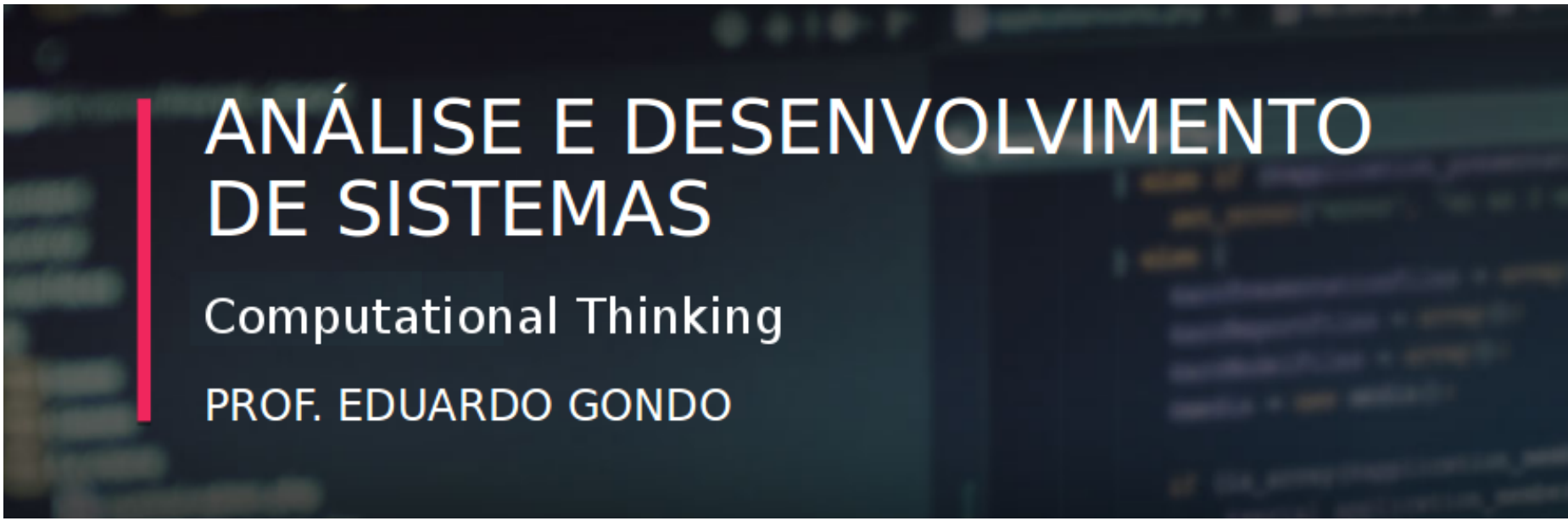
ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
Computational Thinking
PROF. EDUARDO GONDO

API - Introdução

- **API** — Application Programming Interface
- são bibliotecas que provém recursos para a construção de aplicações
- elas podem ser públicas ou privadas
- acessadas localmente ou através da internet
- algumas delas são úteis para integração de sistemas heterogêneos
- apresentaremos a API RESTful nos próximos eslaides

API - RESTful

- independente de plataforma
- ela é disponibilizada através de uma rede sobre o protocolo **HTTP**
- ela define um várias URL's e as chamadas são feitas através de request e response
- a troca de informações entre a API e os sistemas é feita através de **JSON**
- o JSON é um dicionário do Python ou uma lista de dicionários
- no JSON, todas as chaves são String

API - Métodos

- existem vários métodos no protocolo HTTP: GET, POST, PUT, DELETE, HEAD, CONNECT, TRACE, etc
- cada um deles define uma ação sobre um **recurso** disponibilizado pela API
- vamos utilizar 4 deles: GET, POST, PUT e DELETE para construir nossa API
- GET — recupera recurso(s)
- POST — cria um recurso
- PUT — atualiza um recurso
- DELETE — apaga um recurso

API - Flask

- usaremos a biblioteca Flask para criarmos uma API
- para instalar use `pip install Flask`
- vamos trabalhar exemplos de cada um dos métodos para montar um CRUD de veículos
- primeiramente, vamos implementar tudo em memória e a tarefa de vocês será integrar com o Oracle

```
1  carros = [
2      {'id': 1, 'montadora': 'Ford', 'modelo': 'Fiesta', 'ano': 2014, 'placa':
           'KRE-8567'},
3      {'id': 2, 'montadora': 'Ford', 'modelo': 'Fusion', 'ano': 2018, 'placa':
           'FIA-9876'},
4      {'id': 3, 'montadora': 'Fiat', 'modelo': 'Mobi', 'ano': 2021, 'placa': '
           MRE-7433'},
5      {'id': 4, 'montadora': 'Honda', 'modelo': 'Fit', 'ano': 2016, 'placa': '
           GRU-6323'},
6      {'id': 5, 'montadora': 'GM', 'modelo': 'Onix', 'ano': 2020, 'placa': 'DGE
           -0955'},
7      {'id': 6, 'montadora': 'GM', 'modelo': 'Prisma', 'ano': 2019, 'placa': '
           MHN-4032'}
8  ]
```

Figura: arquivo db.py

Flask — exemplos

Estrutura básica de uma API em Python

```
1  #importando as bibliotecas necessarias
2  from flask import Flask, request, jsonify
3
4  #instanciando objeto Flask, o parametro pode ser qualquer
       String
5  app = Flask(__name__)
6
7  # implemente as funcoes que serao disponibilizadas
8
9
10 # finalizando a implementacao
11
12 # colocando o servico no ar
13 app.run(debug=True)
```

Flask — exemplos GET

```
1  from flask import Flask, request, jsonify
2  import db
3
4  app = Flask(__name__)
5
6  # definicao de rotas
7  @app.route("/carros", methods=["GET"])
8  def get_all_carros():
9      return (db.carros, 200)
10
11 @app.route("/carros/<int:id>", methods=["GET"])
12 def get_carro(id):
13     for car in db.carros:
14         if id == car['id']:
15             return jsonify(car), 200
16
17     info = {'title': 'Nao encontrado', 'status': 404}
18     return jsonify(info), 404
19
20 app.run(debug=True)
```

Flask — observações

- na linha 1 e 2 temos as importações das bibliotecas
- `@app.route("...", methods=[...])` define as rotas da API
- elas devem aparecer imediatamente acima das funções que as definem
- na linha 11, definimos uma rota com a recepção do parâmetro `id` na URL que é passado para a função
- o `int:` indica que o parâmetro é um número inteiro
- o retorno das funções são tuplas, o primeiro valor é o dicionário no padrão JSON e o segundo o HTTP Status Code
- veja em https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status maiores detalhes sobre cada código

Flask — exemplo POST

- veja o código para inserir um carro dentro do servidor
- para recuperar as informações enviadas pelo cliente, usamos o método json do objeto request, veja a linha 3 do código abaixo

```
1  @app.route("/carros", methods=['POST'])
2  def insere_carro():
3    dado = request.json
4    for car in db.carros:
5       if car['id'] == dado['id'] or car['placa'] == dado['
            placa']:
6          info = {'title': 'Carro existente', 'status': 406}
7          return jsonify(info), 406
8    db.carros.append(dado)
9    return jsonify(dado), 201
```

- veja a regra de negócio na linha 5 que impede adicionarmos carros com o mesmo id ou a mesma placa
- a RFC 7807 indica como devemos retornar informações quando acontecem erros

Flask — exemplo PUT

```
1  @app.route("/carros", methods=['PUT'])
2  def atualiza_carro():
3      dado = request.json
4      for ind, car in enumerate(db.carros):
5          if car['id'] == dado['id']:
6              db.carros[ind] = dado
7              return jsonify(dado), 200
8
9      info = {'title': 'Carro not found', 'status': 404, "
           type": "https://fiap.com.br/car_not_found", "
           detail": f"Carro nao encontrado com o id
           especificado {dado['id']}", "instance": f"/carros
           /{dado['id']}"}
10     return jsonify(info), 404
```

- de acordo com a RFC 7807 devemos retornar um json com as seguintes informações quando acontece algum erro na API
- veja a linha 9 com um exemplo dessas informações

Flask — exemplo DELETE

Exemplo de delete da aplicação

```
1  @app.route("/carros/<int:id>", methods=['DELETE'])
2  def apaga_carro(id):
3      for ind, car in enumerate(db.carros):
4          if car['id'] == id:
5              dado = db.carros.pop(ind)
6              return jsonify(dado), 200
7
8      info = {'title': 'Nao encontrado carro com este id', '
           status': 404}
9      return jsonify(info), 404
```

Flask — Testando API

- escrever um código cliente: python, JS, Angular, React, etc
- usar alguma ferramenta pronta
- postman
- insomnia
- antes de testar a API, vou acrescentar outro método GET que recupera os carros por montadora

Flask — exemplo GET

Exemplo de delete da aplicação

```
1  @app.route("/carros/montadora/<string:montadora>", methods
       =["GET"])
2  def get_carros_montadora(montadora):
3      retorno = []
4      for car in db.carros:
5          if montadora == car['montadora']:
6              retorno.append(car)
7
8
9      if len(retorno) > 0:
10         return jsonify(retorno), 200
11     else:
12         return jsonify({'title': 'Nenhum carro encontrado'
               , 'status': 204}), 204
```

Flask — exemplo GET

Exemplo de delete da aplicação

```
1  @app.route("/carros/montadora/<string:montadora>", methods
       =["GET"])
2  def get_carros_montadora(montadora):
3      retorno = []
4      for car in db.carros:
5          if montadora == car['montadora']:
6              retorno.append(car)
7
8
9      if len(retorno) > 0:
10         return jsonify(retorno), 200
11     else:
12         return jsonify({'title': 'Nenhum carro encontrado'
               , 'status': 204}), 204
```

Exercício

1. Refaça a API de carro integrando com o banco de dados Oracle.
2. Construa uma API para fornecer serviços para um cadastro de Pessoa. Pense em 1 ou 2 métodos GET para implementar além do recupera todos os registros ou apenas pelo seu id.
3. No projeto da tabela de classificação, crie uma API Rest para Partida e Time. Faça apenas o cadastra partida e o recupera times.

Referência Bibliográfica

- Puga e Rissetti - Lógica de Programação e Estrutura de Dados
- Ascêncio e Campos - Fundamentos da Programação de Computadores
- Forbelone e Eberspacher - Lógica de programação: a construção de algoritmos e estruturas de dados
- Documentação do Python - https://docs.python.org/3.8/
- Python Programming For Beginners: Learn The Basics Of Python Programming (Python Crash Course, Programming for Dummies) (English Edition). Kindle
- Python: 3 Manuscripts in 1 book: - Python Programming For Beginners - Python Programming For Intermediates - Python Programming for Advanced (English Edition). Kindle

Copyleft