Virtual Gym

Análise e Projeto de Sistemas

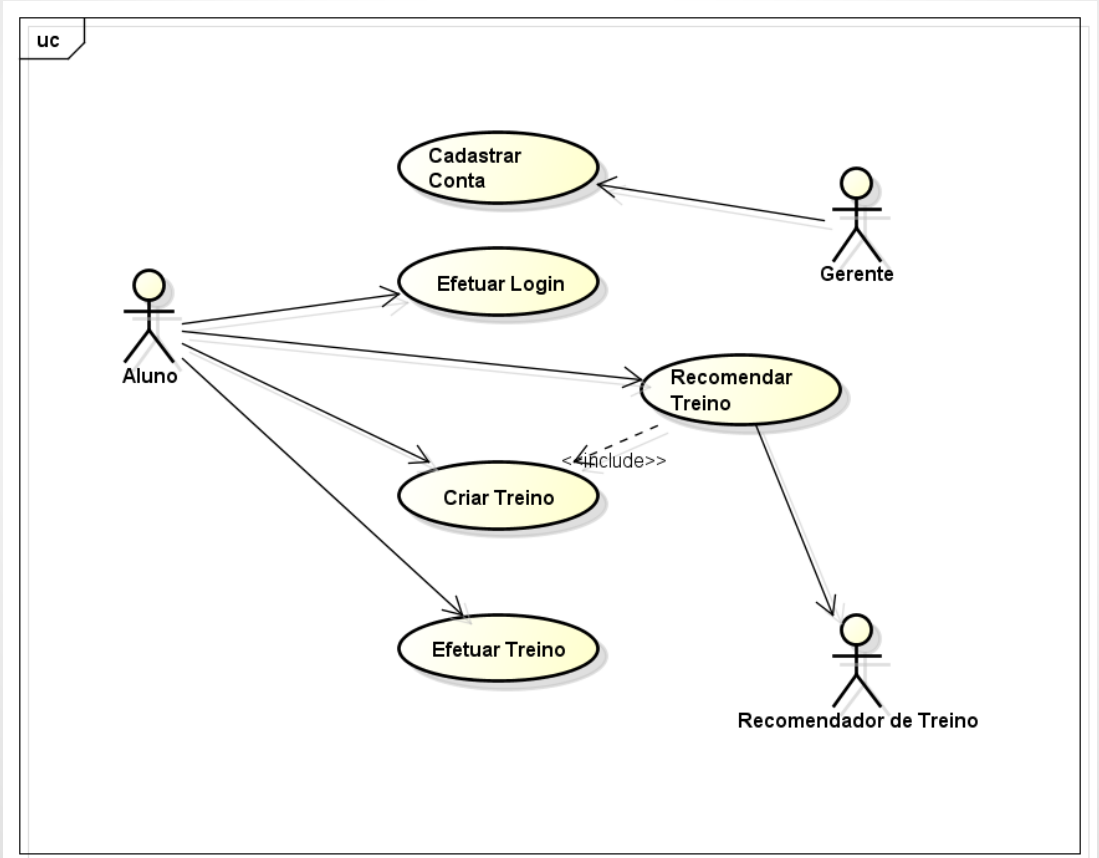
Roteiro

- 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 mesmo 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

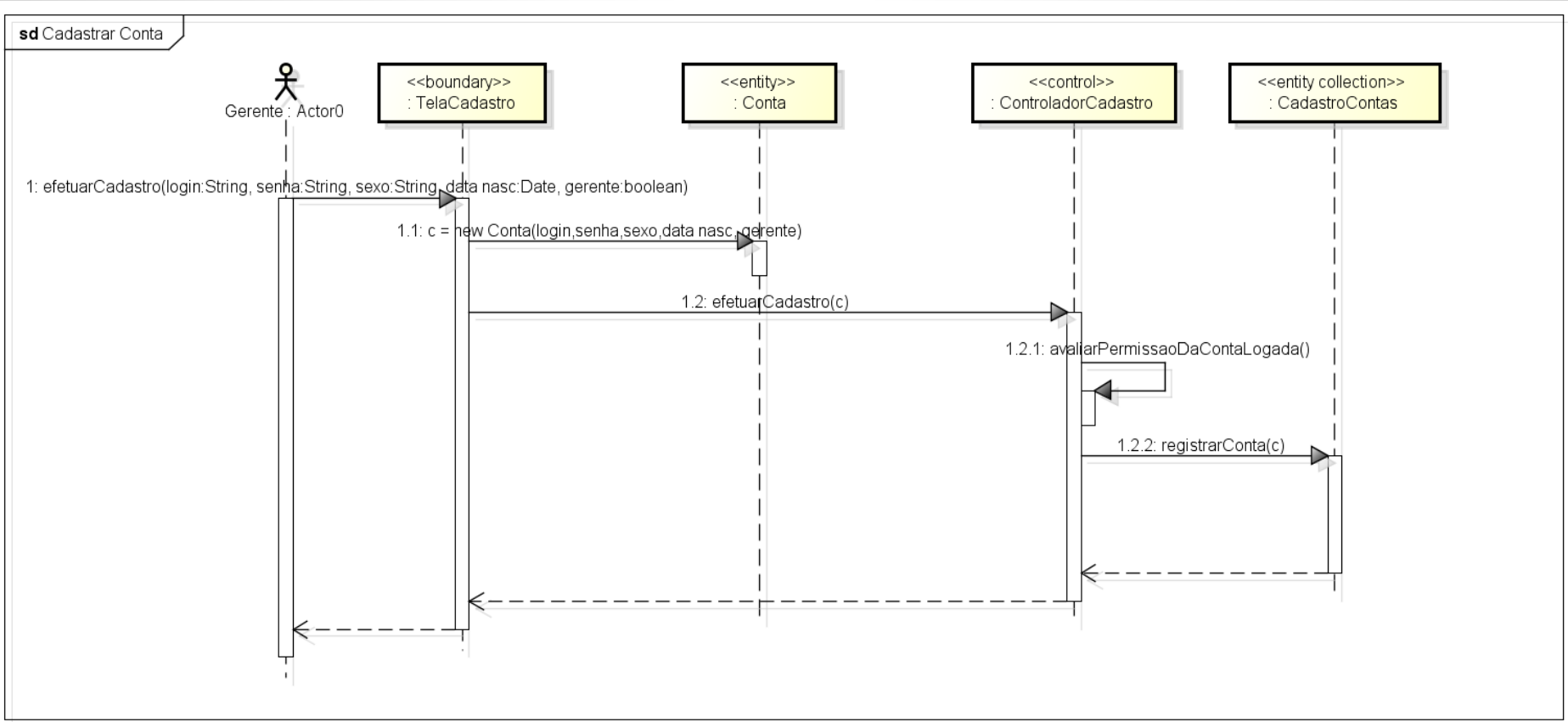
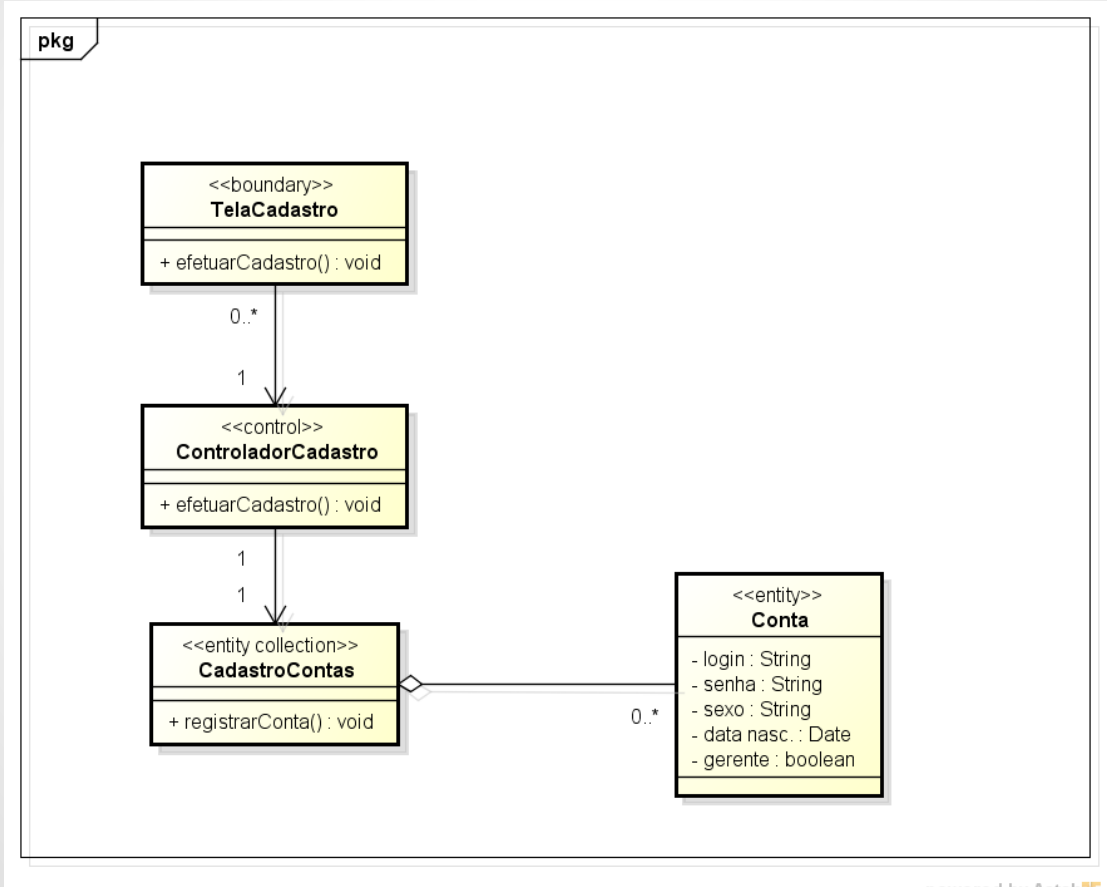


Diagrama de classe



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

1. O ator informa os campos:
   - 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

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

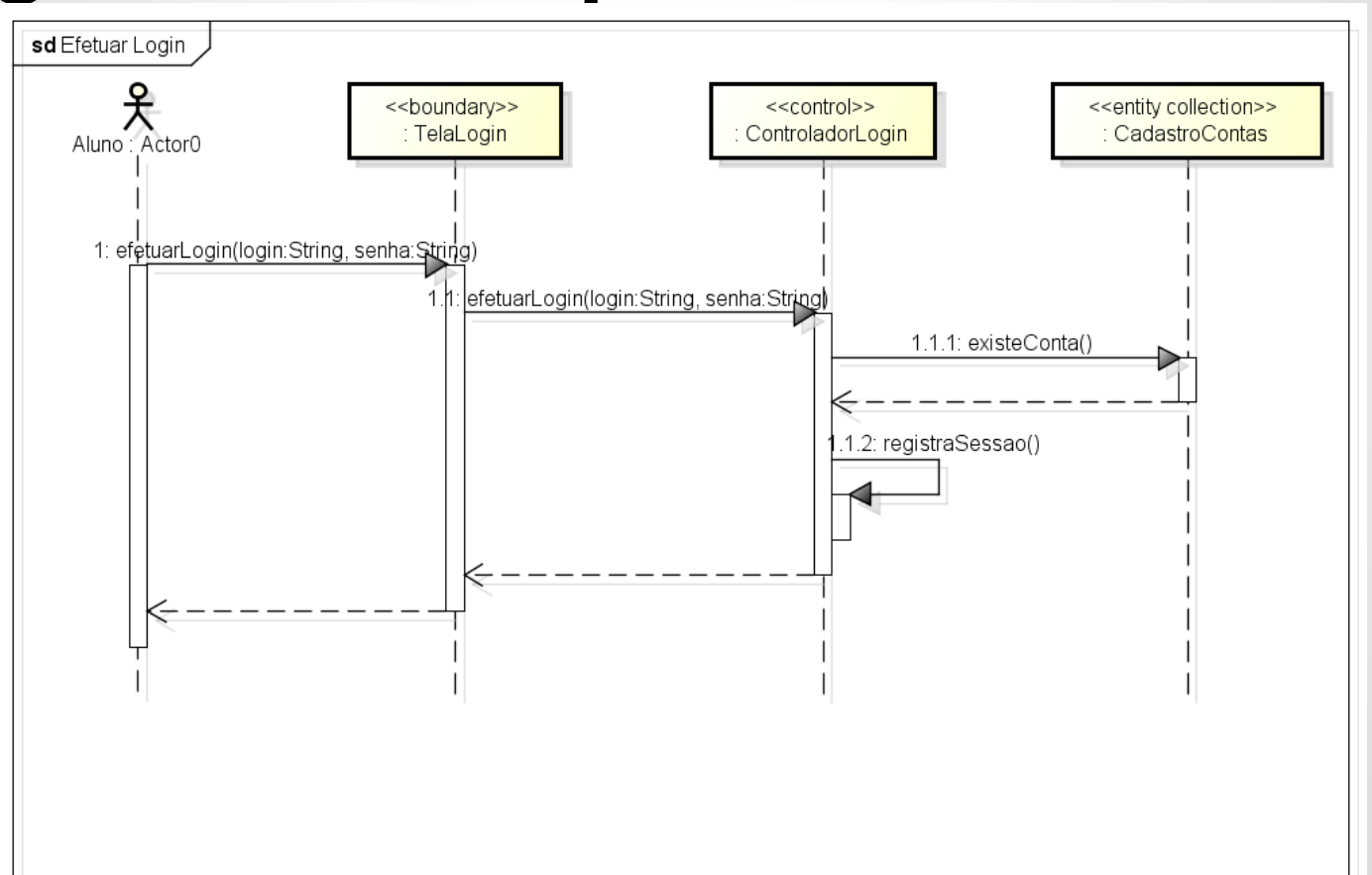
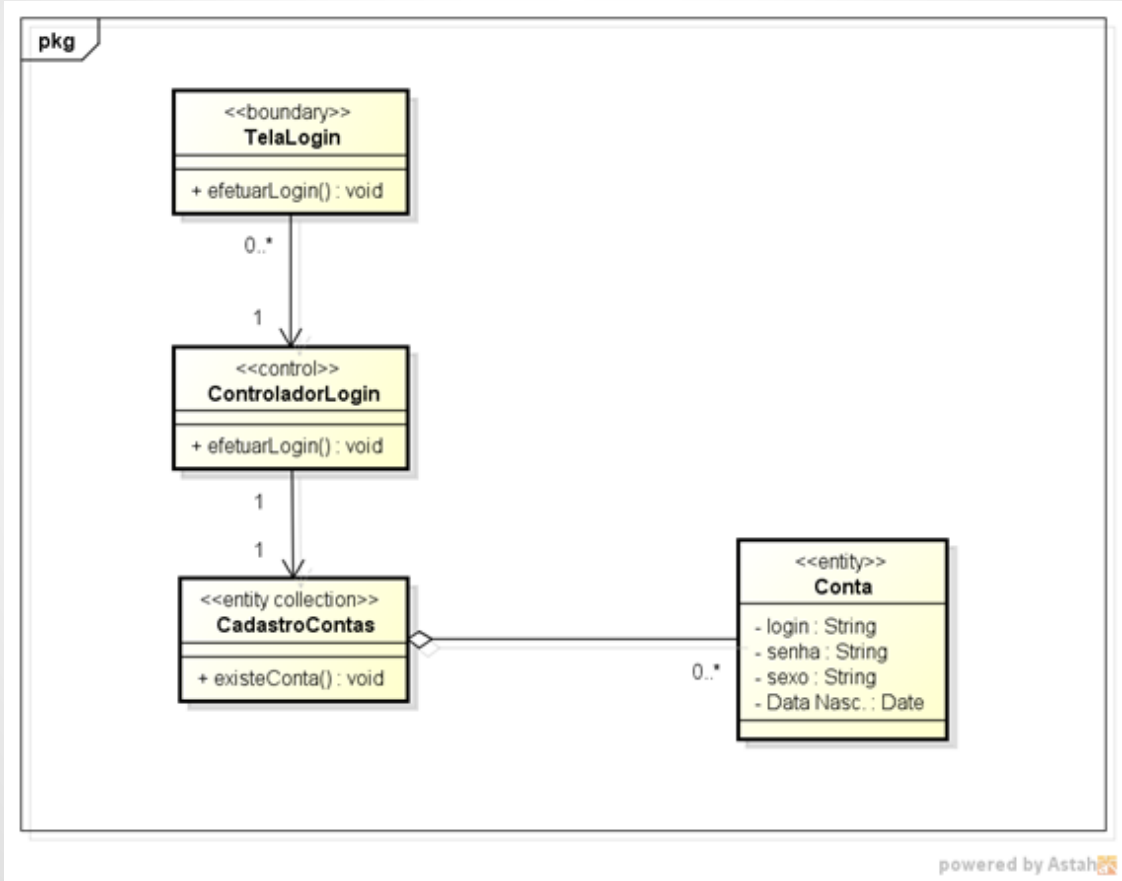


Diagrama de classe



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

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

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 sequência

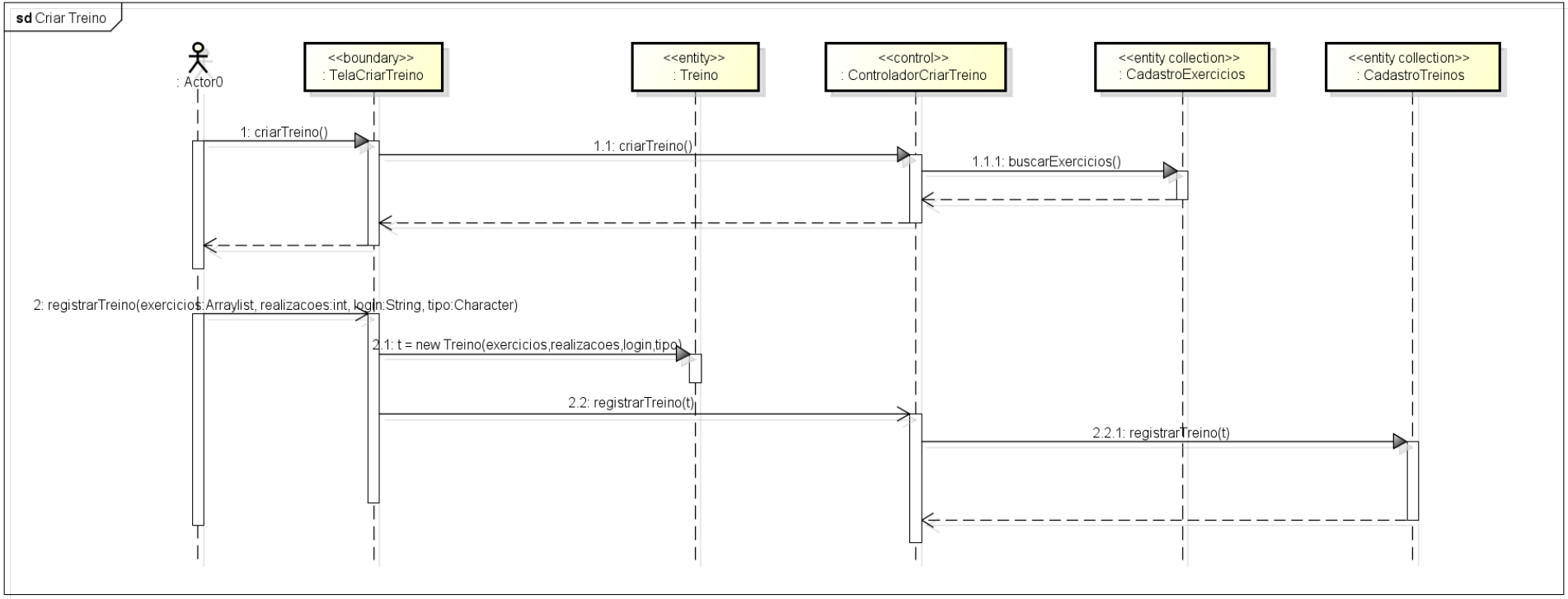
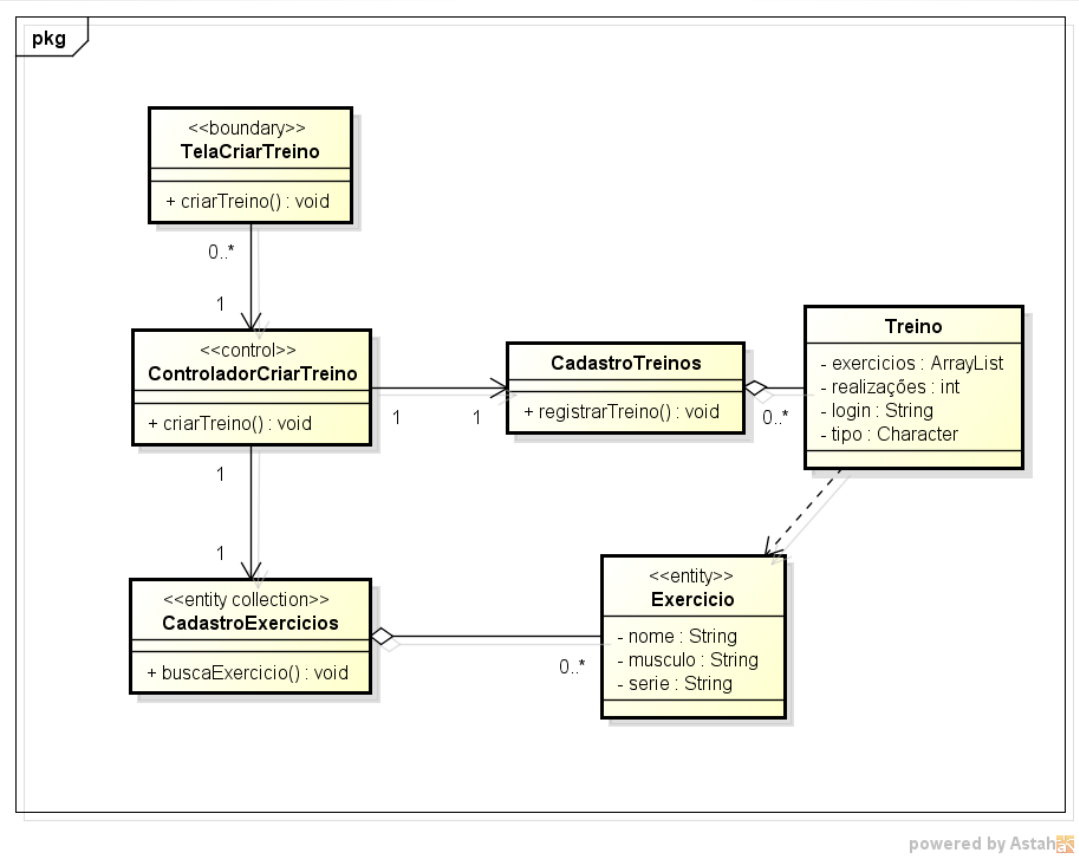


Diagrama de classe



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 exibe 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 exercícios. 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

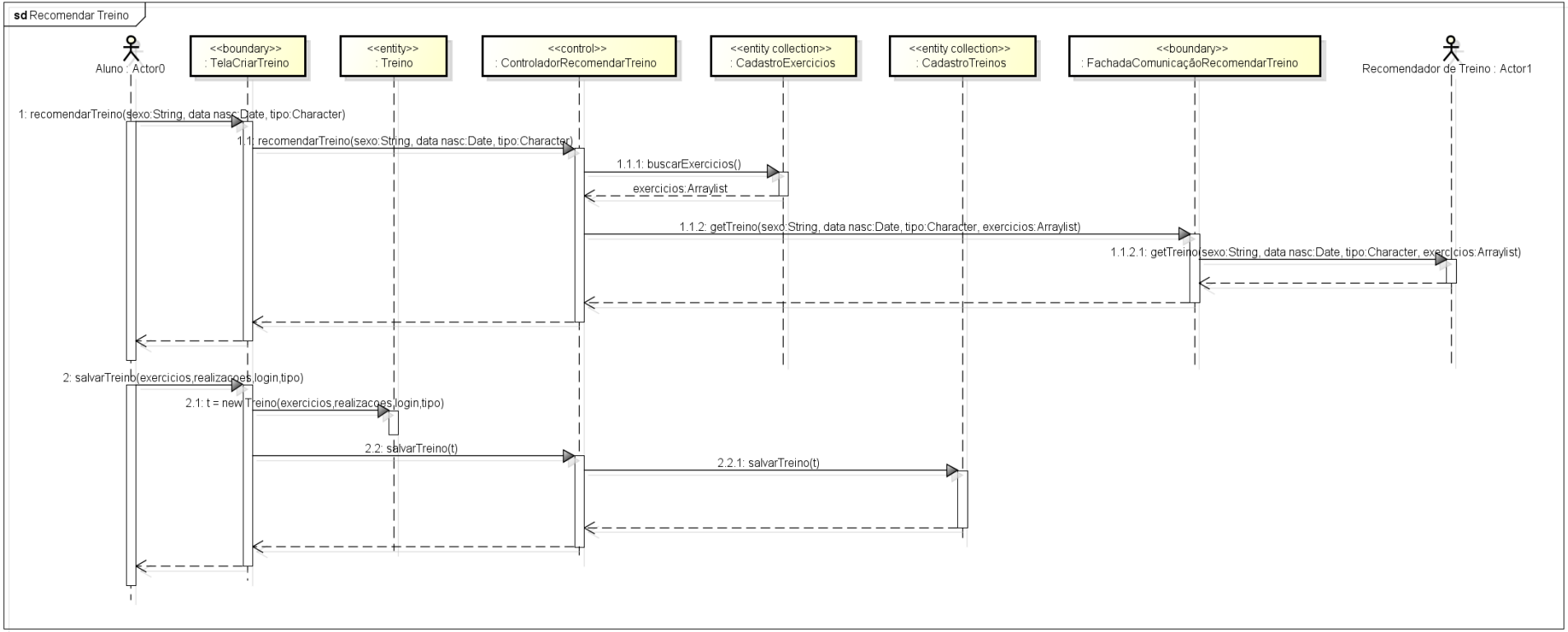
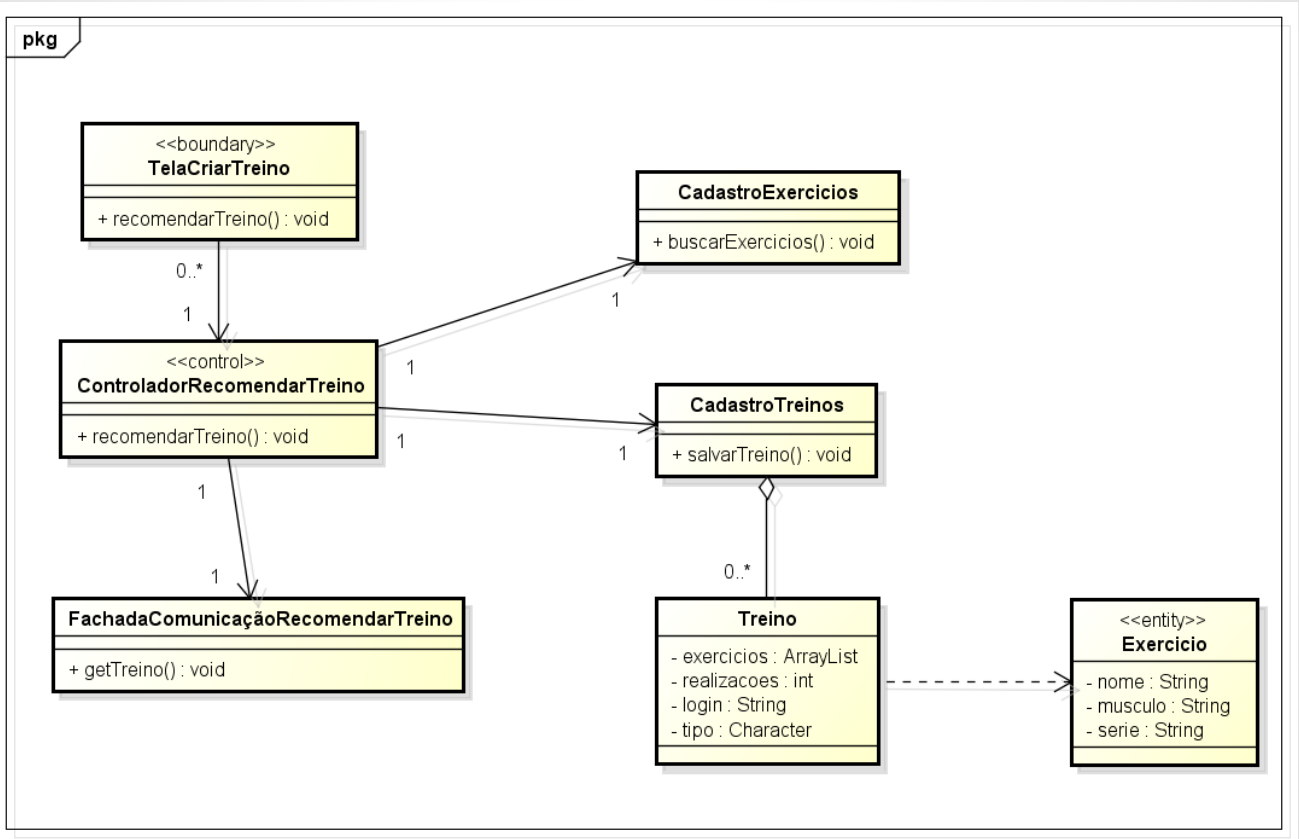


Diagrama de classe



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

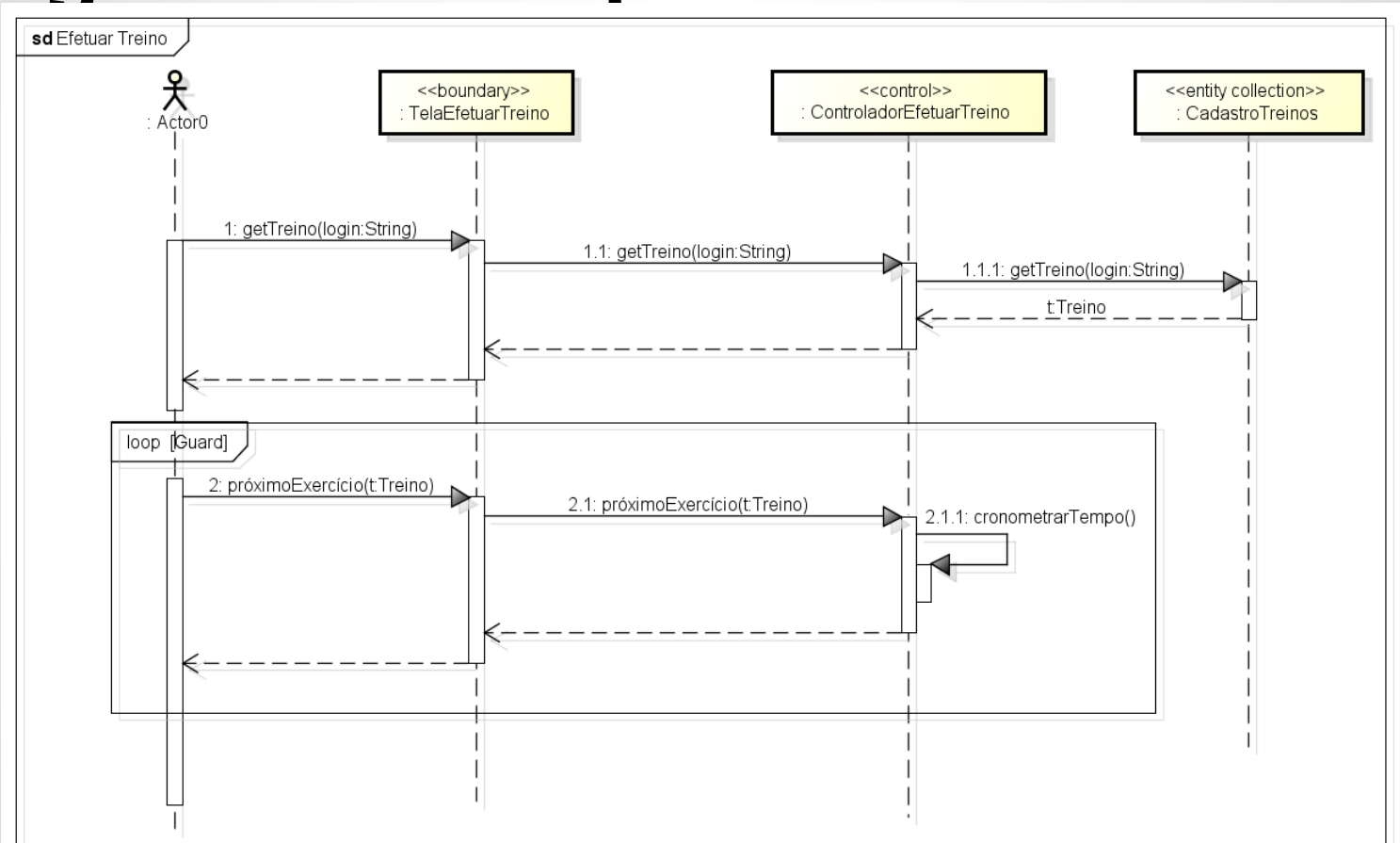
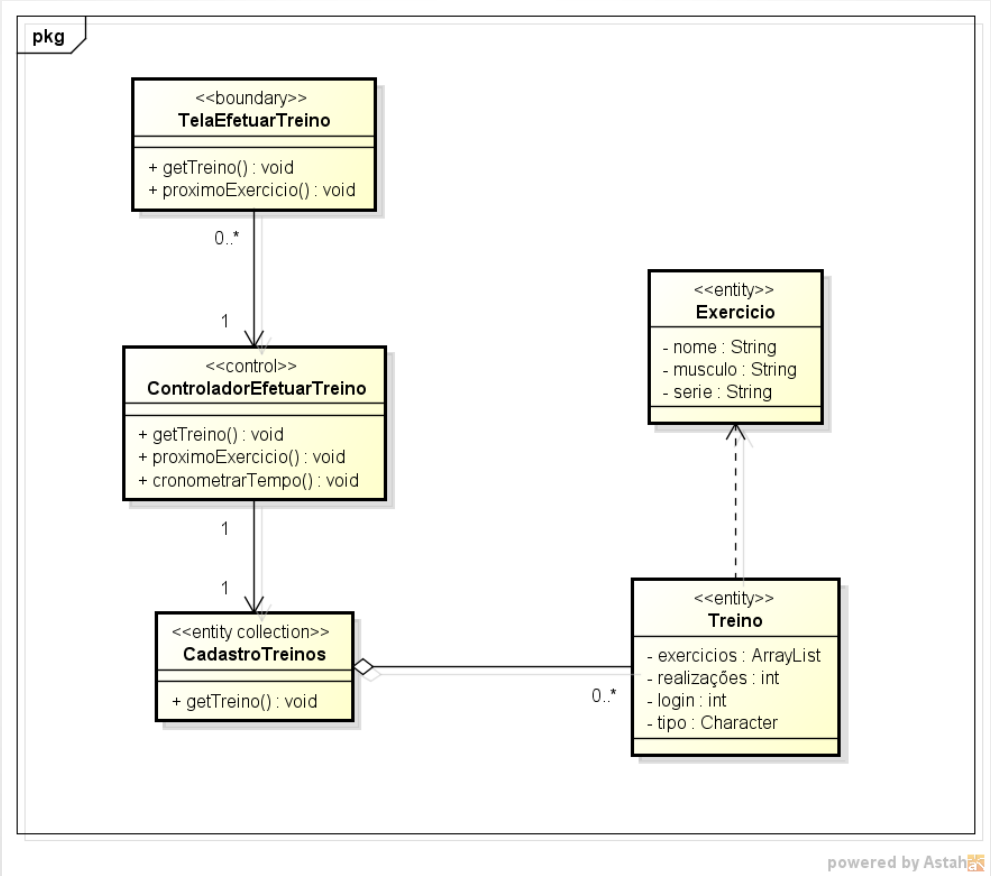


Diagrama de classe



Mapeamento de classes de análise em elementos de projeto

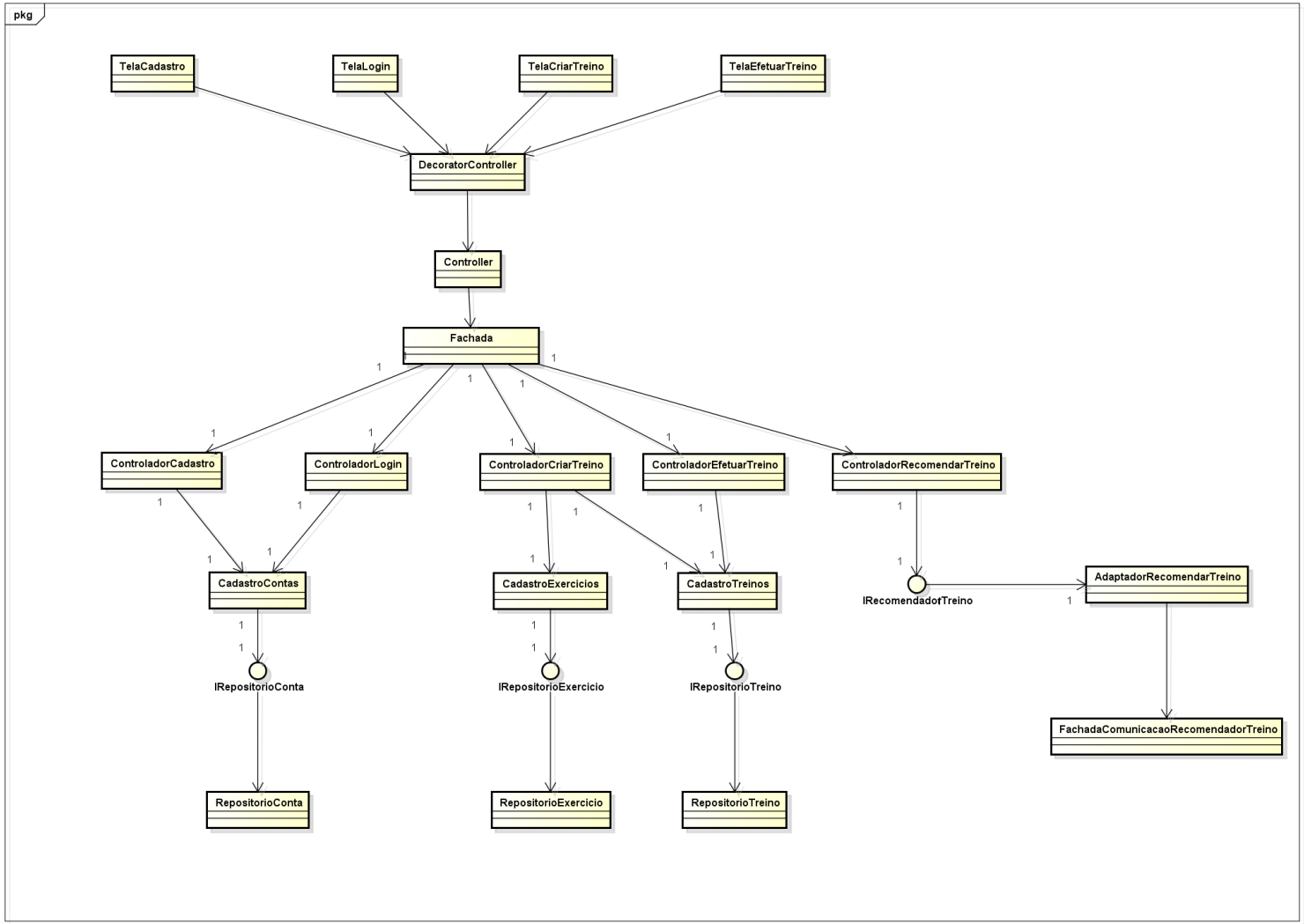
Classes de Análise	Elementos do Projeto
	Fachada
TelaCadastro	TelaCadastro
TelaLogin	TelaLogin
TelaCriarTreino	TelaCriarTreino
TelaEfetuarTreino	TelaEfetuarTreino
ControladorCadastro	ControladorCadastro
ControladorLogin	ControladorLogin
ControladorCriarTreino	ControladorCriarTreino
ControladorEfetuarTreino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

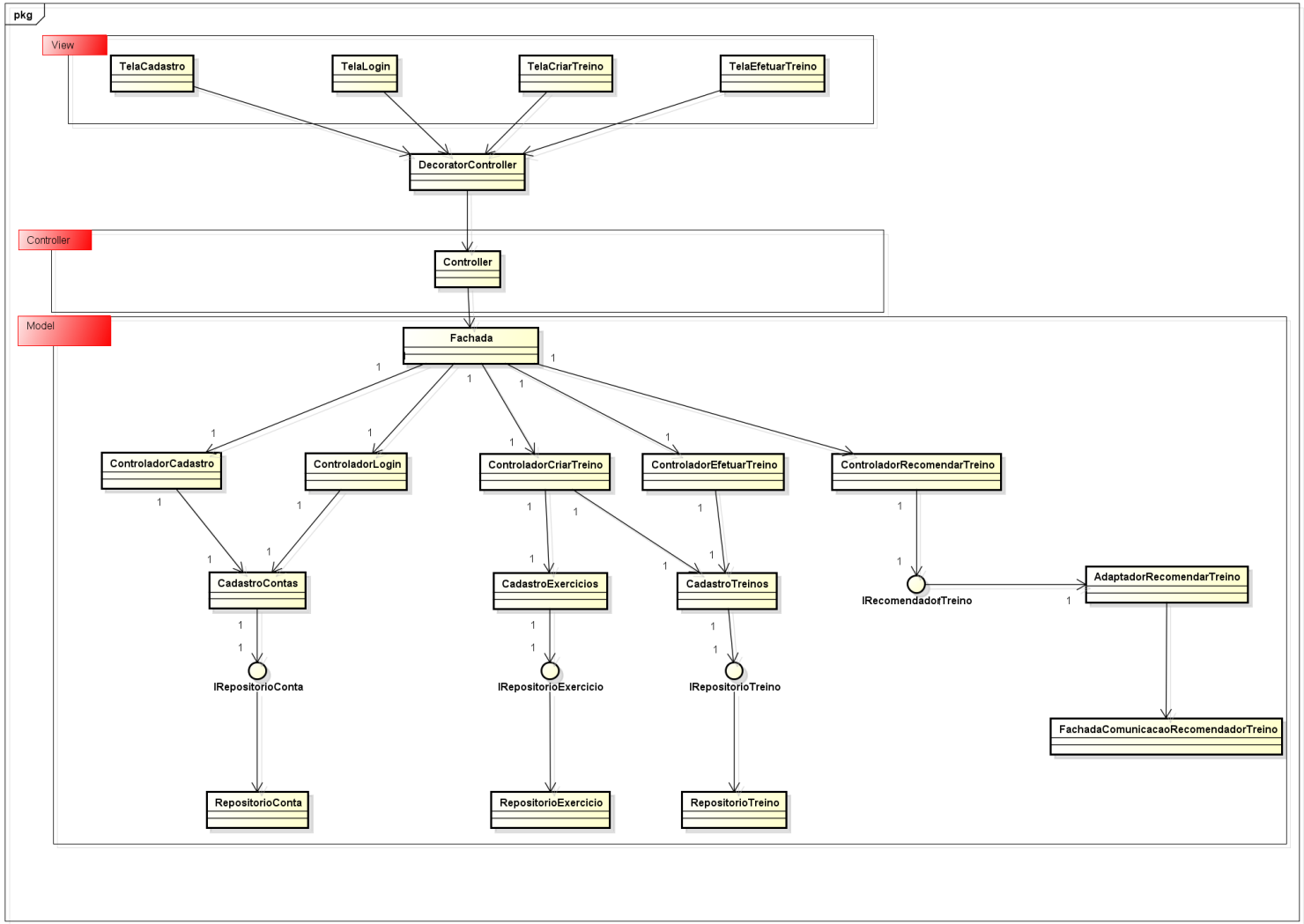
Mapeamento de classes de análise em elementos de projeto

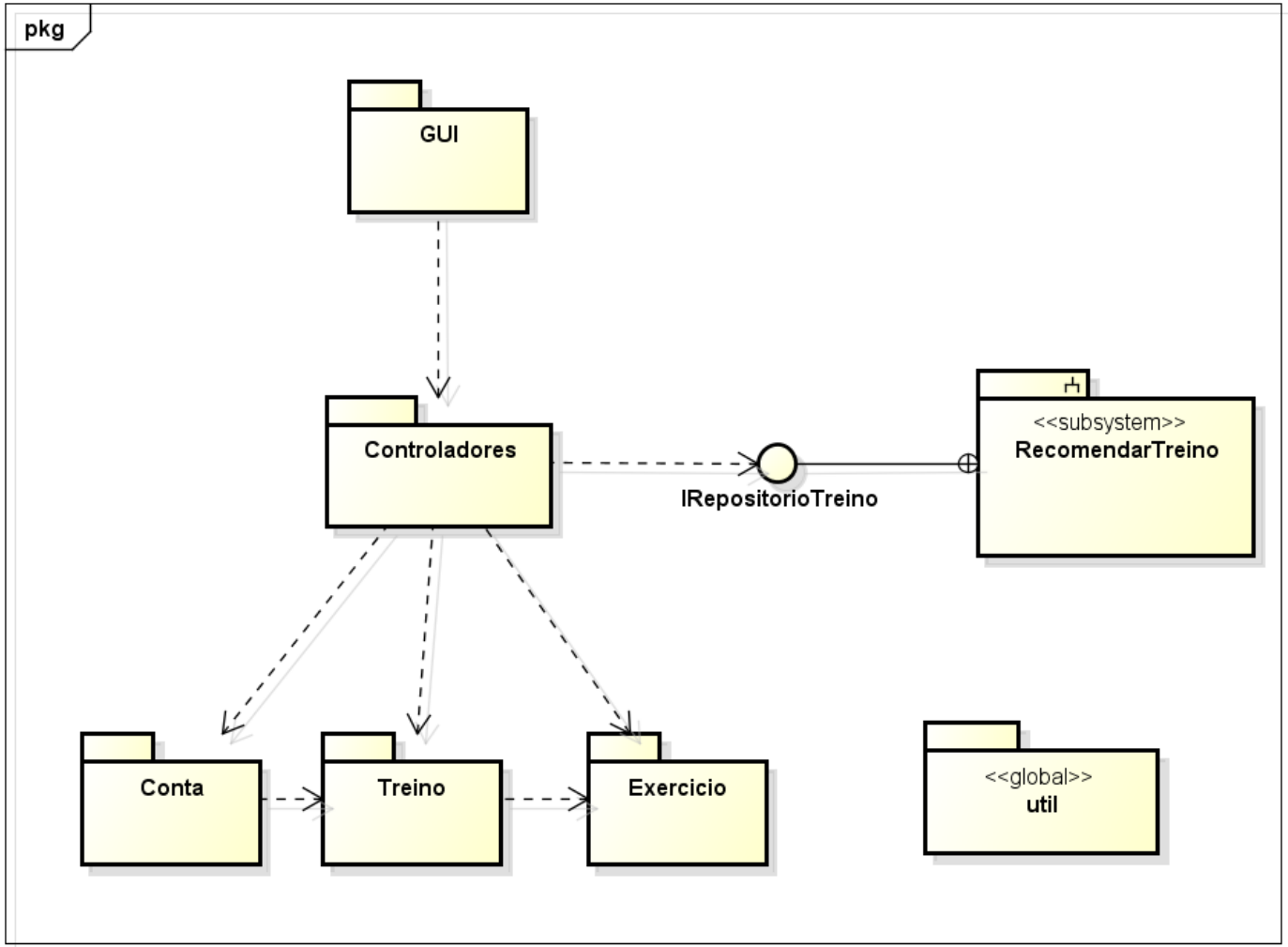
Classes de Análise	**Elementos do Projeto**
CadastroContas	CadastroContas IRepositorioConta RepositorioConta
CadastroExercicios	CadastroExercicios IRepositorioExercicio RepositorioExercicio
CadastroTreinos	CadastroTreinos IRepositorioTreino RepositorioTreino
Conta	Conta
Exercicio	Exercício
Treino	Treino

Arquitetura

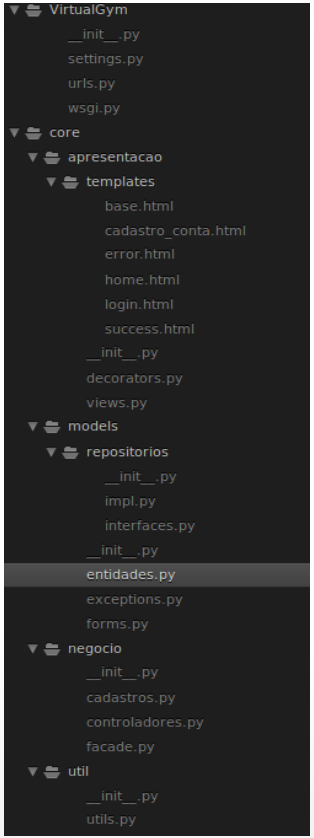
- Organizada em pacotes
- Utiliza subsistema
- Aplicação de padrões de projeto
  - Façade
  - Adapter
  - Bridge
  - Decorator
  - Singleton


pkg
TelaCadastro
TelaLogin
TelaCriarTreino
TelaEfetuarTreino
DecoratorController
Controller
Fachada
ControladorCadastro
ControladorLogin
ControladorCriarTreino
ControladorEfetuarTreino
ControladorRecomendarTreino
CadastroContas
CadastroExercicios
CadastroTreinos
IRecomendadorTreino
AdaptadorRecomendarTreino
IRepositorioConta
IRepositorioExercicio
IRepositorioTreino
FachadaComunicacaoRecomendadorTreino
RepositorioConta
RepositorioExercicio
RepositorioTreino
1


pkg
View
TelaCadastro
TelaLogin
TelaCriarTreino
TelaEfetuarTreino
DecoratorController
Controller
Controller
Model
Fachada
ControladorCadastro
ControladorLogin
ControladorCriarTreino
ControladorEfetuarTreino
ControladorRecomendarTreino
CadastroContas
CadastroExercicios
CadastroTreinos
IRecomendadorTreino
AdaptadorRecomendarTreino
IRepositorioConta
IRepositorioExercicio
IRepositorioTreino
FachadaComunicacaoRecomendadorTreino
RepositorioConta
RepositorioExercicio
RepositorioTreino



Código - Organização



Código - MVC (Models)

entidades.py | exceptions.py | urls.py | impl.py | cadastros.py | controladores.py | login.html | views.py | decorators.py | home.html

```python
class Conta (models.Model):

    """
        Classe que representa a conta de um usuário do sistema
    """

    # contantes de classe
    choices_sex = (
                    ('M', 'Masculino'),
                    ('F', 'Feminino'),
                  )

    # atributos
    login = models.CharField(max_length = 255, unique = True)
    senha = models.CharField(max_length = 16)
    sexo = models.CharField(max_length = 2, choices = choices_sex)
    data_nascimento = models.DateField()
    gerente = models.BooleanField()

    class Meta:
        app_label = 'core'

    def __unicode__ (self):

        return "[login='%s', sex='%s']" % (self.login, self.sexo)

class Exercicio (models.Model):

    """
        Classe que representa um exercício físico
    """

    # atributos
    nome = models.CharField(max_length = 255)
    musculo = models.CharField(max_length = 255)  # add fixed choices
    serie = models.CharField(max_length = 255)

    class Meta:
        app_label = 'core'

    def __unicode__ (self):

        return "[nome='%s', músculo='%s', série='%s']" % (self.nome, self.musculo, self.serie)

class Treino (models.Model):

    """
```

25, Column 36 — Tab Size: 4 — Python

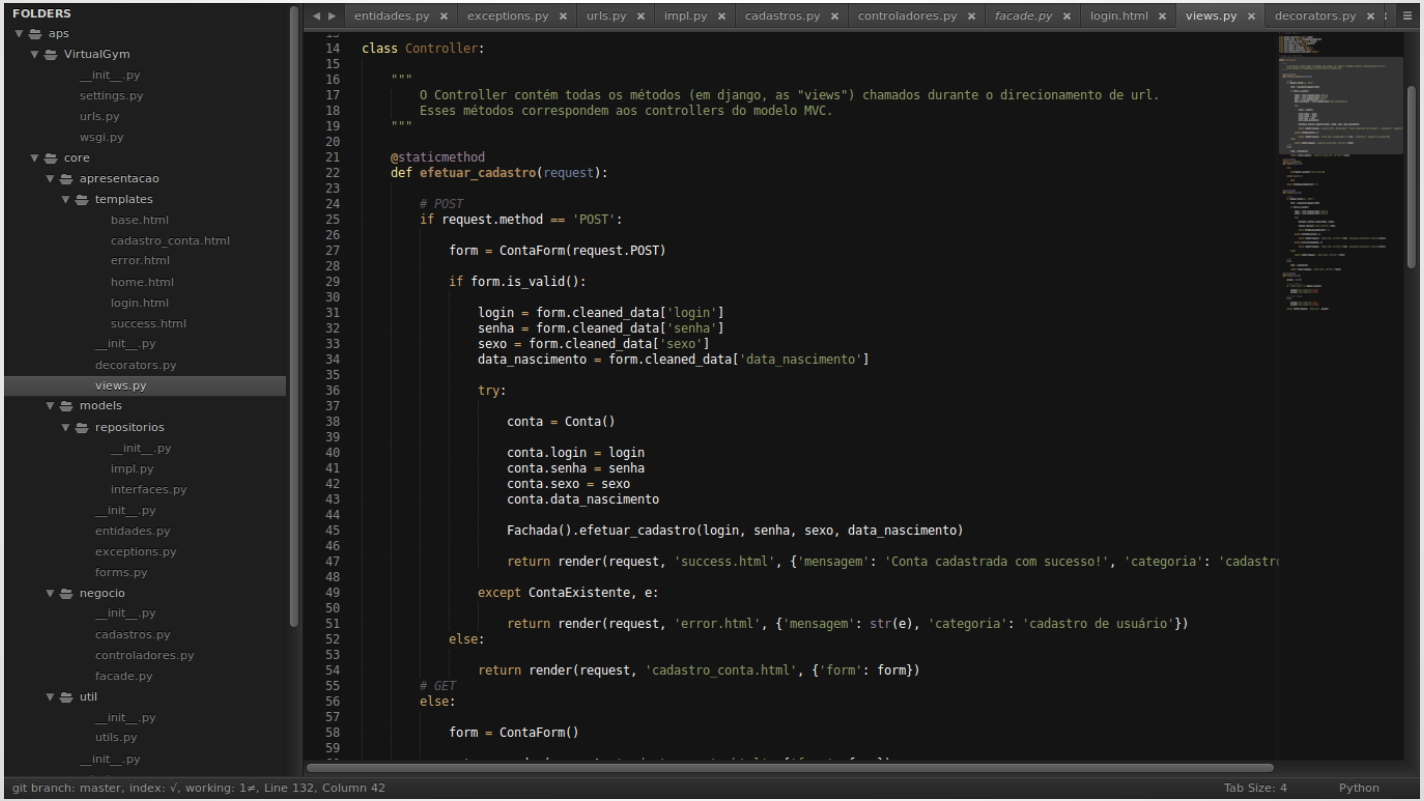
entidades.py | exceptions.py | urls.py | impl.py | cadastros.py | controladores.py | login.html | views.py | decorators.py | home.html

```python
class Exercicio (models.Model):

    """
        Classe que representa um exercício físico
    """

    # atributos
    nome = models.CharField(max_length = 255)
    musculo = models.CharField(max_length = 255)  # add fixed choices
    serie = models.CharField(max_length = 255)

    class Meta:
        app_label = 'core'

    def __unicode__ (self):

        return "[nome='%s', músculo='%s', série='%s']" % (self.nome, self.musculo, self.serie)

class Treino (models.Model):

    """
        Classe que representa um treino, composto de repetições de exercícios
    """

    # constantes de classe
    choices_tipo = (
                    ('1', '1'),
                    ('2', '2'),
                    ('3', '3'),
                   )

    # atributos
    exercicios = models.ManyToManyField('Exercicio')
    realizacoes = models.IntegerField()
    tipo = models.CharField(max_length = 1, choices = choices_tipo)
    conta = models.ForeignKey('Conta')

    class Meta:
        app_label = 'core'

    def __unicode__ (self):

        return "[exercicios='%s', realizações='%d']" % (self.exercicios, self.realizacoes)
```

venv

git branch: master, index: √, working: 1≠, Line 25, Column 36 — Tab Size: 4 — Python

Código - MVC (Controller)



Código - MVC (Views)

entidades.py | exceptions.py | urls.py | impl.py | cadastros.py | controladores.py | login.html | views.py | base.html | decorators.py | home.html

```
<html>

    <head>

        <title> {% block title %} {% endblock %} </title>

    </head>

    <body>

        <h1> {% block main_header %} {% endblock %} </h1>

        {% block content %}

        {% endblock %}

    </body>

</html>
```

git branch: master, index: √, working: 1≠, Line 1, Column 1

entidades.py | exceptions.py | urls.py | impl.py | cadastros.py | controladores.py | login.html | views.py | cadastro_conta.html | decorators.py | home.html

```
{% extends 'base.html' %}

{% block title %} Cadastro de um novo usuário {% endblock %}

{% block main_header %} Novo Usuário {% endblock %}

{% block content %}

    <form action="/cadastrar_conta/" method="post">
        {% csrf_token %}
        {{ form }} <br/>
        {{ form.errors }} <br/>
        <input type="submit" value="Submit" />
    </form>
    {{ form.non_field_errors }} <br/>

{% endblock %}
```

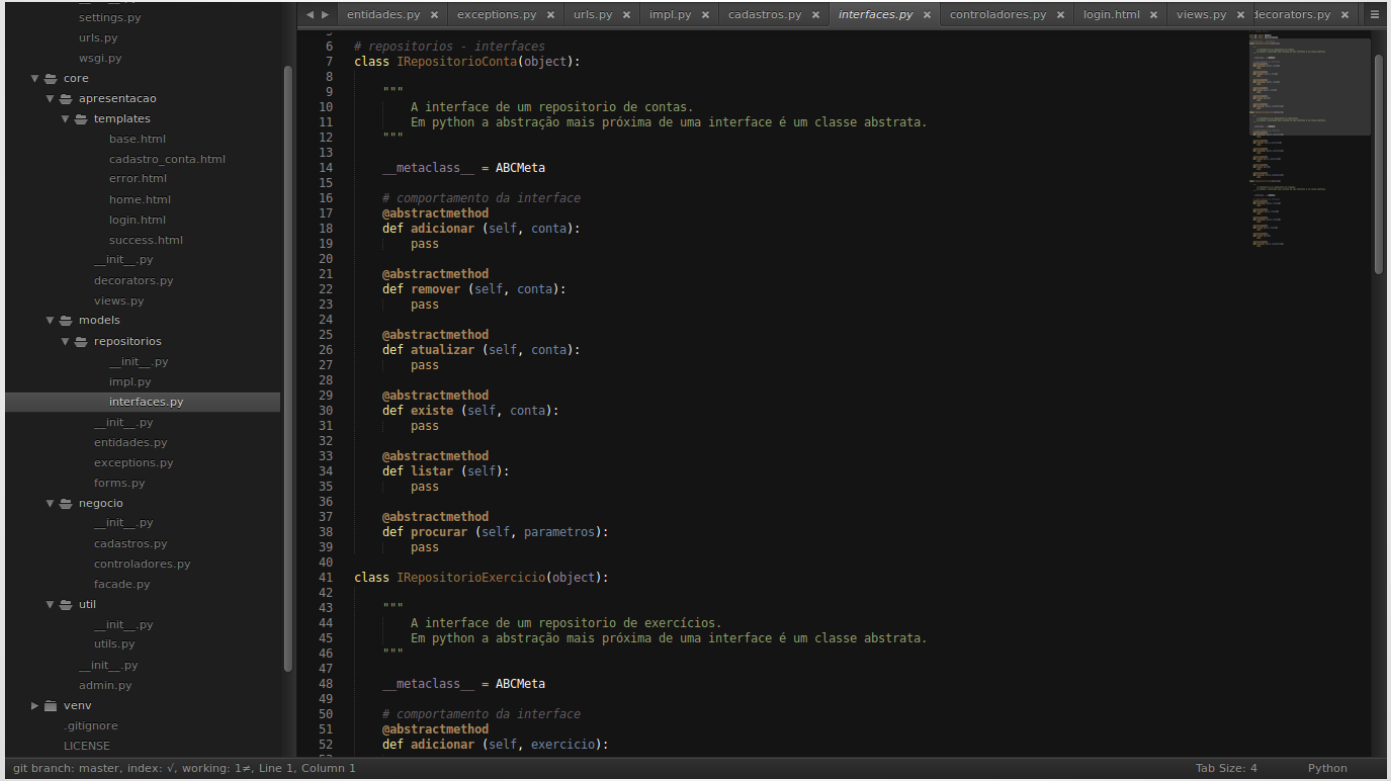
git branch: master, index: √, working: 1≠, Line 1, Column 1

entidades.py | exceptions.py | urls.py | impl.py | cadastros.py | controladores.py | login.html | views.py

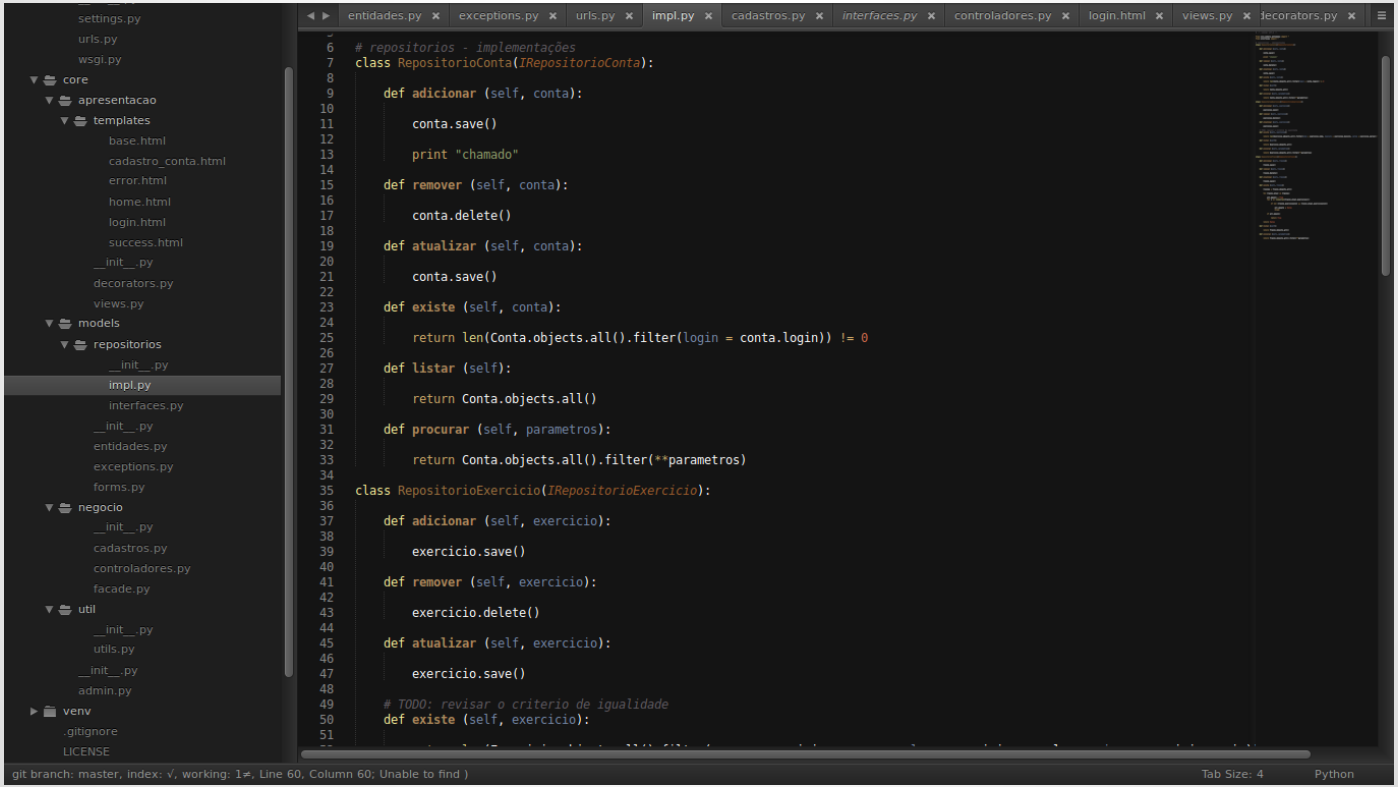
```
{% extends 'base.html' %}

{% block title %} Virtual Gym - Login {% endblock %}

{% block main_header %} Login {% endblock %}

{% block content %}

    <form action="/login/" method="post">
        {% csrf_token %}
        {{ form }} <br/>
        {{ form.errors }} <br/>
        <input type="submit" value="Submit" />
    </form>
    {{ form.non_field_errors }} <br/>

    <hr/>

    {% for message in mensagens_adicionais %}
        {{message}} <br/>
    {% endfor %}

{% endblock %}
```

git branch: master, index: √, working: 1≠, Line 14, Column 12

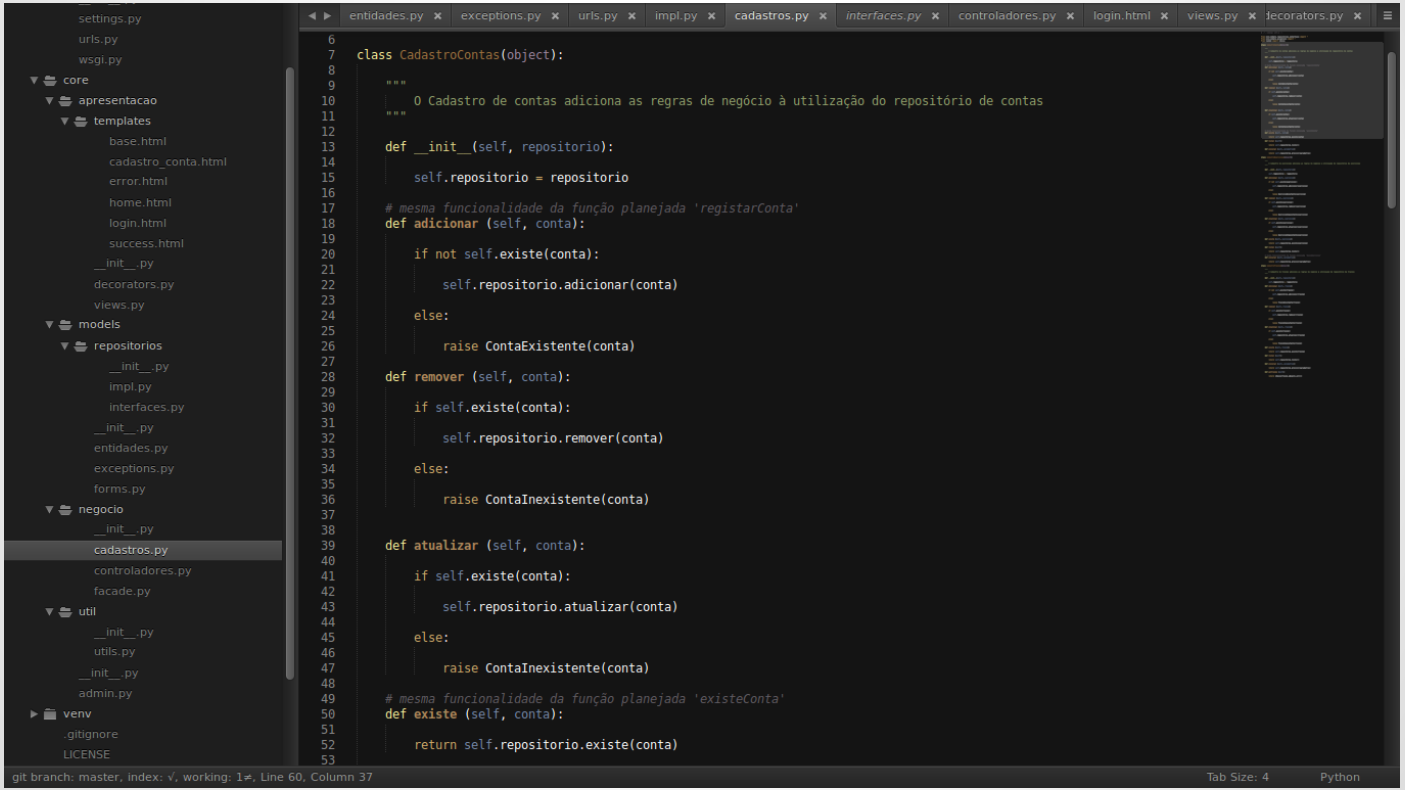
Código - CRUD (Cadastrar Conta)



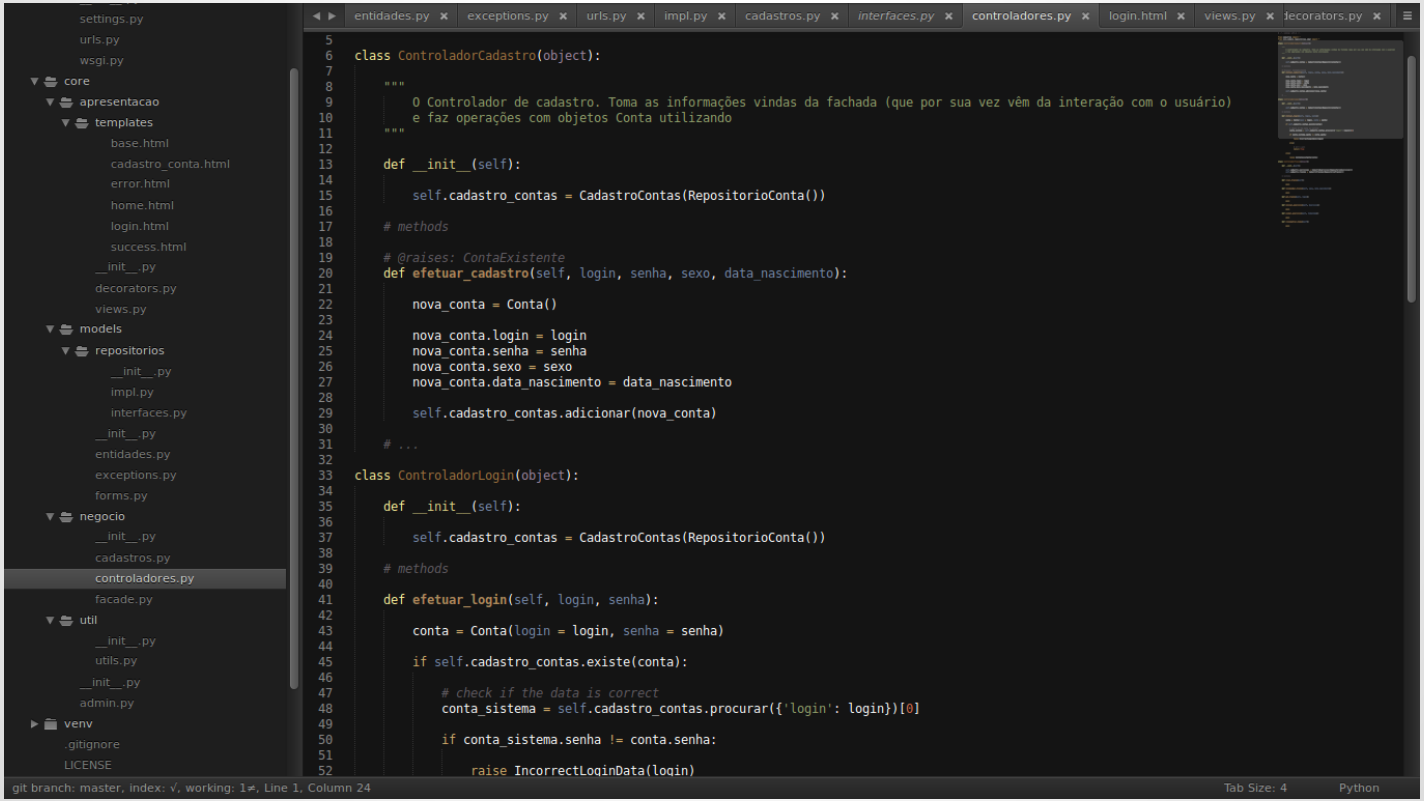
Código - CRUD (Cadastrar Conta)



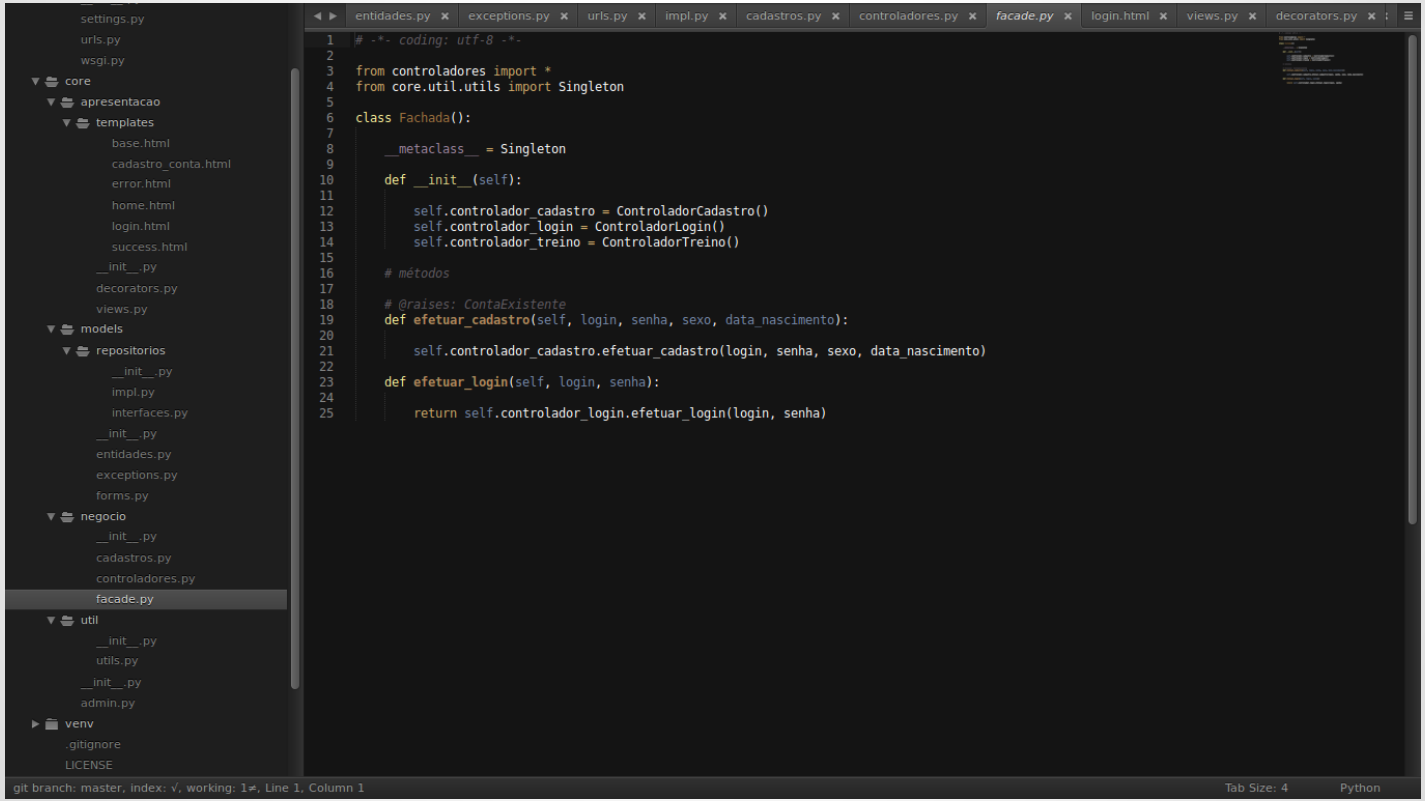
Código - CRUD (Cadastrar Conta)



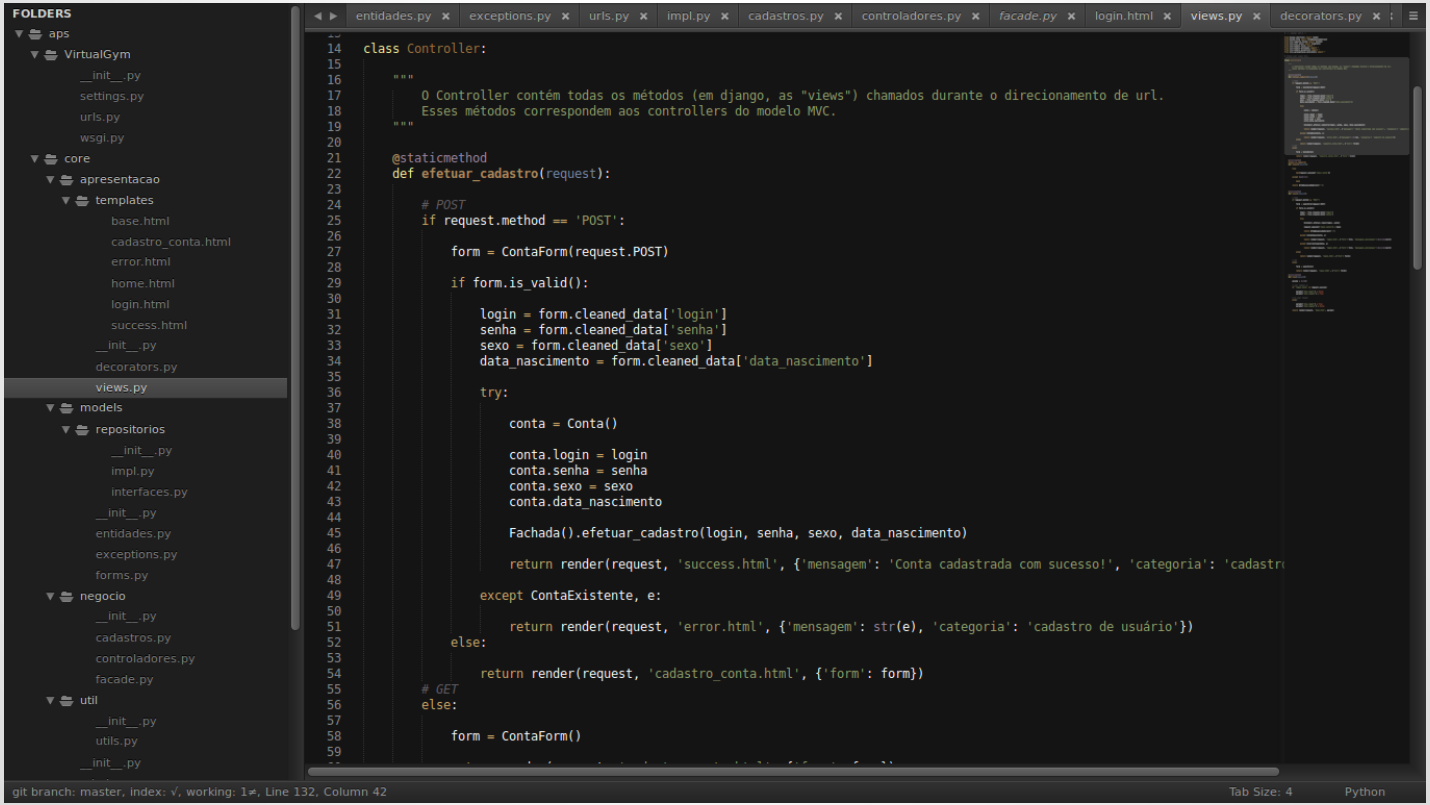
Código - CRUD (Cadastrar Conta)



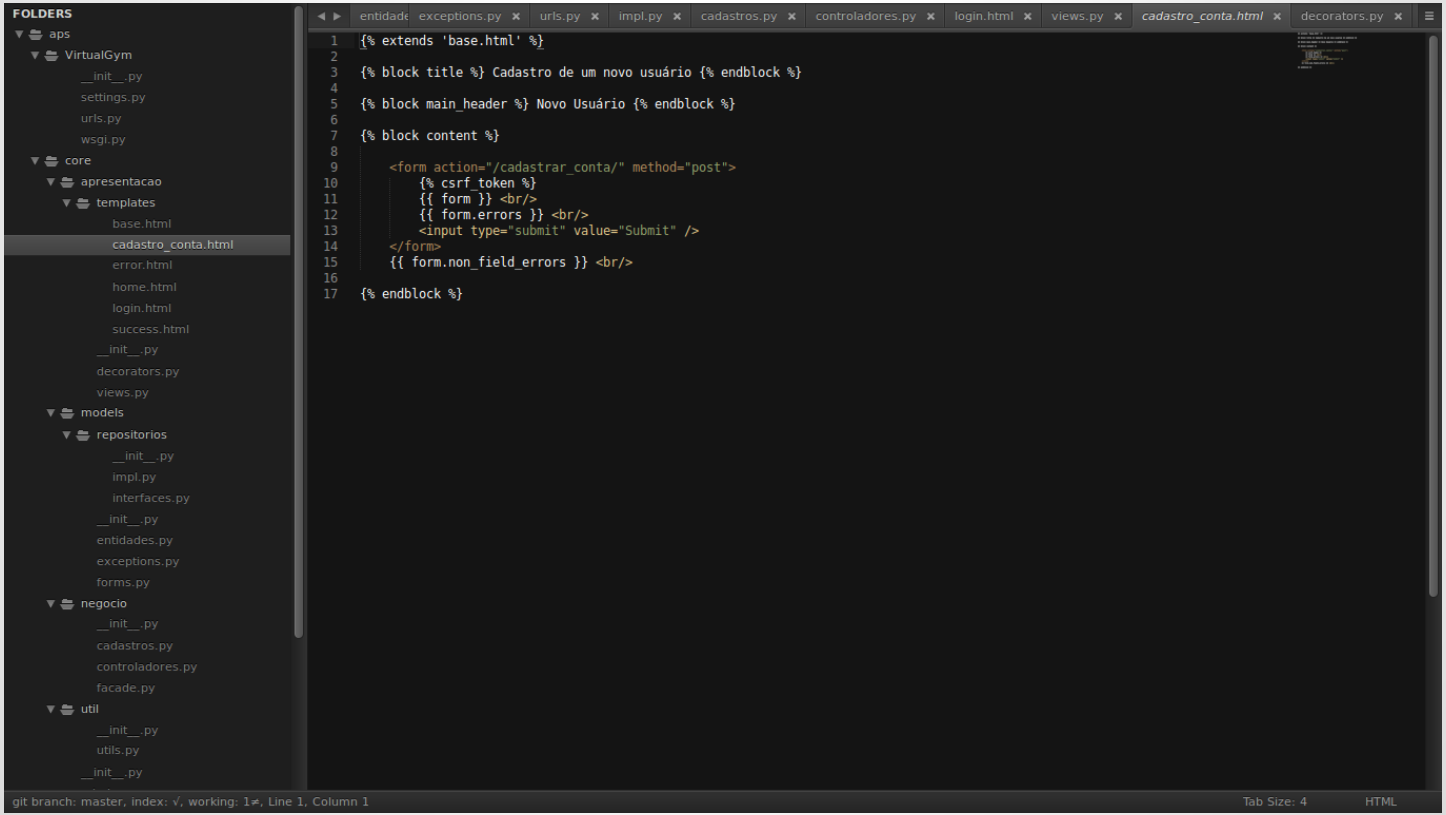
Código - CRUD (Cadastrar Conta)



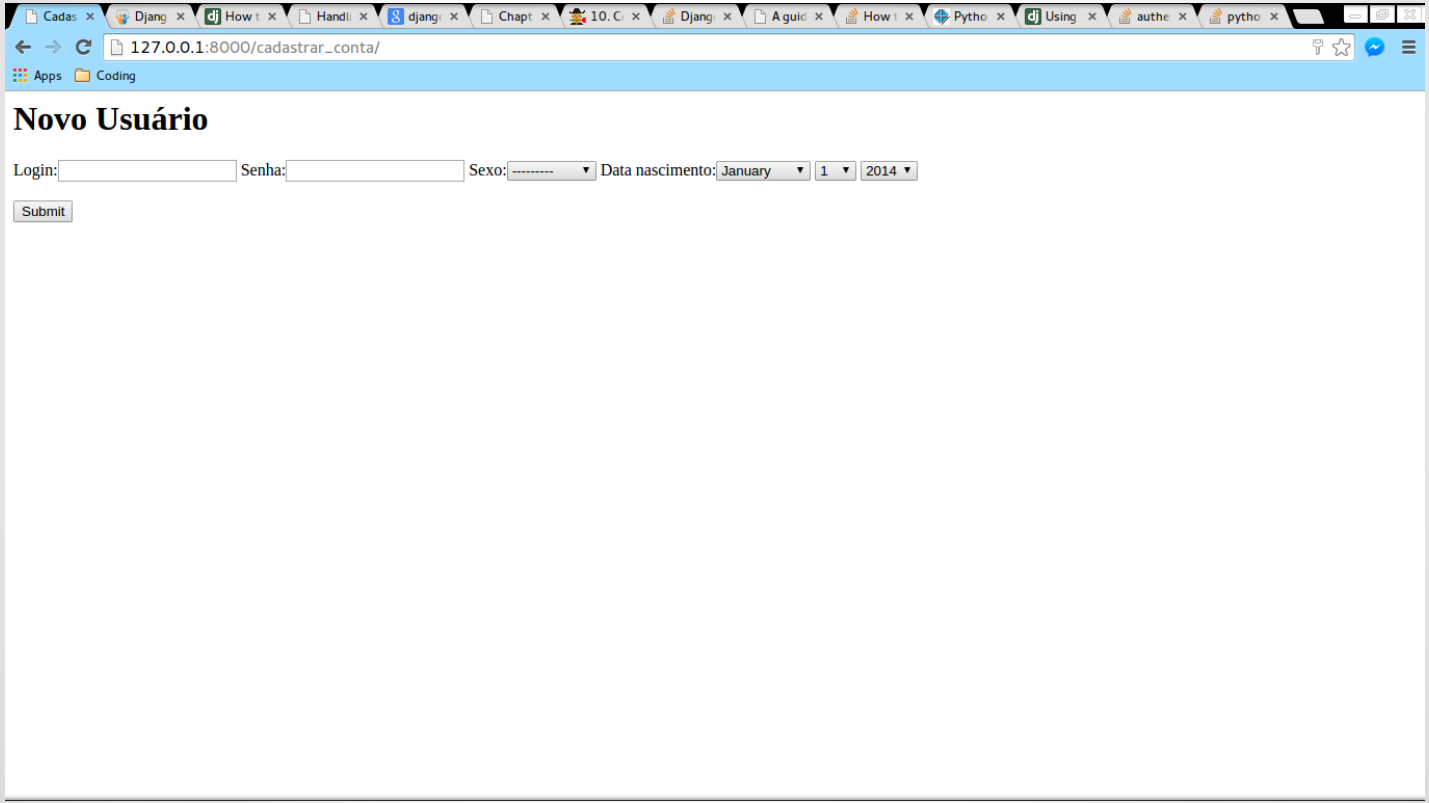
Código - CRUD (Cadastrar Conta)



Código - CRUD (Cadastrar Conta)

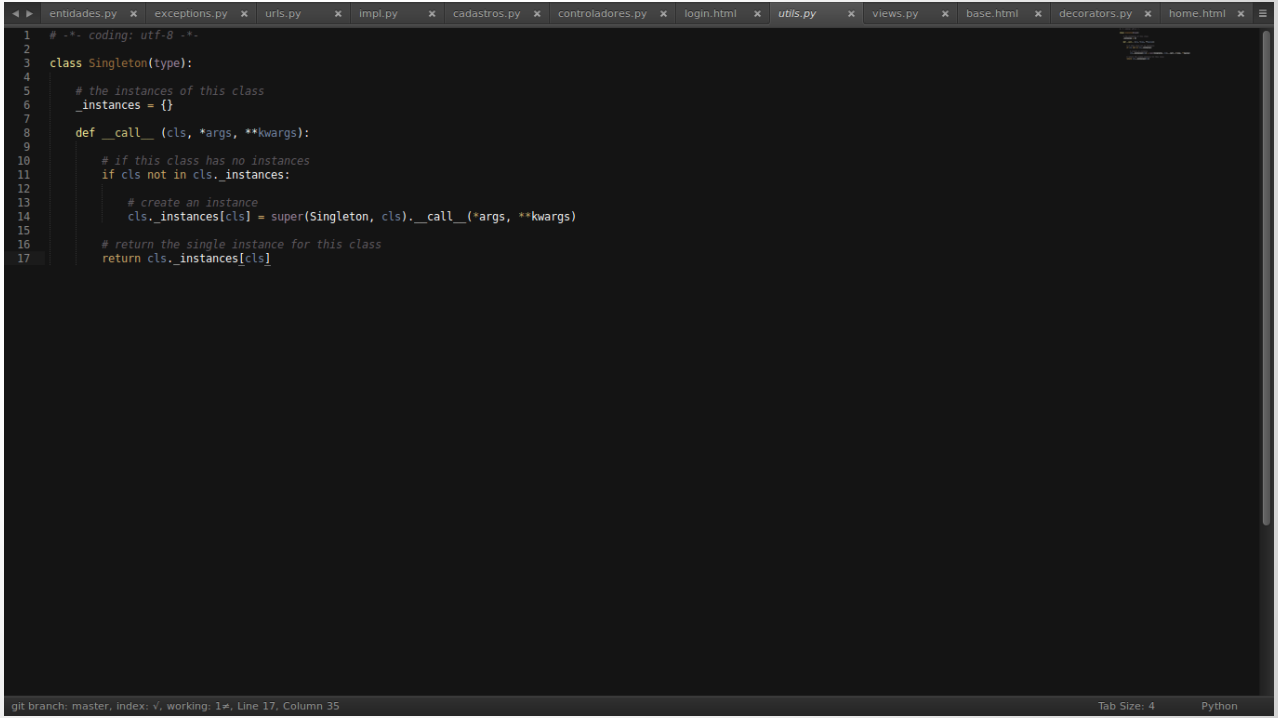


Código - CRUD (Cadastrar Conta)



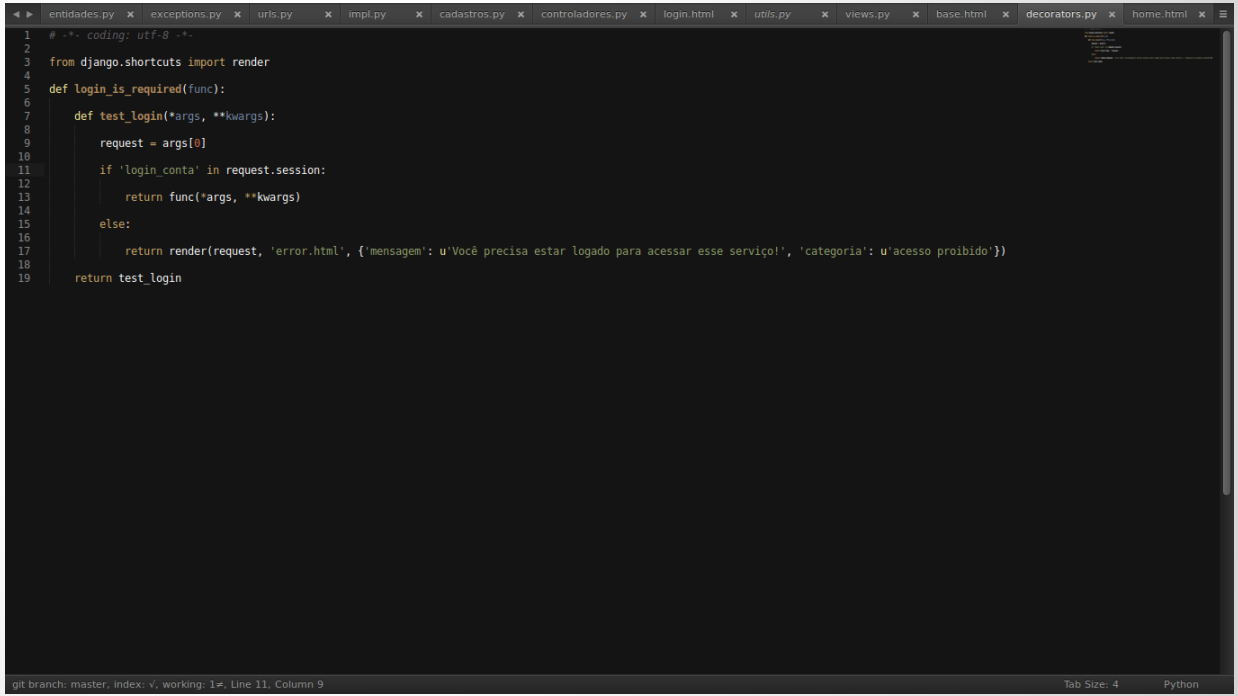
Código - Padrões de Projeto

- Singleton



Código - Padrões de Projeto

- Decorator



Código - Padrões de Projeto

- Decorator