



UNIVERSIDADE ESTADUAL DO TOCANTINS  
CÂMPUS DE PALMAS  
CURSO DE SISTEMAS DE INFORMAÇÃO

**DESENVOLVIMENTO DE API PARA A VIABILIZAÇÃO DE  
APLICAÇÕES DE GERENCIAMENTO DE FREQUENCIA ESCOLAR  
QUE UTILIZAM QR CODE**

HENRIQUE AUGUSTO BECKMANN

Palmas - TO

2023



UNIVERSIDADE ESTADUAL DO TOCANTINS  
CÂMPUS DE PALMAS  
CURSO DE SISTEMAS DE INFORMAÇÃO

**DESENVOLVIMENTO DE API PARA A VIABILIZAÇÃO DE  
APLICAÇÕES DE GERENCIAMENTO DE FREQUENCIA ESCOLAR  
QUE UTILIZAM QR CODE**

HENRIQUE AUGUSTO BECKMANN

Trabalho de Conclusão de Curso apresentado ao  
Curso de Sistemas de Informação da Universidade  
Estadual do Tocantins - UNITINS como parte dos  
requisitos para a obtenção do grau de Bacharel em  
Sistemas de Informação.

Palmas - TO

2023



## **CURSO DE SISTEMAS DE INFORMAÇÃO**

# **DESENVOLVIMENTO DE API PARA A VIABILIZAÇÃO DE APLICAÇÕES DE GERENCIAMENTO DE FREQUENCIA ESCOLAR QUE UTILIZAM QR CODE**

**HENRIQUE AUGUSTO BECKMANN**

Trabalho de Conclusão de Curso apresentado ao Curso de Sistemas de Informação da Universidade Estadual do Tocantins - UNITINS como parte dos requisitos para a obtenção do grau de Bacharel em Sistemas de Informação.

---

**Me. Silvano Maneck Malfatti**  
Orientador

---

**Victor Eduardo de Solsa Silva**  
Examinador

---

**Tayse Virgulino Ribeiro**  
Examinador

Palmas - TO

2023

**Dados Internacionais de Catalogação na Publicação  
(CIP) Sistema de Bibliotecas da Universidade Estadual  
do Tocantins**

---

B397d                      BECKMANN, Henrique Augusto  
Desenvolvimento de API para a viabilização de  
aplicações de gerenciamento de frequência escolar  
que utilizam Qr code. Henrique Augusto Beckmann. -  
Palmas, TO, 2023

Monografia Graduação - Universidade Estadual do  
Tocantins – Câmpus Universitário de Palmas - Curso de  
Sistemas de Informação, 2023.

Orientador: Silvano Maneck Malfatti

1. Frequência. 2. Qr Code. 3. Gerenciamento. 4.  
API.

---

**CDD 610.7**

TODOS OS DIREITOS RESERVADOS – A reprodução total ou parcial, de qualquer forma ou por qualquer meio deste documento é autorizado desde que citada a fonte. A violação dos direitos do autor (Lei nº 9.610/98) é crime estabelecido pelo artigo 184 do Código Penal.

Elaborado pelo sistema de geração automática de ficha catalográfica da UNITINS com os dados fornecidos pelo(a) autor(a).

**ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO DO CURSO DE SISTEMAS DE INFORMAÇÃO DA FUNDAÇÃO UNIVERSIDADE ESTADUAL DO TOCANTINS - UNITINS**

Aos **21** dias do mês de **Junho** de **2023**, reuniu-se na Fundação Universidade Estadual do Tocantins, Câmpus Palmas, Bloco B, às **08:20 horas**, sob a Coordenação do Professor **Silvano Maneck Malfatti** a banca examinadora de Trabalho de Conclusão de Curso em Sistemas de Informação, composta pelos examinadores Professor **Silvano Maneck Malfatti** (Orientador), Professor **Victor Eduardo de Sousa Silva** e Professora **Tayse Virgulino Ribeiro**, para avaliação da defesa do trabalho intitulado “**Desenvolvimento de API para a Viabilização de Aplicações de Gerenciamento de Frequência Escolar com QRCode**” do acadêmico **Henrique Augusto Beckmann** como requisito para aprovação na disciplina Trabalho de Conclusão de Curso II (TCC II). Após exposição do trabalho realizado pelo acadêmico e arguição pelos Examinadores da banca, em conformidade com o disposto no Regulamento de Trabalho de Conclusão de Curso em Sistemas de Informação, a banca atribuiu a pontuação: 8,5 .

Sendo, portanto, o Acadêmico: (x) Aprovado ( ) Reprovado

Assinam esta Ata:



Documento assinado digitalmente

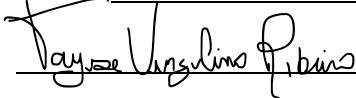
SILVANO MANECK Malfatti

Data: 03/07/2023 15:08:31-0300

Verifique em <https://validar.iti.gov.br>

Professor Orientador: \_\_\_\_\_

Examinador: \_\_\_\_\_



Examinador: \_\_\_\_\_



Documento assinado digitalmente

VICTOR EDUARDO DE SOUSA SILVA

Data: 29/06/2023 22:00:03-0300

Verifique em <https://validar.iti.gov.br>

**Silvano Maneck Malfatti**

**Presidente da Banca Examinadora**

Coordenação do Curso de Sistemas de Informação

*Este trabalho é dedicado à minha família, que me deu apoio e forças para não desistir  
dessa jornada.*

# Agradecimentos

Agradeço a Deus pela força e perseverança que me deu para a jornada, à minha família pelo apoio e por acreditarem na minha capacidade.

*“A informática me distanciou dos livros,  
não da leitura.”*  
*Jeziel L. Carvalho*



# Resumo

O processo de realizar chamada em sala de aula é o meio que os professores utilizam para verificar se um aluno está presente ou não na sala de aula. Normalmente feito à mão, utilizando uma tabela impressa contendo os nomes de todos os alunos da turma, esse processo é muitas vezes interrompido por alunos conversando muito alto ou por precisar repetir o nome de um aluno até que ele ouça por exemplo. A não realização da chamada, entretanto, pode levar sérios problemas para o professor no caso de ele colocar todos presentes, isso faz com que caso um aluno não esteja presente e se machuque ou cometa algum crime, a instituição será responsabilizada. Este trabalho de conclusão de curso mostra a pesquisa e desenvolvimento de uma ferramenta para agilizar esse processo, através de um serviço construído em *Node.JS*, que utiliza QrCodes como forma de verificação, junto com um estudo de caso para validação do mesmo.

**Palavras-chaves:** Chamada, QrCode, Serviço, Facilitar.

# Abstract

The process of taking attendance in the classroom is the means by which teachers verify whether a student is present or not in the classroom. Usually done manually, using a printed table containing the names of all the students in the class, this process is often interrupted by students talking too loudly or by needing to repeat a student's name until they hear, for example. However, not taking attendance can lead to serious problems for the teacher if they mark everyone as present, as this means that if a student is absent and gets injured or commits a crime, the institution will be held responsible. This thesis presents the research and development of a tool to streamline this process, using a service built in Node.JS, that utilizes QR codes for verification, along with a case study for its validation.

**Key-words:** Attendance, QrCode, Service, Facilitate.

# Lista de ilustrações

Figura 1 – Processo de chamada . . . . .	18
Figura 2 – Impressão Digital . . . . .	22
Figura 3 – Reconhecimento facial . . . . .	23
Figura 4 – Diagrama de Funcionamento. . . . .	25
Figura 5 – QRCode . . . . .	26
Figura 6 – Planejamento do trabalho . . . . .	28
Figura 7 – Diagrama de Comparação . . . . .	32
Figura 8 – Exemplo do documento único . . . . .	34
Figura 9 – Exemplo da coleção institution . . . . .	35
Figura 10 – Exemplo da coleção professor . . . . .	35
Figura 11 – Exemplo da coleção group . . . . .	36
Figura 12 – Exemplo da coleção participant . . . . .	36
Figura 13 – Uso de promessas . . . . .	43
Figura 14 – Coleção de requisições no postman . . . . .	44
Figura 15 – Tela de exibição feita pelo ChatGPT . . . . .	45
Figura 16 – Fluxo de Funcionamento da API . . . . .	46
Figura 17 – Comparação entre requisitos funcionais e não funcionais. . . . .	50
Figura 18 – Diagrama de Caso de Uso do Sistema. . . . .	53
Figura 19 – Diagrama de atividade do cadastro do professor. . . . .	54
Figura 20 – Diagrama de atividade de troca de senha. . . . .	55
Figura 21 – Diagrama de atividade de login. . . . .	56
Figura 22 – Diagrama de atividade do cadastro de turma. . . . .	57
Figura 23 – Diagrama de atividade do cadastro de aluno. . . . .	58
Figura 24 – Diagrama de atividade da chamada. . . . .	59
Figura 25 – Diagrama de atividade do relatório. . . . .	60
Figura 26 – Diagrama de Navegação do aplicativo. . . . .	61
Figura 27 – Splash screen do aplicativo. . . . .	62
Figura 28 – Tela de login do aplicativo. . . . .	62
Figura 29 – Tela inicial do aplicativo. . . . .	63
Figura 30 – Tela de detalhes de uma turma. . . . .	64
Figura 31 – Tela de edição de turma. . . . .	65
Figura 32 – Modal para edição das informações de um aluno. . . . .	65
Figura 33 – Tela de perfil. . . . .	65
Figura 34 – Tela de realização de chamada. . . . .	66
Figura 35 – Gráfico da pesquisa - quantidade de turmas que o docente ministra. . . . .	72
Figura 36 – Gráfico da pesquisa – Frequência da realização da chamada. . . . .	72

Figura 37 – Gráfico da pesquisa – Repetir nome do aluno. . . . .	73
Figura 38 – Gráfico da pesquisa - Tempo médio da duração da chamada. . . . .	73
Figura 39 – Gráfico da pesquisa – Quantidade de alunos por turma. . . . .	74

# Lista de tabelas

Tabela 1 – Tabela Comparativa Entre as Aplicações. . . . .	21
Tabela 2 – Tabela Comparativa Entre as Tecnologias . . . . .	27
Tabela 3 – Exemplo de requisições básicas . . . . .	43
Tabela 4 – Dados necessários para outras coleções . . . . .	44
Tabela 5 – Lista de requisições chaves da API . . . . .	44
Tabela 6 – Validação de qualidade de código - API . . . . .	67
Tabela 7 – Validação de qualidade de código - Tela . . . . .	68

# Lista de abreviaturas e siglas

**REST** - Representational State Transfer.

**API** - Application Programming Interface.

**HTTP** - Hypertext Transfer Protocol.

**JSON** - JavaScript Object Notation.

**UML** - Unified Modeling Language.

**QRCode** - Quick-response code.

# Sumário

<b>1</b>	<b>INTRODUÇÃO</b>	<b>17</b>
<b>1.1</b>	<b>Justificativa</b>	<b>18</b>
<b>1.2</b>	<b>Objetivos</b>	<b>19</b>
1.2.1	Objetivo Geral	19
1.2.2	Objetivos Específicos	19
<b>2</b>	<b>REFERENCIAL TEÓRICO</b>	<b>20</b>
<b>2.1</b>	<b>Aplicações com propostas similares</b>	<b>20</b>
2.1.1	Toaqui	20
2.1.2	Factorial	20
2.1.3	Doity	20
2.1.4	Serpro	21
2.1.5	Comparação entre as aplicações	21
<b>2.2</b>	<b>Tecnologias a disposição</b>	<b>22</b>
2.2.1	Impressão digital	22
2.2.2	Reconhecimento Facial	23
<b>2.3</b>	<b>Tecnologias utilizadas</b>	<b>24</b>
2.3.1	JavaScript/Node.js	24
2.3.1.1	Express	24
2.3.1.2	Firebase: Firestore	24
2.3.1.3	Funcionamento	25
2.3.2	QRCode	26
<b>2.4</b>	<b>QRCode vs Reconhecimento facial vs Impressão digital</b>	<b>27</b>
<b>3</b>	<b>METODOLOGIA</b>	<b>28</b>
<b>3.1</b>	<b>Definição de escopo</b>	<b>28</b>
<b>3.2</b>	<b>Tecnologias</b>	<b>29</b>
<b>3.3</b>	<b>Ferramentas utilizadas</b>	<b>30</b>
<b>4</b>	<b>DESENVOLVIMENTO</b>	<b>31</b>
<b>4.1</b>	<b>REST API vs GRAPHQL</b>	<b>31</b>
4.1.1	Diagrama de comparação	32
<b>4.2</b>	<b>Banco de dados: Firebase Firestore</b>	<b>33</b>
4.2.1	Todos os dados em um documento	34
4.2.2	Organizando em coleções	35
4.2.2.1	institution	35

4.2.2.2	professor . . . . .	35
4.2.2.3	group . . . . .	36
4.2.2.4	participant . . . . .	36
<b>4.3</b>	<b>Requisições . . . . .</b>	<b>37</b>
4.3.1	Lógica das requisições . . . . .	37
4.3.1.1	Criação de documento . . . . .	38
4.3.1.2	Leitura de documento . . . . .	39
4.3.1.3	Edição de documento . . . . .	40
4.3.2	Requisições chave . . . . .	41
4.3.2.1	Envio de QRCode . . . . .	41
4.3.2.2	Realização de frequência . . . . .	42
4.3.3	Lista de requisições . . . . .	43
<b>4.4</b>	<b>Estudo de caso . . . . .</b>	<b>45</b>
<b>5</b>	<b>RESULTADOS . . . . .</b>	<b>46</b>
<b>5.1</b>	<b>Funcionamto da API . . . . .</b>	<b>46</b>
<b>5.2</b>	<b>Proposta de aplicação . . . . .</b>	<b>47</b>
5.2.1	Funcionamento do aplicativo . . . . .	47
5.2.2	Requisitos de Sistemas . . . . .	47
5.2.2.1	Requisitos Funcionais . . . . .	47
5.2.2.2	Requisitos Não-Funcionais . . . . .	50
5.2.3	Diagramas . . . . .	52
5.2.3.1	Diagramas de Caso de Uso . . . . .	53
5.2.3.2	Diagrama de atividades . . . . .	54
5.2.3.2.1	Diagrama de atividade - Cadastro do professor . . . . .	54
5.2.3.2.2	Diagrama de atividade - Troca de senha . . . . .	55
5.2.3.2.3	Diagrama de atividade - Login . . . . .	56
5.2.3.2.4	Diagrama de atividade - Cadastro de turma . . . . .	57
5.2.3.2.5	Diagrama de atividade - Cadastro de aluno . . . . .	58
5.2.3.2.6	Diagrama de atividade - Chamada . . . . .	59
5.2.3.2.7	Diagrama de atividade - Relatório . . . . .	60
5.2.3.3	Diagrama de navegação . . . . .	61
5.2.4	Protótipos de telas . . . . .	62
5.2.4.1	Telas iniciais do aplicativo . . . . .	62
5.2.4.2	Tela principal do aplicativo . . . . .	63
5.2.4.3	Tela de detalhes de uma turma . . . . .	64
5.2.4.4	Telas de edição . . . . .	65
5.2.4.5	Tela para a realização da chamada . . . . .	66
<b>5.3</b>	<b>Testando a qualidade do código . . . . .</b>	<b>67</b>
5.3.1	Testando a API . . . . .	67



5.3.2	Testando o estudo de caso . . . . .	68
6	<b>CONCLUSÃO . . . . .</b>	<b>69</b>
6.1	<b>Trabalhos futuros . . . . .</b>	<b>69</b>
	<b>REFERÊNCIAS . . . . .</b>	<b>70</b>
7	<b>APÊNDICE . . . . .</b>	<b>72</b>
7.1	<b>Pesquisa para validação do problema . . . . .</b>	<b>72</b>

# 1 Introdução

De acordo com DINIZ, chamada pode ser descrita como:

"[...]frequência escolar é um indicativo que serve para que a escola tenha noção de quem são os alunos faltosos[...]"(DINIZ, 2020)

Esse processo normalmente é feito chamando o nome de todos os alunos, onde quem está presente responde. Esse processo já passou por mudanças, dependendo das tecnologias disponíveis, como por uma lista, feita de papel, após isso os sistemas eletrônicos surgiram facilitando esse processo, e atualmente já é utilizada a biometria e portais *web* para tal processo.

Dentro da Lei de Diretrizes e Bases da Educação Nacional, (lei nº 9.394/1996), em seu artigo 24º, inciso 6, define:

"O controle de frequência fica a cargo da escola, conforme o disposto no seu regimento e nas normas do respectivo sistema de ensino, exigida a frequência mínima de setenta e cinco por cento do total de horas letivas para aprovação;(Brasil, 1996)"

Esta lei, além de regulamentar a base de educação nacional, também define que a frequência deve ser realizada de forma oral, lista de presença ou outro meio permitido pela instituição.

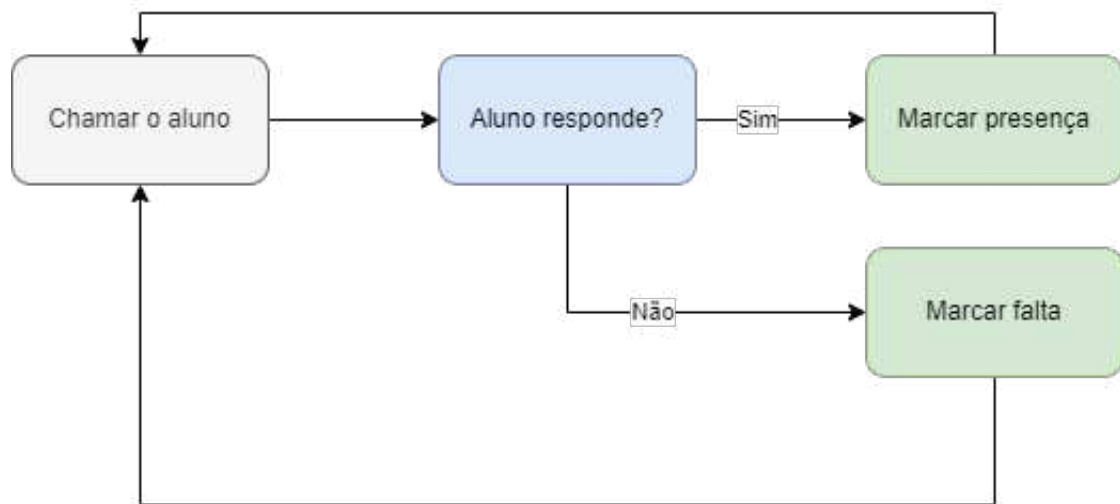
A finalidade deste trabalho é a elaboração de um serviço para ser consumido por aplicações que se propõem a agilizar esse processo.

A aplicação utiliza a tecnologia de *QRCode* para realizar a verificação dos alunos, que é um tipo de código de barras que possui dados codificados. Isso faz com que os professores não percam tempo com esse processo.

## 1.1 Justificativa

O processo de chama de chamda pode ser descrito como: O trabalho foi desenvolvido

Figura 1 – Processo de chamada



**Fonte:** Autoria própria (2023)

para agilizar o processo de chamada no dia a dia dos professores, removendo as etapas do professor chamar o aluno e o mesmo responder, e também organizar as turmas em que ministram, facilitando a realização desse processo e visando reduzir o tempo necessário para o mesmo.

A transformação do aplicativo em um serviço serve para abranger mais plataformas, fazendo com quem for utilizar essa *API* tenha mais liberdade de como e com o que deseja desenvolver, seja em uma plataforma diferente da qual o serviço foi originamente feito para atuar, ou criar um fluxo alternativo para o funcionamento da aplicação.

## 1.2 Objetivos

O objetivo desse trabalho de conclusão de curso é, com base nos resultados de pesquisas e estudos, fazer a validação de um serviço que possibilite a realização da frequência em sala de aula ou reuniões utilizando *QrCode*.

### 1.2.1 Objetivo Geral

Criar o projeto de uma *REST API* que realiza frequência escolar via *QrCode*, gerar relatórios para análise, e organização das turmas de um professor.

### 1.2.2 Objetivos Específicos

- Criar a solução para multiplataformas.
- Realizar testes para viabilizar tal solução.

## 2 Referencial Teórico

Nesse capítulo será apresentado todo o embasamento para o desenvolvimento da ideia da aplicação, como aplicativos similares, tecnologias que tem o mesmo propósito das que foram escolhidas e as que foram necessárias para o desenvolvimento do aplicativo.

### 2.1 Aplicações com propostas similares

É possível encontrar alguns aplicativos e plataformas com a proposta voltada para o meio profissional, na hora de bater o ponto por exemplo, no meio educacional, esses aplicativos são menos populares, justamente por continuarem não sendo de forma automatizada, tendo que marcar aluno por aluno, e fazendo a chamada do mesmo jeito.

#### 2.1.1 Toaqui

O aplicativo toaqui é um aplicativo com a proposta de fazer o controle de presença de alunos, possui mais de 50 mil instalações, segundo a página do app na playstore ([APPS, 2022](#)). O ponto desse aplicativo é que não possui uma forma de realizar a chamada de maneira automática, deixando essa tarefa para o professor.

#### 2.1.2 Factorial

Factorial é um software voltado exclusivamente para controle de entrada e saída dos funcionários de uma empresa, com mais de 60 mil empresas utilizando essa aplicação ([TEAM, 2022](#)), é uma plataforma com alta complexidade, que contempla desde o horário que o ponto foi batido até permissões diferentes para os funcionários, onde um supervisor pode revisar os horários de seus subordinados. Funciona com um QRCode fixo, e os funcionários, com um aplicativo, escaneiam ele.

#### 2.1.3 Doity

É um aplicativo extremamente ágil que possibilita o check-in em eventos de forma mais simples, sendo que o ponto a ser destacado é redução de custo e agilidade na hora de ingressar no evento ([BARBOSA; NEVES, 2013](#)). Sua proposta é voltada quase que inteiramente para eventos, não deixando possibilidade para uma questão tão repetitiva quanto a chamada escolar, por exemplo.

#### 2.1.4 Serpro

Mais um aplicativo voltado para empresas, porém o organizador gera apenas 1 QRCode e os participantes escaneiam para registrar sua presença. Esse aplicativo tem sua proposta voltada para reuniões não rotineiras, que não se repetem toda semana (BARRETOM, 2022).

#### 2.1.5 Comparação entre as aplicações

Para uma visão geral, segue uma tabela comparando algumas das funcionalidades que afetam diretamente o funcionamento destes aplicativos.

Tabela 1 – Tabela Comparativa Entre as Aplicações.

Funcionalidade	Toaqui	Factorial	Doity	Serpro	API
Realizar chamada	X	X	X	X	X
Precisa de um dispositivo auxiliar		X		X	
Mantém registros	X	X	X		X
Realiza cadastros uma única vez	X	X			X
Pode ser utilizado com dispositivos móveis	X		X	X	X

**Fonte:** Autoria Própria (2022).

## 2.2 Tecnologias a disposição

Nesse tópico serão abordadas as diferentes tecnologias que poderiam ser usadas para cumprir o mesmo propósito, a verificação.

### 2.2.1 Impressão digital

A autenticação por impressão digital é amplamente utilizada no cotidiano, desde desbloquear o celular até bater o ponto no trabalho, isso possibilita uma não confusão, já que é impossível que duas pessoas possuam impressões digitais iguais, conforme mostra um artigo da ufmg, a impressão digital é formada por influência genética e pelo movimento do bebê na barriga da mãe ([SILVA, 2022](#)).

Figura 2 – Impressão Digital

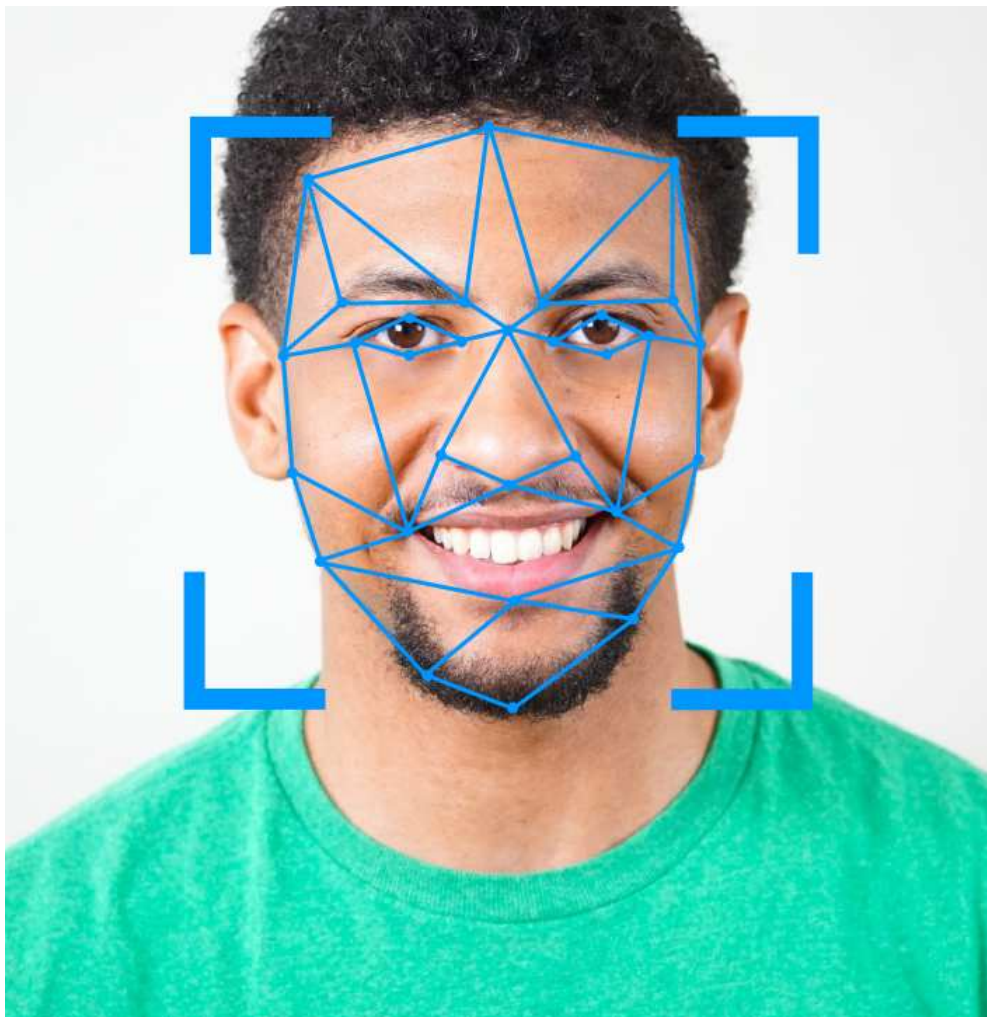


**Fonte:** Vexels (2023)

### 2.2.2 Reconhecimento Facial

Essa tecnologia funciona de uma maneira parecida com a impressão digital, pois ambas atuais sob o princípio da biometria, porém o reconhecimento facial utiliza o rosto da pessoa para realizar a verificação. O primeiro trabalho detalhado sobre o assunto foi publicado em 1981 (KANADE; LUCAS, 1981), por Takeo Kanade, porém o reconhecimento facial só foi posto em prática em 1990. O sistema com reconhecimento facial simplesmente recebe um rosto como entrada e verifica se é correspondente com o desejado.

Figura 3 – Reconhecimento facial



**Fonte:** SimpleID (2023)



## 2.3 Tecnologias utilizadas

Todas as tecnologias que foram selecionadas para o desenvolvimento do projeto.

### 2.3.1 JavaScript/Node.js

A escolha de *JavaScript* para a criação desse serviço se deve à sua versatilidade e popularidade, já que conta com uma infinidade de pacotes que tornam esse tipo de projeto possível, como mostrado na página oficial, o *Node.js* é:

"Como um tempo de execução assíncrono conduzido por eventos de *JavaScript*, Node.js é projetado para construir aplicativos escaláveis e de rede...."([FOUNDATION, 2023c](#))

O projeto foi desenvolvido com o auxílio dos seguintes *packages* e tecnologias:

#### 2.3.1.1 Express

Express é um *package* que permite a criação da *API* através de seu *framework*, que proporciona as ferramentas e métodos robustos para tal. Ele oferece uma pequena camada de *features* que não afeta o *Node.js* ([FOUNDATION, 2023b](#)).

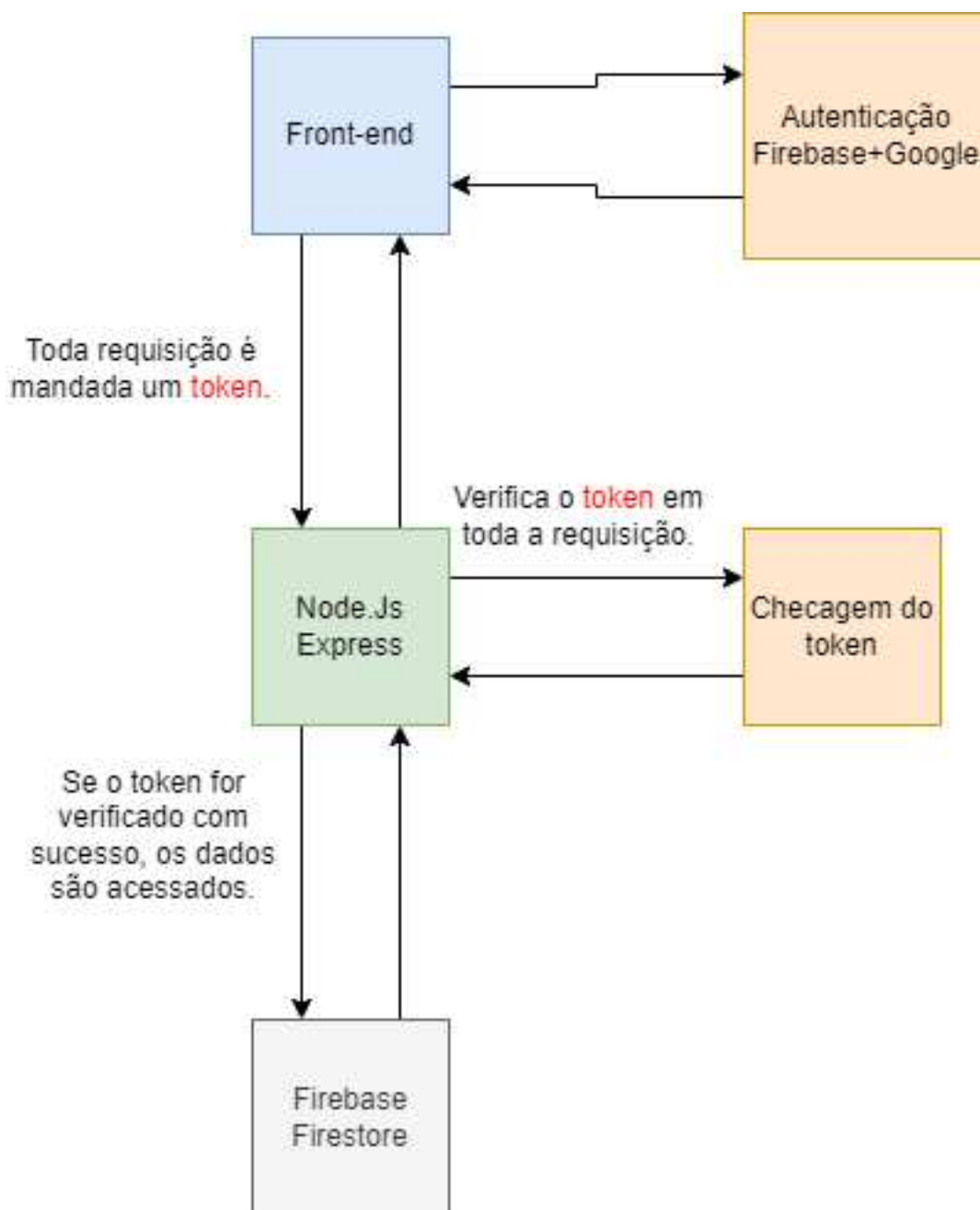
#### 2.3.1.2 Firebase: Firestore

*Firestore* é um recurso de banco de dados do firebase, que armazena dados de maneira não-relacional. Possui versatilidade no que diz respeito à compatibilidade, sendo possível ser integrado com a maioria das tecnologias existentes. É também flexível e escalável ([DEVELOPERS, 2023](#)).

### 2.3.1.3 Funcionamento

O diagrama mostra como exemplo o funcionamento de uma aplicação que utiliza tais tecnologias, utilizando o recurso de autenticação nativo, oferecido pelo *firebase* em conjunto com o *google*:

Figura 4 – Diagrama de Funcionamento.



**Fonte:** Autoria própria (2023)

### 2.3.2 QrCode

*QrCode* é um tipo de código de barras, cada *QrCode* é gerado a partir dos dados que o criador quiser, originalmente criado por uma empresa japonesa em 1994, é facilmente escaneado por qualquer celular com câmera. A combinação desses fatores, faz com que seja possível colocar qualquer tipo de dados ou informações dentro de um *QrCode*, incluindo imagens, *links*, objetos 3d, localizações, textos, basicamente tudo que possa se pensar, tornando o *QrCode* uma das tecnologias mais utilizadas hoje em dia.

Figura 5 – QrCode



**Fonte:** Wikipédia (2023)

## 2.4 QRCode vs Reconhecimento facial vs Impressão digital

Diante das opções para verificar os alunos, o QRCode foi escolhido por ser mais seguro, segue uma comparação entre as opções. Todas as opções poderiam ser utilizadas para realizar tal verificação, porém, o reconhecimento facial e a impressão digital necessitam de muitos recursos para serem utilizados, então o QRCode fica como sendo a mais adequada para a situação, a tabela abaixo representa estes fatores com mais visibilidade, sendo que os fatores apontados são os que dizem respeito exclusivamente à este tipo de aplicação.

Tabela 2 – Tabela Comparativa Entre as Tecnologias

	Prós	Contras
Reconhecimento Facial	Rapidez	requer alto poder de processamento
	Ausência de contato físico	Pode ser driblada com ajuda de impressão 3d
		Iluminação e ângulo influenciam na verificação
Impressão digital	Rapidez	Necessidade do dedo estar na posição e temperatura adequadas
	Privacidade	Precisa recadastrar caso aconteça algo com o usuário
		Pode degradar com o tempo
QRCode	Rapidez	Levemente mais lento
	Facilidade	Pode ser compartilhado
	Versatilidade	
	Geração de códigos únicos	

**Fonte:** Autoria Própria (2022).

## 3 Metodologia

A metodologia utilizada nesse trabalho de conclusão de curso é a pesquisa aplicada e bibliográfica.

A pesquisa aplicada se refere ao planejamento e desenho de um serviço que realiza o controle e organização de presença em certos eventos, como sala de aula, assim como sua implementação feita para a realização de testes.

A pesquisa bibliográfica serviu tanto como embasamento teórico quanto como recurso para validar a transição de somente aplicativo para serviço, resultando numa abordagem multiplataforma.

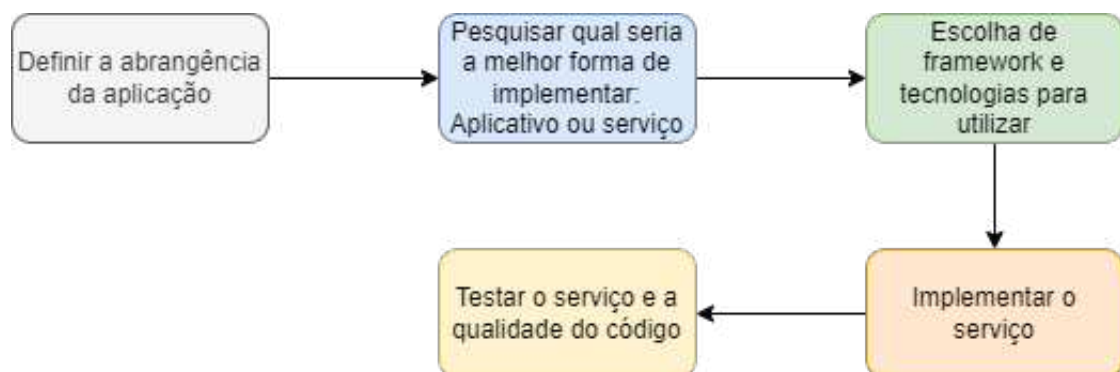
### 3.1 Definição de escopo

O escopo deste trabalho foi definido junto ao professor orientador e o professor da disciplina, através de aulas e reuniões para esse propósito.

A utilização da pesquisa aplicada se deu na construção e teste da *API*, já a pesquisa bibliográfica foi utilizada na escolha de qual *software* e *framework* seriam utilizados.

Com isso, o seguinte plano foi estabelecido para a realização do trabalho:

Figura 6 – Planejamento do trabalho



Fonte: Autoria própria (2023)

## 3.2 Tecnologias

No desenvolvimento deste serviço, foi escolhido o Node.js como framework, que somado com os *packages*, proporcionou uma boa base para construir a *API*. Na parte do banco de dados, o *firebase firestore* é a escolha, ele consegue entregar uma conexão estável, atualização em tempo real e trabalha de maneira não-relacional. O QrCode será decodificado pelo front end da aplicação, a *API* recebe somente o atributo *data*.

### 3.3 Ferramentas utilizadas

Nessa seção serão apresentadas as ferramentas utilizadas neste trabalho de Conclusão de Curso:

- Para a construção do serviço, vscode.
- Figma para a criação dos protótipos de tela.
- Creately e Draw.io para a criação dos diagramas.
- Postman para os testes da *API*.
- SonarQube para métrica do código.
- ChatGPT para geração do código de estudo de caso.

## 4 Desenvolvimento

Após a definição da extensão do serviço, foi elegida a tecnologia para construção da *API*, e a construção do banco, e posteriormente, testes.

### 4.1 REST API vs GraphQL

Para o desenvolvimento de *API's*, a abordagem REST é a mais utilizada e conhecida, porém, o *graphql* se mostra uma forte alternativa para tal função.

*GraphQL* possui uma abordagem diferente, que promete acabar com o problema de *underfetch* e *overfetch*, que se consistem em o retorno da *API* sendo menor do que o necessário ou maior do que o necessário, respectivamente. Com essa abordagem, é possível definir exatamente o que é necessário através de um sistema de *types*, onde é possível criar estruturas semelhantes a classes, para que consiga retornar um JSON da melhor maneira para determinada situação.

No *GraphQL* a requisição é montada com os dados que se deseja conseguir, e somente eles são retornados.

"Ask for what you need, get exactly that" (FOUNDATION, 2023a)

Já em uma *REST API*, o processo é mais estático, a requisição é padrão, o consumidor não escolhe o que será retornado.

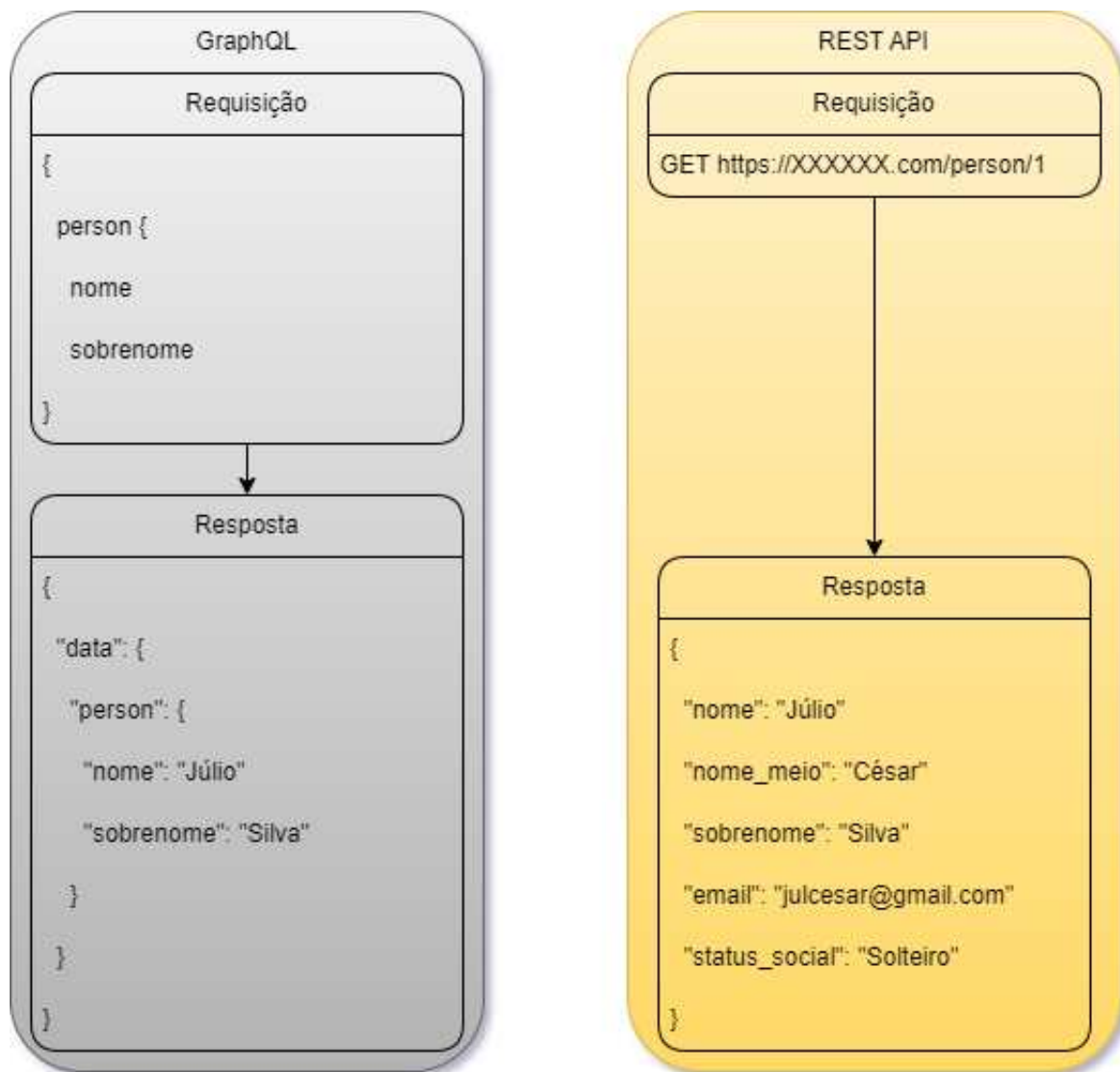
No caso específico dessa aplicação, essa abordagem não é necessária, já que a quantidade e formato de dados que são enviados nas requisições e respostas são simples, o serviço já foi moldado com esse objetivo, visando a redução do *overfetch* e *underfetch*. E em caso de uma expansão futura, o firebase trabalha com armazenamento em JSON, então é simples, já que não é necessária a criação de *types* adicionais.



#### 4.1.1 Diagrama de comparação

Diagrama de comparação entre requisições e respostas com ambas as abordagens:

Figura 7 – Diagrama de Comparação



**Fonte:** Autoria própria (2023)

## 4.2 Banco de dados: Firebase Firestore

Os dados são armazenados em formato *JSON* no *firestore*, o que acaba dando uma boa liberdade para o utilizador, já que não possui nenhum tipo de padronização como um banco de dados relacional. A organização de tais dados poderia ser feita de algumas maneiras, como colocar todos os dados em um unico documento, ou separá-los em coleções diferentes.

### 4.2.1 Todos os dados em um documento

Com essa abordagem, o principal problema é o *overfetch* causado, já que mesmo que o consumidor da *API* precise somente dos dados de um aluno, a resposta da api ainda seria tudo, outro problema encontrado com esse método é a manipulação dos dados, que ficou de maneira confusa, devido ao número de array dentro de outro array que existe. Utilizando esse padrão, os dados ficam organizados da seguinte forma:

Figura 8 – Exemplo do documento único

```
1  {
2    "name": "Silvano",
3    "email": "silvano@gmail.com",
4    "groups": [
5      {
6        "name": "Disp1",
7        "meetings": 40,
8        "maxAbsences": 12,
9        "institution": {
10         "name": "Unitins",
11         "color": "#2403fc"
12       },
13       "participants": [
14         {
15           "name": "João Pedro",
16           "email": "joao@gmail.com",
17           "frequency": [
18             {
19               "day": "25/05/2023",
20               "present": true
21             }
22           ]
23         }
24       ]
25     }
26   ]
27 }
```

**Fonte:** Autoria própria (2023)

## 4.2.2 Organizando em coleções

Essa foi a abordagem escolhida para essa solução, já que elimina a maioria do *overfetch*. Desta maneira, o *JSON* mostrado no exemplo foi dividido, gerando 4 coleções.

### 4.2.2.1 institution

Coleção dedicada à armazenar as instituições, os documentos armazenados nela possuem a seguinte estrutura:

Figura 9 – Exemplo da coleção institution

```
1 {  
2   "name": "Unitins",  
3   "color": "#2403fc"  
4 }
```

**Fonte:** Autoria própria (2023)

### 4.2.2.2 professor

Nesta coleção, os dados básicos do professor são armazenados, sua estrutura:

Figura 10 – Exemplo da coleção professor

```
1 {  
2   "name": "Silvano",  
3   "email": "silvano@gmail.com"  
4 }
```

**Fonte:** Autoria própria (2023)

#### 4.2.2.3 group

Esta coleção armazena os grupos, o campo "professor" é referido ao *ID* único do documento do professor, o que possibilita diferenciar e separar os grupos de um professor.

Figura 11 – Exemplo da coleção group

```
1 {  
2   "name": "Disp1",  
3   "meetings": 40,  
4   "maxAbsences": 12,  
5   "professor": "73oyK1REPNB1u6AMMdkw",  
6   "institution": {  
7     "name": "Unitins",  
8     "color": "#2403fc"  
9   }  
10 }
```

**Fonte:** Autoria própria (2023)

#### 4.2.2.4 participant

Nesta coleção são armazenados os alunos ou participantes, dependendo do que for o objetivo do consumidor da *API*, possui o mesmo esquema de referencia dos documentos da coleção *group*, só que desta vez, ao invés de professor, é o grupo a que ele pertence.

Figura 12 – Exemplo da coleção participant

```
1 {  
2   "name": "João Pedro",  
3   "email": "joao@gmail.com",  
4   "group": "73oyK1REPNB1u6AMMdkw",  
5   "frequency": [  
6     {  
7       "day": "25/05/2023",  
8       "present": true  
9     }  
10  ]  
11 }
```

**Fonte:** Autoria própria (2023)

## 4.3 Requisições

As requisições da *API* foram criadas com os seguintes pontos como guia:

- *Id* é passado como parâmetro
- *Body* da requisição contém os dados para a modificação ou criação
- O *Id* não pode ser alterado em um update

Desta forma, as requisições foram divididas em 4 partes, uma parte direcionada para cada coleção existente.

### 4.3.1 Lógica das requisições

As requisições seguem uma certa lógica, sendo que as básicas são: criação, leitura e edição de documentos.

#### 4.3.1.1 Criação de documento

Para a criação dos documentos, são passadas através do *body* da requisição as informações do documento, como nome e email por exemplo, com exceção do *ID*, que é passado como parâmetro.

O pseudocódigo abaixo mostra uma criação de documento, utilizando como exemplo a coleção *participant*, sendo que essa lógica é a mesma para as outras coleções:

---

**Algoritmo 1:** Exemplo de criação de documento

---

```
input  : Dados do documento
output : Documento criado dentro da coleção
body   : Body da requisição
param  : Parâmetro recebido da requisição

1  /*Tenta realizar a ação*/
   try:
2  |  /*Aguarda a requisição feita para o banco de dados ser concluída*/
      await db.coleção("participant").doc().create(
        name ← body.name,
        group ← param.id,
        email ← body.email,
        frequency ← body.frequency
      )
      /*Se a requisição foi um sucesso, retorna a mensagem*/
      return resposta.send("Participante adicionado")
3  catch erro:
4  |  /*Se a requisição falhar, é exibida a mensagem de erro*/
      console.log(erro)
      return resposta.status(500).send(erro)
5  end
```

---

#### 4.3.1.2 Leitura de documento

Para ler um documento é necessário passar um identificador como parâmetro, no caso do participante é o email, que é um atributo único, desta forma lógica dessa requisição é a seguinte:

---

**Algoritmo 2:** Exemplo de leitura de documento

---

```
input : Identificador do documento
output : Informações do documento
param : Parâmetro recebido da requisição

1  /*Tenta realizar a ação*/
   try:
2  | /*Aguarda a requisição feita para po banco de dados ser concluída*/
   | referenciaInstancia  $\leftarrow$  db.coleção("participant")
   | listaDocumentos  $\leftarrow$ 
   | await referenciaInstancia.where("email" == param.email)
   | itemFinal
   | for doc  $\in$  listaDocumentos do
3  | | item  $\leftarrow$  (
   | |   id  $\leftarrow$  doc.id,
   | |   name  $\leftarrow$  doc.data.name,
   | |   group  $\leftarrow$  doc.data.group,
   | |   email  $\leftarrow$  doc.data.email,
   | |   frequency  $\leftarrow$  doc.data.frequency
   | | )
   | | itemFinal  $\leftarrow$  item
   | | break
4  | end
5  | /*Se a requisição foi um sucesso, retorna a mensagem*/
   | return resposta.send(itemFinal)
6  catch erro:
7  | /*Se a requisição falhar, é exibida a mensagem de erro*/
   | console.log(erro)
   | return resposta.status(500).send(erro)
8  end
```

---

No caso de ler um grupo, a lógica segue quase a mesma, só que ao invés de fazer a comparação com o email, é utilizado o campo *group*, dessa forma retornando um *array* de participantes.



#### 4.3.1.3 Edição de documento

Na edição de documentos, é necessário passar o *ID* como parâmetro e os campos que serão alterados no *body* da requisição, sua lógica é descrita como:

---

**Algoritmo 3:** Exemplo de edição de documento

---

```
input : Identificador do documento
output : Alterações do documento
param : Parâmetro recebido da requisição
1 /*Tenta realizar a ação*/
  try:
2   /*Aguarda a requisição feita para po banco de dados ser concluída*/
     referenciaInstancia  $\leftarrow$  db.coleção("participant").doc(param.id)
     resposta  $\leftarrow$  await referenciaInstancia.update(
       name  $\leftarrow$  body.name,
       email  $\leftarrow$  body.email
     )
     /*Se a requisição foi um sucesso, retorna a mensagem*/
     return resposta.send("participantealterado")
3 catch erro:
4   /*Se a requisição falhar, é exibida a mensagem de erro*/
     console.log(erro)
     return resposta.status(500).send(erro)
5 end
```

---

Note que só é possível atualizar certos campos de um documento, somente os que são necessários e referentes ao participante de forma exclusiva.

### 4.3.2 Requisições chave

As requisições principais para o funcionamento da *API* são enviar os dados de *QRCode* e realizar a frequência.

#### 4.3.2.1 Envio de *QRCode*

Para fazer esse envio de dados, a requisição recebe o *ID* do grupo, e retorna a lista dos *ID's* dos participantes.

---

**Algoritmo 4:** Envio dos dados de *QRCodes*

---

```
input : Identificador do grupo
output : Array de ID's
param : Parâmetro recebido da requisição

1  /*Tenta realizar a ação*/
   try:
2  |  /*Aguarda a requisição feita para o banco de dados ser concluída*/
      referenciaInstancia ← db.coleção("participant")
      listaDocumentos ←
      await referenciaInstancia.where("group" == param.id).get()
      group ← []
      for doc ∈ listaDocumentos do
3  |   | group.add(doc)
4  | end
5  | res ← (
      participants ← group
      )
      /*Se a requisição foi um sucesso, retorna a mensagem*/
      return resposta.send(res)
6 catch erro:
7  | /*Se a requisição falhar, é exibida a mensagem de erro*/
      console.log(erro)
      return resposta.status(500).send(erro)
8 end
```

---

Deste modo, quem consumir o serviço vai receber uma lista com os devidos *ID's*, a partir disso é mandá-los como dados através de *QRCodes*.

#### 4.3.2.2 Realização de frequência

A realização da frequência se dá a partir da comparação entre os participantes do grupo que foi passado como parâmetro e os presentes, que são passados no *body* da requisição.

---

**Algoritmo 5:** Realização da frequência

---

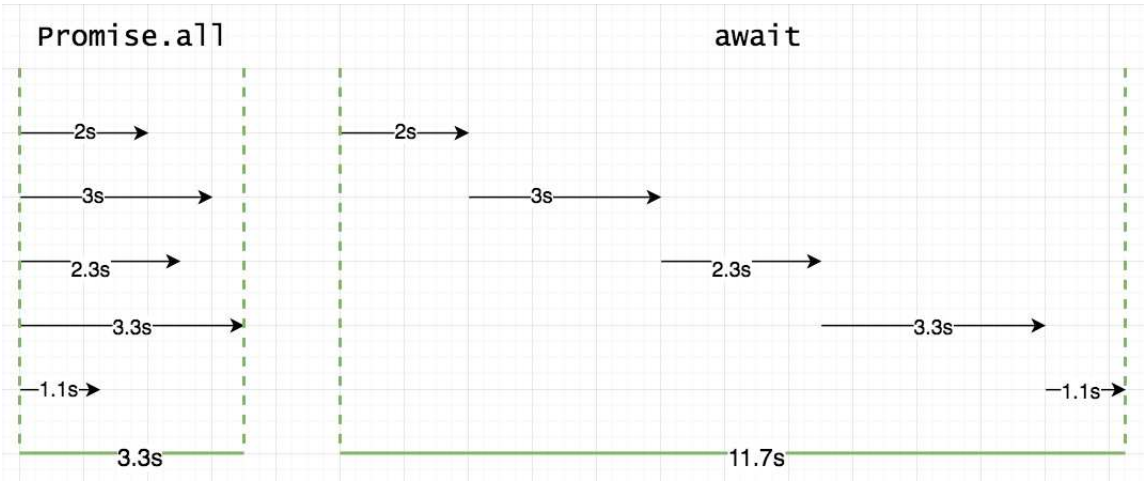
```
input : Array de presentes
output : Alteração dos documentos
param : Parâmetro recebido da requisição

1  /*Tenta realizar a ação*/
   try:
2    /*Aguarda a requisição feita para o banco de dados ser concluída*/
      referenciaInstancia ← db.coleção("participant")
      listaDocumentos ←
      await referenciaInstancia.where("group" == param.id).get()
      group ← []
      presentes ← body.presents
      for doc ∈ listaDocumentos do
3        item ← (
          id ← doc.id,
          name ← doc.data.name,
          group ← doc.data.group,
          email ← doc.data.email,
          frequency ← doc.data.frequency
        )
        group.add(item)
4    end
5    promises ← []
      for g ∈ group do
6      p ← db.coleção("participant").doc(g.id)
        promise ← p.update(
          frequency.add(
            day ← Date.now,
            present ← presents.includes(g.id),
          )
        )
        promises.add(promise)
7    end
8    await Promise.all(promises)
      /*Se a requisição foi um sucesso, retorna a mensagem*/
      return resposta.send("Frequencia realizada")
9  catch erro:
10   /*Se a requisição falhar, é exibida a mensagem de erro*/
      console.log(erro)
      return resposta.status(500).send(erro)
11 end
```

---

Essa requisição é a que mais consome tempo, por conta do alto número de requisições, para minimizar o tempo gasto, foi feito uso de promessas, "*promises*", isso permite que ao invés de fazer uma alteração por vez no banco, todas sejam feitas de maneira simultânea, através de um *array* de promessas, que nesse caso são alterações, assim reduzindo o tempo.

Figura 13 – Uso de promessas



Fonte: (STOJANOVIC, 2023)

4.3.3 Lista de requisições

As requisições da API possuem o mesmo prefixo, "*http://127.0.0.1:5001/tcc—chamada-com-qr-code/us-central1/app/api/*", que representa onde o serviço está alocado e o nome do projeto no *firebase*, e seguido disso, a parte específica de cada requisição.

Tabela 3 – Exemplo de requisições básicas

Método	URL	Descrição	Parâmetro	Body	Retorno
POST	create-institution	Cria uma instituição		Nome e color	
GET	read-institution	Lê uma instituição	Nome da instituição		Informações da instituição
GET	read-institutions	Lista as instituições			Lista de instituições
PUT	update-institution	Atualiza uma instituição	ID da instituição	Nome e color	
DELETE	delete-institution	Deleta uma instituição	ID da instituição		

Fonte: Autoria Própria (2023).

Uma requisição completa fica: "*http://127.0.0.1:5001/tcc—chamada-com-qr-code/us-central1/app/api/read-institution/nome*", onde após o "*app/api*" entra o nome e depois o parâmetro.

Esse padrão de requisições se repete para as demais coleções, com a exceção de *participant* e *group*, que possuem uma relação na requisição de *delete*, toda vez que um grupo é deletado, os participantes de tal grupo também são deletados. Para as outras coleções, basta alterar a url para se adequar a coleção, como por exemplo: alterar *create-institution* para *create-group*, e os dados necessários, que são:

Tabela 4 – Dados necessários para outras coleções

Coleção	Dados necessários
Professor	Name e email
Group	Institution, maxAbsences, meetings e name
Participant	Email, name e frequency

**Fonte:** Autoria Própria (2023).

As outras requisições da *API* são:

Tabela 5 – Lista de requisições chaves da API

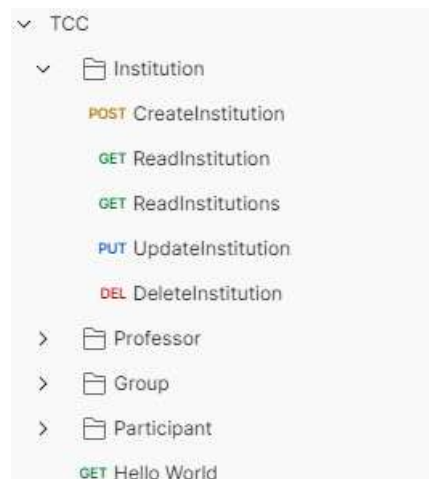
Método	URL	Descrição	Parâmetro	Body	Retorno
GET	send-qr-data	Envia os dados para geração de QrCodes	ID do grupo		Lista de Dados para QrCodes
PUT	frequency	Realiza a frequência	ID do grupo	Lista de presentes	

**Fonte:** Autoria Própria (2023).

Todas foram testadas através do *software Postman*, e o *body* da requisição segue o padrão *JSON*, como mostrado na subseção 4.2.2.

A organização no *Postman* são como exibidas:

Figura 14 – Coleção de requisições no postman



**Fonte:** Autoria própria (2023)

## 4.4 Estudo de caso

Com o auxílio do *ChatGPT*, foi elaborada uma simples tela para demonstrar o consumo da *API* de forma prática, segundo o próprio *ChatGPT*, ele é definido como:

"O *ChatGPT* é um modelo avançado de linguagem desenvolvido pela OpenAI. Ele é treinado em uma ampla variedade de textos para aprender padrões e relações entre palavras e pode realizar várias tarefas de processamento de linguagem natural, como responder perguntas e realizar diálogos interativos. No entanto, ele não possui conhecimento próprio além do treinamento e não pode fornecer informações atualizadas ou precisas sobre eventos recentes. É importante avaliar suas respostas e verificar as informações críticas ou sensíveis fornecidas por ele. (OPENAI, 2023)"

Através de simples perguntas e pedidos, essa inteligência artificial foi capaz de criar a tela, utilizando o *framework flutter*, o caminho seguido foi:

- Pedir para gerar um componente que consiga mostrar um objeto *JSON*.
- Pedir para gerar uma tela que faça a listagem desses objetos.
- Pedir para alterar a forma de recebimento do objeto *JSON*, de manual para através de requisição *http*.

Após essas ações e alguns ajustes na tela, o resultado obtido foi:

Figura 15 – Tela de exibição feita pelo ChatGPT

User Data List			
Aluno: joao	Data: 2023-5-10	Data: 2023-6-7	Data: 2023-6-7
Aluno: maria	Data: 2023-5-10	Data: 2023-6-7	Data: 2023-6-7

**Fonte:** (OPENAI, 2023)

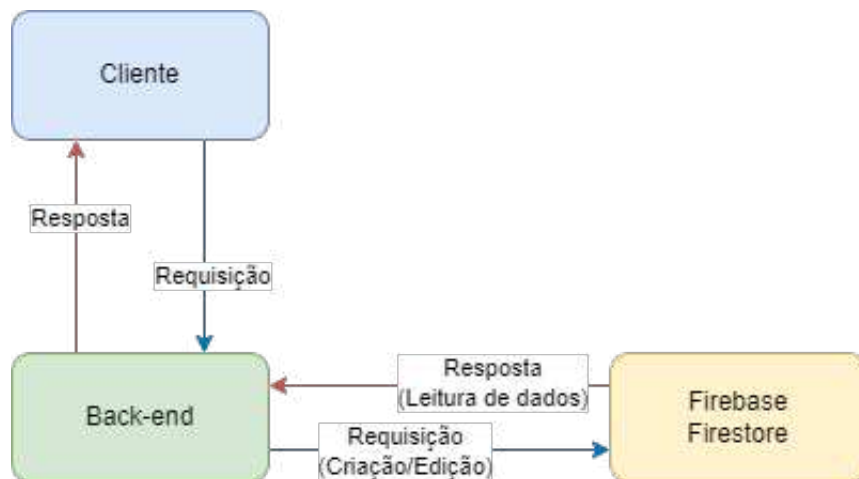
## 5 Resultados

Este capítulo mostra os resultados obtidos com o desenvolvimento do trabalho, segue em detalhes o que foi obtido a partir das definições previamente estabelecidas.

### 5.1 Funcionamto da API

A *API* foi desenvolvida para funcionar da maneira mostrada na imagem abaixo:

Figura 16 – Fluxo de Funcionamento da API



**Fonte:** Autoria própria (2023)

A parte do cliente manda a requisição para o *back-end*, que por sua vez são processadas, tendo a interação com o banco de dados *firebase*, e por fim, se tem o retorno, a resposta, como mostrada na imagem.

## 5.2 Proposta de aplicação

Uma aplicação *mobile* foi projetada como uma proposta de implementação, fazendo o consumo da *API*, segue o que foi desenvolvido:

### 5.2.1 Funcionamento do aplicativo

O aplicativo possuirá uma tela inicial onde o professor poderá visualizar as turmas cadastradas, e adicionar uma nova turma. Quando ele clicar em uma turma, ele poderá adicionar um novo aluno, enviar os QrCodes para os alunos ou realizar a chamada.

Para o professor realizar a chamada, o professor deverá selecionar a turma e simplesmente apertar o botão, isso abrirá a câmera para que ele possa escanear os QrCodes dos alunos presentes.

### 5.2.2 Requisitos de Sistemas

Os requisitos do sistema são funcionalidades ou características que um software deve apresentar.

#### 5.2.2.1 Requisitos Funcionais

Requisito funcional é uma função que o software deve atender, normalmente são feitos via reuniões com o cliente ou com base nelas, neste projeto de conclusão de curso eles foram elicitados após reuniões com o orientador. A norma ISO/IEC/IEEE 24765-2010 define requisito funcional como:

"Uma condição ou capacidade do sistema, solicitada por um usuário, para resolver um problema ou atingir um objetivo. Uma condição ou capacidade que deve ser atendida por uma solução para satisfazer um contrato, especificação, padrão ou quaisquer outros documentos formais impostos. Documentação da representação das condições ou capacidades apresentadas nos dois itens anteriores. Uma condição ou capacidade que deve ser alcançada ou possuída por um sistema, produto, serviço, resultado ou componente para satisfazer um contrato, padrão, especificação ou outro documento formalmente imposto. Requisitos incluem as necessidades quantificadas e documentadas, desejos e expectativas do patrocinador, clientes e outras partes interessadas."([STANDARDIZATION, 2010](#))

Estão listados abaixo os requisitos funcionais do sistema.



Código	Identificação	Classificação	Ator	Objetivo
RF01	Efetuar login	Importante	Usuário	Permitir a autenticação do usuário no app (e-mail e senha).
RF02	Cadastrar turma	Essencial	Usuário	O usuário deve ser capaz de cadastrar uma turma (Instituição, matéria, turno, número de aulas e número de faltas).
RF03	Cadastro de aluno	Essencial	Usuário	Deve ser possível cadastrar alunos em uma turma (Nome e e-mail).
RF04	Geração de QR code	Essencial	App	O aplicativo deve gerar um QR code por aluno, contendo o nome e a data.
RF06	Envio do QR code	Essencial	Usuário	O usuário envia o QR code ao aluno correspondente através do e-mail.
RF06	Frequência	Essencial	Usuário	O usuário pode fazer o controle de quem está presente por meio da leitura do qr code dos alunos.
RF07	Geração de tabela	Essencial	App	O aplicativo deve criar uma tabela que mostra quem esteve presente no dia.
RF08	Relatório diário	Essencial	App	O aplicativo gera um relatório mostrando a porcentagem de presença no dia.
RF09	Envio automático	Importante	App	O professor pode ativar a opção de envio automático dos QrCodes, onde ele definirá o horário.
RF10	Determinação do limite de faltas	Importante	Usuário	O usuário pode estipular um número máximo de faltas para a turma.
RF11	Notificação de limite	Importante	App	Assim que um aluno chega 1 dia próximo ao limite de faltas, é mandado um e-mail notificando o mesmo.
RF12	Editar aluno	Importante	Usuário	O usuário poderá alterar os dados de um aluno, caso tenha escrito algo de errado
RF13	Número de aulas	Importante	Usuário	O usuário deve ajustar o numero total de aulas de uma turma.
RF14	Autodestruição de turmas	Importante	App	O aplicativo exclui uma turma 1 dia após a última aula.
RF15	Remoção de aluno	Essencial	Usuário	O usuário pode remover um aluno de uma turma caso ele tenha saído da instituição.
RF16	Cor da turma	Médio	Usuário	O usuário pode definir uma cor para a turma para que seja localizada mais facilmente.
RF17	Cor da instituição	Médio	Usuário	O usuário pode definir uma cor para uma instituição para que seja de fácil identificação.
RF18	Backup	Importante	App	O aplicativo salva os dados do usuário, assim como suas turmas.
RF19	Exportar lista de presentes	Importante	App	O aplicativo deve exportar a lista dos alunos presentes em formato.xlsx.
RF20	Exportar relatórios	Importante	App	O aplicativo pode exportar os relatórios escolhidos em formato.docx.
RF21	Cadastrar Instituições	Essencial	Usuário	O usuário adiciona as instituições que trabalha
RF22	Adicionar instituição	Importante	Usuário	O usuário pode adicionar uma nova instituição no seu perfil

Código	Identificação	Classificação	Ator	Objetivo
RF23	Remover instituição	Importante	Usuário	O usuário pode remover uma instituição no seu perfil
RF24	Cadastro do professor	Essencial	Usuário	O usuário deverá criar uma conta para a utilização do aplicativo, contendo nome, instituições, e-mail e senha

### 5.2.2.2 Requisitos Não-Funcionais

De acordo com Vazquez, no livro “Engenharia de Requisitos: software orientado a negócio”, faz uma comparação entre requisitos funcionais e não funcionais, afim de evidenciar a diferença entre eles.

"Enquanto os requisitos funcionais referem-se especificamente a uma tarefa do usuário, os requisitos não funcionais indicam restrições de ordem geral que abordam aspectos relativos: À qualidade: por exemplo, a facilidade de uso, a confiabilidade, a eficiência, a portabilidade e a facilidade de manutenção. Essa relação é apenas ilustrativa de alguns tipos de requisitos não funcionais e não busca ser uma lista completa. A Figura abaixo representa o papel dos requisitos não funcionais de estabelecer níveis de serviço que direcionam como será o projeto da arquitetura que suporta as camadas de software voltadas ao atendimento dos requisitos funcionais.

Figura 17 – Comparação entre requisitos funcionais e não funcionais.



Fonte: (VAZQUEZ EDUARDO; SIMÕES SIQUEIRA, 2016)

Um aspecto que facilita a identificação dos requisitos não funcionais é que eles costumam ser constantes entre os projetos ou mudam pouco de um projeto para outro na mesma organização. Por isso, mesmo em fases bem iniciais de um projeto, se consegue ter uma boa visibilidade sobre os aspectos não funcionais necessários ao software."(VAZQUEZ EDUARDO; SIMÕES SIQUEIRA, 2016)

Abaixo segue os requisitos não-funcionais do sistema.

Código	Identificação	Classificação	Ator	Objetivo
RNF01	Usabilidade	Essencial	App	O aplicativo deve ser simples de utilizar, com interface intuitiva
RNF02	Linguagem de programação	Essencial	App	O app deverá ser desenvolvido utilizando a linguagem dart, com o framework flutter
RNF03	Arquitetura	Essencial	App	O aplicativo deve ser desenvolvido utilizando a arquitetura clean-code
RNF04	Armazenamento	Essencial	App	O aplicativo deve armazenar os arquivos de relatórios e afins na memória interna do dispositivo.

### 5.2.3 Diagramas

O livro “UML 2- Uma abordagem prática”, escrito por GUEDES, define UML como:

"A UML – Unified Modeling Language ou Linguagem de Modelagem Unificada – é uma linguagem visual utilizada para modelar softwares baseados no paradigma de orientação a objetos. [...] Essa linguagem é atualmente a linguagem-padrão de modelagem adotada internacionalmente pela indústria de engenharia de software."([GUEDES, 2018](#))

UML é a linguagem que foi utilizada para o desenvolvimento dos diagramas, essa linguagem é amplamente utilizada para a criação dos diagramas dos projetos ao redor do mundo.

### 5.2.3.1 Diagramas de Caso de Uso

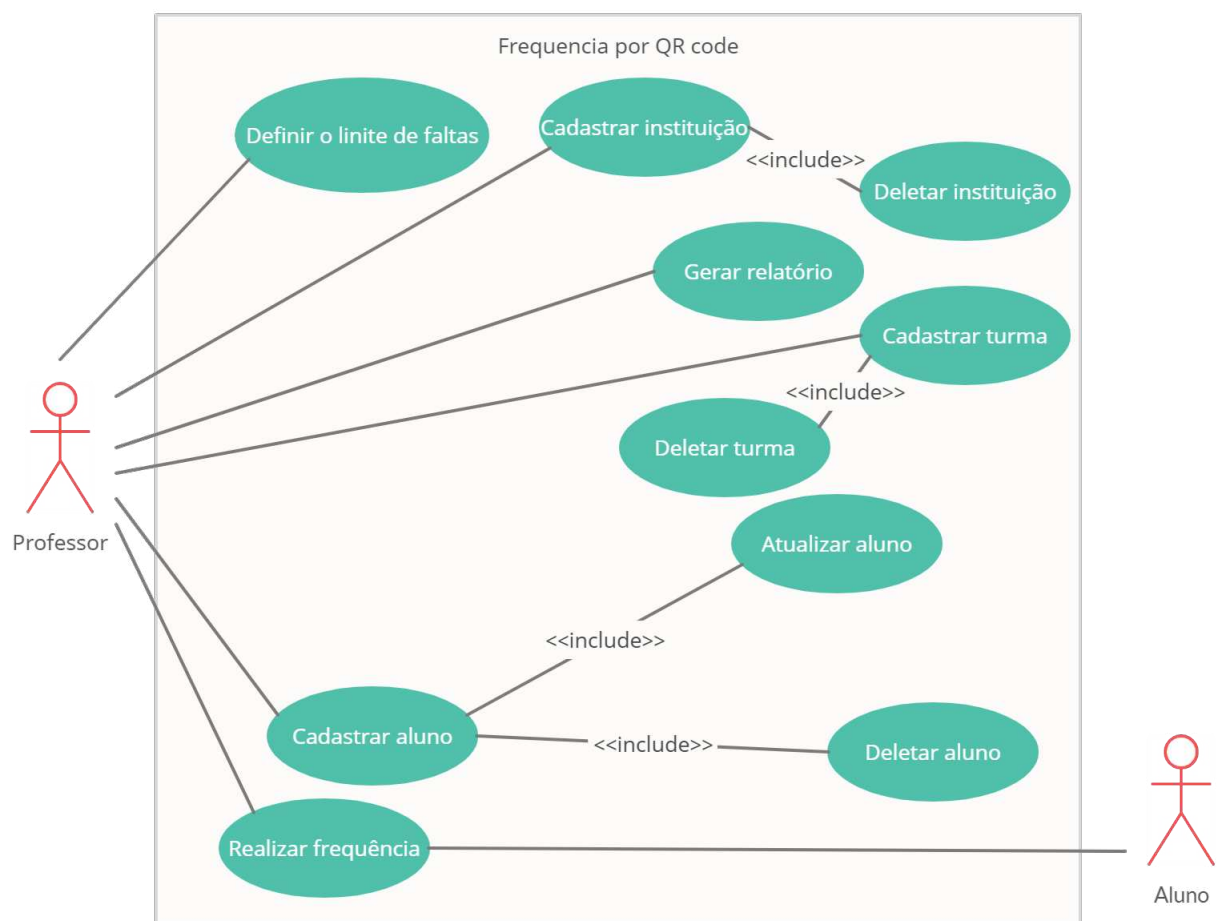
Também no livro “UML 2 – Uma abordagem prática”, o diagrama de caso de uso é definido por:

"O diagrama de casos de uso tem por objetivo apresentar uma visão externa geral das funcionalidades que o sistema deverá oferecer aos usuários, sem se preocupar muito com a questão de como tais funcionalidades serão implementadas."(GUEDES, 2018)

Ou seja, O diagrama de caso de uso deve exemplificar como e com o que os usuários vão interagir com o sistema, ou no caso, o aplicativo.

Segue o diagrama de caso de uso do aplicativo, retratando a utilização do aplicativo por quem vai interagir com o aplicativo.

Figura 18 – Diagrama de Caso de Uso do Sistema.



**Fonte:** Autoria própria (2022)

### 5.2.3.2 Diagrama de atividades

De acordo com GUEDES, no livro “UML 2 – Uma abordagem prática”, o diagrama de atividades pode ser definido como:

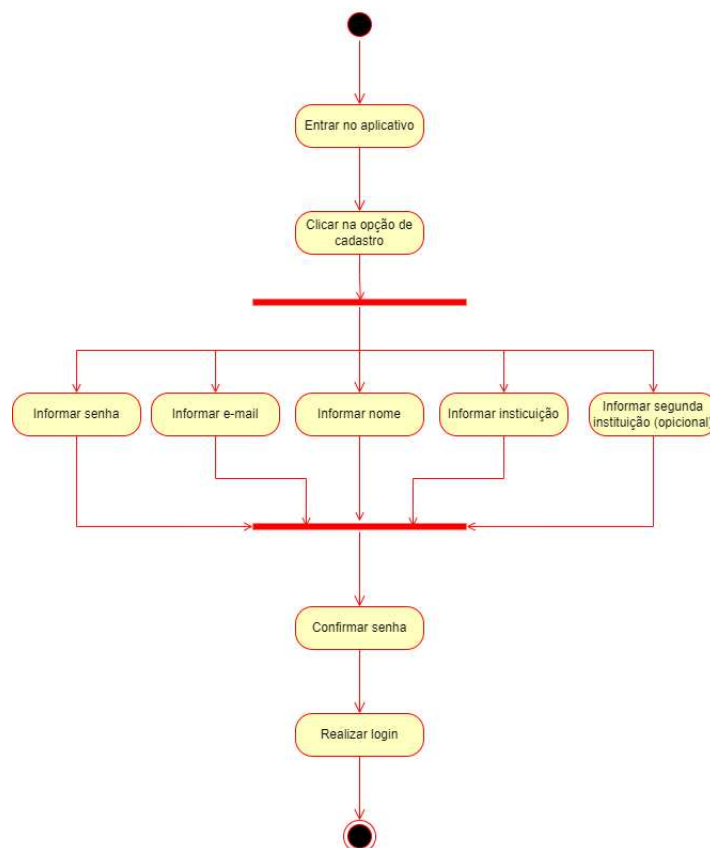
"A modelagem de atividade enfatiza a sequência e condições para coordenar comportamentos de baixo nível. Dessa forma, o diagrama de atividade é o diagrama com mais ênfase no nível de algoritmo da UML e provavelmente um dos mais detalhistas." (GUEDES, 2018)

Com isso especificado, o diagrama de atividades é um meio de detalhar um comportamento do sistema, uma atividade. Foi escolhido justamente por sua capacidade de mostrar o fluxo de uma determinada atividade do sistema de maneira simples e abrangente.

#### 5.2.3.2.1 Diagrama de atividade - Cadastro do professor

Esse diagrama serve para descrever o processo que o professor precisará fazer para que possa utilizar o aplicativo, onde deve informar seus dados. O processo mostrado abaixo é uma conta normal, porém a autenticação com o google é uma opção válida.

Figura 19 – Diagrama de atividade do cadastro do professor.



**Fonte:** Autoria própria (2022)

#### 5.2.3.2.2 Diagrama de atividade - Troca de senha

Tanto no caso de esquecimento de senha quanto no caso de o usuário simplesmente querer mudar, o processo é o mesmo, como descrito abaixo. Nessa atividade, o usuário deverá escolher a opção de esquecimento de senha, e seguir os passos como mostrado.

Figura 20 – Diagrama de atividade de troca de senha.



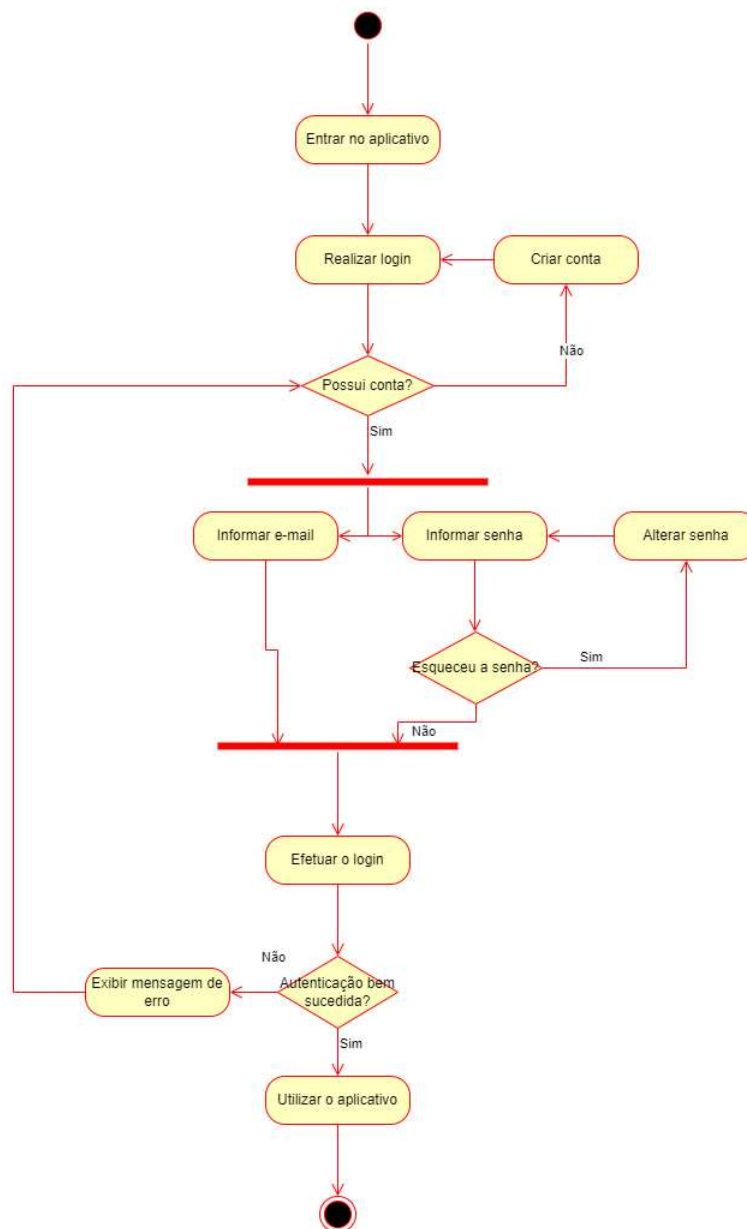
**Fonte:** Autoria própria (2022)



### 5.2.3.2.3 Diagrama de atividade - Login

Para que o professor utilize o aplicativo, o mesmo deve realizar login. As atividades criar conta e alterar senha foram previamente apresentados.

Figura 21 – Diagrama de atividade de login.

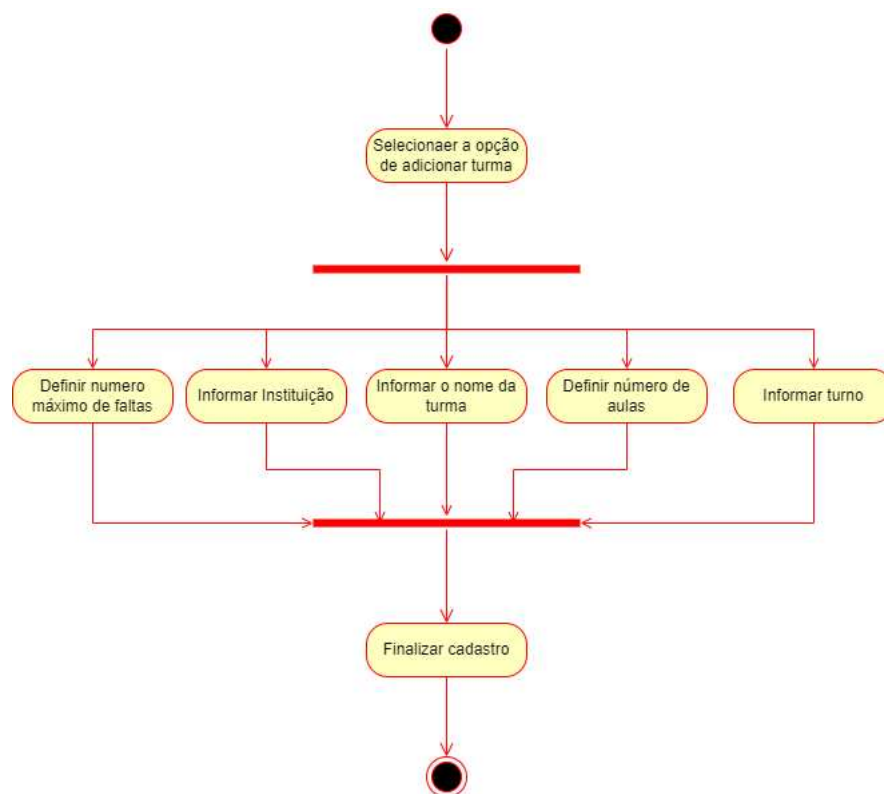


Fonte: Autoria própria (2022)

#### 5.2.3.2.4 Diagrama de atividade - Cadastro de turma

Para cadastrar uma nova turma, o professor deve informar o nome da turma, a instituição, o número de aulas, o número de faltas e o turno. Isso fara com que seja possível cadastrar os alunos.

Figura 22 – Diagrama de atividade do cadastro de turma.

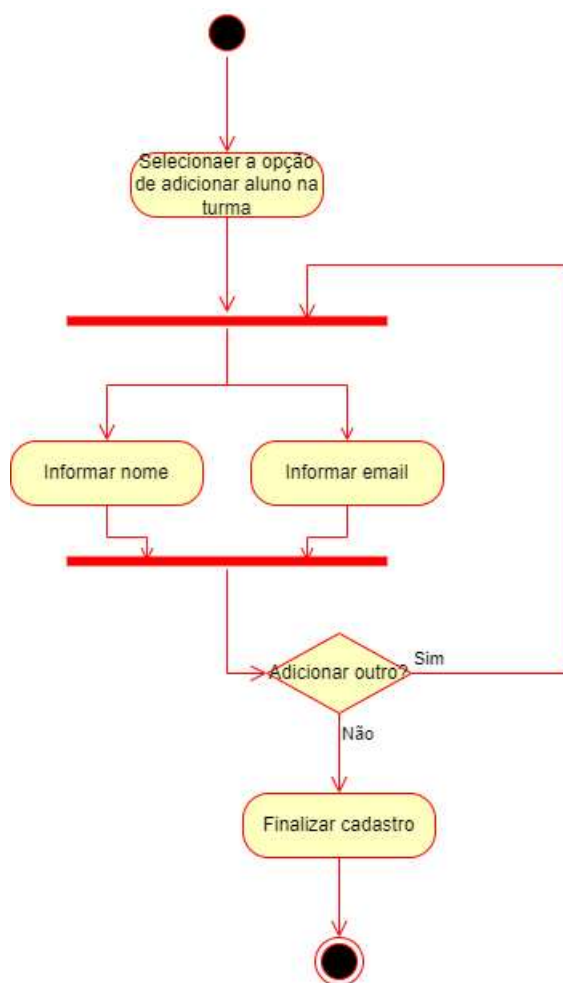


**Fonte:** Autoria própria (2022)

#### 5.2.3.2.5 Diagrama de atividade - Cadastro de aluno

Para adicionar um aluno em uma turma, o professor deve abrir a tela da turma desejada e clicar no botão de adicionar, após isso, informar o nome e e-mail do aluno, tendo a opção de adicionar outro aluno logo em seguida.

Figura 23 – Diagrama de atividade do cadastro de aluno.

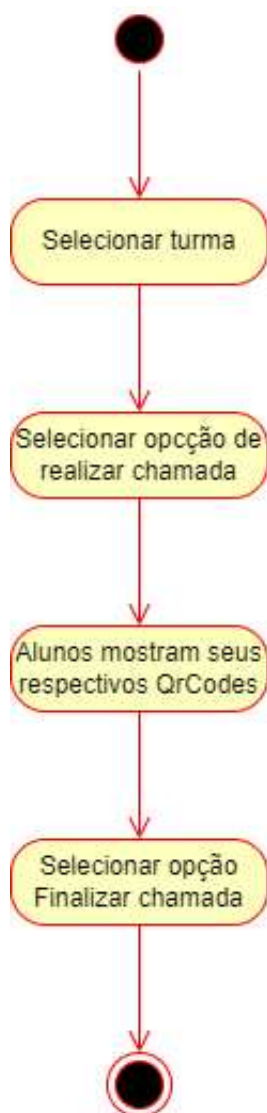


**Fonte:** Autoria própria (2022)

#### 5.2.3.2.6 Diagrama de atividade - Chamada

Para realizar a chamada em uma turma, o professor deve previamente enviar os QrCodes para os alunos, por meio de um botão que chama a função. Após isso já está tudo pronto para realizar a chamada, que segue a ordem descrita abaixo.

Figura 24 – Diagrama de atividade da chamada.

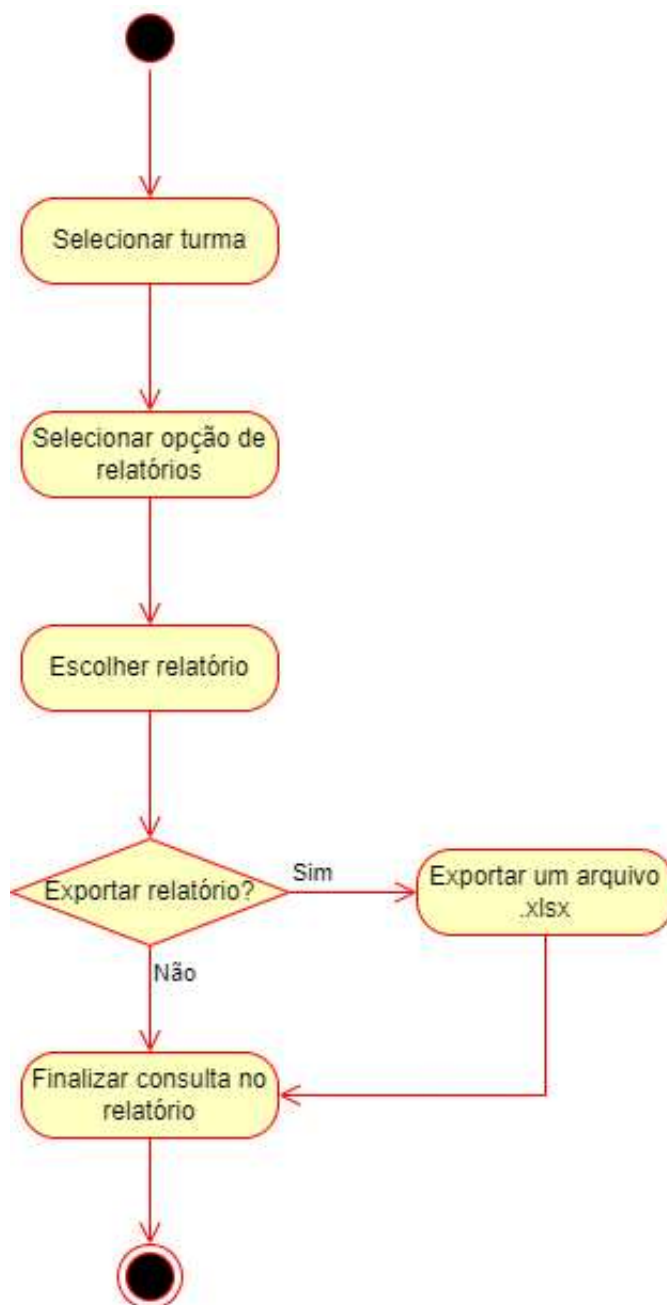


**Fonte:** Autoria própria (2022)

#### 5.2.3.2.7 Diagrama de atividade - Relatório

O professor pode visualizar um relatório geral da turma, ou individual, com a opção de exportá-lo em formato .xlsx.

Figura 25 – Diagrama de atividade do relatório.



**Fonte:** Autoria própria (2022)

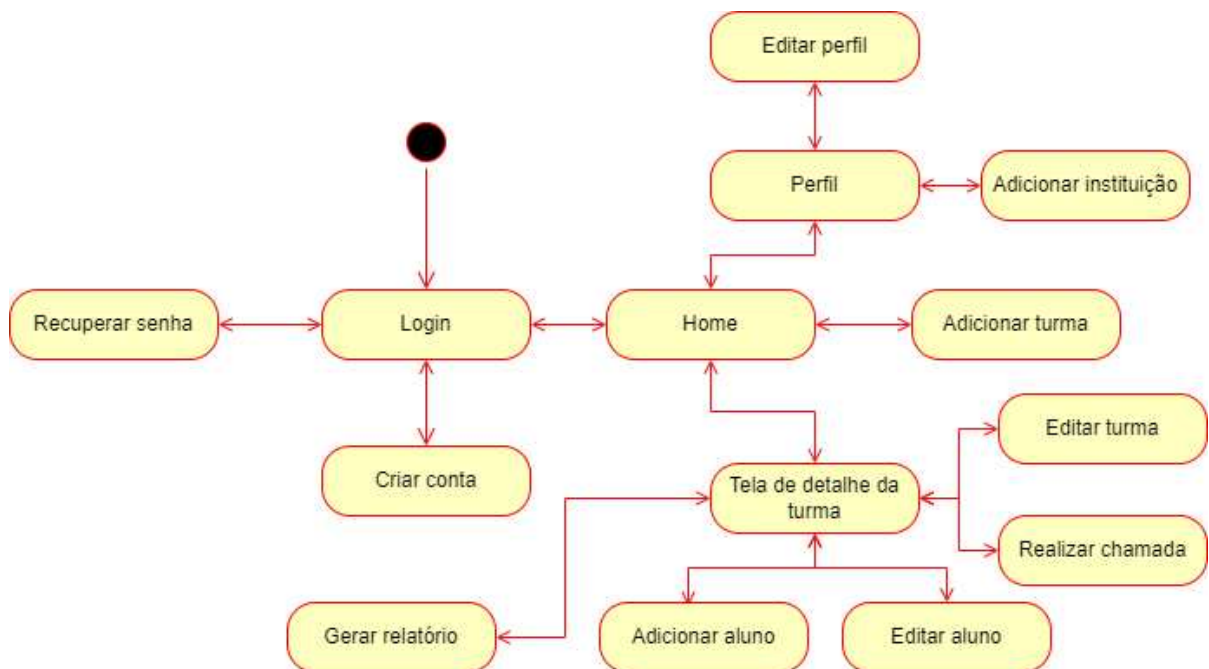
### 5.2.3.3 Diagrama de navegação

O conceito de diagrama de navegação, segundo a IBM, é:

"Um diagrama de navegação é um diagrama temporário não editável que mostra os resultados de uma consulta em um elemento do diagrama de contexto. Você pode utilizar os diagramas de navegação para navegar por elementos e relacionamentos em um projeto. Por exemplo, você pode criar um diagrama de navegação para mostrar uma visualização dinâmica de uma classe e seus elementos relacionados para entender como ela se ajusta a toda a estrutura do projeto."(IBM, 2021)

Este diagrama mostra de cima a navegação entre telas no aplicativo, com o intuito de deixá-la simples de entender.

Figura 26 – Diagrama de Navegação do aplicativo.



**Fonte:** Autoria própria (2022)

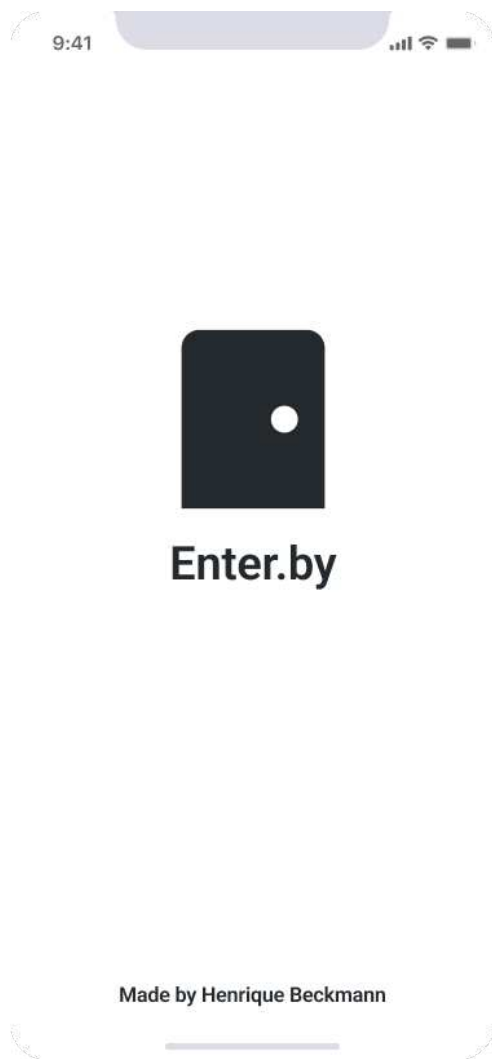
## 5.2.4 Protótipos de telas

Os protótipos das telas serão usados como referência, e apenas referência, para o desenvolvimento da interface, não limitando o desenvolvedor, para caso um fluxo melhor seja encontrado, novos componentes de interface ou um layout diferente, o desenvolvedor esteja livre para fazer as alterações.

### 5.2.4.1 Telas iniciais do aplicativo

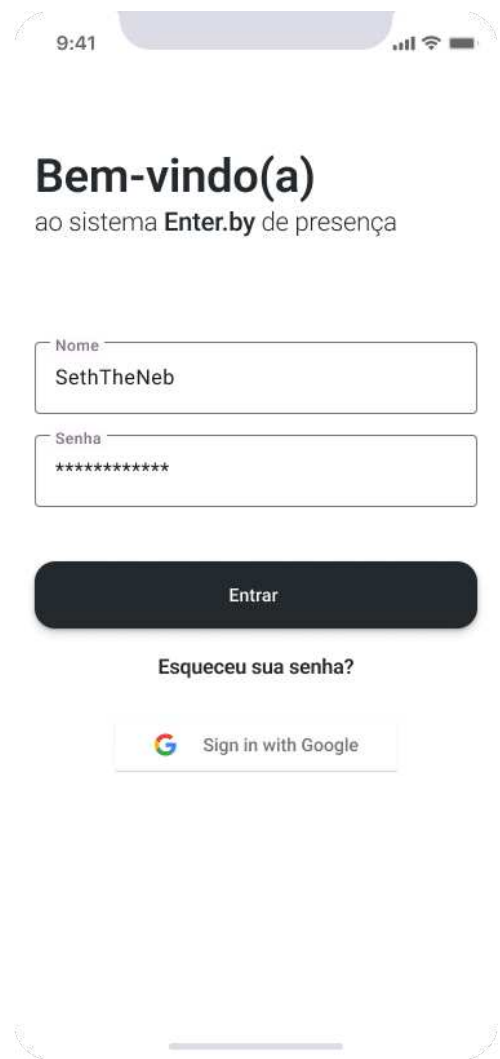
O nome “Enter.by” é apenas fictício, sendo necessário sua mudança no processo de desenvolvimento. As telas abaixo são respectivamente, a tela que o usuário vê no instante em que abre o aplicativo, e a tela de login, que serve para autenticação.

Figura 27 – Splash screen do aplicativo.



Fonte: Autoria própria (2022)

Figura 28 – Tela de login do aplicativo.

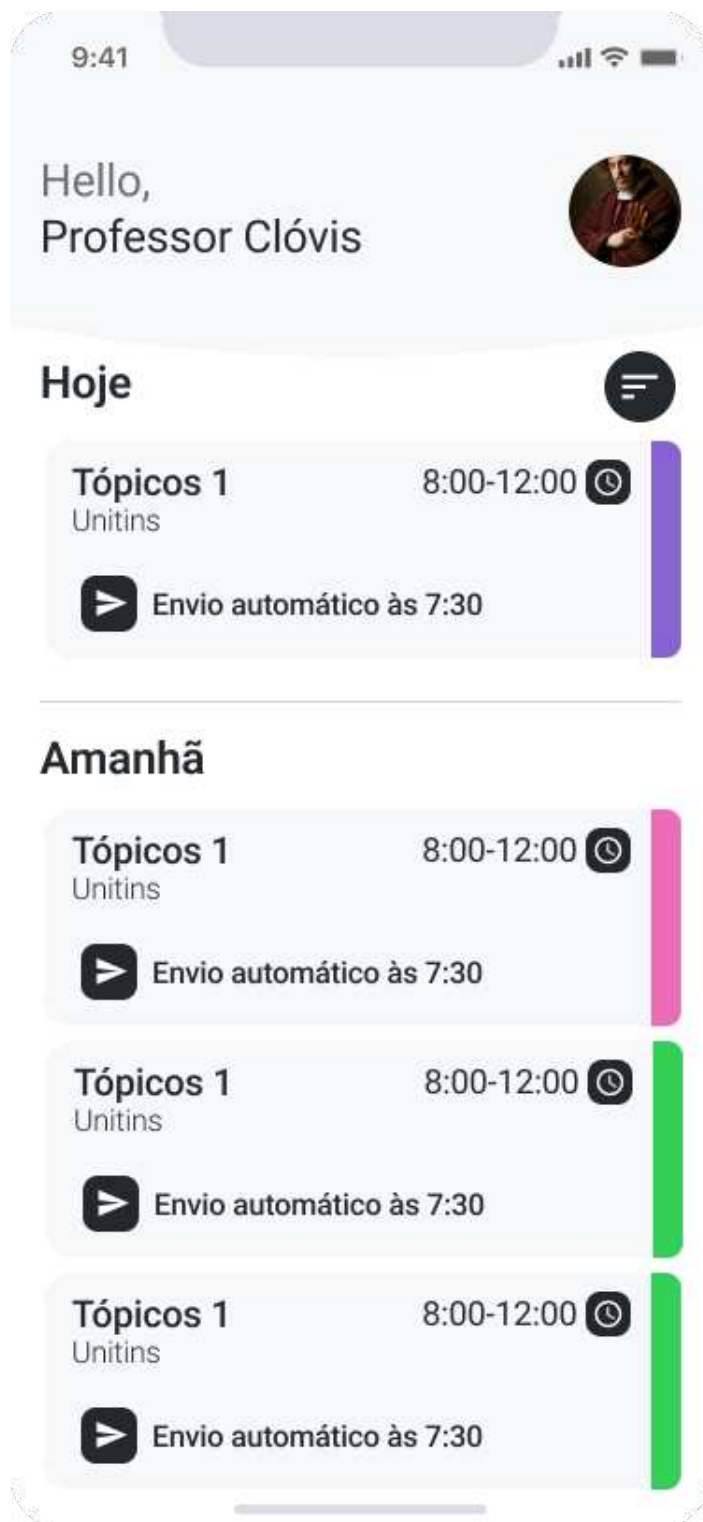


Fonte: Autoria própria (2022)

#### 5.2.4.2 Tela principal do aplicativo

Após o login, o professor será redirecionado para esta tela, que exibe as turmas onde ele ministra, onde ele pode acessá-las para a realização da frequência, adição de alunos, edição da turma ou consultar alguma informação.

Figura 29 – Tela inicial do aplicativo.



Fonte: Autoria própria (2022)



#### 5.2.4.3 Tela de detalhes de uma turma

Mostra os detalhes de uma turma, como por exemplo, a quantidade de faltas de um determinado aluno, nessa tela ele tem acesso ao recurso de realização da frequência.

Figura 30 – Tela de detalhes de uma turma.

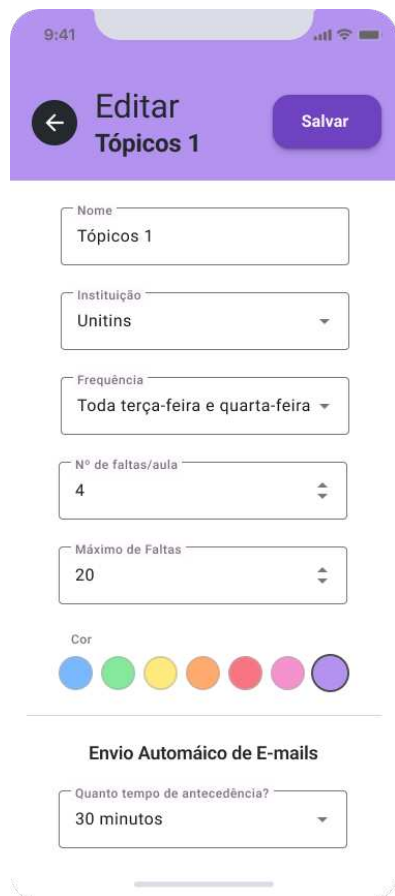


**Fonte:** Autoria própria (2022)

#### 5.2.4.4 Telas de edição

As seguintes telas são as de edição, onde o usuário pode alterar as informações de uma turma, aluno ou professor, respectivamente.

Figura 31 – Tela de edição de turma.

A interface de edição de turma apresenta um cabeçalho com o título "Editar Tópicos 1" e um botão "Salvar". O formulário contém campos para: Nome (Tópicos 1), Instituição (Unitins), Frequência (Toda terça-feira e quarta-feira), Nº de faltas/aula (4), Máximo de Faltas (20), Cor (seleção de cores) e Envio Automático de E-mails (Quanto tempo de antecedência? 30 minutos).


**Fonte:** Autoria própria (2022)

Figura 32 – Modal para edição das informações de um aluno.

O modal de edição de aluno possui o título "Editar" e campos para Nome (Marcos) e E-mail (marcos@email.com). No rodapé, há dois botões: "Salvar" e "Remover".

**Fonte:** Autoria própria (2022)

Figura 33 – Tela de perfil.

A tela de perfil exibe o nome "Professor Clóvis", o e-mail "clovispro@email.com" e a opção "Receber Notificações" (ativada). Um botão "Mudar Senha" está disponível no rodapé.

**Fonte:** Autoria própria (2022)

#### 5.2.4.5 Tela para a realização da chamada

A tela onde o professor realiza a chamada na sala de aula, após a leitura de todos os *QrCodes*, a requisição é enviada para a *API*, para que sejam registradas devidamente tanto as presenças quanto as faltas.

Figura 34 – Tela de realização de chamada.



Fonte: Autoria própria (2022)

## 5.3 Testando a qualidade do código

Esses testes foram feitos por meio do *software SonarQube*, que é definido como:

"O SonarQube é uma ferramenta de revisão automática de código autogerenciada que sistematicamente ajuda você a fornecer código limpo.[...]."  
([SONARQUBE.ORG](https://sonarqube.org), 2008)

O *SonarQube* analisa o código e dá uma nota em suas categorias. A análise feita durou em média 3 minutos, sendo que ele divide as análises em categorias e lhes dá uma nota, que vai de "A" a "E", sendo "A" a maior e "E" a menor. Seus resultados foram:

### 5.3.1 Testando a *API*

Tabela 6 – Validação de qualidade de código - *API*

Atributo	Nota	Número de atribuições
Bugs	C	11
Vulnerabilities	E	1
Hotspot review	E	0%
Code smells	A	48
Duplication	A	0%

**Fonte:** Autoria Própria (2023).

O projeto foi aprovado no teste, após análise dos resultados, foi descoberto que a todos os *bugs* estão localizados na pasta "*node\_modules*", que é uma pasta de códigos auto-gerados pelo próprio node, de acordo com os *packages* que são adicionados no projeto, ou seja, isso foge da competência de quem desenvolve a aplicação, o mesmo aconteceu com *Hotspots review*.

*Code smells* é referente à trechos de códigos que não estão da melhor maneira, mas considerando a nota, não é nada que afetaria a performance da *API*, são coisas simples e o próprio *SonarQube* já fornece uma possível solução, como trocar *var* por *let* ou *final*.

*Vulnerabilities* são as vulnerabilidades do sistema, a encontrada é por conta da chave de acesso ao *firebase* não estar protegida, isso pode ser resolvido com a criação de variáveis de ambiente, como o arquivo *.env* por exemplo, onde ele guarda as variáveis locais, e não pode ser compartilhado.

Não foram encontrados códigos duplicados no projeto.

### 5.3.2 Testando o estudo de caso

Tabela 7 – Validação de qualidade de código - Tela

Atributo	Nota	Número de atribuições
Bugs	C	1
Vulnerabilities	A	0
Hotspot review	E	0%
Code smells	A	5
Duplication	A	0%

**Fonte:** Autoria Própria (2023).

O código gerado pelo *ChatGPT*, como esperado, foi aprovado no teste, e as indicações de problemas são de códigos gerados automaticamente pelo próprio *flutter*, durante a criação do projeto, demonstrando que o código "escrito" pelo *ChatGPT* é limpo.

## 6 Conclusão

Este trabalho de conclusão de curso tem como objetivo a criação de uma solução alternativa para realizar frequência em uma turma ou grupo utilizando *QrCodes* como método de verificação, mostrando uma pesquisa bibliográfica e exibindo o processo metodológico utilizado para chegar nos resultados apresentados.

Houve uma conversão da ideia original, que era um simples aplicativo totalmente voltado para a área acadêmica, neste trabalho isso foi convertido para um serviço, uma *REST API*, que faz a interação com o *front-end* através de requisições *http*, com essa mudança foi necessária a reestruturação da proposta assim como pesquisa para que o resultado fosse obtido.

Durante essas pesquisas, foram decididas as tecnologias que foram utilizadas e o padrão da *API*, visando a menor taxa de modificações necessárias em uma aplicação que em comparação com a escolha de uma conexão direta com um banco de dados.

Dentro dos objetivos, a criação do serviço foi realizada com sucesso, que beneficiou o processo de frequência escolar deixando-o mais ágil, e após a implementação do serviço, um estudo de caso foi feito para a validação da usabilidade da *API*, assim como testes de qualidade de código para obtenção de métricas. Levando isso em consideração, foi concluído que é viável a utilização dessa *API* nesse tipo de aplicação.

### 6.1 Trabalhos futuros

Sugestões de próximos passos a serem tomados, listados.

- Integrar a geração de *QrCodes* com a *API*.
- Projetar a geração de relatórios e sua requisição.
- Hospedagem do serviço.
- Correção de possíveis *bugs* e realizar testes unitários.

# Referências

- APPS, B. *Toaqui - Controle de Presença / Frequência*. 2022. [Online; accessed 10-abril-2022]. Disponível em: <[https://play.google.com/store/apps/details?id=com.bsm.chamada&hl=pt\\_BR&gl=US](https://play.google.com/store/apps/details?id=com.bsm.chamada&hl=pt_BR&gl=US)>.
- BARBOSA, U.; NEVES, G. *Doity - App de check-in para eventos*. 2013. [Online; last accessed 21-maio-2022]. Disponível em: <<https://doity.com.br/blog/app-de-check-in-para-eventos/>>.
- BARRETOM, G. G. *Assinar listas de presença é coisa do passado*. 2022. [Online; last accessed 20-maio-2022]. Disponível em: <<https://www.serpro.gov.br/menu/noticias/noticias-2020/assinar-listas-presenca-ficou-no-passado>>.
- Brasil. *Lei de Diretrizes e Bases da Educação Nacional*. 1996. Brasília. Lei nº 9.394/1996.
- DEVELOPERS, G. for. *Firestore*. 2023. [Online; accessed 23-maio-2023]. Disponível em: <<https://firebase.google.com/docs/firestore?hl=pt-br>>.
- DINIZ, Y. *Entenda a importância de acompanhar a frequência escolar de perto*. 2020. [Online; accessed 22-june-2022]. Disponível em: <<https://educacao.imagine.com.br/frequencia-escolar/#:~:text=O%20%C3%ADndice%20de%20frequ%C3%Aancia%20escolar,alunos%20t%C3%AAm%20faltado%20%C3%A0s%20aulas.>>>.
- FOUNDATION, G. *GraphQL*. 2023. [Online; accessed 20-maio-2023]. Disponível em: <<https://graphql.org/>>.
- FOUNDATION, O. *Express*. 2023. [Online; accessed 25-maio-2023]. Disponível em: <<https://expressjs.com/>>.
- FOUNDATION, O. *Node.js, sobre*. 2023. [Online; accessed 25-maio-2023]. Disponível em: <<https://nodejs.org/pt-br/about>>.
- GUEDES, G. T. *UML 2-Uma abordagem prática*. [S.l.]: Novatec Editora, 2018.
- IBM. *Diagramas de Navegação*. 2021. [Online; accessed 22-june-2022]. Disponível em: <<https://www.ibm.com/docs/pt-br/rsas/7.5.0?topic=models-browse-diagrams>>.
- KANADE, T.; LUCAS, B. *An Iterative Image Registration Technique with an Application to Stereo Vision*. 1981. [Online; last accessed 21-maio-2022]. Disponível em: <[https://www.ri.cmu.edu/pub\\_files/pub3/lucas\\_bruce\\_d\\_1981\\_2/lucas\\_bruce\\_d\\_1981\\_2.pdf](https://www.ri.cmu.edu/pub_files/pub3/lucas_bruce_d_1981_2/lucas_bruce_d_1981_2.pdf)>.
- OPENAI. What is chatgpt. [Online; accessed 31-maio-2023]. 2023. Disponível em: <<https://openai.com/>>.
- SILVA, F. *Impressões digitais: por quê somos únicos?* [S.l.], 2022. [Online; last accessed 21-maio-2022]. Disponível em: <<https://www.ufmg.br/espacodoconhecimento/impressoes-digitais-por-que-somos-unicos/>>.
- SONARQUBE.ORG. Sonarqube documentation. [Online; accessed 30-maio-2023]. 2008. Disponível em: <<https://docs.sonarqube.org/latest/>>.

STANDARDIZATION, I. O. for. *ISO/IEC/IEE 24765-2010: Systems and software engineering - vocabulary*. 2010.

STOJANOVIC, N. *Await loop vs Promise.all*. 2023. [Online; accessed 30-maio-2023]. Disponível em: <<https://stackoverflow.com/questions/53798589/await-loop-vs-promise-all>>.

TEAM, F. *Factorial - Controle de ponto online e registro de frequência*. 2022. [Online; last accessed 20-maio-2022]. Disponível em: <<https://factorialhr.com.br/controle-ponto>>.

VAZQUEZ EDUARDO, C.; SIMÕES SIQUEIRA, G. *Engenharia de Requisitos: software orientado a negócio*. [S.l.]: Editora Brasport, 2016.

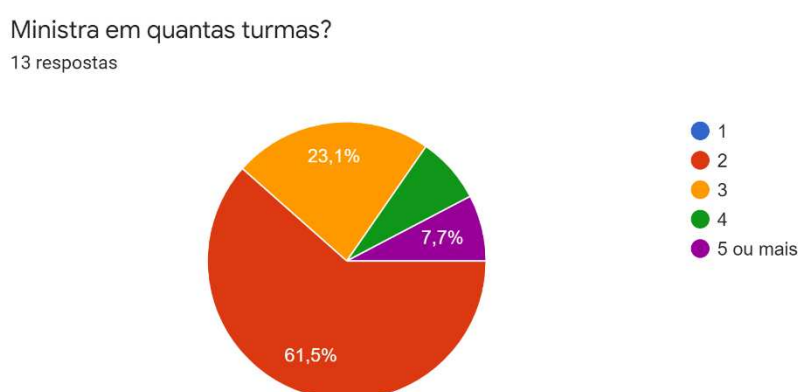


## 7 Apêndice

### 7.1 Pesquisa para validação do problema

Para a validação do problema e justificativa para a criação desse projeto, foi realizada uma pesquisa com professores.

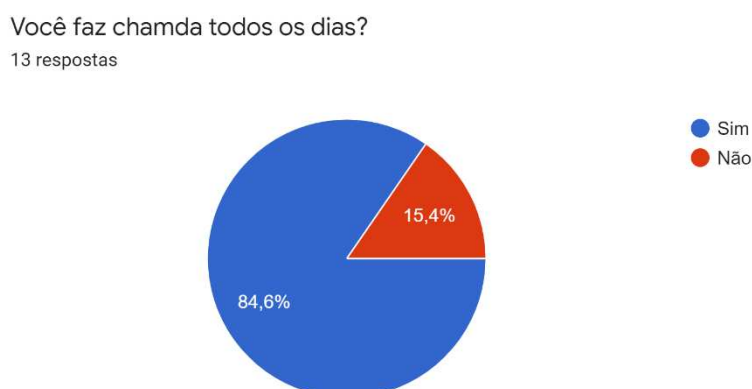
Figura 35 – Gráfico da pesquisa - quantidade de turmas que o docente ministra.



**Fonte:** Pesquisa do autor (2022)

Válido também mencionar que, de acordo com a pesquisa nem todos os professores realizam chamada todos os dias.

Figura 36 – Gráfico da pesquisa – Frequência da realização da chamada.



**Fonte:** Pesquisa do autor (2022)

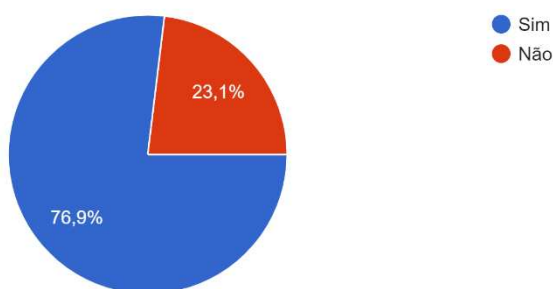
A pesquisa indicou que os professores geralmente precisam repetir o nome de um aluno até que ele ouça, gerando atrasos no processo, também existe a possibilidade de

existirem dois alunos com nomes iguais, devido a quantidade de alunos por turma, que é um fator comum entre todos os entrevistados, como mostram os gráficos abaixo:

Figura 37 – Gráfico da pesquisa – Repetir nome do aluno.

Normalmente é necessário repetir o nome de um aluno até que ele responda?

13 respostas

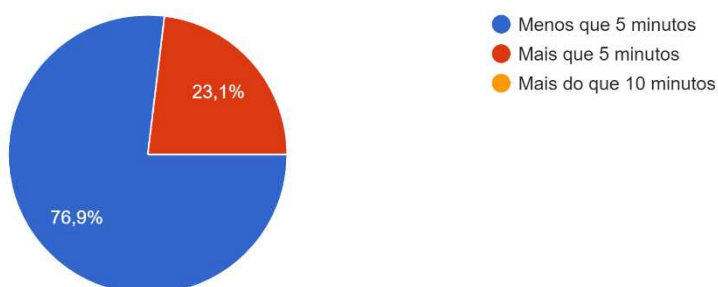


**Fonte:** Pesquisa do autor (2022)

Figura 38 – Gráfico da pesquisa - Tempo médio da duração da chamada.

Tempo médio para a realização da chamada

13 respostas

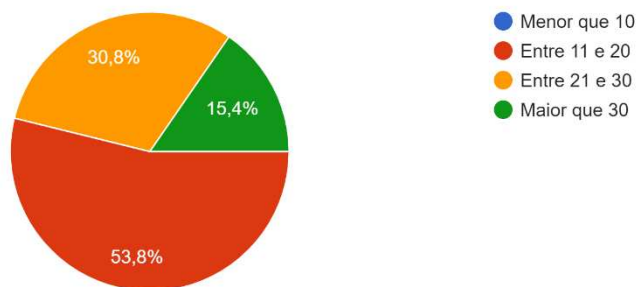


**Fonte:** Pesquisa do autor (2022)

Figura 39 – Gráfico da pesquisa – Quantidade de alunos por turma.

Média de quantos alunos em uma turma?

13 respostas



**Fonte:** Pesquisa do autor (2022)