Introdução a Rest APIs com Python Flask

Prof. Me. Hélio Esperidião

Flask

- Lançado em 2010 e criado por Armin Ronacher, o Flask é um micro-framework voltado para aplicações menores com requisitos simples, como a construção de sites básicos.
- Ele oferece um núcleo leve e expansível, permitindo que a aplicação utilize apenas os recursos essenciais para sua execução.
- Conforme novas necessidades surgem, pacotes adicionais podem ser incorporados para aumentar as funcionalidades.

Características do Flask

- **Simplicidade**: O Flask fornece apenas os recursos essenciais para o desenvolvimento de uma aplicação, o que torna os projetos mais enxutos em comparação com frameworks maiores. Com menos arquivos e uma arquitetura mais simples, o desenvolvimento é facilitado.
- **Desenvolvimento ágil**: No Flask, o foco do desenvolvedor é apenas no que é necessário para o projeto, evitando configurações complexas ou desnecessárias, o que acelera o processo de desenvolvimento.
- **Projetos menores**: Com uma estrutura básica e a possibilidade de iniciar o projeto com um único arquivo, os projetos em Flask tendem a ser mais compactos e leves quando comparados a frameworks mais complexos.

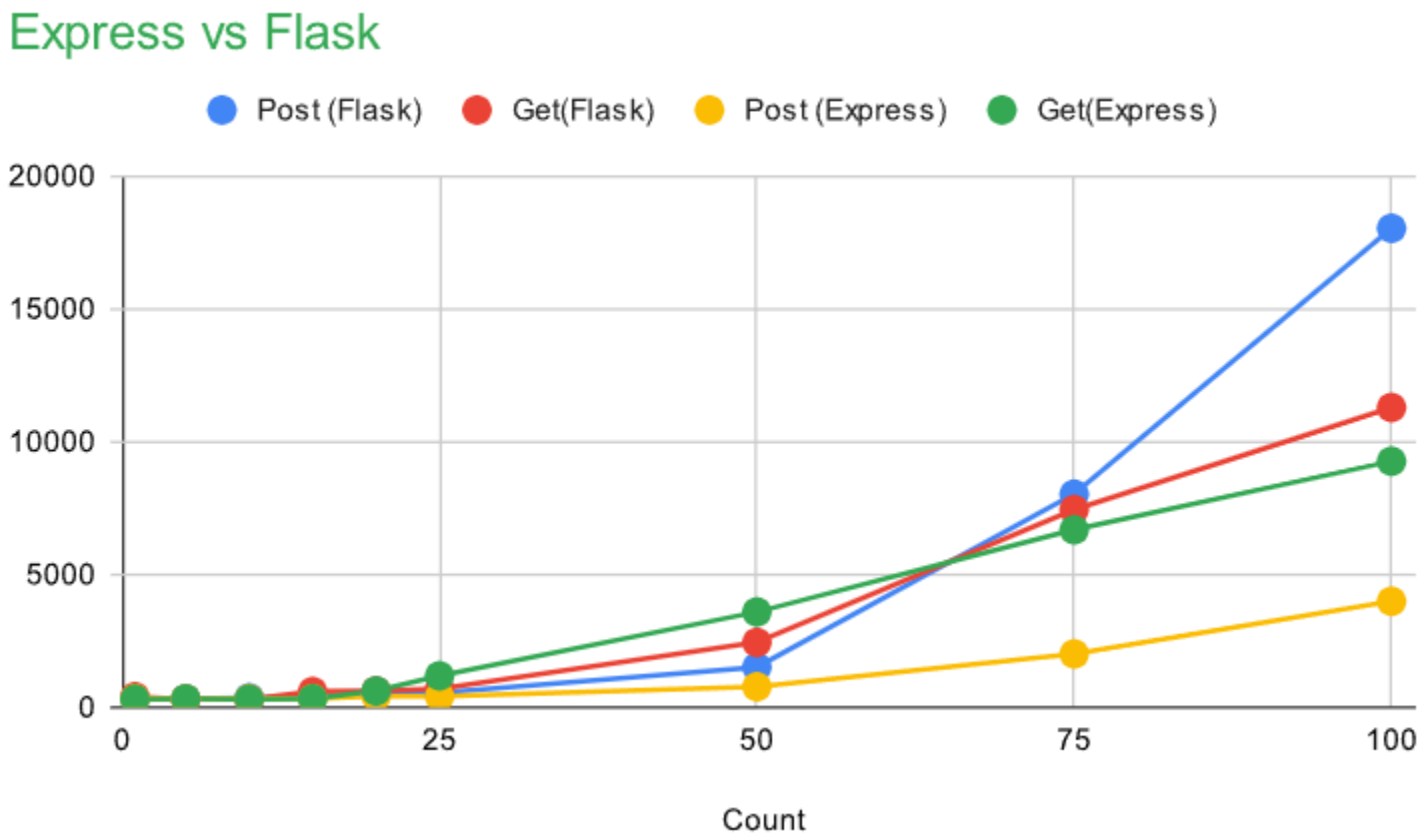
Desvantagens do Flask

- Muitas das vantagens do Flask podem se tornar desvantagens, dependendo do tipo de aplicação a ser desenvolvida.
- **Falta de ferramentas**: Para aplicações maiores, essa limitação pode levar os desenvolvedores a gastarem mais tempo configurando manualmente novas extensões e bibliotecas, resultando em custos adicionais.
- Dificuldade de compreensão: Em projetos maiores, a integração de diversas extensões e bibliotecas pode criar uma falta de padrão a ser seguido. Isso dificulta a adaptação de novos desenvolvedores, que podem ter mais dificuldades para entender as configurações personalizadas

Desvantagens do Flask

- Custo de manutenção: As desvantagens mencionadas anteriormente podem aumentar os custos de desenvolvimento. Além disso, se algum componente da pilha tecnológica se tornar obsoleto, encontrar e implementar um substituto rapidamente pode ser desafiador, resultando em tempos de inatividade prolongados e custos de manutenção mais altos.

Desempenho


Express vs Flask
Post (Flask)
Get(Flask)
Post (Express)
Get(Express)
20000
15000
10000
5000
0
0
25
50
75
100
Count

Makert share

	2023 1 Sep	2023 1 Oct	2023 1 Nov	2023 1 Dec	2024 1 Jan	2024 1 Feb	2024 1 Mar	2024 1 Apr	2024 1 May	2024 1 Jun	2024 1 Jul	2024 1 Aug	2024 1 Sep	2024 22 Sep
PHP	77.1%	76.8%	76.8%	76.6%	76.7%	76.5%	76.5%	76.4%	76.3%	76.2%	76.2%	76.1%	75.9%	75.8%
Ruby	5.4%	5.5%	5.5%	5.6%	5.6%	5.7%	5.7%	5.8%	5.8%	5.9%	5.9%	5.9%	5.9%	6.0%
ASP.NET	6.9%	6.8%	6.8%	6.7%	6.6%	6.5%	6.4%	6.3%	6.2%	6.1%	6.0%	5.9%	5.8%	5.8%
Java	4.7%	4.7%	4.7%	4.7%	4.7%	4.7%	4.8%	4.8%	4.8%	4.8%	4.9%	4.9%	4.9%	5.0%
JavaScript	2.8%	3.0%	3.0%	3.1%	3.1%	3.2%	3.2%	3.3%	3.3%	3.4%	3.4%	3.5%	3.5%	3.6%
Scala	2.9%	2.9%	2.9%	3.0%	3.0%	3.0%	3.0%	3.0%	3.1%	3.1%	3.2%	3.2%	3.4%	3.4%
static files	1.9%	1.9%	1.8%	1.9%	1.8%	1.8%	1.8%	1.8%	1.8%	1.8%	1.8%	1.8%	1.8%	1.7%
Python	1.4%	1.4%	1.4%	1.5%	1.4%	1.4%	1.4%	1.4%	1.4%	1.4%	1.4%	1.3%	1.3%	1.3%
ColdFusion	0.3%	0.3%	0.3%	0.3%	0.3%	0.3%	0.3%	0.3%	0.3%	0.3%	0.2%	0.2%	0.2%	0.2%
Perl	0.2%	0.2%	0.2%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%
Erlang	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%	0.1%

https://w3techs.com/

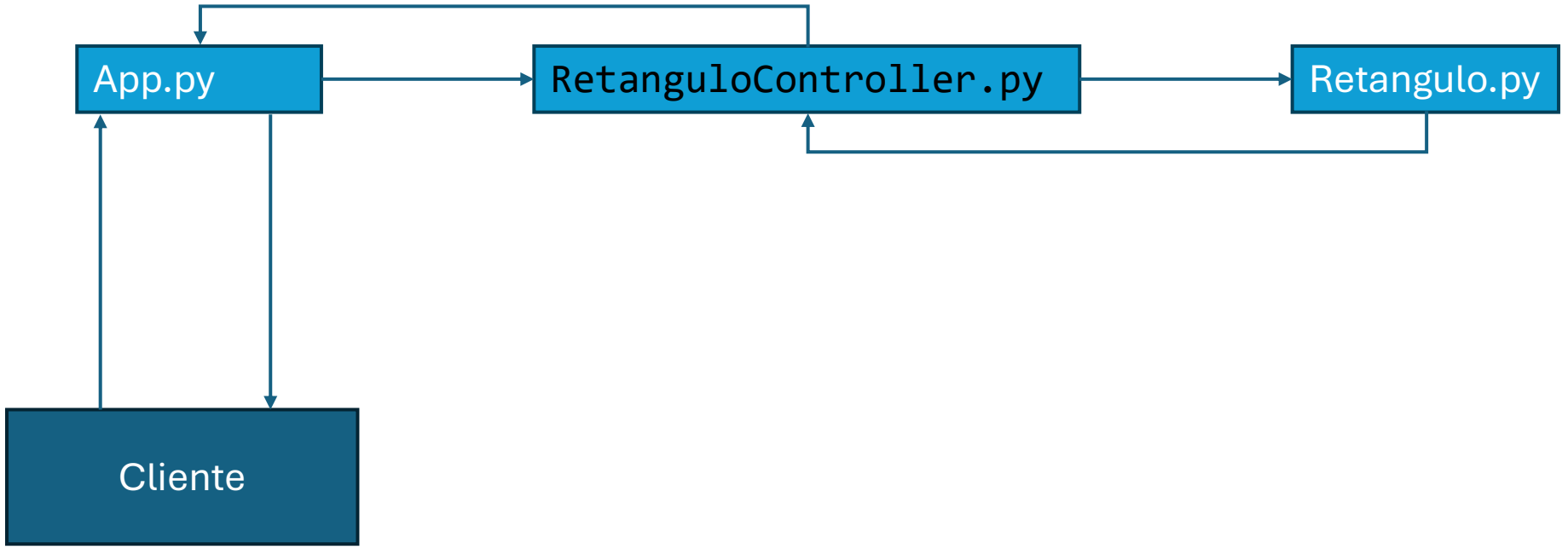
Instalando o Flask

- pip install Flask

Arquitetura MVC

- Model
- View
- Controler

Arquitetura e arquivos:



Classe Retangulo.py 1/2

```python
import math
class Retangulo:
    def __init__(self):
        self._base = None
        self._altura = None
    def calcularArea(self):
        if self._base is not None and self._altura is not None:
            return self._base * self._altura
        else:
            return None
    def calcularDiagonal(self):
        if self._base is not None and self._altura is not None:
            return math.sqrt(self._base ** 2 + self._altura ** 2)
        else:
            return None
    def calcularPerimetro(self):
        if self._base is not None and self._altura is not None:
            return 2 * (self._base + self._altura)
        else:
            return None
```

Definição dos atributos da classe

Métodos da classe

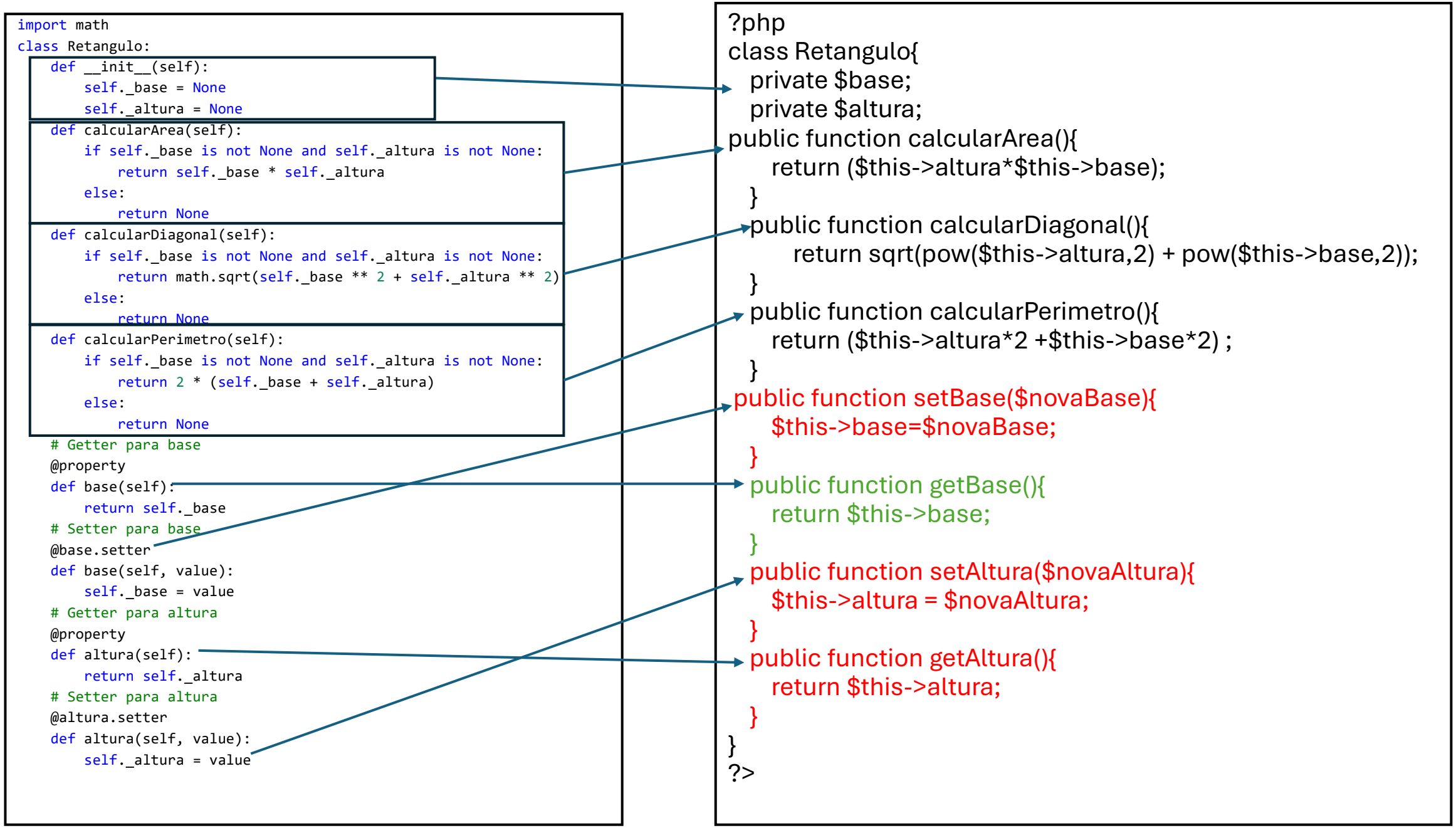
Classe Retangulo.py 2/2

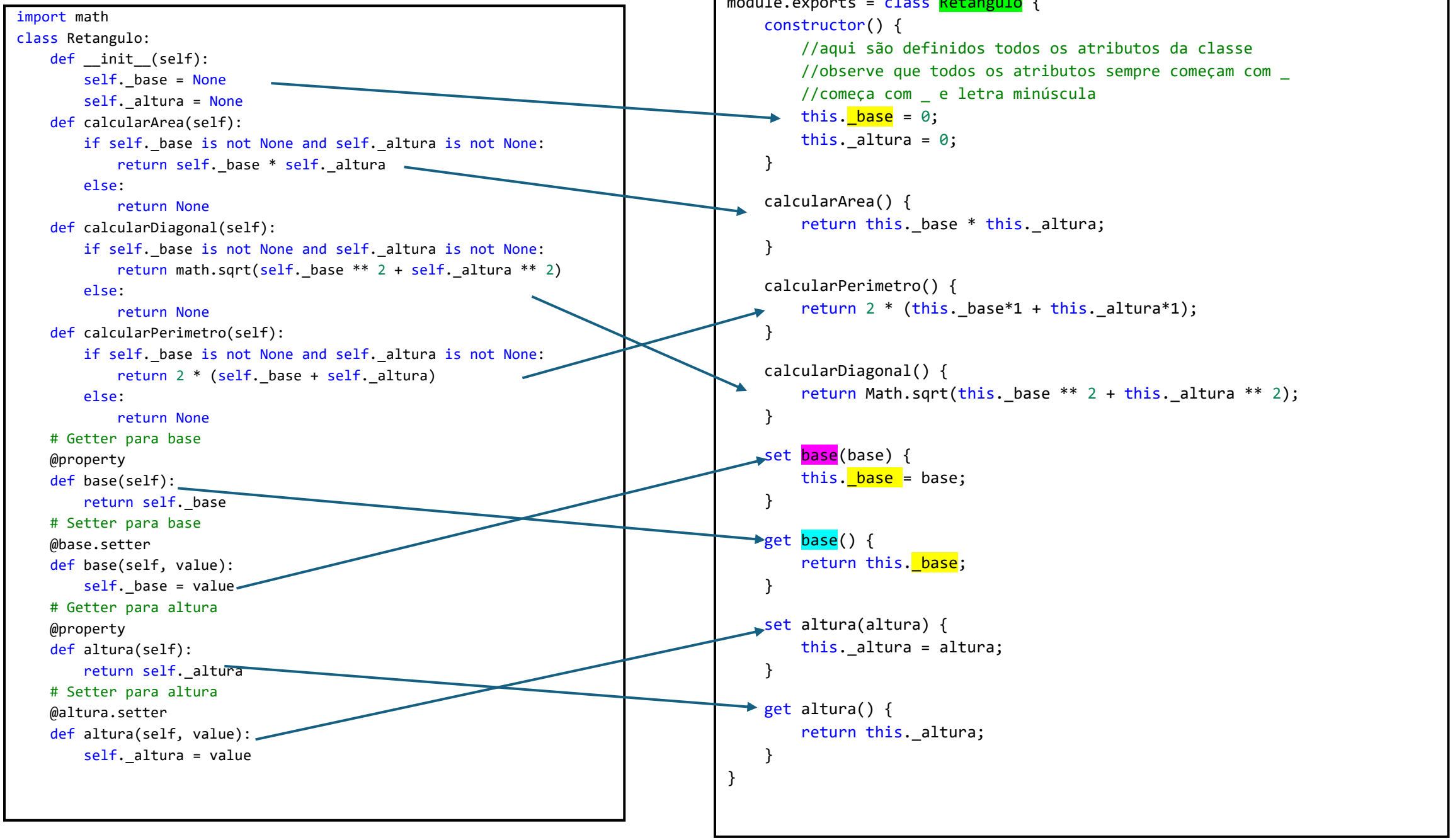
```
    # Getter para base
    @property
    def base(self):
        return self._base

    # Setter para base
    @base.setter
    def base(self, value):
        self._base = value

    # Getter para altura
    @property
    def altura(self):
        return self._altura

    # Setter para altura
    @altura.setter
    def altura(self, value):
        self._altura = value
```


import math
class Retangulo:
def __init__(self):
self._base = None
self._altura = None
def calcularArea(self):
if self._base is not None and self._altura is not None:
return self._base * self._altura
else:
return None
def calcularDiagonal(self):
if self._base is not None and self._altura is not None:
return math.sqrt(self._base ** 2 + self._altura ** 2)
else:
return None
def calcularPerimetro(self):
if self._base is not None and self._altura is not None:
return 2 * (self._base + self._altura)
else:
return None
Getter para base
@property
def base(self):
return self._base
Setter para base
@base.setter
def base(self, value):
self._base = value
# Getter para altura
@property
def altura(self):
return self._altura
Setter para altura
@altura.setter
def altura(self, value):
self._altura = value
?php
class Retangulo{
private $base;
private $altura;
public function calcularArea(){
return ($this->altura*$this->base);
}
public function calcularDiagonal(){
return sqrt(pow($this->altura,2) + pow($this->base,2));
}
public function calcularPerimetro(){
return ($this->altura*2 +$this->base*2) ;
}
public function setBase($novaBase){
$this->base=$novaBase;
}
public function getBase(){
return $this->base;
}
public function setAltura($novaAltura){
$this->altura = $novaAltura;
}
public function getAltura(){
return $this->altura;
}
}
?>


import math
class Retangulo:
def __init__(self):
self._base = None
self._altura = None
def calcularArea(self):
if self._base is not None and self._altura is not None:
return self._base * self._altura
else:
return None
def calcularDiagonal(self):
if self._base is not None and self._altura is not None:
return math.sqrt(self._base ** 2 + self._altura ** 2)
else:
return None
def calcularPerimetro(self):
if self._base is not None and self._altura is not None:
return 2 * (self._base + self._altura)
else:
return None
Getter para base
@property
def base(self):
return self._base
Setter para base
@base.setter
def base(self, value):
self._base = value
# Getter para altura
@property
def altura(self):
return self._altura
Setter para altura
@altura.setter
def altura(self, value):
self._altura = value
module.exports = class Retangulo {
constructor() {
//aqui são definidos todos os atributos da classe
//observe que todos os atributos sempre começam com _
//começa com _ e letra minúscula
this._base = 0;
this._altura = 0;
}
calcularArea() {
return this._base * this._altura;
}
calcularPerimetro() {
return 2 * (this._base*1 + this._altura*1);
}
calcularDiagonal() {
return Math.sqrt(this._base ** 2 + this._altura ** 2);
}
set base(base) {
this._base = base;
}
get base() {
return this._base;
}
set altura(altura) {
this._altura = altura;
}
get altura() {
return this._altura;
}
}

App.py 1/4

```python
from flask import Flask, jsonify
from control.RetanguloController import RetanguloController

app = Flask("API Retangulo")
# Função auxiliar para lidar com validações
def handle_validation_error(e):
    return jsonify({"erro": str(e)}), 400

# Endpoint GET :/retangulos/areas/<float:base>/<float:altura>
@app.route('/retangulos/areas/<float:base>/<float:altura>', methods=['GET'])
def get_retangulos_area(base, altura):
    try:
        retanguloController = RetanguloController()
        retanguloController.retangulo.base= base
        retanguloController.retangulo.altura = altura
        area = retanguloController.calcular_area()
        jsonResposta = {
            "base": base,
            "altura": altura,
            "area": area
        }
        return jsonify(jsonResposta), 200
    except ValueError as e:
        return handle_validation_error(e)
```

App.py 2/4

```python
# Endpoint GET :/retangulos/perimetros/<float:base>/<float:altura>
@app.route('/retangulos/perimetros/<float:base>/<float:altura>', methods=['GET'])
def get_retangulos_perimetro(base, altura):
    try:
        retanguloController = RetanguloController()
        retanguloController.retangulo.base= base
        retanguloController.retangulo.altura = altura
        perimetro = retanguloController.calcular_perimetro()
        jsonResposta = {
            "base": base,
            "altura": altura,
            "perimetro": perimetro
        }
        return jsonify(jsonResposta), 200
    except ValueError as e:
        return handle_validation_error(e)
```

App.py 3/4

```python
# Endpoint GET :/retangulos/diagonais/<float:base>/<float:altura>
@app.route('/retangulos/diagonais/<float:base>/<float:altura>', methods=['GET'])
def get_retangulos_diagonal(base, altura):
    try:
        retanguloController = RetanguloController()
        retanguloController.retangulo.base= base
        retanguloController.retangulo.altura = altura
        diagonal = retanguloController.calcular_diagonal()
        jsonResposta = {
            "base": base,
            "altura": altura,
            "diagonal": diagonal
        }
        return jsonify(jsonResposta), 200
    except ValueError as e:
        return handle_validation_error(e)
```

App.py 4/4

```
# inicia o servidor
app.run(host='0.0.0.0', port=8080)
```

RetanguloController.py
1/3

```python
from model.Retangulo import Retangulo

class RetanguloController:
    def __init__(self):
        self._retangulo = Retangulo()

    def validarBase(self):
        if self._retangulo.base <= 0:
            raise ValueError("A base deve ser maiores que zero.")
    def validarAltura(self):
        if self._retangulo.altura <= 0:
            raise ValueError("A altura deve ser maiores que zero.")
```

```
def calcular_area(self):
    self.validarBase()
    self.validarAltura()
    return self._retangulo.calcularArea()

def calcular_perimetro(self):
    self.validarBase()
    self.validarAltura()
    return self._retangulo.calcularPerimetro()

def calcular_diagonal(self):
    self.validarBase()
    self.validarAltura()
    return  self._retangulo.calcularDiagonal()
```

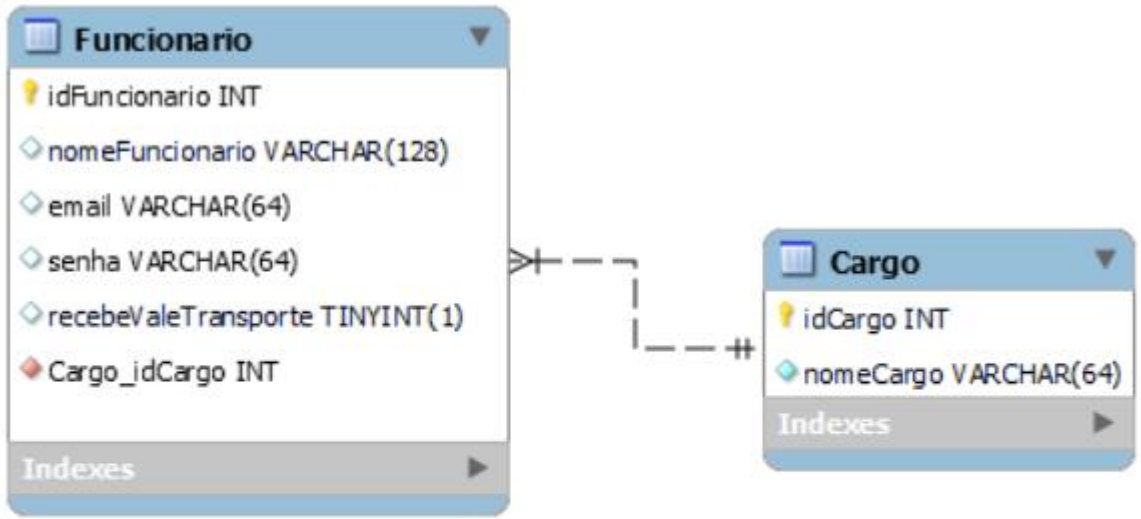
RetanguloController.py
3/3

```
@property
def retangulo(self):
    return self._retangulo

@retangulo.setter
def retangulo(self, valor):
    self._retangulo = valor
```

Exemplo Cargo - Funcionário

Modelo de banco de dados



Para exportar(Database >> Foward Engineer) o modelo para os códigos sql de criação é nessário que seja mudada a versão do mysql para 5.5.6.
Vá em : Edit>> preferences >> mysql >> default target Mysql version: 5.5.6

SQL/Banco

```
DROP SCHEMA  `aula_api_2024`;
CREATE SCHEMA IF NOT EXISTS `aula_api_2024` DEFAULT CHARACTER SET utf8 ;
USE `aula_api_2024` ;
CREATE TABLE IF NOT EXISTS `aula_api_2024`.`Cargo` (
  `idCargo` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `nomeCargo` VARCHAR(64) NOT NULL,
  PRIMARY KEY (`idCargo`),
  UNIQUE INDEX `idCargo_UNIQUE` (`idCargo` ASC),
  UNIQUE INDEX `nomeCargo_UNIQUE` (`nomeCargo` ASC))
ENGINE = InnoDB;
CREATE TABLE IF NOT EXISTS `aula_api_2024`.`Funcionario` (
  `idFuncionario` INT UNSIGNED NOT NULL AUTO_INCREMENT,
  `nomeFuncionario` VARCHAR(128) NULL,
  `email` VARCHAR(64) NULL,
  `senha` VARCHAR(64) NULL,
  `recebeValeTransporte` TINYINT(1) NULL,
  `Cargo_idCargo` INT UNSIGNED NOT NULL,
  PRIMARY KEY (`idFuncionario`),
  UNIQUE INDEX `idFuncionario_UNIQUE` (`idFuncionario` ASC),
  INDEX `fk_Funcionario_Cargo_idx` (`Cargo_idCargo` ASC),
  CONSTRAINT `fk_Funcionario_Cargo`
    FOREIGN KEY (`Cargo_idCargo`)
    REFERENCES `aula_api_2024`.`Cargo` (`idCargo`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
INSERT INTO `aula_api_2024`.`Cargo` (`idCargo`, `nomeCargo`) VALUES (1, 'Administrador');
INSERT INTO `aula_api_2024`.`Cargo` (`idCargo`, `nomeCargo`) VALUES (2, 'Técnico em Informática Jr');
INSERT INTO `aula_api_2024`.`Cargo` (`idCargo`, `nomeCargo`) VALUES (3, 'Técnico em Informática Pleno');
INSERT INTO `aula_api_2024`.`Cargo` (`idCargo`, `nomeCargo`) VALUES (4, 'Analista de Sistemas Jr');

INSERT INTO `aula_api_2024`.`funcionario` (`nomeFuncionario`, `email`, `senha`, `recebeValeTransporte`, `Cargo_idCargo`) VALUES ('adm', 'adm@adm',
md5(123456), 1, 1);
```

instale

- pip install Flask
- pip install mysql-connector-python
- pip install PyJWT

App.py 1/4

```python
from flask import Flask, jsonify, request, Response
from controle.CargoController import CargoController
from model.Cargo import Cargo

app = Flask("rest_api")
# Função auxiliar para lidar com validações
def handle_validation_error(e):
    return jsonify({"erro": str(e)}), 400
```

App.py 2/4

```python
# Endpoint GET /cargos
@app.route('/cargos/', methods=['GET'])
def readAll():
    try:
        cargoController = CargoController()
        return cargoController.read_all()
    except ValueError as e:
        return handle_validation_error(e)

# Endpoint GET /cargos/<int:id>
@app.route('/cargos/<int:id>', methods=['GET'])
def readById(id):
    try:
        objCargoController = CargoController()
        return  objCargoController.read_by_id(id)
    except ValueError as e:
        return handle_validation_error(e)
```

App.py 3/4

```python
# Endpoint POST /cargos
@app.route('/cargos/', methods=['POST'])
def create():
    try:
        body = request.get_json()
        objCargoController = CargoController()
        objCargoController.cargo.nomeCargo = body['cargo']['nomeCargo']

        return  objCargoController.create_control()
    except ValueError as e:
        return handle_validation_error(e)

# Endpoint PUT /cargos/<int:id>
@app.route('/cargos/<int:id>', methods=['PUT'])
def update(id):
    try:
        body = request.get_json()
        objCargoController = CargoController()
        objCargoController.cargo.nomeCargo = body['cargo']['nomeCargo']
        objCargoController.cargo.idCargo = id
        return  objCargoController.update()

    except ValueError as e:
        return handle_validation_error(e)
```

App.py 4/4

```python
# Endpoint DELETE /cargos/<int:id>
@app.route('/cargos/<int:id>', methods=['DELETE'])
def delete(id):
    try:
        cargo = Cargo()
        linhas_afetadas = cargo.delete(id)
        if linhas_afetadas:
            return jsonify({"message": "Cargo deletado com sucesso"}), 200
        else:
            return jsonify({"message": "Cargo não encontrado"}), 404
    except ValueError as e:
        return handle_validation_error(e)

# inicia o servidor
app.run(host='0.0.0.0', port=8080)
```

CargoController.py
1/4

```python
from flask import Flask, jsonify, request
from model.Cargo import Cargo  # Ajuste conforme a localização do arquivo da classe Cargo

class CargoController:
    def __init__(self):
        self._cargo = Cargo()

    def validar_nomeCargo(self):
        if self._cargo.nomeCargo is None:
            raise ValueError("O nome do cargo não pode ser vazio")
        if len(self._cargo.nomeCargo) < 3:
            raise ValueError("O nome do cargo não pode ser vazio e deve ter pelo menos 3 caracteres.")
```

CargoController.py 2/4

```
def read_all(self):
        cargos = self._cargo.readAll()
        if cargos is not None:
            return jsonify(cargos), 200
        else:
            return jsonify({"message": "Não foi possível obter os cargos"}), 500

    def read_by_id(self,id):

        cargo_data = self._cargo.readCargoById(id)
        if cargo_data:
            return jsonify(cargo_data), 200
        else:
            return jsonify({"message": "Cargo não encontrado"}), 404


    def create_control(self):
        self.validar_nomeCargo()
        id_novo_cargo = self._cargo.create()
        if id_novo_cargo:
            return jsonify({"idCargo": id_novo_cargo,
                            "nomeCargo":self._cargo.nomeCargo}), 201
        else:
            return jsonify({"message": "Não foi possível criar o cargo"}), 500
```

CargoController.py
3/4

```
def update(self):
        self.validar_nomeCargo()
        id_novo_cargo = self._cargo.update()
        if id_novo_cargo:
            return jsonify({"idCargo": id_novo_cargo,
                            "nomeCargo":self._cargo.nomeCargo}), 200
        else:
            return jsonify({"message": "Não foi possível atualizar o cargo"}), 500

    def delete(id):
        cargo = Cargo()
        linhas_afetadas = cargo.delete(id)
        if linhas_afetadas:
            return jsonify({"message": "Cargo excluído com sucesso"}), 200
        else:
            return jsonify({"message": "Cargo não encontrado"}), 404
```

CargoController.py
4/4

```
# Getter para nomeCargo
@property
def cargo(self):
    return self._cargo

# Setter para nomeCargo
@cargo.setter
def cargo(self, value):
    self._cargo = value
```

Banco.py 1/3

```python
import mysql.connector
from mysql.connector import Error

class Banco:
    HOST = '127.0.0.1'
    USER = 'root'
    PWD = ''
    DB = 'aula_api_2024'
    PORT = 3306
    CONEXAO = None
```

Banco.py 2/3

```
@staticmethod
    def conectar():
        # Verifica se já existe uma conexão estabelecida
        if Banco.CONEXAO is None
            try:
                # Tenta estabelecer uma nova conexão utilizando as informações fornecidas
                Banco.CONEXAO = mysql.connector.connect(
                    host=Banco.HOST,
                    user=Banco.USER,
                    password=Banco.PWD,
                    database=Banco.DB,
                    port=Banco.PORT
                )
                print("Conexão com o banco de dados estabelecida")
            except Error as err:
                obj_resposta = {
                    'cod': 1,
                    'msg': "Erro ao conectar no banco",
                    'erro': str(err)
                }
                print(obj_resposta)  # Log de erro
                exit(1)  # Encerra o script em caso de erro
```

Banco.py 3/3

```
# Método público para obter a conexão com o banco de dados
@staticmethod
def getConexao():
    # Verifica se já existe uma conexão estabelecida
    if Banco.CONEXAO is None:
        # Se não houver, estabelece uma nova conexão
        Banco.conectar()
    # Retorna a conexão
    return Banco.CONEXAO
```

Cargo.py
1/7

```python
from model.Banco import Banco
from mysql.connector import Error

class Cargo:
    def __init__(self):

        self._idCargo = None
        self._nomeCargo = None
```

Cargo.py
2/7

```
def create(self):
    conexao = Banco.getConexao()
    if conexao:
        try:
            cursor = conexao.cursor()
            sql = "INSERT INTO cargo (nomeCargo) VALUES (%s)"
            cursor.execute(sql, (self.nomeCargo,))
            conexao.commit()
            self.idCargo = cursor.lastrowid  # Atualiza o idCargo após criação
            return self.idCargo
        except Error as e:
            print(f"Erro ao criar cargo: {e}")
            return None
        finally:
            cursor.close()
```

Cargo.py
3/7

```
def readAll(self):
        conexao = Banco.getConexao()
        if conexao:
            try:
                cursor = conexao.cursor(dictionary=True)
                sql = "SELECT * FROM cargo order by nomeCargo asc"
                cursor.execute(sql)
                return cursor.fetchall()
            except Error as e:
                print(f"Erro ao obter cargos: {e}")
                return None
            finally:
                cursor.close()
```

Cargo.py
4/7

```
def readCargoById(self, idCargo):
        conexao = Banco.getConexao()
        if conexao:
            try:
                cursor = conexao.cursor(dictionary=True)
                sql = "SELECT * FROM cargo WHERE idCargo = %s"
                cursor.execute(sql, (idCargo,))
                result = cursor.fetchone()
                if result:
                    self.idCargo = result['idCargo']
                    self.nomeCargo = result['nomeCargo']
                return result
            except Error as e:
                print(f"Erro ao obter cargo por ID: {e}")
                return None
            finally:
                cursor.close()
```

Cargo.py
5/7

```
def update(self):
        conexao = Banco.getConexao()
        if conexao:
            try:
                cursor = conexao.cursor()
                sql = "UPDATE cargo SET nomeCargo = %s WHERE idCargo = %s"
                cursor.execute(sql, (self.nomeCargo, self.idCargo))
                conexao.commit()
                return cursor.rowcount
            except Error as e:
                print(f"Erro ao atualizar cargo: {e}")
                return None
            finally:
                cursor.close()
```

Cargo.py
6/7

```
def delete(self, idCargo):
    conexao = Banco.getConexao()
    if conexao:
        try:
            cursor = conexao.cursor()
            sql = "DELETE FROM cargo WHERE idCargo = %s"
            cursor.execute(sql, (idCargo,))
            conexao.commit()
            return cursor.rowcount
        except Error as e:
            print(f"Erro ao deletar cargo: {e}")
            return None
        finally:
            cursor.close()
```

Cargo.py
7/7

```python
# Getter para idCargo
@property
def idCargo(self):
    return self._idCargo

# Setter para idCargo
@idCargo.setter
def idCargo(self, value):
    self._idCargo = value

# Getter para nomeCargo
@property
def nomeCargo(self):
    return self._nomeCargo

# Setter para nomeCargo
@nomeCargo.setter
def nomeCargo(self, value):
    self._nomeCargo = value
```

JWT

MeuTokenJWT.py
1/4

```python
import jwt
import datetime
import secrets

class MeuTokenJWT:
    def __init__(self):
        self._key = "x9S4q0v+V0IjvHkG20uAxaHx1ijj+q1HWjHKv+ohxp/oK+77qyXkVj/l4QYHHTF3"  # Chave secreta
        self._alg = 'HS256'  # Algoritmo de criptografia
        self._type = 'JWT'
        self._iss = 'http://localhost'  # Emissor do token
        self._aud = 'http://localhost'  # Destinatário do token
        self._sub = "acesso_sistema"  # Assunto do token
        self._duracaoToken = 3600 * 24 * 30  # Duração do token (30 dias)
        self.payload = None
```

MeuTokenJWT.py
2/4

```
def gerar_token(self, parametro_claims):
        headers = {
            'alg': self._alg,
            'typ': self._type
        }

        agora = datetime.datetime.now(datetime.timezone.utc)
        payload = {
            'iss': self._iss,  # Emissor do token
            'aud': self._aud,  # Destinatário do token
            'sub': self._sub,  # Assunto do token
            'iat': int(agora.timestamp()),  # Momento de criação (em segundos)
            'exp': int((agora + datetime.timedelta(seconds=self._duracaoToken)).timestamp()),  # Expiração (em segundos)
            'nbf': int(agora.timestamp()),  # Não é válido antes do tempo especificado
            'jti': secrets.token_hex(16),  # Identificador único (jti)
            'email': parametro_claims['email'],  # Claims públicas
            'role': parametro_claims['role'],
            'name': parametro_claims['name'],
            'idFuncionario': parametro_claims['idFuncionario']  # Claims privadas
        }

        # Gera o token utilizando a biblioteca PyJWT
        token = jwt.encode(payload, self._key, algorithm=self._alg, headers=headers)
        return token
```

MeuTokenJWT.py
3/4

```
def validar_token(self, string_token):
        if not string_token or string_token.strip() == "":
            return False
        token = string_token.replace("Bearer ", "").strip()
        try:
            decoded = jwt.decode(token, self._key, algorithms=[self._alg])
            self.payload = decoded
            return True
        except jwt.ExpiredSignatureError:
            print("Token expirado")
            return False
        except jwt.InvalidTokenError:
            print("Token inválido")
            return False
```

MeuTokenJWT.py
4/4

```
def get_payload(self):
    return self.payload

def set_payload(self, payload):
    self.payload = payload

def get_alg(self):
    return self._alg

def set_alg(self, alg):
    self._alg = alg
```