UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO
INSTITUTO MULTIDISCIPLINAR

LUCAS NUNES DALBONIO DE CARVALHO

# Uma proposta de um sistema de Inteligência Artificial como um Serviço

Prof. Filipe Braida do Carmo, D.Sc.
Orientador

Nova Iguaçu, Agosto de 2022

# Uma proposta de um sistema de Inteligência Artificial como um Serviço

**Lucas Nunes Dalbonio de Carvalho**

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto Multidisciplinar da Universidade Federal Rural do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Apresentado por:

_____

Lucas Nunes Dalbonio de Carvalho

Aprovado por:

_____

Prof. Filipe Braida do Carmo, D.Sc.

_____

Prof. Leandro Guimarães Marques Alvim, D.Sc.

_____

Prof. Marcel William Rocha da Silva, D.Sc.

NOVA IGUAÇU, RJ - BRASIL

Agosto de 2022

# Agradecimentos

Aos meus pais, que sempre me incentivaram, até mesmo nos momentos difíceis. Aos amigos, especialmente Tuan, Carrambert e Yago. Contribuíram muito com todo o apoio demonstrado ao longo do período de tempo em que me dediquei a este trabalho. Aos professores, por todos os conselhos, pela ajuda e pela paciência com a qual guiaram o meu aprendizado e a todos aqueles que contribuíram, de alguma forma, para a realização deste trabalho.

RESUMO

Uma proposta de um sistema de Inteligência Artificial como um Serviço

Lucas Nunes Dalbonio de Carvalho

Agosto/2022

Orientador: Filipe Braida do Carmo, D.Sc.

Com o grande volume de dados sendo gerado e consumido atualmente na internet. Muitas empresas estão querendo se utilizar de inteligência artificial para analisar de forma mais refinada esses dados. Muitas das empresas de pequeno/médio porte não tem renda suficiente para conseguir os equipamentos físicos sofisticados necessários para essas tarefas de alto processamento, além do custo da equipe que faria a manutenção desses equipamentos. Por esses motivos essas empresas não conseguem utilizar a tecnologia. Porém, com o modelo de Software como um Serviço, é possível oferecer remotamente diversos serviços de inteligência artificial através da internet, tornando viável que empresas consigam utilizar dessa tecnologia com custo reduzido. Levando isso em consideração, esse trabalho apresenta uma proposta de uma arquitetura de inteligência artificial como um Serviço, o qual entrega modelos que serão executados remotamente e assim, deixando o processamento longe do equipamento do usuário. Dessa maneira os modelos são entregues e usados sob demanda, reduzindo drasticamente o custo quando comparado ao uso e manutenção de um equipamento físico. Para validar essa arquitetura foi criado um sistema que leva o nome de Capivara, e com ele é possível criar novos modelos de inteligência artificial de forma dinâmica, além de gerenciá-los com facilidade.

ABSTRACT

Uma proposta de um sistema de Inteligência Artificial como um Serviço

Lucas Nunes Dalbonio de Carvalho

Agosto/2022

Advisor: Filipe Braida do Carmo, D.Sc.

*Currently there is a massive amount of data being generated and consumed in the internet. Lots of enterprises wish to use artificial intelligence and analyze efficiently all this data. Nevertheless, tons of small/medium enterprises does not have enough money to buy the sofisticated hardware needed to these high processing tasks. Besides, there is still the cost of maintaining a team to handle the hardware. For these reasons, companies have it hard to use artificial inteligence technology. Although, with Software as a Service it is entirely possible to offer remote artificial intelligence services, making it viable to companies to use it with reduced cost. Taking it all into acount, this work presents a proposal to an architecture of an Artificial Intelligence as a Service, which delivers models that the processing is not in the user hardware. This way, the models are delivered on demand, hence the cost is drastically reduced when compared to the use and maintainance of a dedicated hardware. To validate this architecture, we created a system called Capivara, which allows us to create new artificial intelligence models dynamically, and easily manage them.*

# Lista de Figuras

# Sumário

# Capítulo 1

# Introdução

A demanda por serviços Web atualmente está muito alta. Uma das principais motivações para tal é a facilidade que usuários têm de acessar esse tipo de serviço. Basta utilizar o navegador. Ele obtém arquivos em um formato de texto e os transforma em algo visual para o usuário. Dito isso, serviços Web podem ser usados também para servir dados de entrada para outro programa, mas ao invés de usar formatos que serão renderizados graficamente, são usados arquivos em outro formato especial para estruturar esses dados (RICHARDSON; RUBY, 2008).

Com as constantes melhorias em infraestrutura de redes e de processamento, cada vez mais surge a necessidade de ter um maior espaço de armazenamento. O problema é que espaços físicos de armazenamento têm um custo muito alto. Além disso, o espaço pode ser mal aproveitado, visto que é bastante difícil usar toda a capacidade de armazenamento do dispositivo. A solução para esse problema é usar o modelo de computação em nuvem.

Computação em Nuvem é um modelo de computação amplamente usado nos dias atuais. Diversas empresas como Amazon[1], Microsoft[2] e Google[3] têm alocado muito esforço no desenvolvimento de aplicações nesse modelo. Segundo Hurwitz e Kirsch (2020), é um modelo que consiste na ideia de que tudo pode ser oferecido ao usuário

[1]https://www.amazon.com
[2]https://www.microsoft.com
[3]https://www.google.com

como um serviço. Desde poder computacional, até infraestrutura ou armazenamento. É um modelo que ganhou muita força com o avanço na infraestrutura da rede de Internet, já que esses serviços são oferecidos pela rede.

Esse modelo de computação também tem seus pontos negativos. É muito difícil manter um sistema em nuvem completamente seguro por conta da sua natureza de oferecer tudo via internet. Existem diversas medidas para que a experiência do usuário não seja comprometida por esses problemas. Políticas bem definidas para tratamento de possíveis danos aos pontos de armazenamento e rigorosas restrições de acesso a dados sensíveis são algumas das medidas cabíveis (AVRAM, 2014).

A partir desse modelo muitas empresas buscam se inserir no mercado de Inteligência Artificial e desenvolver soluções usando tecnologias desse tipo. O problema anteriormente era que empresas com pouco capital não tinham viabilidade para comprar máquinas com poder computacional suficiente para executar tarefas de Inteligência Artificial. Como essas tarefas não seriam executadas o tempo inteiro, provavelmente a compra dessa máquina não seria um bom custo-benefício. Problema esse resolvido pelos recursos computacionais oferecidos sob demanda no modelo de computação em nuvem (RIBEIRO; GROLINGER; CAPRETZ, 2015).

Dito isso, o objetivo deste trabalho é apresentar uma arquitetura de software MLaaS (*Machine Learning as a Service* ) que permita a criação de modelos de forma dinâmica, ou seja, permita adicionar ou remover diferentes modelos de inteligencia artificial do sistema com tranquilidade. Modelos esses que podem ser implementados em qualquer linguagem, com qualquer biblioteca e utilizando qualquer serviço, desde que seja possível utilizá-los via internet ou por troca de mensagens internas pela máquina. Para que seja possível validar a arquitetura, criamos um sistema chamado Capivara que se utiliza dessa arquitetura. Desse modo, para que esse objetivo seja alcançado, este trabalho precisa atender os seguintes pontos:

- Propor uma arquitetura de MLaaS.
- Criar um sistema que se utiliza dessa arquitetura.
- O sistema deve ter a capacidade de criar novos modelos de forma dinâmica

- O sistema deve ter a capacidade de aceitar diversos modelos de Inteligência Artificial
- O sistema deve ter a capacidade de gerenciar esses modelos.
- O sistema deve ter a capacidade de gerenciar usuários.

Após essa breve introdução, o próximo capítulo apresenta alguns conceitos de sistemas Web e uma breve introdução sobre Computação em Nuvem, além de uma breve introdução sobre Inteligência Artificial, Aprendizado de Máquina, e como esses dois temas podem ser bem utilizados em sistemas de nuvem. Em sequência, são apresentados a motivação desse trabalho, diversos trabalhos relacionados e a proposta de maneira mais detalhada. No quarto capítulo há uma breve introdução das ferramentas de software utilizadas para o trabalho e detalhes da implementação e uma apresentação da aplicação. O último capítulo contém a conclusão sobre o trabalho e possibilidades de como continuá-lo no futuro.

# Capítulo 2

# Fundamentação

Neste capítulo falaremos brevemente sobre a história dos serviços Web e algumas de suas características, além de Inteligência Artificial com um certo foco em Aprendizado de Máquina. Também será comentado como integrar sistemas de Aprendizado de Máquina em sistemas Web de forma que o processamento não seja efetuado pelo usuário. E, além disso, toda a base teórica necessária para o entendimento das ferramentas utilizadas para a construção do trabalho.

Na seção 2.1 apresentamos diversas informações básicas para um bom entendimento sobre serviços Web. Já na seção 2.2, introduzimos conceitos sobre computação em nuvem e detalhes sobre seus principais modelos. Na seção 2.3 contém uma boa base teórica sobre Aprendizado de Máquina para que os conceitos da proposta sejam claros. Por último, na seção 2.4, apresentamos o conceito de sistemas de Aprendizado de Máquina e Inteligência Artificial usados em um dos modelos de computação em nuvem.

## 2.1 Serviços Web

Para buscar informações sobre um certo tópico, é comum, através do navegador Web, acessar a URL (*Uniform Resource Locator*) de uma página de busca, *e.g.*

Google[1], Bing[2] e DuckDuckGo[3]. Com isso, o servidor serviu a uma página Web, um documento do formato HTML (*HyperText Markup Language*) que o navegador renderiza graficamente. Após visualizar a página, inserir o texto e submeter a sua pesquisa, o navegador realiza uma segunda requisição para uma URL que contenha esse tópico (RICHARDSON; RUBY, 2008).

A Web é recheada de dados, como informações sobre livros, filmes, opiniões, fotografias, esportes e diversas aleatoriedades. É também composta por muitos serviços sobre esses dados. Desde lojas de roupa online, jogos, até ferramentas de busca e enciclopédias. E para utilizar desses serviços, ao invés de instalar todos esses programas e realizar o download de todos os dados, é necessário apenas o navegador Web para acessar e usufruir do serviço (RICHARDSON; RUBY, 2008).

De acordo com Richardson e Ruby (2008), a Web programável é muito similar. Difere apenas no formato de saída dos dados. Ao invés de páginas HTML montadas com visuais atrativos e chamativos, é comum que a Web programável sirva documentos com formatos puramente focados nos dados retornados, como XML e JSON. Ela não é necessariamente para o consumo humano. Muitas vezes o objetivo é que esses dados sejam servidos como entrada para um segundo programa.

A Web é baseada no protocolo HTTP. A grande maioria serve em um formato dentre os seguintes: HTML, JSON, Texto, documentos binários ou XML. Como mostra a figura 2.1, antigamente o XML era muito dominante, mas o formato JSON teve o seu uso bastante elevado nos últimos anos pela facilidade de leitura. Essas são as terminologias com conceitos bem definidos, já que diferentes pessoas usam termos comuns como REST de forma vaga e confusa. O que falta é uma maneira coerente de classificar a Web programável. Ao resolver esse problema, os termos individuais se tornam mais claros (RICHARDSON; RUBY, 2008).

[1] <http://www.google.com/>
[2] <https://www.bing.com/>
[3] <https://duckduckgo.com/>

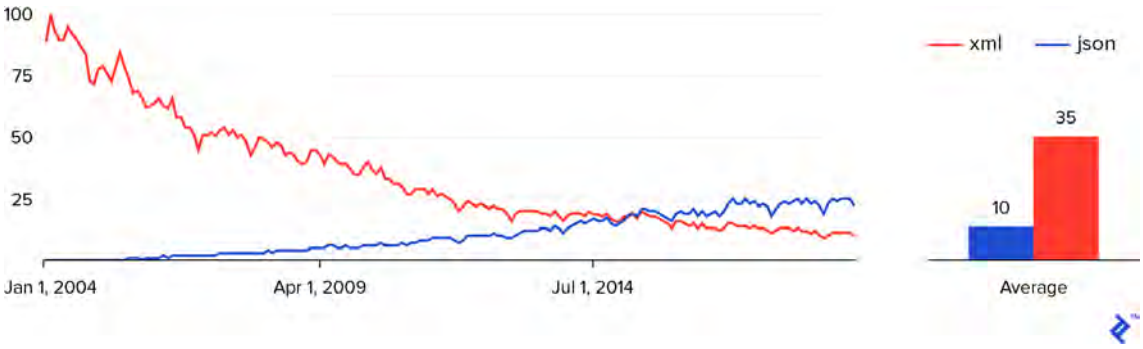


Figura 2.1: Adoção do formato JSON ao longo dos anos. Retirado de <https://www.toptal.com/Web/json-vs-xml-part-1>

### 2.1.1 Protocolo HTTP

Richardson e Ruby (2008) definem HTTP como um protocolo baseado em documentos, em que o cliente envia uma requisição para o servidor. Esse servidor em questão retorna a requisição por meio de um documento de resposta, enviada para o cliente. Esse protocolo tem regras bem definidas sobre como o envelope deve ser feito, mas poucas regras para o que decide enviar através dele. Essa troca é bem ilustrada pela figura 2.2

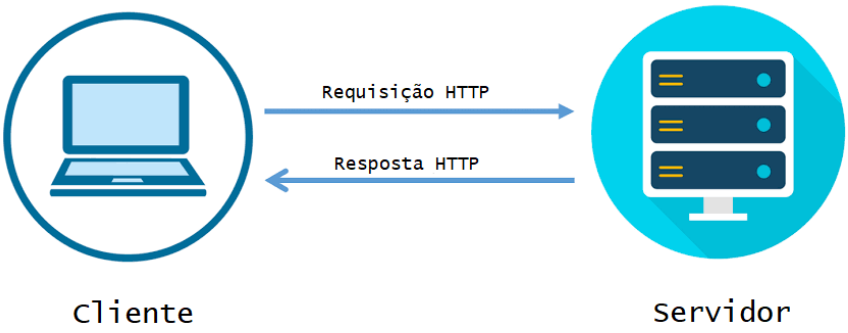


Figura 2.2: Modelo de Cliente e Servidor usando o protocolo HTTP. Retirado de <https://medium.com/@rohitpatil97/http-request-http-response-context-and-headers-part-iii-5c37bd4cb06b>

### 2.1.2 REST

Definido pela primeira vez por Fielding (2000), REST (*Representational State Transfer*) é um estilo de arquitetura para sistemas de hipermídia e representa uma

série de restrições para a arquitetura. Algumas delas são: sistemas cliente-servidor e sem estados por natureza. As requisições devem ter toda a informação necessária para o seu entendimento.

Segundo Richardson e Ruby (2008), serviços Web REST se referem a serviços que parecem a Web e são chamados de serviços orientados a recurso. Dada a primeira linha de uma requisição HTTP para um serviço REST, deve ser claro o que busca o cliente. Por exemplo, a linha "GET /reports/open-bugs HTTP/1.1", dado que o cliente conheça o contexto, deve ser auto explicativa. Caso a informação desejada não esteja descrita na URL, não é um serviço orientado a recurso. Essas não são estritamente as únicas regras, mas fazem parte das mais importantes.

### 2.1.3 JSON e XML

JSON (*JavaScript Object Notation*) e XML (*eXtensible Markup Language*) são maneiras de formatar estruturas de dados em texto. A ideia por trás do JSON é simplificar o trabalho do navegador de obter os dados. Isso se dá porque diferentes navegadores tem diferentes interfaces de conversão para XML. Entretanto, já que um arquivo JSON é um subconjunto do Javascript, há uma compatibilidade natural entre os navegadores (RICHARDSON; RUBY, 2008). A figura 2.3 demonstra a diferença entre os dois formatos.


Figura 2.3: Diferenças entre os formatos XML e JSON.

## 2.2 Computação em Nuvem

Computação em nuvem é um método de oferecer recursos computacionais compartilhados, incluindo aplicações, processamento, armazenamento, rede, desenvolvimento e implantação. Computação em nuvem faz com que esses recursos sejam facilmente utilizáveis através de padronização. Essa técnica corresponde a implementar serviços usando um conjunto de interfaces consistentes. A ideia é que tudo, desde poder computacional até infraestrutura pode ser oferecido ao consumidor como um serviço (HURWITZ; KIRSCH, 2020).

Primeiramente o objetivo da nuvem tinha foco em cortar gastos computacionais e de armazenamento. Apesar do corte de gastos ainda ser um dos principais motivos de uso da nuvem, algumas empresas estão usando esse modelo como para transformar o seu negócio. Fazem isso gerenciando serviços em nuvens privadas e públicas, e líderes de tecnologia aplicam as vantagens dos modelos "como um serviço"e da infraestrutura presente para obter um melhor resultado final (HURWITZ; KIRSCH, 2020).

Hoje nós vivemos em um mundo amplamente conectado. A cada dia que passa, mais pesquisadores buscam maneiras de tornar a conectividade mais eficiente e mais empresas investem na infraestrutura necessária para suportar toda essa melhora. Esse crescimento abre muitas oportunidades para os modelos de computação em nuvem usados nos dias atuais, o Software como um serviço (SaaS), Infraestrutura como um serviço (IaaS) e Plataforma como um serviço (PaaS) (HURWITZ; KIRSCH, 2020).

No SaaS os vendedores entregam todo o processamento da informação sob demanda. Portanto, o serviço oferecido aos usuários não é o software em si, mas sim uma chave de acesso para uso do mesmo. Basta realizar esse acesso, aplicar as configurações necessárias e utilizar todos os serviços oferecidos pelo software. Ele faz bastante sucesso, pois toda a responsabilidade sobre a infraestrutura computacional fica com o comerciante (CAMPBELL-KELLY, 2009).

Alguns exemplos que já vem sendo usado há muitos anos são os provedores de e-mail. Após criar a conta e aplicar as configurações sobre a caixa de entrada,


Figura 2.4: Diferença de como aplicações SaaS são executadas. Retirado de <https://www.cloudflare.com/pt-br/learning/cloud/what-is-saas>

spans, etc, o usuário imediatamente recebe todo o serviço relacionado ao envio e recebimento de e-mails. Em sequência, outro serviço de bastante sucesso é o Youtube, uma plataforma gratuita de compartilhamento de vídeos, em que qualquer usuário pode enviar vídeos para a plataforma e configurá-los de acordo com suas preferências.

O SaaS se difere dos modelos de IaaS e PaaS. O primeiro tem mais relação com o poder computacional entregue, lidando diretamente com máquinas virtuais, armazenamento, servidores e configurações de rede. Muitas vezes também é chamado de Hardware como um serviço (HaaS). A Amazon Web Service (AWS) e a Microsoft Azure são IaaS muito usados atualmente. Ambos oferecerem inúmeros serviços de banco de dados e servidores com diferentes sistemas operacionais (SOWMYA; DEEPIKA; NAREN, 2014).

Por outro lado, Sowmya, Deepika e Naren (2014) descrevem PaaS como os serviços que fornecem recursos como sistemas operacionais, suporte a linguagens de programação, banco de dados e servidores Web que escalam de forma automática sob demanda da aplicação. Ou seja, o PaaS tem o objetivo de concentrar o foco apenas na aplicação a ser implantada, abstraindo a plataforma por trás dela. Docker e Kubernetes são alguns exemplos de tecnologias de virtualização bastante usadas de PaaS.

A engenharia dos dados em uma aplicação SaaS é de extrema importância em

seu funcionamento. Fox e Patterson (2013) levantam importantes vantagens sobre esse aspecto. Quando um grande grupo de usuários deseja interagir com o mesmo conjunto de dados. Em casos onde a base de dados é extensa e frequentemente atualizada, faz mais sentido centralizar-la e oferecer acesso via SaaS. Esses dados são usualmente armazenados diretamente no serviço, então o usuário não precisa se preocupar quanto a recuperação ou perda dos mesmos.

Além disso, como a única cópia efetiva do software é de controle total dos desenvolvedores, podem acontecer melhorias tanto no software quanto no hardware, contanto que não infrinja nenhuma das regras contratuais, de forma transparente para o usuário. E ainda é possível realizar testes com uma pequena fração dos consumidores, sem exigir que atualizações sejam feitas pelo usuário (FOX; PATTERSON, 2013).

Em contrapartida, Michon (2017) aponta esse modelo de aplicações como desvantajoso em situações onde a aplicação necessita ser altamente personalizada porque o controle do software não é do consumidor. E também em situações onde o serviço de internet é precário, já que tudo é baseado em conectividade com a nuvem. Além disso, também é um problema quando há regularizações e restrições legais sobre o espaço onde os dados serão armazenados. E por fim, se recuperar de falhas em sistemas de tempo real pode se mostrar um grande desafio. O fornecedor do serviço não pode garantir nada quanto a esses pontos.

## 2.3 Aprendizado de Máquina

Durante muito tempo nós nos perguntamos se é possível fazer com que computadores aprendam. Um entendimento sobre como fazer computadores aprenderem levaria a muitas outras possibilidades de uso dos computadores. Ainda não somos capazes de fazer computadores aprenderem tão bem quanto os humanos. Porém, já foram inventados alguns algoritmos muito efetivos para certos tipos de tarefas, além de um entendimento teórico inicial sobre o aprendizado (MITCHELL, 1997). Técnicas dessa área têm sido aplicadas com sucesso em vários setores, sendo alguns: reconhecimento de padrões, visão computacional, engenharia espacial e financeiro.

(NAQA; MURPHY, 2015).

Aprendizado é o aumento de desempenho, em um determinado ambiente, através do acúmulo de conhecimento adquirido pela experiência. É muito influenciado também pelo grau de supervisão. Quando o aprendizado é supervisionado, para cada erro cometido pelo algoritmo é sugerida uma direção ao caminho correto. Difere do aprendizado não supervisionado, cujo objetivo é diferenciar ações desejáveis e indesejáveis (LANGLEY, 1996).

Tarefas são de aprendizado supervisionado quando há um vetor de entrada em conjunto com os seus correspondentes valores corretos. Casos como reconhecimento de padrões, onde os possíveis valores corretos são variáveis discretas em forma de categorias, são chamados de problemas de classificação. Já quando os valores esperados são variáveis contínuas, são chamados de problemas de regressão. Um bom exemplo são algoritmos que tentam prever o preço de ações na bolsa de valores (BISHOP, 2006).

Outra técnica de aprendizado é o aprendizado por reforço, que tem apenas a supervisão sobre um sinal, a recompensa obtida pelo sistema após realizar alguma ação. A técnica se preocupa em encontrar ações adequadas em uma determinada situação que maximizem essa recompensa final. Nesse caso, o algoritmo não obtém exemplos ótimos antes de iniciar o aprendizado. Ele precisa descobri-los através de tentativa e erro. Em muitos casos a ação gerada não gera impacto somente na recompensa, mas também no ambiente em que se encontra (BISHOP, 2006).

Um problema bastante relacionado aos dados utilizados nos algoritmos é o *overfitting*. Provost e Fawcett (2013) definem overfitting como o viés dos procedimentos em relação aos dados de treinamento, em troca da generalização para dados que ainda não foram usados como entrada. Existem diversas técnicas para lidar com esses problemas, e uma muito utilizada é separar o conjunto total de dados em dados de treinamento, dados de validação e dados de teste.

Ao treinar um algoritmo diversas vezes utilizando os mesmos dados de treinamento e teste, é evidente que no final os dados de teste fazem parte do viés do algoritmo, já

que ele utilizou desses dados para melhorar seus parâmetros. Portanto, é importante deixar um conjunto de dados apenas para a validação final do algoritmo. Dessa forma podemos verificar se o algoritmo generaliza bem para dados que ainda não recebeu como entrada (PROVOST; FAWCETT, 2013).

## 2.4 *Artificial Inteligence as a Service* e *Machine Learning as a Service*

Com o crescimento da área de inteligência artificial, a cultura de muitas empresas passou a mudar. Muitos dados históricos obtidos durante anos que antes tinham pouca visibilidade, agora estão sendo muito utilizados para procurar melhoras tanto no domínio principal da empresa, quanto em seus departamentos internos. Por exemplo, é possível utilizar dos dados do departamento de Recursos Humanos para tentar entender qual perfil de funcionário costuma permanecer mais tempo na empresa.

Dito isso, empresas de grande porte têm capital o suficiente para investir em seus próprios times e equipamentos de inteligência artificial. Já com relação às pequenas empresas, existe uma dificuldade grande com relação ao custo computacional. Algumas empresas não têm condição de pagar por todo o equipamento especializado para esse tipo de tarefa. Também existem as empresas cujo domínio principal não envolve inteligência artificial, mas tem utilidade caso o custo compense.

Ribeiro, Grolinger e Capretz (2015) mostram que uma das possíveis maneiras de diminuir essa dificuldade é usando alguma plataforma de MLaaS. Nesse modelo os recursos são compartilhados por todos os usuários e alocados sob demanda, reduzindo uma boa parte dos custos. Usuários devem ter acesso aos serviços de Aprendizado de Máquina de forma eficiente e de qualquer local em que estejam. Ademais, é necessário que a interface gráfica apresente os resultados com clareza ao consumidor.

Esse contexto leva a criação de mais uma derivação dos modelos "*as a service*", o AIaaS (*Artificial Inteligence as a Service*). É apenas uma maneira de esclarecer que os serviços a serem oferecidos em nuvem são focados em inteligência artificial. Além disso, outra derivação é o MLaaS (*Machine Learning as a Service*), que é ainda mais

especializado, focado em tarefas de Aprendizado de Máquina. Esses modelos herdam diversos conceitos do SaaS. A diferença está em deixar claro para o usuário qual a temática do programa antes mesmo de apresentá-lo.

Por exemplo, devido ao aumento substancial das tecnologias de redes sociais, naturalmente há também aumento no interesse nos dados postados. Para um usuário comum obter esses dados diretamente da plataforma, seria requerido um grande esforço. Portanto, muitos serviços de processamento de linguagem natural e mineração de texto sobre esses dados são oferecidos pelas próprias empresas através de serviços Web RESTful, poupando o consumidor de diversos custos associados a esse trabalho (POP, 2016).

# Capítulo 3

# Proposta

Este capítulo apresenta o sistema proposto, sua motivação e modelagem da solução, incluindo detalhes e motivos de algumas escolhas, e trabalhos relacionados. A seção 3.1 discutirá a problemática em torno de computação em nuvem para sistemas de Inteligência Artificial (IA). Na seção 3.2 serão apresentados trabalhos que tenham alguma semelhança com este trabalho. Por fim, na seção 3.3 será apresentada a modelagem da solução proposta e a razão por trás das ideias, com base nos problemas já apresentados.

## 3.1 Motivação

Computação em nuvem é uma tecnologia que permite ao usuário acesso a sistemas e serviços em ambientes virtuais. Avram (2014) descreve algumas vantagens desse modelo, como o acesso instantâneo a máquinas sem nenhum investimento prévio. A flexibilidade da infraestrutura permite aumentar a capacidade computacional de maneira extremamente fácil, e muitas vezes com a possibilidade de ser sob demanda.

Porém, é importante atentar aos cuidados necessários quando se trata da nuvem. Toda essa conectividade exige extrema cautela quando se trata da segurança, privacidade e confiabilidade, tanto do serviço, quanto dos dados armazenados. Como o usuário terceiriza o ambiente, a empresa que mantém precisa ter políticas muito

bem planejadas para casos de indisponibilidade, ou até mesmo caso haja perda dos dados (AVRAM, 2014).

Através de dados disponibilizados pela Open Security Foundation, uma instituição que rastreia incidentes públicos reportados, 714 casos de perda de dados foram publicados em 2008, afetando um total de mais de 86 milhões de registros (LIU; KUHN, 2010). Dependendo do tipo de perda, a guardadora dos dados pode sofrer uma grande variedade de consequências, mas na grande maioria dos casos essas perdas acarretam problemas financeiros e de reputação (LIU; KUHN, 2010).

Esses incidentes podem ser divididos em dois tipos, vazamento e dano. O primeiro acontece quando dados sensíveis são acessados por pessoas que não possuem as devidas permissões. Esse tipo é muito comum quando pessoas usam programas maliciosos para ganhar acesso a dados que não podem ser acessados publicamente. Em um dos ataques desse tipo, mais de 130 milhões de registros sobre cartões de crédito foram obtidos de um dos maiores processadores de pagamento localizado nos Estados Unidos (LIU; KUHN, 2010).

Por outro lado, o incidente de dano é acarretado quando uma cópia correta do dado não está mais disponível, ou por acessibilidade, ou porque algo na cópia foi corrompida. Liu e Kuhn (2010) comentam que em 2009 um serviço de provedora de telefone sofreu uma vasta perda nos dados de clientes que era hospedada por um serviço terceirizado de dados na nuvem. Um erro no sistema fez com que milhões de usuários perdessem seus dados em nuvem temporariamente.

Seguindo todo esse aumento sobre gerenciamento de dados, nos últimos anos tem tido um aumento no número de empresas que procuram se utilizar de Inteligência Artificial. Como podemos ver na figura 3.1, o número de material acadêmico publicado na área de Inteligência Artificial cresceu substancialmente nos últimos 20 anos. No mundo todo as indústrias e seus analistas de dados estão capitalizando em novos conteúdos de Inteligência Artificial para melhorar a produção e o crescimento da empresa.

A China planeja ser a maior e melhor do mundo na área de Inteligência Artificial.

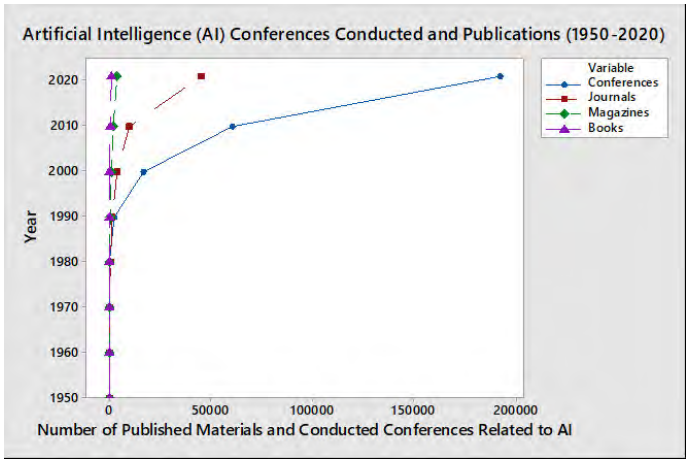


Figura 3.1: Conferências e Materiais publicados sobre Inteligência de 1950 a 2020. Retirado de (ROSALES et al., 2020)

O país em 2018 tinha um plano de governo com marcos específicos para se equiparar ao ocidente em 2020, obter avanços e descobertas até 2025, e se tornar a maior potência da área até 2030 (CHURCH, 2018). No Japão, o mercado de IA tem aumento previsto de 23.51% de 2015 até 2030. A figura 3.2 apresenta o tamanho de mercado global sobre Inteligência Artificial de 2015 a 2026. Além disso, devido à Inteligência Artificial, o crescimento econômico dos Estados Unidos irá progredir de 2.6% para 4.6% até 2035 (ROSALES et al., 2020).

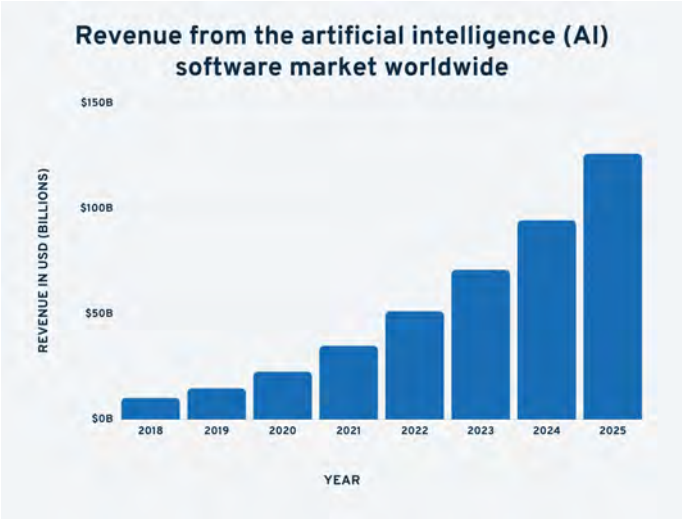


Figura 3.2: Mercado global de softwares de Inteligência Artificial no período de 2018-2025. Retirado de (ROSALES et al., 2020)

Empresas gigantes como Amazon[1], Facebook[2] e Google[3] estão alocando recursos em pesquisa de Inteligência Artificial. Melhor segurança, produtividade e eficiência fazem parte de apenas alguns dos muitos resultados obtidos (ROSALES et al., 2020). Com isso, houve a necessidade de uma mudança estrutural sobre como essa empresa deve se preparar para atacar e solucionar um determinado problema, caso seja realmente necessário e viável o uso de Inteligência Artificial. Fora o elevado custo em equipamentos físicos, muitas vezes necessário que uma grande equipe fizesse parte do projeto pela falta de sistemas facilitadores.

Dito isso, segundo Pop (2016) muitas dessas empresas estão utilizando computação em nuvem para acelerar tarefas relacionadas ao Aprendizado de Máquina, já que, segundo Agrawal, Das e Abbadi (2011), são uma ótima alternativa para esse tipo de tarefa pelo bom suporte para cargas de trabalho pesadas e análise de *Big Data*. Por conta disso, ambientes estatísticos muito usados como as linguagens R, Python e Octave, também tem um amplo suporte para nuvem. Eles oferecem uma boa estrutura de análise de dados e podem ser usados em extensos conjuntos de dados.

Esses ambientes muitas vezes são servidores como Amazon EC2[4], Google Cloud[5], Microsoft Azure[6], etc, que têm os programas estatísticos pré-configurados (POP, 2016). Fora o acesso físico, nesses ambientes o usuário tem total gerenciamento do servidor. Isso permite que otimizações ao nível de infraestrutura sejam possíveis. Portanto, elevando ainda mais a flexibilidade do serviço, a custo de profissionais capazes de gerenciar o servidor.

O problema é que muitas empresas não tem a infraestrutura necessária e a possibilidade de contratar profissionais para tarefas de alta especialização como essas. Seria muito difícil, dependendo do domínio da empresa, seus dados, e seu orçamento, de uma equipe com poucos membros ser capaz de resolver de todos os problemas envolvendo Aprendizado de Máquina nesse modelo onde o servidor é gerenciado

[1] <http://www.amazon.com>
[2] <http://www.facebook.com>
[3] <http://www.google.com>
[4] <https://aws.amazon.com>
[5] <https://cloud.google.com>
[6] <https://azure.microsoft.com>

pelo usuário. Além do Aprendizado de Máquina, serão necessários profissionais especializados em servidores.

O MLaaS (*Machine Learning as a Service*) é uma proposta de SaaS (*Software as a Service*), onde o software é centralizado em tarefas de Aprendizado de Máquina. Ele tem como proposta deixar todo fardo envolvendo infraestrutura, e gerenciamento da mesma, com o fornecedor. Para o usuário é entregue um acesso e com ele, principalmente via serviços web, o usuário faz requisições enviando os dados. Assim que o servidor finaliza a tarefa, recebe os dados já processados (POP, 2016).

Os membros da equipe teriam o trabalho de analisar do problema, decidir toda a infraestrutura necessária para execução dos algoritmos, escrever o código por completo, garantindo sempre que o mesmo está sem erros, e por fim, analisar o resultado obtido. Utilizando-se de um MLaaS, o trabalho de infraestrutura e código seria todo abstraído pelo sistema. Com isso, é permitido que a equipe gaste muito mais tempo focada no negócio da empresa.

Portanto, a ideia do projeto Capivara é construir um sistema que fornece ao usuário diferentes modelos de inteligência artificial. Ele irá possibilitar ao consumidor executar APIs de inteligência artificial pré-definidas e treinadas, sem uso explícito de código. Dessa forma usuários com bastante conhecimento matemático e de negócios sobre inteligência artificial podem usufruir bem do sistema. A ideia principal é a flexibilidade em gerenciar os modelos de inteligência artificial do sistema. E tudo isso com uma interface amigável e intuitiva, tentando oferecer a todo o momento a melhor experiência para o usuário.

## 3.2 Trabalhos Relacionados

O objetivo dessa seção é apresentar um pouco dos diversos trabalhos que recorrem ao Aprendizado de Máquina, quando oferecido como um serviço.

Pop (2016) realizou um levantamento com muitos exemplos de sistemas que utilizam Aprendizado de Máquina, inclusive sistemas de *MLaaS*. Tarefas desse calibre têm um alto consumo de tempo, portanto muito esforço é gasto para tentar diminuir o

tempo de execução das tarefas. E a computação em nuvem se tornou uma alternativa muito interessante para obter esse objetivo por suas características de provedor de serviços.

A Big ML[7] oferece uma grande variedade dos serviços mais comuns de Aprendizado de Máquina. É disponibilizado um painel de controle via Web, ou até mesmo acessos via API REST. Os serviços oferecidos são modelos de aprendizado supervisionado e não supervisionados. É possível que mais de uma pessoa colabore em tempo real para o mesmo projeto e também existe um acesso gratuito para que o sistema possa ser testado por qualquer um (CETINSOY et al., 2016).

Uma funcionalidade muito interessante da Big ML é a possibilidade de automação de tarefas. Foi criada uma linguagem específica de domínio, chamada *WhizzML*, para automatizar fluxos de trabalho complexos. Além da ferramenta *Scriptify*, que converte os fluxos de trabalho em *scripts*. (CETINSOY et al., 2016).

A ferramenta *OptiML* permite uma seleção de modelos automática de algoritmos de classificação e regressão após realizar o teste com diferentes parâmetros.

Kuznetsov, Giommi e Bonacorsi (2021) construíram um serviço completo de Aprendizado de Máquina SaaS para problemas que usam dados HEP (*High Energy Physics*) distribuídos. Eles não encontraram boas maneiras de usar dados HEP em nenhum serviço já feito por conta da inflexibilidade na hora de tratar os dados. Problemas de *HEP* têm um formato de dados específico, e fazer a transformação de dados para se adequar aos serviços existentes seria muito custoso. Também houve uma tentativa usando a plataforma *Spark*, mas a mesma limitaria a flexibilidade do usuário quanto as ferramentas de escolha.

Já na área de processamento de linguagem natural, como apresenta High (2012), o IBM Watson[8] é um sistema cognitivo que se tornou bastante conhecido ao longo dos anos. Ele oferece serviços em diversas áreas, como análise de risco, operações financeiros, saúde e atendimento ao consumidor. A partir da linguagem e dados do domínio específico, são realizadas operações para entendimento dos mesmos e com

[7] <https://bigml.com/>
[8] <https://www.ibm.com/watson>

isso, são geradas decisões informadas e entregues melhores experiências ao usuário.

O IBM Watson trabalha com segurança em qualquer plataforma de nuvem. Sustentado pelo *Red Hat OpenShift*, um sistema de implantação de contêineres, ele é capaz de integrar Inteligência Artificial onde for necessário, inclusive nas plataformas Amazon Web Service, Azure e Google Cloud Platform. Isso traz consistência e flexibilidade para o sistema. As análises podem ser feitas em documentos de texto, páginas Web e bancos de dados.

Kim et al. (2018) decidiram criar o NSML (Naver Smart Machine Learning) pois acreditam que as plataformas existentes em 2018 estavam incompletas com relação ao apoio ao usuário. Falta uma boa integração entre as plataformas de desenvolvimento, muitos modelos têm diferentes dependências de software e não é trivial realizar essa conexão. Além do mais, em muitos serviços atuais há uma forte dependência em serviços da AWS ou a Google Cloud como recurso computacional, não oferecendo suporte a *clusters* privados que o consumidor já tenha.

As funcionalidades mais importantes do NSML é o gerenciamento de recursos. Ele utiliza de forma eficiente e flexível todos os recursos computacionais. Quando um novo nó computacional é registrado no ecossistema, seus recursos são periodicamente verificados pelo NSML. Quando um usuário inicia algum serviço de Aprendizado de Máquina, o agendador de tarefas o aloca em um nó registrado com os recursos necessários (KIM et al., 2018).

Outra funcionalidade chave é a interface de usuário. As ferramentas disponíveis para interagir com o NSML são a linha de comando ou a interface Web. Usuários tem o poder de controlar e as suas tarefas de Aprendizado de Máquina, e obtém os resultados dos modelos e experimentos pela interface. O usuário pode gerenciar os dados pessoas de sua conta, realizar analise de dados e monitorar os recursos alocados (KIM et al., 2018).

One AI[9] é um serviço de Inteligência Artificial focado em linguagem natural e foi construído para desenvolvedores. O usuário tem a opção de analisar, processar

[9] <https://www.oneai.com/>

e transformar o texto de entrada. Não é necessário um conhecimento prévio de Inteligência Artificial ou processamento de linguagem natural já que em todas as tarefas é informado o dado de entrada, e o que é esperado de resposta. Os dois únicos tipos de entrada são artigo e conversa e nem todos os serviços são oferecidos para os dois tipos.

As tarefas oferecidas podem gerar texto, por exemplo, realizar o resumo de um texto ou de uma conversa. Podem também realizar uma análise da entrada, detectando e apresentando as emoções encontradas, ou separando e classificando as palavras de entrada em grupos, como nomes próprios, organizações, datas, localizações. Também é possível procurar palavras-chave de um texto e/ou realçar partes importantes do mesmo.

Para usá-lo, é enviada uma requisição em formato JSON, contendo o texto de entrada, o tipo de entrada, e uma lista das skills que deseja realizar. Os dois tipos de entrada têm formatos específicos para cada um, e são bem descritos na documentação. É permitido também invocar e encadear múltiplas skills em uma única requisição. A resposta de saída é acumulada a cada skill encadeada.

Clarifai[10] oferece uma plataforma completa com serviços de Inteligência Artificial, desde o preparo do conjunto de dados, treinamento de modelo até a implantação do mesmo. Tem como especialidade modelos para entender e identificar conceitos em entidades, sendo elas imagens, vídeos ou áudios. Conceito é algo que descreve a entidade no mundo real, como uma palavra-chave. É possível adiciona-la aos dados de entrada para que o modelo tenha esse conhecimento.

Com essa plataforma é muito simples criar o seu próprio software sobre esse contexto de identificar certas características de imagens. Se torna trivial realizar a buscas de imagens similares em um conjunto de dados de imagens. Basta adicionar todas as imagens do conjunto no seu ambiente virtual, selecionar a imagem correta e dar o comando de busca. Também é possível verificar se algum item específico pertence a uma imagem.

[10] <https://www.clarifai.com/>

A Google criou o Google Dialogflow API[11] com o objetivo de oferecer serviços para desenvolvimento de agentes virtuais. Esses agentes atendem clientes tanto por mensagem, quanto por chamadas de voz. Contém uma visualização gráfica do fluxo para que desenvolvedores consigam se adaptar mais rapidamente, além de ser possível colaboração de todos do time em simultâneo. Ademais, tem suporte a mais de trinta línguas e suas variações e também integração natural com a Web, aplicativos móveis e de telefonia.

Com técnicas avançadas e modelos robustos de entendimento de linguagem natural é possível captar intenções e um contexto geral da conversa. É possível a partir disso, definir transições de estados fora do tópico principal, mas sem perder o fluxo. Além disso, todo o gerenciamento operacional, análise, experimentos e avaliação do software pode ser feita pelo próprio Dialogflow, não é necessária nenhuma integração externa.

## 3.3 Proposta

Como apresentado anteriormente, existe uma grande parcela de empresas usando serviços de Inteligência Artificial, mesmo que não seja seu principal negócio. Isso faz com que o número de sistemas MLaaS disponíveis seja cada vez maior ao longo dos anos. Um sistema desse calibre precisa ter suporte a diversas funcionalidades que sistemas simples de Inteligência Artificial não têm a necessidade. Isso por conta da existência de usuários compartilhados e compartilhamento de recurso.

É necessário que sistemas MLaaS ofereçam segurança sobre os dados recebidos e enviados, já que dados sensíveis podem ser armazenados e processados pelo usuário. Além disso, é necessário que haja autenticação para que diferentes usuários tenham acesso ao mesmo recurso computacional, mas se utilizando de diferentes dados. Também é muito importante apresentar ao consumidor caso ocorra algum erro no modelo de Inteligência Artificial, pela transparência ao prover o serviço.

Por isso, estamos propondo uma arquitetura de MLaaS que permite a criação

[11] <https://cloud.google.com/dialogflow?hl=pt-br>

de modelos de forma dinâmica. Isso quer dizer que ao criar um novo modelo de inteligencia artificial e deixá-lo acessível via requisições http ou por mensagens dentro da mesma máquina, é possível adicioná-lo no sistema. Independente de qual linguagem foi usada, ou quais bibliotecas foram utilizadas. Também será importante a capacidade de ligar ou desligar um modelo específico no sistema. Isso permite que sejam feitas alterações e manutenções em modelos de forma que não afete o sistema principal de gerenciamento dos modelos.

Existem diversas maneiras de se construir a arquitetura para um sistema com as necessidades necessidades. A mais simples é a arquitetura Monolítica. Nessa maneira, todo o código é escrito inteiro como um único serviço. Algumas vantagens são a sua simplicidade e o fato de que não há troca de mensagens entre diferentes serviços na mesma aplicação. Porém, os pontos negativos se sobressaem bastante, principalmente para o nosso projeto, onde a ideia é consumir diferentes serviços, externos ao servidor ou não.

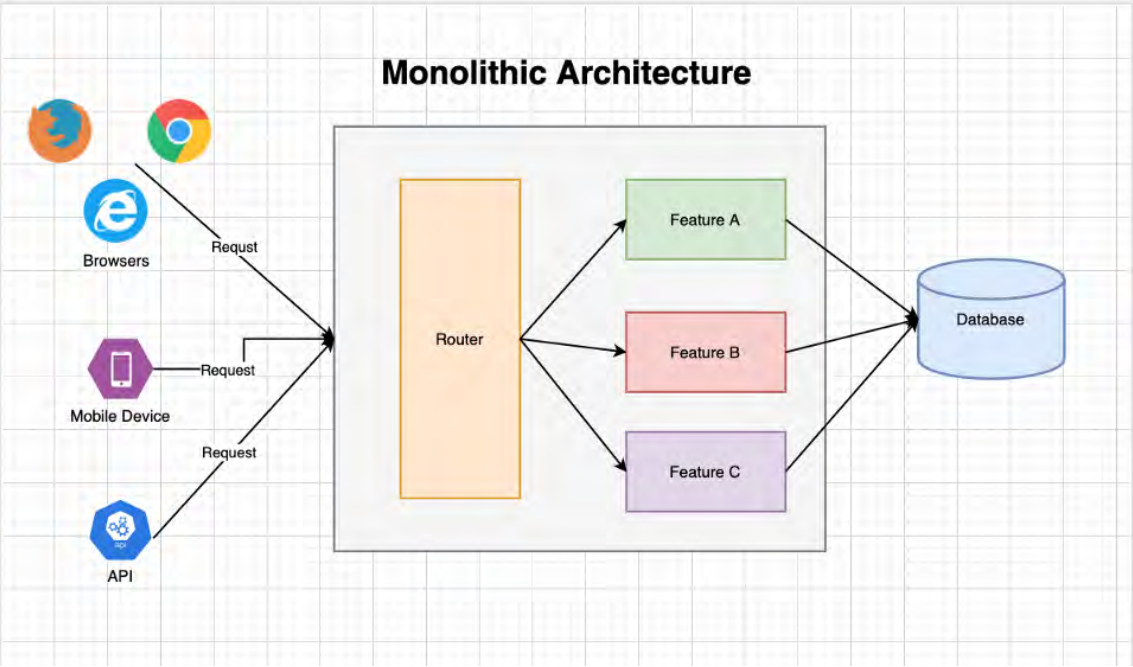


Figura 3.3: Arquitetura usando o padrão Monolítico. Retirado de <https://epsagon.com/development/monolithic-to-microservices-architecture-data-management>

A falta de encapsulamento na arquitetura Monolítica dificulta diversas questões de desenvolvimento. Pequenas mudanças em um componente podem gerar efeitos

colaterais em outros componentes já que tudo está interligado. Também faz com que qualquer mudança precise atualizar a versão do programa por inteiro já que não são componentes separados. Isso é um problema grave para aplicações que demoram para ser atualizadas, já que ficam inoperantes durante essa atualização.

Caso o projeto fosse implementado usando o padrão Monólito, teríamos uma série de limitações e problemas com relação a flexibilidade da aplicação. As vantagens de se fazer dessa forma é que o desenvolvimento é mais rápido e simples. Como tudo é necessariamente feito na mesma linguagem de programação, o desenvolvimento de uma aplicação feita em um único código é bem direta. Além de ser mais fácil capturar erros, gerar históricos de informação e realizar os testes por não depender de nenhum componente externo. Também facilita o processo de implementação, já que no final é apenas um grande componente contendo tudo.

Portanto, optamos pelo uso da abordagem de Microsserviços. Onde cada serviço é uma entidade separada no projeto e são completamente independentes. A princípio essa abordagem encaixa bem com o Capivara, pois nós iremos consumir bastantes serviços diferentes e seria muito impeditivo caso eles não fossem independentes entre si. Porém, nessa abordagem quando o número de serviços é muito grande, pode acontecer de o número de requisições ser um gargalo no sistema.

Sistemas que acessam APIs externas, ou seja, APIs que não fazem parte do sistema, é um problema conhecido e já bastante estudado na literatura. Richardson (2018) descreve diversos problemas sobre isso. Dentre eles, os mais importantes para a nossa proposta são a falta de encapsulamento dos serviços e a péssima experiência do usuário caso as requisições sejam feitas diretamente de seu navegador.

Por outro lado, a péssima experiência do usuário é um problema quando muitas requisições são feitas pelo navegador. Portanto, os dados precisam ser tratados diretamente no cliente. Esse efeito pode deixar a aplicação muito tempo em estado de espera, já que é necessário esperar a resposta das requisições e ainda manipular os dados que chegam. Tarefas de manipulação de dados, principalmente quando são em grande quantidade, idealmente são feitas pelo servidor por conta do poder computacional maior.

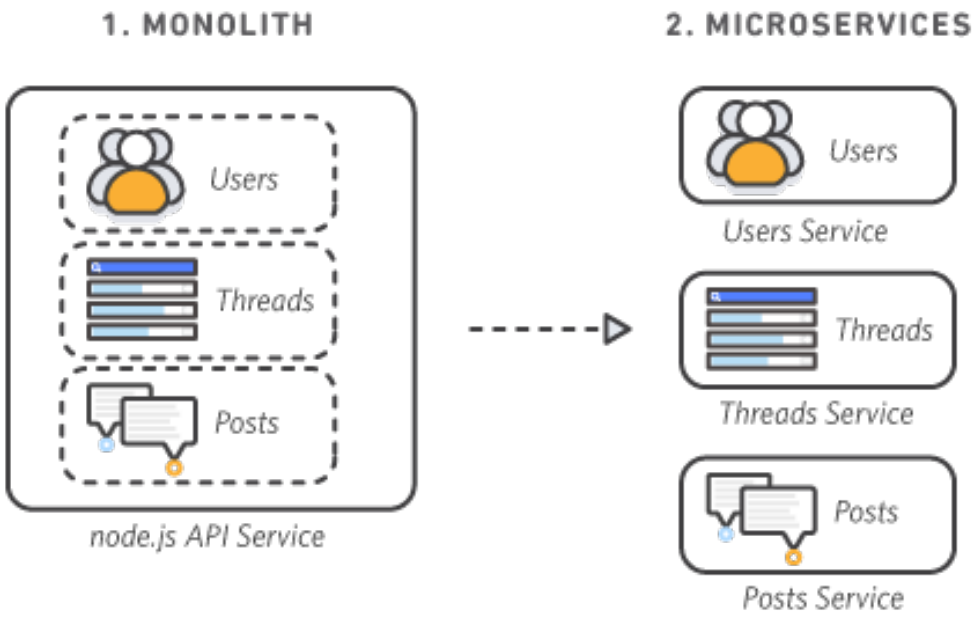


Figura 3.4: Abordagem Monolítica quando comparada a abordagem de Microsserviços. Retirado de <https://aws.amazon.com/pt/microservices>

Uma das soluções para esse problema é usar o padrão de projeto *Gateway*. Ele concentra todas as requisições em um único servidor. Esse servidor tem o trabalho de repassar corretamente os envios e respostas das requisições feitas pelo usuário e também outras funções, por exemplo, a autenticação do mesmo. Ele pode ate mesmo ser preparado para receber requisições de protocolos pouco usados e enviar requisições em HTTP (RICHARDSON, 2018).

Esse padrão não é indicado para casos em que o cliente está localizado próximo ao servidor. Já que a latência não é um empecilho, adicionar um *gateway* apenas aumentaria o número de requisições sem ter um ganho efetivo. Também não é uma boa ideia em casos onde há um grande número de requisições para um único serviço. A não ser que a ideia seja tratar melhor os dados no servidor, o *gateway* também não apresenta ganhos efetivos com relação as requisições.

Diante isso, é um padrão que encaixa perfeitamente com o Capivara. Como são diversos serviços diferentes, o cliente não precisará se preocupar em tratar diferentes formatos de dados. E em casos onde os serviços são implementados no mesmo

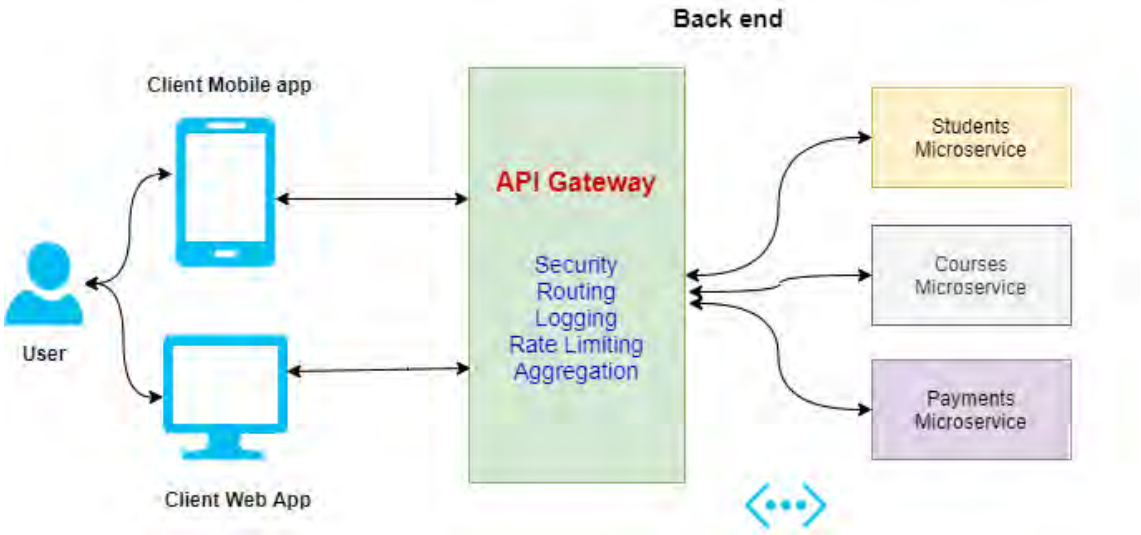


Figura 3.5: Arquitetura usando o padrão Gateway para requisitar separadamente os serviços de IA. Retirado de <https://www.c-sharpcorner.com/article/microservices-design-using-gateway-pattern>

servidor, ou em algum servidor próximo, a sobrecarga para enviar a requisição e receber sua resposta é mínima. Isso reduz bastante um dos problemas do *gateway*, que é adicionar sempre uma requisição a mais para obter a informação.

Também seria um problema limitar a linguagem de programação dos modelos de Inteligência Artificial. Muitas linguagens estão sendo usadas hoje em dia para diferentes modelos, onde suas peculiaridades fazem com que haja diferença de desempenho dependendo do modelo. Pode ser que para uma tarefa sobre sistemas de recomendação seja mais interessante usar Julia, mas para obter informações textuais sobre um documento a melhor opção seja Python.

Optamos pelo *gateway* pois deixaremos o gerenciamento de perfis de acesso e usuários centralizado no *gateway*. Enquanto isso, os modelos de Inteligência Artificial podem ser desenvolvidos em qualquer linguagem de programação e serão serviços independentes, desde que sigam a padronização do sistema. O banco de dados também é acessado apenas pelo gateway, fazendo com que o controle de acesso não seja necessário e problemas envolvendo múltiplos usuários alterando o banco de dados ao mesmo tempo não existam.

Como todo o gerenciamento é centralizado em um único serviço, não há replicação

de código e nem inconsistências por conta da replicação. Por outro lado, surge a necessidade de um gerenciamento dos serviços, já que eles são desenvolvidos puramente para processar informação e enviar a resposta para o programa principal. Isso dá uma flexibilidade muito maior para desenvolver, incluir ou até mesmo retirar serviços do sistema. Como os serviços são independentes, caso seja necessário realizar alguma atualização neles, não gera impacto algum no programa principal.

Dito isso, é importante ressaltar o motivo pelo qual optamos por usar a arquitetura de *gateway*. Ela atende muito bem um sistema em fase inicial, já que não é preciso utilizar conceitos de sistemas distribuídos em toda a aplicação, apenas na interação com os componentes de Inteligência Artificial. O fato de não ter componentes replicados e a implantação ser mais fácil também colaboram bastante. Porém, é importante ressaltar que dessa maneira não é viável ter tantos usuários utilizando simultaneamente o sistema.

Essa arquitetura tem um grave problema caso seja necessário escalar. A escolha de centralizar o gerenciamento de perfis de acesso e o banco de dados gera um gargalo quando muitos usuários usam o sistema simultaneamente, elevando muito a estimativa de tempo de espera por habilidade executada. Apesar disso, é uma arquitetura muito boa para um estágio inicial. Até que um número considerável de usuários acesse o sistema, não há problema em ter componentes centralizados. É possível mitigar esses problemas com tecnologias de mensageria e balanço de carga nos servidores, mas não resolve o problema por completo.

Para validar essa arquitetura, criamos o sistema chamado Capivara. A ideia central dele é fornecer ao usuário acesso a modelos de inteligência artificial, como, por exemplo correção ortográfica, modelo de detecção de ódio em textos e entre outras funcionalidades. Desta forma, ele não precisaria se preocupar em criar um ambiente para treinar esses modelos, mantendo o foco em sua utilização diretamente. No contexto do sistema, esses modelos serão chamados de habilidades. O usuário poderá ter acesso a diversas habilidades.

Para que o usuário consiga se identificar no sistema será permitido que usuários criem um acesso informando dados pessoais como e-mail, nome e uma senha. O

usuário pode ser administrador ou apenas um usuário comum. O sistema precisa de usuários administradores principalmente para gerenciar as habilidades, mas também permitimos que ele gerencie os usuários.

As habilidades serão modelos de Inteligência Artificial. O acesso de uma habilidade vai ser por *Web Service*, ou seja, ele utilizará o protocolo HTTP fornecendo os dados através de uma requisição POST. O formato desses dados é estabelecido pela documentação da habilidade. Após o processamento, o Capivara irá fornecer no retorno da chamada o resultado. Um exemplo dessa chamada pode ser vista na figura 3.6.

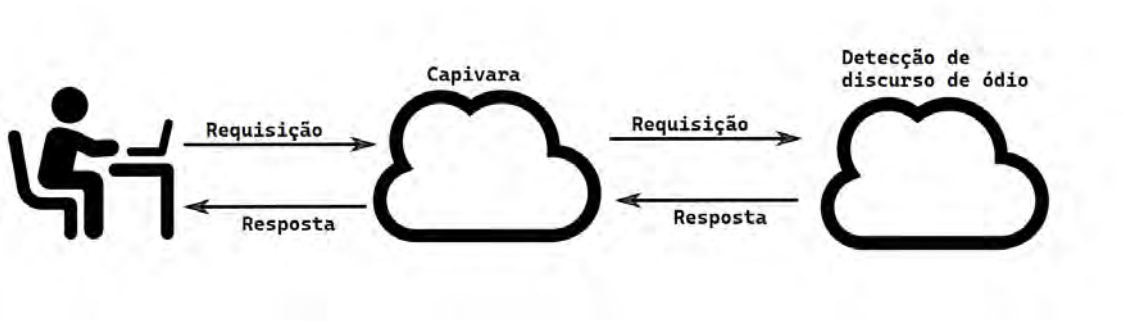


Figura 3.6: Exemplo de requisição para a habilidade

Como é necessário identificar qual usuário está fazendo a requisição, cada usuário tem a sua chave de segurança exclusiva. Ela deverá ser incluída nas chamadas da requisição e será usada para verificar se o usuário realmente tem permissão para utilizar a habilidade. No navegador é incluída automaticamente, mas caso queira usar habilidades usando programas externos, será necessário incluir manualmente a chave.

A responsabilidade de gerenciar as habilidades foi concedida aos administradores pois são entidades que podem sofrer alterações a qualquer momento. Não seria interessante realizar essas ações atualizando o código fonte ou acessando diretamente o banco de dados. Ao criá-las, é necessário informar todas as informações necessárias para fazer a requisição externa, ou seja, o endereço, a porta e o caminho do serviço. A partir do que foi dito, a seguir serão apresentadas algumas maneiras de se construir esse sistema.

Já que o Capivara é um serviço Web, decidimos que seria de suma importância fazê-lo de maneira responsiva, isto é, de forma que suporte dispositivos de diferentes dimensões. Isso permite que o usuário consiga fazer uso completo do sistema até mesmo com seu dispositivo móvel. Essa fácil acessibilidade torna o uso do sistema muito mais dinâmico.

# Capítulo 4

# Implementação

Com todo o avanço e aumento do número de sistemas de Inteligencia Artificial, uma grande barreira que ainda existe é sua implementação. Dito isso, a proposta do capivara é ser um software de Inteligencia Artificial como um serviço, no qual nos aproveitamos das vantagens de um SaaS, facilitando o acesso aos sistemas de Inteligencia Artificial integrados.

Neste capítulo, mostraremos o desenvolvimento da solução, tecnologias utilizadas, as decisões, os principais desafios e suas soluções assim como as vantagens e desvantagens de cada decisão tomada. Assim, a seção 4.1 se refere às tecnologias utilizadas no desenvolvimento. A seção 4.2 apresenta detalhes da implementação do sistema a fim de ter uma perspectiva completa do processo de criação do nosso sistema. Na seção 4.3 contém os desafios e suas soluções, além das vantagens e desvantagens das mesmas.

## 4.1 Tecnologias utilizadas

### 4.1.1 Node.js

Um dos motivos da linguagem JavaScript ser tão popular é a ferramenta Node.js, bastante adotada atualmente em servidores Web. Um dos motivos cruciais é a troca conveniente que ele oferece entre produtividade e desempenho. Isso faz com que

desenvolvedores não percam muito tempo para construir uma aplicação que tenha um desempenho muito bom (SUN et al., 2018). Ele é uma plataforma orientada a eventos, forçando o programa a ser escrito pensando em trechos sendo executados de maneira assíncrona.

### 4.1.2 AdonisJS

Para fazer uma aplicação Web com segurança atualmente é necessário pensar em muitos fatores. É essencial ter em mente boas práticas de como guardar dados sensíveis, como senhas e cartões de crédito. Também é muito importante lidar bem com as permissões dos usuários, para que os mesmos não ganhem acesso a nada indevido. As entradas de dados também precisam de cautela para que programas maliciosos não sejam executados e um usuário sem permissão obtenha dados que não deveria.

Existe uma grande quantidade de ferramentas para desenvolvimento Web que se utilizam do Node.js. De modo geral, eles são considerados micro ferramentas que se especializam em resolver em resolver um único problema, como, por exemplo o Express para o roteamento ou o Sequelize para realizar o ORM(*Object Relational Mapping*). Desta maneira, o desenvolvedor deve juntar todos essas micro ferramentas para facilitar a criação de um sistema Web, elevando o custo de manutenção da aplicação.

Optamos pelo uso do AdonisJS[1], pois é um *framework* em JavaScript que possui todos os recursos necessários para uma aplicação Web completamente funcional. Já que não é um *framework* minimalista, para gerar uma aplicação inicial com o AdonisJS se exige um bom entendimento dos seus arquivos de configuração. A sua estrutura foi inspirada, e por conta disso é muito similar, a de projetos em *frameworks* muito bem avaliados de outras linguagens, como *Ruby on Rails*, para ruby, e *Laravel*, para PHP.

O AdonisJS tem um ORM (*Object Relational Mapper*) bem completo. Ele facilita o acesso e a manipulação de informações no banco de dados, além de possuir suporte

[1]https://adonisjs.com/

para diversos bancos relacionais, como PostgreSQL, MySQL, MSSQL, MariaDB e SQLite. Outra funcionalidade muito importante são os sistemas de autenticação de usuários. Apesar de funcionar muito bem sem muito esforço ao instalar o Adonis, a autenticação é bastante personalizável através dos arquivos de configuração.

Para ajudar no desenvolvimento, o Ace, um *framework* de linha de comando para operar arquivos e ações relacionadas ao projeto também pode ser usado. Existem muitos comandos como base para facilitar o gerenciamento do projeto, mas também é possível criar seus próprios comandos caso seja necessário. Em adição, é oferecido um REPL (*read-eval-print-loop*), uma maneira de rapidamente executar linhas de comando e receber *feedback* instantâneo.

Para usuários que gostam de se autenticar usando serviços de terceiros, o AdonisJS permite integração de autenticação utilizando provedores como a Google, Twitter, GitHub e Facebook. Também é possível configurar um provedor personalizado. E para finalizar, o AdonisJS contém um sistema que monitora a cada período de tempo se a aplicação está recebendo mensagens HTTP, está com acesso ao banco de dados e se nenhuma configuração está desatualizada. Isso é ótimo para conferir se tudo está como deveria.

### 4.1.3 PostgreSQL

PostgreSQL[2] é um sistema de banco de dados relacional e de código aberto com desenvolvimento constante desde que foi criado, no ano de 1986. Tem uma comunidade muito grande por trás e, além disso, uma ótima reputação sobre os aspectos de robustez, desempenho e confiabilidade. O PostgreSQL usa e estende a linguagem SQL (Standard Query Language) combinada com muitos recursos de segurança para armazenar e dimensionar cargas de trabalho complexas.

Esse sistema de banco de dados tem diversas funcionalidades criadas para ajudar desenvolvedores e administradores. Ele permite construir arquiteturas com proteção sobre a integridade nos dados e tolerância a falhas do ambiente, independente se o conjunto de dados é pequeno ou grande. Também é altamente personalizável,

[2]https://www.postgresql.org/about/

possibilitando ao desenvolvedor criar suas próprias funções e tipos de dados. É possível até mesmo utilizar códigos de diferentes linguagens de programação.

O PostgreSQL foi escolhido pela sua flexibilidade, confiabilidade dos dados e bom desempenho em conjuntos grandes e pequenos de dados. Em um estudo feito por Wiseso, Imrona e Alamsyah (2020), onde são feitas comparações sobre alguns sistemas de banco de dados bastante usados hoje em dia, o PostgreSQL desempenhou muito bem na maioria dos casos. Mesmo com entradas de dados na ordem de $10^6$ houve uma grande consistência.

Outro ponto importante para a escolha foi o suporte aos tipos JSON e JSONB (JSON Binary). Em SaaS costuma-se trabalhar bastante com esse tipo de dados, então um bom suporte a ele é crucial. Além de que a enorme quantidade de tipos primitivos, e a possibilidade de criar tipos personalizados, permite uma flexibilidade enorme para um projeto de Inteligencia Artificial.

## 4.2 Arquitetura do Sistema

Como falado no capítulo anterior, optamos pela arquitetura de microsserviços e pela presença de um gateway para lidar com as requisições dos usuários. O Capivara foi implementado usando o modelo MVC (Model View Controller). Pelo nome já é possível inferir que ele divide a aplicação em três camadas: Models (Modelos), Views (Visualizações) e Controllers (Controladores). Também é possível usar mais algumas camadas para melhor encapsular o sistema. Alguns exemplos comuns são a camada de Service (Serviço), para tratar do processamento de informação usando os modelos e as regras de negócio.

Segundo Pop e Altar (2014), a camada de modelos é onde são gerenciadas apenas as tarefas relacionadas aos dados. Algumas dessas tarefas são a definição de seus atributos e suas validações. Nessa camada também se encontra o código com as tarefas que necessitam de acesso aos dados, seja através de um banco de dados ou alguma outra implementação. Com isso, ela se torna uma biblioteca de classe reutilizável (POP; ALTAR, 2014).

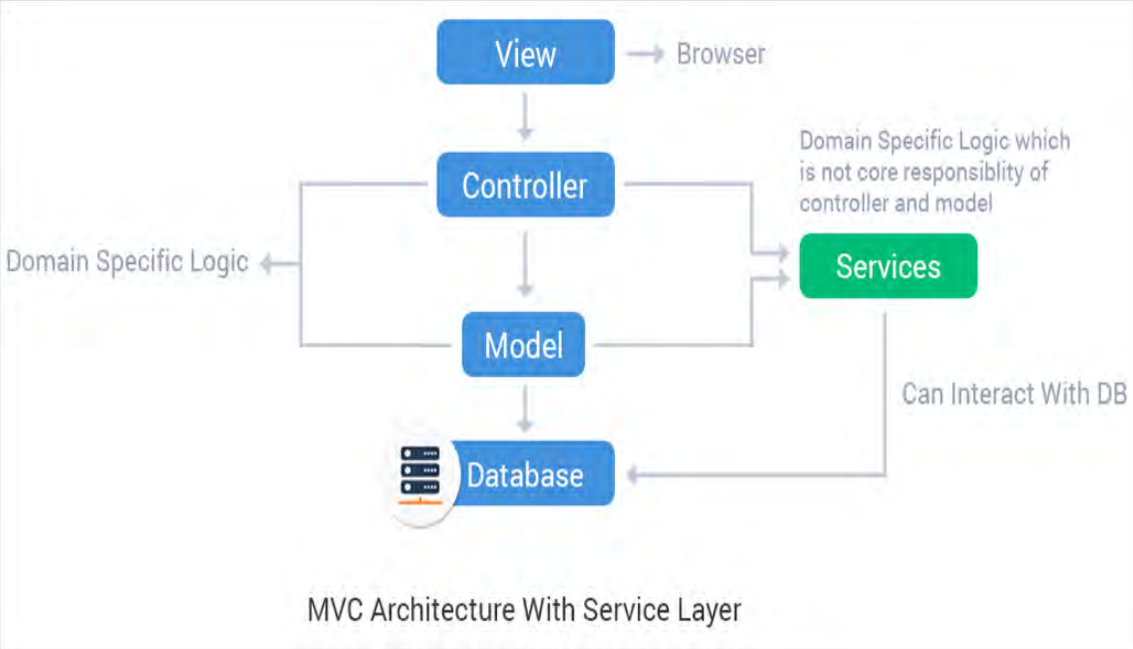


Figura 4.1: Modelo MVC. Retirado de <https://harbinger-systems.com/blog/2015/08/service-layer-in-rails-application>

A camada Views tem o objetivo de gerenciar a parte gráfica do sistema. Todos os elementos gráficos, interações com o usuário e a responsabilidade de enviar os dados corretos para a camada Controller é dela (POP; ALTAR, 2014). Seguindo, a camada Controllers é responsável por gerenciar os eventos do sistema. Esses eventos podem ser requisições emitidas por alguma interação do usuário, requisições externas, ou até mesmo uma troca de mensagens com algum outro sistema local. Ao receber um evento, o Controller se comunica com o Model ou com o Service e obtém os dados para enviar a resposta adequada.

A camada Services contém todas as regras de negócio do sistema. Por conta disso, nela ocorre o processamento e a validação dos dados. Essa camada recebe do Controller os dados que serão usados posteriormente pelo Model. Depois disso, o Model realiza seu trabalho, devolve os dados para o Service, que faz qualquer transformação ou validação necessária e os envia novamente para o Controller. No capivara usamos todas as 4 camadas.

Como toda a implementação foi feita na língua inglesa, criamos os modelos Users (Usuários) e Skills (Habilidades). Também existe um modelo chamado ApiToken

(Token de segurança), mas ele é criado automaticamente pelo Adonis para ser usado em um de seus mecanismos de autenticação. Optamos por deixá-los o mais simples possível, diminuindo ao máximo o processamento de dados nessa camada, como podemos ver na 4.2. O único processamento presente é a função de hash da senha por questões de segurança. Esse trecho é executado toda vez que o usuário é salvo no sistema, mas apenas criptografa a senha caso o usuário tenha alterado.

```
@column({ isPrimary: true })
public id: number

@column()
public email: string

@column()
public name: string

@beforeSave()
public static async hashPassword(user: User) {
  if (user.$dirty.password) {
    user.password = await Hash.make(user.password)
  }
}
```

Figura 4.2: Único processamento realizado na camada Model de Usuários

Criamos os controladores UserController e SkillController para lidar com a gerência desses dois modelos pelo usuário, mas também criamos o GatewayController, presente na figura 4.3. Nele serão captados os eventos emitidos pelo usuário quando ele executar alguma habilidade. Capturamos o texto enviado pelo usuário e repassamos para o Service processá-lo. No GatewayService, como podemos ver na figura 4.4, verificamos se a habilidade que o usuário deseja usar existe, e, caso positivo, é realizado o chamado do serviço externo. Esse serviço pode estar implementado no mesmo servidor, porém como não faz parte direta da aplicação, ainda é um serviço externo.

Nas rotas, como pode ser visto na figura 4.5, separamos o tipo de autenticação das requisições. As requisições que lidam diretamente com a aplicação principal usam a autenticação Web. Nela, usamos a sessão do navegador para verificar qual usuário está conectado e enviando as requisições. Já a autenticação feita pelo gateway usa a

```
export default class GatewayController {
  public async call({ params, request }: HttpContextContract) {
    const data = request.input('text')

    const result = await GatewayService.call(params.path, data)

    return result
  }
}
```

Figura 4.3: Implementação do Controller do Gateway

```
6   export default class GatewayService {
7     private static async invoke(url: string, data: string) {
8       const axiosConfig = {
9         method: 'post',
10        url: url,
11        data: { 'text': data },
12      }
13
14      try{
15        let res = await axios(axiosConfig)
16        return res.data
17      } catch (err) {
18        throw new Exception(err)
19      }
20    }
21
22    public static async call(skillPath: string, data: string) {
23      const skill = await Skill.findByOrFail('path', skillPath)
24
25      return await this.invoke(skill.serverUrl, data);
26    }
27  }
28
```

Figura 4.4: Implementação do Gateway Service para chamadas a serviços externos

autenticação via Token de segurança. Dessa forma, não é necessário o mecanismo de um navegador para enviar requisições. Isso ajuda bastante para enviar requisições autenticadas usando apenas comandos do terminal.

É possível implementar funções que executam antes de a requisição ser efetivamente enviada ao Controller. Essas funções têm o nome de Middleware, e funcionam

como uma camada intermediária entre a requisição HTTP e o sistema. No Capivara usamos apenas o middleware de autenticação, próprio do Adonis. Ele é utilizado para identificar qual usuário está enviando a requisição, seja utilizando o navegador ou incluindo o token de segurança nela.

```
Route.group(() => {
  Route.get('/skills', 'SkillsController.index').as('skills.index')
  Route.get('/skills/edit/:id', 'SkillsController.update').as('skills.update').where('id', /^[0-9]+$/)
  Route.get('/skills/create', 'SkillsController.create').as('skills.create')
  Route.delete('/skills/:id', 'SkillsController.destroy').as('skills.destroy')
  Route.get('/skills/:id', 'SkillsController.show').as('skills.show').where('id', /^[0-9]+$/)
  Route.post('/skills', 'SkillsController.store').as('skills.store')
}).prefix('/admin').middleware(['auth:web'])

Route.group(() => {
  Route.get('/skills/:id', 'SkillsController.about').as('skills.about').where('id', /^[0-9]+$/)
  Route.post('/skills/:id', 'SkillsController.test').as('skills.test').where('id', /^[0-9]+$/)
}).middleware(['auth:web'])
```

Figura 4.5: Código sobre as rotas do sistema utilizando Adonis.js

Por último, serão mostradas as Views do sistema. Usamos as ferramentas de renderização nativas do Adonis, o engine template chamado Edge. Um engine template é uma ferramenta para lidar com páginas HTML de forma dinâmica utilizando código diretamente no arquivo HTML. Isso ajuda muito na legibilidade, pois evita arquivos de script carregados diretamente na página. Existem muitos engine template, mas o Adonis optou por criar o seu e deu o nome de Edge.

Edge é um engine template para o Node.js e pode renderizar qualquer formato baseado em texto, seja HTML, Markdown ou arquivos simples de texto. Foi criado porque os outros engine template tinham detalhes desvantajosos para uma ferramenta em Javascript, principalmente com relação à sintaxe de código no HTML. Por exemplo, o engine chamado Pug deixa muito diferente a escrita do código HTML, enquanto um outro chamado Nunjucks se parece muito mais com a linguagem Python. Essas diferenças apenas aumentam a curva de aprendizado e por conta disso, os criadores do Adonis também criaram essa nova engine template.

Além disso, utilizamos também a ferramenta TailwindCSS. Essa ferramenta apresenta diversas classes utilitárias pré definidas. Isso ajuda na estilização dos arquivos HTML, pois classes básicas como estilos de fonte, tamanho do texto e alinhamento do texto já estão definidas. Além disso, fizemos de forma responsiva, ou

seja, funcionará para diferentes resoluções de tela e dispositivos.

A página inicial do sistema, apresentada em diferentes resoluções pelas Figuras 4.6, 4.7 e 4.8, apresenta de forma geral o objetivo do Capivara. A partir dela, é possível acessar a página de cadastro e a página de login. Para realizar o cadastro basta preencher o formulário com os dados corretos e enviar. Já para o login, apenas o e-mail e a senha são necessários para realizar o acesso. O cadastro e o login são apresentados nas Figuras 4.9 e 4.10.

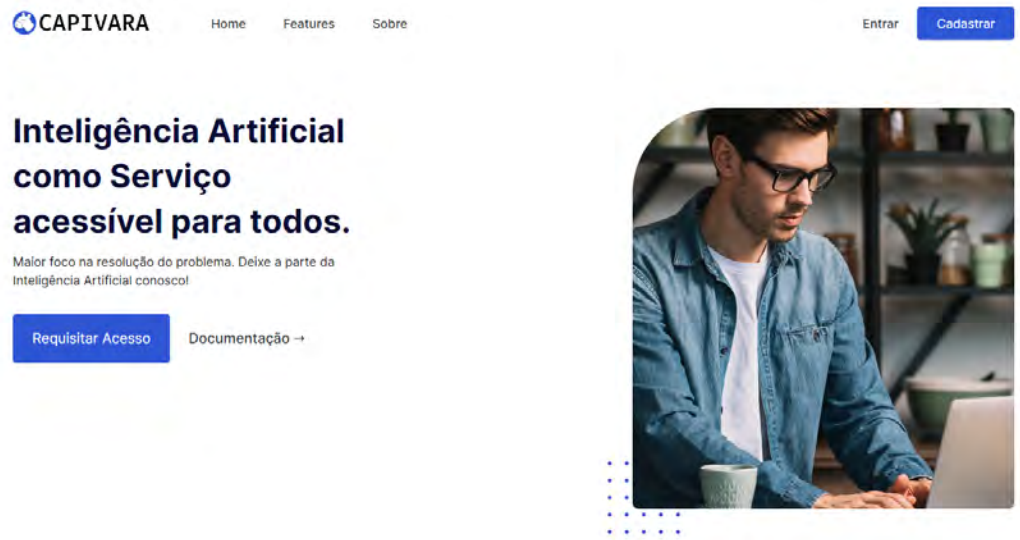


Figura 4.6: Página Inicial em tamanho completo

Quando um usuário comum acessa o sistema, são poucas as ações permitidas pelo sistema. Além de ser possível visualizar as habilidades, e testá-las enviando um texto como entrada, também é possível visualizar o seu token de segurança e alterar o seu nome ou e-mail. O token serve para enviar requisições em contextos fora do navegador. Por exemplo, é possível utilizar o comando curl diretamente do terminal e executar a habilidade autenticado com o seu usuário. Essas ações são indicadas nas Figuras 4.12, 4.11, 4.13, 4.14 e 4.15.

Já como administrador, o controle sobre ações é muito maior. Como o usuário cria a sua própria conta, não há necessidade do administrador realizar essa ação, mas é possível visualizar as informações dos usuários, alterar suas informações e transformá-lo como administrador, remover esse privilégio, ou apagar o usuário, como


Figura 4.7: Página Inicial em versão reduzida

podemos ver na Figura 4.16. Já em respeito as habilidades, o administrador pode fazer o CRUD completo. Permitimos que ele crie, visualize, faça alterações ou até mesmo apague a habilidade. Também permitimos que ele teste as habilidades.


Figura 4.8: Página Inicial para dispositivos móveis

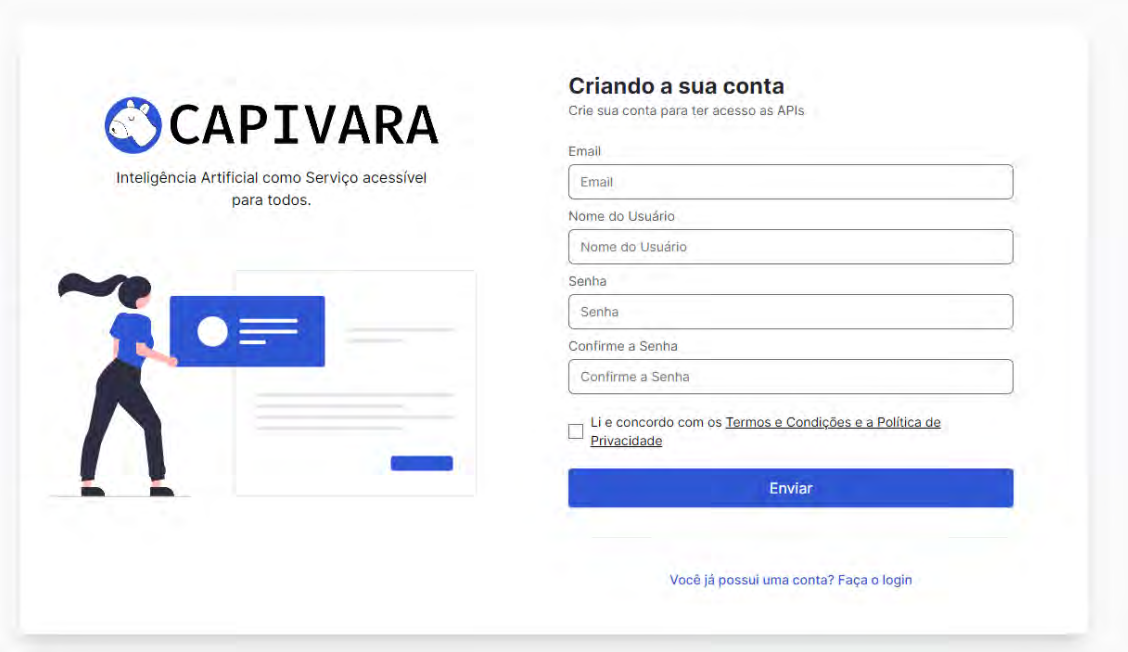


Figura 4.9: Cadastro


Figura 4.10: Login

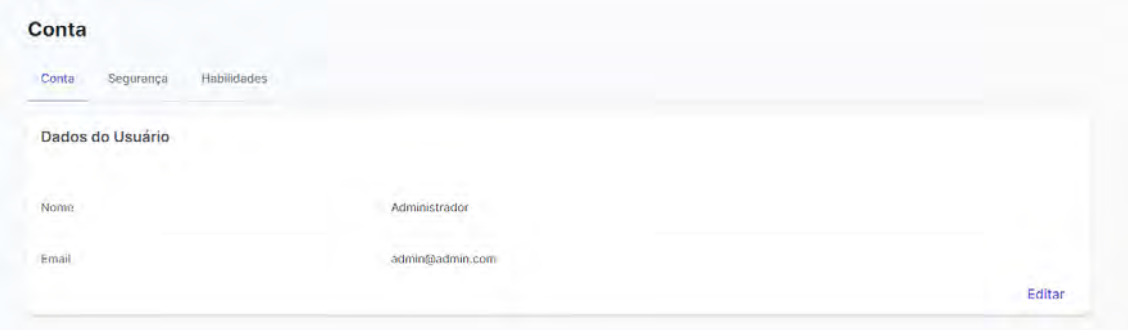


Figura 4.11: Listagem de Habilidades


Figura 4.12: Listagem de Habilidades

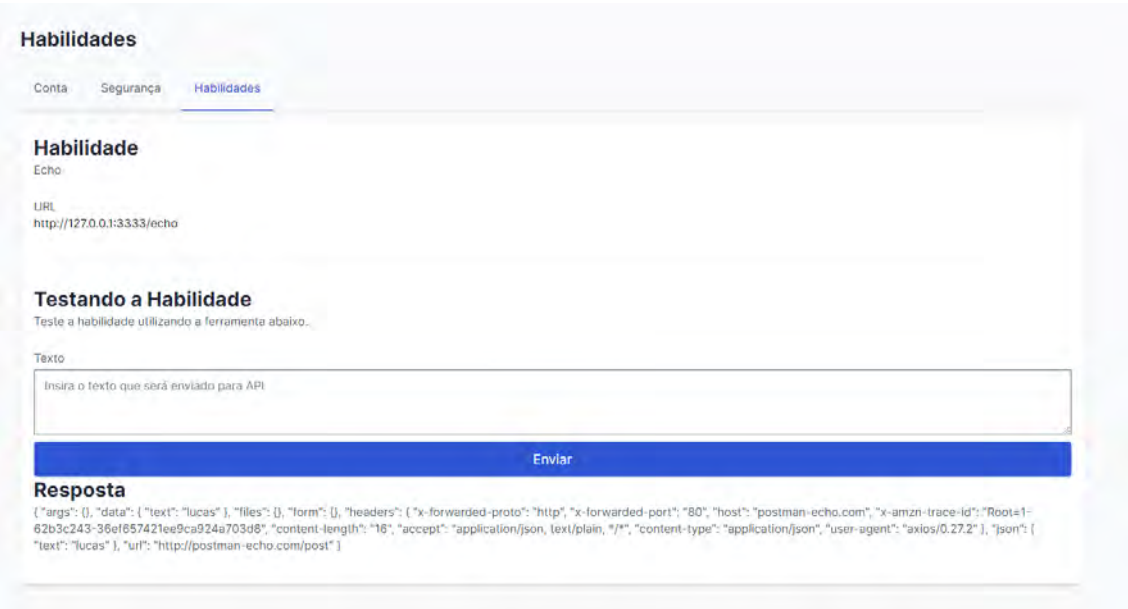


Figura 4.13: Execução de Habilidades diretamente pelo Sistema


Figura 4.14: Token de segurança


Figura 4.15: Execução de Habilidades pelo Terminal

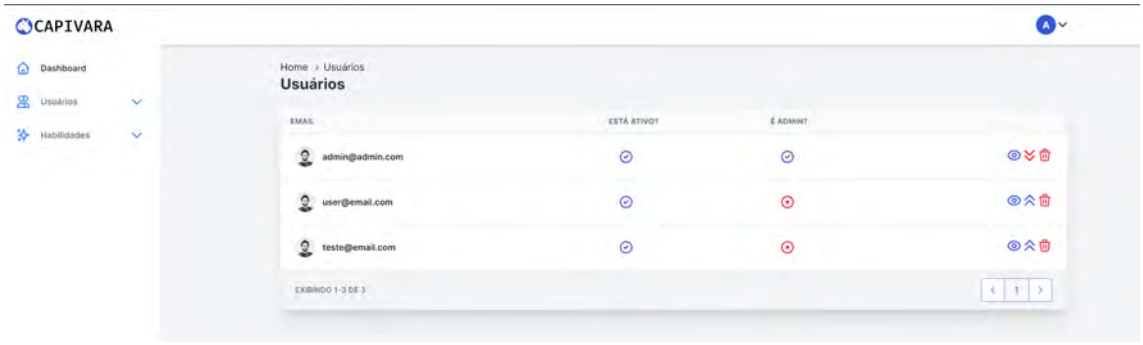


Figura 4.16: Listagem de Usuários

# Capítulo 5

# Conclusão

Este capítulo apresentará as considerações finais sobre o trabalho, retomando o contexto atual das empresas que usam inteligência artificial como negócio, suas limitações e ideias futuras para a melhora desse projeto.

## 5.1 Considerações finais

Levando em conta que sistemas Web e a área de inteligência artificial estão crescendo a cada dia, foi proposto nesse trabalho um projeto que une esses dois conceitos e tenta utilizar o máximo de cada um deles com relação à transparência e usabilidade para pessoas que desejam utilizar Inteligência Artificial em seus negócios, mesmo que não tenham muito conhecimento sobre código e sistemas. Foram utilizadas diversas estratégias de computação em nuvem para implementá-lo e a ferramenta Adonis.js, com alto nível de manutenção e que apresenta muita facilidade para o desenvolvedor.

Sendo assim, o projeto Capivara apresenta uma boa alternativa para centralizar modelos de inteligência artificial e oferecê-los através de serviços web. Muitas empresas desejam investir em inteligência artificial, mas faltam recursos financeiros para obter equipes e equipamentos altamente especializados para tal. Esses são os casos em que o Capivara mais se faz útil, já que o recurso computacional não é de

responsabilidade do cliente.

## 5.2 Limitações e trabalhos futuros

Tendo em vista o que foi apresentado, é importante apontar algumas limitações do projeto em seu estado atual, e o que pode ser feito como melhoria para melhorá-lo. Uma possibilidade para o futuro é fazer com que o usuário tenha a capacidade de gerenciar suas bases de dados na plataforma, além da inclusão de uma ferramenta para visualização desses dados. Como nessa área a base de dados é uma das coisas mais importantes, esse ponto pode ser um atrativo muito interessante para buscar novos clientes. Esse poder de manipular a base de dados diretamente da plataforma também facilita muito outra melhoria, chamadas encadeadas dos serviços. Muitos serviços têm diferentes formatos de entrada e saída, o que dificulta muito encadear chamadas no Capivara mediante o cenário atual. Porém, caso seja possível manipular os dados, essa tarefa passa a ser fácil. Basta aplicar uma transformação entre uma chamada e outra.

Atualmente o sistema comporta apenas um único modelo de banco de dados. Como bancos de dados relacionais e não relacionais tem uma estrutura diferente, será necessário adaptar a maneira como os dados seriam manipulados e visualizados de forma que isso seja invisível para o usuário. Poderíamos também suportar bancos de dados de grafos. Atualmente eles estão ganhando muita visibilidade e empresas estão cada vez mais adotando esse modelo de banco.

Outro atrativo muito interessante em alguns trabalhos relacionados é a presença de uma interface de linha de comando. Já que é bem comum que contratantes tenham pelo menos contato com o time de desenvolvimento responsável pelo projeto, esse tipo de interface é de grande ajuda caso seja necessário realizar automações. Além disso, poderia ser implementado um sistema de automação na própria interface gráfica do Capivara como adicional.

Também seria muito interessante a presença de um sistema agendador. Isso permite que usuários agendem tarefas a serem executadas sempre em intervalos

definidos. Ademais, poderíamos futuramente coletar informação sobre as execuções dos serviços e apresentar diversas estimativas para o usuário. Com base no tamanho da entrada e no tempo de execução de diversas execuções passadas, é possível estimar para o usuário quanto tempo levará a tarefa antes mesmo de agenda-la.

Incluir mecanismos de mensageria no sistema trará muitos benefícios por conta de seus altos índices de escalabilidade. É muito interessante para sistemas distribuídos já que facilita bastante a comunicação assíncrona entre os sistemas. Além disso, ajuda na tolerância de falhas caso ocorra erro em algum dos servidores no meio de uma comunicação. Isso se dá pela persistência dos dados nas filas de mensagem.

# Referências


AGRAWAL, D.; DAS, S.; ABBADI, A. E. Big data and cloud computing. In: . [S.l.]: ACM Press, 2011. p. 530. ISBN 9781450305280.

AVRAM, M. Advantages and challenges of adopting cloud computing from an enterprise perspective. *Procedia Technology*, v. 12, p. 529–534, 2014. ISSN 22120173.

BISHOP, C. *Pattern Recognition and Machine Learning*. Springer, 2006. (Information Science and Statistics). ISBN 9780387310732. Disponível em: <https://books.google.com.br/books?id=qWPwnQEACAAJ>.

CAMPBELL-KELLY, M. Historical reflections the rise, fall, and resurrection of software as a service. *Communications of the ACM*, ACM New York, NY, USA, v. 52, n. 5, p. 28–30, 2009.

CETINSOY, A. et al. The past, present, and future of machine learning apis. In: DORARD, L.; REID, M. D.; MARTIN, F. J. (Ed.). *Proceedings of The 2nd International Conference on Predictive APIs and Apps*. Sydney, Australia: PMLR, 2016. (Proceedings of Machine Learning Research, v. 50), p. 43–49. Disponível em: <https://proceedings.mlr.press/v50/cetinsoy15.html>.

CHURCH, K. W. Emerging trends: Artificial intelligence, china and my new job at baidu. *Natural Language Engineering*, Cambridge University Press, v. 24, n. 4, p. 641–647, 2018.

FIELDING, R. T. *REST: Architectural Styles and the Design of Network-based Software Architectures*. Tese (Doctoral dissertation) — University of California, Irvine, 2000. Disponível em: <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>.

FOX, A.; PATTERSON, D. *Engineering Software as a Service: An Agile Approach Using Cloud Computing*. Strawberry Canyon LLC, 2013. ISBN 9780984881246. Disponível em: <https://books.google.com.br/books?id=3kqjmwEACAAJ>.

HIGH, R. The era of cognitive systems: An inside look at ibm watson and how it works. *IBM Corporation, Redbooks*, v. 1, p. 16, 2012.

HURWITZ, J.; KIRSCH, D. *Cloud Computing For Dummies*. Wiley, 2020. ISBN 9781119546658. Disponível em: <https://books.google.com.br/books?id=yxNuuQEACAAJ>.

KIM, H. et al. Nsml: Meet the mlaas platform with a real-world case study. 10 2018.

KUZNETSOV, V.; GIOMMI, L.; BONACORSI, D. Mlaas4hep: Machine learning as a service for hep. *Computing and Software for Big Science*, v. 5, p. 17, 12 2021. ISSN 2510-2036.

LANGLEY, P. *Elements of machine learning*. Morgan Kaufmann, 1996. 5-12 p. ISBN 1558603018; 9781558603011. Disponível em: <libgen.li/file.php?md5=06d686bd550558a613ea46e924941a18>.

LIU, S.; KUHN, R. Data loss prevention. *IT Professional*, v. 12, p. 10–13, 3 2010. ISSN 1520-9202.

MICHON, R. *The Complete Guide to Software As a Service: Everything You Need to Know About Saas*. CreateSpace Independent Publishing Platform, 2017. ISBN 9781546308492. Disponível em: <https://books.google.com.br/books?id=PjmrswEACAAJ>.

MITCHELL, T. *Machine Learning*. McGraw-Hill, 1997. (McGraw-Hill International Editions). ISBN 9780071154673. Disponível em: <https://books.google.com.br/books?id=EoYBngEACAAJ>.

NAQA, I. E.; MURPHY, M. J. *What Is Machine Learning?* [S.l.]: Springer International Publishing, 2015. 3-11 p.

POP, D. Machine learning and cloud computing: Survey of distributed and saas solutions. 3 2016.

POP, D.-P.; ALTAR, A. Designing an mvc model for rapid web application development. *Procedia Engineering*, v. 69, p. 1172–1179, 2014. ISSN 1877-7058. 24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013. Disponível em: <https://www.sciencedirect.com/science/article/pii/S187770581400352X>.

PROVOST, F.; FAWCETT, T. *Data Science for Business*. O'Reilly, 2013. ISBN 9781449361327. Disponível em: <https://books.google.com.br/books?id=_1b4nAEACAAJ>.

RIBEIRO, M.; GROLINGER, K.; CAPRETZ, M. A. Mlaas: Machine learning as a service. In: . [S.l.]: IEEE, 2015. p. 896–902. ISBN 978-1-5090-0287-0.

RICHARDSON, C. *Microservices Patterns: With examples in Java*. Manning, 2018. ISBN 9781617294549. Disponível em: <https://books.google.com.br/books?id=UeK1swEACAAJ>.

RICHARDSON, L.; RUBY, S. *RESTful Web Services*. O'Reilly Media, 2008. ISBN 9780596554606. Disponível em: <https://books.google.com.br/books?id=XUaErakHsoAC>.

ROSALES, M. A. et al. Artificial intelligence: The technology adoption and impact in the philippines. In: . [S.l.]: IEEE, 2020. p. 1–6. ISBN 978-1-6654-1971-0.

SOWMYA, S.; DEEPIKA, P.; NAREN, J. Layers of cloud–iaas, paas and saas: a survey. *International Journal of Computer Science and Information Technologies*, v. 5, n. 3, p. 4477–4480, 2014.

SUN, H. et al. Efficient dynamic analysis for node.js. In: *Proceedings of the 27th International Conference on Compiler Construction*. New York, NY, USA: Association for Computing Machinery, 2018. (CC 2018), p. 196–206. ISBN 9781450356442. Disponível em: <https://doi.org/10.1145/3178372.3179527>.

WISESO, L. G.; IMRONA, M.; ALAMSYAH, A. Performance analysis of neo4j, mongodb, and postgresql on 2019 national election big data management database. In: . [S.l.]: IEEE, 2020. p. 91–96. ISBN 978-1-7281-7347-4.