PUC Minas
A B A K Ó S
Instituto de Ciências Exatas e Informática

Estudo de caso de uma API em tempo real para consultas de modelos pré-treinados de aprendizado de máquina*

Case study of a real-time API for querying pre-trained machine learning models

Fernando Antônio dos Santos Bretz[1]
Igor Palhares Reis[2]
Leonardo Vilela Cardoso[3]

Resumo

Os modelos de aprendizado de máquina ganham cada vez mais popularidade para resolução de problemas e análises de dados. Porém, tais modelos apresentam um grande problema relacionado ao tempo de resposta para atender as requisições dos usuários. Os avanços da tecnologia, no entanto, proporcionam recursos para resolução deste problema. Uma estratégia é a utilização de uma Interface de Programação de Aplicação (*API*), pois além de proporcionar uma melhor interpretação dos dados, facilita a comunicação entre o modelo e o usuário. Contudo, esse artigo propõe o desenvolvimento de uma aplicação visando abordar a integração entre uma *API* e um modelo de aprendizado de máquina, interface interativa para o usuário, segurança dos dados e o número de acessos simultâneos.

Palavras-chave: Modelos de Aprendizado. Tempo de Resposta. *API*. Desenvolvimento. Integração.

*Trabalho de Conclusão de Curso II

[1]Graduando em Sistemas de Informação PUC Minas, Brasil– fasbretz@sga.pucminas.br
[2]Graduando em Sistemas de Informação PUC Minas, Brasil– igor.palhares@sga.pucminas.br
[3]Doutorando em Análise de Dados e Recuperação de Informação, Brasil– lvcardoso@sga.pucminas.br

Abstract

Machine learning models are gaining more and more popularity for problem solving and data analysis. However, such models have a major problem related to the response time to meet user requests. Advances in technology, however, have provided resources to resolve this problem. One strategy is the use of an Application Programming Interface (*API*), because in addition to providing a better interpretation of the data, it facilitates communication between the model and the user. However, this article proposes the development of an application aimed at addressing the integration between an *API* and a machine learning model, interactive user interface, data security and the number of simultaneous accesses.

Keywords: Learning Models. Response time. *API*. Development. Integration.

1 INTRODUÇÃO

No cenário atual, os modelos de aprendizado de máquina são utilizados no mercado com o objetivo de automatizar análise de dados e solucionar inúmeros problemas tecnológicos e econômicos (LUDERMIR, 2021). Porém, um grande problema está relacionado a lentidão no tempo de resposta desses modelos.

Uma das soluções para resolver o problema é realizar a integração do modelo de aprendizagem a uma Interface de Programação de Aplicação (*API*), a qual serve como meio de comunicação e uma melhor interpretação dos dados. No entanto, muitas pessoas que detêm o conhecimento na área ou que possui algum interesse em programação, podem apresentar dificuldades ao realizar o desenvolvimento de uma aplicação (ARIMOTO; OLIVEIRA, 2019).

Com o intuito de facilitar a transmissão de dados e garantir maior robustez nas aplicações, uma boa estratégia é utilizar tecnologias de gestão, como a plataforma *Kafka*. Esse é um meio de garantir que todos os usuários tenham acesso às informações em tempo real, assim, proporcionando maiores desempenhos em situações maiores e mais complexas nas quais os fluxos de dados derivados de diversas fontes sejam processados e entregues para o usuário de maneira correta.

Ao desenvolver um *software*, uma das preocupações do programador é manter o aplicativo seguro após o seu lançamento para não ocorrer vazamentos de dados. Uma das soluções para aumentar a segurança é a implementação de metodologias de autenticação, no entanto, tem sido um enorme desafio, pois algumas vezes são realizadas de maneira incorreta, acarretando sérios riscos ou falhas na codificação para as aplicações (MONTANHEIRO; CARVALHO; RODRIGUES, 2017).

Outro fator muito importante é a utilização de interfaces gráficas para garantir uma melhor experiência e interação com o usuário. Por isso, as interfaces precisam ser intuitivas, ou seja, conter atributos, textos e imagens que facilitam o entendimento do usuário (LEMES, 2018).

Com base no cenário apresentado, este trabalho tem como objetivo desenvolver uma *API* com controle em tempo real, contendo uma interface gráfica, capacidade de suportar diversos acessos simultâneos aumentando a robustez, garantir a segurança das informações e integrando a qualquer modelo de aprendizado de máquina treinado.

Assim, utilizando a arquitetura, as tecnologias e os conceitos apresentados nesse trabalho, o desenvolvedor poderá criar uma *API* que seja escalável para situações mais complexas, com fluxos de dados contínuos, que possua uma camada de segurança que irá garantir a integridade dos dados, além de integrar a *API* desenvolvida a um modelo de aprendizado de máquina pré-treinado, sem que ele possua conhecimento nessa área, apenas com os conhecimentos de desenvolvimento.

As próximas seções estão organizadas como descrito a seguir. Na seção 2 contém a revisão bibliográfica. Na Seção 3 está descrito os trabalhos relacionados ao tema do projeto. A Seção 4 exibe a metodologia relacionada ao desenvolvimento da solução proposta, desde as

tecnologias utilizadas, testes e experimentos, até que a aplicação esteja pronta para a utilização e coleta dos dados. A seção 5 apresenta os resultados obtidos pelo trabalho, realizando os testes e implementações. A seção 6 está a conclusão do projeto.

2 REVISÃO BIBLIOGRÁFICA

Nesta seção, são apresentados assuntos sobre o projeto separados por temas relacionados a este trabalho. O tema sobre *API* é retratado na Subseção 2.1. A Subseção 2.2. aborda uma plataforma utilizada para transmissão de dados *Apache Kafka*. A Subseção 2.3. contém assuntos relacionados ao *Django* e o *Flask*. Finalmente, a Subseção 2.4. aborta o tema modelo de aprendizagem. Esse tema está subdivido em treinamento supervisionado e treinamento não supervisionado.

2.1 API

A sigla *API* significa *Application Programming Interface* (Interface de Programação de Aplicação), uma *API* é um conjunto de normas que possibilita a comunicação entre aplicações através de uma série de padrões e protocolos (Amazon Web Services, 2022).

A arquitetura de uma *API* pode ser explicada da seguinte forma: De um lado teremos uma aplicação que será o cliente, que irá enviar uma solicitação e do outro teremos a aplicação que envia a resposta que vai ser o servidor. Existem diferentes formas de se construir uma *API*, por isso, podem ser classificadas de acordo com a finalidade ou em que momento foram construídas. Contudo, cada tipo de arquitetura pode atender uma solução específica e com o passar do tempo, a arquitetura das *API's* foram evoluindo e surgiram novas formas de serem desenvolvidas.

Essa interface se tornou muito popular nos últimos anos para transferir dados entre aplicações *WEB*, possuindo diversas características de melhores práticas de codificação, segurança, aceleramento no desenvolvimento de *software*, entre outras. Por estes motivos, torna-se uma ferramenta excelente para ser usada.

Neste trabalho iremos utilizar a arquitetura *API Rest*, esse conceito é o mais popular por englobar melhor diversos cenários possíveis de desenvolvimento. O *REST* no nome significa Transferência Representacional de Estado, essa arquitetura contém características que permitem o desenvolvimento da aplicação com rotinas padronizadas. O modelo Cliente-Servidor é um exemplo, pois tem como funcionalidade respeitar a separação das atividades da interface do usuário e do banco de dados, usando diversas abstrações de dependência entre partes e sem impacto ou quebra de contrato (Amazon Web Services, 2022).

Dentro dessa arquitetura existem algumas funções utilizadas para manipular ou consultar dados, alguns exemplos de nomenclatura das funções são: *GET*, *PUT*, *POST* e *DELETE*.

Existem outros tipos de métodos e funcionalidades, mas estes são os mais utilizados pelo cliente que irá consumir um dado, atualizar, deletar ou passar alguma informação para o servidor (Amazon Web Services, 2022).

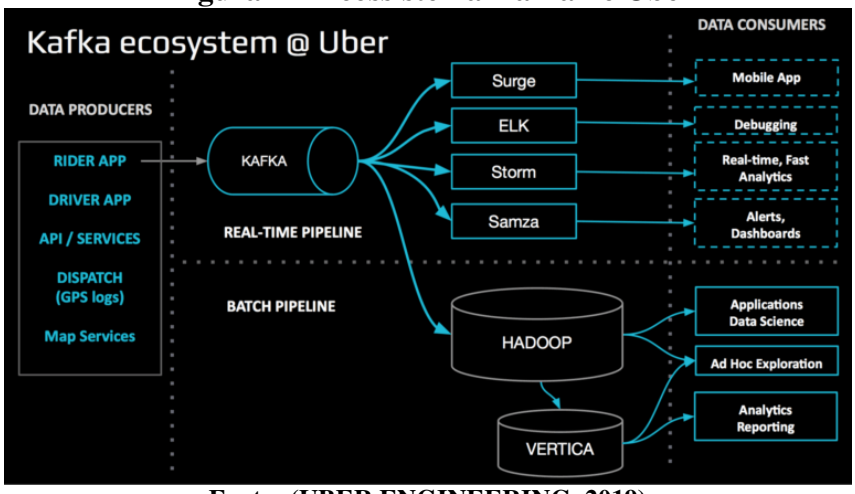
2.2 Apache Kafka

O *Apache Kafka* trata-se de uma plataforma distribuída que tem como finalidade a transmissão de dados, também pode-se dizer que ele é um sistema de mensageira, pois através dela é possível publicar, atualizar, armazenar e processar fluxos de dados em tempo real. A plataforma foi criada com a função de aceitar fluxos de dados vindo de diversas fontes em simultâneo, por isso é possível integrar diversos tipos de aplicações com o Kafka.

A arquitetura do *Kafka* está dividida em: tópicos que são uma forma de organizar dados recebidos, o produtor que irá gravar os fluxos de eventos em um ou mais tópicos e o consumidor que irá ler os fluxos de dados em um ou mais tópicos (RED HAT, 2017).

A utilização do *Kafka* está relacionada ao enorme volume de dados que são gerados e precisam ser enviados para diversas aplicações em tempo real. Visando este cenário, grandes empresas como a *Netflix*, *Spotify*, *Uber*, *LinkedIn* e o *Twitter* utilizam o *Kafka* em suas aplicações para facilitar este processo (LUBY, 2021). A Figura 1 ilustra o funcionamento do Kafka dentro da aplicação *Uber*:

Figura 1 – Ecossistema Kafka no Uber



Fonte: (UBER ENGINEERING, 2019)

Em um aplicativo como o da *Uber*, que possui 30 milhões de usuários ativos, existem milhares de dados sendo gerados e consumidos entre seus usuários e os seus sistemas internos (CNN BRASIL, 2021). O *Kafka* funciona como um conector entre os produtores dos dados que são: aplicativos dos motoristas, passageiros e outros micro-serviços com os consumidores dos dados que são seus serviços internos. É essencial nessa arquitetura para que todas as partes possam estar conectadas e obter os dados gerados pelos produtores de forma rápida e atualizada. Sem o *Kafka*, poderiam acontecer inúmeras falhas nos processos e nas aplicações internas devido a utilização de dados desatualizados. Outro problema está relacionado ao desempenho na comunicação entre todas as partes do sistema devido ao elevado número de dados e requisições

feitas pelos produtores e consumidores.

2.3 Django e Flask

Os *frameworks Django* e *Flask* são os mais populares para construir *API's* utilizando a linguagem de programação *Python*, porém possuem algumas características que o diferenciam um do outro. O *framework Django* usa a arquitetura *model-template-view*, que simplifica o processo no desenvolvimento de uma aplicação WEB e é muito utilizado para construir aplicações WEB completas de pequeno ou grande porte. O *Flask* é um *framework* voltado mais para pequenas aplicações, seu design é leve e modular, isso facilita para quem está iniciando no desenvolvimento de *API's*.

Ambos possuem pontos fortes quando se trata de desenvolvimento de *API*, porém o *Django* possui uma vantagem em relação ao *Flask*, que é o mapeador relacional de objeto. Este recurso vincula objetos a tabelas presentes no banco de dados. Então, por seu leve nível de complexidade, conclui-se que o *Flask* pode ser a melhor escolha ao desenvolver pequenas aplicações ou para ser utilizado por usuários sem muita experiência ou pouco conhecimento nesse segmento. Contudo, o *Django* pode ser usado para sistemas mais complexos onde é preciso um sistema mais robusto que possa lidar com muitos requisitos quanto ao seu desempenho e funcionalidade (H. FATIMA, 2018).

2.4 Inteligência Artificial

A inteligência artificial (IA) surgiu há pouco tempo se comparado com outras ciências, seu início foi após a Segunda Guerra Mundial e hoje essa ciência possui vários subcampos dentro dela. A IA está evoluindo consideravelmente nesses últimos anos, sendo utilizada em diversos momentos no nosso dia a dia, mesmo sem termos essa percepção. Existem diversas formas de se definir o que é uma IA, filosófica, técnica, entre outras. Podemos dizer que é uma ciência que busca através da computação construir algoritmos ou dispositivos que simulem a capacidade do ser humano de pensar e resolver problemas.

A IA está diretamente ligada aos modelos de aprendizado de máquina, pois automatizam os processos buscando coletar, organizar e analisar dados de forma autônoma. Os modelos existentes são divididos em aprendizado supervisionado, aprendizado não-supervisionado, aprendizado semi-supervisionado e Aprendizado por reforço (TOTVS, 2022).

O desenvolvedor ao escolher um modelo de aprendizagem, necessita analisar bem a situação em que o modelo atuará, muitas vezes o desenvolvedor não precisará criar um modelo novo, mas apenas realizar uma consulta em um modelo pronto treinado.

2.4.1 Treinamento Supervisionado

O treinamento supervisionado é uma tarefa de aprendizado de máquina em que o modelo de IA aprende a partir de resultados pré-definidos. Vale ressaltar que esses dados precisam ser disponibilizados por meio de uma base de dados. Esses valores pré-definidos servem como uma espécie de guia/supervisão, para a calibragem do modelo. Assim, o modelo irá utiliza-los como referência para aumentar a sua taxa de acertos com bases nos resultados que estão diferentes do esperado (Tainá Freitas, 2019).

2.4.2 Treinamento não Supervisionado

O treinamento não supervisionado não usa os resultados pré-definidos, ou seja, o modelo irá aprender por conta própria. Um conjunto de dados sem uma identificação é passado para o algoritmo, este algoritmo irá analisar os dados e assim, realizar o seu treinamento por conta própria. O processo de aprendizagem da máquina é mais demorado comparado com o treinamento supervisionado, por isso não é muito utilizado (Tainá Freitas, 2019).

3 TRABALHOS RELACIONADOS

Vainikka (2018) em seu artigo apresenta como é o desenvolvimento de forma detalhada de uma *API Rest* usando o *framework Django*. Destaca também um pouco sobre as dificuldades que os desenvolvedores têm ao escolher o *framework* certo para o seu projeto, pois nos dias atuais existem uma grande gama disponíveis para serem utilizados. O artigo aborda como é feita a escolha desses *frameworks* e como é iniciado o desenvolvimento de projetos utilizando eles. O estudo é voltado para a demanda da *Esports League Filandesa* (FEL), que irá dividir seu site em dois serviços separados para melhorar o gerenciamento desses módulos, estudando a atual estrutura do site e uma maneira de como poderiam solucionar diversos problemas aplicando uma nova estrutura com outras tecnologias. Com a utilização de tecnologias novas como *Django* e *React*, foi possível aprender como selecionar o melhor *framework* e qual estrutura é melhor para determinada situação visando as boas práticas, performance e alcance do objetivo do projeto. O FEL aprovou essas alterações, e a equipe de desenvolvedores viram que as ferramentas escolhidas para a nova estrutura eram melhores que as anteriores. O projeto não conseguiu ser concluído completamente até o fim da tese por falta de tempo devido ao grande nível de demanda dos desenvolvedores.

Conceição (2019) exibe uma forma diferente de utilizar uma aplicação de reconhecimento facial. Utilizando o *Django*, foi desenvolvido uma aplicação completa, com um painel de administração e com suporte a uma *API*. Os algoritmos de reconhecimento facial estão cada vez mais presentes em nosso dia a dia, em diversos ramos e atividades. Com o uso de diversas

tecnologias *Open Source* é possível agregar diversos recursos às aplicações que contém estes algoritmos. Utilizando diversas tecnologias do mercado, tanto na parte de algoritmos de reconhecimento facial, quanto para o desenvolvimento de aplicações como *Django*, *Python*, C++, *JSON* e *MYSQL*, foi possível criar uma aplicação WEB completa de reconhecimento facial que capta imagens e analisa características presentes na face e uma *API* para que todas essas funcionalidades vindas desse sistema sejam utilizadas por outras aplicações. Essa *API* pode ser útil para diversas questões profissionais, acadêmicas ou pessoais. E pela sua flexibilidade é possível ser integrada com aplicativos *Open Source* ou aplicativos proprietários.

O aprendizado de máquina pode ser aplicado em diversas áreas, e um estudo que nos mostra como isso está cada vez mais presente, é no artigo do Nogueira (2019). O foco principal do projeto é realizar previsões de resultados de jogos de tênis, essa necessidade veio devido ao crescimento do mercado de apostas desportivas e o tênis foi escolhido por ser um dos mais populares quando se trata de apostas. Durante o estudo foram feitos diversos experimentos e testes em algoritmos, o modelo de regressão logística foi o que apresentou uma melhor precisão em seu resultado. Ele conseguiu uma taxa de acerto de 68% e um retorno de investimento de 4,32% nos jogos do *US Open* de 2019. Foram também criadas *API's* para integrar com o modelo de previsão, além de uma aplicação web para exibir uma comparação entre as previsões do modelo e a probabilidade de sucesso para apostas oferecidas pelas casas de apostas.

Boteju *et al.* (2020) definem um contexto mais abrangente sobre o tema de aprendizagem de máquina e como ele pode ser utilizado em diversas áreas. Ele traz um problema no dia a dia de muitas pessoas, que é monitorar a saúde dos animais de estimação, precisamente dos cães e através de modelos de *Deep Learing*, é possível identificar comportamentos incomuns nos cães de estimação. O cachorro é observado através de uma câmera e o objetivo principal é construir uma aplicação que tenha funcionalidades como analisar caminhada, níveis de repouso, comportamento ao falar, trazer relatórios diários, entre outras. A estrutura de todo a aplicação consiste em sensores, câmeras, *API* para integrar dados entre essas partes e um banco de dados para armazenar as informações. O sistema foi capaz de identificar com sucesso a raça do cachorro analisado e seu comportamento, além de alertar ao usuário quando ocorre alguma anormalidade durante o dia do cachorro.

Lakshan, Silva e Weerakoon (2021), em seu projeto, abordam a implementação de um sistema de previsão de crimes usando ferramentas de *Machine Learning* e algumas técnicas de mineração de dados. Através de uma IA qualquer ação que possa resultar em um furto será prevenida usando padrões previamente analisados. Usando novas tecnologias disponíveis no mercado é possível de fato criar um modelo de IA e realmente colocá-lo em produção, para ser utilizado no dia a dia dos cidadãos. Foi escolhida uma área específica para ser realizada a análise dos dados, essa análise é crucial para entender os padrões de ocorrência de crimes. O artigo também mostra que existem diversos algoritmos já criados para identificar esses padrões, mas alguns podem ser muito caros para serem utilizados. Neste caso, foi utilizado a linguagem *Python* para implementação desse algoritmo. O Google *Collaboratory Notebooks* foi usado para executar o código nos servidores em nuvem do Google. O *Django* e o*Postman* foram

usados para construir e testar a *API REST* criada para se integrar junto a esse algoritmo de previsão de ocorrência de crimes. Os dados utilizados para treinar esse modelo foram retirados do *"Kaggle"*, do site *"Statistics Canada"* e os dados do Censo (2011 e 2016) do site *"Census Profile of the Statistics of the Canada"*. O objetivo principal dos autores foram alcançados, o nível de precisão do seu algoritmo foi de 92%, a interface e *API REST* foi fundamental para a interação do usuário com os dados gerados, e também ajudaram a reduzir o custo de implementação de hardware, a aplicação então pode ser utilizada no sistema atual de previsão de crimes em vários locais simultaneamente.

O Dileep e Thogaru (2021) aplicam o *Machine Learning* de uma forma diferente em outro contexto, ele traz um Sistema de monitoramento da COVID-19 baseado em computação em nuvem usando IOT e a integração com o *Machine Learning*. Durante a pandemia trazida pelo COVID-19, diversas aplicações e estudos foram realizados, existem uma abundância de dados na área da saúde, o que é um fator muito positivo quando se trata de *Machine Learning* e aplicações que sua capacidade de resolução depende muito da quantidade de dados possíveis para aprimorar os seus algoritmos. O sistema foi construído para prever a presença de doenças cardíacas e renais crônicas usando o modelo de rede neural, diabete usando o modelo de classificação *KNearest Neighbors* e detecção de câncer de pele usando o modelo de rede neural convolucional. Hoje existem diversos sensores voltados para IOT e eles foram usados juntos aos algoritmos para detectar essas doenças. Quando existe uma variação dos dados do paciente, o médico é avisado através de uma notificação e por isso o sistema é muito útil para ter um acompanhamento de perto. Vale lembrar, que o sistema evolui junto a tecnologia e seus dados, assim será melhorado com o tempo. O algoritmo teve um acerto de 90%, assim podendo ser considerado para aplicações médicas.

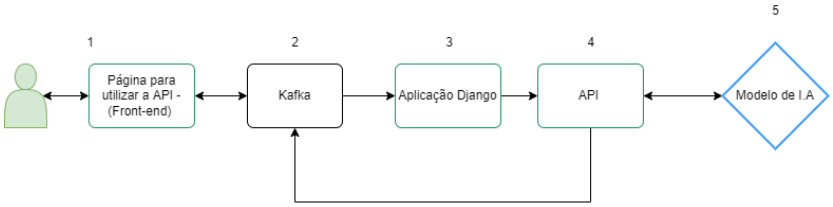
Os Liu e Wang (2021) abordam um problema comum ao desenvolver uma *API*. Este problema está relacionado a falta de padrões nas documentações das *APIs*, assim, impossibilitando a extração de informações ou dificultando na hora do desenvolvimento utilizando *frameworks* para ler a documentação e criar o modelo base da *API*. O *OpenAPI* que é desenvolvido a partir da especificação *Swagger*, descreve a *API REST* no formato *JSON* ou *YAML* legível pela máquina, podendo ser usada para testar a *API REST* automaticamente. O projeto cria uma forma de transformar documentações escritas em *HTML* neste padrão, assim não será necessário o trabalho manual de converter as documentações para o padrão *OpenAPI*. Analisando os resultados foi validado que essa abordagem apresentou resultados positivos, mas mesmo assim, é preciso aumentar a porcentagem de acertos para evitar possíveis erros ao gerar automaticamente essa documentação.

4 METODOLOGIA

O fluxograma contido na Figura 2 representa as etapas de desenvolvimento deste projeto. Para garantir melhor organização, o projeto foi dividido em cinco etapas:

1. Interface do Usuário;
2. *kafka*;
3. Aplicação *Django*;
4. API;
5. Modelo de aprendizagem treinado;

Figura 2 – Fluxo do Projeto



Fonte: Desenvolvido pelos autores.

4.1 Interface do Usuário

O primeiro bloco da Figura 2 representa a primeira interação do usuário com a aplicação. Esta interação foi feita através de uma interface, utilizando as tecnologias *Bootstrap*, *Html* e a biblioteca *JQuery*. Essas tecnologias auxiliaram no desenvolvimento do visual da interface e também nos tratamentos dos dados inseridos. Na tela, o usuário insere dois campos que são solicitados, o sexo e altura. Foram utilizados algumas funções nativas dos formulários feitos em *HTML* como: definição do tipo do campo "Altura"e a função de colocar apenas as opções "Masculino"ou "Feminino"na caixa de seleção do campo "Sexo". Após o ocorrido, o usuário irá clicar no botão "Enviar", nesse momento o *back-end* irá capturar os dados para realizar a comunicação inicial com *Kafka*. Os próximos passos serão explicados nos tópicos seguintes.

4.1.1 Usabilidade

A interface desenvolvida é para consultar e depois exibir os resultados retornados pela *API*. Vale ressaltar que a tela é responsiva e pode ser utilizada em diferentes dispositivos como celulares, *tablets* ou computadores. Para isso, foi utilizado o *framework Bootstrap* e assim teve um grande aumento na velocidade de desenvolvimento da tela. O *JQuery* foi usado para alterar o *layout* da tela criando uma nova caixa de texto com os resultados retornados da *API*.

4.2 *Kafka*

Após o usuário preencher e enviar os dados, o *back-end* da página em que a interface fica, irá enviar os dados para o *Kafka*, representado pelo bloco 2 da Figura 2. É responsável por enviar os dados para a camada da aplicação *Django*, além disso, irá receber os resultados gerados pela *API*, representada pelo quarto bloco da Figura 2, esses resultados serão enviados com destino a interface, para serem exibidos para o usuário.

A principal função do *Kafka* nesse projeto é otimizar a transmissão dos dados em todas as pontas, garantir que os dados possam ser acessados por diversos usuários simultaneamente e todos possam ter esses dados atualizados. Por fim, o *Kafka* irá otimizar a disponibilização dos dados de forma que os usuários não tenham problemas com a performance da aplicação.

4.2.1 Fluxo de dados Continuo

O Kafka permite que nossa aplicação tenha um fluxo continuo de dados, de forma resiliente, se houver algum erro quando for extrair os dados em algum consumidor. O *kafka* irá redirecionar essas mensagens para que não se percam. Então a arquitetura, caso a aplicação necessite de uma grande ingestão de dados, poderá gerenciar múltiplos acessos e garantir que os fluxos de dados não sejam interrompidos.

4.3 Projeto Django

O Projeto *Django*, representado pelo bloco 3 da Figura 2 é a base de toda a nossa aplicação, dentro dessa camada fica todas as configurações da aplicação, como o caminho que aponta para as outras camadas (*API* e Interface do usuário), por isso, todo o processo acaba passando por essa camada.

4.4 API

A *API* é o local aonde contém os métodos que são responsáveis por receber e validar os dados enviados pelo *Kafka*. Caso os dados sejam válidos, irá consultar o modelo pré-treinado que prevê o peso de uma pessoa baseado a altura e no sexo (Omair Aasim, 2019), assim, gerar o resultado e enviar os dados para o *Kafka* e ele os enviar para a tela do usuário. Caso aconteça algum erro ou os dados inseridos pelo usuário sejam inválidos, será enviado uma mensagem informando o erro para o usuário.

O *Postman* foi a ferramenta escolhida para testar e documentar requisições feitas na *API*, é uma ferramenta muito versátil e que deu rapidez aos testes por ser simples e intuitivo.

4.4.1 Segurança da API

O método responsável por receber e gerar os resultados possui uma camada de segurança baseada em *tokens*, antes de ter acesso aos dados é preciso previamente se autenticar e gerar uma chave. Somente com essa chave será possível ter acesso aos resultados gerados pela *API*, caso não seja autenticado, o sistema irá retornar uma mensagem informando que o usuário não foi autorizado a usar aquele método. Isso irá garantir que apenas pessoas autorizadas tenham acesso ao nosso sistema, evitando assim possíveis ataques ao sistema e a base de dados.

4.5 Modelo Pré Treinado

Por fim, temos o modelo de inteligência artificial previamente treinado, responsável por realizar e retornar as operações baseados nos dados recebidos. Lembrando que o programa desenvolvido poderá ser adaptado à qualquer modelo de inteligência artificial, por isso, foi escolhido um modelo pronto e treinado para realizar as análises. O modelo escolhido consegue calcular o peso de uma pessoa baseado na altura e no sexo e foi desenvolvido pelo Omair Aasim (2019).

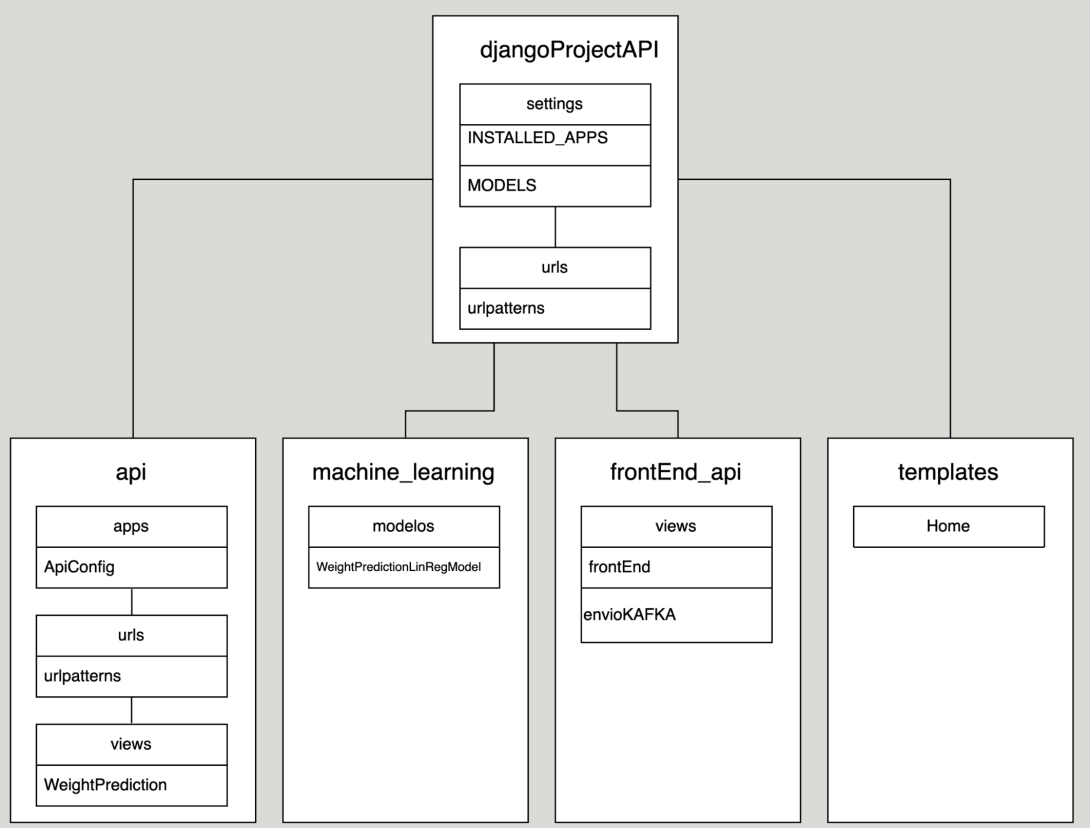
5 RESULTADOS

Nesta seção são apresentados os resultados adquiridos com o desenvolvimento e análises feitas do trabalho. Para melhor organização foi dividido em: Diagrama de Classes, Interface do Usuário, Outras formas de utilizar a *API* e Análise Ablativa.

5.1 Diagrama de Classes

Visando ilustrar a estrutura do como o projeto desenvolvido, a Figura 3 contém um Diagrama de Classes com todos os módulos principais, classes, funções e como é feita a comunicação entre os blocos.

Figura 3 – Diagrama de Classes



Fonte: Desenvolvido pelos autores

5.1.1 djangoProjectAPI

O módulo principal da aplicação é o *djangoProjectAPI*, dentro dele ficam algumas classes com as configurações do projeto. A classe *settings* contém as principais configurações do *Django*, como o idioma da aplicação, possíveis conexões com banco de dados, entre outros. Nesse projeto estamos utilizando duas configurações: a primeira é a *INSTALLED_APPS* aonde são registrados os módulos da aplicação, ou seja, qualquer camada adicionada tem que ser registrada nela ou removida caso seja necessário. A segunda configuração é feita na *MODELS*, nela é registrado a *machine_learning*, que é a pasta aonde está localizada o arquivo do modelo de aprendizado pré-treinado. Caso seja necessário alterar essa pasta do modelo, é preciso alterar essa configuração. A classe *urls* contém todas as rotas das outras camadas da aplicação, a rota da *API* e da interface utilizada pelo usuário.

5.1.2 api e machine_learning

A camada *api*, apresenta três classes principais: *ApiConfig*, *urls* e *views*. A classe *ApiConfig* contém a variável *(*MODEL_FILE) e nessa variável é atribuído o caminho aonde está localizado o arquivo do modelo pré-treinado. Nesta mesma classe engloba outra variável chamada *model* responsável por carregar o arquivo do modelo no sistema para que outras classes possam utilizar. A segunda classe é a *views*, nessa classe acontece todo processamento da *API* e apresenta o método *WeightPrediction* responsável por receber os dados, realizar as validações e depois consultar o modelo configurado, além de realizar o envio e receber os dados do *Kafka*. Por último temos a classe *urls* contendo a rota da *API*, essa rota está diretamente ligada na rota registrada na classe *djangoProjectAPI*, essa rota é o caminho utilizado para enviar os dados para os métodos que irão processar as informações.

5.1.3 frontEnd_api e templates

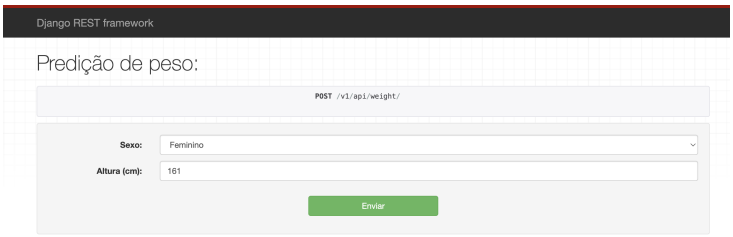
Por fim, temos os últimos módulos que são responsáveis pela interface gráfica. O módulo *frontEnd_api* contém a classe *views*, aonde estão os métodos: *frontEnd* que retorna o arquivo que abrange a interface do usuário, esse arquivo é retirado do módulo *templates* e o método *envioKAFKA* que recebe os dados da interface e os envia para o *kafka*.

5.2 Interface do Usuário

Como mencionado na metodologia, o usuário irá preencher os dados e em seguida irá clicar no botão "Enviar"para executar nossa *API*. A Figura 4 contém uma captura de tela com a

interface do usuário:

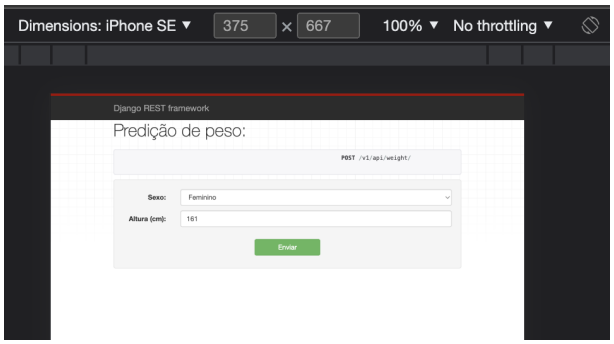
Figura 4 – Interface do Usuário



Fonte: Desenvolvido pelos autores

Na Figura 5, temos uma versão dela em outro tipo de dispositivo, para ilustrar um pouco da responsividade do site:

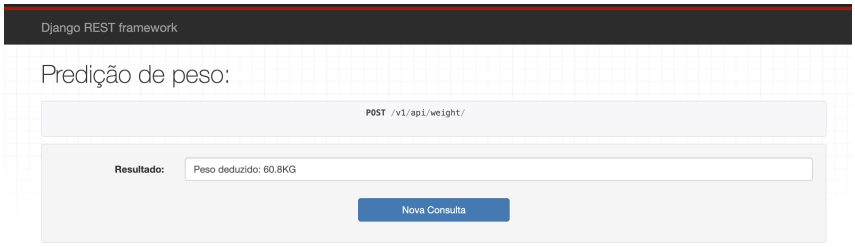
Figura 5 – Interface do Usuário Responsiva



Fonte: Desenvolvido pelos autores

A Figura 6 mostra a interface já com o resultado gerado pela *API*, nessa tela também é possível fazer uma nova requisição caso for necessário, para isso é preciso clicar no botão "Nova Consulta":

Figura 6 – Interface do Usuário Com Resultado

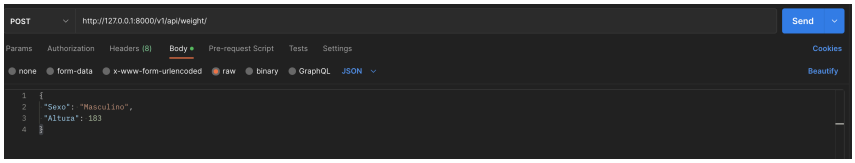


Fonte: Desenvolvido pelos autores

5.3 Outras formas de utilizar a *API*

Existem outras formas de se consumir uma *API*, e elas se aplicam a nossa, caso o contexto seja uma integração entre sistemas, por exemplo, o usuário irá fazer a requisição diretamente na *API* passando ambos os dados em formato *JSON*. Na Figura 7, temos uma imagem utilizando a ferramenta *Postman* para ilustrar esse tipo de uso:

Figura 7 – Exemplo de requisição na API



Fonte: Desenvolvido pelos autores

Em ambos os casos, interface do usuário ou pelo *Postman*, a aplicação recebe a requisição e os dados serão enviados para o *Kafka*:

Figura 8 – Dados no KAFKA

Fonte: Desenvolvido pelos autores

Esses dados serão utilizados pela camada da *API* junto com o modelo de IA para gerar o resultado, na Figura 9 podemos ver o *JSON* retornado pela API:

Figura 9 – Exemplo de resultado retornado pela API

Fonte: Desenvolvido pelos autores

Por serem gerados em *JSON*, os resultados podem ser facilmente integrados a outros sistemas, ou serem exibidos por uma interface para o usuário, que é o caso deste projeto.

5.4 Análise ablativa

Como forma de comprovar a funcionalidade da arquitetura desenvolvida, retiramos o *Kafka* dela. Com isso, fizemos algumas observações que determinaram sua importância nesse

projeto. Além de deixar a arquitetura simples, sem o *Kafka* na aplicação ficou suscetível a falhas no fluxo de transmissão de dados e assim não é possível assegurar que em um cenário maior, com outro modelo, os dados sejam transmitidos de ponta-ponta de forma atualizada, segura e continua, pois o *Kafka* é necessário para fazer esse gerenciamento de múltiplas requisições. Em cenários menores não é estritamente necessário, porém como o objetivo desse trabalho é a construção de uma arquitetura sólida, escalável, que pode servir como base para a criação de aplicações que sejam introduzidas em grandes cenários, com diferentes modelos de aprendizado de máquina, o *Kafka* é importante para o funcionamento de todo o fluxo planejado.

6 CONCLUSÃO

O presente trabalho foi possível desenvolver uma aplicação com objetivo de comprovar a facilidade do desenvolvimento e integração dos sistemas. Assim, propondo uma arquitetura robusta e segura com capacidade de ser utilizada em situações maiores e mais complexas.

Um ponto para destacar é sobre a funcionalidade do *Kafka* no projeto, pois foi implementado com objetivo de atribuir robustez e seguranças dos dados. Sem o *Kafka* a aplicação irá funcionar normalmente, porém não será possível garantir a qualidade e a segurança na transmissão dos dados em tempo real, principalmente em cenários maiores e mais complexos.

Para o desenvolvimento do projeto foi realizado uma busca por tecnologias que vem sendo mais utilizadas no mercado, assim, realizando uma análise do funcionamento e definindo as tecnologias com base na popularidade e familiaridade dos autores.

Este trabalho pode ser útil para pessoas que trabalham ou gostam da área de desenvolvimento de aplicações e que queiram se aprofundar mais no assunto e entender a forma que as tecnologias se integram.

Para trabalhos futuros, o desafio será como aplicar o projeto desenvolvido em outra situação maior e mais complexa, visando abordar uma boa integração entre os sistemas, garantindo a segurança dos dados, implementando a um banco de dados e utilizando um outro modelo de aprendizado de máquina pré-treinado.

REFERÊNCIAS

Amazon Web Services. **O que é uma API?** 2022. Disponível em: <https://aws.amazon.com/pt/what-is/api>. Acesso em: 03 de nov. 2022.

ARIMOTO, M.; OLIVEIRA, W. Dificuldades no processo de aprendizagem de programação de computadores: um survey com estudantes de cursos da Área de computação. In: **Anais do XXVII Workshop sobre Educação em Computação**. Porto Alegre, RS, Brasil: SBC, 2019. p. 244–254. ISSN 2595-6175. Disponível em: <https://sol.sbc.org.br/index.php/wei/article/view/6633>.

BOTEJU, W. *et al.* Deep learning based dog behavioural monitoring system. In: IEEE. **2020 3rd International Conference on Intelligent Sustainable Systems (ICISS)**. [S.l.], 2020. p. 82–87.

CNN BRASIL. **Uber atinge 30 milhões de usuários no Brasil e supera nível pré-pandemia**. 2021. Disponível em: <https://www.cnnbrasil.com.br/business/uber-atinge-30-milhoes-de-usuarios-no-brasil-e-supera-nivel-pre-pandemia>. Acesso em: 04 de nov. 2022.

CONCEIÇÃO, D. N. C. M. da. **Open source face recognition API**. 2019. Tese (Doutorado) — Instituto Universitário de Lisboa, Lisboa, 2019. Disponível em: <http://hdl.handle.net/10071/20307>.

DILEEP, P.; THOGARU, M. Real time cloud computing based covid-19 health monitoring system using iot with integration of machine learning approach to create safety environment. In: **Annals of the Romanian Society for Cell Biology**. [s.n.], 2021. p. 2076–2086. Disponível em: <https://www.annalsofrscb.ro/index.php/journal/article/view/326>.

H. FATIMA. **Flask x Django: como escolher o framework correto para seu aplicativo web**. 2018. Disponível em: <https://imasters.com.br/back-end/flask-x-django-como-escolher-o-framework-correto-para-seu-aplicativo-web> Acesso em: 04 de nov. 2022.

LAKSHAN, W.; SILVA, A.; WEERAKOON, W. An enhanced ensemble model for crime occurrence prediction using machine learning. In: **Proceedings of the International Conference on Applied and Pure Sciences (ICAPS 2021-Kelaniya)Volume 1**. Faculty of Science, University of Kelaniya, Sri Lanka., 2021. Disponível em: <http://repository.kln.ac.lk/handle/123456789/24070>.

LEMES, D. de O. Aspectos gerais de uso das interfaces gráficas de usuário. **TECCOGS: Revista Digital de Tecnologias Cognitivas**, n. 18, 2018.

LIU, W.; WANG, L. A machine reading comprehension framework for rest api information extraction. In: **2021 2nd International Conference on Artificial Intelligence and Information Systems**. New York, NY, USA: Association for Computing Machinery, 2021. (ICAIIS 2021). ISBN 9781450390200. Disponível em: <https://doi-org.ez93.periodicos.capes.gov.br/10.1145/3469213.3470294>.

LUBY. **O que é Apache Kafka e como funciona?** 2021. Disponível em: <https://luby.com.br/desenvolvimento-de-software/programacao/o-que-e-apache-kafka>. Acesso em: 04 de nov. 2022.

LUDERMIR, T. B. Inteligência artificial e aprendizado de máquina: estado atual e tendências. **Estudos Avançados**, SciELO Brasil, v. 35, p. 85–94, 2021.

MONTANHEIRO, L. S.; CARVALHO, A. M. M.; RODRIGUES, J. A. Utilização de json web token na autenticação de usuários em apis rest. **XIII Encontro Anual de Computação. Anais do XIII Encontro Anual de Computação EnAComp**, p. 186–193, 2017.

NOGUEIRA, E. F. S. **Machine Learning para previs ao de resultados de jogos de Tenis**. 2019. Tese (Doutorado) — Instituto Superior de Engenharia de Porto, Portugal, 2019. Disponível em: <https://recipp.ipp.pt/handle/10400.22/15552>.

Omair Aasim. **Machine Learning Project 9 — Predict weight based on height and gender**. 2019. Disponível em: <https://towardsdatascience.com/machine-learning-project-9-predict-weight-based-on-height-and-gender-9ed47285bcbb>. Acesso em: 05 de mai. 2022.

RED HAT. **O que é Apache Kafka?** 2017. Disponível em: <https://www.redhat.com/pt-br/topics/integration/what-is-apache-kafka>. Acesso em: 04 de nov. 2022.

Tainá Freitas. **Os três tipos de aprendizado no machine learning, um ramo da inteligência artificial**. 2019. Disponível em: <https://www.startse.com/noticia/nova-economia/machine-learning-inteligencia-artificial-aprendizado> Acesso em: 04 de nov. 2022.

TOTVS. **Inteligência Artificial: o guia completo sobre o assunto!** 2022. Disponível em: <https://www.totvs.com/blog/inovacoes/o-que-e-inteligencia-artificial> Acesso em: 04 de nov. 2022.

UBER ENGINEERING. **uMirrorMaker: Replicador robusto Kafka da engenharia da Uber**. 2019. Disponível em: <https://imasters.com.br/analytics/umirrormaker-replicator-robusto-kafka-da-engenharia-da-uber>. Acesso em: 04 de nov. 2022.

VAINIKKA, J. **Full-stack web development using Django REST framework and React**. 2018. Tese (Doutorado), 2018. Disponível em: <https://www.theseus.fi/bitstream/handle/10024/146578/joel_vainikka.pdf?sequence=1>.