# **DESENVOLVIMENTO WEB COM WEB2PY**

# https://upload.wikimedia.org/wikipedia/commons/thumb/c/c3/Python-logo-notext.svg/1024px-Python-logo-notext.svg.png

# Web2py - Introdução ao desenvolvimento ágil para web utilizando Python.

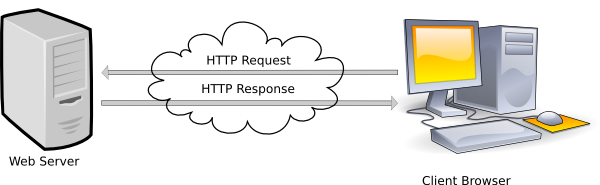
Neste livro, vamos aprender como desenvolver uma aplicação rápida para web, utilizando um framework construído em Python e persistência em banco de dados.

## Web Framework

Antes de falar sobre o framework propriamente dito, vamos entender como funciona uma aplicação básica web: Uma aplicação web é composta de diversas funções que são executadas em determinadas URL’s, o resultado é renderizado para o navegador do usuário. Dessa forma, os web frameworks servem como um facilitador de desenvolvimento, com ferramentas construídas para facilitar o seu desenvolvimento.

## Protocolo HTTP

Ao navegar na web, você deve ter percebi no seu navegador, mais precisamente na barra de endereços, o protocolo HTTP (Hypertext Transfer Protocol), que é base da web atualmente, sua função é realizar a comunicação entre cliente e servidor e para isso utiliza os serviços request (requisitar) e response (responder).



## Sobre o web2py

Web2py é um framework para aplicações web de código aberto escrito na linguagem de programação Python. O web2py permite que os desenvolvedores web programem conteúdo web dinâmico usando Python. Ele foi projetado para ajudar a reduzir tarefas tediosas de desenvolvimento web, como o desenvolvimento de formulários web do zero, apesar de um desenvolvedor web poder construir um formulário do zero se necessário.

O web2py foi originalmente projetado como uma ferramenta de ensino com ênfase na facilidade de uso e implementação. Entretanto, ele não possui quaisquer arquivos de configuração de nível de projeto. O projeto do web2py foi inspirado pelos frameworks Ruby on Rails e Django. Assim como esses frameworks, o web2py foca no desenvolvimento rápido, favorece a abordagem convenção sobre configuração e segue o padrão arquitetural modelo-visão-controlador (MVC).

## Modelo MVC

Model-view-controller (MVC) é um padrão de arquitetura de software que separa a representação da informação da interação do usuário com ele. O modelo (model) consiste nos dados da aplicação, regras de negócios, lógica e funções. Uma visão (view) pode ser qualquer saída de representação dos dados, como uma tabela ou um diagrama. É possível ter várias visões do mesmo dado, como um gráfico de barras para gerenciamento e uma visão tabular para contadores. O controlador (controller) faz a mediação da entrada, convertendo-a em comandos para o modelo ou visão. As ideias centrais por trás do MVC são a reusabilidade de código e separação de conceitos.

## MODELS

Nos models são guardadas informações de bancos de dados, conexão, tabelas e colunas. A partir disso, poderemos gravar e armazenar dados da aplicação e interagir com eles: visualizar, editar, deletar, etc.

Os models também são sempre os primeiros arquivos a serem lidos, em ordem alfabética, pelo web2py. Ou seja, se eu tenho um arquivo db.py e um model.py, as informações do arquivo db.py serão lidas e interpretadas antes de model.py. Portanto, se alguma informação presente em db.py depender de algo em model.py, ocorrerá um erro!

Além disso, tudo o que for definido em um model poderá ser acessado em todas as partes da aplicação.

## CONTROLLERS

Nos controllers são definidas as funções da aplicação que farão com que ela funcione da forma como designada. São nos **controllers** que poderemos fazer com que a aplicação interaja com o banco de dados e envie informações para o usuário. No web2py, cada função sem parâmetros no controller é mapeada para uma URL, ou seja, cada função define uma diferente página da sua aplicação.

Caso você passe parâmetros para uma função, ela deixa de ser mapeada como uma página e passa a ser uma função Python comum, que poderá ser chamada no código. Isso é especialmente feito em **models**, pois assim a função será acessível em todas as camadas de código.

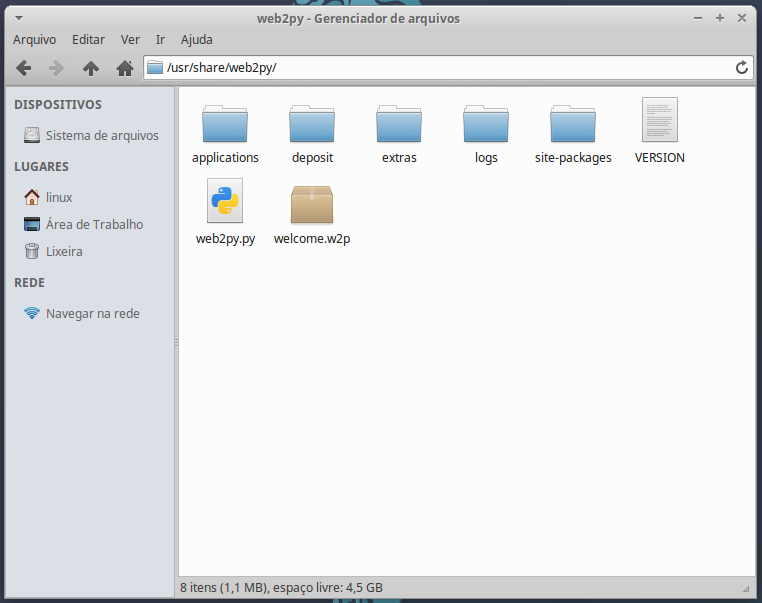
## VIEWS

As views definem o que a aplicação vai exibir para o usuário e a forma como isso se dará. No web2py, views são arquivos HTML que podem receber código Python. O código Python em um view deve estar dentro dos delimitadores {{ }} para ser executado.

Views podem herdar ou incluir outras views. Isto é, você pode fazer com que uma view contenha outra, exibindo ambas em conjunto.

## DESENVOLVENDO SUA PRIMEIRA APLICAÇÃO COM WEB2PY

O primeiro passo é organizar o seu ambiente de desenvolvimento. Ao criar uma nova aplicação na interface administrativa do web2py, teremos a respectiva pasta da aplicação dentro do nosso diretório do web2py. Para começar a desenvolver, precisamos entender como funciona as estruturas de pastas de nosso Framework



A maior parte do conteúdo desta pasta não fará grande diferença para você enquanto desenvolve, mas existem alguns que merecem um certo destaque:

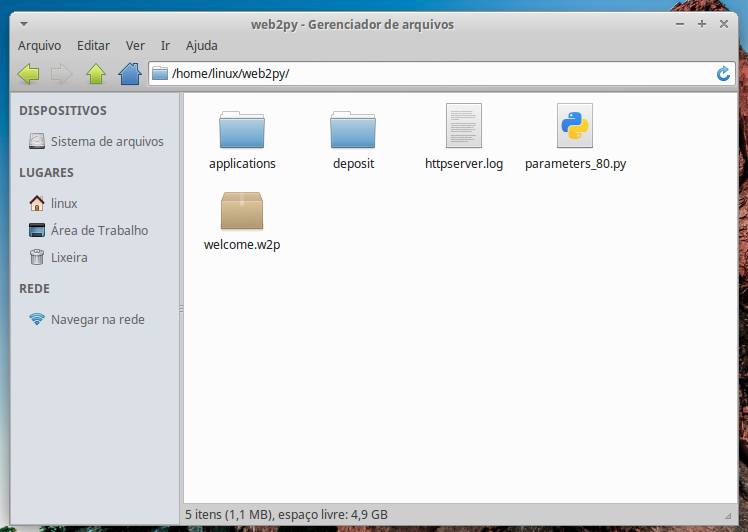
O diretório applications contém todas as aplicações que você criou no web2py. Cada aplicação é uma pasta que contém diversos outros conteúdos que analisaremos também adiante.

O diretório examples contém arquivos de exemplo que você pode usar como modelo para várias funcionalidades do web2py. Por exemplo: configurações de rota, configurações de logging, configurações do appengine, entre outros.

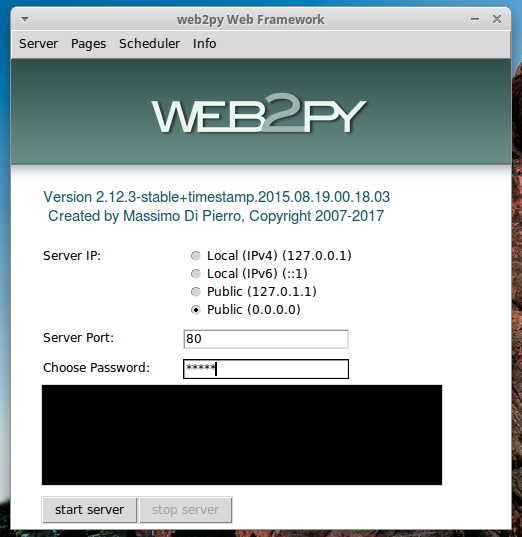
O diretório gluon contém os módulos e arquivos que literalmente constroem o framework.

O arquivo web2py.py, que é o arquivo que inicia a execução do framework.

Você usará a pasta criada dentro do seu diretório /home/, lá terá nosso diretório de trabalho onde desenvolveremos nossa aplicação.

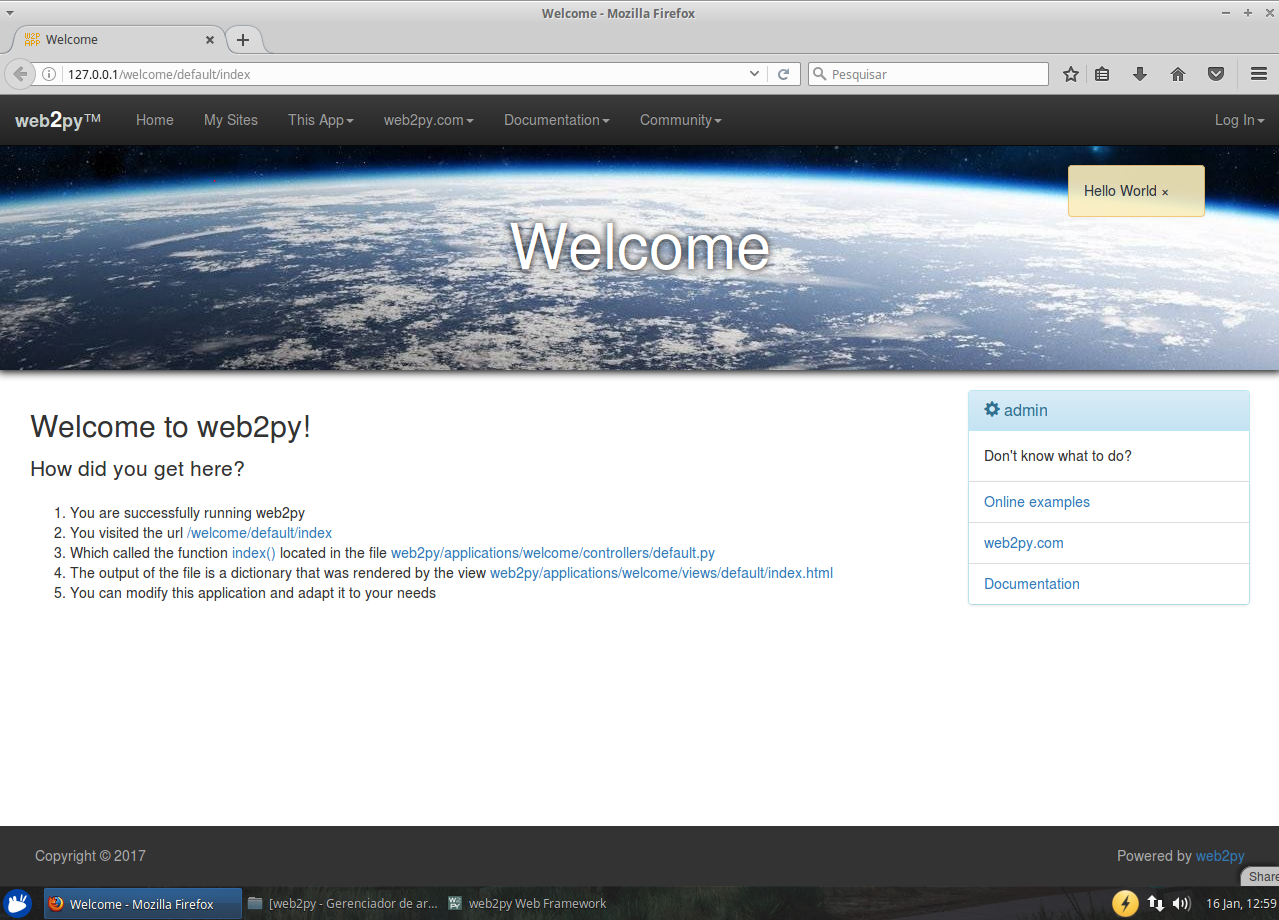


Essa estrutura é criada automaticamente quando executamos o web2py pela primeira vez.

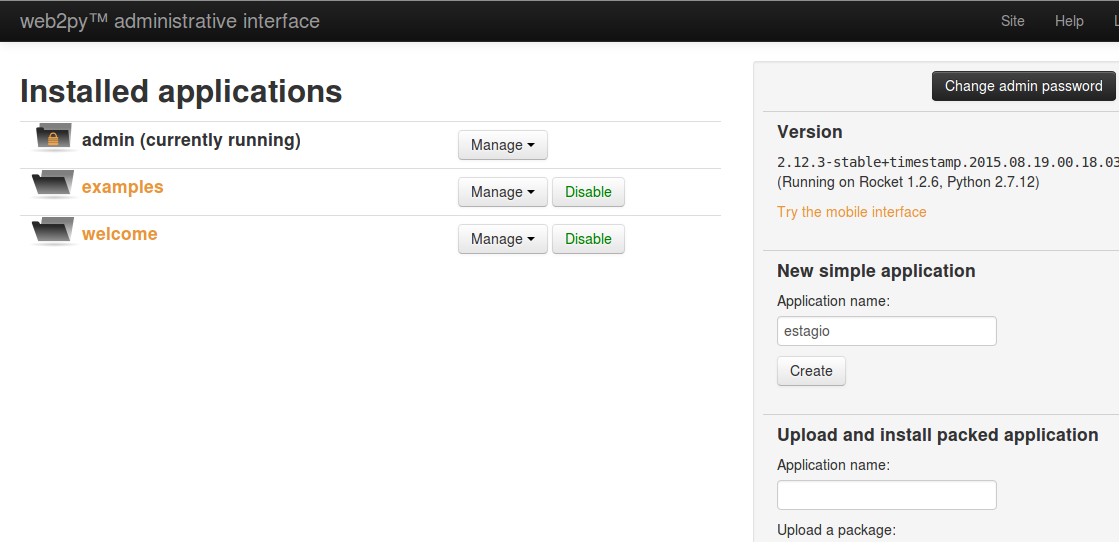


## Abrindo nosso Sistema Gerenciador de Estágio

No seu navegador, acesso o seu endereço local (o web2py já abre para você). Você terá acesso a tela de boas vindas



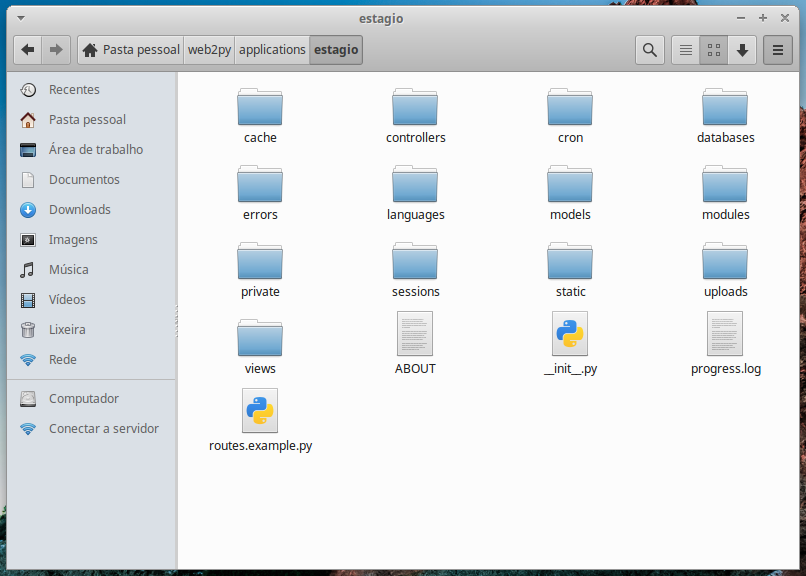
Acesse a interface administrativa e crie uma nova aplicação chamada estagio.



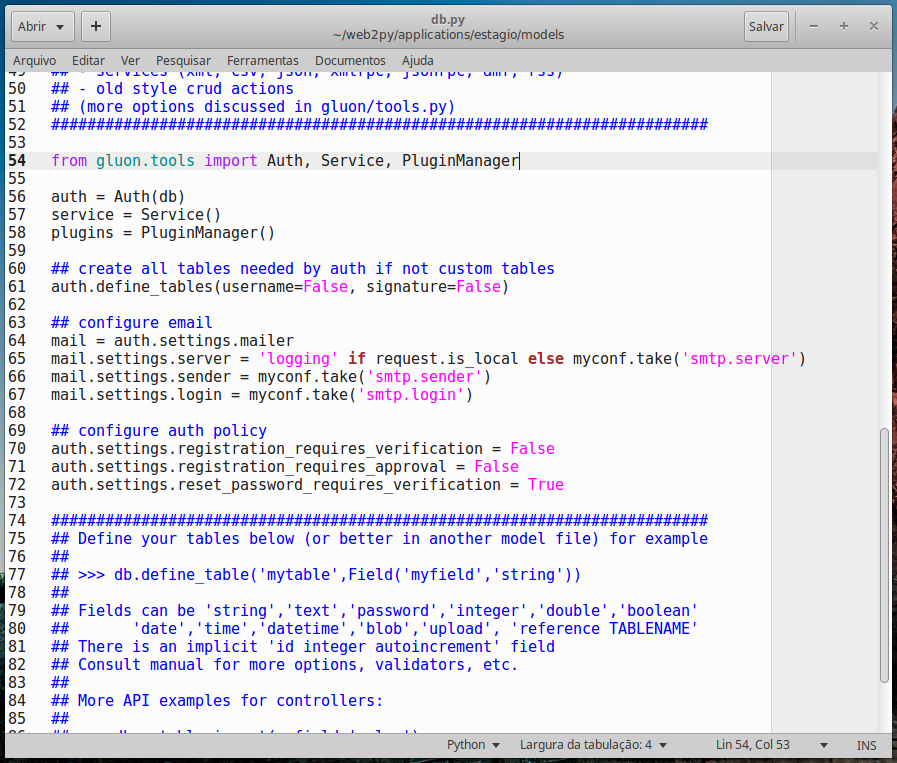
Seguindo o modelo MVC, devemos começar pelos **models**, assim teremos conteúdo para trabalhar nos **controllers** e para exibir ao usuário nas **views**. Agora, vamos pensar: o que um sistema gerenciador de estágios precisa ter?

Um sistema que gerencie estágio precisa armazenar informações da empresa, professor orientador e aluno. Preciso também saber a relação entre esses três: Qual aluno está alocado em qual empresa e qual orientador está responsável por qual aluno.

Vamos abrir o diretório da nossa aplicação. Você terá uma visão semelhante a essa:



O web2py já criou todos os diretórios necessários para o desenvolvimento de nossa aplicação. Aqui você encontra os arquivos que serão interpretados pelo framework. Começando pelo o arquivo **db.py** que fica dentro da pasta models da sua aplicação.



Agora vamos analisar esse arquivo.

from gluon.contrib.appconfig import AppConfig

myconf = AppConfig(reload=True)

Aqui estamos associando as configurações da aplicação a uma variável **myconf**.

if not request.env.web2py\_runtime\_gae:

    db = DAL(myconf.take('db.uripool\_size=myconf.take('db.pool\_sizeast=int), check\_reserved=['all'])

else:

    db = DAL('google:datastore+ndb')

    session.connect(request, response, db=db)

Continuando a analizar o arquivo **db.py**, agora que entendemos como ele trouxe as configurações, temos:

from gluon.tools import Auth

auth = Auth(db)

auth.define\_tables(username=False, signature=False)

O que temos é a importação da classe Auth. Essa classe é o que permite ao web2py fazer todo o controle de autenticação de usuários. Basicamente, o Auth irá criar todas as tabelas de usuário no banco de dados que passamos para ele (no nosso caso, aquele associado à variável db) sozinho a partir do comando auth.define\_tables() que usamos.

Além disso, ele irá fazer o controle de permissões, como veremos adiante, para saber se aquele usuário tem permissão para acessar determinada página, qual a categoria do usuário, se ele está realmente logado na página, etc.

Para fazer todo esse controle, o web2py cria, além de uma tabela de usuários, uma tabela de grupos, uma de permissões, etc. Mas a única que irá nos interessar no momento é a tabela de usuários. Os campos da tabela que nos interessam (e não são preenchidos automaticamente pelo web2py) são:

* First name: O primeiro nome do usuário.
* Last name: O sobrenome do usuário.
* E-mail: O e-mail do usuário.
* Password: A senha do usuário, que é criptografada automaticamente e armazenada dessa forma.

Outros campos que por ventura você veja nessa tabela, são para o web2py fazer controle de algumas das suas funcionalidades.

Começando com nossa primeira tabela, escreva ao final do arquivo db.py:

Aluno = db.define\_table('aluno',

Field('nome', 'string', label = 'Nome'),

Field('matricula', 'string', label = 'Matricula'),

Field('curso', 'string', label = 'Curso'),

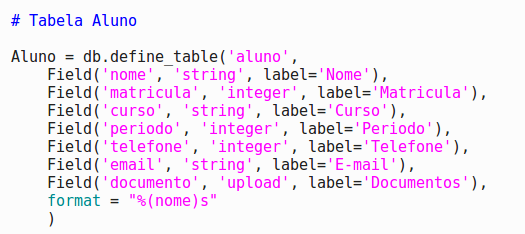
Field('periodo', 'integer', label = 'Periodo'),

Field('telefone', 'string', label = 'Telefone'),

Field('email', 'string', label = 'E-mail'),

Field('documento', 'upload', label = 'Documentos')

)



Vamos entender:

db.define\_table(nome, campos...)

É o método que usamos para criar novas tabelas. **db** é uma variável que definimos no início do arquivo **db.py** e ela é a nossa conexão com o banco de dados. Assim, ao chamar

db.define\_table(nome, campos...)

Estamos falando para criar uma tabela nome com os campos passados. No nosso caso, criamos uma tabela chamada **aluno** com os campos que mencionamos anteriormente. **FORMAT** é para que exiba o nome quando visualizarmos os resultados e não os **ID’s**

Além disso, associamos essa tabela à variável **Aluno**. Ou seja, sempre que chamarmos **Aluno**, estamos apenas chamando essa tabela que acabamos de criar.

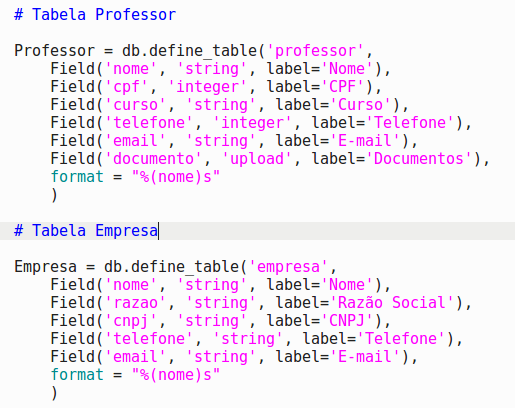
Cada campo da tabela é criado como um Field(). Um Field pode ter várias opções.

Nome do campo é o nome que esse campo terá na sua tabela. Os nomes devem ser definidos quase com as mesmas regras das variáveis, sem caracteres especiais, etc.

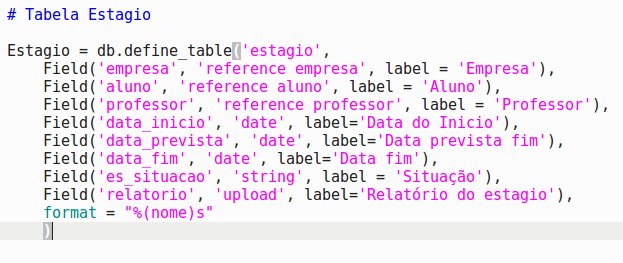
Tipo do campo, que é quase a mesma tipagem das variáveis de Python. Podem ser strings, integers, etc. Alguns tipos são especiais, como o tipo upload, que guarda um arquivo.

Label do campo, que é o nome visível do campo. Quando aprendermos a gerar formulários a partir dos campos da tabela, você verá que o título de cada campo do formulário baseia-se no nome do campo. No nosso caso, alguns campos como o email, tem nomes sem acento e sem ifen, o que prejudica a leitura desse nome. Por isso, passamos na label aquele nome que queremos que seja exibido no lugar do nome real do campo.

Ok, podemos riscar da lista também a tabela de aluno, ela está pronta! Próximo passo: criar nossa tabela professor e empresa.



Agora só falta criar a tabela resultante do estágio, responsável por criar a relação entre as tabelas.



Perceba que estamos utilizando um novo tipo de campo: “reference”. Esse tipo de campo diz que estamos referenciando uma informação de outra tabela em nossa tabela estagio, ou seja, armazenaremos informações que existem em outra tabela e criamos assim relações entre elas. Em banco de dados, chamamos de chave estrangeira.

Páginas no Web2py

Agora que temos nosso banco de dados, precisaremos criar páginas para que o usuário possa interagir com o sistema, cadastrar informações e coisas do tipo.

No web2py, cada página da nossa aplicação é uma função Python nos **controllers**. Para entendermos como isso funciona, primeiro temos que entender como ler uma URL do web2py. Por exemplo, quando estamos na index da nossa aplicação: <http://127.0.0.1/estagio/default/index>.

Vamos analisar parâmetro por parâmetro, do primeiro ao último:

* **estagio** é o nome da nossa aplicação;
* **default** é o nome do nosso controller;
* **index** é o nome da nossa página/função (o web2py também as chama de actions);

Basicamente, o que temos aqui é que esta página é uma função chamada index que está dentro do controller default que pertence à aplicação estagio. Ou seja, se você abrir a pasta controllers dentro da pasta da sua aplicação, verá que lá existe um controller chamado default.py e que, neste arquivo, existe uma função chamada index.

O controller default.py vem por padrão nas novas aplicações e traz funções padrão também, como a index, a user (que lida com páginas de autenticação, cadastro, perfil, etc de usuários), a download, entre outras.

Como você pode ver neste arquivo, as páginas são todas definidas como uma função Python e tem a seguinte estrutura:

def index():

return dict(message=T('Welcome to web2py!'))

A nome da função é o que irá definir o nome da página, ou seja, a função index() define uma página que será roteada para a URL /index.

## Criando páginas da nossa aplicação

Vamos pensar no que precisamos para gerenciar um estágio: Para existir um estágio, precisamos ter alunos, professores e uma empresa. Vamos começar criando a página para cadastrar alunos.

Vá ao final do arquivo *default.py* e adicione:

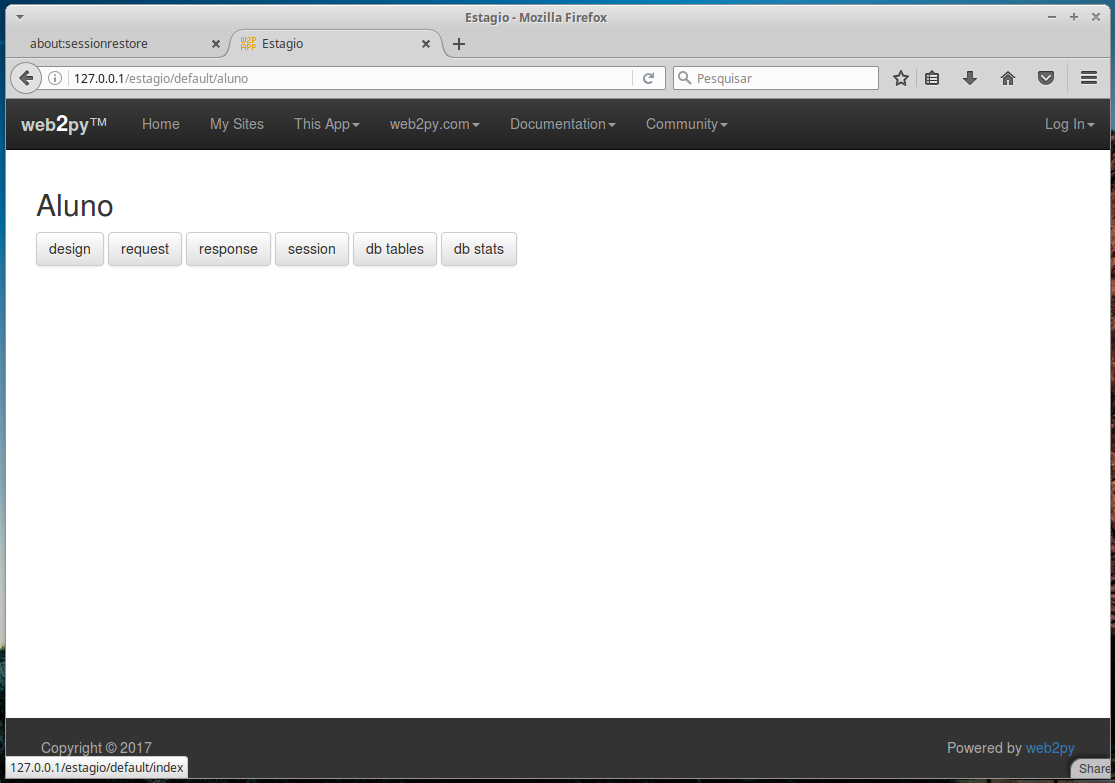
def aluno():

return dict()

## Views Genéricas

Nem sempre quando estamos trabalhando no desenvolvimento back-end de uma aplicação temos pronto desde o início todo o front-end do que queremos fazer e isso geralmente dificulta nossa vida. Por isso, o web2py traz por padrão um html gerado automaticamente quando estamos em ambiente de desenvolvimento, para que possamos visualizar aquilo que estamos fazendo e só por último nos preocuparmos com a parte “bonita” da aplicação.

O web2py utiliza o Bootstrap para gerar essas telas padrão. Por isso, quando você cria uma nova página, já é capaz de ver uma tela com informações básicas mesmo que você ainda não tenha criado todo o html.



Essas views genéricas que o web2py aparecem com os seguintes elementos:

* **Navbar** e **footer** padrão do web2py;
* **Título** da nova página, que é uma versão tratada do nome que você deu para sua função;
* **Menu de botões** com opções para visualizar informações sobre a página atual. Iremos entender melhor como interpretar estes dados que os botões trazem no futuro, no momento eles não são importantes.

Agora, sempre que você enviar uma informação para a sua página utilizando a partir do dicionário retornado pela função, essas informações devem ser apresentadas na página. Faça um teste escrevendo:

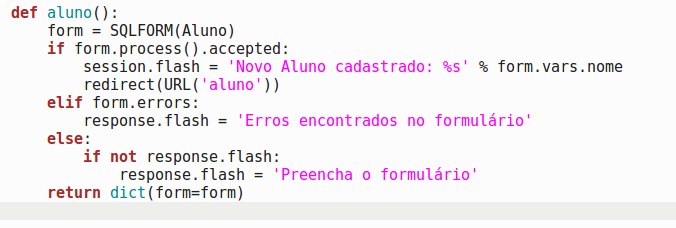
def aluno():

return dict(mensagem="Cadastre aluno!")

## Criando formulários

Tendo nossa página criada, precisamos fazer com que ela realmente funcione para algo. Agora nossa página de novos filmes tem que realmente adicionar novos filmes.

Vamos adicionar o seguinte código à nossa função aluno():



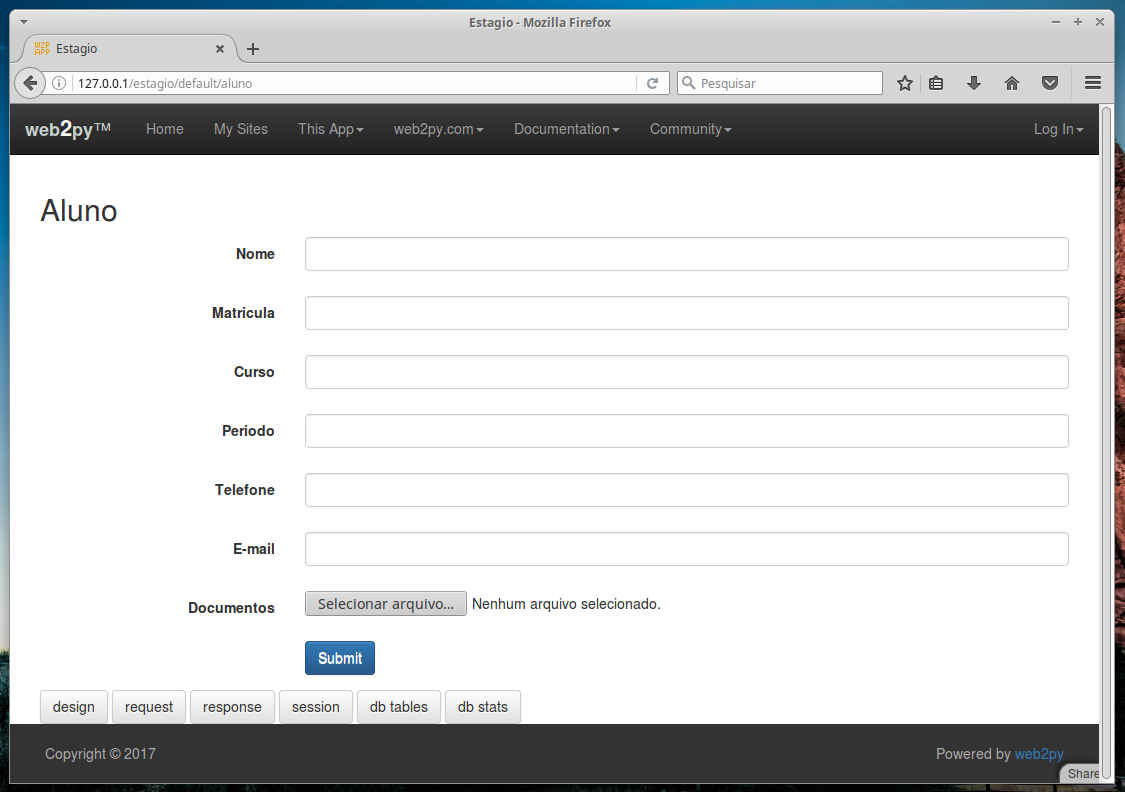
Nesse trecho de código temos muitas coisas novas, mesmo que sejam poucas, precisamos da atenção a tudo que está acontecendo nesse formulário.

Vamos analisar linha por linha:

form = SQLFORM(Filmes)

**SQLFORM** é mais um construtor do web2py. É quase como mágica: nós informamos para ele qual tabela que vamos usar para inserir registros no banco de dados e ele gera um formulário a partir dos campos dessa tabela. Ou seja, se na nossa tabela de alunos nós temos um campo chamado nome, agora temos um formulário com um campo nome também.

Lembre-se que Aluno é a variável que está guardando a tabela Aluno que criamos



Depois de criar nosso formulário, fizemos o seguinte:

if form.process().accepted:

...

elif form.errors:

...

else:

...

O que estamos fazendo é perguntar: o formulário foi processado sem errors? Se a resposta for sim, o código que está no nosso if será processado. Se ocorrer algum erro, o código do elif será processado. Se nada acontecer, ou seja, se não houver nenhuma submissão do formulário e nenhum erro, o código do else é processado.

Agora, o que temos dentro do if, do elif e do else?

if form.process().accepted:

session.flash = 'Novo aluno cadastrado: %s' % form.vars.nome

redirect(URL('aluno'))

O método process() do nosso formulário já verifica se tudo ocorreu sem errors e insere o registro no banco de dados, então não precisamos fazer nenhum tipo de inserção. Resta-nos apenas falar para o usuário que o filme foi cadastrado, para ele saber que ocorreu tudo como planejado.

É isso que fazemos com o session.flash. Ambos session.flash e response.flash recebem uma mensagem que deve ser exibida para o usuário na forma de notificação. A diferença entre eles é que o response.flash é exibido na mesma página em que foi chamado, já o session.flash só será exibido depois de uma redireção ou mudança de página.

Faça um teste e adicione um filme. Você verá que tivemos um problema, pois a mensagem do nosso session.flash não será exibida, apenas o response.flash do else. Isso acontece porque o session.flash depois da redireção é “convertido” em response.flash e, quando a página recarrega, o response.flash do else substitui o do if. Para evirtarmos isso, podemos adicionar essa linha ao else:

else:

if not response.flash:

response.flash = 'Preencha o formulário!'

No nosso session.flash do if informamos também um valor: form.vars.titulo. Isso é porque depois de processado o formulário, todos os valores recebidos pela submissão podem ser acessados no dicionário form.vars. Assim, cada campo do formulário pode ser acessado por form.vars.nome\_do\_campo e o valor submetido pode ser visto.

Após essas mensagens, chamamos a função redirect() que faz uma redireção para a URL informada. Para montarmos a URL da nossa página, utilizamos a função URL() do web2py. Essa função recebe os seguintes parâmetros:

URL(a=’application’, c=’controller’, f=’action’, args=[], vars={}, host=False, scheme=’http’)

Cada parâmetro especifica:

* a: a aplicação na qual estamos procurando a página;
* c: o controller em que esta página deve estar;
* f: o nome da página para qual queremos ir;
* args: argumentos da página;
* vars: variáveis da página;
* host: se queremos montar uma URL absoluta (com o domínio) ou parcial (sem o domínio). Por exemplo: http://meusite.com/index é uma URL absoluta, e /index é uma URL parcial.
* scheme: o protocolo. Por exemplo, http, https, etc.

Para aqueles que não estão acostumados com esse conceito, variáveis e argumentos são valores que podemos passar via URL. Por exemplo:

http://meusite.com/index/arg1/arg2?var1=2&var2=3

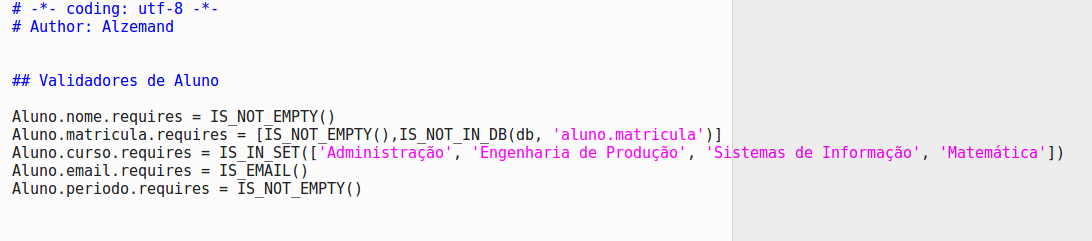
Os argumentos são aqueles separados do nome da página por / e são um valor apenas. Já as variáveis são aquelas separadas por ? e são um par chave-valor, onde o que vem antes do sinal = é a chave e o que vem depois do sinal = é o valor. Várias variáveis são separadas por &.

No momento, não temos interesse em passar esses valores pela URL. No futuro, veremos como resgatá-los. Por enquanto, vamos utilizar a função URL() apenas informando o nome da página para qual queremos ir, pois o nome da aplicação e do controller são, por padrão, os que nós já estamos utilizando.

Feito isso, terminamos nossa página! Tente adicionar um novo aluno.

## Validadores

Perceba que ao cadastrar um aluno, você pode deixar campos vazios se quiser, cadastrar o mesmo aluno várias vezes e causar um verdadeiro caos no seu banco de dados. Para resolver isso, utilizaremos validadores de informações. Crie um novo model chamado *valitadors.py* e adicione isso:



Obs: Um campo pode ter mais de um validador, e estes devem ser passados dentro de uma lista.

Agora que você já sabe como criar páginas e validadores, deixarei que crie os demais

* IS\_NOT\_EMPTY(): requer que o campo esteja preenchido com algum tipo de valor;
* IS\_NOT\_IN\_DB(banco, campo): requer que o valor já não exista no campo especificado do banco especificado;
* IS\_EMPTY\_OR(validador): o campo pode ficar vazio, mas , caso não fique, o valor deve ser válido para o validador secundário especificado;
* IS\_IMAGE(): o tipo de arquivo do upload deve ser um tipo de imagem;
* IS\_IN\_DB(banco, campo, formato): o valor deve ser um registro existente no banco de dados especificado e será salvo pelo campo especificado. O formato é apenas para exibição. Por exemplo: estou fazendo referência a um usuário já existente a partir do seu id, mas no formulário quero que aparece o seu nome, por isso meu valor de campo será o id e meu valor de formato será o nome;

Este validador irá gerar um campo do tipo select no seu formulário. Caso você não queira um campo do tipo select, deverá colocar o validador dentro de uma lista, mesmo que ele seja o único elemento.

No caso de ele não ser o único validador dentro da lista e você ainda quiser que ele seja transformado em um campo select, você deve passar os outros validadores no parâmetro \_and deste validador.

* IS\_DATETIME(formato): requer que o valor seja uma data e/ou hora no formato especificado;

Todos os validadores também têm um parâmetro chamado error\_message que recebe a mensagem que deve aparecer quando o usuário inserir um valor que não bate com o validador. Por exemplo, quando o usuário deixa em branco um campo com validador IS\_NOT\_EMPTY deve aparecer uma mensagem avisando que o campo não pode ser deixado em branco.

## CRUD

**CRUD** é um termo do inglês que indica **Create**, **Read**, **Update** e **Delete**, ou em português: **Criar**, **Ler**, **Atualizar** e **Apagar**. Cada uma dessas ações reflete uma operação que realizamos no banco de dados**.**

* **Create:**O objetivo da operação *create* é adicionar informações ao banco de dados. É exatamente o que fizemos no passo anterior: gravamos dados de alunos no banco de dados.
* **Read:** A operação *read* é aquela que solicita informações ao banco de dados. Ou seja, é quando pedimos ao banco de dados para nos mostrar algo que está salvo dentro dele.
* **Update:** Quando pedimos para fazer o *update* de algo, significa que estamos pedindo para atualizar ou mudar aquilo. Da mesma forma, a operação *update* é aquela que demanda a alteração de alguma informação contida no banco de dados.
* **Delete:** A operação *delete* é exatamente o que você deve estar imaginando. Com ela apagamos alguns registros do banco de dados.

Dentre essas quatro operações, já experimentamos a primeira no passo anterior. Vimos como é possível nos conectar ao banco de dados e inserir informações para que estas fiquem salvas. Portanto, vamos prosseguir descobrindo as outras três.

## Read

Fomos capazes de inserir alunos, professores e empresas no nosso banco de dados, mas ainda não conseguimos visualizar. Ou seja, eu tenho alunos salvos, mas eu não consigo ver quais alunos são esses. Por isso, agora precisamos de uma página para exibi-los. Existem vários modos de se fazer isso, com comandos puros de SQL em HTML, com funções de db.select do próprio web2py e etc. Para agilizar nosso aprendizado, vou mostrar um elemento do web2py pode resolver tudo e nos poupar de muita programação.

Adiciona o seguinte trecho ao seu controller:

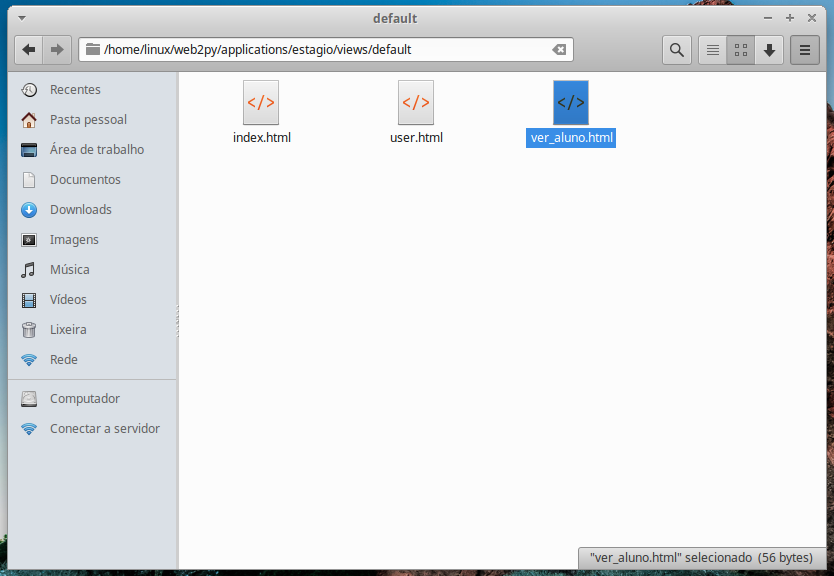
def ver\_aluno():

grid = SQLFORM.grid(Aluno)

return dict(grid=grid)

Agora a Viwer.

Acesse “web2py/applications/estagio/views/default” e crie um novo arquivo de HTML com o mesmo nome da função



Dentro do HTML, preencha com:

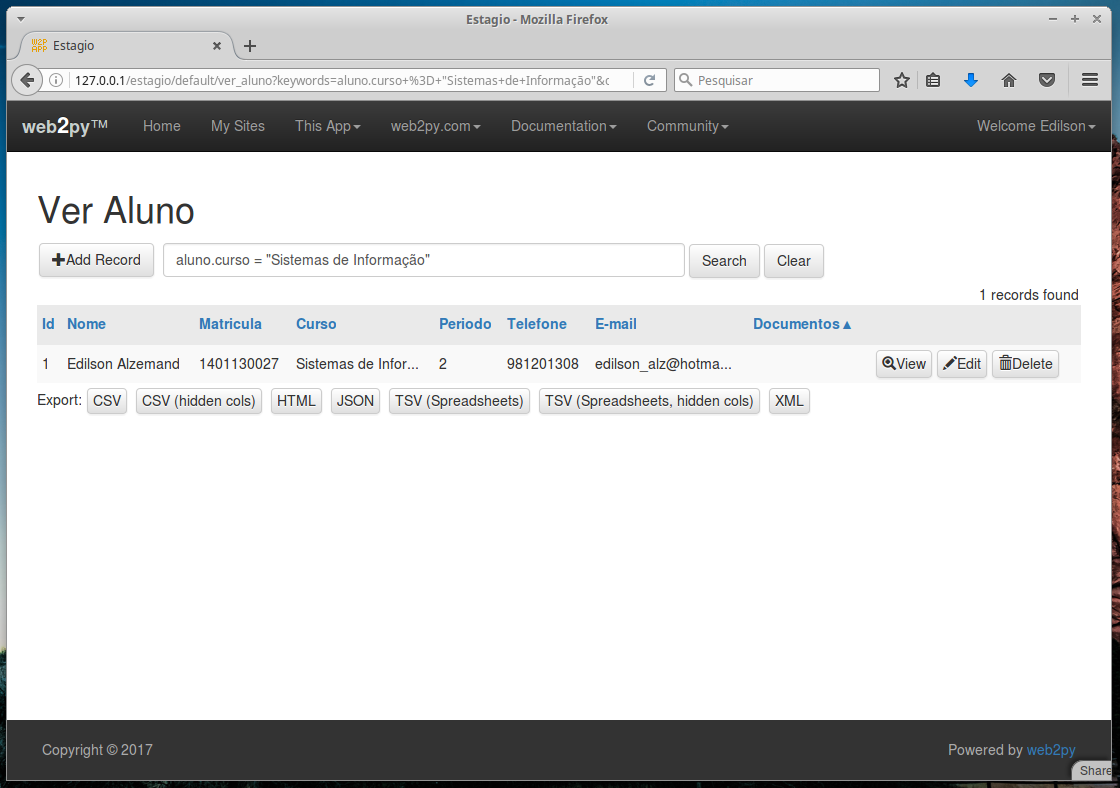
{{extend 'layout.html'}}

<h1>Ver Alunos</h1>

{{=grid}}

A SQLFORM.grid() recebe a tabela do seu banco de dados da qual você quer exibir os registros e cria uma tabela html bem organizada para isso. Ela também traz um campo de pesquisa no qual é possível você pesquisar por registros. Ela nos poupa de muita programação, não é mesmo?

Nós podemos personalizar a grid de diversas maneiras, como exibir apenas campos específicos ou mesmo ativar as funções de edição e deleção, para que sejamos capazes de editar ou deletar os registros a partir dela, além de várias outras funcionalidades.



Agora que usamos a função GRID, temos o CRUD completo de nossa aplicação. Podemos grava, visualizar, modificar e deletar facilmente. Podemos arriscar a dizer que nossa aplicação está quase completa, faltando apenas a gestão de usuários. Você deve ter percebido que qualquer pessoa pode fazer o que quiser no sistema, e não queremos isso. Vamos precisar ter autenticadores.

## Autenticação

Você não vai precisar se preocupar com criação de tabelas e usuários. O web2py faz tudo isso para você! Veja algumas das funções já prontas:

**Funções (páginas) do Usuário**

* Cadastro
* Login/Logout
* Perfil
* Recuperar Senha
* Etc.

**Tipos de usuário (Permissões)**

* Administrador
* Moderador
* Usuário padrão
* Visitante
* Etc.

A primeira tabela, **auth\_user**, é responsável por armazenar nossos usuários e todas as informações referentes a eles, como nome, sobrenome, e-mail, senha (criptografada), etc.

A segunda, **auth\_group**, cria grupos para o usuário. Estes grupos são os tipos de usuários que nossa plataforma terá. O web2py, por padrão, cria um novo grupo para cada novo usuário, sem ter permissões específicas definidas. Para separar os usuários em grupos específicos, somos nós que teremos que fazê-lo, pode ser pelo próprio banco de dados ou pelo appadmin do web2py.

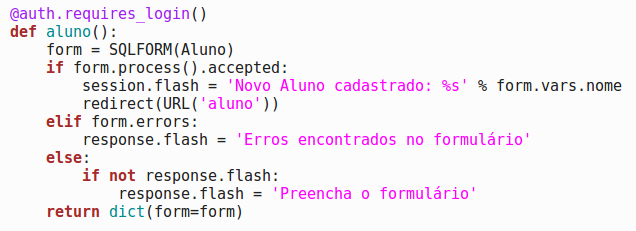
A terceira tabela, **auth\_membership**, associa as duas primeiras. Ou seja, é nela que vamos falar qual usuário pertence a qual tipo/grupo para o web2py e ele vai conseguir checar se o usuário tem permissão para fazer aquilo que está pedindo.

Neste caso, temos dois **decorators** principais:

* **@auth.requires\_login()**
* **@auth.requires\_membership(‘nome\_do\_grupo’)**

O primeiro decorator faz com que, para acessar determinada página, o usuário precisa **obrigatoriamente** estar logado. O segundo indica que, além de estar logado, ele precisa ter uma permissão específica, ou seja, pertencer a um grupo.

E como vamos usar estes decorators? Basta colocá-los antes da função da página que deve possuir tais restrições. Por exemplo, não queremos que nenhum usuário que não seja do grupo de funcionários acesse a página de cadastrar novos filmes, então podemos fazer:



Assim, o acesso a essa página ficará restrito. Agora, é seu dever aplicar qualquer um destes decorators nas outras páginas onde julgar necessário.

E o que acontecerá se um usuário sem permissão tentar acessar essa página? Nesse caso, se o usuário não estiver logado, ele será redirecionado para a página de login;

REFERENCIAS:

<https://juliarizza.wordpress.com/2015/08/03/passo-a-passo-web2py-1o-passo/>

<https://www.gitbook.com/book/juliarizza/curso-web2py/details>

<http://web2py.com/books/default/chapter/37>