**CENTRO PAULA SOUZA**

**FACULDADE DE TECNOLOGIA DE MOGI MIRIM**

**Everton do Prado**

**Luana Alves dos Santos**

**Luís Carlos Lima dos Santos**

**Tarciso de Jesus Bandeira**

**APLICAÇÃO MULTIPLATAFORMA PARA SOLICITAÇÃO DE SOCORRO PARA AUTOMOVÉIS**

**MOGI MIRIM**

**2022**

**Everton do Prado**

**Luana Alves dos Santos**

**Luís Carlos Lima dos Santos**

**Tarciso de Jesus Bandeira**

**APLICAÇÃO MULTIPLATAFORMA PARA SOLICITAÇÃO DE SOCORRO PARA AUTOMOVÉIS**

Trabalho de Graduação apresentado ao Curso de Tecnologia em Análise e Desenvolvimento de Sistemas da Faculdade de Tecnologia de Mogi Mirim como pré-requisito para a obtenção do Título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

**Orientador:**

**MOGI MIRIM**

**2022**

FOLHA DE APROVAÇÃO

Será entregue pela coordenação

Dedicamos este trabalho à ..... pois .................Destinada a homenagear alguém importante na vida do autor. (Dedicatória é OPCIONAL)

**AGRADECIMENTOS**

Elemento optativo, destinado àqueles cujo auxílio foi fundamental para a concretização do trabalho. (Apresenta-se sob a mesma configuração da introdução, ver elementos textuais). Devem ser inseridos após a dedicatória.

*“A tecnologia move o mundo;”*

*(Steve Jobs)*

**Lista de Figuras**

[Figura 1 - Estrutura lógica do método científico 19](#_Toc121070738)

[Figura 2 - Etapas da metodologia científica 20](#_Toc121070739)

[Figura 3 - CHAOS Report (1994) 27](#_Toc121070740)

[Figura 4 - Processo *Waterfall* 28](#_Toc121070741)

[Figura 5 - Processo Ágil 28](#_Toc121070742)

[Figura 6 - Pirâmide de testes 30](#_Toc121070743)

[Figura 7 - Exemplos de Classes 35](#_Toc121070744)

[Figura 8 - Exemplo de Associação entre Classes 36](#_Toc121070745)

[Figura 9 - Exemplo de Herança entre Classes 37](#_Toc121070746)

[Figura 10 - Exemplo de Dependência entre Classes 38](#_Toc121070747)

[Figura 11 - Exemplo de Diagrama de Atividades 39](#_Toc121070748)

[Figura 12 - Exemplo de Nodo Inicial 40](#_Toc121070749)

[Figura 13 - Exemplo de Ações 40](#_Toc121070750)

[Figura 14 - Exemplo de Decisões 40](#_Toc121070751)

[Figura 15 - Exemplo de *Merges* 41](#_Toc121070752)

[Figura 16 - Exemplo de *Forks* 41](#_Toc121070753)

[Figura 17 - Exemplo de *Joins* 42](#_Toc121070754)

[Figura 18 - Exemplo de Nodo Final 42](#_Toc121070755)

[Figura 19 - Tabela Exemplo Modelo Relacional 44](#_Toc121070756)

[Figura 20 - História das linguagens de programação 46](#_Toc121070757)

[Figura 21 - Fluxo final com os comandos *push* e *pull* 59](#_Toc121070758)

[Figura 22 - Calculadora de amostra da Survey Monkey 61](#_Toc121070759)

[Figura 23 - Tela de Apresentação 73](#_Toc121070760)

[Figura 24 - Tela de Apresentação 2 74](#_Toc121070761)

[Figura 25 - Tela de *Login* 75](#_Toc121070762)

[Figura 26 - Tela de Cadastro 76](#_Toc121070763)

[Figura 27 - Tela Inicial 77](#_Toc121070764)

[Figura 28 - Tela de Busca 78](#_Toc121070765)

[Figura 29 - Tela de Filtros 79](#_Toc121070766)

[Figura 30 - Tela de Categoria de Serviços 80](#_Toc121070767)

[Figura 31 - Tela de Seleção de Serviços 81](#_Toc121070768)

[Figura 32 - Tela de Detalhes do Serviço 82](#_Toc121070769)

[Figura 33 - Tela de Agendamento 83](#_Toc121070770)

[Figura 34 - Tela dos Dados 84](#_Toc121070771)

[Figura 35 - Tela de Pagamento 85](#_Toc121070772)

[Figura 36 - Tela de Adicionar Novo Cartão 86](#_Toc121070773)

[Figura 37 - Tela de Localização 87](#_Toc121070774)

[Figura 38 - Tela de Histórico 88](#_Toc121070775)

[Figura 39 - Tela de Favoritos 89](#_Toc121070776)

[Figura 40 - Tela de Perfil do Usuário 90](#_Toc121070777)

**LISTA DE ABREVIATURAS E SIGLAS**

ABNT – Associação Brasileira de Normas Técnicas

SGBD – Sistema Gerenciador de Banco de Dados

XP – *Extreme Programming*

**RESUMO**

**Palavras-chave:** Solução; React; Web; Mobile; Veículos.

***ABSTRACT***

*The present project proposes to develop a Web and Mobile solution using React technology, in order to solve the problem found in the day-to-day scenario of thousands of drivers who tend to deal with failures and unexpected technical problems with their vehicles. The solution, in addition to providing emergency assistance to its users, proposes to help in the search for personalized services for their vehicles. To progress with the development and present the solution raised based on the hypothesis methodology, the project applies agile methodology techniques in order to deliver a modern, simple and safe solution that meets the proposed requirements.*

***Keyword:*** *Solution; React; Web; Mobile; Vehicles.*

Sumário

[Sumário 13](#_Toc121070900)

[1. INTRODUÇÃO 14](#_Toc121070901)

[1.1. Estrutura do Trabalho 16](#_Toc121070902)

[2. METODOLOGIA DA PESQUISA CIENTÍFICA 18](#_Toc121070903)

[3. FUNDAMENTAÇÃO TEÓRICA 24](#_Toc121070904)

[3.1. Engenharia de Software 24](#_Toc121070905)

[3.2. Linguagem de Programação 45](#_Toc121070906)

[4. DESENVOLVIMENTO 59](#_Toc121070907)

[4.1. Levantamento de Requisitos 61](#_Toc121070908)

[4.2. Proposta de Solução 70](#_Toc121070909)

[**REFERÊNCIAS BIBLIOGRÁFICAS** 91](#_Toc121070910)

# INTRODUÇÃO

## Estrutura do Trabalho

O presente trabalho está dividido em XX capítulos:

O primeiro capítulo fala sobre a

O capítulo 2 diz respeito a

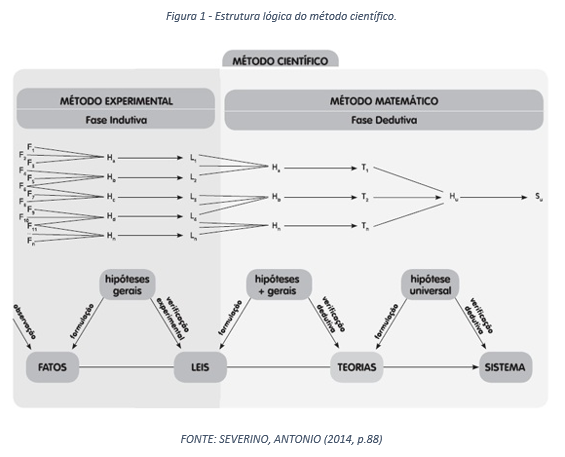
O terceiro capítulo  
No quarto capítulo

# METODOLOGIA DA PESQUISA CIENTÍFICA

Segundo Severino (2014) a ciência usa seu método, elemento básico do processo de conhecimento que realiza, para distinguir o método científico, não apenas do senso comum, mas também da expressão da subjetividade humana como filosofia, arte e religião. Enquanto para Lakatos (2003) o conhecimento científico diferencia-se do popular muito mais no que se refere ao seu contexto metodológico do que propriamente ao seu conteúdo. E isso, ocorre também em relação aos conhecimentos filosófico e religioso(teológico). Constitui um conhecimento contingente, pois suas proposições ou hipóteses têm sua veracidade ou falsidade conhecida através da experiência e não apenas pela razão, como ocorre no conhecimento filosófico. É sistemático, já que se trata de um saber ordenado logicamente, formando um sistema de ideias (teoria) e não conhecimentos dispersos e desconexos. Possui a característica da verificabilidade, a tal ponto que as afirmações (hipóteses) que não podem ser comprovadas não pertencem ao âmbito da ciência. Constitui-se em conhecimento falível, em virtude de não ser definitivo, absoluto ou final e, por este motivo, é aproximadamente exato: novas proposições e o desenvolvimento de técnicas podem reformular o acervo de teoria existente. (LAKATOS, 2003, p.80).

Este é um conjunto de procedimentos lógicos e técnicos operacionais que permitem estabelecer relações causais permanentes entre fenômenos. Com base no conhecimento científico, a ciência através de seus conjuntos de atitudes e atividades racionais, busca alcançar objetivos e detectar erros e auxiliar nas decisões científicas, traçando um caminho válido que se utiliza metodologias científicas, segundo Lakatos (2003).

Figura - Estrutura lógica do método científico



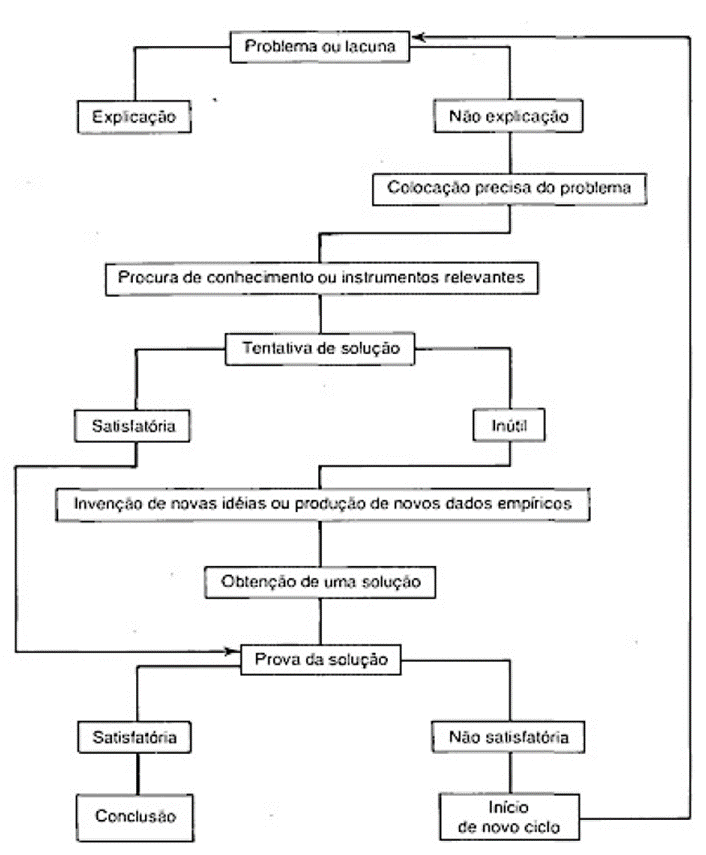
Fonte: SEVERINO, Antônio (2014, p.88)

Com base na figura 1 acima podemos afirmar que:

A ciência utiliza-se de um método que lhe é próprio, o método científico, elemento fundamental do processo do conhecimento realizado pela ciência para diferenciá-la não só do senso comum, mas também das demais modalidades de expressão da subjetividade humana, como a filosofia, a arte, a religião. Trata-se de um conjunto de procedimentos lógicos e de técnicas operacionais que permitem o acesso às relações causais constantes entre os fenômenos. O método científico pode ser representado. (SEVERINO, 2014, p.88).

Com base na observação dos fatos e no conceito de metodologia, podemos começar a construir uma fundamentação científica, que segundo Lakatos (2003), a preocupação em descobrir e, portanto, explicar a natureza vem desde os primórdios da humanidade, mas com o passar do tempo, muitas modificações foram feitas nos métodos existentes.

Figura - Etapas da metodologia científica



FONTE: LAKATOS, EVA MARIA (2003, p. 85)

Com base na figura 2 acima podemos afirmar que:

No momento, o que nos interessa é o conceito moderno de método (independentemente do tipo). Para tal, consideramos que o método científico é a teoria da investigação. Esta alcança seus objetivos, de forma científica, quando cumpre ou se propõe a cumprir as etapas de acordo com a figura. (LAKATOS, 2003, p.85)

Portanto, usando o método experimental de forma indutiva, que para Lakatos (2003), o objetivo dos argumentos indutivos é levar a conclusões cujo conteúdo é muito mais amplo do que o das premissas nas quais se buscaram. Uma característica que não pode deixar faltar de ser assinalada é que o argumento indutivo, da mesma forma que o dedutivo, fundamenta-se em premissas. Mas, se nos dedutivos, premissas verdadeiras levam inevitavelmente à conclusão verdadeira, já nos indutivos, conduzem apenas a conclusões prováveis.

Portanto, tratamos de fatos, cujo, observando esses fatos, começamos a formular o trabalho, para depois implicar na verificação experimental, dentro das leis científicas. Segundo Severino (2014) mas os fatos por si só não explicam. Assim como vemos objetos caem, não podemos observar porque caem! É aqui que você precisa seguir em frente: na realidade não existem "fatos concretos", propriamente falando; quando "observamos" os fatos, já estamos "problematizados", sentimos certas dificuldades e já temos algum esquema perceptível. Perguntamos exatamente por que tais fatos procedem desta maneira. Então não basta ver, é preciso olhar, e para isso é preciso problematizar, e a presença de um problema é de ordem racional, lógica.

Os fatos implicam-se em nosso conhecimento que consiste no conjunto de expectativas que formam como que uma moldura. A quebra desta provoca uma dificuldade, desencadeando na problematização. Segundo Lakatos (2003), o problema é que vai desencadear a pesquisa, pois toda investigação nasce de algum problema teórico/ prático sentido. Este dirá o que é relevante ou irrelevante observar, os dados que devem ser selecionados. Esta seleção exige uma hipótese, que servirá de guia ao pesquisador.

Portanto, conforme Lakatos (2003), os métodos e as técnicas a serem empregados na pesquisa científica podem ser selecionados desde a proposição do problema, da formulação de hipótese e da delimitação do universo ou da amostra para a pesquisa. Tanto os métodos, quanto as técnicas devem adequar-se aos problemas a ser estudado, às hipóteses levantadas e que se queira confirmar, ao tipo de informantes com que se vai entrar em contato.

O rigoroso controle na aplicação dos instrumentos de pesquisa é fator fundamental para evitar os erros e defeitos resultantes das técnicas aplicadas. Sendo assim, são vários os procedimentos para a realização de coleta de dados e pesquisa.

Devido a problematização dos fatos, como instrumentos de pesquisa usamos uma pesquisa de formulário do google, para que as pessoas opinassem sobre uma solução de seus problemas do cotidiano no que diz respeito a essa pesquisa. Fica tangível que a problematização não é apenas uma especulação, mas sim um fato. Foi levantado também uma pesquisa voltada ao prestador de serviço que seria parte da solução do problema, que estão prontos e dispostos a contribuir para a solução do caso. O termo que com base no Severino (2014) é a pesquisa-ação significa que, além de compreender, visa intervir em uma situação com o objetivo de mudá-la. O conhecimento direcionado é articulado com o objetivo deliberado de mudar a situação sob investigação. Ao realizar diagnósticos e análises de uma determinada situação, a pesquisa-ação propõe mudanças a um conjunto de entidades participantes que levam ao aprimoramento dos procedimentos analisados.

Feito também uma pesquisa bibliográfica segundo o Severino:

A pesquisa bibliográfica é aquela que se realiza a partir do registro disponível, decorrente de pesquisas anteriores, em documentos impressos, como livros, artigos, teses etc. Utiliza-se de dados ou de categorias teóricas já trabalhados por outros pesquisadores e devidamente registrados. Os textos tornam-se fontes dos temas a serem pesquisados. O pesquisador trabalha a partir das contribuições dos autores dos estudos analíticos constantes dos textos. (SEVERINO, 2014, p.106).

Trazendo todas as informações necessárias para o trabalho voltado à pesquisa com levantamento de coletas de registros de livros, artigos de matérias de jornais. Para que, quando for feito todo o mapeamento, identificação da problematização no seu maior nível, levando em consideração as pesquisas feitas, levantamento de informações, mostrando que a pesquisa-ação, pode levar num resultado positivo para elucidação da problemática. Deixando toda a documentação para nível de prova e consulta posteriores, com base no que Severino (2014) escreveu sobre documentação é qualquer forma de registro e sistematização de dados, informações e as coloca em termos de análise pelo pesquisador. Pode ser entendida em três significados básicos: como técnica de coleta, organização e preservação de documentos; como ciência que processa critérios de coleta, organização, sistematização, conservação, divulgação de documentos; no âmbito da pesquisa, é uma técnica de identificação, pesquisa, pesquisa de documentos-fonte do objeto pesquisado e registro de informações retiradas dessas fontes, que serão utilizadas no processamento do trabalho.

Feito também um questionário conforme citado acima sobre pesquisa de formulário do google, para termos a ciência com base nos indivíduos nas respostas para podermos nos basear ainda mais dentro da pesquisa, sendo um fator necessário do trabalho. Severino afirma que:

QUESTIONÁRIO Conjunto de questões, sistematicamente articuladas, que se destinam a levantar informações escritas por parte dos sujeitos pesquisados, com vistas a conhecer a opinião dos mesmos sobre os assuntos em estudo. As questões devem ser pertinentes ao objeto e claramente formuladas, de modo a serem bem compreendidas pelos sujeitos. As questões devem ser objetivas, de modo a suscitar respostas igualmente objetivas, evitando provocar dúvidas, ambiguidades e respostas lacônicas. Podem ser questões fechadas ou questões abertas. No primeiro caso, as respostas serão escolhidas dentre as opções predefinidas pelo pesquisador; no segundo, o sujeito pode elaborar as respostas, com suas próprias palavras, a partir de sua elaboração pessoal. De modo geral, o questionário deve ser previamente testado (pré-teste), mediante sua aplicação a um grupo pequeno, antes de sua aplicação ao conjunto dos sujeitos a que se destina, o que permite ao pesquisador avaliar e, se for o caso, revisá-lo e ajustá-lo. (SEVERINO, 2014, p.109).

Com o questionário podemos desenvolver da melhor forma o trabalho, com intuito de realmente identificar a dor do paciente e solucionar aquele desconforto que foi sinalizado pelo paciente. Usando ferramentas possíveis e plausíveis para que possa ser feito de uma maneira lúcida, clara e possível essa solução. Preservando sempre o usuário para que ele possa da melhor forma ser atendido sua demanda, demonstrando confiança, segurança e integridade dos dados. Mesmo com as exposições de informações e dados atualmente, podemos coletar essas informações preservando os dados do usuário, usando as normas técnicas e de segurança da informação.

O trabalho será levantado com total de informações necessários para podermos desenvolver o projeto imposto, mostrando todos os pontos e focando principalmente nos cruciais para que, fique de forma contundente e simples para que todos possam ter ciência das informações, formulações e desenvolvimento do trabalho, sempre visando o usuário final que ele será nosso grande alvo deste projeto.

# FUNDAMENTAÇÃO TEÓRICA

Mas o que ela quer que escreve mann? Ataaa é oq já ia fazer msm ta safe

## Engenharia de Software

No mundo atual tudo está conectado a software, desde uma pequena criança que tem um contato com uma TV inteligente, ou um celular, até uma empresa de grande porte que precisa de todas os processos, informações e automatização de sua produção. Implicando em diversas integrações com a engenharia de máquinas, veículos, robôs.

Segundo Valente podemos definir Engenharia de Software:

Portanto, devido a sua relevância no nosso mundo, não é surpresa que exista uma área da Computação destinada a investigar os desafios e propor soluções que permitam desenvolver sistemas de software — principalmente aqueles mais complexos e de maior tamanho — de forma produtiva e com qualidade. Essa área é chamada de **Engenharia de Software**. Engenharia de Software trata da aplicação de abordagens sistemáticas, disciplinadas e quantificáveis para desenvolver, operar, manter e evoluir software. Ou seja, Engenharia de Software é a área da Computação que se preocupa em propor e aplicar princípios de engenharia na construção de software. (VALENTE, 2020, p.2).

Com base no que Valente (2020) escreveu: Através do IEEE Computer Society (uma sociedade científica internacional) temos 12 áreas de Conhecimento em Engenharia de Software. São elas:

* + Engenharia de Requisitos
  + Projeto de Software
  + Construção de Software
  + Testes de Software
  + Manutenção de Software
  + Gerência de Configuração
  + Gerência de Projetos
  + Processos de Software
  + Modelos de Software
  + Qualidade de Software
  + Prática Profissional
  + Aspectos Econômicos

Neste tópico iremos apenas dar um esboço das três primeiras áreas de engenharia de software que são: Engenharia de Requisitos, Projeto de Software e Construção de Software, pois é o pilar para a base do conhecimento da engenharia de software, claro que isso não significa que os demais são menos importantes, apenas que iremos detalhar alguns outros em tópicos específicos e outros que não citaremos pois não será necessário ao desenvolvimento do trabalho.

Engenharia de Requisitos podem ser definidos **funcionais** e **não-funcionais**, como definiu Valente (2020) sobre os **Requisitos Funcionais** “definem *o que* um sistema deve fazer; isto é, quais funcionalidades ou serviços ele deve implementar.” (VALENTE, 2020, p. 7).

Já os **Requisitos Não-Funcionais** segundo Valente (2020) “definem *como* um sistema deve operar, sob quais restrições e com qual qualidade de serviço. Exemplos: desempenho, disponibilidade, tolerância a falhas, segurança, privacidade, capacidade, manutenibilidade e usabilidade.” (VALENTE, 2020, p.7).

Projeto de Software definido por Valente (2020) “são definidas suas principais unidades de código, porém apenas no nível de interfaces, incluindo **interfaces providas** e **interfaces requeridas**.” (VALENTE, 2020, p.7).

Significa que segundo Valente (2020) As interfaces providas são os serviços que uma unidade de código expõe para uso pelo resto do sistema. As interfaces requeridas são aquelas interfaces das quais a funcionalidade de uma unidade de código depende.

Sobre Construção de Software sendo importante a colocação que Valente (2020) descreveu:

Codificação do sistema. Nesse momento, existem diversas decisões que precisam ser tomadas, como, por exemplo: definir os algoritmos e estruturas de dados que serão usados, definir os frameworks e bibliotecas de terceiros que serão usados; definir técnicas para tratamento de exceções; definir padrões de nomes, leiaute e documentação de código e, por último, mas não menos importante, definir as ferramentas que serão usadas no desenvolvimento. (VALENTE, 2020, p.8).

* + 1. **Metodologia Ágil**

Metodologia Ágil faz parte do Processo da Engenharia de Software, o processo da engenharia segue procedimentos bem definidos, isso deve ser seguido de conforme os padrões impostos pelos métodos as quais irão optar, que são diversos os métodos que podem ser escolhidos. Se levarmos em conta as complexidades do desenvolvimento de software que hoje são levantados, a metodologia traz toda uma bagagem para poder nortear, definir os processos e seguir as etapas levantadas. Como bem definiu Valente (2020) sobre o processo de software:

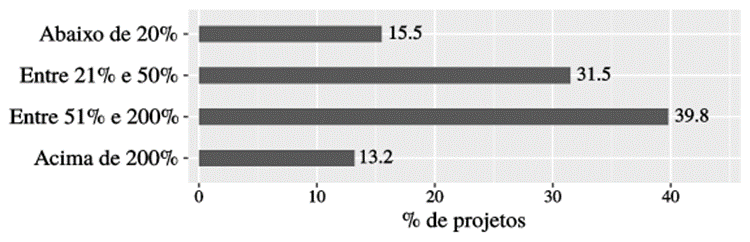
E essas equipes, para produzir software com qualidade e produtividade, precisam de um ordenamento, mesmo que mínimo. Por isso, empresas dão tanto valor a processos de software. Eles são o instrumento de que as empresas dispõem para coordenar, motivar, organizar e avaliar o trabalho de seus desenvolvedores, de forma a garantir que eles trabalhem com produtividade e produzam sistemas alinhados com os objetivos da organização. (VALENTE, 2020, p.29).

Para abordarmos a Metodologia Ágil precisamos citar o Manifesto Ágil pois são valores e princípios do Manifesto Ágil que se deu a Metodologia Ágil. Se olharmos na história conforme citou Valente (2020) na década de 70 o desenvolvimento de software era sequencial, começando de uma fase bem específica de requisitos até as fases de implementação, testes e configuração do sistema, que no caso são as fases finais. Mas, se compararmos a Engenharia de software, com as outras engenharias, ela se diferencia das demais, pois devido aos prazos e cronogramas estabelecidos previamente, não eram seguidos, conforme citado por Valente (2020) sobre:

Em 1994, um relatório produzido pela empresa de consultoria *Standish Group* revelou informações mais detalhadas sobre os projetos de software da época. Por exemplo, o relatório, que ficou conhecido pelo sugestivo nome de CHAOS Report (link), mostrou que mais de 55% dos projetos estouravam os prazos planejados entre 51% e 200%; pelo menos 12% estouravam os prazos acima de 200%, conforme mostra o próximo gráfico. (VALENTE, 2020, p.31).

Conforme Valente (2020) enfatizou os resultados de custos não foram mais animadores: quase 40% dos projetos estavam entre 51% e 200% acima do orçamento, conforme mostra o gráfico a seguir:

Figura - CHAOS Report (1994)



FONTE: VALENTE, MAR (2020, p.33)

Com base nesses dados entende-se que, o processo de engenharia de software ele se difere dos demais processos naturais da engenharia. Se formos levar em consideração com frequência os requisitos do software são alterados, mais do que com quaisquer outros requisitos de outros tipos de engenharia, como de veículos e outros equipamentos. Outro ponto importante é que geralmente o cliente ele idealiza um projeto de software, mas sem ter noção daquilo que deseja.

Segundo Valente (2020) decidiram lançar o que se chama de Manifesto Ágil, um novo conceito para engenharia de software, com documentação. Uma definição que Valente usou é:

A característica principal de processos ágeis é a adoção de ciclos curtos e iterativos de desenvolvimento, por meio dos quais um sistema é implementado de forma gradativa; começando por aquilo que é mais urgente para o cliente. De início, implementa-se uma primeira versão do sistema, com as funcionalidades que segundo o cliente são para ontem, isto é, possuem prioridade máxima. Em seguida, essa versão é validada pelo cliente. Se ela for aprovada, um novo ciclo — ou iteração — inicia-se, com mais algumas funcionalidades, também priorizadas pelos clientes. Normalmente, esses ciclos são curtos, com duração de um mês, talvez até um pouco menos. (VALENTE, 2020, p.34).

Através disso o sistema vai tomando forma e sendo adicionado e alterado no momento de sua construção. O projeto é validado depois que o cliente sinaliza que todos os requisitos foram entregues.

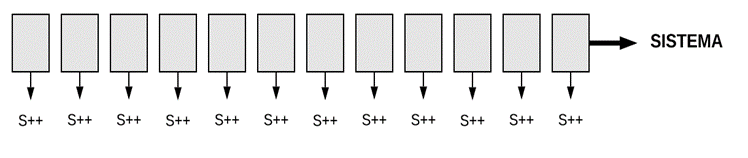
Comparação de metodologia convencional de *Waterfall*, usado na grande parte de outras engenharias e Metodologia Ágil.

Figura - Processo *Waterfall*



FONTE: VALENTE, MARCO (2020, p.33)

Figura - Processo Ágil



FONTE: VALENTE, MARCO (2020, p.33)

Método *Waterfall* como a figura 5 mostra fica pronto apenas no final o projeto, já o Método Ágil conforme a figura 5 o S++ como fosse o incremento no sistema, que já é validado e testado pelos usuários por cada etapa do projeto, ou seja, ele é liberado por etapa do projeto, diferentemente do *Waterfall* que somente depois de todas as etapas implementada que o usuário poderá usar o software.

Conforme Valente (2020) citou o objetivo final é deixar um processo de entrega do sistema cada vez mais funcional para o cliente final.

Metodologia ágil ele não é focada em tanta documentação, apenas o necessário, também dá menor ênfase na detalhes do planejamento, pois muitas vezes o cliente e o desenvolvedor do software ainda não tem certeza do projeto que irá implementar, a parte de design é desenvolvida com andamento do projeto, dá se a possibilidade de desenvolver e times pequenos, não precisa de uma quantidade de profissionais para interagirem no desenvolvimento do projeto, tendo também programação em pares, testes automatizados e integração contínua, escreve VALENTE (2020).

Existem três métodos ágeis, que tem sua maior expressão no meio da programação: *Extreme Programming* (XP) proposto por Kent Beck em 1999; Scrum proposto por Jeffrey Sutherland e Ken Schwaber em 1995; Kanban sua origem no controle de produção nas fábricas da Toyota, década de 50, conforme citado por VALENTE (2020).

* + 1. **Testes**

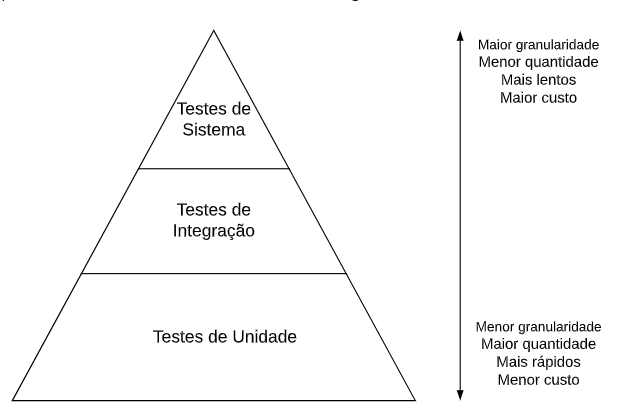
Com a evolução na área de engenharia de software, as abordagens de seus princípios e etapas que atuam diretamente no desenvolvimento, foram profundamente reformuladas, consequências de surgimento de variáveis técnicas e metodologias modernas, como – as ágeis por exemplo.

Atuar no desenvolvimento de software, é uma das inovações humanas mais desafiadoras, portanto é compreensível falhas e variáveis erros e inconsistências nos sistemas. Para que tais resultados errôneos não chegassem a atingir os usuários de formas a qual causem prejuízos maiores, é fundamental a introdução de atividades de testes em projetos de desenvolvimento de software. Segundo Valente (2020), de fato, teste é uma das práticas de programação mais valorizadas hoje em dia, em qualquer tipo de software. É também uma das práticas que sofreram mais transformações nos anos recentes.

Assim, como todo o contexto os métodos ágeis, proporcionou mudanças significativas para a prática de testes de software, visto que nos primórdios, o objetivo de tal testes eram apenas detectar bugs. Quando o desenvolvimento era em cascata, os testes ocorriam em uma fase separada, após as fases de levantamento de requisitos, análise, projeto e codificação. Além disso, existia uma equipe separada de testes, responsável por verificar se a implementação atendia aos requisitos do sistema. (VALENTE, 2020, seção.8.2).

Os testes inseridos no contexto dos métodos ágeis, teve sua identidade e objetivo reformulado, transformando essa etapa do processo, essencial, principalmente quando parte do princípio de desenvolvimento com qualidade. Deixando de apenas detectar bugs, e passando a assegurar um processo de desenvolvimento de software com qualidade, seguindo como pilar a pirâmide de testes.

Figura - Pirâmide de testes



Fonte: Valente (2020)

Como podemos observar na figura acima, os testes estão divididos em três grupos. Na base da pirâmide, identifica-se os Testes de unidade ou testes unitários, onde se classificam como testes automatizados que são utilizados para testar pequenas partes de um código. No próximo nível da pirâmide, observa-se os testes de integração ou testes integrados, onde são testes mais complexos para testar funcionalidades completa de um software. E no último nível, no topo da pirâmide, encontra-se os testes de interface (*end-to-end*), cujo se assemelham-se a experiencia real de um usuário, testando sua usabilidade por completo. Por serem mais fiel, são mais caros e lentos.

Com base nas metodologias ágeis, essas transformações tornaram testes uma das práticas de programação mais valorizadas em desenvolvimento moderno de software, segundo Valente (2020). E nesse contexto, grandes partes dos testes, passou a ser automatizados, os testes se tornaram específicos possibilitando testar classe por classe do software, os testes andam lado a lado no processo de desenvolvimento e não é mais somente para detectar bugs ao final de todo o processo. Não existe maiores equipes de testes, e em alguns casos a própria equipe efetua alguns tipos de testes enquanto desenvolve.

O principal benefício do novo modelo de testes, é encontrar os bugs ainda na fase de desenvolvimento, para evitar regressões no código e prejuízos de custos com possíveis correções. Na prática de desenvolvimento propostas originalmente por métodos ágeis, segundo o Valente (2020), os testes de unidades é provavelmente a que alcançou o maior impacto e que é mais largamente usada. Hoje, os mais diversos sistemas de software, de empresas dos mais diferentes tamanhos, são desenvolvidos com o apoio de testes de unidade.

A seguir, temos dois casos de grandes empresas de software: Google e Facebook. Os trechos foram extraídos de artigos que documentam a metodologia de desenvolvimento de ambas as marcas.

“Testes de unidade são fortemente encorajados e amplamente praticados no Google. Todo código de produção deve ter testes de unidade e nossa ferramenta de revisão de código automaticamente destaca código submetido sem os correspondentes testes. Os revisores de código normalmente exigem que qualquer mudança que adiciona novas funcionalidades deve também adicionar os respectivos testes.”

“No *Facebook*, engenheiros são responsáveis pelos testes de unidade de qualquer código novo que eles desenvolvam. Além disso, esse código deve passar por testes de regressão, os quais são executados automaticamente, como parte dos processos de *commit* e *push*.”

Com base na importância de garantir essa etapa de testes, aplicada de forma adequada, existem ferramentas que medem o cálculo de cobertura de testes. Que por sua vez, cobertura de testes, segundo Valente (2020), é uma métrica que ajuda a definir o número de testes que precisamos escrever para um programa. Ela mede o percentual de comandos de um programa que são cobertos por testes, isto é: cobertura de testes = (número de comandos executados pelos testes) / (total de comandos do programa).

Sendo assim, as coberturas de testes, não existe números mágicos e absoluto, varia de projeto para projeto, de acordo com os requisitos, complexibilidade e sua criticidade. Levando-se em consideração as informações, geralmente recomenda-se os times valorizarem uma cobertura de teste acima de 90%, e valores abaixo de 50% tendem ser a preocupantes. Como podemos conferir no trecho a seguir: times que valorizam a escrita de testes costumam atingir facilmente valores de cobertura próximos de 70%. Por outro lado, valores abaixo de 50% tendem a ser preocupantes. Por fim, mesmo quando se usa TDD, a cobertura de testes costuma não chegar a 100%, embora normalmente fique acima de 90%. (VALENTE, 2020, seção.8.3).

Para assegurar que os desenvolvedores escrevam os testes, a metodologia ágil (XP), criou a princípio o desenvolvimento dirigido por testes ou o TDD, onde a ideia é testar antes de desenvolver. Onde o intuito é fazer falhar. E assim, segundo o autor Vicente (2020), no fluxo de trabalho defendido por TDD, o próximo passo consiste em escrever o código que faz esse teste passar, mesmo que seja um código trivial. Em seguida, esse primeiro código deve ser finalizado e refinado. Por fim, se necessário, ele deve ser refatorado, para melhorar seu projeto, legibilidade, manutenibilidade, para seguir princípios e padrões de projeto entre outros.

O TDD, além de assegurar que os desenvolvedores realizem os testes, ele assegura um código com mais testabilidade e influencia diretamente na melhoria de design de interface, de um software.

* + 1. **Casos de Uso – UML**

Podemos citar que, [...] de uma perspectiva mais ampla, um modelo pode ser visto como uma representação idealizada de um sistema a ser construído (BEZERRA, 2014, p.12).

Portanto podemos dizer que modelos são a simplificação da realidade, sendo este responsável por expressar as características, atributos, funcionamento e comportamentos do sistema, de modo a facilitar o entendimento para todos os participantes de um projeto de desenvolvimento de sistema.

Para ajudar no entendimento e análise do sistema, é criado modelos que são mais detalhados que os requisitos e menos complexos que o código fonte. Portanto, entre a etapa de requisitos e escrita do código fonte entra a etapa de modelagem de dados.

Uma das técnicas mais utilizadas na etapa de modelagem de dados de um sistema é a elaboração de Casos de Uso. Esta técnica tem como objetivo principal modelar o comportamento, descrever funções em alto nível e auxiliar a expressar os requisitos e características do software de um sistema a ser desenvolvido.

Citando COCKBURN (2005):

Um caso de uso captura um contrato que descreve o comportamento do sistema sob várias condições à medida que ele responde a requisições de um de seus usuários. (COCKBURN, 2005)

Blz man, quando terminar da um toque

Sim, pra ficar bom mesmo vou ter que refazer saporra

combinado

**Linguagem de Modelagem Unificada (UML)**

Originada pela contribuição de GradyBooch, James Rumbaugh e Ivar Jacobson e muitos outros contribuintes, esta notação foi formada pelo aproveitamento das melhores características das notações preexistentes, principalmente das técnicas que se haviam proposto anteriormente e uma união de diversas notações preexistentes, com alguns elementos removidos e outros adicionados com o objetivo de torná-la mais expressiva.

Segundo BEZERRA (2014), a definição de UML é:

A UML é uma linguagem visual para modelar sistemas orientados a objetos. Isso quer dizer que a UML é uma linguagem que define elementos gráficos (visuais) que podem ser utilizados na modelagem de sistemas. Esses elementos permitem representar os conceitos do paradigma da orientação a objetos. Por meio dos elementos gráficos definidos nesta linguagem pode-se construir diagramas que representam diversas perspectivas de um sistema.

A UML é uma linguagem de notação (um jeito de escrever, ilustrar, comunicar) utilizada como ferramenta na modelagem de dados e com o objetivo de facilitar a compreensão de todas as características e rotinas de um sistema.

Esta linguagem é expressa através de diagramas e diagrama é composto por elementos dispostos em formas gráficas que possuem relação entre si, sendo estes diagramas divididos em dois grandes grupos: diagramas estruturais e diagramas comportamentais.

Segundo VALENTE (2020), diagramas estruturais ou estáticos são responsáveis pela modelagem da estrutura e organização de um sistema, contemplando todas as informações sobre as classes, atributos, métodos, relacionamentos, pacotes, contextos etc.

Há dois importantes diagramas estruturais na UML, o Diagrama de Classes e o Diagrama de Pacotes, e cada um destes é escolhido para uso de acordo com o paradigma de programação do sistema a ser implementado.

Ainda segundo VALENTE (2020), diagramas comportamentais ou dinâmicos são responsáveis por modelar eventos e rotinas que devem ocorrer durante a execução de um sistema, ou seja, são utilizados para modelar uma sequência de chamadas, atividades ou métodos.

Para diagramas comportamentais há também dois importantes diagramas na UML, o Diagrama de Sequência e o Diagrama de Atividades.

Trataremos aqui sobre um diagrama estrutural e um diagrama comportamental, sendo estes o Diagrama de Classes e o Diagrama de Atividades respectivamente.

Diagrama de Classes

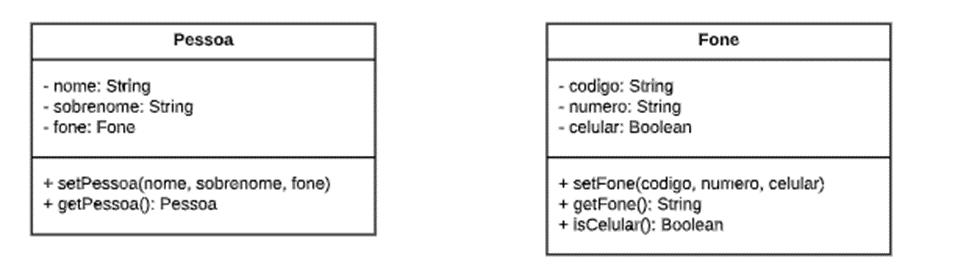
Sabendo que classes são abstrações de objetos da vida real, as quais agrupam atributos e rotinas relacionados ao seu contexto, os diagramas de classe têm como objetivo principal oferecer uma representação gráfica do conjunto destas classes abstraídas, provendo informações sobre os atributos, rotinas e sobre os relacionamentos que existem entre as classes modeladas (VALENTE, 2020).

O diagrama de classes representa graficamente como será a estrutura do sistema em níveis básicos ou detalhado e como cada um dos componentes da sua estrutura estarão relacionados e pode ser utilizando tanto para o contexto de Engenharia Avante e Engenharia Reversa (ambos citados anteriormente).

Para formar um Diagrama de Classes utiliza-se principalmente da representação gráfica de Classe e suas respectivas Associações, Heranças e Dependências. Estes três últimos são utilizados para [...] mostrar visualmente os relacionamentos que existem entre as classes de um sistema [...] (VALENTE, 2020) aos quais são representados por linhas e setas.

**Classes**

Como sabemos, Classe é uma abstração de objetos da vida real, as quais agrupam atributos e rotinas. E tendo isso e vista, graficamente na UML uma classe possui três compartimentos, sendo uma para definição do nome da classe, outra para definição dos atributos e por fim outra para descrição das operações ou métodos (rotinas).

Figura - Exemplos de Classes

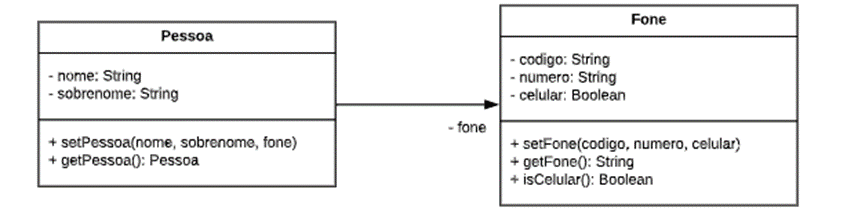
Fonte: Valente (2020)

**Associação**

É um tipo de relacionamento usado entre classes. Aplicável a classes que são independentes (vivem sem dependência umas das outras), mas que em algum momento no ciclo de vida do sistema (enquanto ele está em execução) podem ter alguma relação conceitual.

É representada graficamente com uma seta que liga a classe associada a outra classe sendo possível também informar o nome do atributo responsável pela associação.

Figura - Exemplo de Associação entre Classes



Fonte: Valente (2020)

Utilizando notação de UML e seguindo orientações de uso, podemos também aplicar informações de multiplicidade neste relacionamento. Segundo VALENTE (2020):

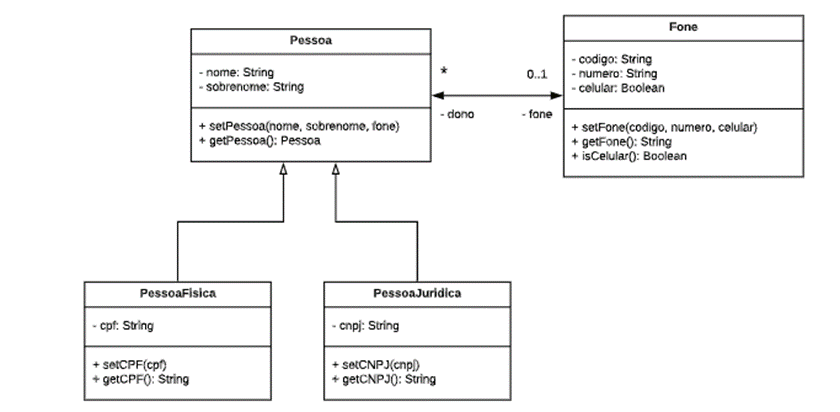
Frequentemente, associações incluem informações de multiplicidade, que indicam quantos objetos podem estar associados ao atributo responsável pela associação. As informações de multiplicidade mais comuns são as seguintes: 1 (exatamente um objeto), 0.1 (zero ou um objeto) e \* (zero ou mais objetos).

**Herança**

É um tipo de relacionamento onde a classe generalizada (onde a “ponta da seta” do conector fica) fornece recursos para a classe especializada (herdeira). Sendo assim, tudo que a classe mãe (generalizada) tem, a filha (especializada) terá.

Em diagramas de classes, as relações de herança são representadas graficamente por meio de setas com a extremidade não preenchida.

Figura - Exemplo de Herança entre Classes



Fonte: Valente (2020)

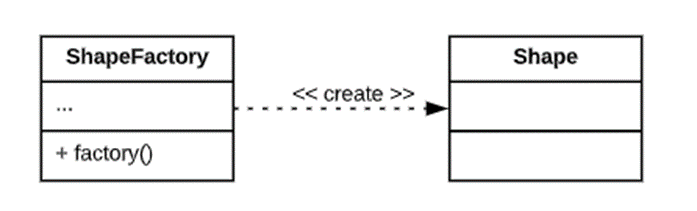
**Dependência**

Na UML, um relacionamento de dependência é um relacionamento no qual um elemento usa ou depende de outro elemento. Um relacionamento de dependência também pode ser utilizado para representar precedência, em que um elemento de modelo deve preceder outro.

Citando VALENTE (2020), podemos dizer que uma dependência é considerada uma modalidade menos forte de relacionamento entre classes do que relacionamentos que ocorrem por meio de associação e herança.

Além da representação gráfica conter uma seta igualmente a associação e herança, embora esta tenha sua linha tracejada, é possível representar o tipo de dependência usando palavras como *create* ou *call* para indicar que a classe origem instancia objetos da classe de destino da dependência. Essas palavras são escritas entre sinais de menor (<<) e maior (>>).

Figura 10 - Exemplo de Dependência entre Classes



Fonte: Valente (2020)

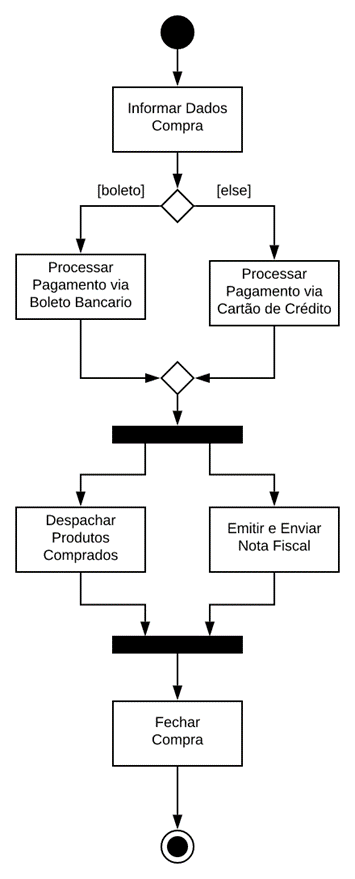
No diagrama da Figura XX acima, por exemplo, fica claro o tipo de dependência que *ShapeFactory* estabelece com a classe *Shape*.

**Diagrama de Atividades**

De acordo com VALENTE (2020), Diagramas de atividades são usados para representar, em alto nível, um processo ou fluxo de execução.

A elaboração de diagramas de atividade oferece uma série de benefícios para os analistas e desenvolvedores do sistema, de forma a demonstrar a lógica de um algoritmo, descrever as etapas realizadas em um caso de uso, ilustrar um processo de negócio ou fluxo de trabalho entre usuários e o sistema, simplificar e melhorar qualquer processo ao esclarecer casos de uso complicados e também modelar elementos de arquitetura de software, como método, função e operação.

Figura - Exemplo de Diagrama de Atividades

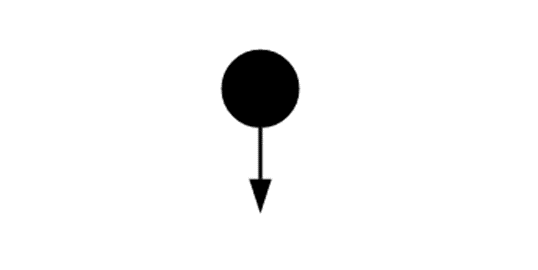


Fonte: Valente (2020)

Um diagrama de atividades graficamente conta com componentes ou elementos que definem uma rotina composta por ações, decisões e sequências dinâmicas. Os componentes mais comuns são Nodo Inicial, Ações, Decisões, Merges, *Forks*, *Joins* e Nodo Final.

* Nodo Inicial: É uma notação gráfica com a responsabilidade única de representar onde se inicia o processo, função ou operação e possui apenas fluxo de saída.

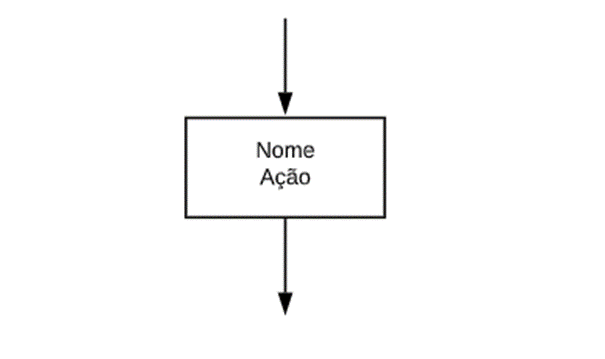
Figura - Exemplo de Nodo Inicial



Fonte: Valente (2020)

* Ações: São representadas por retângulos com a descrição da atividade ou ação dentro tendo fluxo de entrada e fluxo de saída além da possibilidade de ser executada mais de uma vez no mesmo diagrama.

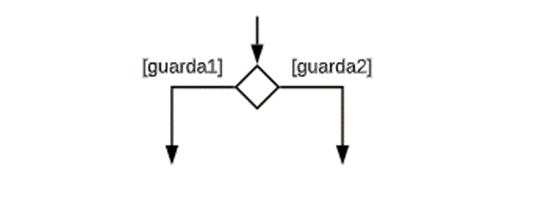
Figura - Exemplo de Ações



Fonte: Valente (2020)

* Decisões: Sua representação gráfica se dá por um losango que possui um único fluxo de entrada e dois ou mais fluxos de saída. Como o próprio nome diz, é um ponto de decisão dentro do método a ser executado.

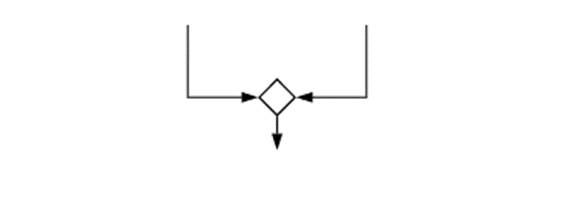
Figura - Exemplo de Decisões



Fonte: Valente (2020)

* *Merges*: Sua representação gráfica se dá por um losango semelhante ao de Decisões, mas possui dois ou mais fluxos de entrada e um único fluxo de saída, sendo este utilizando para junção dos fluxos de nós de decisão.

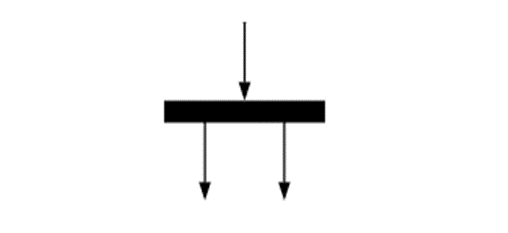
Figura - Exemplo de *Merges*



Fonte: Valente (2020)

* *Forks*: Possuem um único fluxo de entrada e um ou mais fluxos de saída. Atuam como multiplicadores de processos, recebendo um único fluxo e criando e repassando copias destes fluxos adiante. Tendo com resultado a existência de múltiplos processos em execução de forma paralela.

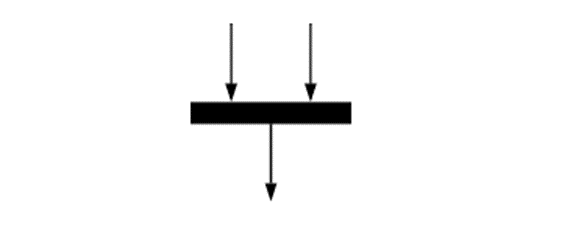
Figura - Exemplo de *Forks*



Fonte: Valente (2020)

* *Joins*: Possuem vários fluxos de entrada, mas um único fluxo de saída. Atuam dissolvedor de processos, transformando vários fluxos de execução em um único fluxo.

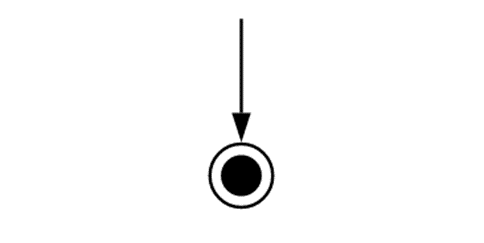
Figura - Exemplo de *Joins*



Fonte: Valente (2020)

* Nodo Final: Pode possuir mais de um fluxo de entrada; mas não possui fluxos de saída e é utilizado para fechar o fluxo do diagrama.

Figura - Exemplo de Nodo Final



Fonte: Valente (2020)

* + 1. **Banco de Dados**

Banco de dados é um sistema capaz de armazenar e gerenciar um conjunto de dados, ou seja, são informação que mantemos registradas num computador para a utilização no dia a dia.

Um banco de dados é um software que armazena um conjunto de dados inter-relacionados, representando informações sobre um domínio específico, ou seja, sempre que for possível agrupar informações que se relacionam e tratam de um mesmo assunto, posso dizer que tenho um banco de dados (KORTH, 1999).

Existem vários tipos de bancos de dados, no entanto, os principais são: banco de dados relacional e banco de dados orientado a objetos.

Modelos de Bancos de Dados:

Segundo HEUSER (2009), um modelo de (banco de) dados é uma descrição dos tipos de informações que

estão armazenadas em um banco de dados.

**Banco de dados relacional**

Um banco de dados do tipo relacional é um modelo que usa relações para representar e armazenar os dados tendo em vista que relação é um conceito da álgebra relacional que é utilizado para modelar banco de dados relacionais.

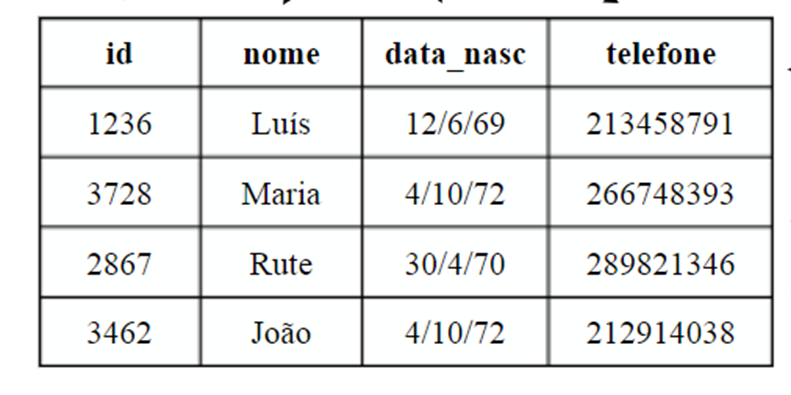
O modelo relacional apareceu devido as necessidades de aumentar a independência de dados nos sistemas gerenciadores de banco de dados, prover um conjunto de funções apoiadas em álgebra relacional para armazenamento e recuperação de dados além de outros.

Este modelo revelou-se ser o mais flexível e adequado ao solucionar os vários problemas que se tinham no nível da concepção e implementação da base de dados de um sistema. Sua estrutura é formada por tabelas que é constituída por um ou mais atributos (campos) que traduzem o tipo de dado que será armazenado onde cada linha da tabela é um registro.

Implementando estruturas de dados organizadas em relações (tabelas), é necessário se ter em mente que para trabalhar com essas tabelas, algumas restrições precisaram ser consideradas a fim de evitar aspectos indesejáveis, como: repetição de informação, incapacidade de representar parte da informação ou até perda de informação. Essas restrições comumente são denominadas como: integridade referencial, chaves e integridade de junções de relações.

A figura 22 abaixo nos traz exemplos de tabelas que seguem o modelo relacional.

Figura - Tabela Exemplo Modelo Relacional



Fonte: SANTOS, Luis (Tabela Exemplo Modelo Relacional)

**Banco de dados orientado a objetos**

Esse modelo usa a mesma ideia de objetos da programação orientada a objetos para representar e armazenar os dados. Esse tipo de banco de dados é muito empregado em aplicações que demandam georreferenciamento.

Seu surgimento de deu em função dos limites de armazenamento e representação semântica impostas pelo modelo relacional. A possibilidade de criar tipos de dados necessários é uma característica de linguagens de programação orientadas a objetos e alguns sistemas necessitavam armazenar representações das estruturas de dados que utilizariam no armazenamento permanente.

**Sistemas de gerenciamento de banco de dados (SGBD)**

Um sistema de gerenciamento de banco de dados é um conjunto de programas de software que permite aos usuários criar, editar, atualizar, armazenar e recuperar dados em tabelas de banco de dados. Dados em um banco de dados podem ser acrescentados, apagados, alterados, classificados usando um SGBD além da possibilidade de estruturar todo o armazenamento de um sistema.

Em outras palavras, consiste em um sistema que possui recursos capazes de manipular as informações de um banco de dados e interagir com o usuário. Alguns exemplos atuais de SGBD são: Oracle, SQL Server, DB2, PostgreSQL, MySQL, dentre outros.

De acordo com ELMASRI e NAVATHEU (2011):

Um sistema gerenciador de banco de dados (SGBD) é uma coleção de programas que permite aos usuários criar e manter um banco de dados. O SGBD é, portanto, um sistema de software de propósito geral que facilita os processos de definição, construção, manipulação e compartilhamento de bancos de dados entre vários usuários e aplicações. A definição de um banco de dados implica especifica os tipos de dados, as estruturas e as restrições para os dados a serem armazenados em um banco de dados. (ELMASRI e NAVATHEU, 2011).

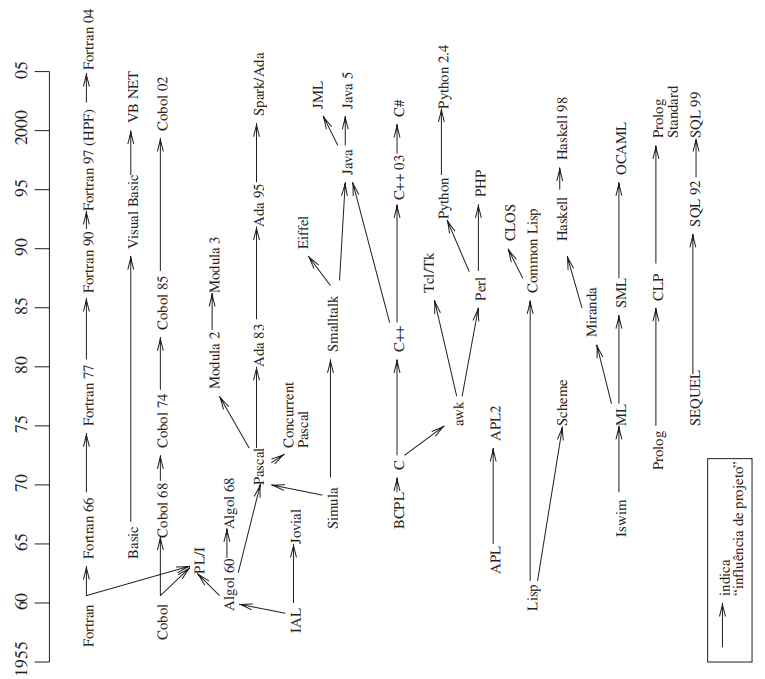
## Linguagem de Programação

A linguagem de programação é a ferramenta fundamental para a comunicação do homem com a máquina, de acordo com Tucker e Noonan (2010, p.6), “As primeiras linguagens de programação foram as linguagens de máquinas.” e a linguagem *assembly* começando esse novo passo na década de 1940. A partir desse início, diversas linguagens de programação foram desenvolvidas, muitas com um tempo de vida muito curto, assim como a sua utilidade, já outras com uma qualidade ímpar que inspiram novas linguagens e duram até os dias de hoje.

Em 1950, deu início a era das “linguagens de ordem mais alta” (abreviada no inglês como HOLs). A HOL era o termo dado para as linguagens de máquinas que são independentes do *assembly*. Linguagens como Cobol, Fortran, Algol e Lisp foram as primeiras a ganhar o termo, mas somente Cobol e Fortran se mantiveram funcionais até o fim de 1950.

A figura 3.2.1. mostra um pouco do histórico das linguagens de programação, por mais que esteja incompleta, podemos ver como as linguagens mais populares surgiram. Cada seta na figura 3.2.1. indica a influência da linguagem mais antiga para as atuais.

Figura - História das linguagens de programação



Fonte: Linguagens de Programação: Princípios e Paradigmas

* + 1. **Desenvolvimento Web, Front-end e Back-end**

A programação WEB surgiu nos anos 90, por Tim Berners-Lee, cientista da computação, também responsável pela criação do “WWW” ou “*World Wide Web*” (MONTEIRO, 2001, p. 29), utilizando apenas a linguagem hipertexto HTML (do inglês *Hypertext Markup Language*), originando os primeiros sites webs da internet, gerando a Tim, o apelido de “pai da internet” (TORRES, 2018, p. 99).

Com o passar do tempo, a programação web evoluiu e seus conceitos foram se transformando e evoluindo. Hoje em dia, para a programação de um site com funções simples ou avançadas, só o HTML não é o suficiente, sendo necessário o *Front-end* e o *Back-end* para realização de tarefas diárias, se mostrando peças fundamentais para a programação.

De acordo com Mario Souto (2022), o *front-end* é a parte visual do desenvolvimento para o programa, utilizado para sites, aplicativos de celular e aplicativos para computadores. O *front-end* tem como objetivo facilitar a comunicação com o usuário final do programa. Usando uma mistura de HTML, CSS (do inglês *Cascading Style Sheet*) e Javascript, é possível a criação de site com funcionalidades simples.  
 Com o avanço da tecnologia, o *front-end* foi se tornando uma ferramenta de imensa importância para comunicação e visualização de informações em tempo real, através do *front-end* que o usuário interage com o site e consegue usufruir de todas as funções que oferece.

Ainda nessa temática, o *back-end* se mostra uma plataforma pertencente aos bastidores do desenvolvimento de um site ou de uma aplicação, se interliga com o *front-end* de maneira que o *front-end* apresenta uma relação de dependência com o *back-end*, em suma, o *front-end* passa as informações para o *back-end*, que por sua vez armazena essas informações no banco de dados.

Dessa maneira, a parte visual pertence ao *front-end*, mas todas as linhas de comando do desenvolvimento são pertencentes ao *back-end*, algo que o usuário comum não irá visualizar ou ter acesso a essas linhas de código.

Conforme Mario Souto (2022), o *back-end* carrega com si um escopo da: “ideia daquilo que tem por trás de uma aplicação”, ou seja, é a ponte entre o banco de dados com a sua aplicação.

Logo, toda a parte robusta do programa e incompreensível para o usuário comum, pertencem à *back-end*, seja para a criação de um novo usuário ou trazer uma lista extensa de itens ao banco de dados, o *back-end* é uma peça fundamental para qualquer tipo de programa funcional.

O *back-end* em sites surgiu em 1995 com o PHP, cujo objetivo era enviar séries de informações por protocolo HTTP (do inglês *Hypertext Transfer Protocol*) para um servidor (FERREIRA, 2022).

* + 1. **Desenvolvimento Mobile**

Os dispositivos móveis vêm proporcionando uma revolução na forma de comunicação e interação na sociedade. Através do crescente avanço na disponibilização de acesso à Internet, estes aparelhos ganharam destaque no cotidiano de uma boa parcela da população. A universalização de padrões técnicos e a ampliação de conteúdos e acessos à rede internet se intensifica, resultando num aumento significativo de aquisições de smartphones para a conexão. Até a metade de 2014, dos 7 bilhões de habitantes do planeta, pouco mais de 40% tinham acesso à internet (WASHINGTON POST, 2014).

Com essa popularização vieram grandes investimentos na melhoria dos aparelhos e suas funcionalidades. Apoiados por uma indústria de tecnologia composta principalmente por grandes fabricantes como Samsung, Apple, Google, Nokia e Blackberry, eles desenvolveram vários sistemas operacionais focados em seu próprio hardware (ROSARIO, 2015).

Desenvolvimento constante tecnologias de software ao longo dos anos, estimulou indústria em desenvolvimento a buscar alternativas para melhorar qualidade na construção de software, sob diferentes panoramas e a infraestrutura. Este cenário deu origem a novos paradigmas como a Computação Móvel que eles criam novos modelos de negócios.

Conforme Sanchez (2016) assim, os desenvolvedores estão se tornando elementos chave na estrutura organizacional desse modelo emergente, pois criam oportunidades externas àquelas usadas dentro das empresas e criam um ecossistema de software. O novo modelo de negócios proporcionado por esta plataforma incentiva os desenvolvedores de software a usar suas plataformas e métodos de distribuição para oferecer inovação tecnológica.

Uma plataforma computacional consiste em uma coleção de recursos de hardware, como o processador, memória principal, módulos de entrada e saída, *timers*, drives de disco e 14 recursos de software que se comunica com o hardware e provê uma interface com o usuário (STALLINGS, 2012). Já a plataforma móvel segundo (Rosario, 2015):

As plataformas móveis são um conjunto de tecnologias que envolvem sistemas operacionais, linguagens de programação e ferramentas de desenvolvimento. Os aparelhos celulares necessitam de uma plataforma para funcionarem. Estas plataformas são responsáveis por gerenciar os recursos dos aparelhos como o banco de dados, câmera, GPS e áudio. Também são responsáveis pela interação com o usuário, através de aplicativos e recursos multimídia. (ROSARIO, 2015, p.14).

Durante a última década, 2005 em diante dezenas de plataformas móveis surgiram, mas não conseguiram manter-se no mercado por muito tempo. São poucas as plataformas que possuem uma parcela relevante no mercado mundial (VISION MOBILE, 2011).

Atualmente as predominantes no mercado são Android e IOS, sendo o sistema mais utilizado hoje Android com 71,47%, sendo que estrutura do sistema é código aberto, tendo implementação e distribuição sem *royalties*. Em segundo lugar e a iOS com 27,88%, mais importante consideravelmente e fechado, dando segurança e com uma excelente otimização com o hardware detentora a Apple com a marca mais valiosa do mundo atualmente. Outros 0,65% do sistema operacional é variado não tendo expressão no mercado global mobile (ALMENARA, 2022).

A importância do desenvolvimento mobile proporciona ao usuário, uma praticidade e uma melhor imersão nas plataformas, gerando uma melhor otimização do tempo, a ideia de ter tudo na palma da mão nunca fez tanto sentido, atualmente as companhias que querem uma melhor experiência, propagação de sua marca, estão na plataforma online, sempre buscando melhorias na experiência do usuário. Gerando versões atualizadas, corrigindo erros da plataforma, sendo sinalizado ou não pelo usuário, até implementando inteligência artificial para analisar as movimentações dos usuários e analisar os dados gravados das plataformas.

* + 1. **API (Application Programming Interface)**

*Application Programming Interface* (do inglês, interface de programação de aplicativos) é um intermediário de software que permite que duas aplicações se comuniquem entre si.

Segundo BIEHL (2015), uma API pode ter muitas definições diferentes, sendo as mais comuns:

* Uma API pode ser um serviço que fornece através de tecnologias *web*, tais como HTTP, podendo ser utilizadas para o desenvolvimento de aplicações com baixo acoplamento.
* Uma API é simples, clara, e de fácil acesso: é [...] como uma tomada à qual diferentes aplicações podem ligar-se facilmente, tal como diferentes aparelhos são ligados a uma tomada eléctrica [...] BIEHL (2015).

Uma API estabelece uma conexão entre os fornecedores de dados internos da aplicação e consumidores de dados externos desta mesma aplicação.

O funcionamento de uma API se inicia com a aplicação cliente realizando uma chamada da API para solicitar informações. Esta solicitação é processada da aplicação para o servidor da Web através do *Uniform Resource Identifier* (URI) da API e inclui um verbo de solicitação (*verb*), cabeçalhos (*header*) e alguns casos o corpo de informações da solicitação (*body*).

Após processa a solicitação e classificá-la como válida, a API faz a chamada para um programa externo ou para um servidor web e este retorna uma resposta à API com as informações solicitadas.

Por fim a API capta as informações retornadas pelo servidor e encaminha ou transfere para a aplicação solicitante.

Embora a transferência de dados seja diferente dependendo do serviço web utilizado, este processo de solicitação e resposta acontece através de uma API. Enquanto uma interface gráfica de usuário fornece acesso humano aos dados e funcionalidades de uma aplicação, uma API fornece esse acesso a computadores ou outras aplicações.

As APIs oferecem segurança por design porque sua posição como intermediária facilita a abstração da funcionalidade entre dois sistemas – o *endpoint* da API desvincula o aplicativo de consumo da infraestrutura que fornece o serviço. As chamadas de API geralmente incluem credenciais de autorização para reduzir o risco de ataques ao servidor, e um gateway de API pode limitar o acesso para minimizar as ameaças à segurança. Além disso, durante a troca, cabeçalhos HTTP, cookies ou parâmetros de *string* de consulta fornecem camadas adicionais de segurança aos dados.

**Tipos de API:**

Atualmente, em maior número as APIs são hospedadas na Web, ou seja, dados e funcionalidades de uma aplicação são expostos em um navegador Web, dispositivo móvel ou qualquer outro tipo de cliente da Internet. As solicitações para as APIs podem ser realizadas usando qualquer linguagem de programação que provenha mecanismos para comunicação web, ou também podem ser realizadas por páginas web usando HTML (linguagem de marcação). Os principais tipos de API são 4 atualmente:

APIs abertas (também conhecidas como APIs públicas) que podem ser facilmente acessadas utilizando o protocolo HTTP (*Hypertext Transfer Protocol*). Este tipo não requer autenticação para solicitação de informações e atuam apenas no formato de solicitação e resposta.

APIs de parceiros que atuam no mesmo formato das APIs abertas, mas que demandam acesso através de credenciais para autenticação. É normalmente disponibilizada para desenvolvedores utilizarem (consumirem) em aplicações em desenvolvimento, porém este devem solicitar credenciais para realizar solicitações.

APIs internas já são interfaces de programação de aplicativos que permanecem ocultas de usuários externos, ou seja, são APIs restritas (não públicas). Normalmente são destinadas a uso interno por uma empresa como forma de contribuir para a comunicação interna por diferentes equipes ou programas.

APIs compostas são combinações de várias APIs de serviços ou dados. Estas APIs fornecem a possibilidade de acesso a vários endpoints em uma única solicitação. As APIs compostas são úteis na arquitetura de microsserviços, onde a execução de uma única tarefa pode exigir informações de várias fontes.

**Tipos de protocolos de API:**

Para que os canais de comunicação entre a aplicação e a API consigam manter padrões constantes na hora de oferecer um ou mais serviços, é necessário utilizar procedimentos e convenções padrão, os chamados protocolos de API. Estes protocolos estabelecem regras para a comunicação com APIs, garantindo a estabilidade e a eficiência da troca de solicitações e respostas.

Cada API opera de acordo com um protocolo específico, dependendo das suas necessidades e objetivos. Os quatro protocolos API mais utilizados comumente são os protocolos REST, SOAP, XML-RPC e JSON-RPC.

***REST*:**

Um dos protocolos mais populares para o desenvolvimento de *APIs*, *REST* é uma sigla para *Representational State Transfer* (Transferência de Estado Representacional) onde suas *APIs* e aplicações que utilizam a arquitetura *REST* são denominados serviços *RESTful*.

A arquitetura do protocolo *REST* contempla uma dinâmica sem estado (*stateless*) de modo a separar o *front-end* do *back-end* da *API* oferecendo agilidade, ponto único de falhar, desacoplamento e flexibilidade na implementação.

É possível construir *APIs RESTful* com protocolos *SOAP*, mas os dois padrões geralmente são vistos como especificações concorrentes.

**SOAP:**

*Simple Object Access Protocol* (em português, Protocolo Simples de Acesso a Objetos) é um protocolo API construído com XML, que permite aos usuários enviar e receber dados através de SMTP e HTTP. Além disso, este é um protocolo definido como neutro, ou seja, ele pode operar por diferentes protocolos de comunicação, como os já citados HTTP, SMTP e também TCP e outros. Tendo isso em vista, este protocolo é independente de estilos de programação específicos.

Este protocolo visa intermediar o fluxo de trocas de informações utilizando uma estrutura descentralizada, dando a possibilidade de comunicação entre aplicações ou componentes de software executados em diferentes ambientes ou até escritos em diferentes linguagens.

**XML-RPC:**

O nome da tecnologia RPC vem de Remote Procedure Call, ou Chamada Remota de Procedimento. Este protocolo permite que uma aplicação solicite, remotamente, um procedimento de outra aplicação (normalmente por internet ou intranet).

Apesar de também ser construído em XML, este protocolo oferece um método mais simples de envio de parâmetros e recebimento de resultados em comparação com o SOAP. E por ser simples e relativamente leve, o XML-RPC usa largura de banda mínima para comunicação.

**JSON-RPC:**

Seguindo semelhantemente o protocolo XML-RPC, o JSON-RPC também realiza chamadas de procedimento remoto (RPCs), mas diferentemente este usa JSON em vez do formato XML para transferir dados. Ambos os protocolos são simples. Embora as chamadas possam conter vários parâmetros, elas esperam apenas um resultado.

As APIs oferecem segurança pois sua posição como intermediária facilita a abstração da funcionalidade entre dois sistemas, ou seja, endpoint da API desvincula o aplicativo de consumo da infraestrutura que fornece o serviço. Isso contribui para a não dependência e desacoplamento. As chamadas de API geralmente incluem credenciais de autorização para reduzir o risco de ataques ao servidor, e um gateway de API pode limitar o acesso para minimizar as ameaças à segurança. Além disso, durante a troca, cabeçalhos HTTP, cookies ou parâmetros de string de consulta fornecem camadas adicionais de segurança aos dados.

* + 1. **Integração BFF (Backend for Frontend)**

Devido a evolução das tecnologias que abordam o desenvolvimento frontend o padrão chamado BFF (Backend for frontend) cresceu em popularidade. Soluções como SPA (Single Page Application) ou soluções mobiles, tem exigido novas soluções de backend e com isso esse padrão tem se popularizado.

Em tempos anteriores aos dias de hoje, a maioria das aplicações eram monolíticas e tinham o seu desenvolvimento frontend e backend juntos, ou seja, compartilhavam do mesmo nível da aplicação. Mas com o passar do tempo a arquitetura das aplicações cresceu de forma exponencial surgindo para o backend outras necessidades mais complexas.

E, principalmente, devido a questões envolvendo aumento no número de usuários, escalabilidade e manutenção do código de uma aplicação, este padrão de projeto foi adotado fortemente atualmente.

O BFF é um padrão de projeto (Design Pattern) ou também definido como padrão de arquitetura, que é escrito em linguagem de programação e que tem por objetivo exclusivo atender as necessidades de um frontend (cama de apresentação da aplicação ou interface).

A arquitetura Backend For Frontend consiste em vários back-ends desenvolvidos para atender às necessidades das respectivas estruturas de front-end, seja desktop, navegador web ou aplicativos móveis nativos.

Um dos maiores objetivos do BFF é garantir uma interação perfeita do usuário, independentemente da plataforma em que o aplicativo front-end está sendo executado. Ele também aumenta a eficiência geral dos dispositivos inteligentes, pois as aplicações desenvolvidas seguindo a arquitetura BFF otimizam o uso de recursos enquanto o aplicativo está em uso. No BFF, existem APIs para usos específicos; reduzindo significativamente a área de superfície que precisa ser protegida enquanto aumenta consideravelmente a confiança do usuário.

Em aplicativos monolíticos, as APIs contêm dados desnecessários que às vezes são inúteis para aplicativos de consumo ou, na melhor das hipóteses, uma única instância de aplicativo de front-end.

Para explicar melhor, tomemos como exemplo um caso de transações bancárias. Se o usuário quiser saber sua última transação, um aplicativo front-end executado em um smartwatch precisa apenas do valor da transação da API. Desenvolver uma API genérica nesse caso trará como resultado dados desnecessários em resposta que são inúteis para um smartwatch, mas que podem ser usados para uma aplicação cliente em execução no navegador. Essa API causará sobrecarga nos aplicativos do smartwatch.

Desenvolver um back-end BFF dedicado para o smartwatch ajudaria a alcançar uma solução otimizada, não apenas sem custos indiretos, mas também melhoraria a experiência do usuário.

No modelo de microsserviços com apenas um frontend era preciso fazer várias chamadas em diferentes serviços, o que levava a UI exibindo informações de domínios diferentes e serviços retornando dados que não seriam usados pela tela. Com o padrão BFF (Backend for Frontend) temos serviços mais especializados, obtenção das informações que compõe a tela em menos chamadas e simplicidade das chamadas no frontend.

Os benefícios de implementação deste padrão de arquitetura se estendem até para a questão de segurança da aplicação. Podemos citar como exemplo os casos em que a aplicação necessita que o usuário seja autenticado, tendo diferentes modelos de autenticação (por e-mail, telefone, redes sociais, etc.) disponíveis. Sendo assim, protocolos de segurança podem ser combinados e serem únicos tendo a segurança das APIs consumidas gerenciada pelo BFF.

Quando houver a necessidade de mudanças em APIs de outros domínios e até de outras empresas, estas mudanças podem ser abstraídas ao frontend através apenas do tratamento no BFF, sem envolver o processo de novos deploys para o cliente (loja de aplicativos móveis, aplicação desktop, smartwacth, smart-tv, etc.).

APIs podem ser substituídas com segurança sem que haja impacto no frontend da aplicação.

O BFF está ligado diretamente a arquiteturas que visam o desacoplamento. Sendo o desacoplamento um padrão de programação que consiste em montar um design que deixe a aplicação mais flexível a mudanças. Programar um código desacoplado garante uma maior qualidade e facilita futuras mudanças.

Podemos citar também como benefícios da implementação do BFF:

* Com BFFs diferentes, pode-se escalar isoladamente a infraestrutura de uma aplicação, por exemplo: se o acesso de aplicativo móvel é maior que o da aplicação que é executada num navegador web, apenas a infraestrutura que serve o aplicativo móvel precisa escalar.
* SoC — Separação de Responsabilidade (Separation os Concerns), consegue delegar responsabilidade de negócio para APIs de domínio e responsabilidade de tela para o time de Front.
* Tratamento de erro específico para cada tipo de front. Cada plataforma poderá ter sua própria tratativa de erro ou exceções.
* Manutenção de APIs não impactam no frontend da aplicação. Podemos realizar manutenções nas APIs da aplicação e até substituição sem que ocorra implicações no nível de interface do usuário.

É muito comum atualmente que, por exemplo, uma mesma aplicação ter uma interface web, outra móvel e outra responsiva. Neste contexto, entendo que seja bastante tentador projetar uma única API de back-end para todas as interfaces, que seja reutilizável.

Os BFFs resolvem uma preocupação premente de desenvolvimento móvel quando se utilizam microserviços. Além disso, proporcionam uma alternativa convincente ao backend API de uso geral, e muitas equipas fazem uso deles para outros fins que não apenas o desenvolvimento móvel. O simples acto de limitar o número de consumidores que apoiam torna-os muito mais fáceis de trabalhar e de mudar, e ajuda as equipas que desenvolvem aplicações voltadas para o cliente a manterem uma maior autonomia.

* 1. **Git**

Atrelado a engenharia de software, o capítulo atual descreve sobre a ferramenta Git, que é um sistema de controle de versões mais usado no mundo de desenvolvimento na atualidade.

O Git, ele funciona como uma rede social para os desenvolvedores. Partindo dessa ideia, a ferramenta se comporta como uma grande comunidade de compartilhamento de projetos e conhecimento do mundo de desenvolvimento de software.

A ferramenta surgiu, inspirada pela frase:

“*The best way to learn git is probably to first only do very basic things and not even look at some of the things you can do until you are familiar and confident about the basics”*, segundo seu criador Linus Torvalds.

O git, possui um fluxo a qual possibilita o compartilhamento dos repositórios e projetos na comunidade, sendo assim, para atingir esse compartilhamento, a ferramenta depende de comandos básicos para enviar os projetos e subir em produção, segundo Valente (2020).

Para iniciar todo o fluxo de desenvolvimento, utilizando a ferramenta de controle de versionamento, na interface do console do terminal, basta digitar o comando *init* ou *clone*. O comando *init* cria um repositório vazio. O segundo comando — *clone* — primeiro chama *init* para criar um repositório vazio. Em seguida, ele cópia para esse repositório todos os *commits* de um repositório remoto, passado como parâmetro, segundo Valente (2020).

Portanto, devemos usar o comando clone, quando vamos trabalhar com um projeto já em desenvolvimento, e gostaríamos de alterá-lo, assim o comando permite clonar seu repositório localmente para desenvolvimento. Já o *init*, é recomendado para iniciar um projeto do zero, criando um repositório vazio para ser preparado para realizar seu compartilhamento. Abaixo confere-se o comando clone:

*git clone* [*https://github.com/NOME-USER/NOME-REPO*](https://github.com/NOME-USER/NOME-REPO)

Após o início do fluxo da ferramenta, com o *git clone* ou *git init*, para prosseguir com o seu fluxo, o próximo comando seria a criação de uma *Branch* para seu desenvolvimento das novas alterações. Então, para a criação de uma nova Branch a partir da master que já contêm a versão do projeto completo, é: *git checkout -b [nome-sua-branch].*

Para trocar de *Branch* se caso queira desenvolver a partir de uma *Branch* já criada e específica, basta apenas utilizar o comando: *git checkout [nome-branch].*

Com base no fluxo, após a finalização das novas alterações, para compartilhar para a *Branch* master onde se encontra a versão do projeto por completo, basta você utilizar o comando: *git add [nome-arquivo-alterado]* ou ainda para facilitar, *git add* , o parâmetro, funciona como *all*, para selecionar e adicionar todos os arquivos alterados.

Para visualizar, e checar se adicionou na fila para subir os arquivos corretos que foram feitos as alterações, utiliza-se o comando: *git status*.

O comando *status*, permite você verificar se não adicionou nada a mais e nada a menos, servindo como ponto de segurança, antes de subir para a master as alterações.

Sendo assim, após verificar as mudanças, a próxima etapa é o *commit*. O comando *commit*, segundo o Valente (2020), são usados para criar *snapshots* (ou fotografias) dos arquivos de um sistema. Uma vez tiradas essas fotografias, elas são armazenadas no sistema de controle de versões, de forma compactada e otimizada, para não ocupar muito espaço em disco. Posteriormente, pode-se recuperar qualquer uma das fotografias, para, por exemplo, restaurar uma implementação antiga de um determinado arquivo.

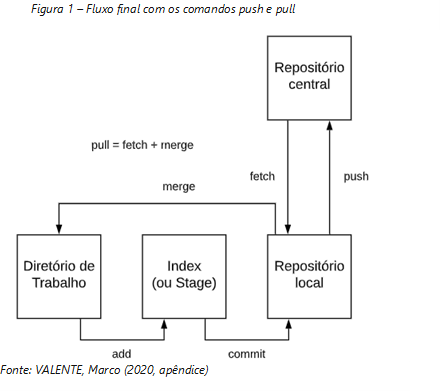
Recomenda-se que desenvolvedores realizem *commits* periodicamente, sempre que tiverem efetuado uma mudança importante no código. A seguir temos o comando *commit*: *git commit – m “sua-mensagem-de-alteração”.*

*Commits* também possuem metadados, incluindo data, hora, autor e uma mensagem, que descreve a modificação realizada pelo *commit*. Em seguida, para de fato enviar os *commits*, utiliza-se o comando *push* e *pull*.

O comando *git push* copia os *commits* mais recentes do repositório local para o repositório remoto. Portanto, ele é uma operação mais lenta, pois envolve comunicação pela rede. Um *push* deve ser usado quando o desenvolvedor deseja tornar uma modificação visível para os demais desenvolvedores. Para atualizar seu repositório local, os outros desenvolvedores do time devem executar um comando *git pull*, segundo o Valente (2020), e por fim o comando *pull* atualiza os arquivos do diretório de trabalho. Essa operação é chamada de merge, e demarca o fim do fluxo da ferramenta para compartilhamento e versionamento de projetos.

A figura a seguir mostra como funciona o processo final do fluxo da ferramenta git:

Figura - Fluxo final com os comandos *push* e *pull*



Fonte: VALENTE, Marco (2020, apêndice)

# DESENVOLVIMENTO

Imagina-se a seguinte situação, um usuário saí para uma viagem com sua família e, com seu veículo doméstico, porém no meio dessa viagem acontece um imprevisto com seu veículo, ele está em um local onde não conhece nenhum profissional que consegue naquele momento lhe ajudar, para fazer uma manutenção corretiva em seu veículo, pode ser um problema mecânico, elétrico, pane seca, ou até mesmo um guincho. Atualmente temos inúmeros veículos que não possuem seguro, segundo site Folha Vitoria: apenas 30% dos carros que circulam no Brasil possuem algum tipo de seguro, segundo pesquisa realizada pela Confederação Nacional das Empresas de Seguros Gerais (CNSEG). (FOLHA VITORIA, 2022).

Com base na matéria citado acima, observamos que existem uma expressiva quantidade de veículos circulando nas ruas e rodovias, sem seguro podendo gerar um transtorno ao motorista e passageiros que estão no veículo, se caso precisar de algum tipo de socorro para o veículo, podendo não ter alternativa de forma rápida e otimizada. E isso acontece com frequência pois segundo site blog Nakata citou: O Brasil ocupa o quinto lugar no ranking dos países com maiores fatalidades no trânsito e as causas, segundo a Polícia Rodoviária Federal (PRF), são: falta de atenção (30,8%); velocidade incompatível com a via (21,9%); mistura de álcool e direção (15,6%); desobediência à sinalização das vias (10%); outras causas (21,7%). (BLOG NAKATA, 2019).

Se analisarmos OUTRAS CAUSAS (21,7%) que são um número expressivo, está incluso falta de manutenção preventiva, eventuais fatalidades envolvendo outros fatores, e muitas vezes isso pode ser intensificado até pelo horário, como por exemplo dirigindo em períodos noturnos, em datas de finais de semana e feriados, que também são bem expressivas as datas dessas causalidades. Pessoas ficando à mercê das situações muitas das vezes sem alternativa, para resolver aquele problema.

Outro agravante que, carros atuais estão saindo de fábricas com estepe de medida mais fina com as rodas originais do carro, gerando um transtorno grande para os motoristas, com base na matéria da UOL escreveu Alessandro Reis: “Já faz algum tempo que muitos carros têm saído de fábrica com estepe do tipo "bicicleta": mais fino do que os demais pneus, o item sobressalente traz menos segurança e tem uso limitado a velocidades de até 80 km/h e a uma distância de 80 km, em média.” (UOL, 2022). Outro ponto Alessandro ressalta que: “Além disso, veículos já têm sido comercializados sem nenhum estepe - substituído por pneus do tipo "run flat", capazes de rodar mesmo furados. Essa tecnologia traz a vantagem de você não ter de parar para substituir o pneu, mas existem várias ressalvas.” (UOL, 2022).

No caso dos veículos que tem o estepe ele fica expostos a outros problemas que pode ser causado passando por buracos, outras irregularidades na pista, sendo isso corriqueiro no nosso país, os motoristas não podem percorrer a mesma velocidade se tivesse com os pneus originais em condições, tendo que reduzir essa velocidade para não correr risco de acidentes. Já o veículo que tem o pneu “run flat” ele custa mais caro que outro carro, que tem o pneu convencional, e ele se a caso este pneu furar terá que andar com uma velocidade reduzida, ambos os casos geram um transtorno para o motorista.

Atualmente não temos nenhuma solução que implica uma solicitação de socorro veicular, tirando casos que a pessoa tenha seguro do veículo, ou esteja em uma rodovia com pedágio.

## Levantamento de Requisitos

Neste tópico, é mostrado um questionário feito pelo *Google Forms* com o nome: Pesquisa do uso de veículos, criado no dia 30 de agosto de 2022, esse questionário ficou aberto entre os dias 30 de agosto de 2022 a 20 de novembro de 2022, com 3 pré-perguntas e 10 perguntas com seus devidos objetivos, com 135 pessoas respondendo o questionário e todos do estado de São Paulo.

Para calcular o tamanho necessário de uma amostra a ser considerada, foi usado o site *Survey Monkey*, na ferramenta, o tamanho da população foi de 12 milhões (o número da população do estado de São Paulo), com um grau de confiança de 80% e uma margem de erro de 7%, o cálculo retornou 84 do número mínimo da amostra.

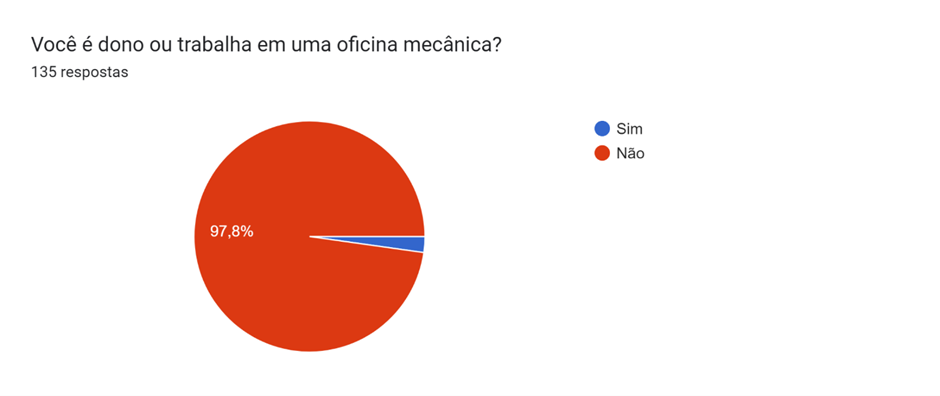
Figura - Calculadora de amostra da Survey Monkey



Fonte: Survey Monkey(2022)

Tendo duas partes, o questionário divide pessoas que são donas ou trabalham em uma oficina mecânica das pessoas que só são motoristas, onde 2,2% das pessoas (3 pessoas) responderam que tem ou trabalha em uma oficina e 97,8% das pessoas (132 pessoas) responderam que não, o que acaba anulando uma amostra de quem trabalha com oficinas mecânicas, trabalhando apenas com uma amostra de 132 dados, vulgo motoristas. Algumas questões foram adicionadas conforme algumas análises, sendo as perguntas 5, 6 e 8 ficando com 128 dados e a questão de número 9 com 88 dados.

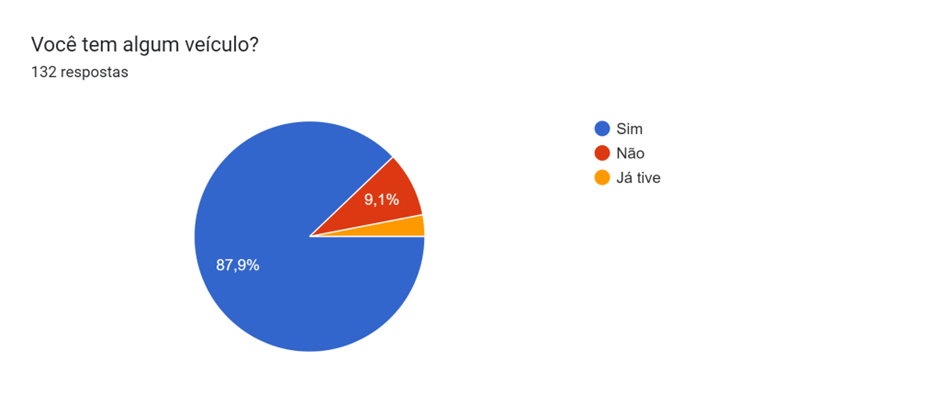
Gráfico – Pergunta separatória



Fonte: Pesquisa do uso de veículos - Google Forms

Na parte para os motoristas, a primeira pergunta é se o motorista tem algum veículo, tendo 87,9% dos motoristas (116) respondido “Sim”, 9,1% (12) responderam “Não” e 3% (4) responderam “Já tive”.

Gráfico 2 – 1° pergunta



Fonte: Pesquisa do uso de veículos - Google Forms

Essa pergunta teve o objetivo de introduzir quem está iniciando o formulário e para ter uma noção do tamanho do público-alvo do aplicativo.

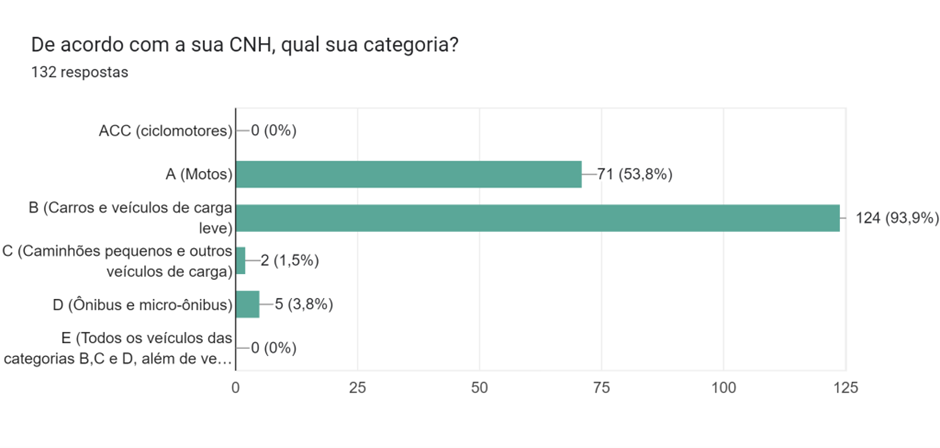
Na segunda pergunta, é perguntada a categoria da Carteira Nacional de Habilitação (CNH), tendo como opções:

* ACC (ciclomotores)
* A (Motos)
* B (Carros e veículos de carga leve)
* C (Caminhões pequenos e outros veículos de carga)
* D (Ônibus e micro-ônibus)
* E (Todos os veículos das categorias B, C e D, além de veículos com reboque)

Lembrando que poderia selecionar mais de uma opção, o resultado foi:

* ACC - 0% (0)
* A - 53,8% (71)
* B - 93,9% (124)
* C - 1,5% (2)
* D - 3,8% (5)
* E - 0% (0)

Gráfico – 2° pergunta

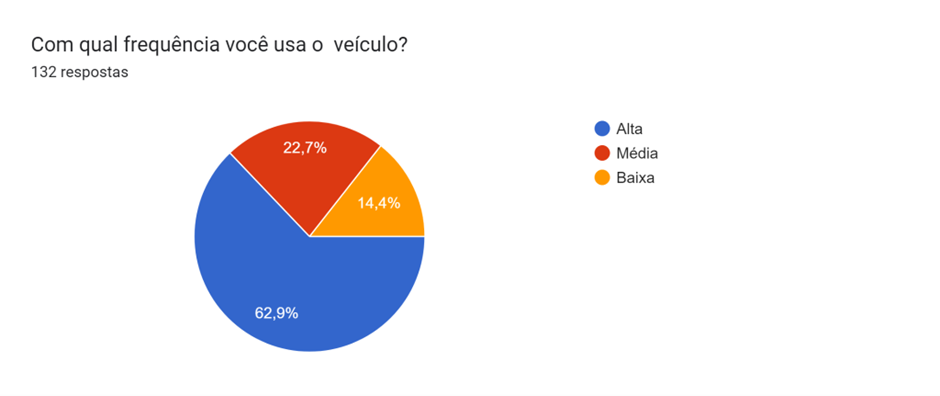


Fonte: Pesquisa do uso de veículos - Google Forms

Com essa questão, é possível observar com mais clareza que os carros são maioria entre os motoristas, logo, são eles que recebe uma devida atenção para diversos aspectos em nossa sociedade.

Na terceira questão, é perguntado a frequência do uso do veículo, 62,9% (83) responderam “Alta”, 22,7% (30) responderam “Média” e 14,4% (19) responderam “Baixa”.

Gráfico – 3° pergunta

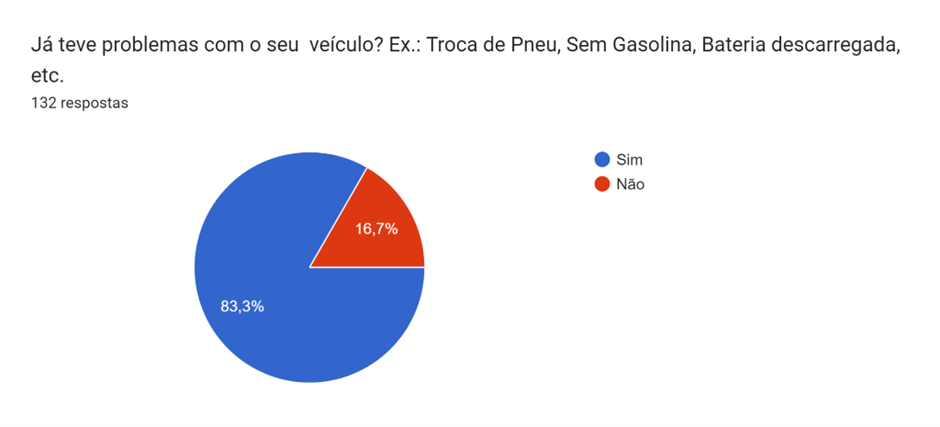


Fonte: Pesquisa do uso de veículos - Google Forms

Com esse gráfico pode-se analisar que a maioria dos motoristas fazem uso do seu veículo com muita frequência, fazendo com que o veículo acabe precisando de manutenções com mais constância.

Na quarta questão, é perguntado se já teve problemas como troca de pneu, ficar sem combustível, bateria descarregada entre outros com o veículo, 83,3% (110) responderam “Sim” e 16,7% (22) responderam “Não”.

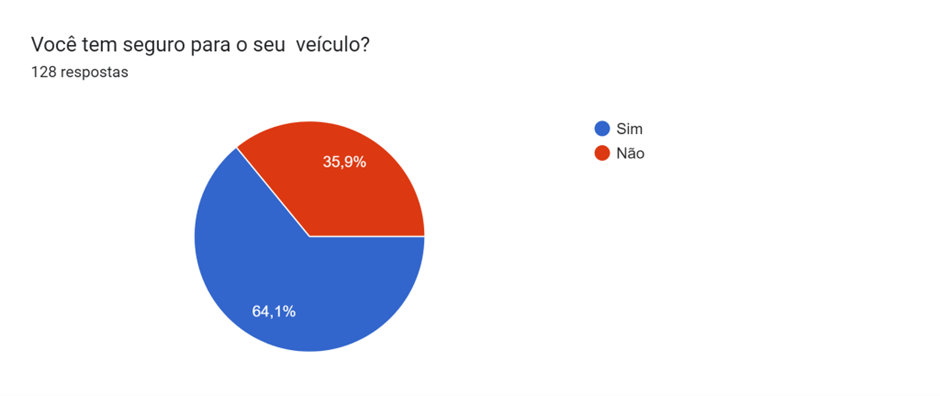
Gráfico 5 – 4° pergunta



Fonte: Pesquisa do uso de veículos - Google Forms

Conforme a maioria, muitos motoristas têm diversos tipos de problemas com o seu veículo, de acordo com Andrade L. (2020), no Brasil, a polícia rodoviária federal entregou um relatório em que mais de 12 mil acidentes de trânsito aconteceram por conta de defeitos mecânicos, no período de 2017 a 2019, mostrando que a manutenção do carro é muito importante para a vida do motorista.  
  
Na quinta questão, é perguntado se há algum serviço de seguro para o veículo, tendo 128 respostas, 64,1% (82) responderam “Sim” e 35,9% (46) responderam “Não”.

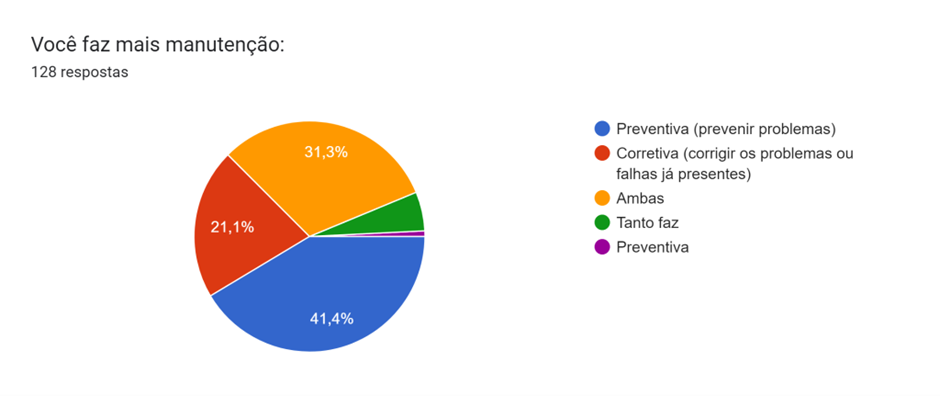
Gráfico – 5° pergunta



Fonte: Pesquisa do uso de veículos - Google Forms

Na pergunta, consegue-se analisar um grande número de motoristas com o seguro do veículo, mesmo assim, quase 40% não contrataram serviço de seguro, demostrando que muitos motoristas estão sujeitos a sofrerem panes em seus veículos advindos de situações casuísticas adversas.  
  
 Na sexta questão, é perguntado sobre o tipo de manutenção que o motorista faz em seu veículo, com 128 respostas, 42,2% (54) escolheram a manutenção preventiva (prevenindo de falhas futuras), 21,1% (27) selecionaram a manutenção Corretiva (correção de falhas já existentes), 31,3% (40) preferem fazer os dois tipos de manutenção e 5,5% (7) acham que tanto faz o tipo da manutenção.

Gráfico – 6° pergunta

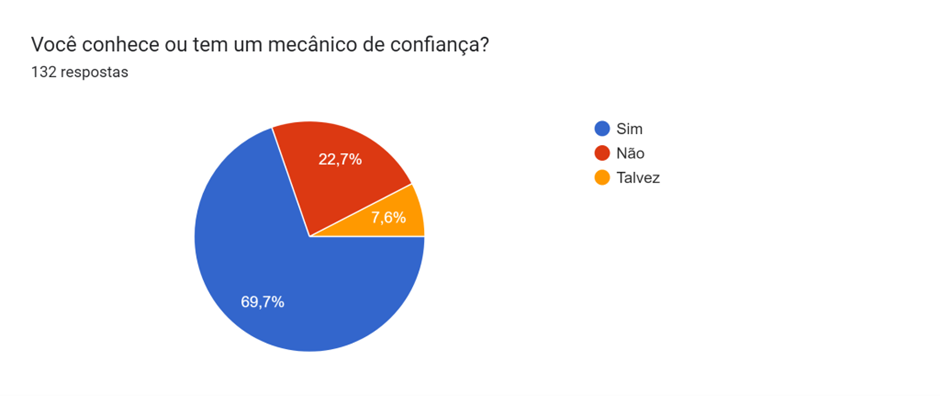


Fonte: Pesquisa do uso de veículos - Google Forms

Por problemas com uma alteração na pergunta, as opções “Preventiva (prevenir problemas)” e “Preventiva” são a mesma pergunta e foram juntadas para a apresentação, “Preventiva” teve uma seleção (0,8%).  
  
 Observando o gráfico podemos analisar que a maioria dos motoristas preferem prevenir a manutenção antes mesmo que ocorra, mostrando que se programam com antecedência em procurar um mecânico, porém, muita das vezes não se sabe como é o trabalho do mecânico em si, tendo que perguntar há amigos e conhecidos para ter uma segurança de que o trabalho será bem-feito.

Na sétima questão, perguntamos se há um mecânico de confiança para o motorista, 69,7% (92) responderam “Sim”, 22,7% (30) responderam “Não” e 7,6% (10) responderam “Talvez”.

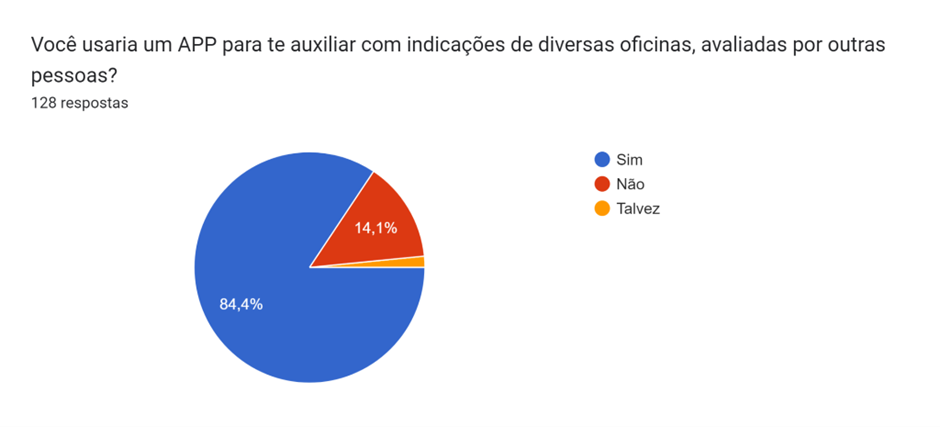
Gráfico – 7° pergunta



Fonte: Pesquisa do uso de veículos - Google Forms

Com essa pergunta, é possível verificar a enorme confiança do motorista com o seu mecânico, entre tanto, em situações de viagens entre cidades ou estados, se aparece qualquer problema com o veículo, o motorista terá que confiar no primeiro mecânico que aparecer, gerando dúvidas se aquela oficina é boa no seu trabalho e se é o melhor preço da região.  
  
 Na oitava questão, foi feita uma pergunta diretamente ao uso de um possível aplicativo de celular, onde auxiliaria com indicações de oficinas avaliadas por outras pessoas, tendo 128 respostas, 84,4% (108) responderam “Sim”, 14,1% (18) responderam “Não” e 1,6% (2) responderam “Talvez”.

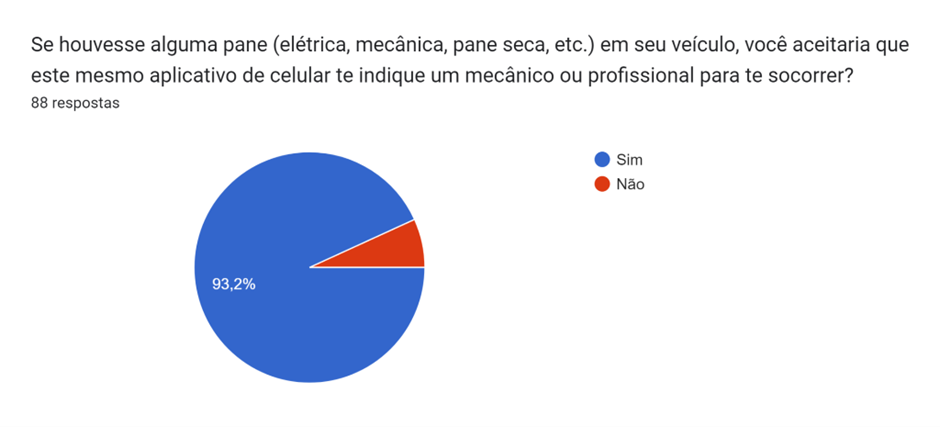
Gráfico 9 – 8° pergunta



Fonte: Pesquisa do uso de veículos - Google Forms

Com base no gráfico, muitos motoristas gostariam de um aplicativo para auxiliá-lo na pesquisa de diversos tipos de oficinas, confiando na avaliação de outras pessoas para facilitar a busca, mostrando que há uma carência na população em achar um mecânico do seu agrado.  
  
 Na nona questão, foi feita como um complemento para a pergunta anterior, se o motorista gostaria de receber indicações do próprio aplicativo de oficinas mecânicas ou profissionais para socorrer (em caso de acabar o combustível no meio da estrada e outros tipos de pane em seu veículo), com 88 respostas, 93,2% (82) responderam “Sim” e 6,8% (6) responderam “Não”.

Gráfico 10 – 9° pergunta



Fonte: Pesquisa do uso de veículos - Google Forms

O intuito dessa questão foi analisar até onde os motoristas gostão de receber indicações, com mais de 90% gostando de receber indicações até mesmo do próprio aplicativo.  
  
 Como uma pergunta opcional, foi perguntado o que gostaria de ver em um futuro aplicativo de auxílio com o veículo, das 135 pessoas apenas 40 responderam, tendo algumas respostas interessantes como:

* Dicas para melhorar a conservação do carro;
* Botão S.O.S onde venha um mecânico próximo da sua localização;
* Campos para colocar peças e comparar com a quilometragem, para saber a próxima manutenção e troca de óleo;
* Troca de mensagens com o mecânico;
* Mostrar o mapa com todas as localizações de mecânicos cadastrados em um raio próximo, com avaliações, comentários, preços entre outros;

## Proposta de Solução

* + 1. **Sobre o Aplicativo**

Com base na problemática e tendo em vista a proposta de solução, o aplicativo móvel contará prioritariamente com um mecanismo de solicitação rápida de socorro para atender uma necessidade extraordinária de pane veicular.

Tratando-se da solicitação de socorro com urgência, o aplicativo dará a possibilidade para o usuário solicitar atendimento, onde o usuário deverá informar seus dados, selecionar a natureza ou categoria do socorro (se é mecânica, elétrico, borracharia, pane seca etc.), selecionar qual dos prestadores de serviço (profissionais ou oficinas) recomendados pela aplicação ele deseja solicitar e confirmar a solicitação. Após a confirmação pelo usuário, a aplicação notificará o prestador de serviço solicitado dando a possibilidade de este aceitar ou não o atendimento.

Caso o prestador de serviço aceite o atendimento, a aplicação notificará o usuário solicitante e dará início ao processo de atendimento que é contemplado pelas seguintes etapas:

* **Rastreio do Prestador:** nesta etapa o usuário terá em seu aplicativo a possibilidade de rastrear em tempo real o prestador de serviço se locomovendo até o local de atendimento.
* **Realizando Atendimento:** nesta etapa o prestador de serviço já está no local realizando o atendimento da solicitação de socorro.
* **Atendimento Finalizado:** nesta última etapa o atendimento é tido como finalizado.

Em situação em que o prestador de serviço não aceitar a solicitação de atendimento, a aplicação notificará o usuário, que consequentemente poderá selecionar outro prestador para lhe atender.

Para casos não urgentes, será disponibilizado também uma rotina para solicitação de agendamento de serviços de natureza mecânica, elétrica, borracharia, etc. Esta rotina é iniciada com o usuário indicando a categoria do serviço desejado, selecionando o prestador de serviço recomendado, selecionando uma data e horário disponíveis na agenda do prestador e solicitando o agendamento por fim.

A aplicação se encarregará de notificar o prestador do serviço, dando a este a possibilidade de aceitar ou não o agendamento. Se aceito, a aplicação notificará a confirmação do agendamento ao usuário dando início ao processo de atendimento. Após a realização do serviço, este é definido como finalizado e ficará disponível no histórico de serviços solicitados pelo usuário.

As recomendações de prestadores de serviço disponíveis serão realizadas seguindo os critérios de localização (mais próximos do usuário solicitante), avaliação (de atendimentos anteriores e níveis de confiabilidade) e categoria de serviço em que este se enquadra.

A fim de estabelecer confiança entre o usuário do aplicativo e o profissional que lhe irá atender, e a fim de prover autenticação de acesso a aplicação e suas funções, o aplicativo contará com um cadastro para o usuário e para o profissional. Este cadastro contemplará a inserção de dados pessoais reais (seja pessoa jurídica ou pessoa física) para ambos.

Todos os dados de cadastro, tanto do usuário como também do provedor de serviços serão dispostos em uma seção de perfil. Sendo possível a edição destes dados e solicitação de exclusão do cadastro atendendo ao artigo 18 da Lei Geral de Proteção de Dados (LGPD) que em seu artigo 18, menciona que o titular poderá a qualquer momento e mediante requisição, solicitar a eliminação dos dados pessoais tratados mesmo com o devido consentimento.

O titular dos dados pessoais tem direito a obter do controlador, em relação aos dados do titular por ele tratados, a qualquer momento e mediante requisição: VI - eliminação dos dados pessoais tratados com o consentimento do titular, exceto nas hipóteses previstas no art. 16 desta Lei. (BRASIL, 2018).

Todos os atendimentos sejam eles solicitados por socorro ou por agendamento serão disponibilizados num histórico para consultas posteriores.

As demais funções da aplicação estão relacionadas a ajustes, configurações ou preferências do usuário, por exemplo: alteração de tema do aplicativo.

Como é prevista a escalabilidade da aplicação, haverá demais funções que poderão ser implementadas futuramente através de atualizações.

* + 1. **Protótipos**

Figura - Tela de Apresentação



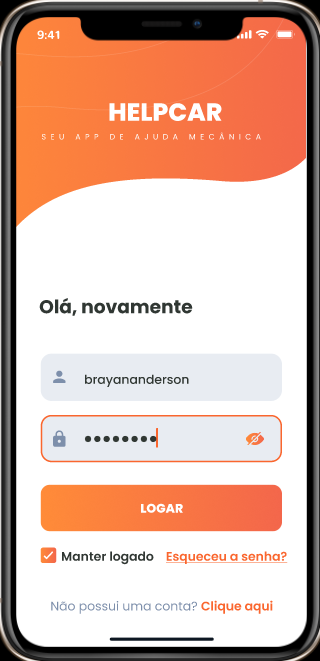
Fonte: SANTOS, Luana ( Figma, Helpcar - seu app de ajuda mecânica)

Figura - Tela de Apresentação 2



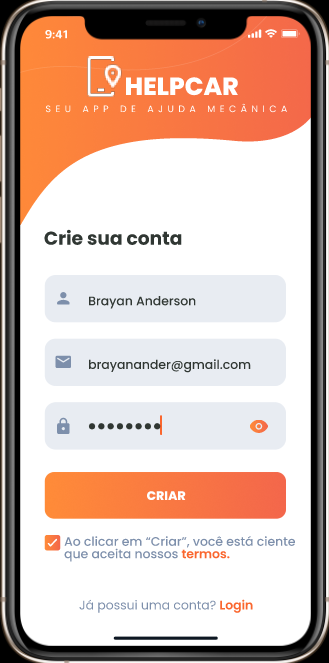
Fonte: SANTOS, Luana ( Figma, Helpcar - seu app de ajuda mecânica)

Figura - Tela de *Login*



Fonte: SANTOS, Luana ( Figma, Helpcar - seu app de ajuda mecânica)

Figura - Tela de Cadastro



Fonte: SANTOS, Luana ( Figma, Helpcar - seu app de ajuda mecânica)

Figura - Tela Inicial



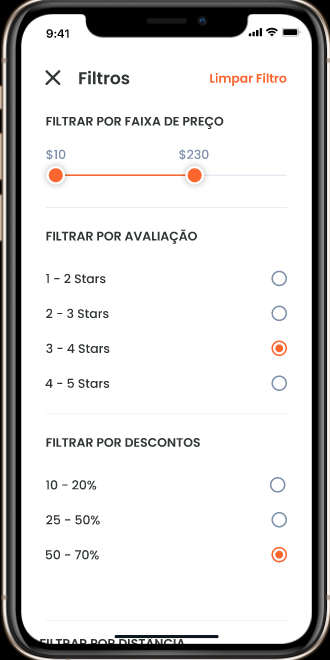
Fonte: SANTOS, Luana ( Figma, Helpcar - seu app de ajuda mecânica)

Figura 28 - Tela de Busca



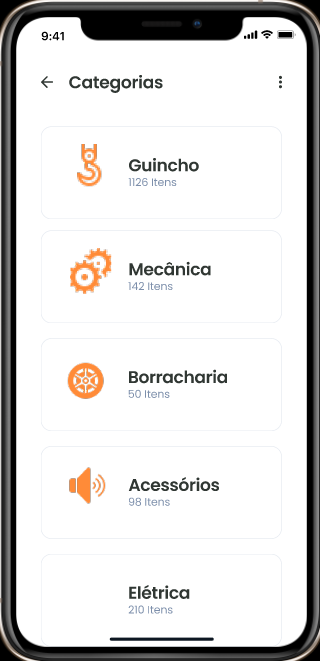
Fonte: SANTOS, Luana ( Figma, Helpcar - seu app de ajuda mecânica)

Figura - Tela de Filtros



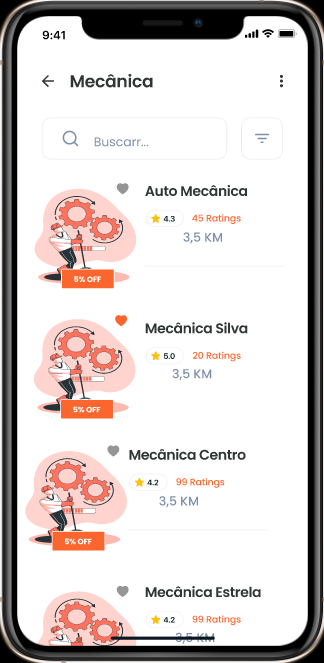
Fonte: SANTOS, Luana ( Figma, Helpcar - seu app de ajuda mecânica)

Figura - Tela de Categoria de Serviços



Fonte: SANTOS, Luana ( Figma, Helpcar - seu app de ajuda mecânica)

Figura - Tela de Seleção de Serviços



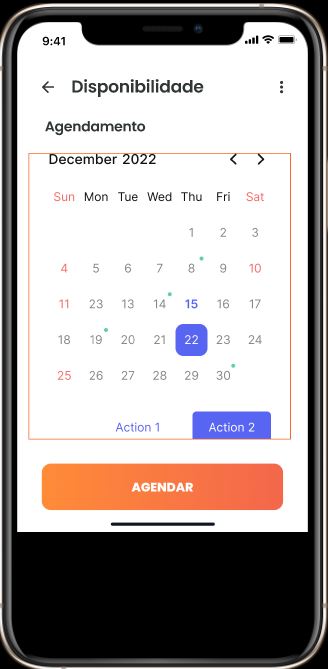
Fonte: SANTOS, Luana ( Figma, Helpcar - seu app de ajuda mecânica)

Figura 32 - Tela de Detalhes do Serviço



Fonte: SANTOS, Luana ( Figma, Helpcar - seu app de ajuda mecânica)

Figura - Tela de Agendamento



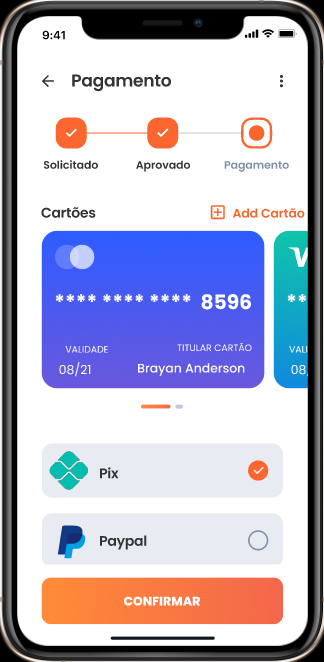
Fonte: SANTOS, Luana ( Figma, Helpcar - seu app de ajuda mecânica)

Figura - Tela dos Dados



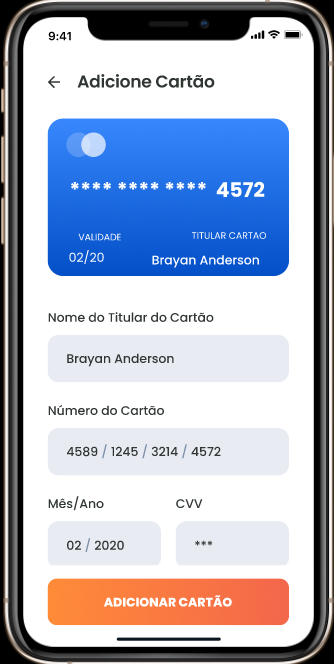
Fonte: SANTOS, Luana ( Figma, Helpcar - seu app de ajuda mecânica)

Figura - Tela de Pagamento



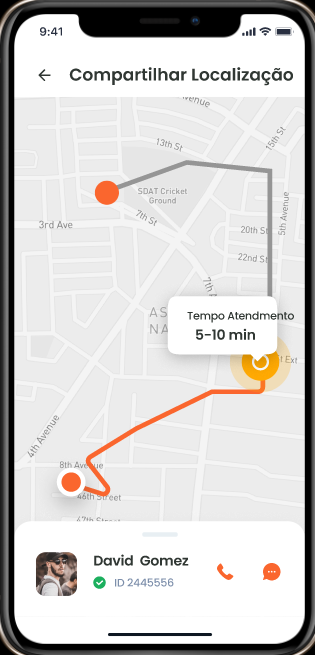
Fonte: SANTOS, Luana ( Figma, Helpcar - seu app de ajuda mecânica)

Figura - Tela de Adicionar Novo Cartão



Fonte: SANTOS, Luana ( Figma, Helpcar - seu app de ajuda mecânica)

Figura - Tela de Localização



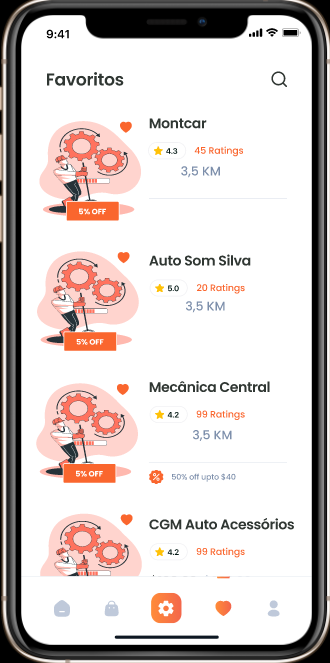
Fonte: SANTOS, Luana ( Figma, Helpcar - seu app de ajuda mecânica)

Figura 38 - Tela de Histórico



Fonte: SANTOS, Luana ( Figma, Helpcar - seu app de ajuda mecânica)

Figura - Tela de Favoritos



Fonte: SANTOS, Luana ( Figma, Helpcar - seu app de ajuda mecânica)

Figura - Tela de Perfil do Usuário



Fonte: SANTOS, Luana ( Figma, Helpcar - seu app de ajuda mecânica)

**REFERÊNCIAS BIBLIOGRÁFICAS**

Survey Monkey: <https://pt.surveymonkey.com/mp/sample-size-calculator/>

ANDRADE Laurie, Falta de manutenção é uma das principais causas de acidentes de trânsito, 2020. Disponível em: <https://autopapo.uol.com.br/noticia/acidentes-transito-falta-manutencao/>. Acessado em: 21 de novembro de 2022

CASTELLS. M. a Sociedade em rede. São Paulo. Paz e Terra, 1999

MONTEIRO, Luís. A internet como meio de comunicação: possibilidades e limitações. In: Congresso Brasileiro de Comunicação. 2001.

TORRES, Victor Monteiro. HTML e seus Componentes. Revista Ada Lovelace, v. 2, p. 99-101, 2018. Disponível em: <http://anais.unievangelica.edu.br/index.php/adalovelace/article/view/4652>. Acesso em 15 out. de 2022.

SOUTO, Mario. Front-end, Back-end e Full Stack. Alura, 2022. Disponível em: <https://www.alura.com.br/artigos/o-que-e-front-end-e-back-end>. Acesso em: 15 out. de 2020.

FERREIRA, André. A história da internet - Back-end, 2022. Disponível em: <https://andreferreira.com.br/2022/09/29/a-historia-da-internet-back-end/>. Acesso em 16 out. de 2022.

TUCKER, A. B.; NOONAN, R. E. Linguagens de Programação: Princípios e Paradigmas. Porto Alegre: AMGH, 2010

BRASIL. Lei nº 13.709, de 14 de agosto de 2018. Lei Geral de Proteção de Dados (LGPD), 2020. Disponível em: <https://www.in.gov.br/materia/-/asset_publisher/Kujrw0TZC2Mb/content/id/36849373/do1-2018-08-15-lei-no-13-709-de-14-de-agosto-de-2018-36849337> . Acesso em: 30 nov. 2022.

TAPSCOTT, D. Economia digital: promessa e perigo na era da inteligência em rede. São Paulo: Makron Books. 1997.

PORTER, M. E. Estratégia competitiva: técnica para a análise da indústria e da concorrência. Rio de Janeiro: Campus, 1986.

SERRA, J. Paulo. Manual de Teoria da Comunicação. Covilhã: Livros Labcom, 2007.

MORESI, E. A. D. Monitoramento ambiental. In: TARAPANOFF, Kira (Org.). Inteligência organizacional e competitiva. Brasília: Editora Universidade de Brasília, 2001.

BEZERRA, Eduardo. Princípios de Análise e Projeto de Sistemas com UML. 1ª Edição, Elsevier, 2014.

VALENTE, Marco Tulio. Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade, Editora: Independente, 2020.

KORTH, Henry F.; SUDARSHAN, S.; SILBERSCHATZ, Abraham. Sistema de banco de dados. São Paulo: Makron Books, 1999.

HEUSER, C. A. Projeto de banco de dados. 6. ed. São Paulo: Editora Bookman, 2009.

BIEHL, Matthias. API Architecture: The Big Picture for Building APIs - Volume 2. API-University, 2015.

ELMASRI, Ramez e NAVATHE, Shamkant B. Sistemas de Banco de Dados. Pearson Addison Wesley. 6a Edição, 2011.

Conceptual, Logical and Physical Data Model. Visual Paradigm, 2022. Disponível em:https://www.visual-paradigm.com/support/documents/vpuserguide/3563/3564/85378\_conceptual,l.html. Acesso em: 15 out. de 2020.