

CENTRO PAULA SOUSA
ESCOLA TÉCNICA ESTADUAL DA ZONA LESTE
M-Tec Desenvolvimento de Sistemas AMS

Ester Rodrigues Soares
Gabrielly Nascimento Bento
Gustavo Henrique Ribeiro da Silva
Jhonata Alves do Nascimento

**SAFEVIEW: Sistema de diminuição de pontos cegos para veículos
de grande porte**

São Paulo
2025

Ester Rodrigues Soares
Gabrielly Nascimento Bento
Gustavo Henrique Ribeiro da Silva
Jhonata Alves do Nascimento

**SAFEVIEW: Sistema de diminuição de pontos cegos para veículos
de grande porte**

Trabalho de Conclusão de Curso
apresentado ao M-Tec
Desenvolvimento de Sistemas da Etec
Zona Leste, orientado pelo Prof. Esp.
Jeferson Roberto de Lima, com
requisito parcial para obtenção do título
de técnico em Desenvolvimento em
Sistemas.

São Paulo
2025

DEDICATÓRIA

AGRADECIMENTOS

ΕΠΙΓΡΑΦΕ

RESUMO

Palavras-Chave:

ABSTRACT

Keywords:

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

LISTA DE SÍMBOLOS

SUMÁRIO

1	INTRODUÇÃO	1
2	REFERENCIAL TEÓRICO	3
2.1	Acidentes veiculares e consequências dos pontos-cegos	3
2.2	Tecnologias Utilizadas.....	3
2.2.1	Internet of Things (IoT).....	4
2.2.2	Raspberry PI	4
2.2.3	ESP32.....	5
2.2.4	Wireless Fidelity (Wi-Fi)	7
2.2.5	Python.....	7
2.2.5.1	Package Installer for Python (PIP).....	8
2.2.6	Open Search Computer Vision Library (OpenCV)	9
2.2.7	Flask	13
2.2.8	C++	13
2.2.9	HyperText Markup Language (HTML).....	13
2.2.10	Cascading Style Sheets (CSS)	16
2.2.11	JavaScript (JS).....	20
2.2.12	React Native	21
2.2.13	Node JS	22
2.2.13.1	NPM.....	22
2.2.14	Banco de Dados	22
2.2.15	Firebase	23
2.2.15.1	Cloud Firestore	23
2.2.16	Wireframe	24
2.2.17	Unified Modeling Language (UML)	25
2.2.17.1	Diagrama de Caso de Uso.....	26
2.2.17.2	Diagrama de Sequência	28

2.2.17.3	Diagrama de Atividade.....	29
2.2.17.4	Diagrama de Máquina-Estado	30
3	DESENVOLVIMENTO.....	31

1 INTRODUÇÃO

É amplamente reconhecido que a segurança é um fator indiscutível no que diz respeito à condução de caminhões em vias públicas. Estima-se que uma parcela significativa dos acidentes relacionados a esses veículos ocorre por conta da falta de visibilidade dos motoristas com relação a outros elementos ao seu redor, como ciclistas, pedestres, veículos menores, dentre outros, que são vítimas regulares desses incidentes. Tendo em vista os dados posteriormente apresentados, este trabalho se ampara na necessidade de fornecer maior segurança aos condutores de caminhão, considerando a visibilidade limitada do entorno do veículo. Para tanto, propõe-se implementar um sistema integrado, físico e digital, capaz de disponibilizar imagens e alertas sobre o que está acontecendo em volta do caminhão para evitar distrações desnecessárias e facilitar a execução de manobras durante o trajeto.

O direcionamento do projeto é consolidado com a metodologia qualitativa, pois as pesquisas e estudos existentes não oferecem um número expressivo de dados estatísticos, o que inviabiliza a utilização correta dos elementos mencionados. Contudo, essas pesquisas e estudos evidenciam fatores que explicam as tragédias no trânsito. Como descrito por Marconi e Lakatos (2017), as mudanças qualitativas não provêm das quantitativas, mas as substituem, desfazendo uma progressão latente, introduzindo algo novo que se baseia em fenômenos distintos para justificar-se. Portanto, nos casos em que as medidas quantitativas não demonstram considerável aplicabilidade, optamos pelas qualitativas, como acontece nessa pesquisa.

As questões mencionadas nesse estudo advêm da realidade incontestável de que o trabalho exercido pelos caminhoneiros é extremamente crucial para a economia brasileira, pois resguardam o fluxo econômico do país. Uma evidência clara desse fenômeno foi a greve dos caminhoneiros ocorrida em 2018, que impactou vários setores da economia nacional.

As extensas e ininterruptas jornadas de trabalho e as circunstâncias desfavoráveis às quais os motoristas estão submetidos, como condições climáticas, estradas em péssimo estado e a visibilidade reduzida no período noturno, dão origem à fadiga e exaustão. Esses fatores contribuem para uma maior dificuldade na percepção de

obstáculos nas vias ou nos entornos do veículo, levando os condutores a seus limites físicos e psicológicos, o que intensifica a frequência dos acidentes.

A partir de uma análise inicial superficial sobre o assunto, surgiram as principais hipóteses acerca das causas dos incidentes: o tamanho dos caminhões, principalmente devido ao baú, relacionado à visibilidade prejudicada em determinados pontos ao redor do veículo. Na atualidade, o financiamento de tecnologias similares à proposta, por empresas de pequeno porte e motoristas autônomos, representa uma dificuldade devido aos seus custos elevados. Em um cenário no qual a segurança é essencial, a adoção do presente projeto, denominado SafeView, possibilitaria ao motorista a visualização do ambiente externo, bem como o investimento em uma tecnologia acessível e de operação simples, tornando essa solução um recurso vantajoso para aqueles que buscam certificar-se tanto da própria segurança quanto da segurança daqueles ao seu redor, contribuindo, conseqüentemente, de modo ativo para a diminuição de acidentes durante a condução do veículo.

Para definir o problema, foram selecionados estudos nos relatórios anuais emitidos entre 2019 e 2021 pela Companhia de Engenharia de Tráfego (CET), responsável pelo gerenciamento e planejamento do trânsito no estado de São Paulo. Da mesma forma, serão consideradas notícias fornecidas pela Polícia Rodoviária Federal (PRF), cujas ações têm impacto direto na segurança viária e na fluidez do tráfego. Relacionada à presente pesquisa, a Internet das Coisas (conhecida como IoT, do inglês *Internet of Things*) será usada com uma abordagem próxima à de Sérgio de Oliveira (2021), na qual, além da interconexão dos aparelhos, eles funcionam de maneira inteligente, sendo capazes de coletar e processar as informações dos dispositivos físicos. Com essa combinação de fontes, é possível obter uma visualização mais ampla e integrada dos desafios em questão, o que é essencial para a construção de uma solução no contexto estudado.

Neste documento, serão descritas as etapas de pesquisa para o referencial teórico e os processos de desenvolvimento do projeto SafeView – Revelando o que os olhos não veem.

2 REFERENCIAL TEÓRICO

Neste presente capítulo será apresentada, de forma a abranger o máximo possível, uma perspectiva sobre os conceitos nos quais o projeto SafeView é baseado, contendo as principais tecnologias que o fundamentam e os componentes utilizados durante a produção do sistema.

2.1 Acidentes veiculares e consequências dos pontos-cegos

Em 2019, o Relatório Anual de Acidentes no Trânsito, fornecido pela CET (Companhia de Engenharia de Tráfego do estado de São Paulo), relata cerca de 758 acidentes fatais relacionados à atropelamentos, colisões, choques, dentre outros, e 791 óbitos. Grande parte das vítimas de acidentes de trânsito são os motociclistas, que muitas vezes passam despercebidos e são atingidos nas ruas por outros automóveis.

A Polícia Rodoviária Federal (2024) confirma que, dentre as causas que resultam essa classe de sinistros (acidentes de trânsito), estão os pontos-cegos de veículos de grande porte como os caminhões, ônibus e os veículos rodoviários de maior comprimento. Esses pontos-cegos dificultam a visualização do que está em volta do veículo por conta da baixa angulação nos retrovisores, tornando as manobras no trânsito mais complicadas para os condutores.

Para evidenciar a relevância da temática apresentada, em 2019, os caminhões representaram apenas 1,4% do total de veículos no trânsito. No entanto, estiveram envolvidos em 8,2% dos atropelamentos fatais – proporção aproximadamente 5,89 vezes maior que sua participação no tráfego. Já os ônibus corresponderam a 3% de atuação nas ruas, vinculados a 15,8% dos atropelamentos fatais, percentual 5,26 vezes superior à sua participação inicial. (CET, 2021).

Diante disso, é de extrema importância que esse tema seja abordado, pois há a necessidade em criar uma solução que ofereça segurança e assistência aos motoristas de caminhão, para que tenham maior controle do entorno do veículo e visão ampla de seus pontos-cegos.

2.2 Tecnologias Utilizadas

Na seção a seguir, serão listadas e descritas todas as tecnologias para o funcionamento do projeto SafeView, englobando componentes físicos, linguagens de programação, *frameworks*, dentre outros recursos utilizados neste trabalho.

2.2.1 Internet of Things (IoT)

Conforme apresentado por Magrani (2018), *IoT* (em português Internet das Coisas) é uma nova tecnologia que se comunica entre si para oferecer um sistema efetivo para a resolução de um problema: a Internet das Coisas. Essa denominação foi atribuída porque as “coisas” se referem a pequenos dispositivos eletrônicos que são interconectados, em alguns casos sendo incorporados a utensílios, para torná-los inteligentes, coletando e compartilhando dados.

Por Moraes e Hayashi (2021), a IoT tornou a vida mais tranquila, principalmente em centros urbanos. Um exemplo comum de IoT é a casa inteligente, que proporciona uma comodidade muito grande de ter o controle dos seus dispositivos eletrônicos domésticos. Entretanto, a Internet das Coisas vai além do ambiente residencial, pois, com o poder de coleta e processamento de dados dos ambientes e redes que estão interconectadas, transforma nossa percepção sobre o mundo ao nosso redor, trazendo diversos impactos em áreas de logística, trânsito e segurança. (OLIVEIRA, 2021).

2.2.2 Raspberry PI

Segundo Ebermam *et al.* (2017), o *Raspberry PI* surgiu com a ideia de criar um computador pequeno e acessível para crianças, o que era um grande desafio tendo em vista que já naquela época as crianças estavam acostumadas com eletrônicos mais sofisticados como os *smartphones*.

Consoante a Oliveira (2021), o Raspberry PI teve algumas facilidades que o levaram para o caminho dos sistemas embarcados, já que, mesmo com uma arquitetura reduzida, ele trazia diversas portas de entrada e saída juntamente de uma ótima integração com *Python*, que traz um ambiente de programação completo.

Figura 1 - Raspberry PI 4.



Fonte: Raspberry Pi, 2025.

2.2.3 ESP32

Lançado em 2016 pela empresa *Espressif Systems* (RIBEIRO; VALLE JUNIOR; MARTINS, 2022), o *ESP32* trata-se de um microcontrolador que tem sido cada vez mais conhecido por conta de suas interfaces de comunicação *Wi-Fi* e *Bluetooth*. Outro aspecto importante é que mesmo sendo semelhante a outros microcontroladores como o Arduino Uno, o ESP32 tem maior velocidade de processamento e conectividade, gerando mais uso para projetos de *Internet of Things* (Internet das Coisas), os chamados projetos IoT. (LIMA, 2022).

Neste projeto usaremos a ESP32-CAM, uma versão diferente do microprocessador ESP32, cuja qual possui uma pequena câmera OV2640 de 2MP e espaço para cartão MicroSD acoplados à placa. Ademais, é possível enviar as imagens capturadas por ela através da Internet, podendo armazená-las no cartão MicroSD.

Informações técnicas:

- Modelo: ESP32-CAM;
- Processador: Xtensa® Dual-Core 32-bit LX6;
- Flash: 4;
- RAM: 520 KBytes;
- ROM: 448 KBytes;

- Clock Máximo: 240 MHz;
- Pinos Digitais GPIO: 11;
- Resolução do PWM: até 16 bits;
- Resolução da Foto: 2 Megapixels;
- Wireless 802.11 b/g/n ADC;
- Modos de Operação: Access Point/Estação/Access Point + Estação;
- Bluetooth Low Energy: padrão 4.2 integrado.

Figura 2 - Microcontrolador ESP32 com câmera.



Fonte: Robocore, 2023.

2.2.4 Wireless Fidelity (Wi-Fi)

Conforme constatado por Oliveira (2017), o *Wi-Fi* tem como função conectar dispositivos em redes locais sem a utilização de cabos. A rede considera um determinado número de elementos conectados, embora a extensão desses usuários possa ser infinita.

2.2.5 Python

Com base na Documentação Oficial do Python (2025), a linguagem de programação Python foi desenvolvida por *Guido van Rossum*, em fevereiro de 1991, com o objetivo de superar limitações das linguagens da época, especialmente a falta de extensibilidade dos sistemas desenvolvidos.

Sob a ótica de Menezes (2019), o Python é uma linguagem poderosa e objetiva que tem se tornado amplamente utilizada em diversas áreas da computação – como a inteligência artificial e a biotecnologia – devido à alta escalabilidade de projetos e a fácil manutenção de sistema, pela sua sintaxe clara.

Figura 3 - Exemplo de código em Python.

```
1 def ImparOuPar():  
2     num = int(input("Digite um número: "))  
3  
4     if num % 2 == 0:  
5         print("O número é par.")  
6     else:  
7         print("O número é ímpar.")  
8  
9     ImparOuPar()
```

Fonte: Autoria própria, 2025.

O exemplo acima é uma função básica em Python chamada *ImparOuPar* que determina se um número fornecido pelo usuário é par ou ímpar. Abaixo, uma breve explicação sobre o código:

- Linha 1: Declara uma função com o nome “*ImparOuPar*”, que organiza a lógica responsável por verificar a paridade do número.
- Linha 2: Declara uma variável que armazena o número informado pelo usuário
- Linhas 4 a 7: Utiliza uma condição de se/senão. Onde caso o número seja divisível por 2 (ou seja, o resto da divisão é igual a 0), a mensagem exibida será “O número é par”. Caso contrário, será exibida a mensagem “O número é ímpar”.
- Linha 9: A função *ImparOuPar()* é chamada, executando a lógica para identificar e exibir a paridade do número inserido.

Figura 4 - Resultado do código em Python.

```
Digite um número: 5  
O número é ímpar.
```

Fonte: Autoria própria, 2025.

2.2.5.1 Package Installer for Python (PIP)

Como evidenciado na documentação oficial de empacotamento Python (2025), o *PIP* é o gerenciador de pacotes nativo do Python ele faz instalação e remoção de

bibliotecas, sendo o gerenciador mais usado por vir pré-instalado nas versões a partir do (Python 3.4).

Abaixo, como o comando PIP é utilizado para a instalação da biblioteca Numpy:

Figura 5 - Exemplo de código de instalação em Python.

```
C:\Users\prast\Etec3º> pip install NumPy
```

Fonte: Autoria própria, 2025.

2.2.6 Open Search Computer Vision Library (OpenCV)

Como evidenciado por Barelli (2018), a *OpenCV* é uma biblioteca multiplataforma de código aberto criada pela Intel no ano de 2000. A biblioteca foi desenvolvida com o objetivo de tornar mais simples o contato do programador à visão computacional, tendo funções otimizadas, conforme apontado por Marengoni e Stringhini (2010).

Figura 6 - Exemplo de instalação usando o PIP.

```
C:\Users\prast\Etec3º> pip install opencv-python
```

Fonte: Autoria própria, 2025.

Acima temos a demonstração de como é feita a instalação da biblioteca OpenCV em um projeto Python.

Figura 7 - Exemplo de código usando o OpenCV.

```
1  import cv2
2
3  image = cv2.imread('ExemploOpenCV.jpg')
4
5  if image is None:
6      print("Erro: Não foi possível carregar a imagem.")
7      exit()
8
9  gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
10
11 edges = cv2.Canny(gray_image, 100, 200)
12
13 cv2.imwrite('gray_image.jpg', gray_image)
14 cv2.imwrite('edges_image.jpg', edges)
15
16 cv2.imshow('Imagem em Tons de Cinza', gray_image)
17 cv2.imshow('Bordas Detectadas', edges)
18 cv2.waitKey(0)
19 cv2.destroyAllWindows()
```

Fonte: Autoria própria, 2025.

O exemplo acima demonstra um código simples utilizando a biblioteca do OpenCV, na linguagem de programação em Python, que retorna duas imagens: uma que deixará a imagem toda em tons de cinza e outra que mostra apenas as bordas da imagem:

- Linha 1: Importação da biblioteca OpenCV, possibilitando a utilização de suas funcionalidades no sistema.
- Linha 3: Carrega a imagem *ExemploOpenCV.jpg* do diretório atual e armazena-a na variável *image* para processamento.
- Linhas 5 a 7: Verifica se a imagem foi aberta corretamente; se não, mostra um erro e o programa para.
- Linha 9: Converte a imagem colorida carregada do espaço de cores BGR para escala de cinza, facilitando a aplicação de algoritmos de detecção.
- Linha 11: Aplica o algoritmo de detecção de bordas *Canny* na imagem em tons de cinza, utilizando os limiares 100 e 200, e armazena o resultado na variável *edges*.
- Linha 13: Salva a imagem convertida em tons de cinza como um arquivo *gray_image.jpg* no diretório atual.
- Linha 14: Salva a imagem com as bordas detectadas como um arquivo *edges_image.jpg* no diretório atual.

- Linha 16: Exibe uma janela com o título "Imagem em Tons de Cinza" contendo a imagem convertida.
- Linha 17: Exibe uma janela com o título "Bordas Detectadas" contendo a imagem com as bordas detectadas pelo algoritmo *Canny*.
- Linha 18: Aguarda indefinidamente até que uma tecla seja pressionada, permitindo que o usuário visualize as imagens.
- Linha 19: Fecha todas as janelas criadas pelo OpenCV, liberando os recursos utilizados pelo programa.

Figura 8 - Imagem original antes do OpenCV.



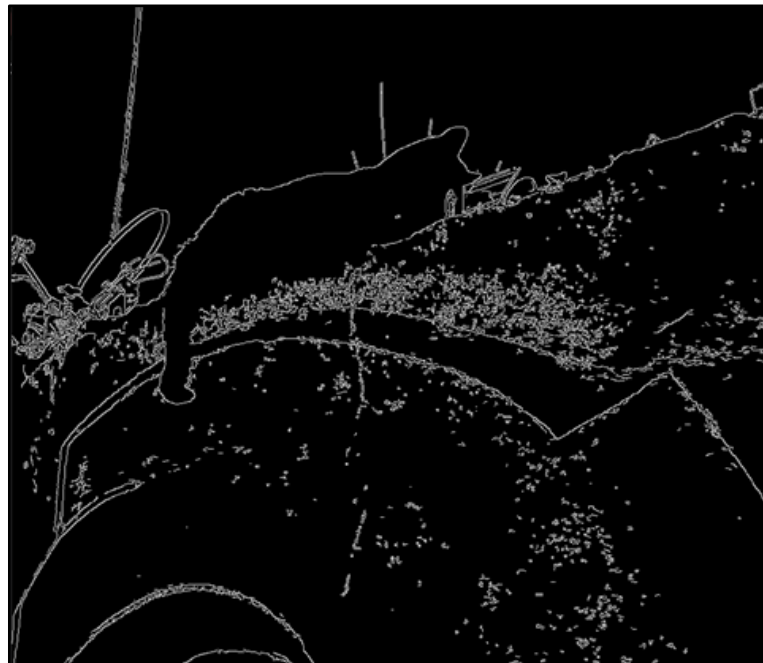
Fonte: Autoria própria, 2025.

Figura 9 - Resultado do código para conversão em preto e branco com OpenCV.



Fonte: Autoria própria, 2025.

Figura 10 - Resultado do código para a detecção de bordas com OpenCV.

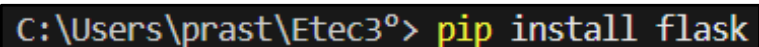


Fonte: Autoria própria, 2025.

2.2.7 Flask

Consoante a Pereira (2018), *Flask* é um micro framework com uma excelente ferramenta de APIs *REST* (*Representational State Transfer*), usada para comunicação entre sistemas, é amplamente usada para a transmissão de informação de um aplicativo a uma base de dados externa.

Figura 11 - Resultado do código em Python.



```
C:\Users\prast\Etec3º> pip install flask
```

Fonte: Autoria própria, 2025.

2.2.8 C++

De acordo com Horstmann (2008), o C++ foi uma linguagem desenvolvida sobre C, tendo como diferencial características para Orientação a Objeto, isto é, um estilo de programação para modelagem de objetos do mundo real.

A programação em C++, além de ser usada para orientar objetos, pode ser utilizada como linguagem estruturada quando se deseja trabalhar com algoritmos e estrutura de dados. (AGUILAR, 2011).

Horstmann (2008) destaca que a linguagem C++ é de uso geral, afirmando que pode ser usada para automatizar qualquer tarefa que o desenvolvedor queira.

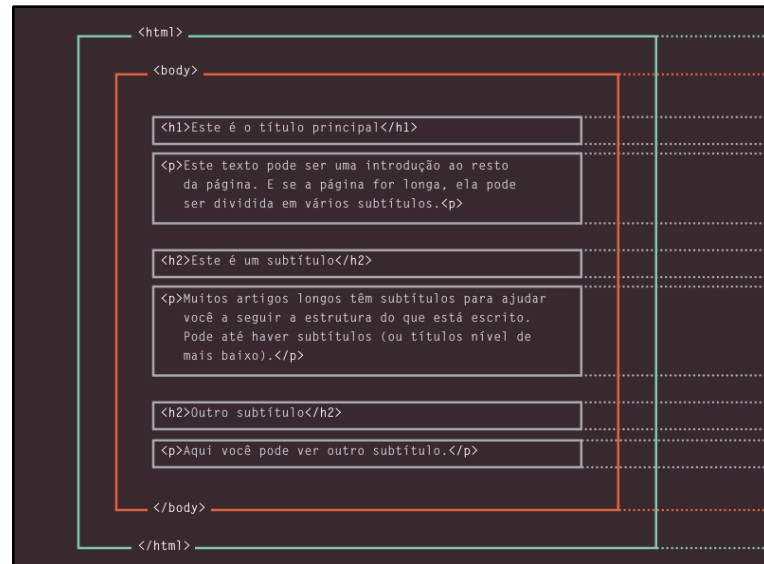
2.2.9 HyperText Markup Language (HTML)

Segundo os autores Eis e Ferreira (2012), a marcação de texto em HTML é a parte mais importante na construção de um website, pois nele há a informação que precisa ser transmitida ao usuário. Em outras palavras, mesmo que o endereço web não carregue a formatação dos elementos presentes – como, por exemplo, as imagens do site –, o texto será apresentado, fornecendo o conteúdo necessário para o entendimento do leitor. Portanto, o HTML é essencial para a criação de aplicações web.

Um código em HTML é formado por etiquetas, chamadas tags, que enunciam a estrutura de cada texto presente na página. Cada tag tem sua sigla e essa sigla fica entre o símbolo de ‘menor que’ (<) e ‘maior que’ (>). É importante mencionar que, para cada função, são necessárias duas tags: uma para a abertura e outra para o

fechamento. Para fechar uma tag basta adicionar a 'barra' após o 'menor que' (</). (DUCKETT, 2015).

Figura 12 - Exemplo de tags de abertura e de fechamento.



Fonte: Duckett, 2015.

Em primeiro lugar, vemos o início do código com a tag `<html>`. Todo o conteúdo que estiver entre as tags `<html>` e `</html>` estão indicados como código HTML. Entre as tags `<body>` e `</body>` estará presente tudo o que for relacionado ao corpo do texto e que for visível na janela do navegador. Os títulos sempre serão representados pelas tags `<h1>` e `<h2>`, com seus respectivos fechamentos, já os parágrafos pelas tags `<p>` e `</p>`, mostrando o início e o fim de cada parágrafo.

Figura 13 - Exemplo de estrutura HTML.

```

1  <!DOCTYPE html>
2
3  <html>
4    <head>
5      <meta charset="UTF-8">
6      <title>Meus Filmes Favoritos</title>
7    </head>
8
9    <body>
10     <h1>Meus Filmes Favoritos</h1>
11     <div class="filme">
12       <h2>Interestelar</h2>
13       <p>Gênero: Ficção científica &nbsp; Ano: 2014</p>
14       <p>
15         Um dos filmes que mais me fazem refletir sobre tempo,
16         espaço e os limites do amor.
17       </p>
18     </div>
19
20     <div class="links">
21       <p>
22         Voltar para: <a href="https://www.google.com">Google</a>
23         Ou ir para: <a href="https://www.imdb.com">IMDb</a>.
24       </p>
25     </div>
26   </body>
27 </html>

```

Fonte: Autoria própria, 2025.

Na imagem acima é possível observar os elementos anteriormente explicados e analisados, porém, existem outras tags que não foram vistas antes. São essas:

- **Doctype:** ‘<!DOCTYPE html>’ é uma tag independente que informa ao navegador qual linguagem de programação está sendo usada. Não precisa de fechamento.
- **Head:** ‘<head></head>’ indica o cabeçalho local onde serão colocadas informações como o título da página.
- **Meta charset:** ‘<meta charset = “UTF-8”>’ ajusta a codificação de caracteres em HTML do site.
- **Title:** ‘<title></title>’ define o título da sua página web.
- **Div:** ‘<div></div>’ representa uma divisão (ou um bloco) de um conteúdo da página.
- **A:** ‘’ serve para adicionar um hiperlink ao texto que se encontra entre as tags.
- **NBSP:** ‘ ’ (do inglês ‘non-breaking space’), embora não seja uma tag, é um elemento que adiciona um espaçamento entre palavras e/ou tags, já que o espaçamento comum (“ ”) pode ser desconsiderado quando fora de tags.

Após montarmos o código no documento em HTML, podemos acessar a página criada:

Figura 14 - Exemplo de estrutura HTML na web.



Fonte: Autoria própria, 2025.

2.2.10 Cascading Style Sheets (CSS)

Cascading Style Sheets, em português Folha de Estilo em Cascata, trata-se de uma estilização para websites onde apenas a linguagem de programação HTML foi aplicada. As folhas de estilo, comumente chamadas de CSS, são responsáveis pelas regras de formatação e apresentação dos elementos presentes na aplicação, como aponta Silva (2011).

De acordo com os autores Eis e Ferreira (2012), o CSS tem a finalidade de causar um impacto visual ao usuário, sendo o principal agente na formatação e estilização de páginas web. Desse modo, é possível acessar websites em diferentes leitores de tela – como celulares, computadores, tablets, entre outros –, sem que as proporções sejam negativamente afetadas, já que se adaptam ao tamanho do *display*. Diante de tamanhas evoluções nas aplicações, o CSS tem sido cada vez mais usado, sendo um ótimo recurso para desenvolvedores *front-end*.

O CSS possui uma sintaxe simples e de fácil entendimento:

Figura 15 - Exemplo descritivo de sintaxe CSS.

```

seletor {
    propriedade: valor;
}

```

Fonte: Eis e Ferreira, 2012.

O seletor faz referência à tag que será afetada, a propriedade define o elemento será alterado na tag e o valor é o número ou palavra que realizará a mudança no seletor. Após colocar o valor, é necessário terminar com ponto e vírgula (;) para finalizar a sintaxe.

Colocando a sintaxe em prática, é possível visualizar com mais clareza:

Figura 16 - Exemplo de sintaxe CSS.

```

2  body {
3      font-family: Arial, sans-serif;
4      background-color: #f4f4f4;
5      margin: 50px;
6      padding: 20px;
7      color: #333;
8  }

```

Fonte: Autoria própria, 2025.

Analisando o código acima encontramos o seletor *body*, as propriedades e seus valores:

- *Font-family*: define a fonte do texto, sendo nesse caso a fonte Arial como primeira opção e Sans-Serif como segunda opção.
- *Background-color*: modifica a cor de fundo; a cor pode ser colocada em hexadecimal – como no exemplo acima –, em RGB, ou por seu nome, se a cor já estiver pré-definida no editor de texto.
- *Margin*: delimita a margem externa entre as bordas e os outros elementos. Pode ser exclusivamente para apenas uma direção, sendo embaixo, em cima, esquerda ou direita (exemplo: *margin-bottom*, *margin-top*, *margin-left*, *margin-right*).

- *Padding*: cria um espaçamento interno entre as bordas do *body* e o conteúdo. Pode ser direcionada para baixo, para cima, esquerda ou direita (exemplo: *padding-bottom*, *padding-top*, *padding-left*, *padding-right*).
- *Color*: define a cor do texto.

Para ilustrar, usaremos o código base em HTML feito no tópico anterior e aplicar uma estilização baseada em CSS. Sendo assim, é necessário incrementar no *head* a tag *link*, para que as alterações realizadas em CSS afetem o texto em HTML.

Figura 17 - Exemplo com a tag *link* CSS no código HTML.

```
<head>  
  <meta charset="UTF-8">  
  <title>Meus Filmes Favoritos</title>  
  <link rel="stylesheet" href="estilo.css">  
</head>
```

Fonte: Autoria própria, 2025.

Existem várias maneiras de criar estilos de páginas com CSS e seus elementos básicos, trazendo mais atenção aos objetos certos, de maneira simples, assim como pode-se observar na figura abaixo:

Figura 18 - Código para estilização do documento HTML.

```
body {
  font-family: Arial, sans-serif;
  background-color: #f4f4f4;
  margin: 50px;
  padding: 20px;
  color: #333;
}

h1 {
  text-align: center;
  color: #2c3e50;
  margin-bottom: 30px;
}

.filme {
  background-color: #fff;
  border: 1px solid #ddd;
  border-radius: 5px;
  padding: 15px;
  margin-bottom: 20px;
  box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}

h2 {
  color: #2980b9;
  margin-top: 0;
}

p {
  line-height: 1.6;
}

a {
  color: #e74c3c;
  text-decoration: none;
}

a:hover {
  text-decoration: underline;
}

.links {
  text-align: center;
  margin-top: 30px;
}
```

Fonte: Autoria própria, 2025.

Elementos não explicados anteriormente:

- *Text-align*: alinha o texto na direção indicada no valor.
- *Border*: modifica a borda do elemento.
- *Border-radius*: altera especificamente a angulação das pontas para mais ou menos redondas.
- *Box-shadow*: cria um sombreamento para o elemento, podendo modificar a direção da sombra.
- *Line-height*: define o espaçamento vertical entre as linhas.
- *Text-decoration*: para adicionar ou retirar qualquer tipo de decoração no texto; por exemplo o *underline*: com o comando *text-decoration* é possível colocá-lo ou retirá-lo do texto.

- *Hover*: ‘:hover’ indica que, quando o mouse passar por cima do elemento, algo acontecerá. No caso, o texto será decorado com *underline*.

Por fim, o documento HTML anteriormente sem nenhum elemento decorativo foi alterado com código CSS para uma apresentação mais agradável.

Figura 19 - Aplicação HTML alterada com código CSS.



Fonte: Autoria própria, 2025.

2.2.11 JavaScript (JS)

JavaScript trata-se de uma linguagem responsável por controlar os comportamentos dinâmicos e interativos de elementos. É frequentemente empregada em aplicações web, tanto no desenvolvimento da interface – o que o usuário visualiza e interage –, como na estrutura lógica, processando dados e comunicando-se com os servidores, esclarece Silva (2010).

Segundo Lepsen (2022), essa linguagem faz parte de um trio fundamental para o desenvolvimento de aplicações web, em conjunto com HTML, uma linguagem de marcação cuja finalidade é estruturar as páginas, e CSS, encarregada pela estilização da estrutura da página.

Flanagan (2020) aponta que a linguagem teve sua primeira versão lançada em 1995, sendo desenvolvida pela *Netscape Communications Corporation* e a *Sun Microsystems*. Foi adotada pela ECMA, uma organização internacional de padronização, em 1997 para evitar problemas de incompatibilidade, tornando-se uma linguagem essencial e consagrada entre os desenvolvedores.

Figura 20 - Exemplo de código em JavaScript.

```
1  function soma(){  
2      x = 5;  
3      y = 6;  
4  
5      Resultado = x + y;  
6      return Resultado;  
7  }  
8  console.log(`O resultado é: ${Resultado()}`);
```

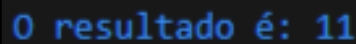
Fonte: Autoria própria, 2025.

É apresentado na imagem acima um exemplo de código em JavaScript. Explicação de cada linha:

- Linha 1: É declarada uma função chamada soma, sem parâmetros. Parâmetros são valores definidos para uma entrada, permitindo que o valor possa ser reutilizável, mas com diferentes dados.
- Linha 2 e 3: São declaradas duas variáveis com valores inteiros.
- Linha 5: É armazenado a soma das duas variáveis na variável Resultado.
- Linha 6: Retorna o Resultado calculado, para o local onde a função foi chamada.
- Linha 8: É definido que a apresentação do resultado deve ser feita no console seguida do valor retornado pela função.

Apresentação do resultado:

Figura 21 - Resultado exibido no console.



```
O resultado é: 11
```

Fonte: Autoria própria, 2025.

2.2.12 React Native

De acordo com Falcão (2022), o *React Native* é uma biblioteca que foi criada pelo Facebook, utilizando dos mesmos benefícios do React, a biblioteca JavaScript que o antecede, porém, tendo como objetivo de desenvolver aplicativos móveis.

O React Native foi criado para que qualquer um que quisesse desenvolver um aplicativo, sentisse o mesmo conforto que sentiriam ao desenvolver para web com React.Js, conforme Silva e Sousa (2019).

Como descrito por Escudelario e Pinho (2021), o React Native agrupa o que há de mais moderno no desenvolvimento front-end em uma série de ferramentas que possibilitam o melhor ambiente para aplicações mobile híbridas, isto é, tanto Android quanto IOS. O diferencial do React Native é que todo código que foi desenvolvido nele, é convertido para a linguagem nativa do sistema operacional do aparelho em que está sendo executado.

2.2.13 Node JS

Parafraseando Rubens (2017), houve uma época de transformação na área do desenvolvimento de sistemas, onde desenvolver uma aplicação web, ou um software, passou a ser muito além de escolher a linguagem mais fácil, sendo nesse cenário que o *Node.js* se moldou, já que ele é uma ferramenta versátil e que, constante, surpreende desenvolvedores com o seu avanço.

Como consoante a Moraes (2018), o Node.js é uma plataforma de arquitetura orientada a eventos, sendo *non-blocking thread*, isto é, não-bloqueante, o que torna o desenvolvimento muito mais leve e eficiente.

O sistema Node tem uma boa performance quando se trata da utilização eficiente da memória e do poder de processamento dos servidores, despreocupando aqueles que o utilizarem, pois terão seus resultados de forma rápida, assim como afirma Pereira (2014).

2.2.13.1 NPM

O NPM é um gerenciador de pacotes, que vem instalado juntamente do Node.js, permitindo o reuso de módulos feitos por outros desenvolvedores, trabalhando com a importação dos mesmos, de acordo com Rubens (2017).

2.2.14 Banco de Dados

Banco de dados refere-se a um conjunto de informações que servem para o funcionamento de um determinado sistema. Esses dados são manipulados por SGBD, que se trata de um sistema cuja função é realizar a gestão desse conjunto de dados de forma segura e estruturada, provendo funcionalidades de acesso, transação,

extração, entre outras, de forma eficiente. O banco de dados torna-se essencial para o funcionamento de aplicações a partir do momento em que facilita o armazenamento de grandes coleções de dados. Está presente em diversas áreas, como sistemas bancários, redes sociais, plataformas de ensino entre outras áreas. (AZURE).

2.2.15 Firebase

Existem diversas ferramentas especializadas para suprir diferentes necessidades e objetivos no desenvolvimento de aplicações, dentre elas, o *Firebase* se destaca sendo amplamente utilizado por desenvolvedores. O Firebase é uma plataforma web desenvolvida pela Google que oferece diversos recursos voltados para o desenvolvimento do *back-end* de aplicações mobile e web. Conhecido como *Backend-as-a-Services*, ele disponibiliza as configurações do back-end já prontas, dispensando a necessidade de configurar servidores, assim, acelerando o processo de desenvolvimento da aplicação. (FIREBASE, 2023).

Figura 22 – Logotipo Firebase.



Fonte: Firebase, 2025.

2.2.15.1 Cloud Firestore

Conforme mencionado anteriormente, o Firebase fornece diferentes tipos de serviços, desde autenticação de usuários até a hospedagem de projetos, armazenamento de arquivos e monitoramento de falhas em tempo real. Um desses serviços é o *Cloud Firestore* que será o encarregado por auxiliar na gestão do banco de dados. (GOOGLE CLOUD).

O Firestore é um serviço voltado para a criação e gerenciamento de bancos de dados no SQL, que consiste em um banco de dados não-relacional. Foi elaborado com o objetivo de ser compatível com dispositivos móveis, aplicações web e mobile e servidores, proporcionando maior acessibilidade à aplicação, segurança aprimorada, escalabilidade e eficiência no processamento dos dados. Entre os principais

benefícios, está o fato de que os dados são atualizados em tempo real. Mesmo sem conexão com a Internet, eles são guardados localmente e, depois que a conexão é restabelecida, são sincronizados de forma automática. (FIREBASE).

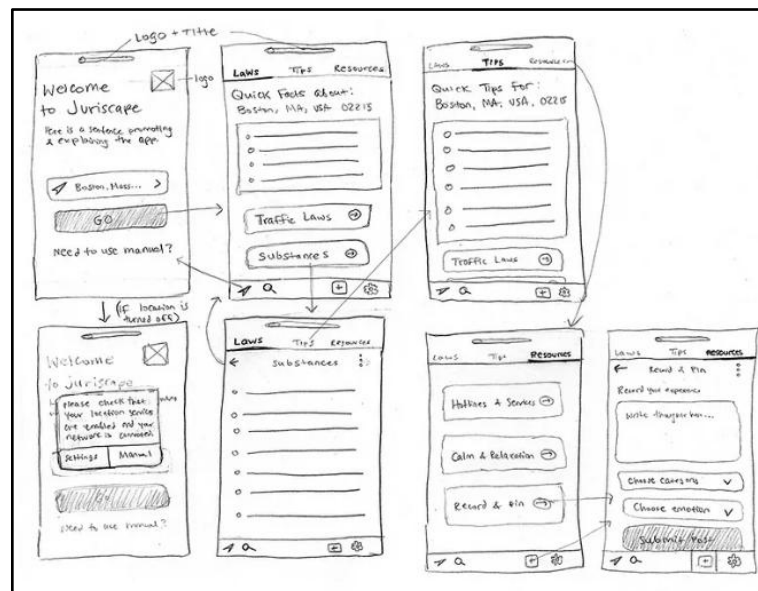
2.2.16 Wireframe

No contexto de design de interfaces web, o *wireframe* é essencial para a construção e visualização de uma página e deve ser feito antes de qualquer documento de codificação. Wireframes constroem o mapa do site e têm a função de um esqueleto, ele molda o resultado da interface. (LUCIDCHART, 2023).

Por conseguinte, existem três formatos de wireframe que evidenciam a progressão de um projeto: wireframe de baixa, média e alta fidelidade.

Logo, o design da interface deve ser inicialmente feito de forma simples, como um primeiro protótipo – como um rabisco em um papel –, seguido de um design digitalizado e com elementos mais organizados, ainda sem cores ou conteúdo (BECKER via ORGÂNICA DIGITAL, 2022), para, então, o resultado: trazendo cores, identidade visual e imagens.

Figura 23 - Exemplo de wireframe de baixa fidelidade.



Fonte: Medium, 2021.

Dentre as variadas plataformas para desenvolvimento de wireframes, neste projeto utilizamos o website da ferramenta *Figma*, que fornece diversas funcionalidades com o objetivo de tornar a criação de designs mais acessível. (EBAC ONLINE, 2023).

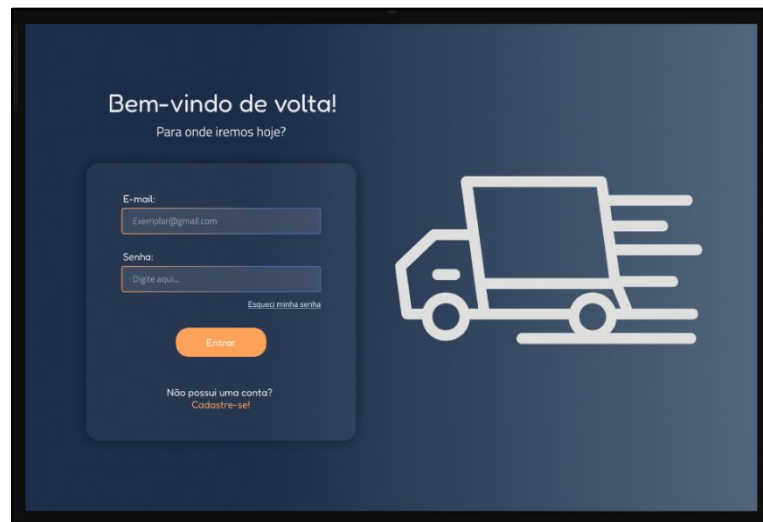
Para exemplificar, a imagem abaixo mostra como um wireframe de média fidelidade pode ser criado com facilidade nessa plataforma:

Figura 24 - Wireframe de média fidelidade feito no Figma.



Fonte: Autoria própria, 2025.

Figura 225 - Wireframe de alta fidelidade feito no Figma.



Fonte: Autoria própria, 2025.

2.2.17 Unified Modeling Language (UML)

Guedes (2018) afirma que a UML é uma linguagem que possibilita a modelagem de sistemas, ou seja, especifica de forma gráfica as características, comportamentos e estruturas do sistema. A maneira como os projetos eram desenvolvidos antes da UML não era eficaz pois havia uma falta de padronização, o que gerava grandes problemas.

Com o intuito de ajudar engenheiros de software a modelar seu sistema de forma padronizada, foi criada a UML.

O desenvolvimento da UML resultou da união dos métodos de modelagem orientada a objetos: o método *Booch*, as técnicas de modelagem de objetos de *Ivar Jacobson* e a engenharia de software orientada a objetos de *James Rumbaugh*, três figuras importantes da área de engenharia de software. Sua primeira versão oficial foi lançada em 1996, contudo, foi adotada como padrão oficial em 1997 pela OMG (Object Management Group). (BOOCH, 2012).

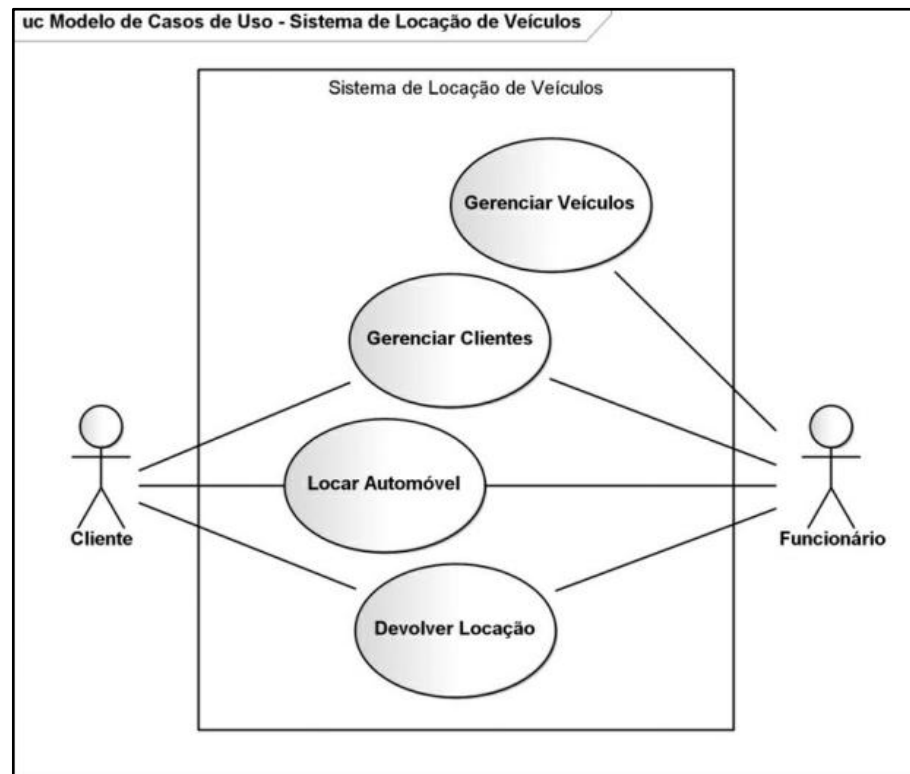
A UML possui uma gama de diagramas, com diferentes características, que visam tornar a modelagem de sistemas mais eficiente e facilitar a compreensão do sistema para o leitor.

2.2.17.1 Diagrama de Caso de Uso

Este diagrama tem como objetivo deixar claro e compreensível as funcionalidades e o fluxo do sistema ao todo. São fundamentais na modelagem de sistemas, já que é com ele que se especificam os requisitos funcionais e como e com quem eles se relacionam. Ele é comumente usado como base para outros diagramas, de acordo com Guedes (2018).

Para desenvolver um diagrama de caso de uso, os elementos utilizados são atores e Casos de uso. Os atores podem indicar desde usuários a outros sistemas externos. Já os casos de uso referem-se a um conjunto de ações que ocorrem de acordo com a interação do usuário, descreve Booch (2012).

Figura 26 - Exemplo de Diagrama de Casos de Uso.



Fonte: Guedes, 2018.

Descrição do diagrama:

Esse diagrama representa as funcionalidades de um Sistema de Locação de Veículos. Observamos que estão presentes no diagrama os atores (Cliente e Funcionários) e os Casos de Uso (Gerenciar Veículos, Gerenciar Clientes, Locar Automóveis e Devolver Locação). O ator “Cliente” é o responsável por realizar as operações de alugar os veículos e a devolução de veículos. Já o ator “Funcionário” é o encarregado de gerenciar as operações administrativas do sistema, como o gerenciamento de veículos e clientes, em conjunto a isso, pode dar assistência nas locações e devoluções.

Para o autor Guedes (2018), a documentação de casos de uso nada mais é do que um texto explicativo de como funcionará a execução de um determinado caso de uso; geralmente é autoexplicativo, para que o leitor tenha uma interpretação descomplicada.

Figura 27 - Exemplo de Documentação de Caso de Uso.

Nome do Caso de Uso		UC06 – Emitir Saldo	
Ator Principal	Cliente		
Atores Secundários			
Resumo	Descreve os passos necessários para um cliente obter o saldo referente a uma determinada conta		
Pré-condições			
Pós-condições			
		Cenário Principal	
Ações do Ator	Ações do Sistema		
1. Informar o número da conta	2. Verificar a existência da conta		
	3. Solicitar a senha da conta		
4. Informar a senha	5. Verificar se a senha está correta		
	6. Emitir o saldo		
Restrições/Validações	1. A conta precisa existir e estar ativa		
	2. A senha precisa estar correta		
		Cenário de Exceção I – Conta não encontrada	
Ações do Ator	Ações do Sistema		
	1. Comunicar ao cliente que o número da conta informada não foi encontrado		
		Cenário de Exceção II – Senha inválida	
Ações do Ator	Ações do Sistema		
	1. Comunicar ao cliente que a senha fornecida não combina com a senha da conta		

Fonte: Guedes, 2018.

2.2.17.2 Diagrama de Sequência

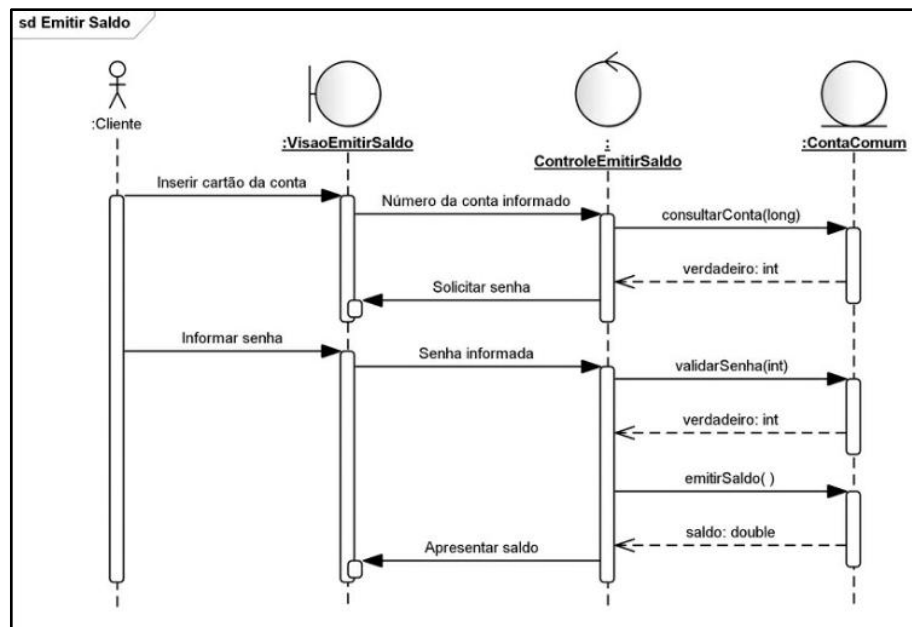
A representação das interações que ocorrem quando um caso de uso específico é posto em execução, é essencial em diversos projetos. O diagrama de sequência se mostra muito eficiente nesse cenário, sendo um diagrama de comportamento comumente utilizado na modelagem de sistemas, conforme aponta Fowler (2005).

Como descrito por Guedes (2018), em diagramas de sequência estão presentes os elementos atores, *lifetimes* e mensagens ou estímulos. Os atores, um conceito que já foi explicado, fazem ações reais em diagramas de comportamentos, ou seja, solicitam ações e iniciam processos. A *lifetime*, ou linha de vida, é uma linha vertical tracejada que segue abaixo de um objeto ou ator, que será usada para indicar quanto tempo ele está ativo na interação. Já as mensagens podem representar a chamada de um método, ou a troca de informações entre atores.

Booch (2012) destaca que esse diagrama é baseado em casos de uso, determinando a sequência de ações e mensagens trocadas entre os elementos no sistema, mas

com ênfase na ordenação temporal das mensagens. Simplificando, vamos desmembrar um caso de uso:

Figura 28 - Exemplo de diagrama de Sequência.



Fonte: Guedes, 2018.

Descrição do diagrama:

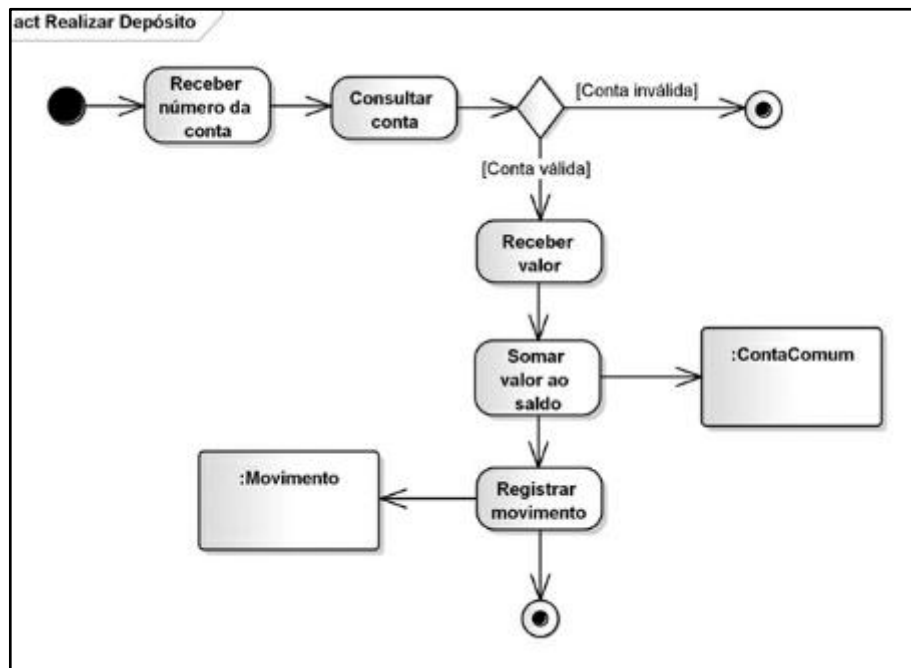
Pode-se observar que, nesse diagrama, há o ator “Cliente”, que será quem iniciará o caso de uso “Emitir Saldo”. Ele começa comunicando-se com o objeto “VisaoEmitirSaldo”, que será o encarregado de trocar informações com o objeto “ControleEmitirSaldo”, que fará o mesmo com o objeto “ContaComum”, para realizar a verificação dos dados informados.

2.2.17.3 Diagrama de Atividade

Sob a perspectiva de Booch (2012), os diagramas de atividade são elaborados para retratar a ordem das etapas da execução de uma funcionalidade, visando atingir um objetivo.

Em contraste com os outros diagramas, não há atores representados aqui. Este diagrama é constituído por nó inicial, nó final, nó de ação, setas, nó de condição e objetos. O nó inicial e o nó final são usados para determinar o início do processo e o seu término. Nó de ação é uma pequena ação que deve ser executada dentro da atividade. As setas determinam a sequência das atividades. Nó condicional refere-se ao ponto de decisão de um fluxo. Conforme Guedes (2018).

Figura 29 – Exemplo de diagrama de Atividade.



Fonte: Guedes, 2018.

Descrição do diagrama:

Para o processo de realizar um depósito, o sistema precisa fazer uma verificação da conta, contudo quando entra em um nó de decisão, ele explora cenários alternativos. Se o cenário for válido, ele prossegue com as outras atividades. Percebe-se que a atividade “Somar valor ao saldo” se comunica com o objeto “ContaComum”, que é uma classe. O mesmo acontece com a atividade “Registrar movimento”.

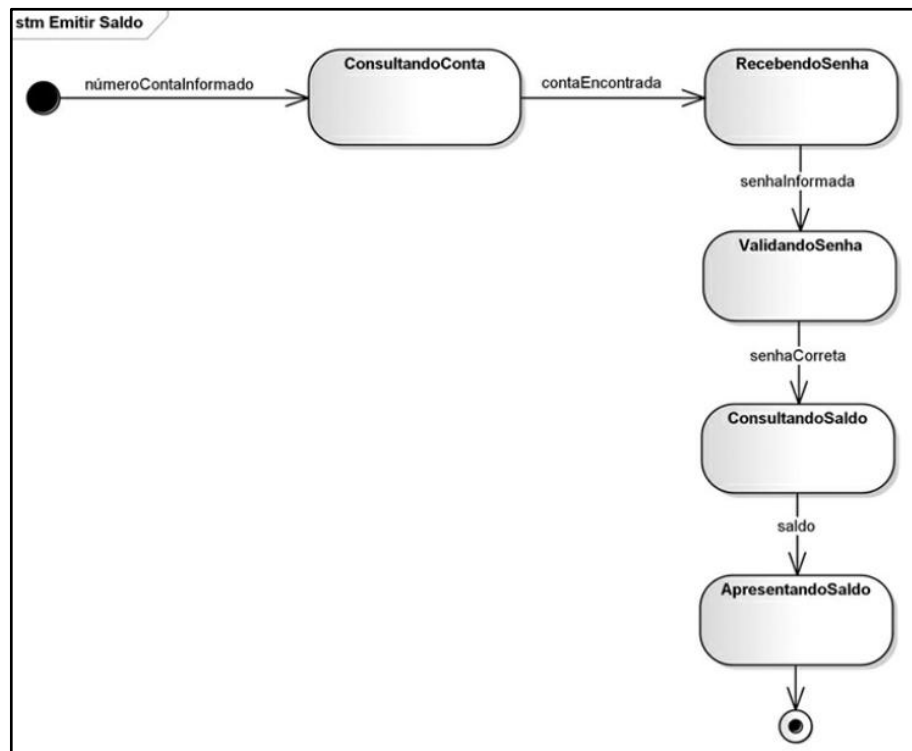
2.2.17.4 Diagrama de Máquina-Estado

Com base em Fowler (2005), este diagrama tem como finalidade descrever o comportamento de um objeto, mais especificamente, ele indica a alteração do estado de um objeto, apontando as condições e eventos para tal acontecimento.

Os componentes utilizados neste diagrama são estados, transições, estado inicial e estado final. Pode-se entender que o estado se refere a situação atual do objeto. Transição é a descrição do evento ou condição que o objeto realiza. Já o estado inicial, diz respeito ao início do ciclo de vida do objeto, e para indicar o final usamos o estado final. (GUEDES, 2018).

Booch (2012) destaca que a máquina de estado modela o ciclo de vida de um único objeto, enquanto os de interação interagem e se relacionam com um ou mais objetos.

Figura 30 – Exemplo de diagrama de Máquina-Estado.



Fonte: Guedes, 2018.

3 DESENVOLVIMENTO

REFERENCIAL