

Virtual Gym

Análise e Projeto de Sistemas

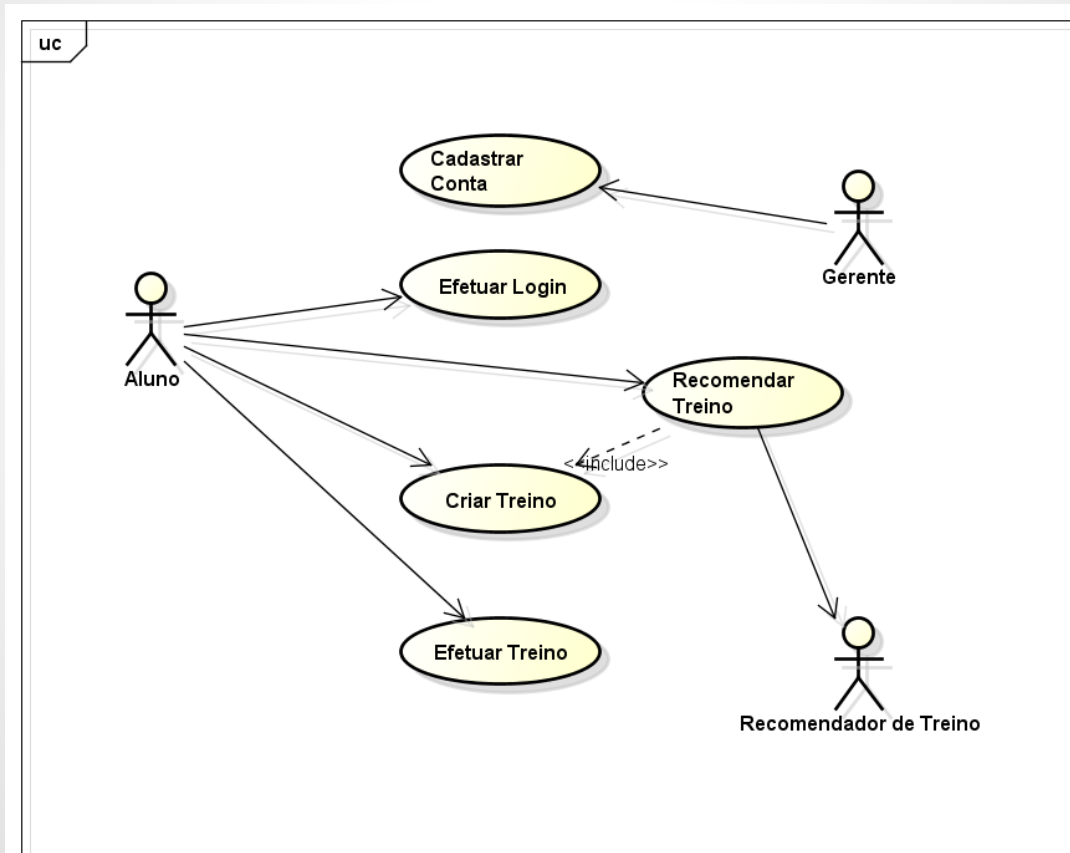
Roteiro

- Introdução
- Casos de Uso
- Mapeamento de classes de análise em elementos de projeto
- Arquitetura
- Código

Introdução

- O Virtual-Gym é um site web responsável por auxiliar o treino de um aluno na academia, como também a comunicação entre o professor e o aluno.

Modelagem de Casos de Uso



Casos de Uso

1. Cadastrar Conta
2. Efetuar Login
3. Criar Treino
4. Recomendar Treino
5. Efetuar Treino

Cadastrar Conta

- Este caso de uso é responsável por cadastrar um usuário no sistema.
- Apenas o gerente pode realizar o cadastro de uma conta.
- Para isso, o sistema verifica a permissão do usuário logado para poder criar uma nova conta.

Atores:	Gerente
Pré-condições:	Nenhuma
Pós-condições:	Uma conta válida é criada.

Fluxo de Eventos Principal

1. O ator seleciona a opção de cadastrar uma conta no sistema;
2. O ator informa os dados do aluno para realização do cadastro:
 - Login
 - Senha
 - Sexo
 - Data de nascimento
 - Gerente (valor booleano que indica se a conta é de gerente ou não)
1. O sistema verifica se há outro usuário cadastrado com os mesmos dados;
2. Uma conta é criada no sistema.

Fluxo Secundário

1. O ator deixa um campo obrigatório em branco.
2. A mensagem “Campo obrigatório não preenchido” é exibida.
3. O campo em branco fica destacado.

Diagrama de sequência

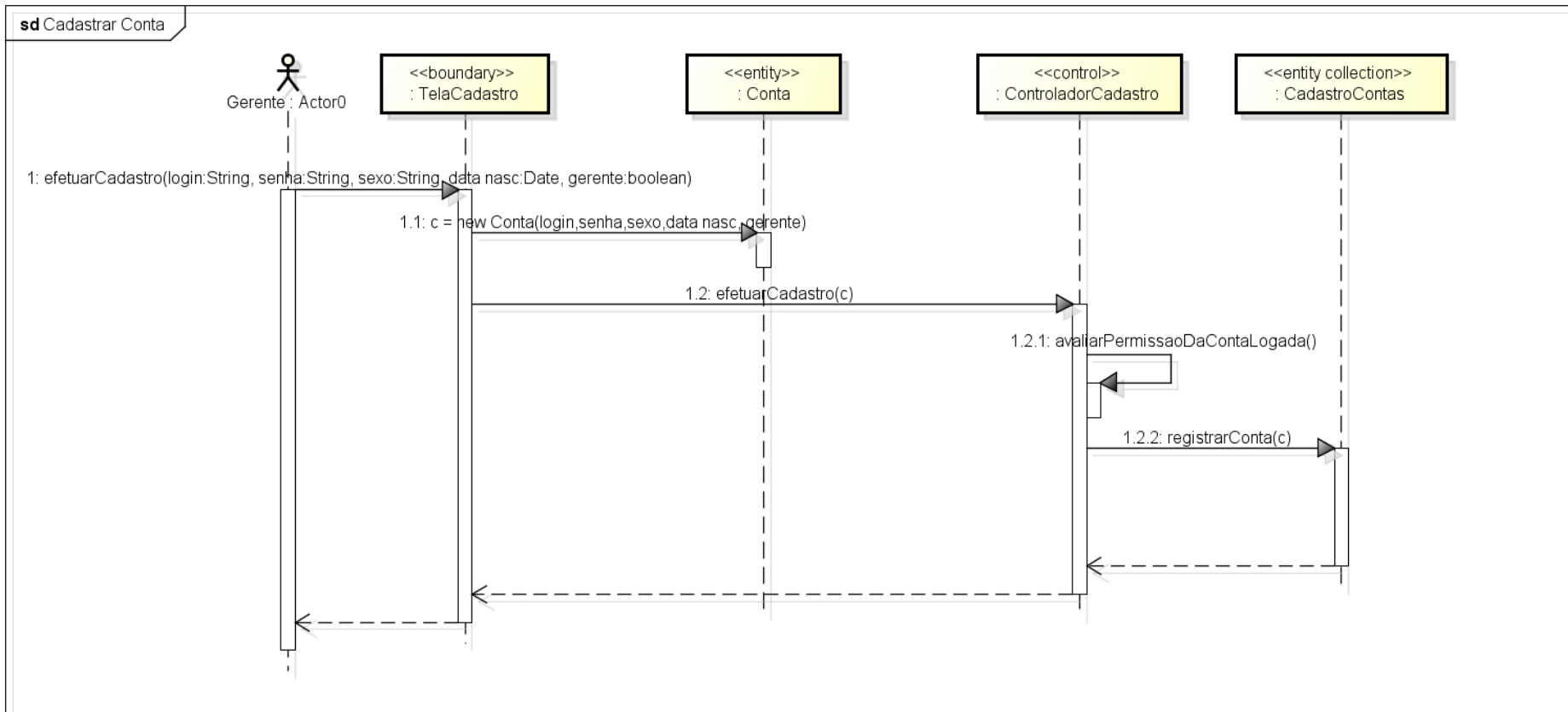
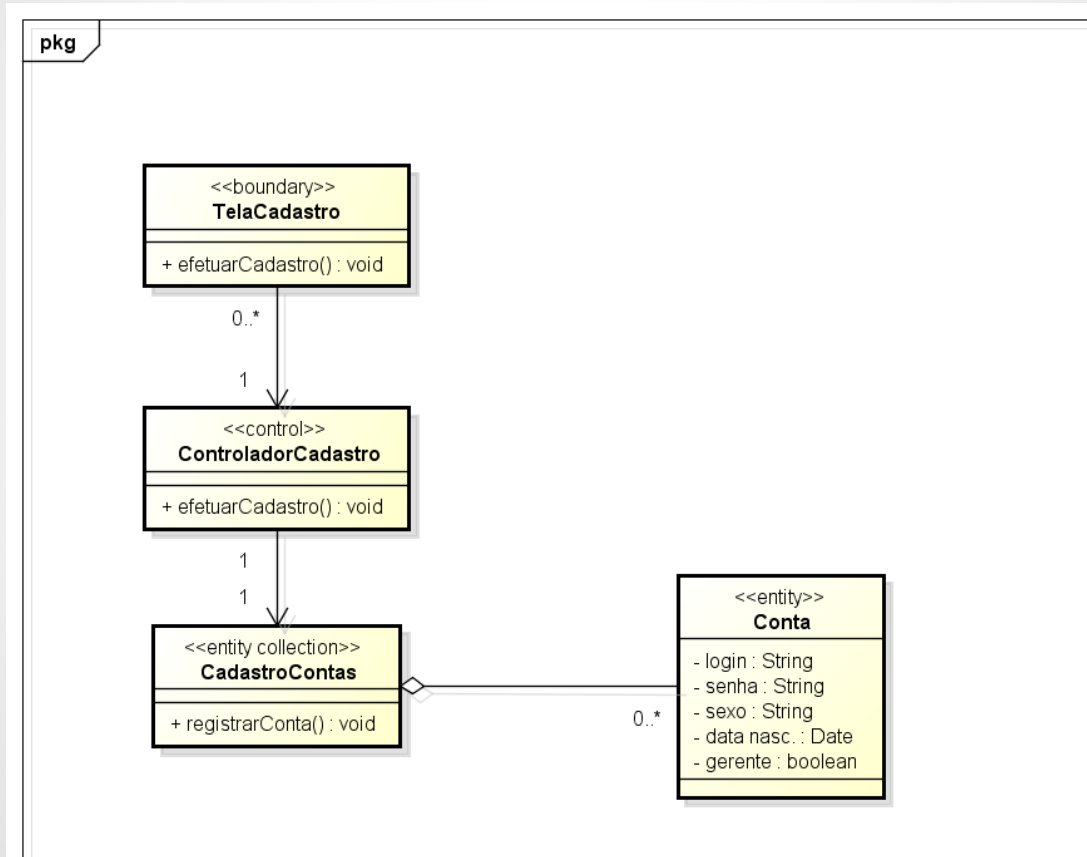


Diagrama de classe



Efetuar Login

- Este caso de uso é responsável por autenticar um usuário no sistema.

Atores:	Aluno
Pré-condições:	Nenhuma
Pós-condições:	Um usuário válido é logado e sua sessão é registrada no sistema.
Fluxo de Eventos Principal	
<ol style="list-style-type: none">1. O ator informa os campos:<ul style="list-style-type: none">· Login· Senha.2. O ator seleciona a opção de fazer o logon no sistema;3. O sistema verifica se o login e a senha preenchidos são válidos (verifica-se se o login e senha pertencem a uma conta);4. O sistema registra o início de uma sessão de uso.	
Fluxo Secundário	
<ol style="list-style-type: none">1. O ator deixa um campo obrigatório em branco.2. A mensagem “Campo obrigatório não preenchido” é exibida.3. O campo em branco fica destacado.	

Diagrama de sequência

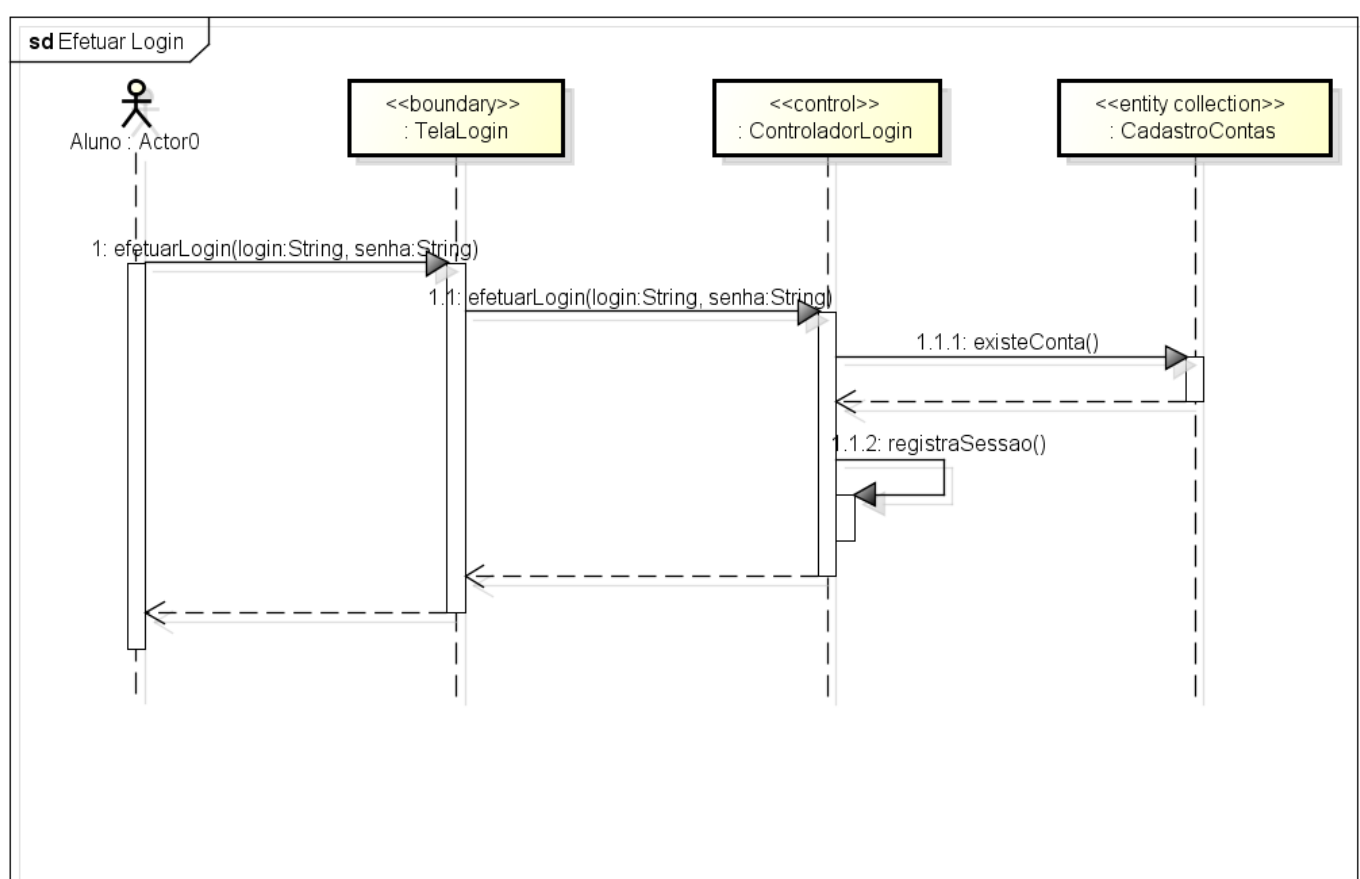
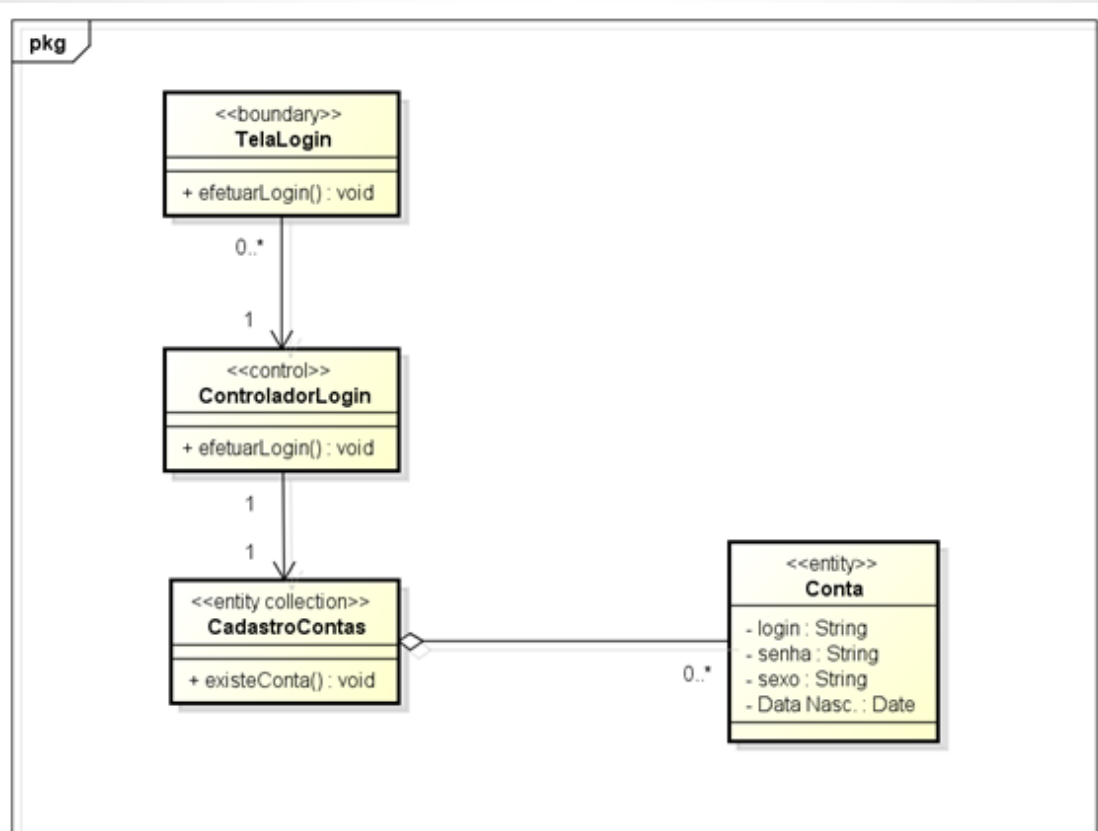


Diagrama de classe



Criar Treino

- Este caso de uso é responsável por criar um treino para o usuário.
- O treino é uma lista de exercícios que o aluno deve seguir na academia, e possui uma quantidade de vezes que deve ser realizada para que seja completada.

Atores:	Aluno
Pré-condições:	Estar logado no sistema.
Pós-condições:	É criado um treino para o aluno.
Fluxo de Eventos Principal	
<ol style="list-style-type: none">1. O usuário seleciona a opção criar treino;2. O sistema exibe uma lista de exercícios;3. O usuário seleciona os exercícios que deseja para o seu treino, e informa o número de realizações que deseja para este treino e o tipo de treino;4. O sistema registra um treino com o login do usuário logado.	
Fluxo Secundário	
<ol style="list-style-type: none">1. O ator insere dois exercícios iguais na lista. A mensagem “Exercício já existente na lista” é exibida.2. O ator já possui um treino e ainda não completou sua carga horária de exercícios. A mensagem “Carga horária do treino incompleta” é exibida.	

Diagrama de seqüência

sd Criar Treino

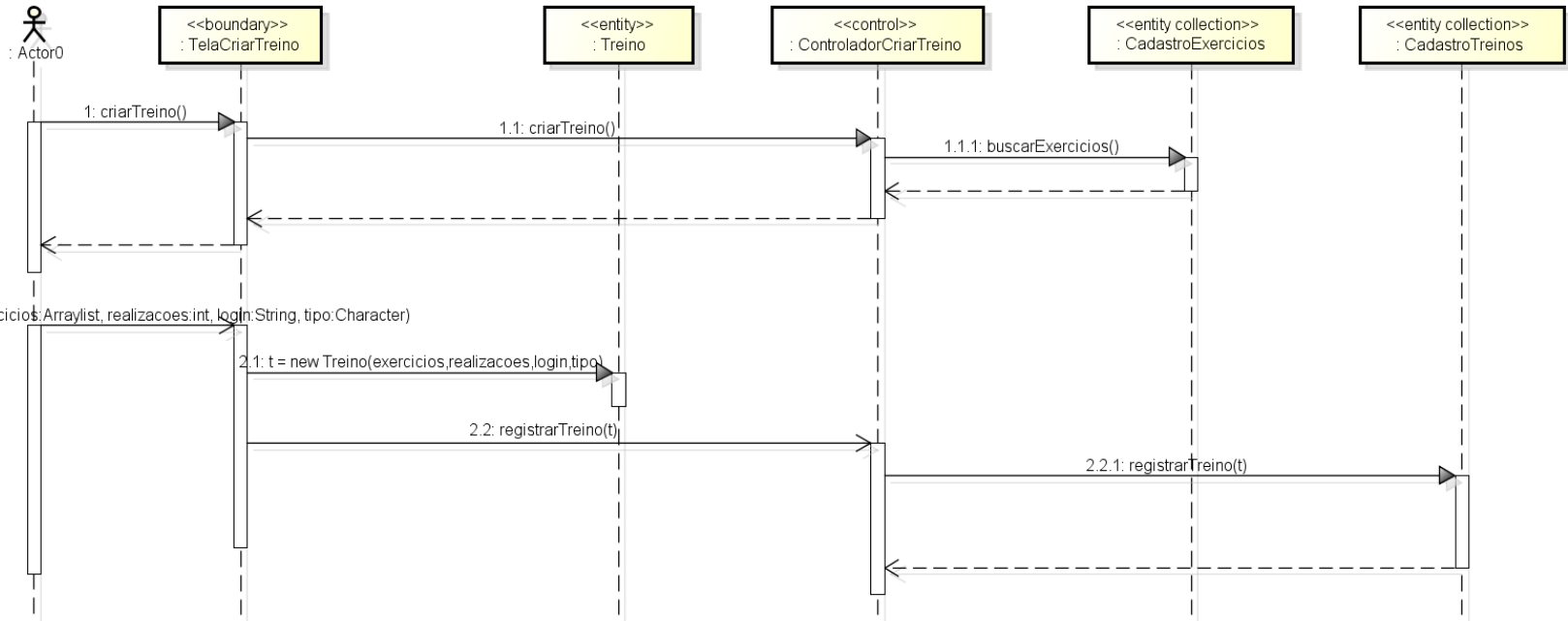
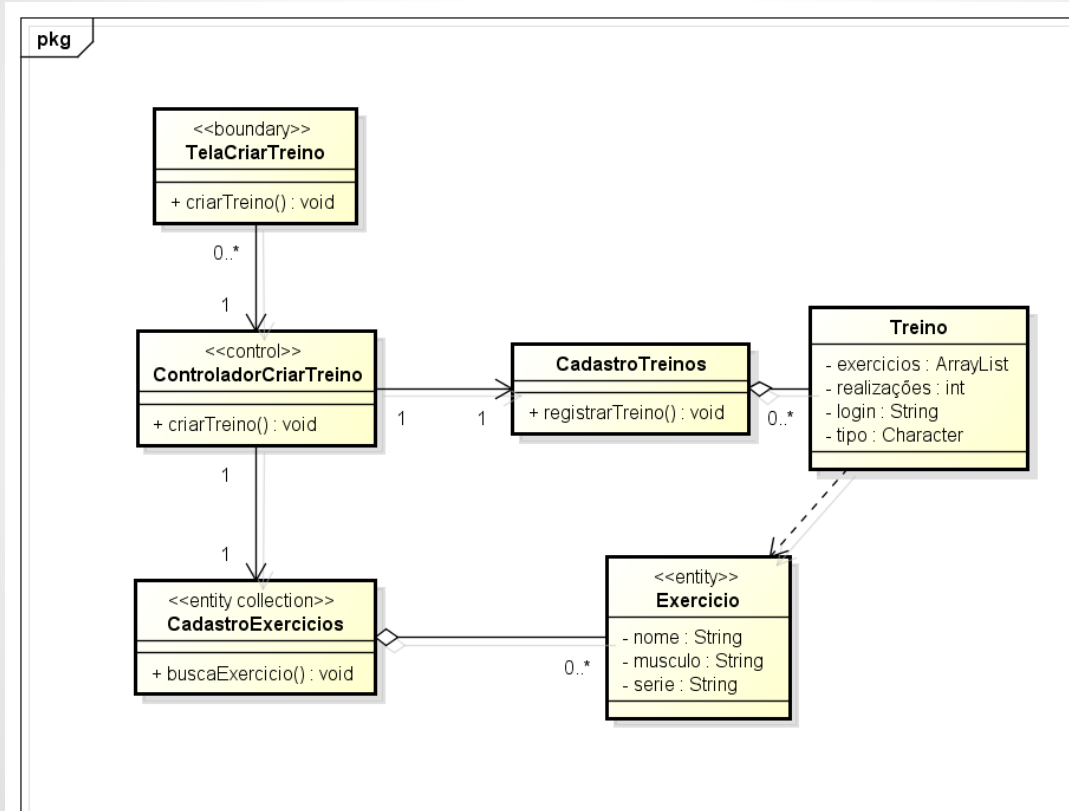


Diagrama de classe



Recomendar Treino

- Este caso de uso é responsável por recomendar um treino para o usuário.
- Isto é feito a partir de um serviço externo provido pelo sistema Recomendador de Treino, que por sua vez, é abastecido com os dados do aluno.

Atores:	Aluno e Recomendador de Treino
Pré-condições:	Estar logado no sistema.
Pós-condições:	É recomendado um treino para o usuário.

Fluxo de Eventos Principal

1. O usuário seleciona a opção de recomendar um treino;
2. O usuário informa o tipo do treino;
3. O sistema levanta as informações da conta do usuário (sexo, data de nascimento);
4. O sistema busca a lista de exercicios cadastrada no sistema;
5. O sistema envia estes dados (tipo,sexo,data nasc., exercicios) para o Recomendador de Treino;
6. O Recomendador de Treino analisa os dados recebidos e gera um treino apropriado para o aluno;
7. O sistema exhibe o treino gerado para o aluno, e pergunta se ele deseja salvar o treino recomendado;
8. Se o usuário confirmar, o sistema salva o treino gerado.

Fluxo Secundário

1. O ator já possui uma lista recomendada e ainda não completou sua carga horária de exercícos. A mensagem “Carga horária do treino incompleta” é exibida.
2. Quando o treino gerado é exibido, o usuário cancela a opção de salvar o treino. O sistema não salva o treino gerado.

Diagrama de sequência

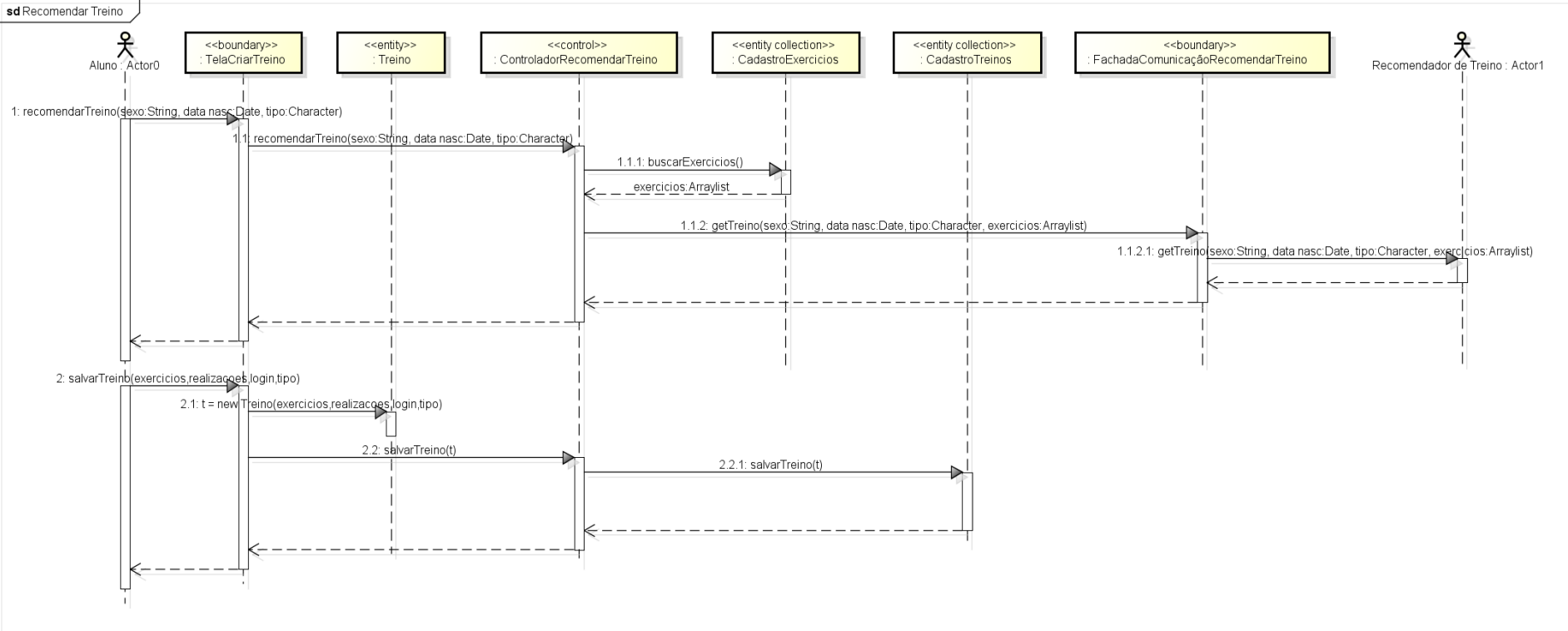
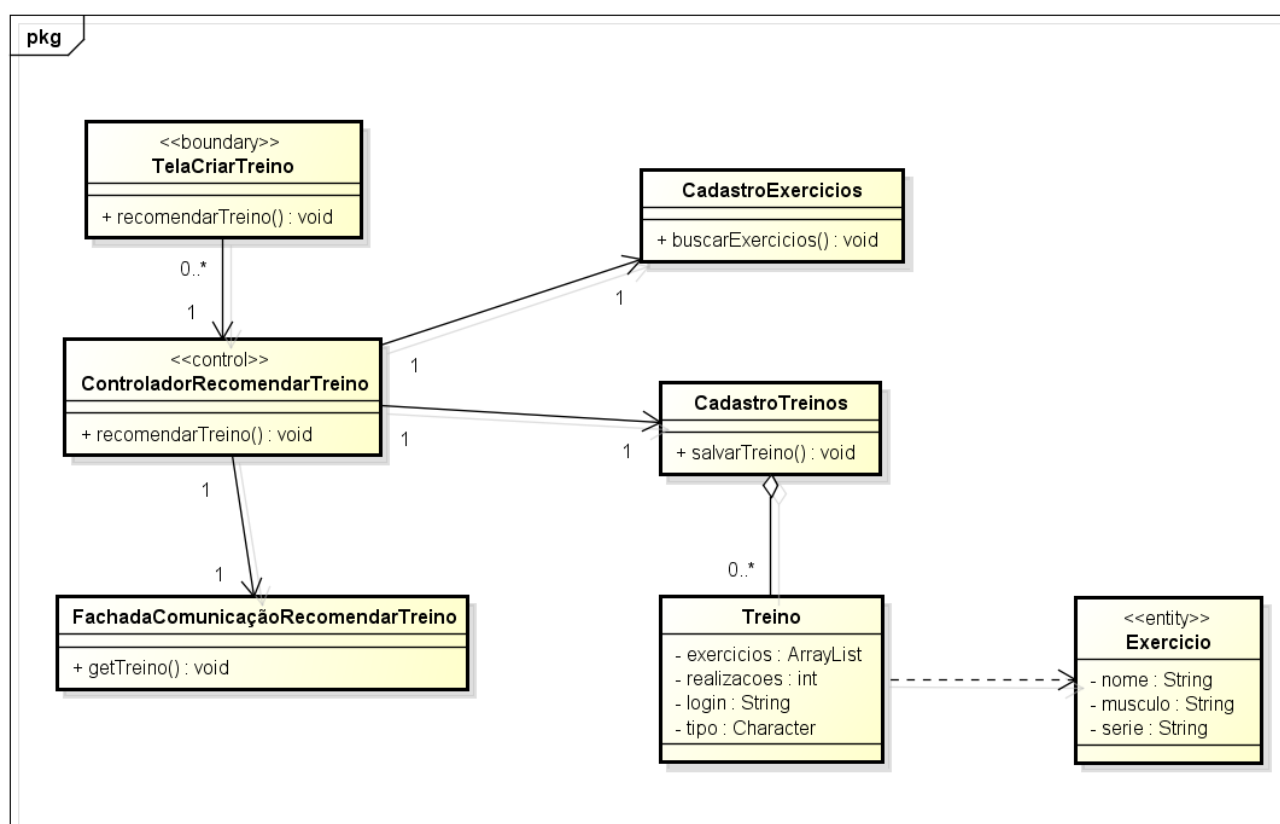


Diagrama de classe



Efetuar Treino

- Este caso de uso é responsável por desenvolver o treino do usuário.
- O seu treino é iniciado e será acompanhado cada exercício de sua lista, respeitando o tempo de descanso necessário para cada exercício.

Atores:	Aluno
Pré-condições:	Estar logado no sistema e possuir um treino.
Pós-condições:	O treino que o usuário possui é iniciado.

Fluxo de Eventos Principal

1. O ator seleciona a opção de efetuar o treino;
2. O sistema apresenta a lista de exercícios do usuário, começando pelo primeiro item;
3. O sistema aguarda o usuário selecionar o próximo exercício;
4. O sistema cronometra o tempo de descanso, e em seguida caminha para o próximo exercício;
5. Enquanto existir o próximo exercício, os passos 3 e 4 são repetidos;
6. Se não houver mais exercícios, o treino é encerrado.

Fluxo Secundário

1. O ator não possui um treino.
2. A mensagem “Treino inexistente” é exibida.

Diagrama de sequência

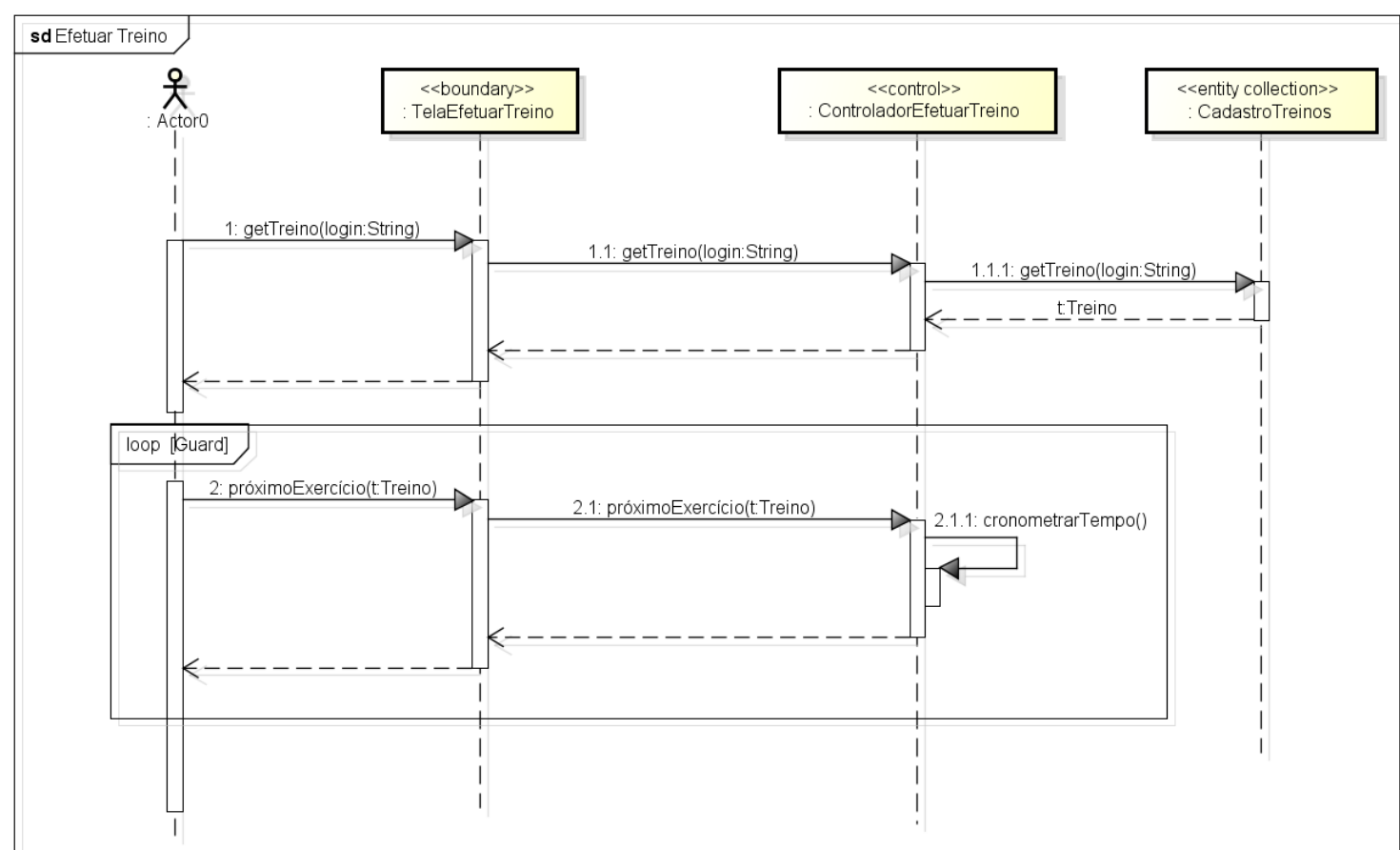
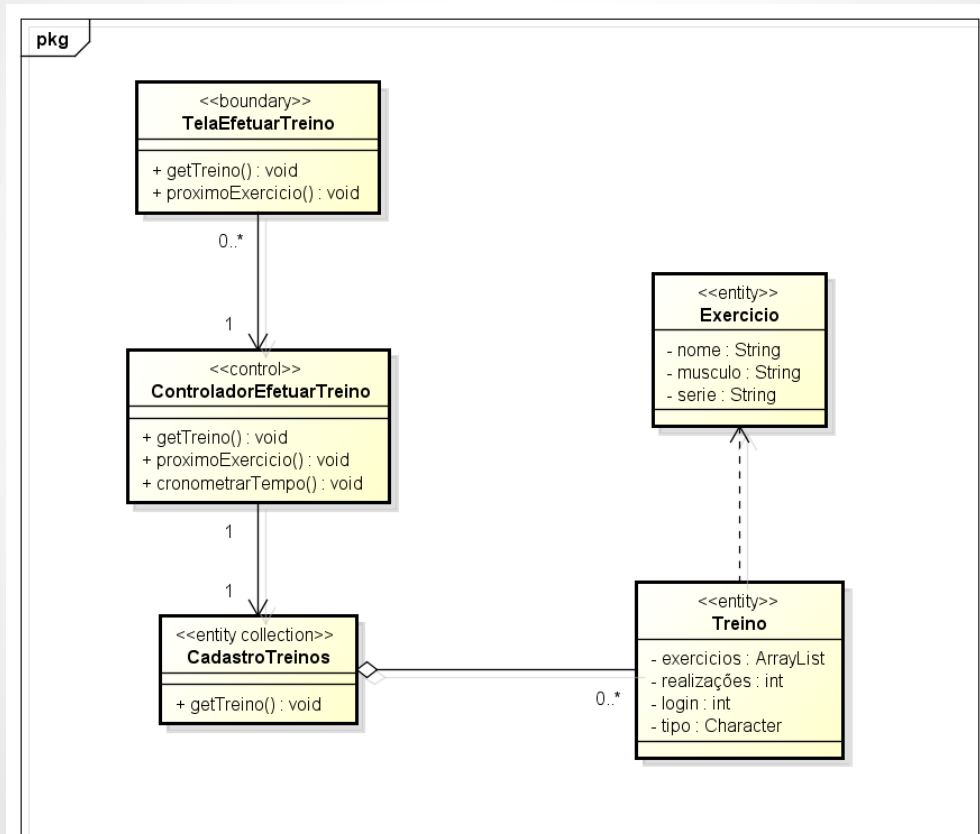


Diagrama de classe



Mapeamento de classes de análise em elementos de projeto

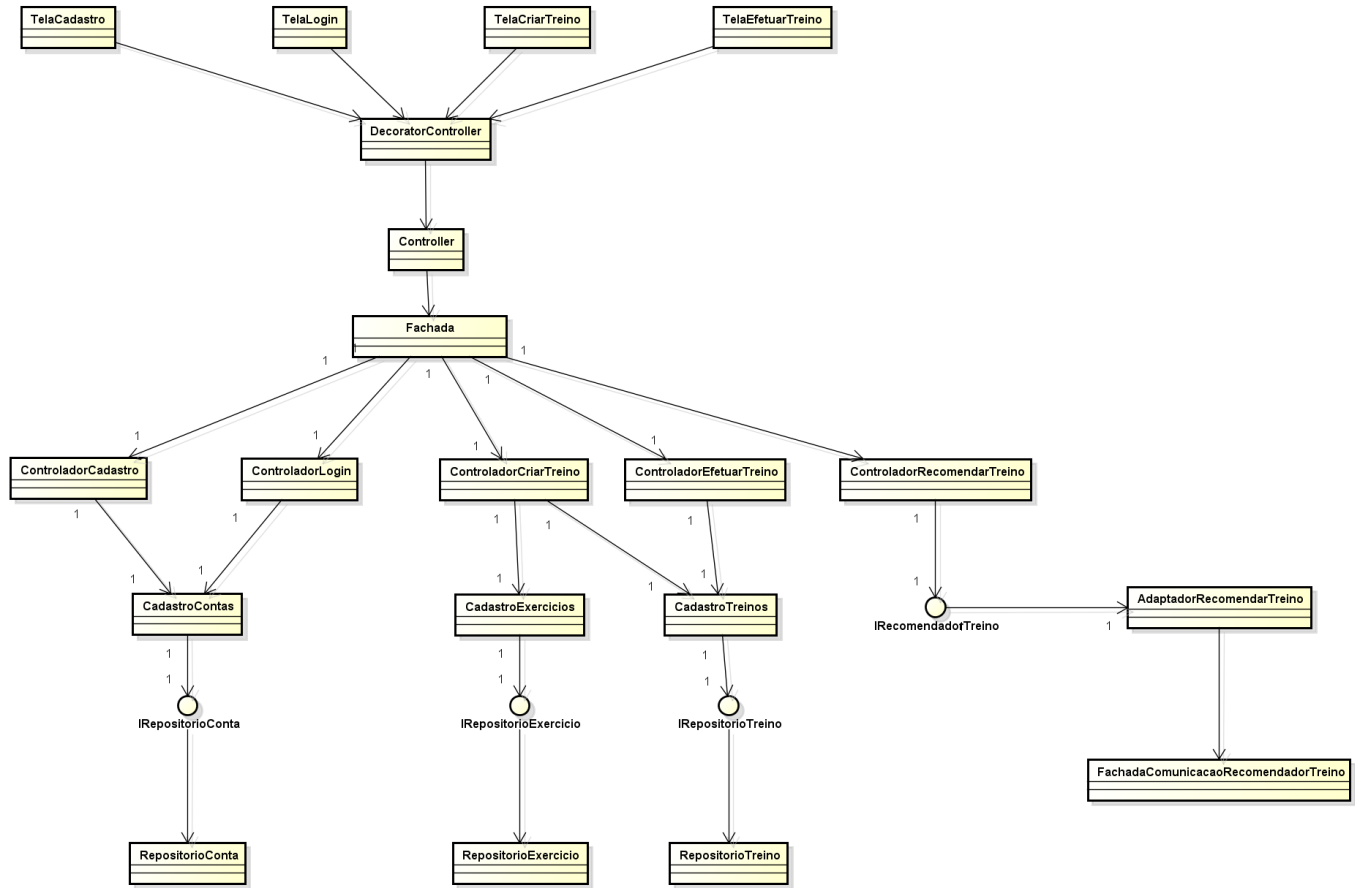
Classes de Análise	Elementos do Projeto
	Fachada
TelaCadastro	TelaCadastro
TelaLogin	TelaLogin
TelaCriarTreino	TelaCriarTreino
TelaEfetuarTreino	TelaEfetuarTreino
ControladorCadastro	ControladorCadastro
ControladorLogin	ControladorLogin
ControladorCriarTreino	ControladorCriarTreino
ControladorEfetuarTreino	ControladorEfetuarTreino
ControladorRecomendarTreino	ControladorRecomendarTreino
FachadaComunicacaoRecomendarTreino	IRecomendarTreino AdaptadorRecomendarTreino FachadaComunicacaoRecomendarTreino

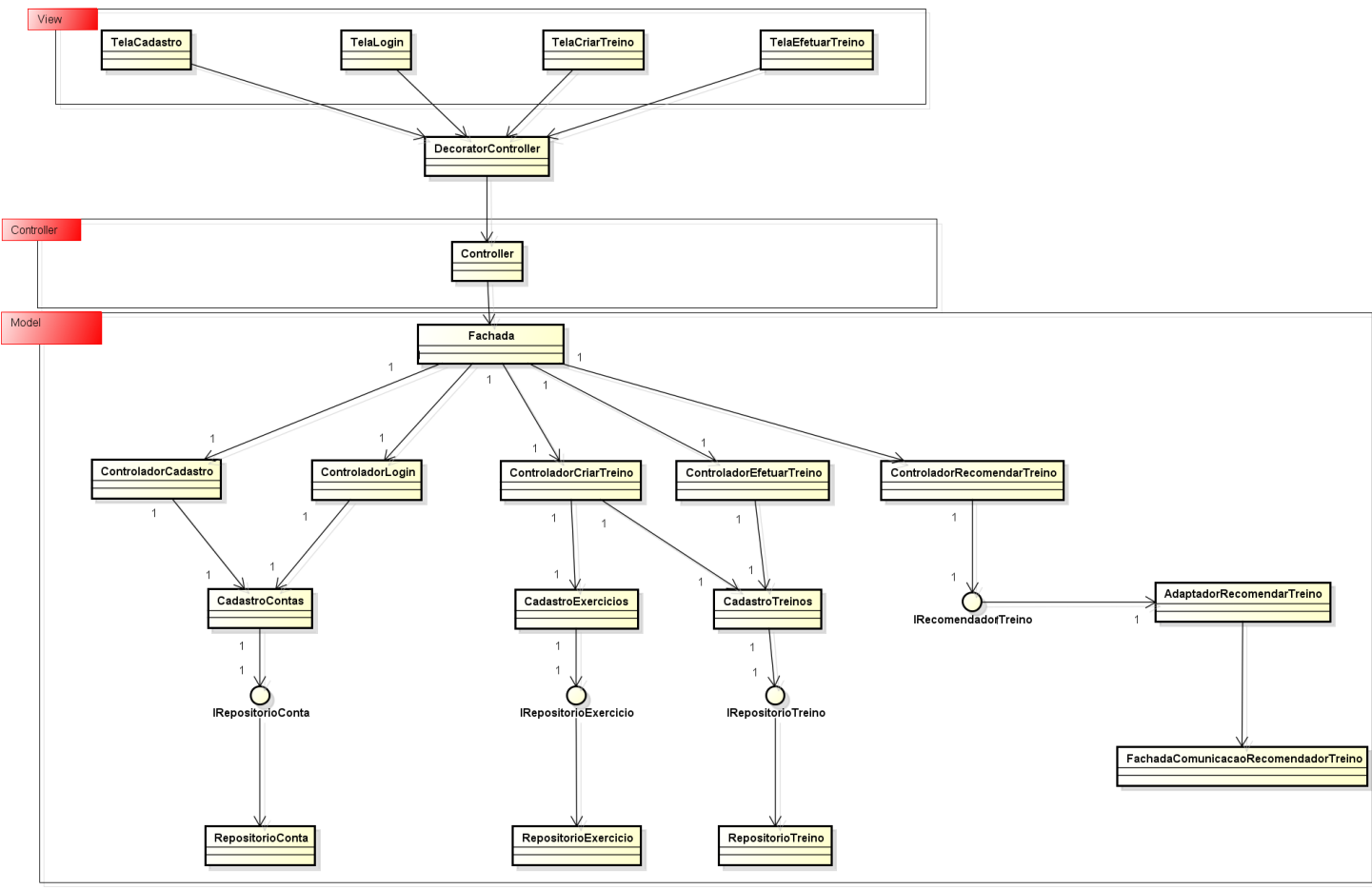
Mapeamento de classes de análise em elementos de projeto

Classes de Análise	Elementos do Projeto
CadastroContas	CadastroContas IRepositorioConta RepositorioConta
CadastroExercicios	CadastroExercicios IRepositorioExercicio RepositorioExercicio
CadastroTreinos	CadastroTreinos IRepositorioTreino RepositorioTreino
Conta	Conta
Exercicio	Exercício
Treino	Treino

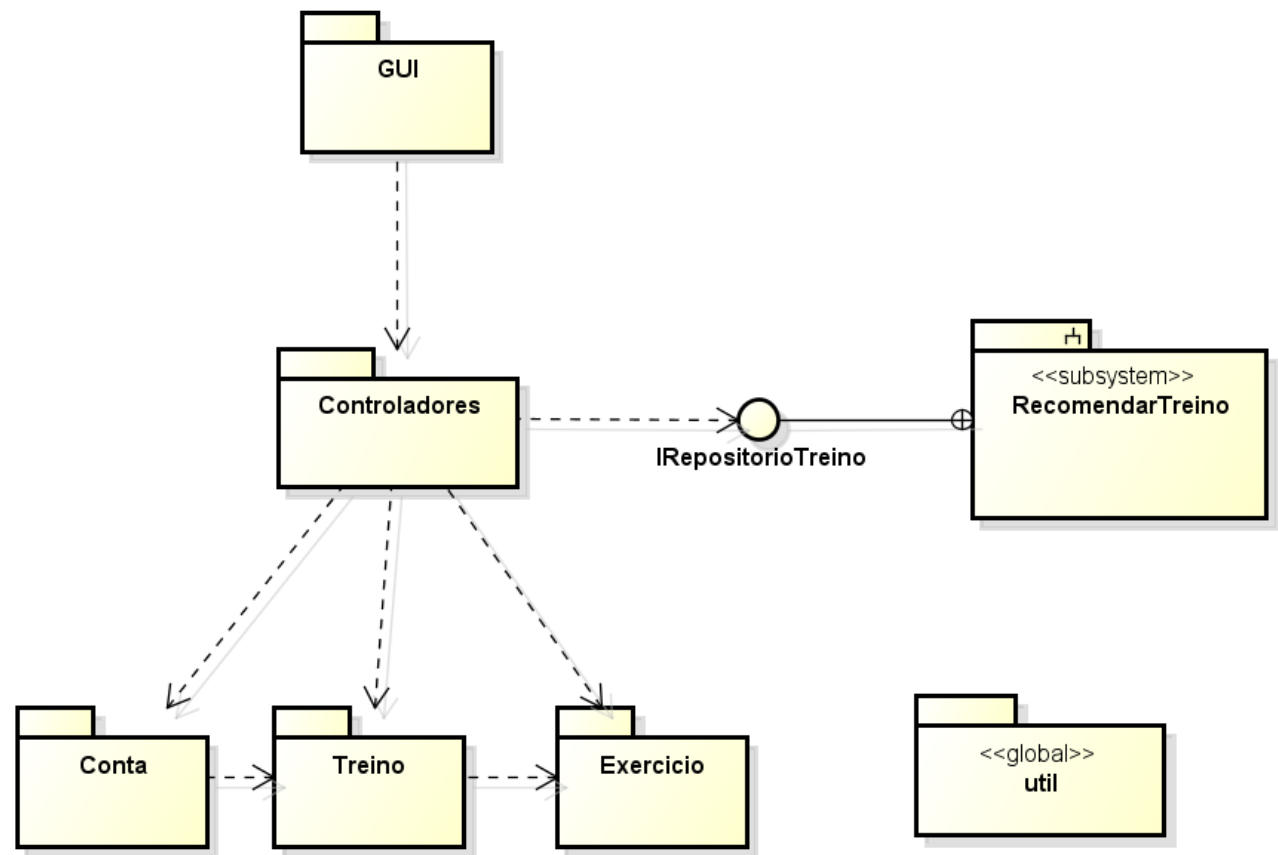
Arquitetura

- Organizada em pacotes
- Utiliza subsistema
- Aplicação de padrões de projeto
 - Façade
 - Adapter
 - Bridge
 - Decorator
 - Singleton

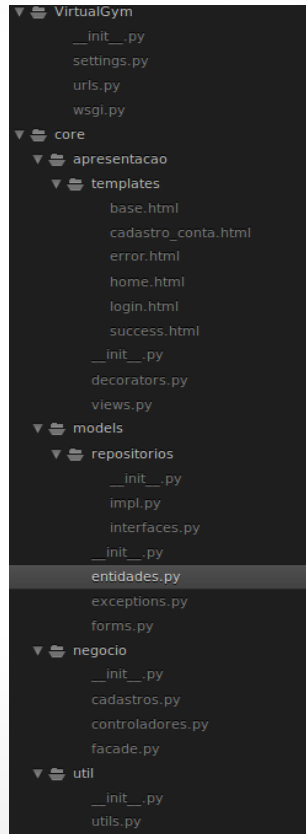




pkg



Código - Organização



Código - MVC (Models)

```
7 class Conta (models.Model):
8     """
9     """
10    """
11    Classe que representa a conta de um usuário do sistema
12    """
13    """
14    # constantes de classe
15    choices_sex = (
16        ('M', 'Masculino'),
17        ('F', 'Feminino'),
18    )
19    """
20    # atributos
21    login = models.CharField(max_length = 255, unique = True)
22    senha = models.CharField(max_length = 16)
23    sexo = models.CharField(max_length = 2, choices = choices_sex)
24    data_nascimento = models.DateField()
25    gerente = models.BooleanField()
26    """
27    class Meta:
28        app_label = 'core'
29    """
30    def __unicode__(self):
31    """
32    return "[login='%s', sexo='%s']" % (self.login, self.sexo)
33    """
34    class Exercico (models.Model):
35    """
36    """
37    """
38    Classe que representa um exercicio fisico
39    """
40    """
41    # atributos
42    nome = models.CharField(max_length = 255)
43    musculo = models.CharField(max_length = 255) # add fixed choices
44    serie = models.CharField(max_length = 255)
45    """
46    class Meta:
47        app_label = 'core'
48    """
49    def __unicode__(self):
50    """
51    return "[nome='%s', musculo='%s', serie='%s']" % (self.nome, self.musculo, self.serie)
52    """
53    class Treino (models.Model):
54    """
55    """
56    """
57    Classe que representa um treino, composto de repetições de exercicios
58    """
59    """
60    # constantes de classe
61    choices_tipo = (
62        ('1', '1'),
63        ('2', '2'),
64        ('3', '3'),
65    )
66    """
67    # atributos
68    exercicios = models.ManyToManyField('Exercico')
69    realizacoes = models.IntegerField()
70    tipo = models.CharField(max_length = 1, choices = choices_tipo)
71    conta = models.ForeignKey('Conta')
72    """
73    class Meta:
74        app_label = 'core'
75    """
76    def __unicode__(self):
77    """
78    return "[exercicios='%s', realizacoes='%d']" % (self.exercicios, self.realizacoes)
79    """
```

```
33 class Exercico (models.Model):
34    """
35    """
36    """
37    Classe que representa um exercicio fisico
38    """
39    """
40    # atributos
41    nome = models.CharField(max_length = 255)
42    musculo = models.CharField(max_length = 255) # add fixed choices
43    serie = models.CharField(max_length = 255)
44    """
45    class Meta:
46        app_label = 'core'
47    """
48    def __unicode__(self):
49    """
50    return "[nome='%s', musculo='%s', serie='%s']" % (self.nome, self.musculo, self.serie)
51    """
52    class Treino (models.Model):
53    """
54    """
55    """
56    Classe que representa um treino, composto de repetições de exercicios
57    """
58    """
59    # constantes de classe
60    choices_tipo = (
61        ('1', '1'),
62        ('2', '2'),
63        ('3', '3'),
64    )
65    """
66    # atributos
67    exercicios = models.ManyToManyField('Exercico')
68    realizacoes = models.IntegerField()
69    tipo = models.CharField(max_length = 1, choices = choices_tipo)
70    conta = models.ForeignKey('Conta')
71    """
72    class Meta:
73        app_label = 'core'
74    """
75    def __unicode__(self):
76    """
77    return "[exercicios='%s', realizacoes='%d']" % (self.exercicios, self.realizacoes)
78    """
```

Código - MVC (Controller)

```
14 class Controller:
15
16     """
17     O Controller contém todas os métodos (em django, as "views") chamados durante o direcionamento de url.
18     Esses métodos correspondem aos controllers do modelo MVC.
19     """
20
21     @staticmethod
22     def efetuar_cadastro(request):
23
24         # POST
25         if request.method == 'POST':
26
27             form = ContaForm(request.POST)
28
29             if form.is_valid():
30
31                 login = form.cleaned_data['login']
32                 senha = form.cleaned_data['senha']
33                 sexo = form.cleaned_data['sexo']
34                 data_nascimento = form.cleaned_data['data_nascimento']
35
36                 try:
37
38                     conta = Conta()
39
40                     conta.login = login
41                     conta.senha = senha
42                     conta.sexo = sexo
43                     conta.data_nascimento
44
45                     Fachada().efetuar_cadastro(login, senha, sexo, data_nascimento)
46
47                     return render(request, 'success.html', {'mensagem': 'Conta cadastrada com sucesso!', 'categoria': 'cadastro'})
48
49                 except ContaExistente, e:
50
51                     return render(request, 'error.html', {'mensagem': str(e), 'categoria': 'cadastro de usuário'})
52
53                 else:
54
55                     return render(request, 'cadastro_conta.html', {'form': form})
56
57         # GET
58         else:
59
60             form = ContaForm()
```

git branch: master, index: v, working: 1#, Line 132, Column 42

Tab Size: 4 Python

Código - MVC (Views)

```
1 <html>
2
3 <head>
4
5 <title> {% block title %} {% endblock %} </title>
6
7 </head>
8
9 <body>
10
11 <h1> {% block main_header %} {% endblock %} </h1>
12
13 {% block content %}
14
15 {% endblock %}
16
17 </body>
18
19 </html>
```

```
1 {% extends 'base.html' %}
2
3 {% block title %} Cadastro de um novo usuário {% endblock %}
4
5 {% block main_header %} Novo Usuário {% endblock %}
6
7 {% block content %}
8
9 <form action="/cadastros/conta/" method="post">
10     {% csrf_token %}
11     {{ form }} <br/>
12     {{ form.errors }} <br/>
13     <input type="submit" value="Submit" />
14 </form>
15 {{ form.non_field_errors }} <br/>
16
17 {% endblock %}
```

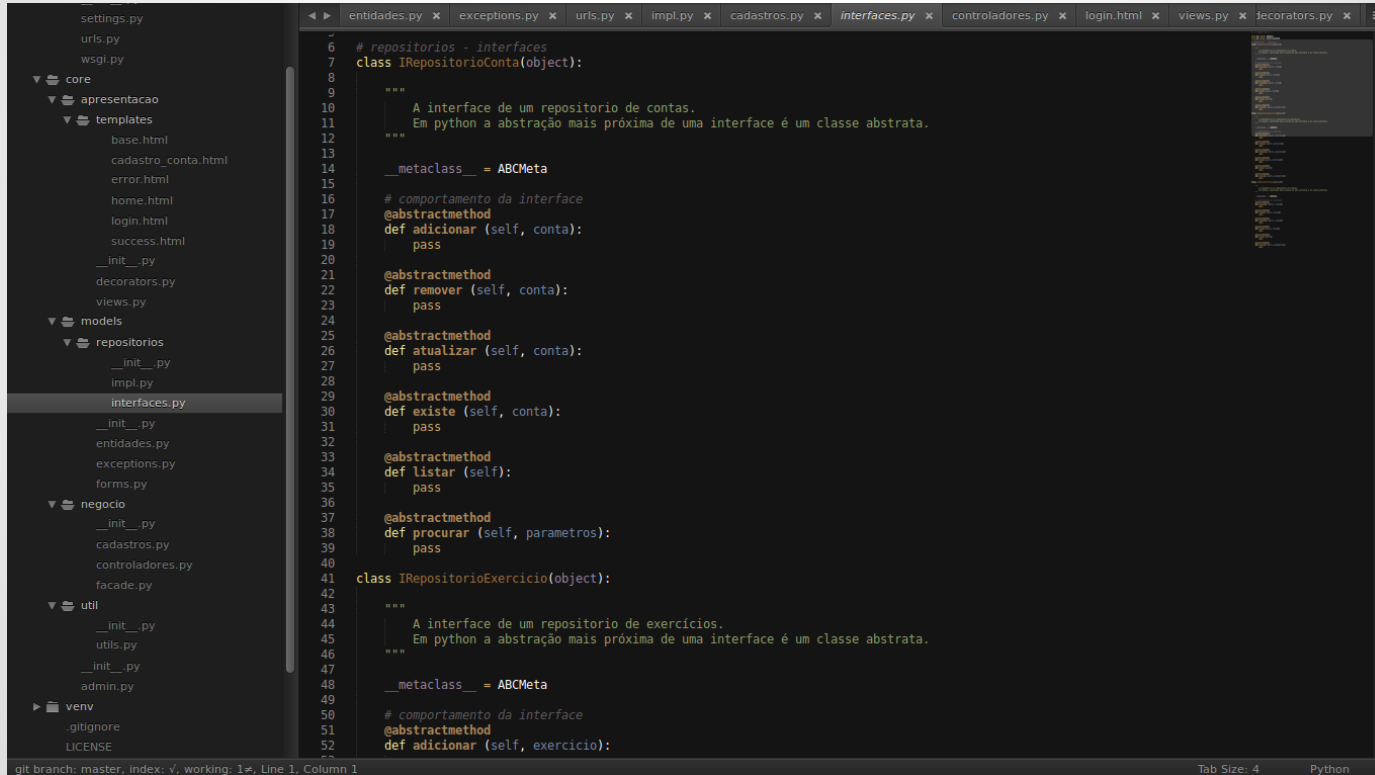
```
1 {% extends 'base.html' %}
2
3 {% block title %} Virtual Gym - Login {% endblock %}
4
5 {% block main_header %} Login {% endblock %}
6
7 {% block content %}
8
9 <form action="/login/" method="post">
10     {% csrf_token %}
11     {{ form }} <br/>
12     {{ form.errors }} <br/>
13     <input type="submit" value="Submit" />
14 </form>
15 {{ form.non_field_errors }} <br/>
16
17 <hr/>
18
19 {% for message in mensagens_adicionais %}
20     {{ message }} <br/>
21 {% endfor %}
22
23 {% endblock %}
```

git branch: master, index: v, working: 1#, Line 1, Column 1

git branch: master, index: v, working: 1#, Line 1, Column 1

git branch: master, index: v, working: 1#, Line 14, Column 12

Código - CRUD (Cadastrar Conta)



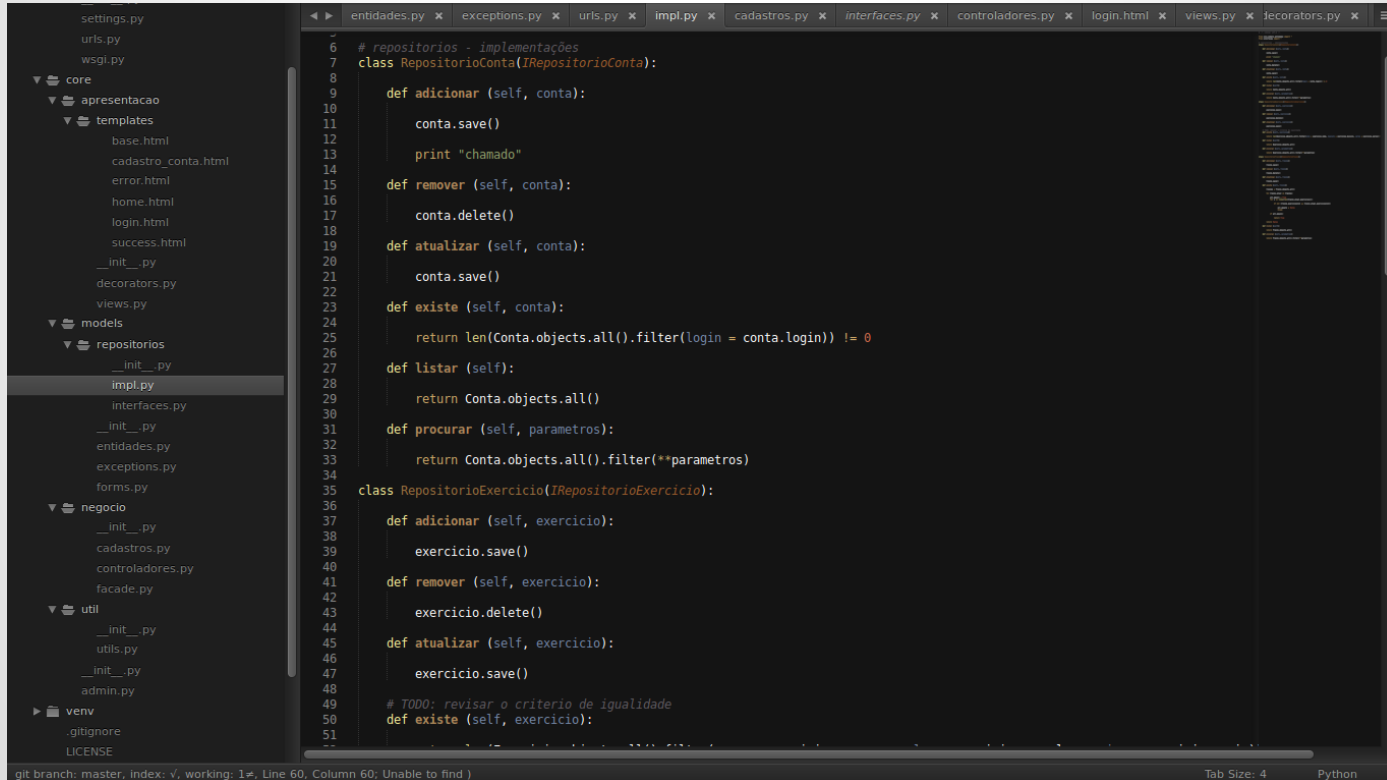
The image shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like 'core', 'negocio', and 'util', and files like 'interfaces.py'. The code editor shows the content of 'interfaces.py', which defines two abstract classes: 'IRepositorioConta' and 'IRepositorioExercicio'. Both classes use 'ABCMeta' as a metaclass and define abstract methods for adding, removing, updating, listing, and searching for items. The 'IRepositorioConta' class has methods for 'adicionar', 'remover', 'atualizar', 'existe', and 'listar'. The 'IRepositorioExercicio' class has a method for 'adicionar'.

```
6 # repositorios - interfaces
7 class IRepositorioConta(object):
8
9
10     """
11     A interface de um repositorio de contas.
12     Em python a abstração mais próxima de uma interface é um classe abstrata.
13     """
14
15     __metaclass__ = ABCMeta
16
17     # comportamento da interface
18     @abstractmethod
19     def adicionar(self, conta):
20         pass
21
22     @abstractmethod
23     def remover(self, conta):
24         pass
25
26     @abstractmethod
27     def atualizar(self, conta):
28         pass
29
30     @abstractmethod
31     def existe(self, conta):
32         pass
33
34     @abstractmethod
35     def listar(self):
36         pass
37
38     @abstractmethod
39     def procurar(self, parametros):
40         pass
41
42 class IRepositorioExercicio(object):
43
44     """
45     A interface de um repositorio de exercicios.
46     Em python a abstração mais próxima de uma interface é um classe abstrata.
47     """
48
49     __metaclass__ = ABCMeta
50
51     # comportamento da interface
52     @abstractmethod
53     def adicionar(self, exercicio):
```

git branch: master, index: ✓, working: 1✗, Line 1, Column 1

Tab Size: 4 Python

Código - CRUD (Cadastrar Conta)



```
6 # repositórios - implementações
7 class RepositorioConta(IRepositorioConta):
8
9     def adicionar (self, conta):
10
11         conta.save()
12
13         print "chamado"
14
15     def remover (self, conta):
16
17         conta.delete()
18
19     def atualizar (self, conta):
20
21         conta.save()
22
23     def existe (self, conta):
24
25         return len(Conta.objects.all().filter(login = conta.login)) != 0
26
27     def listar (self):
28
29         return Conta.objects.all()
30
31     def procurar (self, parametros):
32
33         return Conta.objects.all().filter(**parametros)
34
35 class RepositorioExercicio(IRepositorioExercicio):
36
37     def adicionar (self, exercicio):
38
39         exercicio.save()
40
41     def remover (self, exercicio):
42
43         exercicio.delete()
44
45     def atualizar (self, exercicio):
46
47         exercicio.save()
48
49     # TODO: revisar o critério de igualdade
50     def existe (self, exercicio):
51
```

git branch: master, index: v, working: 1, Line 60, Column 60; Unable to find)

Tab Size: 4 Python

Código - CRUD (Cadastrar Conta)

```
6
7 class CadastroContas(object):
8
9     """
10    """
11    """
12    O Cadastro de contas adiciona as regras de negócio à utilização do repositório de contas
13    """
14
15    def __init__(self, repositorio):
16
17        self.repositorio = repositorio
18
19    # mesma funcionalidade da função planejada 'registrarConta'
20    def adicionar(self, conta):
21
22        if not self.existe(conta):
23
24            self.repositorio.adicionar(conta)
25
26        else:
27
28            raise ContaExistente(conta)
29
30    def remover(self, conta):
31
32        if self.existe(conta):
33
34            self.repositorio.remover(conta)
35
36        else:
37
38            raise ContaInexistente(conta)
39
40    def atualizar(self, conta):
41
42        if self.existe(conta):
43
44            self.repositorio.atualizar(conta)
45
46        else:
47
48            raise ContaInexistente(conta)
49
50    # mesma funcionalidade da função planejada 'existeConta'
51    def existe(self, conta):
52
53        return self.repositorio.existe(conta)
```

git branch: master, index: ✓, working: 1✗, Line 60, Column 37 Tab Size: 4 Python

Código - CRUD (Cadastrar Conta)

```
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52

class ControladorCadastro(object):
    """
    O Controlador de cadastro. Toma as informações vindas da fachada (que por sua vez vêm da interação com o usuário)
    e faz operações com objetos Conta utilizando
    """
    def __init__(self):
        self.cadastro_contas = CadastroContas(RepositorioConta())

    # methods

    @raises: ContaExistente
    def efetuar_cadastro(self, login, senha, sexo, data_nascimento):
        nova_conta = Conta()

        nova_conta.login = login
        nova_conta.senha = senha
        nova_conta.sexo = sexo
        nova_conta.data_nascimento = data_nascimento

        self.cadastro_contas.adicionar(nova_conta)

    # ...

class ControladorLogin(object):
    def __init__(self):
        self.cadastro_contas = CadastroContas(RepositorioConta())

    # methods

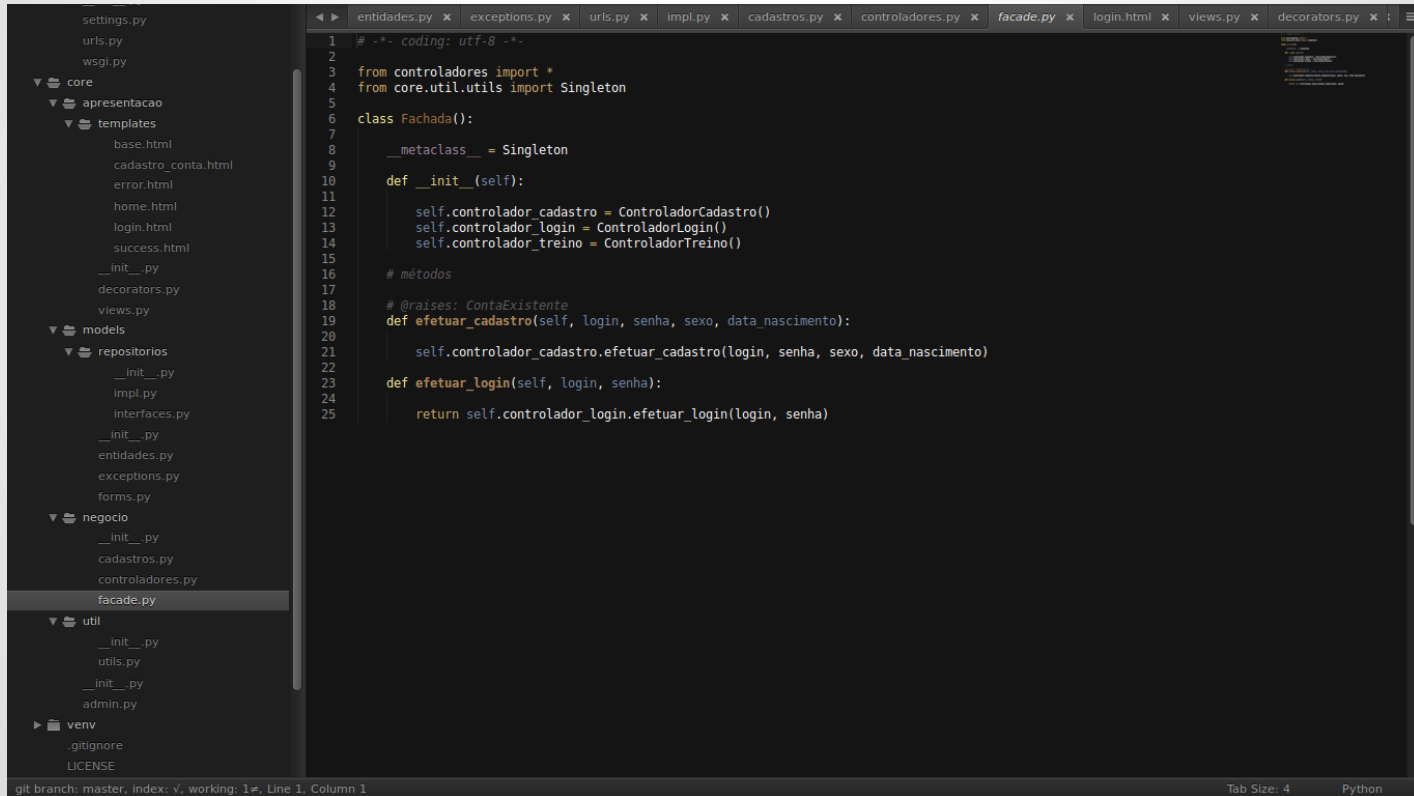
    def efetuar_login(self, login, senha):
        conta = Conta(login = login, senha = senha)

        if self.cadastro_contas.existe(conta):
            # check if the data is correct
            conta_sistema = self.cadastro_contas.procurar({'login': login})[0]

            if conta_sistema.senha != conta.senha:
                raise IncorrectLoginData(login)
```

git branch: master, index: v, working: 1, Line 1, Column 24 Tab Size: 4 Python

Código - CRUD (Cadastrar Conta)

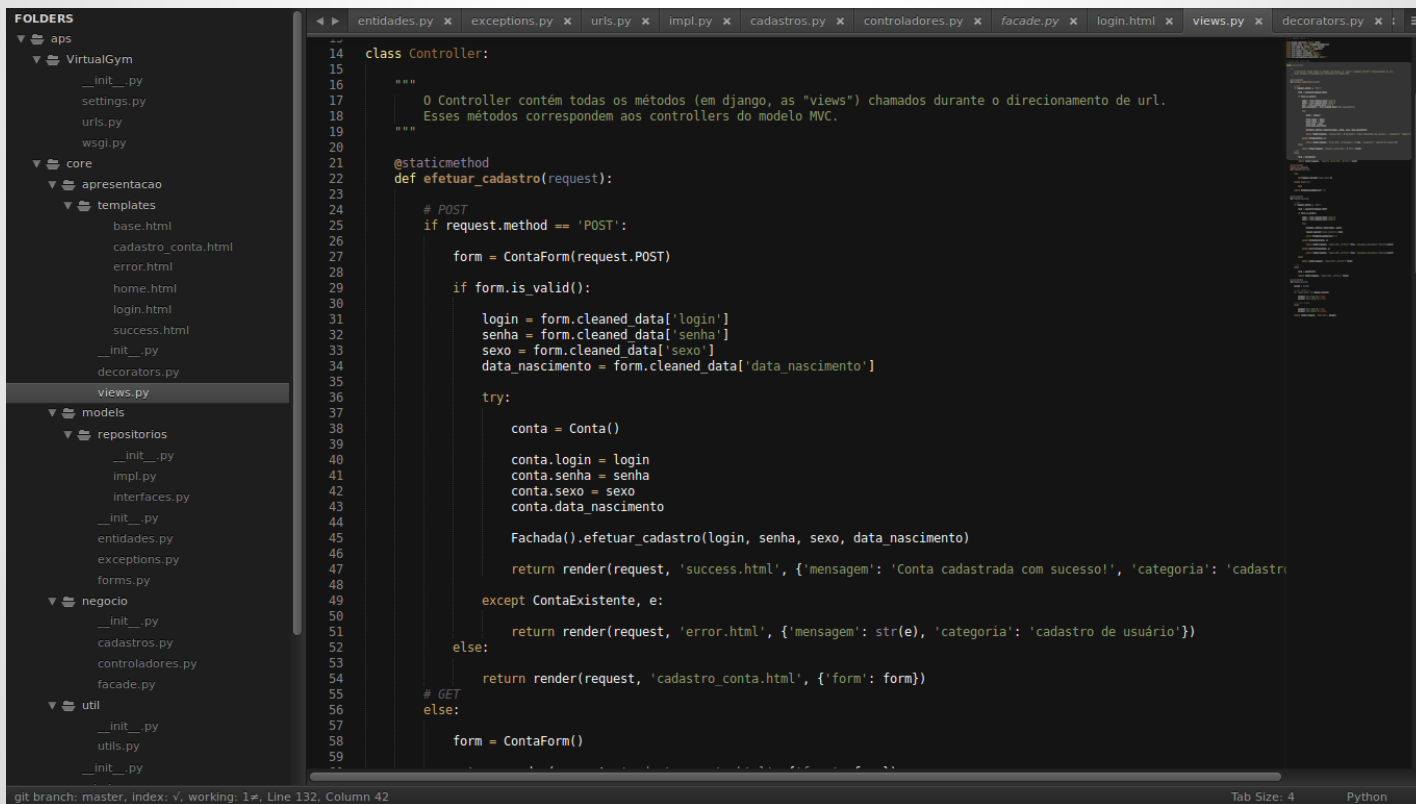


The image shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like 'core', 'templates', 'models', 'negocio', and 'util'. The code editor shows the following Python code:

```
1 # -*- coding: utf-8 -*-
2
3 from controladores import *
4 from core.util.utils import Singleton
5
6 class Fachada():
7
8     __metaclass__ = Singleton
9
10    def __init__(self):
11
12        self.controlador_cadastro = ControladorCadastro()
13        self.controlador_login = ControladorLogin()
14        self.controlador_treino = ControladorTreino()
15
16    # métodos
17
18    @raises: ContaExistente
19    def efetuar_cadastro(self, login, senha, sexo, data_nascimento):
20
21        self.controlador_cadastro.efetuar_cadastro(login, senha, sexo, data_nascimento)
22
23    def efetuar_login(self, login, senha):
24
25        return self.controlador_login.efetuar_login(login, senha)
```

At the bottom of the editor, the status bar shows: "git branch: master, index: ✓, working: 1#, Line 1, Column 1" on the left, "Tab Size: 4" in the middle, and "Python" on the right.

Código - CRUD (Cadastrar Conta)

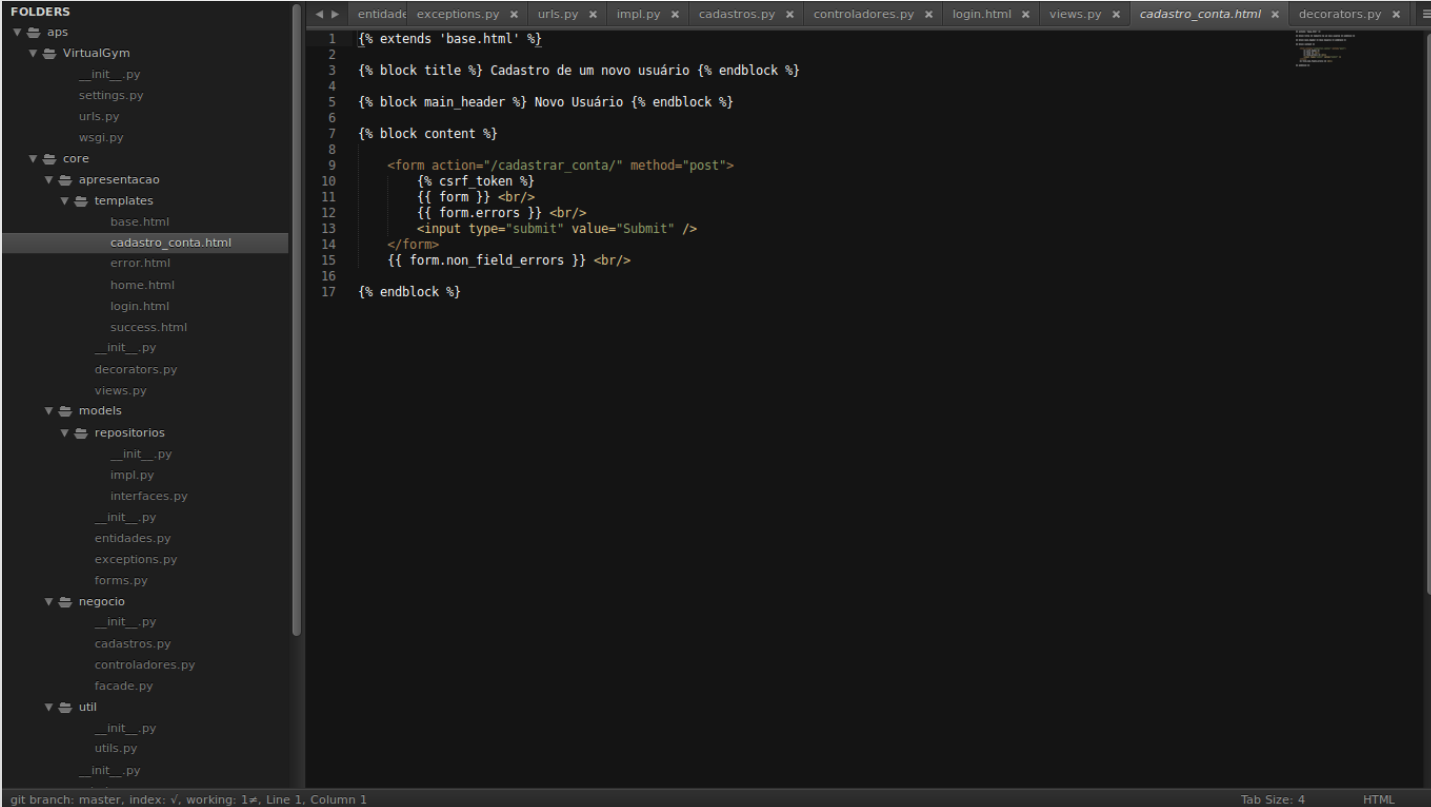


```
14 class Controller:
15
16     """
17     O Controller contém todos os métodos (em django, as "views") chamados durante o direcionamento de url.
18     Esses métodos correspondem aos controllers do modelo MVC.
19     """
20
21     @staticmethod
22     def efetuar_cadastro(request):
23
24         # POST
25         if request.method == 'POST':
26
27             form = ContaForm(request.POST)
28
29             if form.is_valid():
30
31                 login = form.cleaned_data['login']
32                 senha = form.cleaned_data['senha']
33                 sexo = form.cleaned_data['sexo']
34                 data_nascimento = form.cleaned_data['data_nascimento']
35
36                 try:
37
38                     conta = Conta()
39
40                     conta.login = login
41                     conta.senha = senha
42                     conta.sexo = sexo
43                     conta.data_nascimento
44
45                     Fachada().efetuar_cadastro(login, senha, sexo, data_nascimento)
46
47                     return render(request, 'success.html', {'mensagem': 'Conta cadastrada com sucesso!', 'categoria': 'cadastro'})
48
49                 except ContaExistente, e:
50
51                     return render(request, 'error.html', {'mensagem': str(e), 'categoria': 'cadastro de usuário'})
52             else:
53
54                 return render(request, 'cadastro_conta.html', {'form': form})
55
56         # GET
57         else:
58
59             form = ContaForm()
```

git branch: master, index: v, working: 1*, Line 132, Column 42

Tab Size: 4 Python

Código - CRUD (Cadastrar Conta)



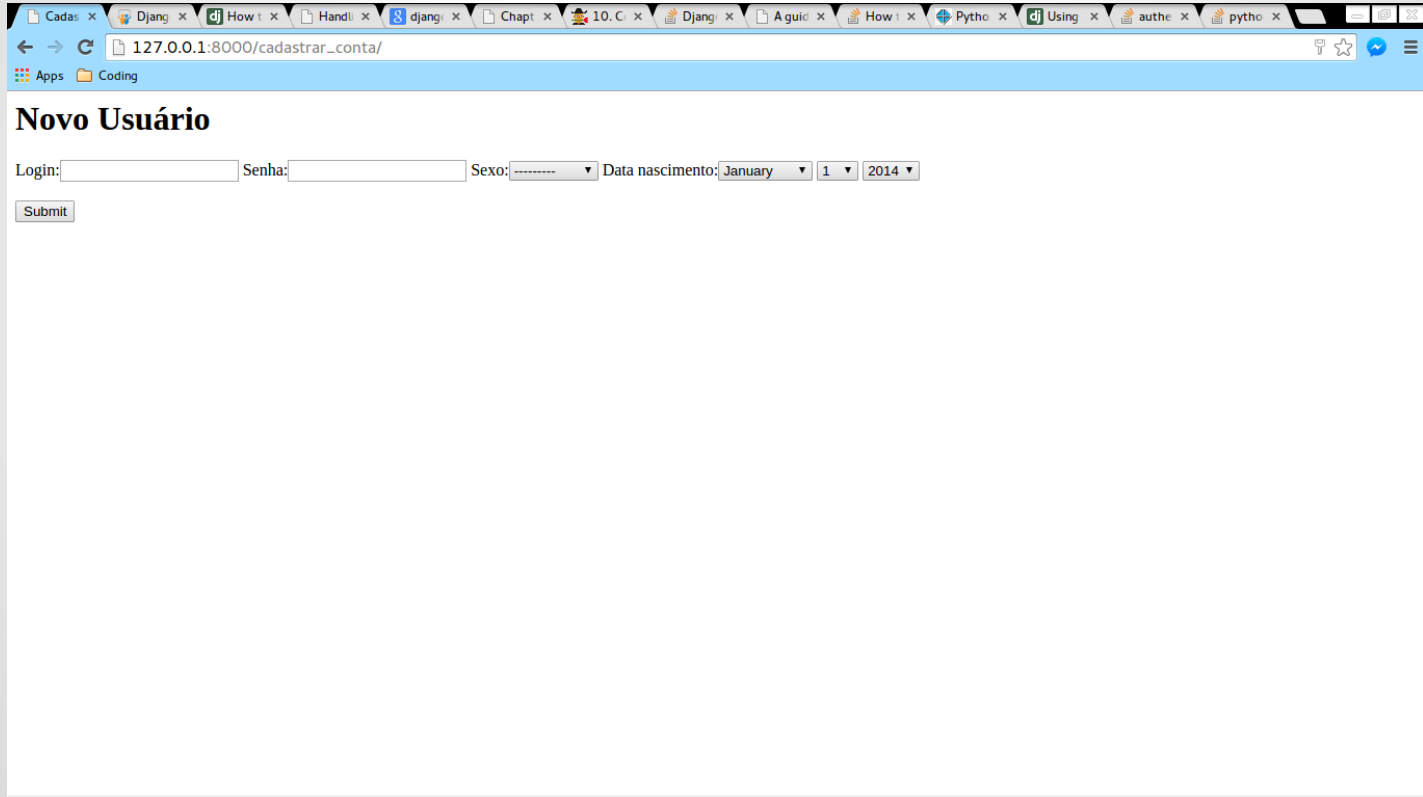
The image shows a code editor window with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with folders like 'aps', 'core', 'models', and 'negocio'. The code editor displays the content of 'cadastrar_conta.html'.

```
1 {% extends 'base.html' %}
2
3 {% block title %} Cadastro de um novo usuário {% endblock %}
4
5 {% block main_header %} Novo Usuário {% endblock %}
6
7 {% block content %}
8
9     <form action="/cadastrar_conta/" method="post">
10         {% csrf_token %}
11         {{ form }} <br/>
12         {{ form.errors }} <br/>
13         <input type="submit" value="Submit" />
14     </form>
15     {{ form.non_field_errors }} <br/>
16
17 {% endblock %}
```

git branch: master, index: v, working: 1*, Line 1, Column 1

Tab Size: 4 HTML

Código - CRUD (Cadastrar Conta)



The screenshot shows a web browser window with the address bar displaying `127.0.0.1:8000/cadastrar_conta/`. The browser's tab bar contains several open tabs, including 'Cadas', 'Djang', 'How t', 'Handl', 'djang', 'Chapt', '10. C', 'Djang', 'A guid', 'How t', 'Pytho', 'Using', 'authe', and 'pytho'. The browser's address bar also shows navigation icons and a search icon. The main content area of the browser displays a registration form titled 'Novo Usuário'. The form includes the following fields and controls:

- Novo Usuário** (Section Header)
- Login:**
- Senha:**
- Sexo:** (dropdown menu)
- Data nascimento:** (dropdown menu), (dropdown menu), (dropdown menu)
-

Código - Padrões de Projeto

- Singleton

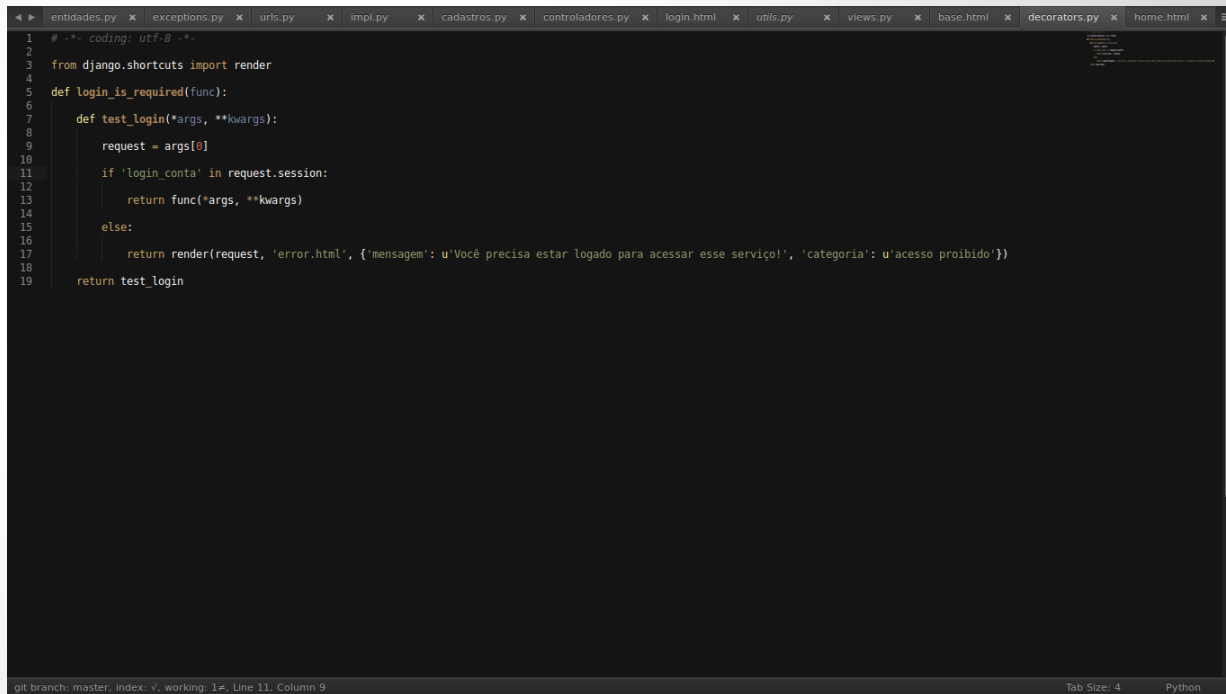
```
entidades.py x exceptions.py x urls.py x impl.py x cadastros.py x controladores.py x login.html x utils.py x views.py x base.html x decorators.py x home.html x
1  #-*- coding: utf-8 -*-
2
3  class Singleton(type):
4
5      # the instances of this class
6      _instances = {}
7
8      def __call__(cls, *args, **kwargs):
9
10         # if this class has no instances
11         if cls not in cls._instances:
12
13             # create an instance
14             cls._instances[cls] = super(Singleton, cls).__call__(*args, **kwargs)
15
16         # return the single instance for this class
17         return cls._instances[cls]
```

git branch: master, index: v, working: 1#, Line 17, Column 35

Tab Size: 4 Python

Código - Padrões de Projeto

- Decorator



```
1  #-*- coding: utf-8 -*-
2
3  from django.shortcuts import render
4
5  def login_is_required(func):
6
7      def test_login(*args, **kwargs):
8
9          request = args[0]
10
11          if 'login_conta' in request.session:
12              return func(*args, **kwargs)
13
14          else:
15              return render(request, 'error.html', {'mensagem': u'Você precisa estar logado para acessar esse serviço!', 'categoria': u'acesso proibido'})
16
17      return test_login
18
19
```

git branch: master, index: v, working: 1, Line 11, Column 9

Tab Size: 4 Python

Código - Padrões de Projeto

- Decorator

```
61
62
63 @staticmethod
64 @login_is_required
65 def logout(request):
66
67     try:
68         del(request.session['login_conta'])
69
70     except KeyError:
71         pass
72
73     return HttpResponseRedirect('/')
74
75
76
77 @staticmethod
78 def login(request):
79
80     # POST
81     if request.method == 'POST':
82
83         form = LoginForm(request.POST)
84
85         if form.is_valid():
86
87             login = form.cleaned_data['login']
88             senha = form.cleaned_data['senha']
89
90             try:
91                 Fachada().efetuar_login(login, senha)
92
93                 request.session['login_conta'] = login
94                 return HttpResponseRedirect('/')
95
96             except ContaInexistente, e:
97                 return render(request, 'login.html', {'form': form, 'mensagens_adicionais': [unicode(e)]})
98
99             except IncorrectLoginData, e:
100                 return render(request, 'login.html', {'form': form, 'mensagens_adicionais': [unicode(e)]})
101
102         else:
103
104
105
106
107
```

git branch: master, index: v, working: 1+, Line 132, Column 42

Tab Size: 4 Python