

Curso: Ciência da Computação

Campus: Ribeirão Preto

77B5 - ATIVIDADES PRÁTICAS SUPERVISIONADAS

Nome: Eduardo dos Santos Rodrigues RA: N604794

João Paulo Rodrigues da Cruz RA: F30IBA8

Outubro, 2023

Sumário

Objetivo e Motivação do Trabalho..................................................03

Introdução.......................................................................................05

Fundamentos dos conceitos envolvidos na API Kaggle……………07

Plano de Desenvolvimento da Aplicação........................................11

Projeto.............................................................................................20

Relatório com as Linhas de Código.................................................26

Bibliografia.......................................................................................29

Objetivo e motivação do trabalho

O principal objetivo deste trabalho consiste na elaboração, implementação e validação de uma API inovadora, que, por meio de um sistema de autenticação robusto, garanta segurança de alto padrão e acessibilidade exemplar. A API será desenvolvida com o propósito específico de extrair, manipular e disponibilizar dados de um dataset selecionado hospedado no site Kaggle. A abrangência deste objetivo inclui não apenas a criação técnica da interface, mas também a implementação de práticas avançadas de segurança da informação, assegurando a confidencialidade e integridade dos dados durante todo o ciclo de vida da API.



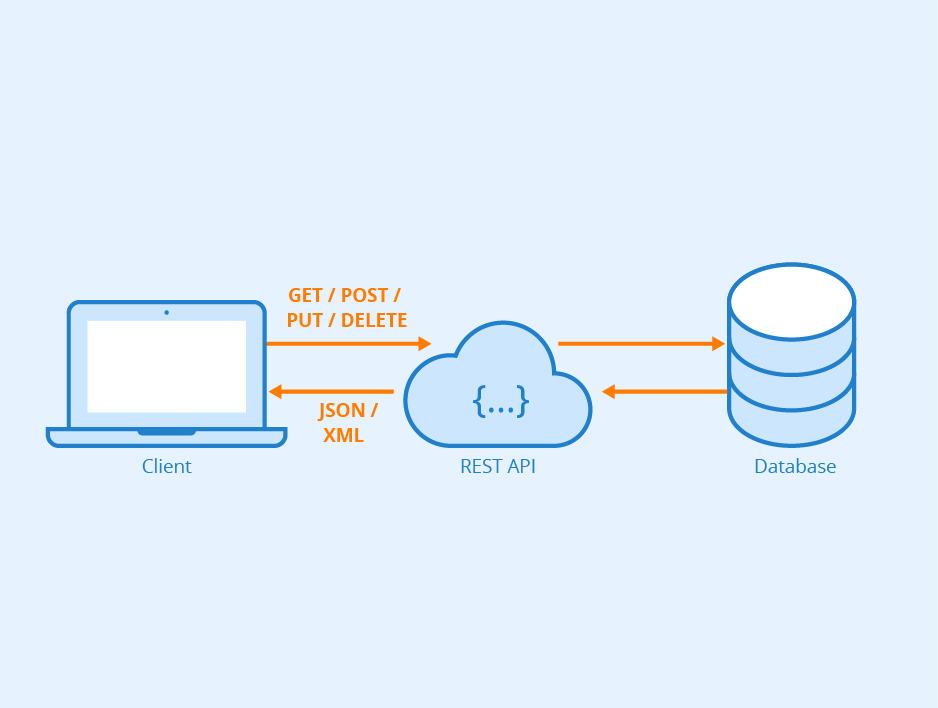
A autenticação, como componente fundamental, será projetada para oferecer uma camada adicional de proteção, garantindo que apenas usuários autorizados tenham acesso aos recursos da API, reforçando, assim, a salvaguarda dos dados sensíveis manipulados. A acessibilidade da API será priorizada, visando atender a uma ampla diversidade de usuários, independentemente de suas habilidades técnicas, por meio de uma interface intuitiva e eficiente.

Além disso, busca-se explorar e integrar as mais recentes práticas e tecnologias no campo de APIs seguras, considerando não apenas os aspectos técnicos, mas também as implicações éticas e legais. Este trabalho não apenas endereça uma demanda específica de construção de API, mas também busca contribuir para o avanço do conhecimento e aplicação de metodologias seguras e acessíveis no desenvolvimento de sistemas distribuídos na contemporaneidade, impulsionando a inovação e a inclusão digital.



Introdução

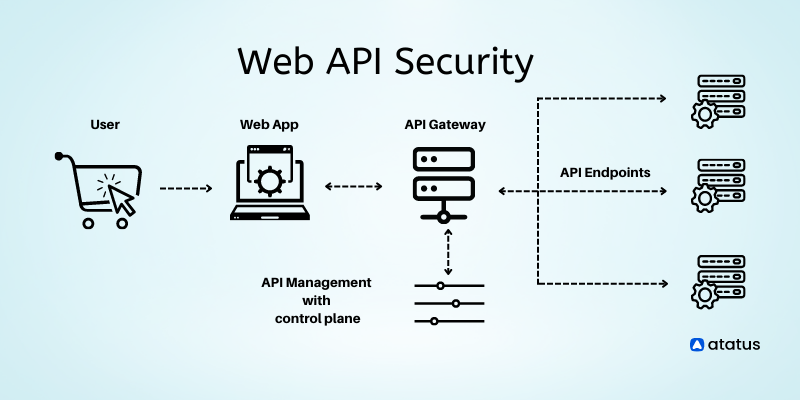
No cenário contemporâneo da revolução digital, a interconexão e interdependência de sistemas, aliada à crescente quantidade de dados disponíveis, impõem a necessidade inquestionável de desenvolver soluções tecnológicas que sejam simultaneamente seguras, acessíveis e eficientes. Nesse contexto, a construção de APIs (Interfaces de Programação de Aplicações) assume um papel central, servindo como a espinha dorsal para a troca de informações entre diferentes aplicações e serviços.



O presente trabalho visa abordar de maneira abrangente o desafio de conceber e implementar uma API segura e acessível, cuja autenticação seja uma salvaguarda robusta, capaz de extrair dados de um dataset específico hospedado no renomado site Kaggle. A escolha deste site como fonte de dados não apenas reflete a sua vasta coleção de conjuntos de dados, mas também destaca a necessidade de adotar práticas seguras na manipulação dessas informações valiosas.

A segurança da informação é um imperativo inegociável na era digital, e a construção de uma API que cumpra com os mais elevados padrões de segurança é um desafio complexo, mas essencial. A autenticação, como componente crítico desse processo, assume um papel proeminente na garantia de que apenas usuários autorizados possam acessar e manipular os dados da API, protegendo-os contra ameaças potenciais.

Paralelamente à segurança, a acessibilidade é um fator determinante na eficácia e na utilidade de uma API. A construção de uma interface que seja intuitiva e eficiente, tornando-a acessível a um amplo espectro de usuários, é uma meta intrínseca a este trabalho. A busca pela acessibilidade não se restringe apenas à usabilidade, mas também à eficiência na transmissão e recepção de dados, considerando a diversidade de contextos e requisitos dos usuários.



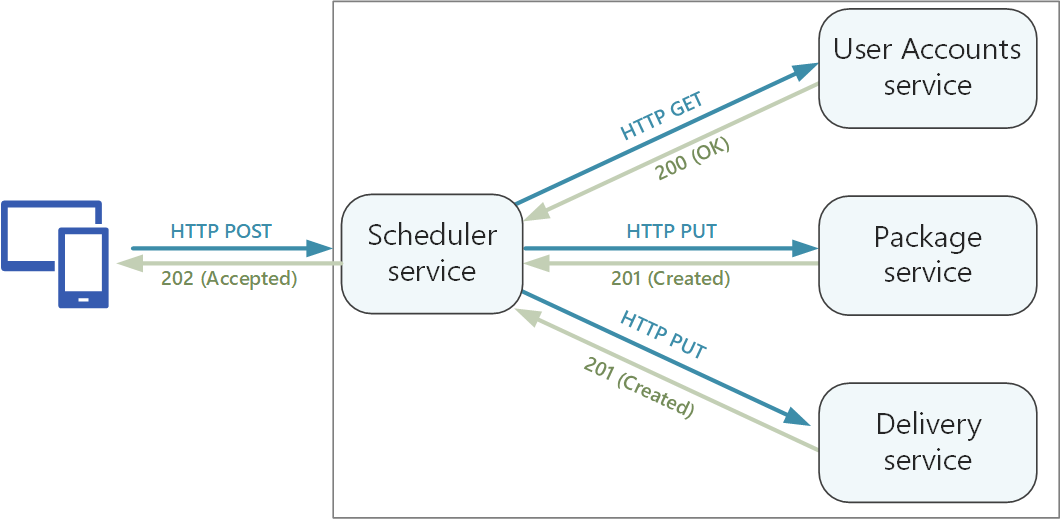
Dentro desse contexto amplo e desafiador, este trabalho busca não apenas realizar a implementação técnica da API, mas também discutir as práticas de segurança adotadas, explorar estratégias para tornar a interface acessível e, por fim, contribuir para a evolução das metodologias de desenvolvimento de APIs seguras e inclusivas. A construção dessa API não é apenas uma resposta a uma demanda específica, mas uma contribuição para o avanço da segurança e acessibilidade no desenvolvimento de sistemas distribuídos na era da informação.

Fundamentos dos conceitos envolvidos na API Kaggle

**API (Interface de Programação de Aplicações):**

Definição: Uma API é um conjunto de regras e protocolos que permitem a interação entre diferentes softwares, possibilitando a comunicação e a troca de dados de forma eficiente.

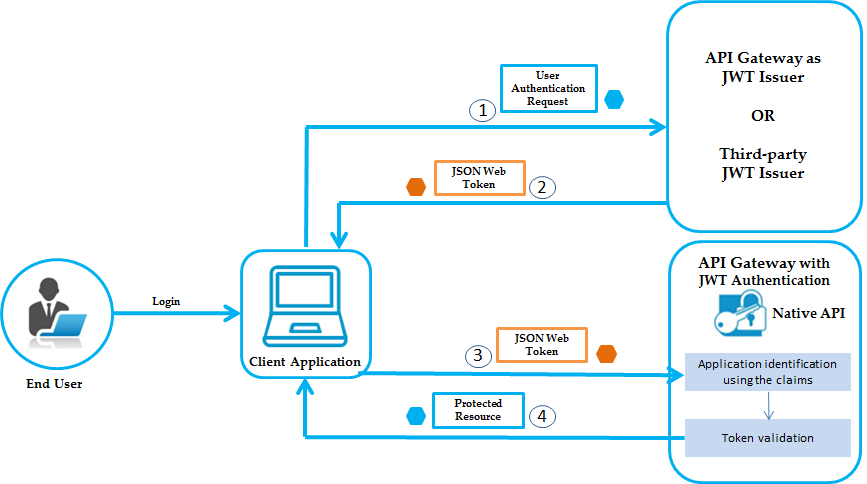
Importância: As APIs desempenham um papel fundamental na integração de sistemas, facilitando a interoperabilidade entre aplicações e serviços diversos.



**Segurança da API:**

Definição: A segurança da API refere-se às práticas e medidas adotadas para proteger os dados e os recursos manipulados pela interface, garantindo a confidencialidade, integridade e autenticidade das informações.

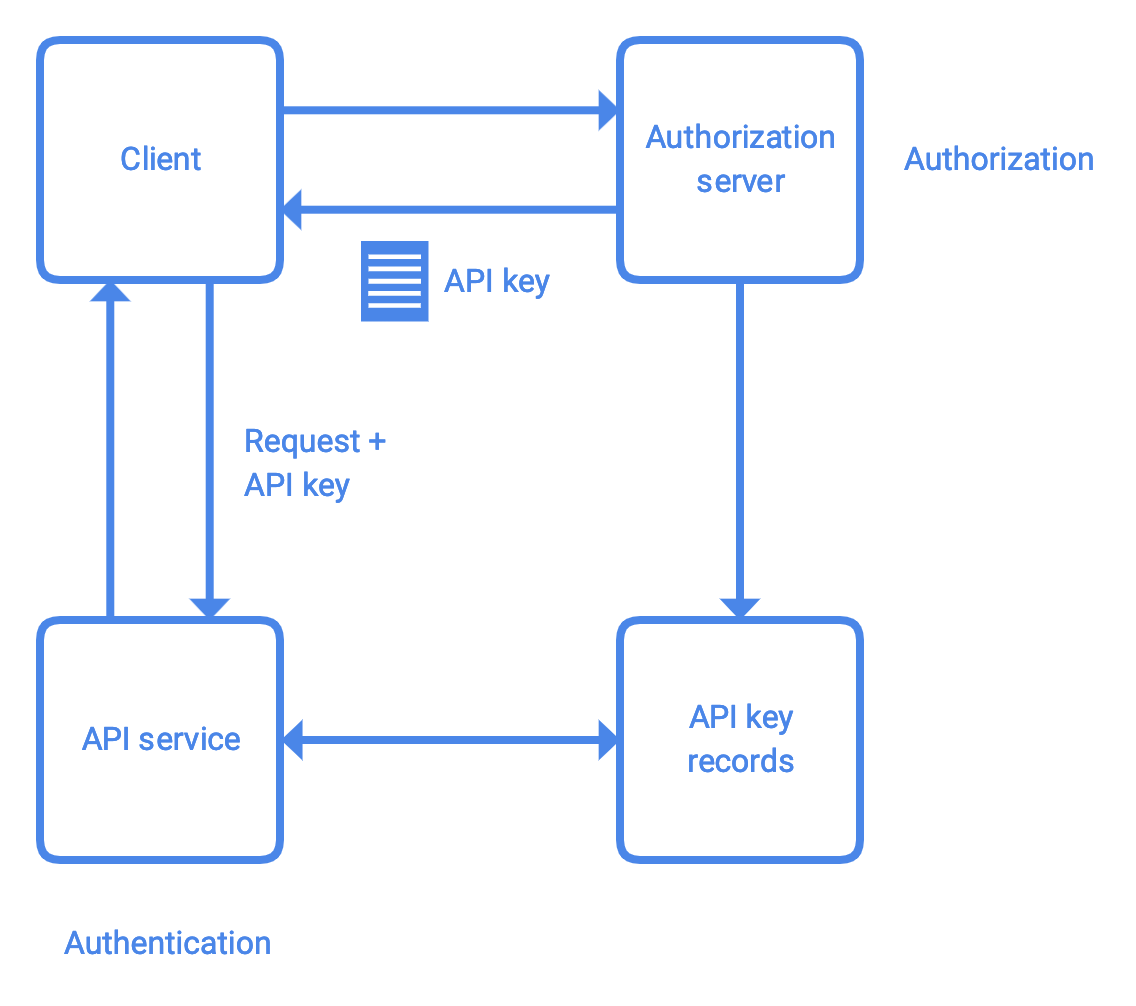
Elementos-chave: Autenticação, autorização, criptografia e gestão de tokens são aspectos essenciais para fortalecer a segurança da API.



**Autenticação:**

Definição: A autenticação é o processo de verificar a identidade de um usuário ou sistema que tenta acessar a API. Isso assegura que apenas usuários autorizados tenham permissão para utilizar os recursos da interface.

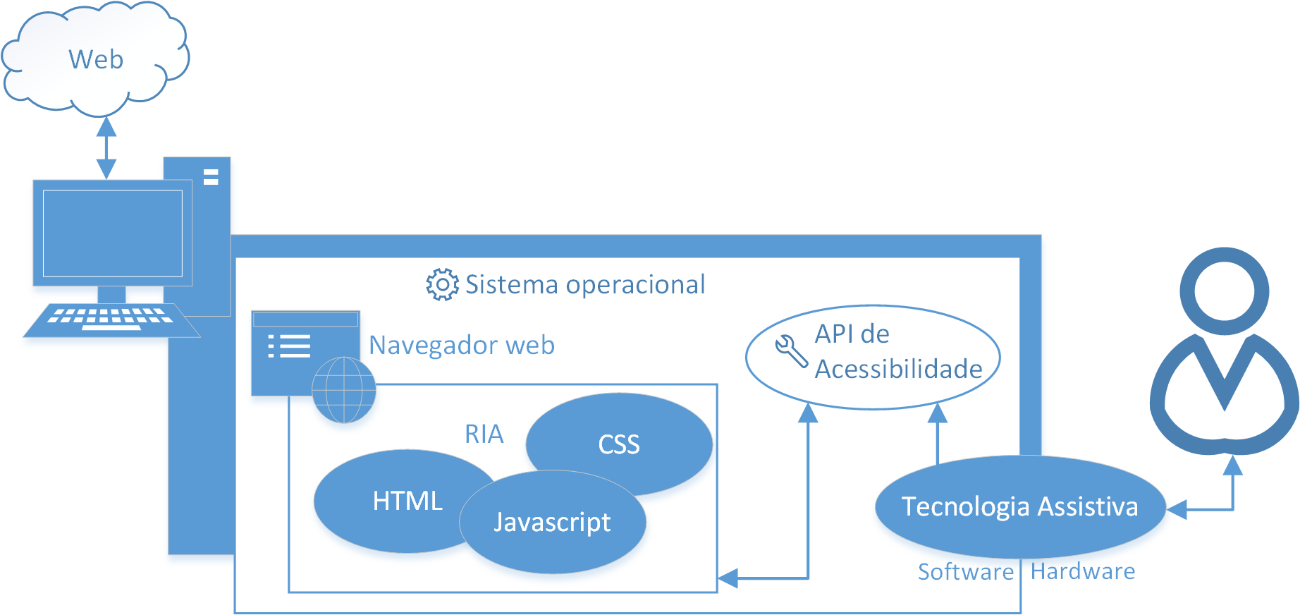
Métodos comuns: Autenticação baseada em token, OAuth, e certificados digitais são abordagens frequentemente empregadas para validar identidades.



**Acessibilidade:**

Definição: Acessibilidade em APIs refere-se à capacidade de disponibilizar informações e funcionalidades de forma compreensível e utilizável por diferentes perfis de usuários, independentemente de suas habilidades técnicas ou limitações.

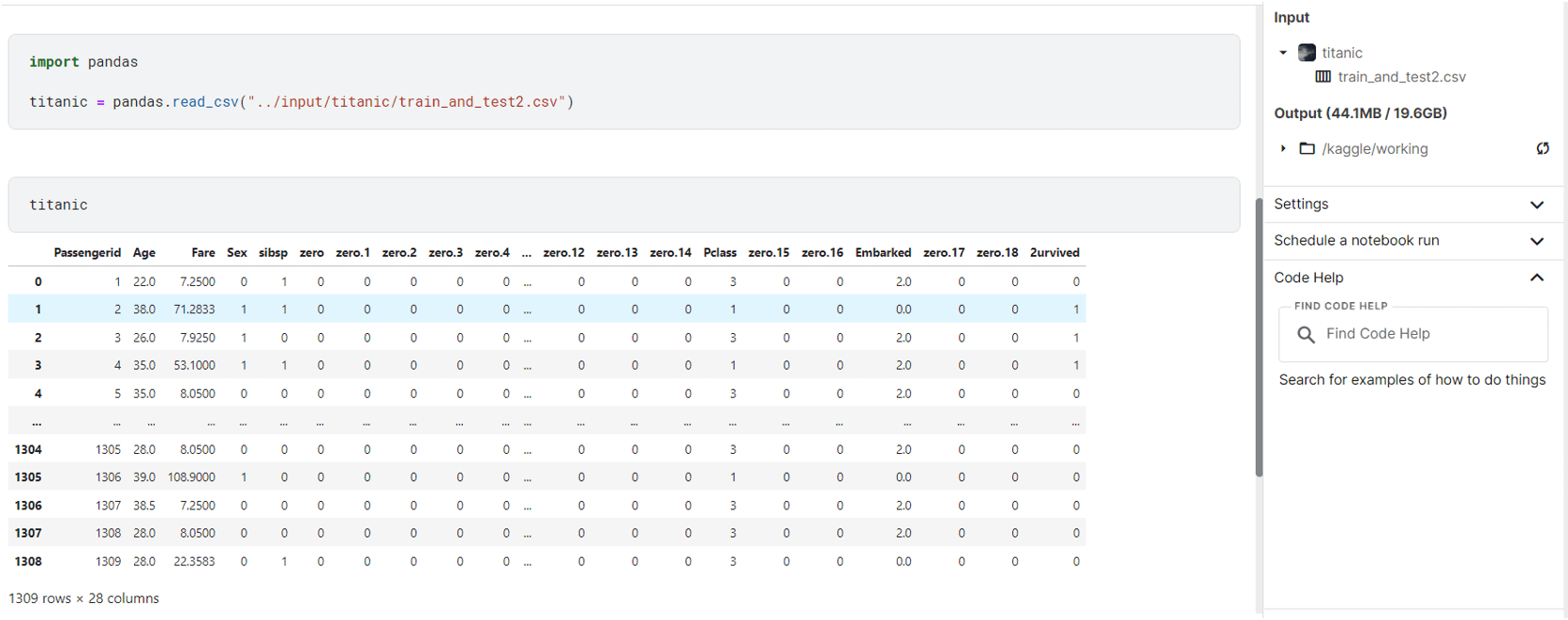
Considerações: A estrutura da API, a documentação clara e o design intuitivo são fatores cruciais para garantir a acessibilidade.



**Dataset e Kaggle:**

Definição: Um dataset é um conjunto de dados estruturados que podem ser analisados e interpretados. Kaggle é uma plataforma que hospeda uma variedade de datasets e promove desafios e competições de ciência de dados.

Importância: Utilizar datasets do Kaggle oferece uma rica fonte de informações para desenvolvimento e teste de aplicações, sendo crucial entender como extrair e manipular esses dados de maneira eficaz e ética.

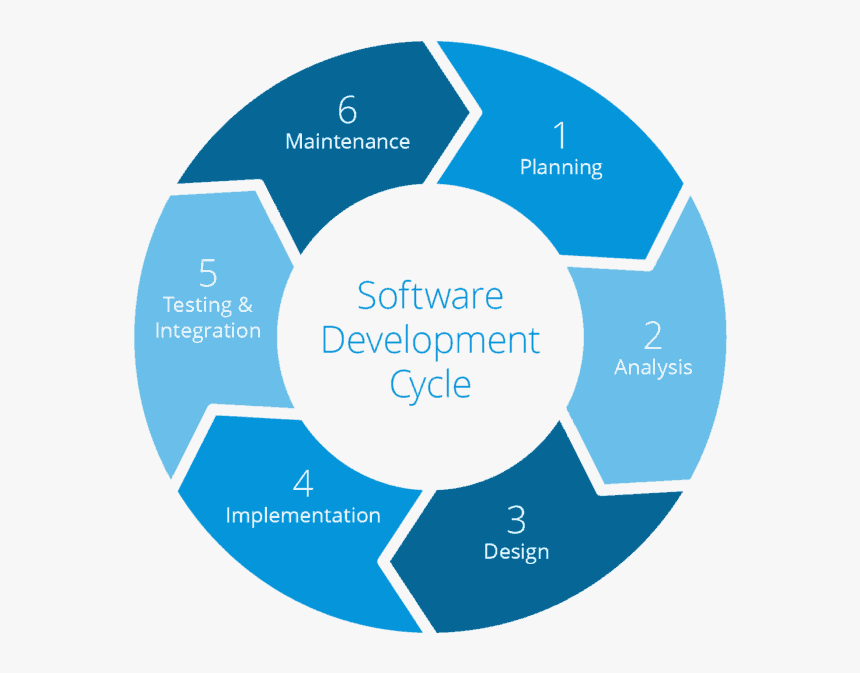


**Práticas de Desenvolvimento Seguro:**

Definição: São diretrizes e técnicas empregadas no desenvolvimento de software para mitigar riscos de segurança desde a concepção até a implementação.

Exemplos: Revisões de código, testes de segurança, e a aplicação de princípios como o princípio do menor privilégio são práticas essenciais.

Ao compreender e aplicar esses fundamentos, será possível embasar de forma sólida o desenvolvimento da API proposta, assegurando não apenas sua segurança, mas também sua acessibilidade e eficácia na manipulação de dados provenientes do Kaggle.



Plano de desenvolvimento da aplicação

**Análise de Requisitos:**

Identificação e Documentação: Levantamento detalhado dos requisitos funcionais e não funcionais da API, com especial atenção aos aspectos de segurança e acessibilidade.

Definição do Dataset: Seleção específica do dataset do Kaggle que será utilizado como fonte de dados.

Diagrama

Descrição gerada automaticamente

**Arquitetura da API:**

Especificação da Arquitetura: Definição da estrutura da API, incluindo endpoints, métodos HTTP e formato dos dados (JSON).

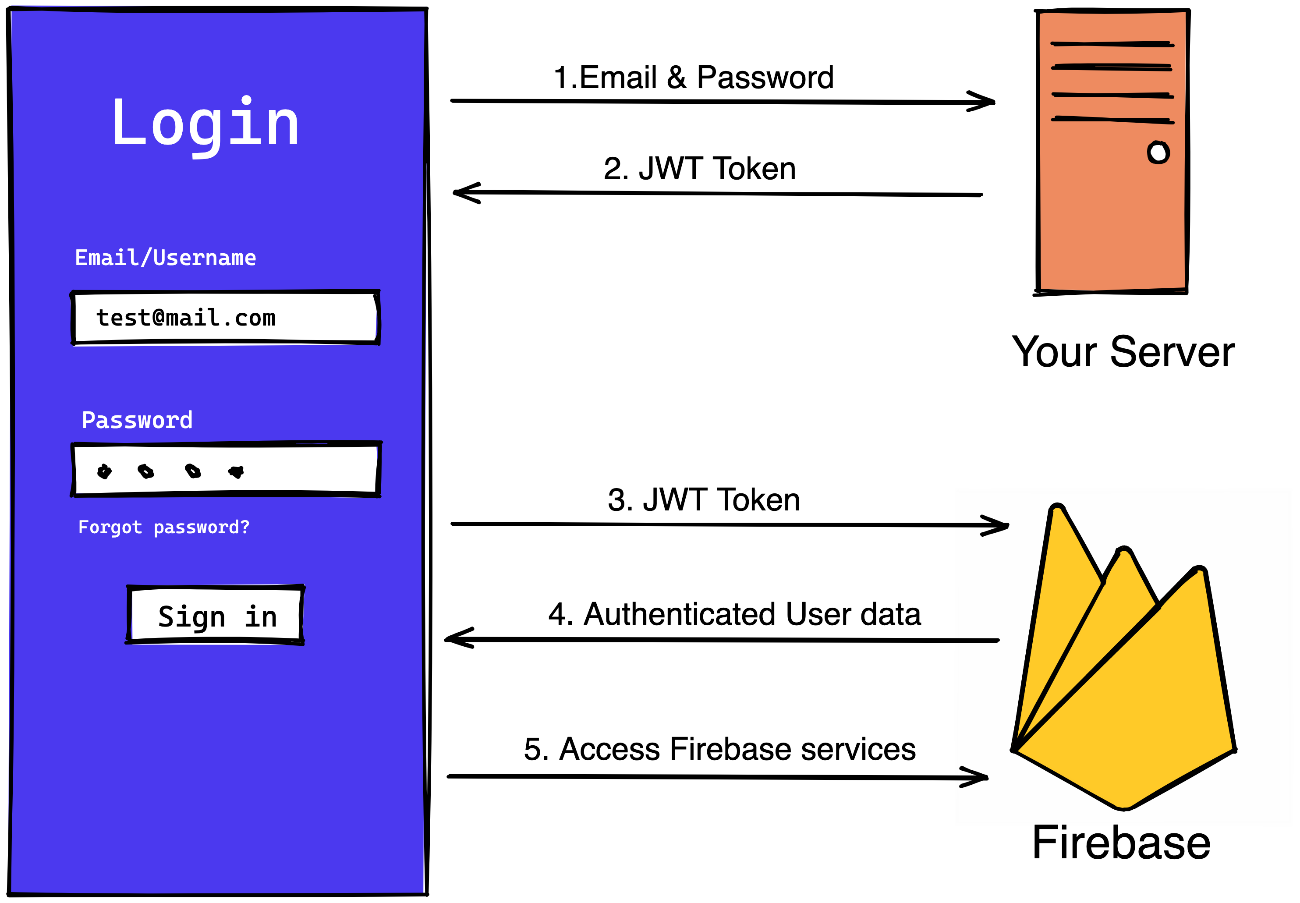
Escolha de Tecnologias: Seleção das tecnologias mais adequadas para implementação, considerando frameworks, bibliotecas e ferramentas de segurança.



**Segurança da API:**

Implementação de Autenticação: Desenvolvimento de um sistema de autenticação robusto, utilizando métodos como tokens JWT (Tokens), garantindo a validação adequada dos usuários.

Controle de Acesso: Definição de políticas de autorização para controlar o acesso aos recursos da API com base nos perfis e privilégios dos usuários.



**Acessibilidade:**

Documentação Clara e Concisa: Criação de documentação abrangente e fácil de entender, incluindo exemplos práticos de uso da API e detalhes sobre os dados retornados.

**Integração com Kaggle:**

Configuração da API Kaggle: Estabelecimento de conexão segura e autenticada com a API do Kaggle para a extração de dados.

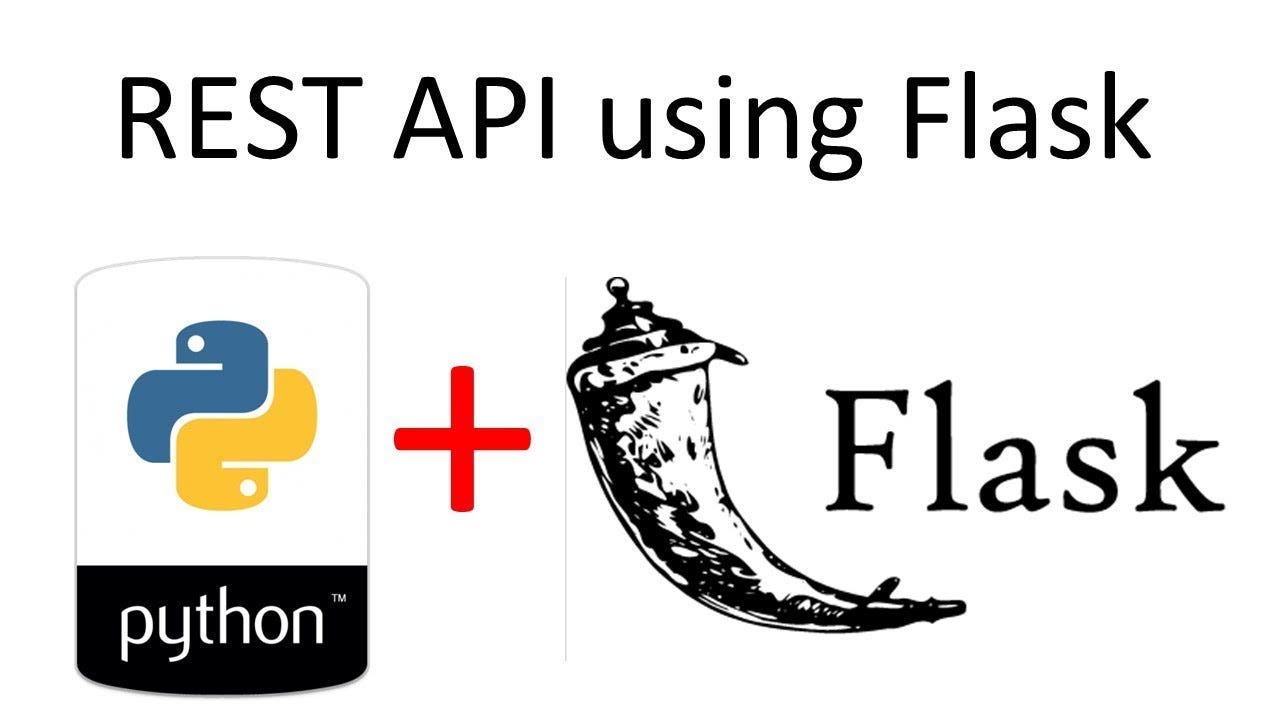
Manipulação de Dados: Desenvolvimento de métodos eficientes para manipulação e processamento dos dados provenientes do Kaggle, adaptando-os ao formato desejado pela API.



**Desenvolvimento Incremental:**

Implementação por Módulos: Desenvolvimento por etapas, implementando funcionalidades básicas antes de avançar para recursos mais complexos, facilitando testes e correções ao longo do processo. Utilizamos a biblioteca Flask do Python para fazer esse processo, definindo rotas para cada funcionalidade!

Revisões Contínuas: Realização de revisões de código periódicas para garantir a conformidade com as práticas de segurança e acessibilidade definidas.



**Testes:**

Testes de Unidade e Integração: Implementação de testes automatizados para garantir a funcionalidade individual de cada componente da API e a integração eficaz entre eles.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Testes de Segurança: Realização de testes de segurança, incluindo análise de vulnerabilidades e avaliação da resistência a possíveis ataques.

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**Documentação Final:**

Elaboração de Documentação Completa: Finalização e aprimoramento da documentação da API, detalhando todos os aspectos relevantes para desenvolvedores e usuários.

Guia de Utilização: Criação de um guia prático para facilitar a integração de desenvolvedores externos à API. O mesmo se encontra no GitHub.

Link:

**Implantação e Monitoramento:**

Implantação Gradual: Publicação controlada da API em ambiente de produção, monitorando seu desempenho e identificando possíveis problemas.

Estabelecimento de Monitoramento: Configuração de ferramentas de monitoramento contínuo para acompanhar a segurança, acessibilidade e performance da API em tempo real.

**Avaliação e Ajustes:**

Avaliação de Desempenho: Análise do desempenho da API em ambiente de produção, identificando áreas de melhoria.

Ajustes Iterativos: Realização de ajustes com base nos feedbacks recebidos, visando aprimorar continuamente a segurança e acessibilidade da API.

**Detalhamento do projeto:** O projeto da API foi feito em Python utilizando diversas bibliotecas, segue abaixo a construção do código.



Importação de diversas bibliotecas, sendo elas:

* Flask: Framework web em Python.
* jsonify: Função para converter objetos Python em JSON.
* request: Permite acessar dados enviados por um cliente (como dados de formulário).
* pandas as pd: Biblioteca para manipulação e análise de dados.
* JWTManager, jwt\_required, create\_access\_token, get\_jwt\_identity: Funções relacionadas à autenticação JWT para Flask.



Cria uma instância do aplicativo Flask.

Texto

Descrição gerada automaticamente

* Define a chave secreta para geração de tokens JWT.
* Ativa a lista negra para invalidação de tokens.
* Configura a lista de verificação para invalidar somente tokens de acesso.



* valid\_tokens: Dicionário para armazenar tokens de usuários autenticados.
* data: Carrega um arquivo CSV contendo dados de salários no futebol.

Texto

Descrição gerada automaticamente

* Rota /login (Método POST): Autentica um usuário e fornece um token JWT válido.
* Rota /logout (Método DELETE): Invalida o token JWT de um usuário autenticado.

Texto

Descrição gerada automaticamente

* Rota /protected (Método GET): Exemplo de rota protegida por autenticação JWT.
* Rota /dados (Método GET): Retorna os dados carregados do arquivo CSV.
* Rota /api/dados (Método POST): Adiciona novos dados ao arquivo CSV.

Texto

Descrição gerada automaticamente

* Rota /api/dados/<int:dado\_id> (Métodos PUT, PATCH, DELETE): Atualiza, modifica ou deleta um dado específico do arquivo CSV.

Texto

Descrição gerada automaticamente

* Rota /dados-paginados (Método GET): Retorna dados paginados do arquivo CSV.

Texto

Descrição gerada automaticamente

* Inicia o servidor Flask para executar a aplicação web.

Projeto

Para ter acesso ao bot, você vai precisar rodar o programa APS.py para iniciar o servidor Flask em localhost.

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Após isso acessamos o Postman para ter uma melhor visualização e manipulação dos dados.

Interface gráfica do usuário, Aplicativo, Teams

Descrição gerada automaticamente

Com os métodos já criados basta iniciarmos o método “Login” com as credenciais designadas no código APS.py.

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

Após fazer isso ele irá retornar o token de acesso para os demais métodos

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Com isso vamos acessar o método “Protected” e mudar o token na Key “Authorization”, o mesmo irá retornar se a rota está protegida ou não.

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Após ele retornar que o token é valido vamos apenas substituir o token na Key “Authorization” dos demais métodos e realizar as suas respectivas funções (GET, POST, PUT, PATCH ou DELETE). Para o método GET vamos apenas enviar a solicitação, para o método POST vamos enviar e definir no Body quais informações serão enviadas seguindo o cabeçario, para o método PUT vamos precisar inserir no URL qual é o bloco que queremos atualizar e definir no Body quais informações serão enviadas seguindo o cabeçario, para o método PATCH vamos fazer semelhante ao método anterior porem no Body vamos mencionar apenas o dado do bloco que queremos atualizar, não o bloco por inteiro e por fim para o método DELETE vamos precisar inserir no URL qual é o bloco que queremos atualizar e enviar para que o mesmo seja apagado!

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Tela de celular com publicação numa rede social

Descrição gerada automaticamente

Caso tente atualizar alguma informação sem utilizar o token gerado no método Login o mesmo irá retornar erro

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Por fim quando for encerrar a utilização apenas envie o método “Logout” o mesmo irá invalidar o token que estiver em sua Key “Authorization”. Para maior segurança caso não seja feito o logout o token utilizado será expirado alguns minutos após sua geração!

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

Relatório com as linhas de código

from flask import Flask, jsonify, request

import pandas as pd

from flask\_jwt\_extended import JWTManager, jwt\_required, create\_access\_token, get\_jwt\_identity

# Inicialização do aplicativo Flask

app = Flask(\_\_name\_\_)

# Configuração do JWT (JSON Web Tokens)

app.config['JWT\_SECRET\_KEY'] = 'super-secret'

app.config['JWT\_BLACKLIST\_ENABLED'] = True

app.config['JWT\_BLACKLIST\_TOKEN\_CHECKS'] = ['access']

jwt = JWTManager(app)

# Criação de variáveis

valid\_tokens = {}

data = pd.read\_csv('data/football\_wages.csv')

# Autentica um usuário e fornece um token JWT válido

@app.route('/login', *methods*=['POST'])

def login():

    username = request.json.get('username', None)

    password = request.json.get('password', None)

    # Verifique aqui as credenciais do usuário (exemplo simplificado)

    if username == 'jordan' and password == '654321':

        access\_token = create\_access\_token(*identity*=username)

        valid\_tokens[username] = access\_token

        return jsonify({'access\_token': access\_token}), 200

    else:

        return jsonify({'message': 'Credenciais inválidas'}), 401

# Invalida o token JWT de um usuário autenticado

@app.route('/logout', *methods*=['DELETE'])

@jwt\_required()

def logout():

    current\_user = get\_jwt\_identity()

    if current\_user in valid\_tokens:

        del valid\_tokens[current\_user]

        return jsonify({'message': 'Logout realizado com sucesso'}), 200

    else:

        return jsonify({'message': 'Usuário não autenticado'}), 401

# Exemplo de rota protegida por autenticação JWT

@app.route('/protected', *methods*=['GET'])

@jwt\_required()

def protected():

    return jsonify({'message': 'Rota protegida!'})

# Retorna os dados carregados do arquivo CSV

@app.route('/dados', *methods*=['GET'])

@jwt\_required()  # Esta rota agora exige autenticação JWT

def get\_dados\_protegido():

    current\_user = get\_jwt\_identity()

    return jsonify(data.to\_dict(*orient*='records'))

# Adiciona novos dados ao arquivo CSV

@app.route('/api/dados', *methods*=['POST'])

@jwt\_required()

def add\_dado():

    current\_user = get\_jwt\_identity()

    global data  # Indica que 'data' será usada globalmente nesta função

    new\_data = request.get\_json()

    new\_data\_df = pd.DataFrame([new\_data])  # Convertendo o novo dado em um DataFrame

    data = pd.concat([data, new\_data\_df], *ignore\_index*=True)  # Usando o método concat para adicionar o novo dado ao DataFrame existente

    return jsonify({'message': 'Dado adicionado com sucesso!'})

# Atualiza um dado específico do arquivo CSV

@app.route('/api/dados/<int:dado\_id>', *methods*=['PUT'])

@jwt\_required()

def update\_dado(*dado\_id*):

    current\_user = get\_jwt\_identity()

    data\_to\_update = request.get\_json()

    data.iloc[*dado\_id*] = data\_to\_update

    return jsonify({'message': 'Dado atualizado com sucesso!'})

# Modifica um dado específico do arquivo CSV

@app.route('/api/dados/<int:dado\_id>', *methods*=['PATCH'])

@jwt\_required()

def modify\_dado(*dado\_id*):

    current\_user = get\_jwt\_identity()

    data\_to\_modify = request.get\_json()

    data.iloc[*dado\_id*] = {\*\*data.iloc[*dado\_id*], \*\*data\_to\_modify}

    return jsonify({'message': 'Dado modificado com sucesso!'})

# Deleta um dado específico do arquivo CSV

@app.route('/api/dados/<int:dado\_id>', *methods*=['DELETE'])

@jwt\_required()

def delete\_dado(*dado\_id*):

    current\_user = get\_jwt\_identity()

    data.drop(*index*=*dado\_id*, *inplace*=True)

    return jsonify({'message': 'Dado deletado com sucesso!'})

# Retorna dados paginados do arquivo CSV

@app.route('/dados-paginados', *methods*=['GET'])

@jwt\_required()

def get\_dados\_paginados():

    current\_user = get\_jwt\_identity()

    page = request.args.get('page', *default*=1, *type*=int)

    items\_por\_pagina = request.args.get('items\_por\_pagina', *default*=2, *type*=int)

    paginated\_data = data.iloc[(page - 1) \* items\_por\_pagina:page \* items\_por\_pagina]

    return jsonify(paginated\_data.to\_dict(*orient*='records'))

# Execução do aplicativo

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(*debug*=True)

Bibliografia

Python & APIs: A Winning Combo for Reading Public: https://realpython.com/python-api/. Acesso em 25/09/2023.

Kaggle: https://www.kaggle.com/. Acesso em 25/09/2023.

4 essential characteristics of successful APIs: https://opensource.com/article/21/5/successful-apis. Acesso em 05/10/2023.

Python API Tutorial: Getting Started with APIs: ://www.dataquest.io/blog/python-api-tutorial/. Acesso em 25/09/2023.

Como adicionar autenticação JWT no FastAPI: ://www.freecodecamp.org/news/how-to-add-jwt-authentication-in-fastapi/.Acesso em 05/10/2023.

Tabela

Descrição gerada automaticamente

Tabela

Descrição gerada automaticamente