Carinho Escolar: Um Sistema Gratuito de Controle de Entradas de Estudantes para Pré-escolas e Escolas

Matheus Gabriel de Andrade[1], Alexandre Grotta[1] (Orientador)

1 Instituto Federal de Educação, Ciência e Tecnologia - Campus Hortolândia (IFSP)
CEP: 13183-250 – Hortolândia – SP – Brasil

matheusandrade193@gmail.com, grotta@ifsp.edu.br

Abstract. *The national context has challenges regarding the children safety in school's environment. One of the most relevant is the security during school entry and exit periods, in which, for example, unauthorized persons can pose as responsible for them. For this challenge, in this research, the system Carinho Escolar, an acronym for Registry for Humanized and Optimized Individual Analysis and Recognition, was developed. The main goal was to create a free system to help school monitors and teachers to control the entry and exit of students, through students' and their responsible person data: photos and personal data. Carinho Escolar is a web system, therefore accessible from smartphones or computers with internet or schools' internal networks. As a result of the research, the system was developed according to their goals, and it received positive feedback from end users. This system may contribute to the schools' safety in future phases of this research.*

Resumo. Há no contexto nacional desafios à segurança das crianças em ambiente escolar. Uma das mais relevantes é segurança nos períodos de entrada e saída da escola, na qual, por exemplo, pessoas não autorizadas podem se passar por responsáveis. Para esse desafio, nesta pesquisa foi criado e desenvolvido o sistema *Carinho Escolar*, acrônimo para Cadastro para Análise e Reconhecimento Individual Humanizado e Otimizado. O objetivo principal foi criar um sistema gratuito, para auxiliar monitores e professores a controlarem a entrada e saída de alunos, por meio de fotos e dados pessoais de responsáveis e alunos. Carinho Escolar é um sistema *web*, portanto acessível de um *smartphones* ou computador que tenha acesso à internet ou redes internas das escolas. Como resultado da pesquisa, o sistema foi desenvolvido de acordo com os seus objetivos e obteve *feedback* positivo de usuários finais. Este sistema possui potencial para contribuir com a segurança de escolas, cuja implantação planeja-se ocorrer em fases futuras da pesquisa.

1. Introdução

De acordo com Atlas da Violência, o índice de violência contra crianças e adolescentes é alto no contexto nacional [IPEA 2019]. Dentre os problemas de violência, no contexto escolar destaca-se os casos relacionados ao período de entrada e saída nas escolas, tais como o relatado pela imprensa, no qual um casal de desconhecidos conseguiu se passar como responsáveis da menina e facilmente levá-la da escola [GLOBO 2019]. Por outro lado, não foram encontrados sistemas gratuitos e que possam ser acessados facilmente de celulares e computadores disponíveis para tais escolas.

Visando contribuir para a solução deste relevante desafio, essa pesquisa tem por objetivo principal criar um sistema *web*, portanto facilmente acessível a partir de computadores e celulares. Este sistema foi nomeado *Carinho Escolar*, por ser acrônimo para Cadastro para Análise e Reconhecimento Individual Humanizado e Otimizado, e por

visar um olhar com Carinho para este horário da jornada escolar em que as crianças podem ficar mais vulneráveis ao acesso de pessoas não autorizadas.

O objetivo principal foi criar um sistema gratuito, para auxiliar monitores e professores a controlarem a entrada e saída de alunos, por meio de fotos e dados pessoais de responsáveis e alunos. Carinho Escolar é um sistema *web*, portanto acessível de um *smarthphones* ou computador que tenha acesso à internet ou redes internas das escolas. Deste objetivo principal, os objetivos específicos do sistema foram estabelecidos como:

I. Criar um sistema para controle digital de entrada e saída de alunos, cuja validação ocorre por meio de professores e monitores escolares, que se apoiam em dados cadastrais e fotografias da criança e dos responsáveis;

II. Gerenciar turmas de alunos

III. Emitir relatórios de entrada e saída de alunos;

IV. Criar perfis diferentes de acesso para monitores de alunos, professores e direção escolar (administrador);

V. Permitir que esse sistema seja acessível para mais de uma escola;

2. Referencial Teórico

Da subseção 2.1 até a subseção 2.18, são detalhados conceitos e tecnologias utilizados para o desenvolvimento desta pesquisa. A subseção 2.19 apresenta a pesquisa de estudos e sistemas correlatos.

2.1. Máquina Virtual

Máquina virtual é um sistema computacional com os seguintes recursos: memória, CPU, armazenamentos e interface de rede. É utilizado recursos de uma máquina física, apresentando a característica de ser isoladas do restante do sistema principal. Além de permitir o uso de diferentes sistemas operacionais executados simultaneamente no mesmo computador como por exemplo MacOS e Linux [RedHat 2021].

2.2. Computação em Nuvem

Computação em nuvem provêm diversos serviços de computação incluindo servidores, banco de dados e *softwares* utilizando internet, oferecendo recursos flexíveis e escaláveis pagando somente pelos serviços utilizados, ajudando a reduzir custos operacionais e escalar recursos conforme sua necessidade. [Microsoft Azure 2021].

2.3. API

Uma API é um conjunto de definições e protocolos para construir e integre um *software*. Às vezes, é chamado de contrato entre um provedor de informações e um usuário de informações, estabelecendo o conteúdo exigido do consumidor (a chamada) e o conteúdo exigido pelo produtor (a resposta). Em outras palavras, se você deseja interagir com um computador ou sistema para recuperar informações ou executar uma função, uma API o ajuda a comunicar o que deseja a esse sistema para que ele possa entender e atender à solicitação, uma variação e chamada de API *Rest* (*Representational State Transfer*) que são boas práticas de desenvolvimento de API para *Web* [RedHat 2022].

2.4. Arquitetura MVC

É um padrão de arquitetura que divide a aplicação em três camadas (*Model-View-Controller)*: A primeira camada cuida da manipulação logica dos dados *Model*, a segunda

é a interface que o usuário interage *View* e a última por fim, é responsável pelo fluxo da aplicação e está entre a camada do *View* e o *Controller* [GSTI 2022].

2.5. Banco de Dados

Um banco de dados tem como finalidade organizar dados ou informações, geralmente sendo gerenciado por um SGBD (Sistema de Gerenciamento de Banco de Dados). A maioria dos bancos de dados utiliza a linguagem SQL (Linguagem de Consulta Estruturada) para consultar e editar dados [Oracle 2021].

2.6. HTML

HTML é uma linguagem de marcação de hipertexto, "hipertexto refere-se aos links que conectam páginas da Web entre si, seja dentro de um único site ou entre sites. Links são um aspecto fundamental da web. Ao carregar conteúdo na Internet e vinculá-lo a páginas criadas por outras pessoas, você se torna um participante ativo no *World Wide Web*. O HTML usa "Marcação" para anotar texto, imagem e outros conteúdos para exibição em um navegador" [Mozilla Foundation 2020].

2.7. JavaScript

JavaScript é uma linguagem leve, interpretada conhecida como uma linguagem de script para páginas Web, suporta paradigma orientado a objetos, programação funcional e pode ser utilizado também no conhecido node.js [Mozilla Foundation 2020].

2.8. Java

Java é uma linguagem de programação e plataforma computacional criado pela Sun Microsystems em 1995. Existem muitas aplicações e sites que não funcionarão, a menos que você tenha o Java instalado, e mais desses são criados todos os dias. O Java é rápido, seguro e confiável [Oracle 2020].

2.9. Spring Boot

O Spring Framework facilita a programação e modelo de configuração de para modernas aplicações empresariais baseadas em Java. Spring é para que programadores foquem na camada de lógica de negócio, sem se preocuparem com específico ambiente de programação. Spring boot torna fácil criar aplicações *standalone* para poder somente rodar sem se preocupar com configurações de ambiente [Spring 2020].

2.10. Hibernate

É uma biblioteca que tem como finalidade o mapeamento de objetos Java para Relacionamento de dados SQL, além disso facilita a consulta de dados, diminui o tempo de desenvolvimento do banco de dados, por não precisar utilizar operações manuais para criações das tabelas e relacionamento utilizando comandos SQL [DevMedia 2021].

2.11. Hibernate Envers

Hibernate Envers é um modulo da biblioteca Hibernate que permite auditar classes pré-selecionadas através de um controle de versionamento baseado no mapeamento objeto/relacionamento do Hibernate. Tem como vantagem poder realizar a auditoria independente do banco de dados, aumentando a produtividade e reduzindo o custo de manutenção [Oracle 2021].

2.12. MariaDB

MariaDB é um dos mais populares servidores de banco de dados relacionais no mundo. Foi feito pelos desenvolvedores originais do MySQL. MariaDB é usado porque é rápido, escalável e robusto e com um rico ecossistema de engenharia de armazenamento, plugins e muitas outras ferramentas que fazem dele muito versátil para uma larga variedade de casos de usos [MariaDB 2019].

2.13. Git Hub

Git Hub é um serviço de nuvem que permite a hospedagem de arquivos de sistema de versionamento em formato GIT, permitindo que desenvolvedores possam alterar código em projetos compartilhados com registro de progresso [Hostinger 2021].

2.14. SourceTree

É uma ferramenta de interface gráfica de usuário para *desktop* que simplifica como você interage com repositórios GIT, permitindo focar totalmente em codificar. Aumentando sua velocidade e produtividade pela interface amigável [Globallogic 2021].

2.15. Bulma

Bulma é um *framework* CSS baseado em Flexbox, leve e configurável. Bulma é projetado para ser amigável tanto com *mobile* e *desktop* sendo responsivo, fácil de aprender e com uma sintaxe simples tornando o código simples de ler e escrever [Jeremy Thomas 2018].

2.16. Amazon Simple Storage Service (Amazon S3)

Amazon Simple Storage Service (Amazon S3) é um serviço de armazenamento de dados flexível que armazena aplicações *web*, arquivos de *backup*, *logs* e customizar receptivos acessos em seu servidor, e permitir acessar de qualquer lugar por utilizar tecnologia de nuvem [Amazon AWS 2021].

2.17. Postman

Postman é Plataforma de desenvolvimento e testes para APIs, podendo ser utilizado para criar especificação de APIs, documentação e fluxos de trabalhos e testes, por exemplo, testes de rotas HTTP não suportados em um navegador *web* convencional (POST, PUT, DELETE) [Postman 2021].

2.18. Escala de Likerd

É uma escala para o indivíduo expressar o quanto concorda ou discorda de uma determinada afirmação. Permitindo que os respondentes indiquem, sua força de concordância positiva a negativa ou força de sentimento em relação à pergunta ou afirmação [SimplyPsychology 2022]

2.19. Trabalhos Correlatos

Até o limite de conhecimento desta pesquisa, foi encontrado apenas um sistema similar e correlato, o *Sistema Escolar 3.0* da SoftPontes . Este é sistema comercial e não gratuito, brasileiro, para controle escolar. É baseado em Java, e é um sistema "completo" para a gestão instituições de ensino, não somente entrar e saída de alunos [Pontes 2020].

Apenas para ilustração de um sistema Java *desktop*, conforme Figura 1, é uma solução que requer mais recursos computacionais e não é acessível a partir de *smarthphones*.

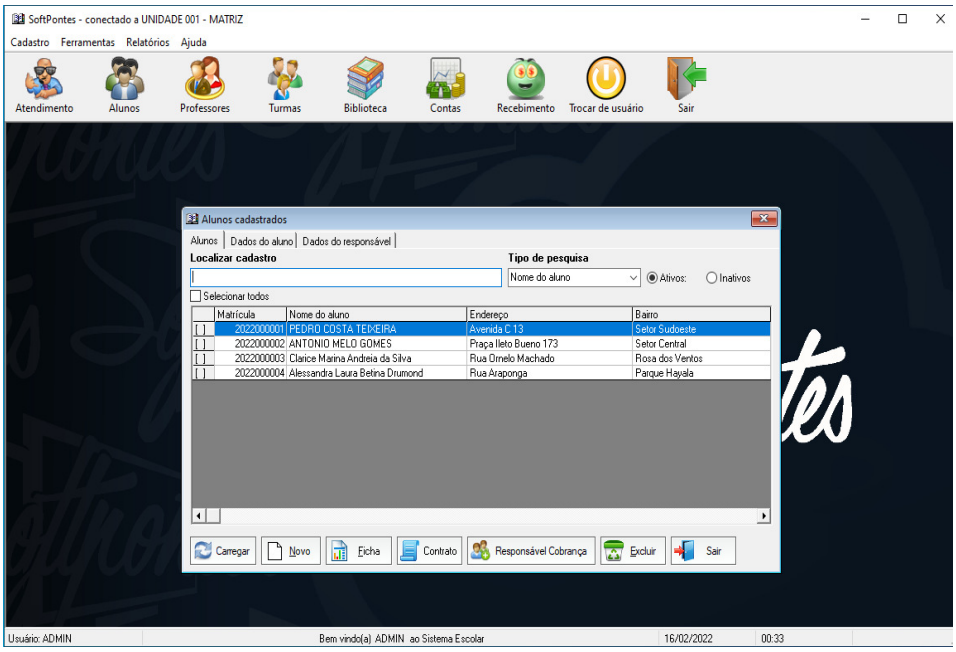

Figura 1. Tela de Cadastro de Alunos
Fonte: Sistema Escolar 3.0

Assim, por meio da Figura 2, pode-se comparar os principais requisitos funcionais daquele sistema com o sistema Carinho Escolar. Já sobre requisitos não funcionais de cada sistema: Carinho Escolar possui um layout otimizado para dispositivos móveis, e é de utilização gratuita enquanto Sistema Escolar 3.0 não possui essas características. Ambos os sistemas possuem criptografia com credenciais de acesso para acessar o sistema (usuário e senha).

REQUISITOS FUNCIONAIS	CARINHO ESCOLAR	SISTEMA ESCOLAR 3.0
GERENCIAMENTO TURMAS	✅	✅
GERENCIAMENTO ALUNOS	✅	✅
GERENCIAMENTO DE RESPONSÁVEIS	✅	✅
EXPORTAR RELATORIOS	✅	✅
FOLHA DE PAGAMENTO	❌	✅
PERFIL DE USUÁRIO	✅	✅
RELATÓRIOS	✅	✅

Figura 2. Tabela comparativa de requisitos funcionais de cada sistema

Fonte: Elaborado pelo Autor

Até o limite desta pesquisa não foram encontrados outros *softwares* similares por razão de que a maioria dos sistemas serem pagos. O principal motivo é que a pesquisa não obteve acesso aos aplicativos após contato comercial e agendamento direcionado às instituições de ensino.

3. Metodologia

Foi escolhido como estratégia de pesquisa um estudo de caso, pois a essência é tentar sanar uma decisão ou conjunto de decisões, utilizando a investigação empírica com várias fontes de dados. Portanto trata-se de uma pesquisa descritiva com abordagem de análise qualitativa [Yin 2001]. Foi planejado entrar em contato com escolas e creches da região

metropolitana de Campinas para avaliação do sistema em um ambiente real. Porém devido ao afastamento social, causado pelo surto de COVID-19 não foi possível realizar o estudo. O estudo, no entanto, teve um grande avanço e esta avaliação foi substituída por algumas avaliações remotas do sistema.

A metodologia de desenvolvimento escolhida foi a DevOpsBL, que tem característica híbrida entre abordagem de aprendizagem por projeto e abordagem operacional [Grotta e Prado 2021]. Foi elaborado um plano de ação em conjunto do professor. Em seguida, o desenvolvimento foi feito pelo aluno e também compartilhados resultados de intermediário com demais em processo de orientação de pesquisa, para que cada aluno pudesse ter de ser suas percepções a respeito do desenvolvimento do sistema.

Após a criação de incrementos do sistema, era iniciada a parte de *Ops* (Operações) para se manter o sistema disponível enquanto as demais funcionalidades são desenvolvidas. Isto tornava possível apresentar os resultados parciais para professores e demais alunos. Em seguida, era feito o *Feedback* (Comentários) sobre o sistema para cada pesquisador, iniciado uma nova interação no ciclo da metodologia DevOpsBL, conforme a Figura 3.

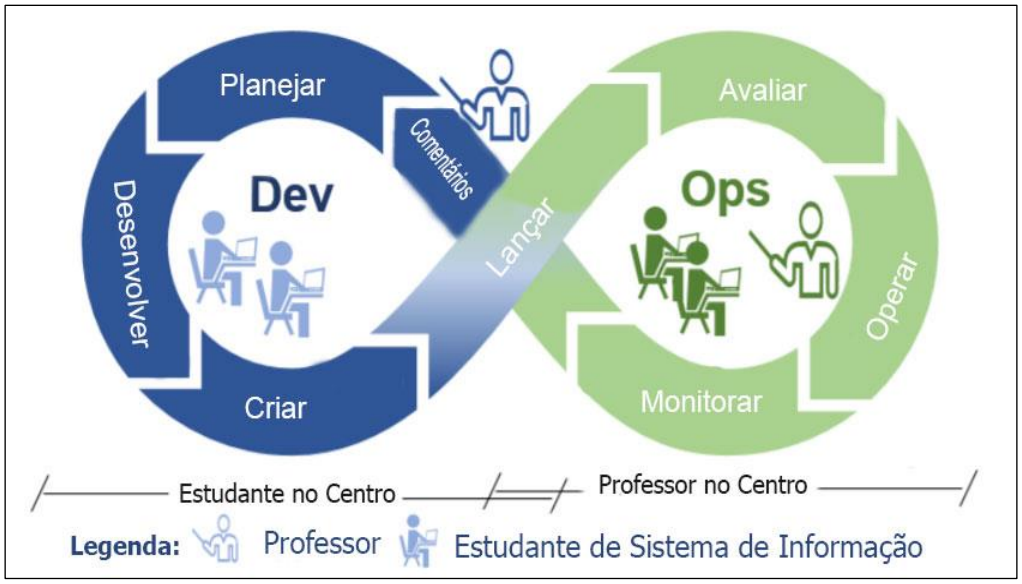

Figura 3. Visão Geral DevOpsBL (Adaptado de Grotta e Prado, 2021)

5. Desenvolvimento

Para o Desenvolvimento do sistema Carinho Escolar, foi escolhida a arquitetura *model-view-controller* (MVC). Nesta arquitetura para *back end* foi utilizado Java SE, framework Spring Boot e API *Rest*, para o Banco de Dados foi escolhido MariaDB. Por fim, para o *front end* foram escolhidas tecnologias *Web*, tais como HTML5, CSS e JavaScript com chamadas às APIs *Rest*, para testar a API foi utilizado a plataforma Postman e por fim, para o versionamento de código foi utilizado o *software* SourceTree. Foram escolhidas tecnologias que têm por objetivo permitir, caso necessário, que o sistema Carinho Escolar possa ser utilizado de forma *standalone,* ou seja, é possível ser acessado a partir de dispositivos mais simples, que muitas vezes é o caso de escolas e funcionários de certas escolas.

5.1 Caso de Uso

Após a concepção da ideia (fase de "ideação") do Carinho Escolar, foram detalhados os casos de uso. A Figura 4 é um diagrama geral de casos de uso e foi elaborado utilizando uma plataforma gratuita. Nesta figura, há três atores o monitor, administrador e professor. Eles interagem com o sistema conforme destacado. É pré-condição para todos ter efetuado *login* no sistema.

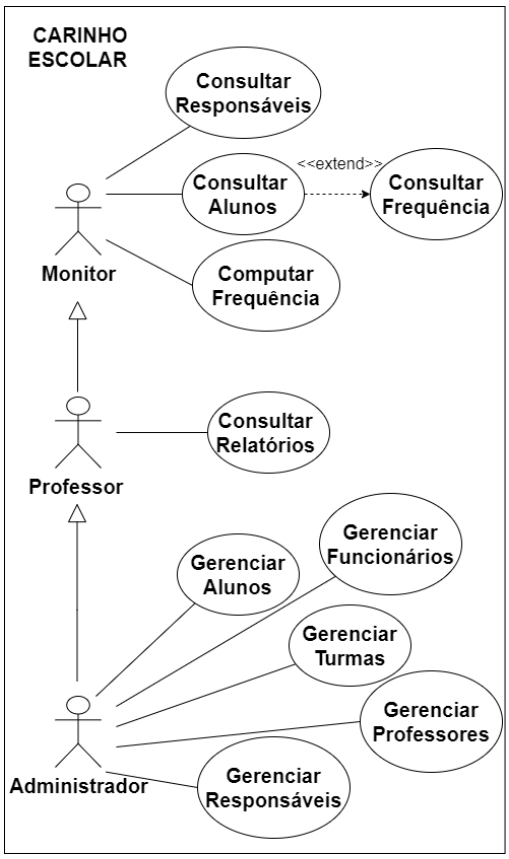

Figura 4. Diagrama de Caso de Uso

Fonte: Elaborado pelo Autor

A seguir são detalhados os principais fluxos de caso de uso, conforme Figura 5 é descrito o caso de uso Gerenciar Alunos que o Administrador executa.

Caso de Uso: Gerenciar Alunos
Ator: Administrador
Descrição: Gerenciar alunos, realizar todas as operações CRUD (Criar, Ler, Atualizar, Deletar)

Figura 5. Fluxo Casos de Uso – Gerenciar Alunos

Na Figura 6 é descrito o caso de uso Consultar Alunos que o Monitor executa.

Caso de Uso: Consultar Alunos
Ator: Monitor
Descrição: Pesquisar e consultar dados de um aluno, também consultar a frequência do aluno (histórico de entrada e saída)

Figura 6. Fluxo Casos de Uso – Consultar Alunos

5.2 Requisitos

A Figura 7 apresenta os requisitos funcionais e não funcionais do Carinho Escolar, com os principais requisitos funcionais e não funcionais do *software*:

Requisitos Funcionais	Requisitos Não Funcionais
Controle de Entrada e Saída	Criptografia
Gerenciamentos de Alunos	Integração com API REST
Gerenciamentos de Responsáveis	Integração com Bancos de Dados SQL
Gerenciamentos de Turmas	Back End Desenvolvido em Java
Perfil de Usuários	Front End Desenvolvido em Java Script
Relatórios	
Auditoria	

Figura 7. Comparativo Requisitos Funcionais e Não Funcionais do Carinho Escolar

5.3 Banco de Dados

O banco de dados foi criado com o *framework* Hibernate, utilizando a anotação *@entity* conforme a Figura 8 é feito um mapeamento da classe Java, criando uma tabela no SGBD, o que é chamado de Entidade.

```
1.  package com.ifsp.hto.carinho.backend.model;
2.
3.  import java.io.Serializable;
4.
5.  @Entity
6.  public class Aluno implements Serializable {
```

Figura 8 – Exemplo de utilização da anotação Hibernate

As principais tabelas do banco de dados é a tabela *aluno* que tem os todos os atributos pertinentes aos dados pessoais do aluno, segundo pela tabela *responsável* que armazena dados pessoais do responsável. Sendo primordial para o monitor de alunos consultar a legitimidade do responsável.

Em continuação, a Figura 9 apresenta a modelagem do banco de dados do Carinho Escolar.

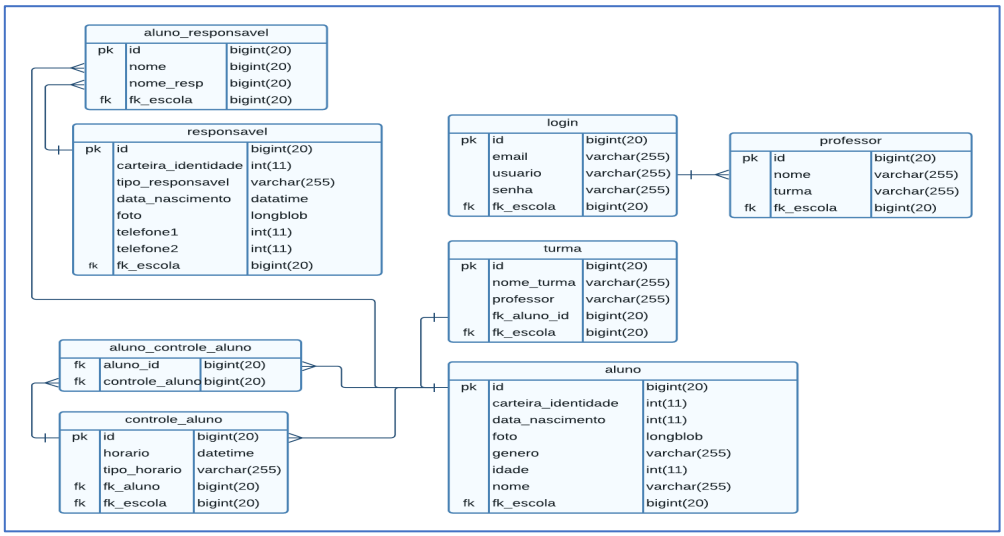

Figura 9. Diagrama Banco de Dados (MER)

Fonte: Elaborado pelo Autor

5.4 Estrutura de Diretórios

No desenvolvimento do *front end* do projeto foi utilizado uma estrutura agrupada, que cada página do sistema possui uma pasta e os arquivos do projeto seguem esse padrão: index.html, script.js e style.css, conforme a Figura 10.

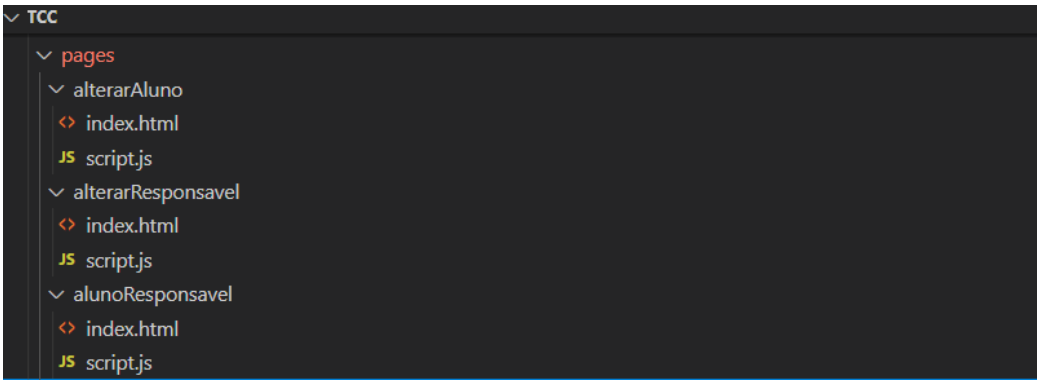

Figura 10. Estrutura de Páginas no *front end*

A estrutura de diretórios no Java é composta por pacotes, cada um contendo um agrupamento de classes Java por finalidade, conforme Figura 11.

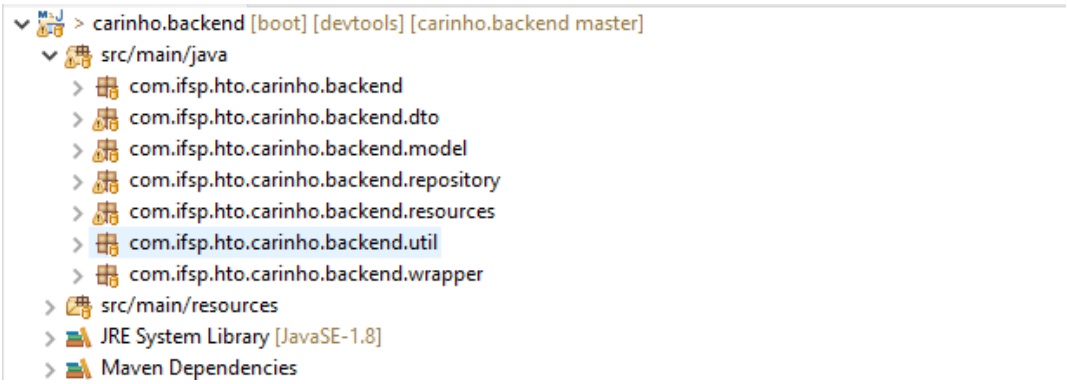

Figura 11. Estrutura de Páginas no *back end*

5.5 Desenvolvimento API

Para o desenvolvimento da API foi utilizado o *framework* Spring Boot, que disponibiliza uma API robusta em linguagem Java, podem de maneira simples, reduzindo o tempo de desenvolvimento. A seguir será descrito os principais métodos de API em HTTP: GET, POST, PUT, DELETE. Estes são utilizados na implementação do sistema Carinho Escolar e também a interceptação desses dados na linguagem JavaScript por meio da função *fetch*.

5.5.1 Métodos HTTP

A Figura 12 apresenta um trecho de código para função Get que tem como objetivo obter dados cadastrados no banco de dados. Para tal, é necessário utilizar a notação "@GetMapping" em cima do método Java. Nesse trecho é retornado uma lista de frequência dos alunos filtrados pelo parâmetro opcional {idTurma} e {idEscola} escritos na notação.

```
1.  @GetMapping("/frequencia/{idTurma}/{idEscola}")
2.      public List<FrequenciaDTO> listaFrequencia (@PathVariable(value = "
    idTurma")
3.      long idTurma,@PathVariable(value = "idEscola") long idEscola){
4.          return alunoRepository.listaFrequencia(idTurma,idEscola);
5.      }
```

Figura 12. Trecho código método GET

Em seguida, o método POST tem como função enviar dados do cliente para o servidor no trecho de código da Figura 13, por meio da notação "@PostMapping" em cima do método *professor,* que espera receber dois parâmetros *String nome* e *String turma*, na linha 3 é instanciado um objeto do tipo *professor* com esses parâmetros e na linha 4 é salvo o objeto *professor*.

```
1.  @PostMapping("/professor")
2.      public Professor salva(String nome, String turma )  {
3.      Professor professor = new Professor(nome,turma);
4.      return professorRepository.save(professor);
5.      }
```

Figura 13. Trecho código método POST

O Método PUT tem como função atualizar dados já existentes no banco de dados. Sendo assim, conforme trecho de código na Figura 14, é necessário utilizar a notação "@PutMapping" em cima do método conforme exemplo a seguir. A função do método é atualizar a dados de um responsável do aluno. O método tem o nome de "atualizar" e espera em seu retorno um objeto do tipo *Responsável*. Por parâmetro é passado os seguintes atributos: *nomeResp*, *carteiraIdentidade*, *telefone*, *tipoResponsavel*, *foto* e *idResponsavel*.

Nas linhas seguintes é criado um objeto do tipo *responsável* com esses parâmetros recebidos, vale salientar que para persistir dados do atributo *foto* é necessário extrair *Bytes* pois o formato *MultipartFile* fornecido pelo *framework* StringBoot não é suportado pelo Hibernate. Após criar esse objeto do tipo responsável é retornado o mesmo método *resposanavelRepository* que vai localizar o *responsável* pelo *idResponsavel* e sobrescrever o registro no banco de dados com os novos dados.

```
1.  @PutMapping("/responsavel")
2.  public Responsavel atualizar(String nomeResp, long carteiraIdentidade,
3.  long telefone, TipoResponsavel tipoResponsavel,MultipartFile foto,int
4.  dEscola,int idResponsavel) throws IOException {
5.
6.      Responsavel responsavel = responsavelRepository.findbyIdResponsavel
    (idResponsavel, idEscola);
7.      responsavel.setCarteiraIdentidade(carteiraIdentidade);
8.      responsavel.setFoto(foto.getBytes());
9.      responsavel.setNomeResp(nomeResp);
10.     responsavel.setTelefone(telefone);
11.     responsavel.setTipoResponsavel(tipoResponsavel);
12.
13.     return responsavelRepository.save(responsavel);
14.     }
```

Figura 14. Trecho código método PUT

Por fim, o método DELETE tem como função remover um registro existente no banco de dados, por meio da notação "@DeleteMapping" em cima do método *deletaAluno*, conforme mostrado no trecho de código da Figura 15. São passados por

parâmetro deste método, via rota URL "/aluno/{idAluno}/escola/{idEscola}" o id do aluno e id da escola. Entretanto, não é possível diretamente excluir um aluno no banco de dados, pois a tabela aluno possui dependência de chave estrangeira com as outras relações. Sendo assim, é necessário antes por meio de um laço de repetição deletar a relacionamento do aluno com as outras tabelas. Após isso é realizado a exclusão do aluno e retornado a *String* "Aluno deletado" nesse método.

```
1.  @DeleteMapping("/aluno/{idAluno}/escola/{idEscola}")
2.      public String deletaAluno(@PathVariable(value = "idAluno") long idA
    luno,@PathVariable(value = "idEscola") long idEscola) {
3.          ArrayList<Long> listIdAluno = alunoResponsavelRepository.getIdR
    elacionamentoAluno(idAluno,idEscola);
4.          alunoResponsavelRepository.flush();
5.          for(long elem : listIdAluno){
6.              alunoResponsavelRepository.deleteById(elem);
7.          }
8.          List<AlunoControleAluno> listAlunoControleAluno = alunoControle
    AlunoRepository.getAlunoControleAlunoByIdList(idAluno);
9.          for(AlunoControleAluno thisAlunoControleAluno : listAlunoContro
    leAluno){
10.             alunoControleAlunoRepository.delete(thisAlunoControleAluno)
    ;
11.         }
12.         alunoRepository.deleteById(idAluno);
13.         return "Aluno deletado";
14.     }
```

Figura 15. Trecho código método DELETE

5.5.2 Função Fetch

Fetch é um recurso do Javascript utilizado no *front end* do Carinho Escolar que interpreta métodos HTTPS, uma alternativa ao recurso *XMLHttpRequest* por sua baixa complexidade não precisando se preocupar com *call-backs*, como uma requisição HTTPS utilizando *promises (.then* ou *.catch*) presente no Javascript.

Conforme o trecho de código na Figura 16 tem o objetivo persistir os dados em uma tabela de controle de frequência, no banco de dados com os seguintes atributos: *status* para informar se o aluno entrou ou saiu da escola, *idAluno* um id único representando qual o aluno e *idEscola* outro id único representando em qual Escola o aluno está matriculado.

```
1.  const formData = new FormData();
2.  formData.append('tipoHorario', status);
3.  formData.append('idAluno', alunoId);
4.  formData.append('idEscola', escolaId);
5.  const url = "http://localhost:8080/api/controle"
6.  const request = new Request(url, {
7.      method: 'POST',
8.      body: formData
9.  });
10. fetch(request)
11.     .then(response => response.text())
12.     .then(console.log)
13. }
```

Figura 16. Trecho código função fetch para enviar o POST via Javascript

Primeiro, é montado um *formData* com cada um desses atributos citados anteriormente, então é informado por meio da constante url qual o endereço dessa rota na

API. Por fim, é montado uma requisição com a url, *formData* e o método que foi escolhido para essa rota, que nesse caso é o POST. Para outros tipos de requisições HTTPS, o trecho de código a ser utilizado é similar, mudando somente a informação do parâmetro "método". Após ser montado a requisição é passado por parâmetro no método fetch para ser enviado a requisição.

Para recuperar e exibir os dados salvos no front end, conforme trecho do código apresentado na Figura 17 é informado na variável *getTurmas* uma rota válida do método GET. No exemplo é recuperado uma lista de todas as turmas de uma determinada escola, por razão da requisição HTTP ter que consultar no servidor essas informações e o retorno não ser instantâneo, isso não é caracterizado por uma requisição síncrona, em que todos os dados retornam de uma vez, e sim uma requisição assíncrona. No método fetch, após informar via parâmetro a rota solicitada, é utilizado o recurso da linguagem *.then* que recebe os dados de maneira assíncrona.

Para percorrer os dados recebidos é utilizado *data.map*, com laço de repetição de cada um dos elementos recebidos. Este é passado por parâmetro para *innerHTML* que é um recurso do HTML, que pode manipular o *DOM* (Árvore de Elementos HTML) pois as páginas do *front end* são todas renderizadas estaticamente, então com esse recurso é possível manipular essas páginas web dinamicamente, de acordo com informações recebidas no servidor, para exibir essas informações na página web.

```
1.  var escolaId = parseInt( sessionStorage.getItem('escola'))
2.  const getTurmas = "http://localhost:8080/api/turmas/escola/"+escolaId
3.  fetch(getTurmas)
4.      .then(response => response.json())
5.      .then(data => {
6.          document.getElementById('dados').innerHTML = data.map(turmaTemp
    late).join(' ')
7.      }).catch(err => console.log(err))
8.  function turmaTemplate(turma) {
9.      return `
10.     <option value="${turma.id}"> ${turma.numeroTurma}</option><br/>
11.   `
12. }
```

Figura 17. Trecho código função fetch para receber e exibir dados

5.6 Auditoria

Sistemas modernos tem a demanda de poder acessar versões anteriores de dados, seja para fins de auditoria quando solicitado ou para restauração de dados modificados erroneamente. Por esse motivo, foi utilizado o recurso presente no Hibernate chamado Hibernate Envers. Esse recurso possui a vantagem de não precisar utilizar no banco de dados nem *Triggers* e *Procedures*.

Ele é compatível com todos os bancos de dados relacionais, caracterizado pelo baixo acoplamento da aplicação. Para utilização do *framework* é utilizado a notação "@Audited" em cima de cada classe Java que se pretende auditar (Figura 18). Por padrão, a entidade no banco de dados que for auditada vai criar uma tabela com o nome da classe mais o prefixo *_aud*.

```
1.  @Entity
2.  @Audited
3.  public class Aluno implements Serializable {
4.  .....
5.  .....
6.  }
```

Figura 18. Trecho de código Hibernate Envers

Conforme a Figura 19 informa, de acordo com tipo de operação realizada é informado na tabela de Auditoria no banco de dados.

HIBERNATE ENVERS	
REV	ID DA REVISÃO DE VERSÃO
REVTYPE = 0	INSERÇÃO
REVTYPE = 1	EDIÇÃO
REVTYPE = 2	REMOÇÃO

Figura 19. Tabela de códigos do Hibernate Envers

5.7 Infraestrutura e Hospedagem

A fim de hospedar a aplicação, foram utilizados serviços de computação em nuvem. Esta parte do projeto tem forte relação para a parte de Ops do projeto. O serviço escolhido foi EC2 da Amazon AWS, que permite criar máquinas virtuais em nuvem, disponível a utilização gratuita durante o período de avaliação de 3 meses. No painel de máquinas virtuais, é possível visualizar uma ou mais máquinas virtuais existentes (Figura 20). Para executar o sistema Carinho Escolar foi escolhido o sistema operacional Linux com a distribuição Ubuntu 20.4.

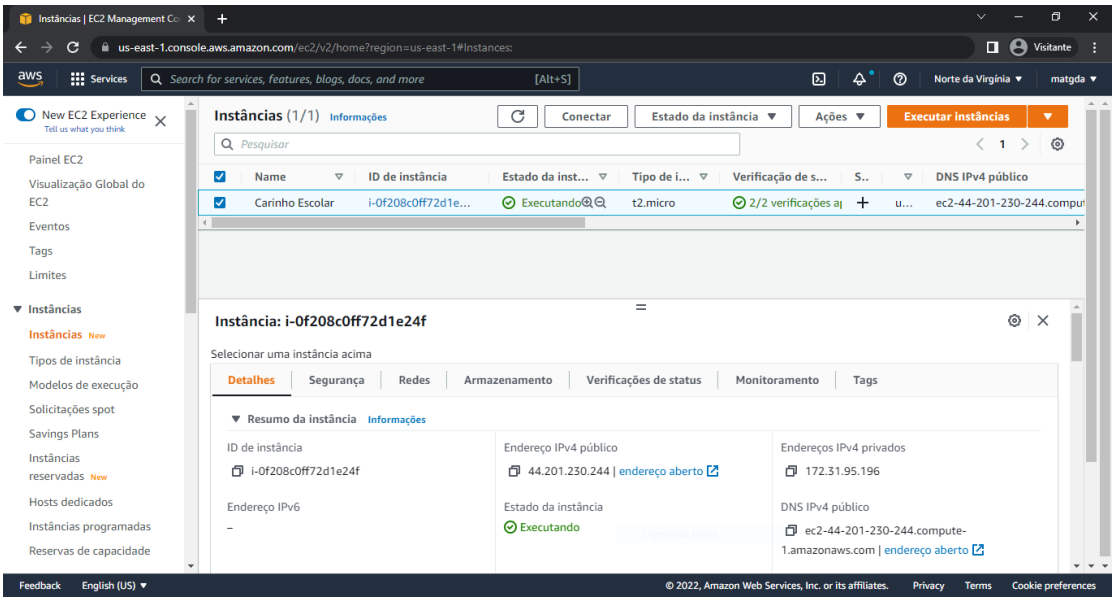

Figura 20. Painel EC2 – Amazon AWS

Já para hospedagem do *front end* do sistema Carinho Escolar foi utilizado o serviço Amazon Simple Storage Service (Amazon S3) (Figura 21) para hospedar páginas web do projeto, podendo ser acessado por meio de um endereço na web disponibilizado pelo serviço.

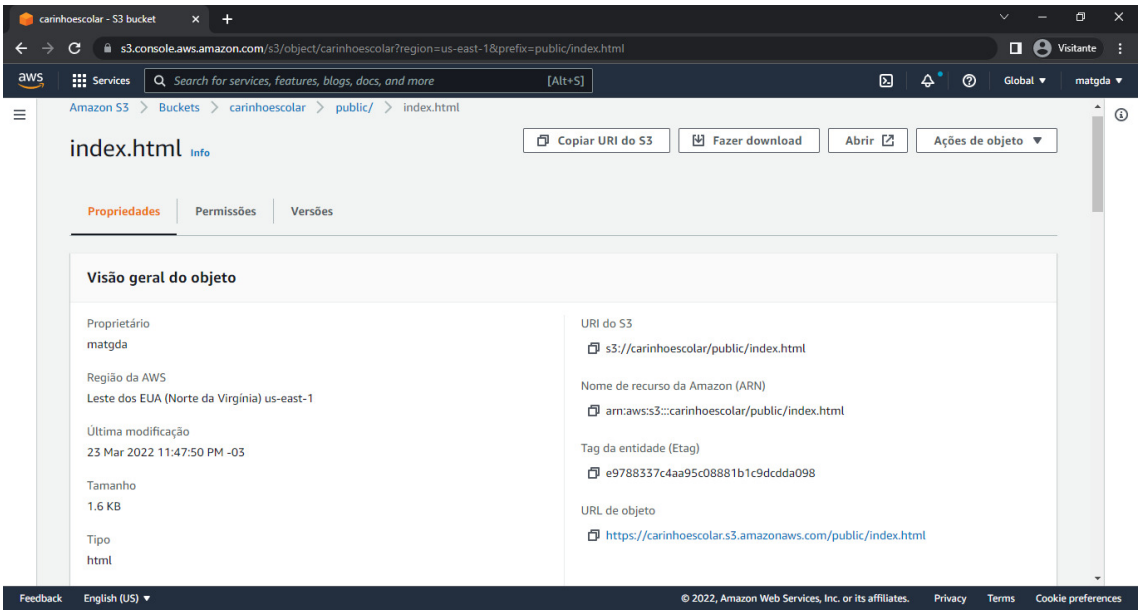

Figura 21. Painel S3 – Amazon AWS

A arquitetura do sistema descrita na Figura 22 se baseia na Arquitetura MVC, apresentando como o usuário interage com sistema, por meio do *Web View* (*Front End*). Essa interação pode ser um computador, celular ou tablet que comunica bilateralmente com *back end* por meio de uma requisição *http request* criada com SpringBoot que por sua vez persiste os dados no banco de dados MariaBD.

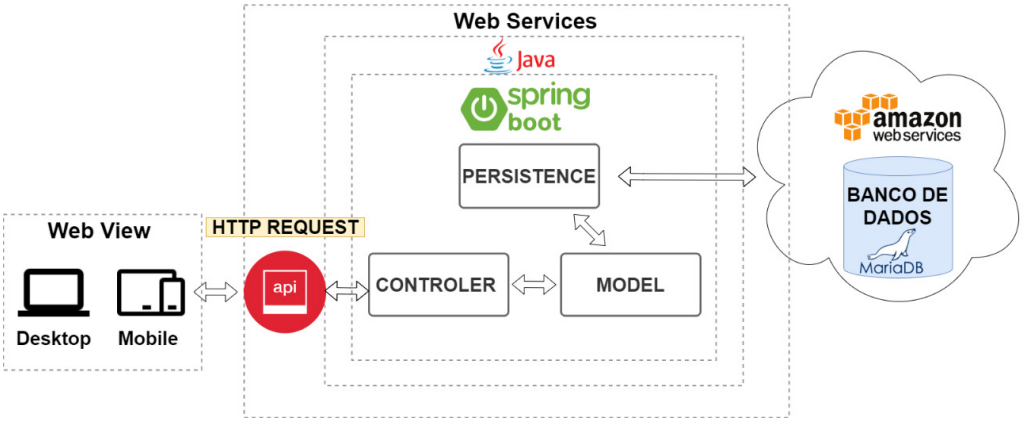

Figura 22. Arquitetura do Sistema Carinho Escolar

6. Resultados

O sistema foi finalizado e entregue de acordo com o objetivo principal e os objetivos específicos da pesquisa. O sistema evolui muito durante a pesquisa. Novas perspectivas das funcionalidades foram implementas de modo incremental, como preconiza DevOpsBL. Como resultados, obteve-se

I. Controle de entrada e saída de alunos: é responsável pelo controle dos alunos em si. Em linha gerais, dado o nome do aluno o sistema apresenta foto e dados previamente cadastradas no sistema, no mesmo modo que auxilia o monitor responsável pela liberação da criança;

II. Cadastro de pessoas: cadastro das pessoas que utilizarão o sistema (perfis de administrador, professor, monitor) e de pessoas gerenciadas pelo sistema (alunos e respectivos pais/responsáveis);

III. Perfis de usuários: Por meio da página de usuários é possível cadastrar um novo usuário no sistema e escolher qual tipo de perfil de acesso, poderá ser escolhido entre os perfis de Professor, Monitor e Administrador. Cada perfil possui visualizações e funções diferentes, o Monitor pode visualizar a lista de alunos, turmas e frequência. O Professor tem a funcionalidade de visualizar o perfil de Monitor, porém pode ser responsável por uma turma. Por fim, o administrador pode excluir e modificar dados e cadastrar usuários.

IV. Relatórios: Na página de turmas é possível visualizar a lista de alunos filtrados por turmas cadastradas no sistema, apresentando os seguintes dados: nome, carteira de identidade, gênero e turma. Na página controle é possível visualizar todos os registros de frequência de aluno, possuindo os seguintes campos: nome do aluno, status e horário. Em ambas as páginas são possíveis exportar os dados por meio do padrão CSV (*Common Separated Value*) que é um padrão de dados que separa as informações com delimitador virgula.

V. Múltiplas escolas: o sistema foi desenvolvimento de modo que múltiplas escolas possam acessar o sistema, sem que tenham acesso aos dados umas das outras, zelando assim pela segurança do sistema.

O resultado do desenvolvimento das principais páginas do Carinho Escolar inicia-se pela Figura 23. Pode-se observar que a tela principal do Carinho Escolar apresenta os seguintes elementos em uma lista de alunos, cada um representado por um *Card* contendo: foto, dados pessoais e por último status no sistema, dois botões para alterar o status e para mostrar dados relacionados aos responsáveis cadastrados no sistema.

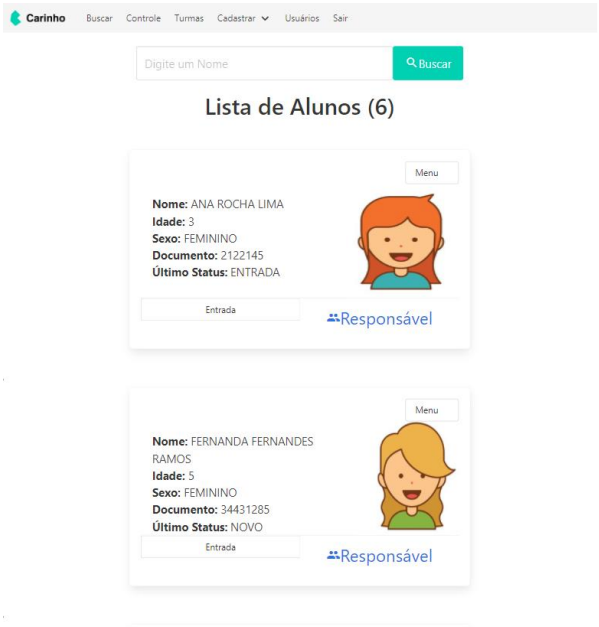

Figura 23. Lista de Alunos – Carinho Escolar

Ao pressionar o botão Responsáveis, o usuário é direcionado a uma página Lista de Responsáveis (Figura 24) que mostra por meio de *Cards* uma foto de cada

responsável, informações pessoais e botão para editar dados do responsável e para excluir, porém esses botões são somente visíveis no perfil Administrador.

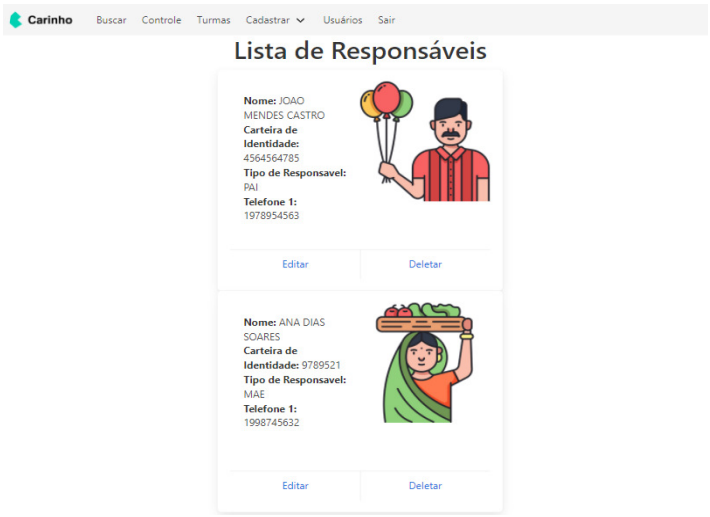

Figura 24. Lista de Responsável por Aluno – Carinho Escolar

Para cadastrar um novo aluno, é utilizado o menu Cadastrar opção Aluno, conforme Figura 25, nessa página é cadastrado as seguintes informações nome, idade, data, foto de identificação e qual turma o aluno vai pertencer. Um dos principais desafios no desenvolvimento foi enviar fotos capturadas no navegador para o banco de dados por meio de uma API desenvolvida ao longo do projeto.

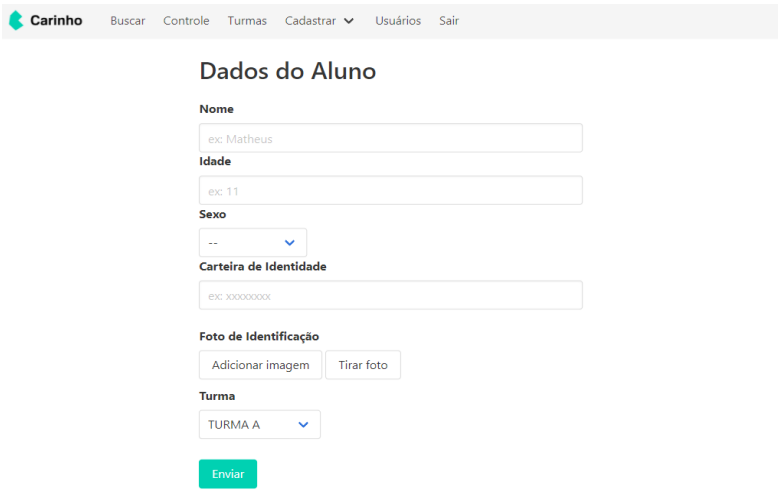

Figura 25. Cadastro de Alunos – Carinho Escolar

Na página Controle, está disponível um menu que leva à lista de alunos por turma, conforme Figura 26. Há também dados controle de entrada e saída dos alunos, e é possível filtrar por nome. Pressionando o botão no canto inferior direito é possível exportar em um arquivo .CSV, gerando assim um relatório no formato de planilha eletrônica.

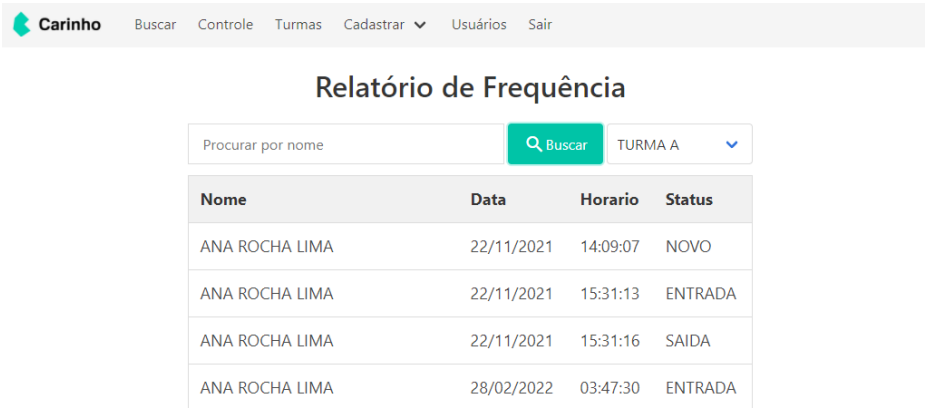

Figura 26. Relatório de Frequência – Carinho Escolar

Na página Turmas, está disponível um menu para listar todos os alunos filtrado por turma (Figura 27), apresentando informações cadastrais de cada Aluno. Pode-se filtrar por nome. É possível exportar este relatório para planilha eletrônica também.

Figura 27. Relatório de Turmas – Carinho Escolar

6.1 Pesquisa com Especialistas

Foi feita uma pesquisa com especialistas da área de TI, Educação e usuário final em fevereiro de 2022. Utilizou-se a escala de *Likert* conforme a Tabela 1 para avaliação do sistema.

No.	Pergunta
1	Você é um profissional da área de TI?
2	Você já atuou, ou atua na área de educação?
3	Caso seja um usuário você tem conhecimento avançado em informática?
4	Você conhece um sistema similar?
5	É fácil encontrar informações no sistema?
6	A utilização do sistema é fácil?
7	O sistema auxilia no gerenciamento dos alunos?
8	Sobre sistema você concorda que esse sistema aumenta a segurança na portaria escolar, não permitindo o acesso de pessoas não autorizadas.

No.	Pergunta
9	Você considera que o sistema gerência a entrada e saída dos alunos?
10	Se você tivesse um filho(a) você escolheria uma escola que possui o benefício de ter um software para gerenciamento de alunos?

Tabela 1. Perguntas Questionário Carinho Escolar

Conforme a Tabela 2, constatou-se que a maioria dos participantes gostaram do sistema Carinho Escolar. É razoável inferir que sistema, por fim, atingiu os objetivos propostos.

Participante	Qualificação dos participantes				Percepções do software Carinho Escolar					
	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
P1	d	c	d	d	c	n	c	c	c	c
P2	c	d	c	d	c	c	c	c	c	c
P3	d	d	n	d	c	c	c	c	c	c
P4	c	d	c	c	c	c	c	c	c	c
Resultados / mediana	**2 TI**	**1 EDU**	**2 TI**	**NC**	**fácil**		**útil**		**Atendeu ao objetivo**	

Legenda:
c = concordo, n = neutro, d = discordo;
TI = pessoas com formação em TI; EDU = pessoas com conhecimento em educação;
NC = Não conhece um *software* similar ao sistema Carinho disponível no mercado

Tabela 2. Respostas Questionário

7. Considerações Finais

Nesta pesquisa, foi desenvolvido o sistema Carinho Escolar, com o objetivo de ser uma alternativa gratuita para a ausências de sistemas de controle de entrada e saída de alunos, principalmente para as escolas mais necessitadas deste tipo de sistema. O sistema Carinho Escolar visa assim contribuir para uma maior segurança no ambiente escolar, por meio da conferência de informações cadastrais e os alunos e responsáveis.

O método de pesquisa adotado foi de estudo de caso. O desenvolvimento do Carinho Escolar foi conduzido utilizando a metodologia DevOpsBL mesclando a entrega das atividades de desenvolvimento e operações. A arquitetura do *software* foi desenvolvida em duas partes principais, front *end* e *back end.* O *software* teve por objetivo ser acessível tanto de computadores como de *smartphones.* Além da criação do sistema Carinho Escolar em si, a pesquisa consultou dois especialistas da tecnologia, dois especialistas em educação e um usuário final. A percepção foi que o Carinho Escolar é fácil de utilizar, possui funcionalidades pertinentes e útil e cumpre com o que foi proposto.

Destacam-se as seguintes disciplinas estudadas no decorrer do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas que auxiliaram diretamente no desenvolvimento desse artigo: Serviços de Rede, Engenharia de Software, Banco de dados I e II, Linguagem de Programação I e II e por fim Desenvolvimento Web.

Para trabalhos futuros, sugere-se implementar o Carinho Escolar em escolas da região metropolitana de Campinas, na qual será possível realizar a pesquisa de campo

observando os alunos e entrevistar funcionários da escola sobre a utilização do Carinho Escolar.

8. Referências

AMAZON. **O que é o Amazon S3?**. Disponível em: https://docs.aws.amazon.com/pt_br/AmazonS3/latest/userguide/Welcome.html. Acesso em 20 nov. 2021.

DEVMEDIA. **Desenvolvendo com Hibernate**. Disponível em: https://www.devmedia.com.br/desenvolvendo-com-hibernate/14756. Acesso em: 22 dez. 2021.

GSTI. **Padrão MVC | Arquitetura Model-View-Controller.** Disponível em: https://www.portalgsti.com.br/2017/08/padrao-mvc-arquitetura-model-view-controller.html Acesso em: 30 mar. 2022.

GLOBO. **Polícia investiga sequestro de criança em escola no bairro do Guamá.** Disponível em: https://g1.globo.com/pa/para/noticia/2019/04/03/policia-investiga-sequestro-de-crianca-em-escola-no-bairro-do-guama-em-belem.ghtml. Acesso em: 2 abr. 2020.

GLOBO. **Pais procuram a polícia após mulher se passar por avó e tentar retirar criança de 2 anos da escola: 'Poderia não estar comigo'.** Disponível em: https://g1.globo.com/sp/sao-jose-do-rio-preto-aracatuba/noticia/2020/02/19/pais-procuram-a-policia-apos-tentarem-levar-filho-de-2-anos-de-escola-poderia-nao-estar-comigo.ghtml. Acesso em: 19 out. 2020.

GROTTA, ALEXANDRE; PRADO, EDMIR P. V. **DevOpsBL: DevOps-based Learning on Information Systems Higher Education.** Disponível em: https://aisel.aisnet.org/amcis2021/is_education/sig_education/6. Acesso em 1 nov. 2021.

HOSTINGER. **O Que é GitHub e Como Usá-lo**. Disponível em: https://www.hostinger.com.br/tutoriais/o-que-github. Acesso em: 22 dez. 2021

INSTITUTO DE PESQUISA ECONÔMICA APLICADA - IPEA (2019). **Atlas da Violência.** Disponível em: https://www.ipea.gov.br/portal/images/stories/PDFs/relatorio_institucional/190605_atlas_da_violencia_2019.pdf. Acesso em: 2 abr. 2020.

JEREMY THOMAS (2018). **Creating Interfaces with Bulma. Bleeding Edge Press.** Disponível em: https://bleedingedgepress.com/book_excerpts/01E9D1/creating_interfaces_with_bulma_sample.pdf Acesso em: 8 abr. 2020.

MOZILLA. **HTML**. Disponível em: https://developer.mozilla.org/pt-BR/docs/Web/HTML. Acesso em: 1 set. 2020.

MOZILLA. **Java Script**. Disponível em https://developer.mozilla.org/pt-BR/docs/Web/JavaScript. Acesso em: 14 set. 2020.

MARIADB**. About mariadb server**. Disponível em: https://mariadb.org/about/. Acesso em 14 set. 2020.

MICROSOFT AZURE. **O que é computação em nuvem?**. Disponível em: https://azure.microsoft.com/pt-br/overview/what-is-cloud-computing/#cloud-computing-models. Acesso em 20 nov. 2021.

ORACLE. **O que é a Tecnologia Java e porque preciso dela?.** Disponível em: https://www.java.com/pt-BR/about/whatis_java.jsp#:~:text=O%20Java%20%C3%A9%20uma%20tecnologia,%C3%A9%20executado%20no%20seu%20browser. Acesso em: 3 abr. 2020.

ORACLE. **Auditoria Avançada com Hibernate Envers**. Disponível em: https://www.oracle.com/br/technical-resources/articles/java/audit-advanced-hibernate-envers.html Acesso em: 23 dez. 2021.

ORACLE. **O que é um Banco de Dados | Oracle Brasil**. Disponível em: https://www.oracle.com/br/database/what-is-database/ Acesso em 20 nov. 2021.

POSTMAN. **What is Postman?**. Disponível em: https://www.postman.com/product/what-is-postman/. Acesso em: 23 dez. 2021.

RED HAT. **O que é uma máquina virtual (VM)?**. Disponível em: https://www.redhat.com/pt-br/topics/virtualization/what-is-a-virtual-machine. Acesso em 20 nov. 2021.

RED HAT. **What is a REST API?.** Disponível em: https://www.redhat.com/en/topics/api/what-is-a-rest-api. Acesso em 30 mar. 2022.

SOFTPONTES. **SoftPontes - Sistema Escolar.** Disponível em: http://www.softpontes.com.br/. Acesso em: 10 abr. 2020.

SPRING. **Spring boot**. Disponível em: https://spring.io/projects/. Acesso em:

14 Set. 2020.

SOURCETREE. **Sourcetree.** Disponível em: https://www.globallogic.com/services/offerings/atlassian/products/sourcetree/#:~:text=Sourcetree%20is%20a%20free%20graphical,visualize%20and%20manage%20your%20repositories. Acesso em 30 mar. 2022.

SIMPLYPSYCHOLOGY. **Likert Scale Definition, Examples and Analysis.** Disponível em: https://www.simplypsychology.org/likert-scale.html

YIN, R. K. (2011). **Case study research: Design and methods. modern language journal**.

Documento Digitalizado Público

ADS-HTO: TCC versão final de Matheus Gabriel de Andrade (HT1620932)

Assunto: ADS-HTO: TCC versão final de Matheus Gabriel de Andrade (HT1620932)
Assinado por: Alexandre Grotta
Tipo do Documento: Relatório
Situação: Finalizado
Nível de Acesso: Público
Tipo do Conferência: Cópia Simples

Documento assinado eletronicamente por:

- **Alexandre Grotta**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 04/04/2022 20:53:19.

Este documento foi armazenado no SUAP em 04/04/2022. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 936354
Código de Autenticação: 4d76e90510