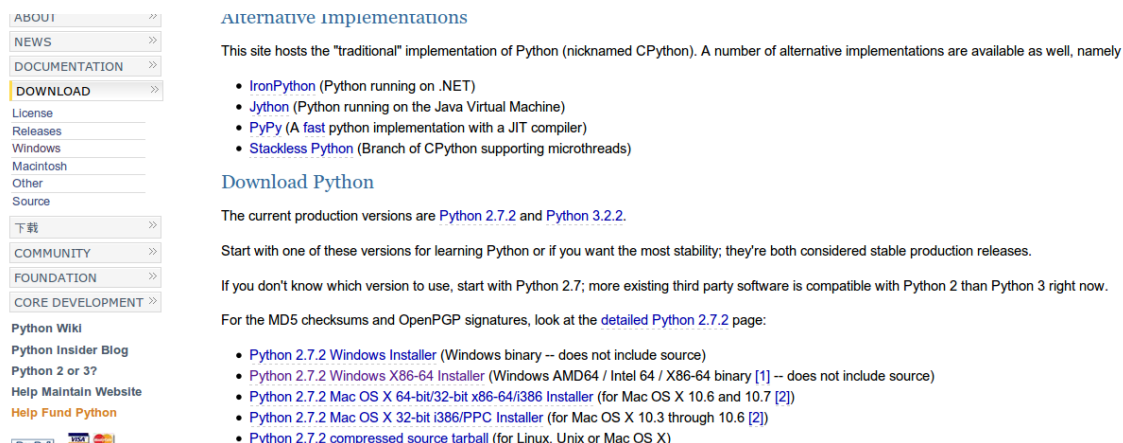
WIKIPEDIA. Framework. 2011. Disponível em:
<<http://pt.wikipedia.org/wiki/Framework>>. Acesso em: 22 out. 2011.

APÊNDICE A – INSTALAÇÃO DO PYTHON E DO DJANGO

Caso o sistema seja Windows ou Mac Os X, acesse o site <http://python.org>, dirija-se ao menu e clique em download, como pode ser visto abaixo.

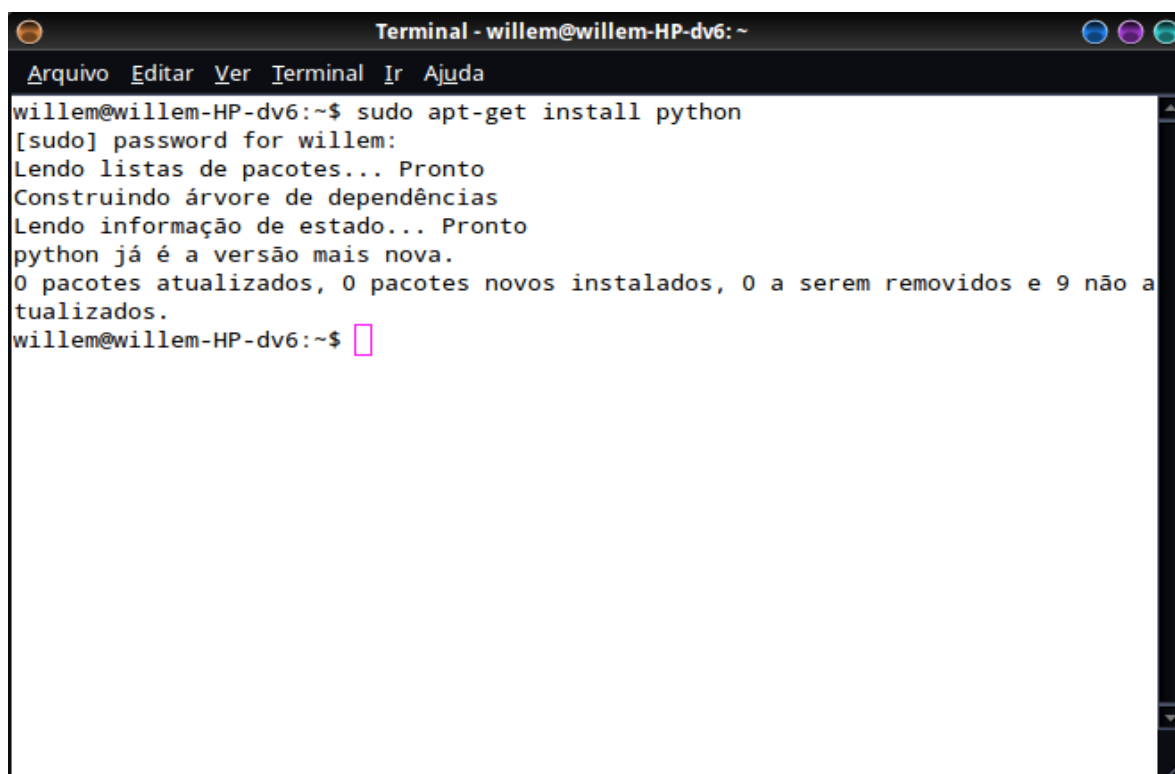


SITE OFICIAL DO PYTHON



PÁGINA DE DOWNLOADS DO SITE OFICIAL DO PYTHON

Para instalar ou atualizar em distribuições Linux, que são baseadas no Debian, abra o terminal e digite *sudo apt-get install python*, conforme a figura abaixo.



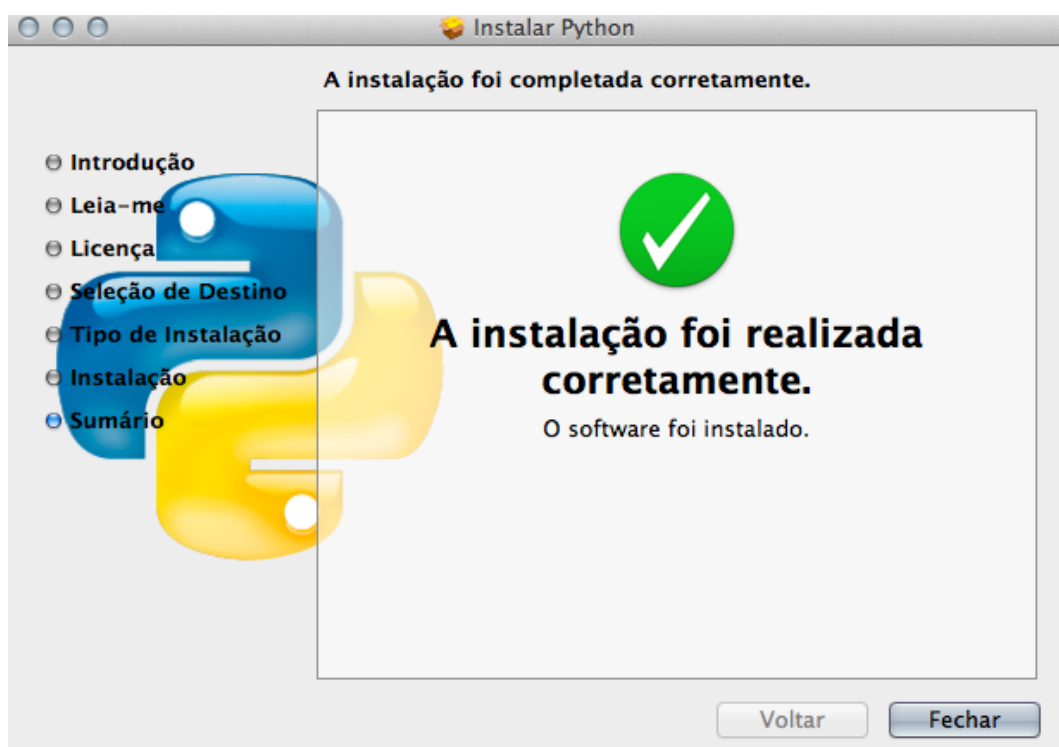
```
Terminal - willem@willem-HP-dv6: ~
Arquivo  Editar  Ver  Terminal  Ir  Ajuda
willem@willem-HP-dv6:~$ sudo apt-get install python
[sudo] password for willem:
Lendo listas de pacotes... Pronto
Construindo árvore de dependências
Lendo informação de estado... Pronto
python já é a versão mais nova.
0 pacotes atualizados, 0 pacotes novos instalados, 0 a serem removidos e 9 não a
tualizados.
willem@willem-HP-dv6:~$
```

INSTALAÇÃO DO PYTHON NO LINUX

Na instalação no Mac Os X, basta baixar o arquivo, executar, avançar as etapas seguintes e concordar com a licença, conforme as figuras abaixo.

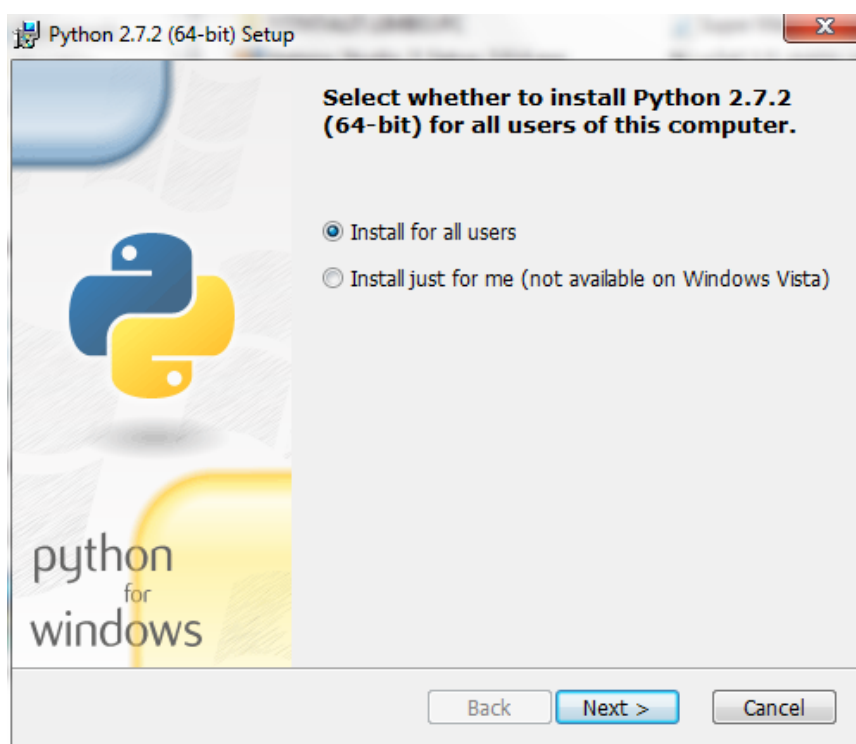


INSTALAÇÃO DO PYTHON NO MAC PARTE 1

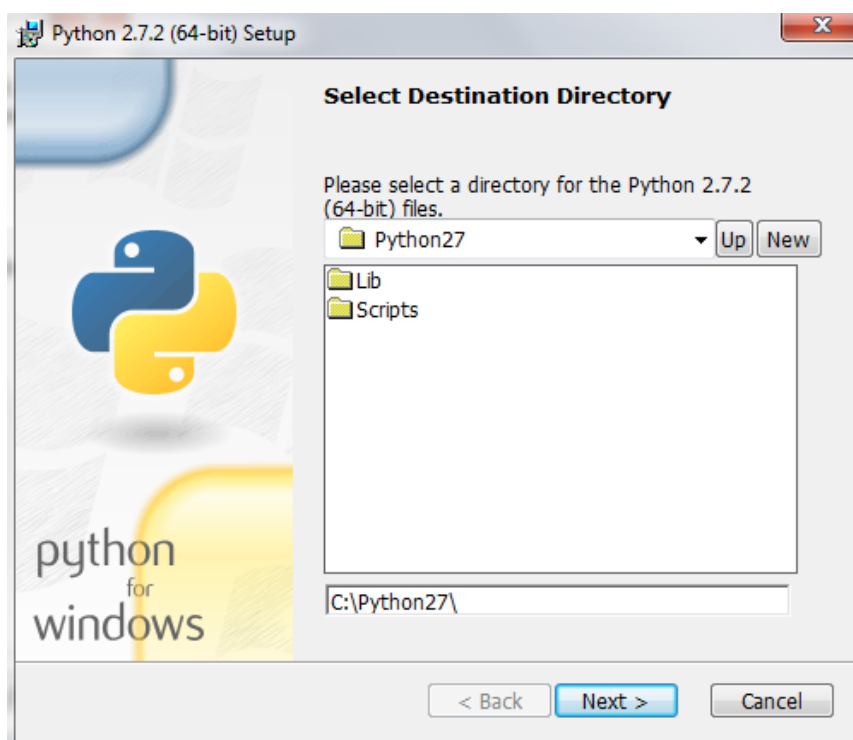


INSTALAÇÃO DO PYTHON NO MAC PARTE 2

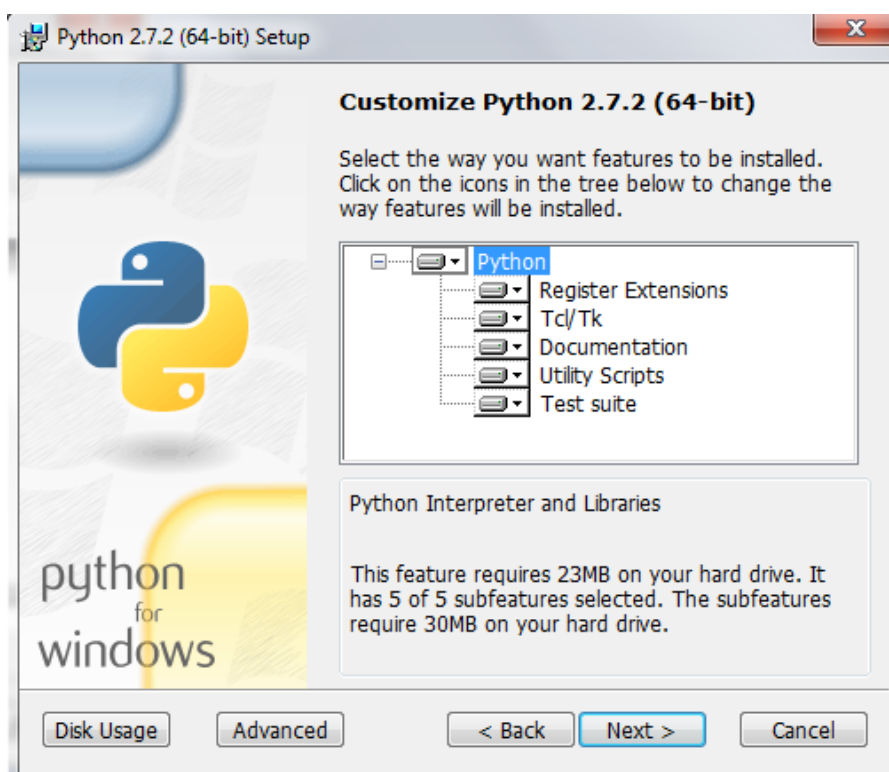
A instalação no Windows é muito semelhante à no Mac Os X, basta baixar o arquivo, executar e avançar as etapas, segundo o previsto nas figuras.



INSTALAÇÃO NO WINDOWS PARTE 1



INSTALAÇÃO NO WINDOWS PARTE 2



INSTALAÇÃO NO WINDOWS PARTE 3



INSTALAÇÃO NO WINDOWS PARTE 4

Para a instalação do Framework Django é preciso acessar o site www.djangoproject.com e baixar o arquivo, como na figura abaixo.

django

Home Download Documentation Weblog Community Code

The Web framework for perfectionists with deadlines.
Django makes it easier to build better Web apps more quickly and with less code.

Meet Django

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.

Developed by a fast-moving online-news operation, Django was designed to handle two challenges: the intensive deadlines of a newsroom and the stringent requirements of the experienced Web developers who wrote it. It lets you build high-performing, elegant Web applications quickly.

Django focuses on automating as much as possible and adhering to the [DRY principle](#).

Dive in by [reading the overview](#) →

When you're ready to code, read the [installation guide](#) and [tutorial](#).

The Django framework

Object-relational mapper

Define your *data models* entirely in Python. You get a rich, *dynamic database-access API* for free — but you can still write SQL if needed.

Download

Latest release: **1.3.1**

Open source, [BSD license](#)

Documentation

[Installation guide](#)

[Tutorial](#)

[Full index...](#)

Sites that use Django

lawrence.com
An internationally renowned local-entertainment site with events, stories, bands, drink specials and more.

washingtonpost.com
The Washington Post's growing selection of innovative Web database applications.

EveryBlock
A news feed for your block.

Weblog

Django 1.2.7 released
by James Bennett on Sep. 10, 2011
Django 1.2.7 is being released today to correct an issue with yesterday's 1.2.6 package.
[Read more](#)

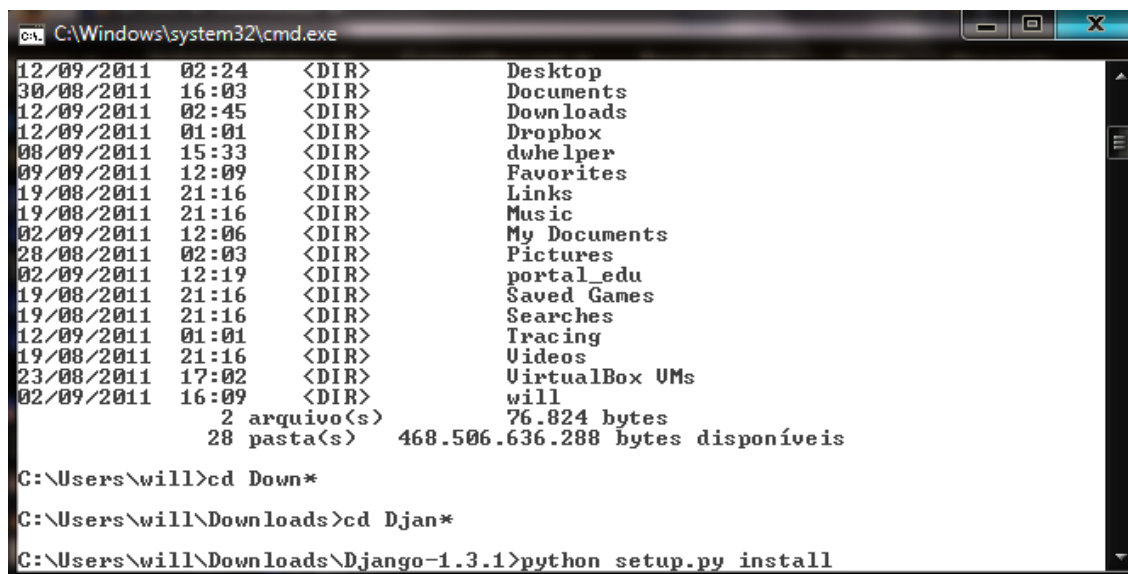
Security releases issued
by James Bennett on Sep. 9, 2011
Today the Django project is issuing releases to mitigate several security issues reported to us.
[Read more](#)

PyCon AU 2011 sprints
by Russell Keith-Magee on Aug. 19, 2011
Join us for a sprint as part of PyCon AU 2011!
[Read more](#)

DjangoCon 2011 is almost here
by Russell Keith-Magee and Steve Holden on Aug. 15, 2011

PÁGINA OFICIAL DO DJANGO

A instalação do Django é quase idêntica em todos os sistemas operacionais, bastando descompactar o arquivo, abrir o terminal ou prompt de comando e navegar pelo terminal até atingir o diretório descompactado. Se estiver utilizando Linux ou Mac, basta digitar o comando *sudo python setup.py install*, no Windows sendo apenas necessário tirar o *sudo*, restando o comando *python setup.py install*, segundo as figuras abaixo.



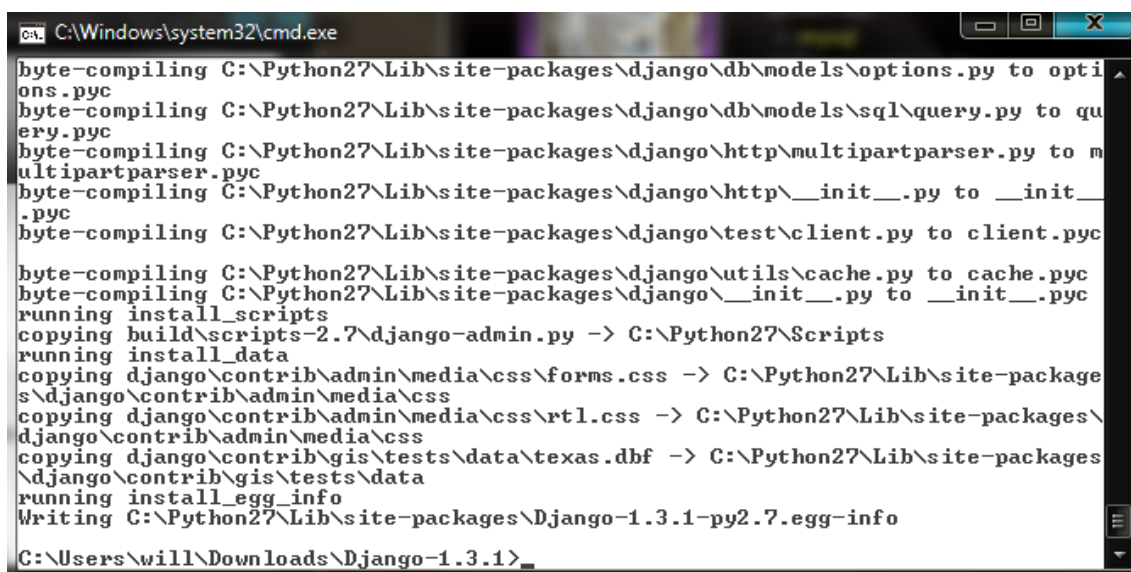
```

C:\Windows\system32\cmd.exe
12/09/2011 02:24 <DIR> Desktop
30/08/2011 16:03 <DIR> Documents
12/09/2011 02:45 <DIR> Downloads
12/09/2011 01:01 <DIR> Dropbox
08/09/2011 15:33 <DIR> dwhelper
09/09/2011 12:09 <DIR> Favorites
19/08/2011 21:16 <DIR> Links
19/08/2011 21:16 <DIR> Music
02/09/2011 12:06 <DIR> My Documents
28/08/2011 02:03 <DIR> Pictures
02/09/2011 12:19 <DIR> portal_edu
19/08/2011 21:16 <DIR> Saved Games
19/08/2011 21:16 <DIR> Searches
12/09/2011 01:01 <DIR> Tracing
19/08/2011 21:16 <DIR> Videos
23/08/2011 17:02 <DIR> VirtualBox VMs
02/09/2011 16:09 <DIR> will
                2 arquivo(s)      76.824 bytes
                28 pasta(s)    468.506.636.288 bytes disponíveis

C:\Users\will>cd Down*
C:\Users\will\Downloads>cd Djan*
C:\Users\will\Downloads\Django-1.3.1>python setup.py install

```

INSTALAÇÃO DO DJANGO NO WINDOWS



```

C:\Windows\system32\cmd.exe
byte-compiling C:\Python27\Lib\site-packages\django\db\models\options.py to options.pyc
byte-compiling C:\Python27\Lib\site-packages\django\db\models\sql\query.py to query.pyc
byte-compiling C:\Python27\Lib\site-packages\django\http\multipartparser.py to multipartparser.pyc
byte-compiling C:\Python27\Lib\site-packages\django\http\__init__.py to __init__.pyc
byte-compiling C:\Python27\Lib\site-packages\django\test\client.py to client.pyc
byte-compiling C:\Python27\Lib\site-packages\django\utils\cache.py to cache.pyc
byte-compiling C:\Python27\Lib\site-packages\django\__init__.py to __init__.pyc
running install_scripts
copying build\scripts-2.7\django-admin.py -> C:\Python27\Scripts
running install_data
copying django\contrib\admin\media\css\forms.css -> C:\Python27\Lib\site-packages\django\contrib\admin\media\css
copying django\contrib\admin\media\css\rtl.css -> C:\Python27\Lib\site-packages\django\contrib\admin\media\css
copying django\contrib\gis\tests\data\texas.dbf -> C:\Python27\Lib\site-packages\django\contrib\gis\tests\data
running install_egg_info
Writing C:\Python27\Lib\site-packages\Django-1.3.1-py2.7.egg-info
C:\Users\will\Downloads\Django-1.3.1>_

```

INSTALAÇÃO DO DJANGO NO WINDOWS TERMINADA

