FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Arnaldo Assis Ferreira

OClaRy – Online Class Diary

Americana, SP

2018

FACULDADE DE TECNOLOGIA DE AMERICANA

Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

Arnaldo Assis Ferreira

OClaRy – Online Class Diary

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, sob a orientação do Prof. Dr. Kleber de Oliveira Andrade

Área de concentração: Desenvolvimento de Software e Plataforma Web.

Americana, SP.

2018

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

F439o FERREIRA, Arnaldo Assis

OClaRy: Online Class Diary. / Arnaldo Assis Ferreira. – Americana, 2018.

100f.

Monografia (Curso de Tecnologia em Análise e Desenvolvimento de Sistemas) - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Dr. Kleber de Oliveira Andrade

1 Desenvolvimento de software 2. Informática - educação I. ANDRADE, Kleber de Oliveira II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: 681.3.05

681.3:37

Arnaldo Assis Ferreira

OClaRy – Online Class Diary

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas pelo CEETEPS/Faculdade de Tecnologia – FATEC/ Americana.

Área de concentração: Desenvolvimento de Software e Plataforma Web.

Americana, 28 de junho de 2017.

Banca Examinadora:

Kleber De Oliveira Andrade (Presidente)
Doutor
Fatec Americana

Eduardo Antônio Vicentini (Membro)
Mestre
Fatec Americana

Benedito Aparecido Cruz (Membro)
Mestre
Fatec Americana

AGRADECIMENTOS

Aos meus pais, que sempre se esforçaram ao máximo para me dar uma boa educação. A minha mãe, que sendo professora, foi a inspiração para esse trabalho. E ao meu pai, que mesmo não estando aqui, serviu como alicerce da minha vida e um exemplo a ser seguido.

Aos docentes e colegas da FATEC Americana, que me ensinaram muito sobre desenvolvimento e a ter o conhecimento necessário para desenvolver esse trabalho.

Ao professor Dr. Kleber de Oliveira Andrade, que além me orientou no desenvolvimento desse trabalho, além de ser meu treinador nas maratonas de programação e outros eventos.

E a todos que fizeram e fazem parte da minha vida e sempre me incentivam a fazer o meu melhor.

RESUMO

Os professores possuem várias reponsabilidades quando ministram suas aulas: preparar e aplicar o conteúdo aos alunos, assim como anotar a presença e as notas dos mesmos, entre outras atribuições. Para ajudar a organizar essas tarefas, o professor dispões do documento oficial chamado diário de classe que contém a lista de alunos da turma, os dias que estiveram presentes, as suas notas e também as provas e o planejamento para a turma. Porém, como o diário é apenas um por turma, pode ficar difícil gerenciar todos os diários se o professor lecionar me várias turmas. Com o objetivo de melhorar o gerenciamento dessas tarefas, esse trabalho apresenta o desenvolvimento de um software chamado OClaRy, que possui as funções básicas de um diário de classe, como cadastro de alunos, registro de presenças e notas, cadastro de entradas de planejamentos, provas e presenças. O software foi desenvolvido na linguagem Javascript usando o framework Vue.js, a interface foi desenvolvida usando Materilize.css, um framework CSS que usa componentes em *Material Design* e os dados foram armazenados no banco de dados NoSql Firebase. O resultado desse trabalho se apresenta em um protótipo funcional do software.

Palavras Chave: Educação, Engenharia de Software, Javascript

ABSTRACT

Teachers have a lot of responsibilities when they tach classes: preparing and providing lesson content, as well as registering the attendance and student's grade, among many other assignments. In order to organize these tasks, the teacher has an official document know as "class diary" which provides a list of students, the days they attended classes, the record of their their grades, and the exams and also classes planning. However, the class diary in only one per class, it may be difficult to manage all the diaries if the teacher has a lot of classes. With the purpose of improving the management of these tasks, this work presents the development of the OClaRy software, that has the basic functions of a class diary, like students' register, their attendances and grades, and the registry entries of planning, exams and attendances entry. The software was developed in Javascript, using the Vue.js framework, the interface was developed using Materialize.css, a CSS framework that uses Material Desing components, and the data was stored in the Firebase, a NoSQL database. This work's result is presented by a functional prototype.

Keywords: *Education; Software Engineering; Javascript*

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

1 INTRODUÇÃO

A educação é uma parte importante da vida do ser humano. As pessoas passam em média 10 anos de sua vida nas escolas, nas salas de aulas, recebendo educação dos mais diversos campos do conhecimento. O responsável por administrar esse conhecimento e passar para os alunos é a autoridade máxima na sala de aula: o professor.

Para guardar as informações importantes da turma em questão, o professor dispõe do diário de Classe. O diário de classe é o documento oficial que os professores usam para registrar e controlar as turmas em que lecionam. Esse documento apresenta os alunos, atividades avaliativas, informações de presença e notas finais, dentre outros aspectos da turma que sejam dignos de nota.

Porém, um problema que o diário de classe apresenta é que ele serve apenas para uma turma em particular. Se o professor lecionar em mais de uma turma, o que acontece na maioria das vezes, terá que controlar um diário para cada turma. Isso pode se tornar trabalhosos com o passar do tempo.

Com o crescimento e a popularização das plataformas digitais usadas em diversas aplicações, se tornou uma alternativa atraente, além de prática, mover o diário de classe para o ambiente digital. Assim nasceu o *OClaRy* o *Online Class Diary* (Diário de Classe Digital).

Esse trabalho tem como objetivo geral descrever o processo de desenvolvimento do software OClaRy utilizando informações obtida em uma pesquisa sobre como o diário de classe é organizado e administrado. Os objetivos específicos desse trabalho são:

- Fazer um estudo sobre os diários de classe, para definir melhor o comportamento do software.
- Aplicar a metodologia Kanbam com algumas modificações no desenvolvimento do sistema
- Apresentar os resultados e um protótipo funcional do Software OClaRy

1.2 Estrutura do Trabalho

O conteúdo do trabalho está estrutura da seguinte maneira

- Capítulo 2: explicação sobre o que consiste em diário de classe, suas funções. Também apresenta uma analise de softwares existentes com a mesma premissa.
- Capítulo 3: descrição completa do processo de desenvolvimento do software: planejamento, metodologia, diagramas de engenharia de software, assim como a codificação propriamente dita
- Capítulo 4: considerações finais

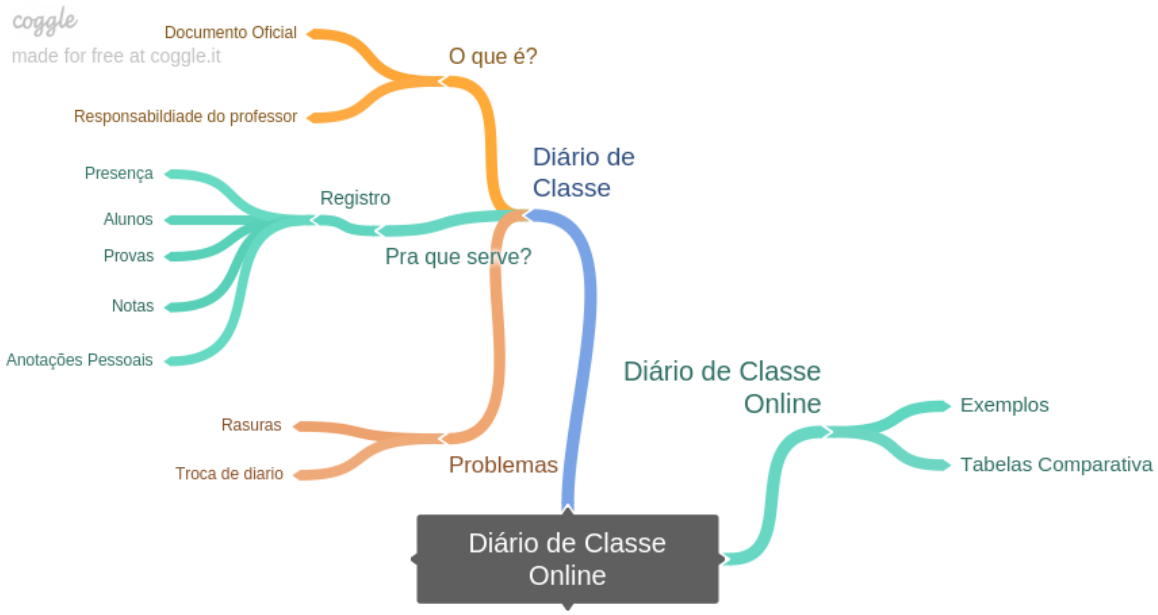
Além dos capítulos, no final do trabalho há dois apêndices que apresentam conteúdo complementar, tais como:

- Apêndice A: apresenta as telas do sistema;
- Apêndice B: detalha os eventos do Firebase e como se relacionam com o código do vue.js;

2 DIARIOS DE CLASSE ONLINE

Para estruturar esse capítulo, foi usado um mapa mental para orientar o fluxo das informações. Criado em nos anos de 1960 por Tony Buzan, o mapa mental é um diagrama para organizar ideias “colocando-as de dentro do seu cérebro para fora do seu cérebro” (BUZAN, 2005), sem se preocupar com ordem ou estrutura dos dados. Com o mapa mental, é possível focar apenas nas ideias e organizá-las posteriormente. A figura 1 representa um mapa mental para a estrutura desse capitulo, feita com a ferramenta Coggle.

Figura 1 – Mapa mental – estrutura do capitulo.



Fonte: Autoria Própria

O diário de classe é um documento importante para as instituições educadoras e para os seus frequentadores: alunos e professores. Se trata de um documento oficial pertencente à Unidade Escolar, contendo informações importantes de uma determinada turma e disciplina.

O diário de classe é elaborado pela Secretaria da escola, e, após finalizado, passa ser responsabilidade do professor daquela turma e disciplina em particular. O professor é responsável por preencher os dados corretos da turma, e cuidar da veracidade dos mesmos. Se o mesmo professor lecionar mais de uma disciplina na

mesma turma, será responsável por outro diário de classe e assim por diante. Ou seja, é um diário de classe para cada turma e disciplina.

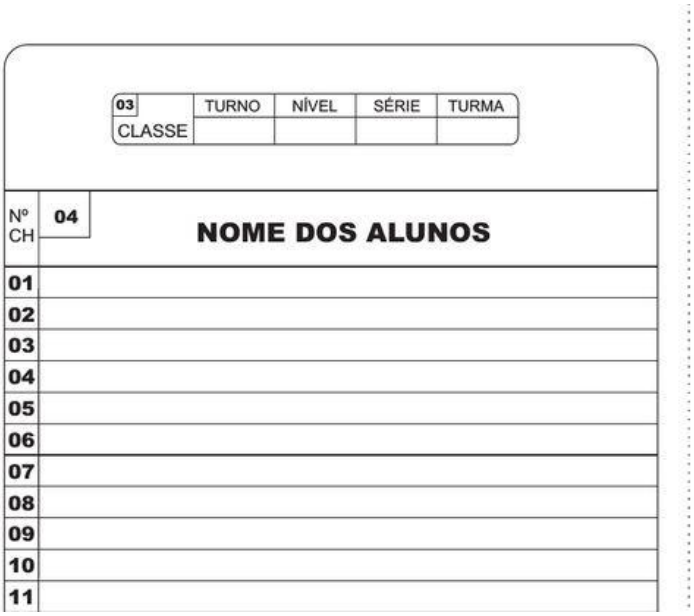
Mas, mesmo sendo responsabilidade do professor preencher o diário, é encargo da Unidade Escolar guardar e manter o diário de classe, não sendo permitido ao professor deixar o prédio da unidade com o mesmo. E esse documento deve ser devolvido á escola assim que o período letivo da turma de encerrar.

Um diário de classe contém várias informações sobre uma turma, que devem ser preenchidas e controladas constantemente pelo professor responsável, essas informações consistem em:

- Nome dos alunos e seus respectivos números de chamada;
- As atividades avaliativas da disciplina, e as notas que os alunos iraram nessas avaliações;
- A média final dos alunos com base nas atividades avaliativas;
- Anotação da frequência: as presenças e ausências (justificadas ou não) dos alunos nas aulas;
- Anotações particulares;
- Planejamento das aulas contendo data a conteúdo a ser estudado, deve-se anotar o planejamento de dias não-letivos

As figuras 1 a 4 mostram em detalhes como esses dados são organizados no diário de classe:

Figura 2 – Scan Parcial de um diário de classe – alunos.

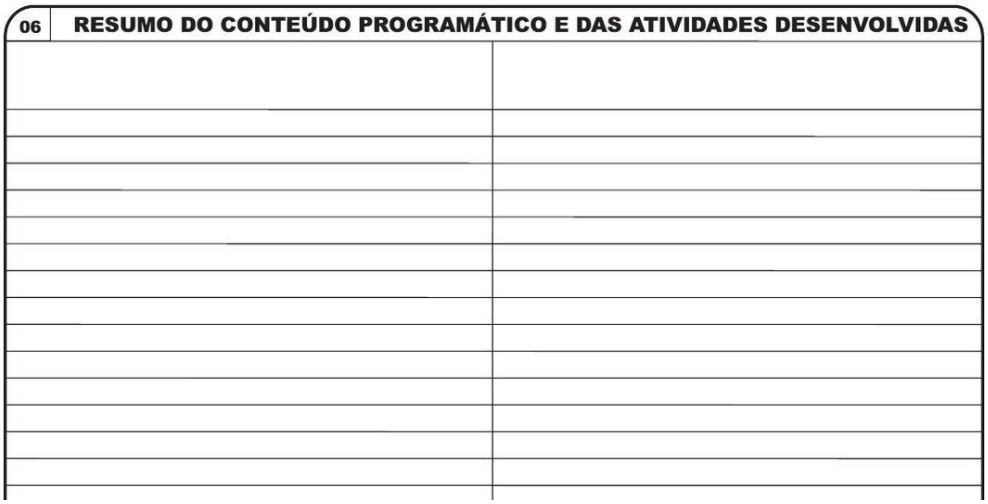


Fonte: Adaptado de YOUBLISHER (2017)

Figura 3 – Scan parcial de um diário de classe – frequência dos alunos.

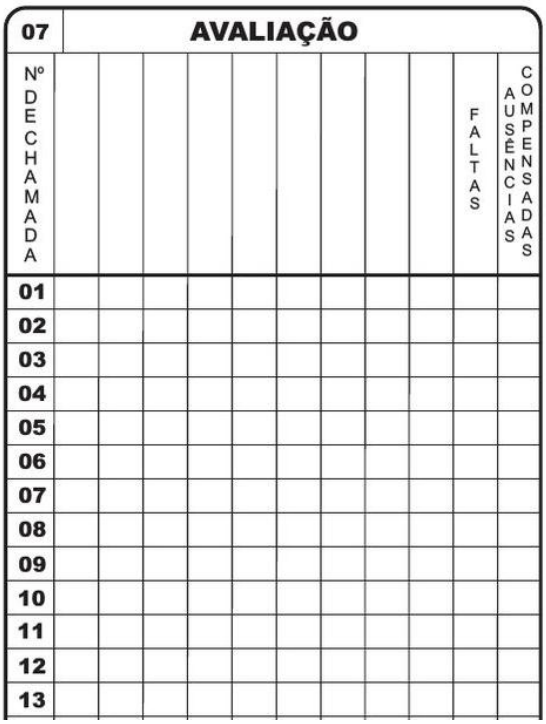
Fonte: Adaptado de YOUBLISHER (2017)

Figura 4 – Scan Parcial de um diário de classe – conteúdo programático.



Fonte: Adaptado de YOUBLISHER (2017)

Figura 5 – Scan Parcial de um diário de classe – avaliação.



Fonte: Adaptado de YOUBLISHER (2017)

Porém, esse documento apresenta alguns problemas e desvantagens. Uma delas é que, como há apenas um diário por turma e disciplina, se o professor dá muitas aulas, logo ele possui muitos diários para preencher, e cuidar de vários

diários ao mesmo tempo pode ser cansativo. Outra é que os diários oficiais não permitem rasuras, o que exige do professor um cuida extra ao preenche-lo. Além disso, as informações estão dispostas de um jeito muito abrangente, ou seja, são muitas informações ao mesmo tempo. Isso se torna trabalhoso se o professor quer uma informação específica.

Com esses problemas, surgiu a necessidade de criar um sistema que tenha as funções de um diário de classe, como cadastrar alunos, atribuir notas etc. A próxima secção irá apresentar alguns sistemas com esse objetivo.

2.1 Sitemas de Diário de Classe Online

Essa seção irá abranger alguns sistemas de diários de classe online. Há vários aplicativos e programas que fazem isso. Muitos deles são disponibilizados pelas próprias unidades escolares, sendo usado de forma oficial no lugar do diário de classe impresso. Outros são aplicativos próprios para o professor se organizar, não sendo vinculados a nenhuma instituição educacional.

2.1.1 Sistema de Gestão Escolar

O sistema de gestão escolar foi criado pela empresa Tecsoft[1]. É uma aplicação web voltada principalmente para os professores e os gestores de educação. Apesar do seu foco ser para professores, ele dispõe de funções quanto ao contato com os pais dos alunos, como por exemplo, um cadastro de pais e a opção de enviar e-mail a eles quando necessário.

No caso de escolas particulares, o sistema também disponibiliza opções sobre a mensalidade, como informações financeiras e 2 º vias de boletos. Com acesso restrito, o administrador do colégio tem controle total nas diversas funcionalidades que o sistema oferece. A integração das informações do sistema disponibiliza aos pais informações em tempo real da participação dos alunos na escola.

[1] Informações sobre o software disponível em <http://www.tecsoft.com.br/produtos/sistema-gestao-escolar/> Acesso em 19/05/201

2.1.2 NotaDez

O diário NotaDez é um software criado por Anselmo Gomes, professor e analista de sistemas[2]. O diário NotaDez é um software que atende professores e outras pessoas do ambiente escolar, como alunos, funcionário da secretaria escolar, direção e análise pedagógica.

Esse software foi construindo usando uma plataforma web, e possui 2 opções de licenciamento do software são: serviço online decentralizado, onde o software é hospedado na nuvem e não é necessária mudanças na infraestrutura da escola para acessa-lo, e serviço da rede local da escola, onde o software é instalado em uma rede interna da escola, sendo necessária alterações na infraestrutura da escola, mas com a vantagem de os dados ficarem na rede da escola, não sendo possível acessa-los de fora da rede escolar.

2.1.3 Proesc

“O proesc.com é uma ferramenta completa para gestão da educação. Uma forma simples e econômica de integrar todas as áreas da educação, proporcionando agilidade e qualidade nos serviços prestados pelas instituições de ensino”[3]. O Proesc possui vários módulos: gestão acadêmica online, gestão financeira escola, portal doso pais e alunos e, principalmente, gestão pedagógica escolar, que possui as funções de diário de classe.

O sistema Proesc também apresenta um aplicativo: Proesc professor, que apresenta as funções de diário de classe, além de disponibilizar opções offline. Contém opção de importar dados de sistemas prévios, ou ajudar os clientes a montar suas próprias bases de dados.

[2] DIARIO NOTA DEZ. Disponível em <https://www.diarionotadez.com.br/>. Acesso em 19/05/2018
[3] Citação retirada do site oficial. Disponível em < https://www.proesc.com/>. Acesso em 19/05/2018

2.1.4 Metasys Diário de Classe

"O Metasys Diário de Classe é um software que permite o lançamento de informações para o dia-a-dia do professor, como notas, frequência, ocorrências em sala de aula e planejamento das aulas, torna os processos escolares relacionados ao boletim e histórico escolar mais ágeis"[4]. Esse software foi criado pela MetaSys, empresa de soluções digitais educacionais.

O Metasys Diário de Classe pode ser usado por docentes, alunos e principalmente gestores de educação, que possuem acesso a todos os módulos do sistema. O sistema tem a opção de rodar em um servidor data center ou uma rede local, assim como o Proesc.

2.1.5 Quadro comparativa entre os softwares

Para visualizar melhor as diferentes funções de cada sistema citado na sessão acima, foi elaborado uma tabela comparativa, representada no quadro 1. O software Sistema de gestão escolar da Tecsoft será representado por S1; o software NotaDez de Anselmo Gomes será representado por S2; o software Proesc será representado por S3 e o software Diário de Classe da empresa Metasys será representado por S4. Essa tabela apresenta os recursos que cada sistema tem, representados por um X.

[4] Citação retirada do site oficial. Disponível em <http://www.metasys.com.br/produtos/ensino-aprendizagem/metasys-diario-classe/>. Acesso em 19/05/2018

Quadro 1 –Quadro comparativa entre os softwares.

	S1	S2	S3	**S4**
Funciona Offline			X	
Oferece funções financeiras	X	X		
Usado por outros funcionários			X	X
Integração com outros sistemas				X
Possui aplicativo			X	
Comunicação com os pais dos alunos	X	X	X	

Fonte: Elaborado pelo autor

3 PROJETO DO APLICATIVO

Este capítulo detalha o processo de desenvolvimento do software OClaRy. Esses detalhes são documentados usando a engenharia de software. Ela responsável por padronizar a documentação das várias etapas do desenvolvimento do software, como seus requisitos, sua metodologia e as ferramentas necessárias para o seu desenvolvimento.

E este capítulos também detalha mais do software em si, as suas funções e a sua aparência.

3.1 Metodologia de Desenvolvimento

A metodologia de desenvolvimento de um software se trata de um conjunto específico de regras e métodos coordenados para um desenvolvimento consistente de software. Na década de 1990, a maioria das metodologias presentes era voltadas para sistemas de grande porte, portanto, viu-se a necessidade de criar métodos ágeis, cujo foco era o software em si.

Sommerville (2007, p262) destaca que os métodos ágeis “foram criados, principalmente para apoiar o desenvolvimento de aplicações de negócio que os requisitos de sistema mudam rapidamente durante o processo de desenvolvimento” e trabalham com 5 princípios principais: envolvimento com o cliente; entrega incremental; pessoas, não processo; aceite as mudanças e mantenha a simplicidade. A quadro 2, desenvolvido por Sommerville, descreve cada uma dessas 5 etapas:

Quadro 2 – Princípios dos métodos ágeis”.

Princípio	**Descrição**
Envolvimento do cliente	Cliente devem ser profundamente envolvidos no processo de desenvolvimento. Seu papel é fornecer e priorizar novos requisitos do sistema e avaliar as iterações do sistema.

Entrega incremental	O software é desenvolvido em incrementos e o cliente especifica os requisitos a serem incluídos em cada incremento
Pessoas, não processo	As habilidades da equipe de desenvolvimento devem ser reconhecidas e exploradas. Os membros da equipe devem desenvolver suas próprias maneiras de trabalhar sem processos prescritivos.
Aceite as mudanças	Tenha em mente que os requisitos do sistema vão mudar, por isso projete o sistema para acomodar essas mudanças**.**
Mantenha a simplicidade	Concentre-se na simplicidade do software que está desenvolvido e do processo de desenvolvimento. Sempre que possível, trabalhe ativamente para eliminar a complexidade do sistema

Fonte: SOMMERVILLE (2007)

Para esse projeto, foi usada a metodologia ágil baseada em *Kanban* (sinal visual). É interessante notar que *Kanban* não começou como uma metodologia de desenvolvimento de software, mas sim como uma metodologia para uma linha de montagem da Toyota. Mas a sua adoção também é valida no sistema de software, pois proporciona aos desenvolvedores “opções de planejamento mais flexíveis, saída mais rápida, foco mais claro, e transparência ao longo do ciclo de desenvolvimento” (RADIGAN, 2017).

O *Kanban* consiste em um quadro, geralmente divididos em 3 colunas: a fazer; em progresso e feito. Cada atividade do desenvolvimento do software é um cartão, contendo o mínimo de informações possíveis para aquela atividade. Esses cartões são colocados nas colunas segundo o seu progresso. Isso promove uma visão ampla e transparente do progresso do projeto. A estrutura de colunas é flexível e podem ser mudadas dependendo das necessidades do projeto e da equipe.

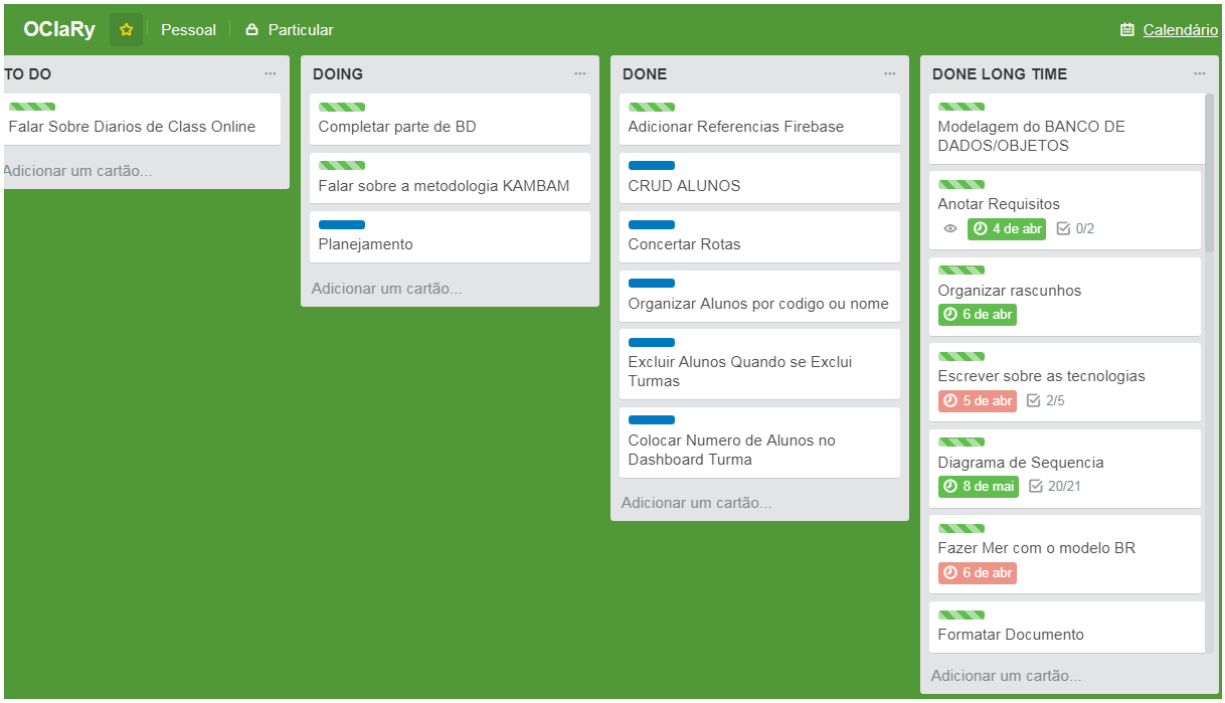
Os quadros do *Kanban* podem ser físicos ou virtuais, sendo os últimos mais fáceis de usar, pois podem ser vistos virtualmente por toda a equipe. Para esse projeto, foi usada a ferramenta online Trello para virtualizar o quadro *Kanbam*, pois apresenta vários recursos como etiquetas e data de entregas para os quadros.

Para esse projeto, foi criado no Trello um quadro com 4 colunas: *TO DO* (a fazer), *DOING* (em progresso), *DONE* (feito), e *DONE LONG TIME AGO* (feito há muito tempo). As atividades eram definidas semanalmente, seguindo o princípio de

entrega incremental descrito por Sommerville, e colocadas na coluna *TO DO*, quando a atividade estava sendo feita, era colocada na coluna *DOING*, e quando a atividade era concluída, era colocada na coluna *DONE*. A coluna *DONE LONG TIME AGO* não está presente no modelo tradicional, ela foi adicionada para armazenar as atividades já feitas depois de cada semana, desse jeito o quadro não fica poluído com varias atividades a muito tempo concluídas.

As atividades também eram separadas em: Programação e Documentação. Para identificar qual quadro pertencia a qual categoria, foram usadas etiquetas: verde para documentação e azuis para programação. A figura 6 apresenta o quadro do projeto com as atividades:

Figura 6 – Quadro kanban do projeto.



Fonte: Elaborado pelo autor

Nos próximos capítulos serão detalhados os processos de desenvolvimento do projeto, como a engenharia de requisitos, diagramas, ferramentas utilizadas assim como o cronograma do projeto.

3.2 Engenharia de Requisitos

A engenharia de requisitos faz parte do processo de engenharia de *software*, e tem como objetivo criar e manter o documento contendo os requisitos do sistema.

Os requisitos são basicamente a origem do sistema, pois são eles que irão definir as funções, as restrições, e até mesmo a aparência do sistema. Sommerville (2007) define os requisitos como sendo “o processo de comunicação entre os clientes e os usuários de software e os desenvolvedores de *software*”. A seguir estão descritos os requisitos do sistema, divididos entre requisitos funcionais e não funcionais.

3.2.1 Requisitos Funcionais

Os requisitos funcionais, como o nome sugere, são requisitos que se referem especificamente às funções do *software* solicitado: as entradas, as saídas e seus comportamentos específicos. Esses requisitos afetam diretamente as características do *software*, como o tipo de linguagem de programação usada e a plataforma a ser implantada, pois cada linguagem e cada plataforma possui suas vantagens e desvantagens, por isso esses requisitos devem ser detalhados. O quadro 3 detalha os requisitos funcionais do sistema.

Quadro 3 – Requisitos funcionais do projeto.

Identificação	Requisito Funcional	Categoria	Prioridade
RF001	Criar, Editar nome e Excluir escolas	Programação	Alta
RF002	Criar turmas em cada escola	Programação	Alta
RF003	Criar, Editar nome e Excluir Alunos	Programação	Alta
RF004	Criar, Editar e Excluir Provas	Programação	Alta
RF005	Atribuir pesos para cada prova	Programação	Alta
RF006	Atribuir conteúdo para cada prova	Design e Programação	Alta
RF007	Armazenar o arquivo (.doc ou .pdf) de cada prova	Design e Programação	Baixa
RF008	Calcular a média de cada aluno com base no peso de cada	Programação	Média

	prova.		
RF009	Atribuir presença aos alunos em cada dia	Design e Programação	Alta
RF010	Os dias de presença serão inseridos manualmente	Design e Programação	Média
RF011	Calcular a porcentagem de presença de cada aluno automaticamente	Programação	Média
RF012	Criar, Editar e Excluir dias de planejamento	Programação	Alta
RF013	Dias do planejamento serão inseridos manualmente	Design e Programação	Média
RF014	Armazenar presenças, média de presença, notas de provas e médias de cada aluno	Design e Programação	Alta
RF015	Mostrar os dias de planejamento próximos	Design e Programação	Baixa
RF016	Mostrar os dias de prova próximos	Design e Programação	Baixa
RF017	Mostrar os dias de planejamento próximos de uma turma específica	Design e Programação	Baixa
RF018	Mostrar os dias de prova próximos de uma turma específica	Design e Programação	Baixa

Fonte: Elaborado pelo autor

3.2.2 Requisitos Não Funcionais

Requisitos não funcionais, por outro lado, se referem a características mais abrangentes do sistema, e geralmente são mais importantes que os requisitos funcionais porque, segundo Somerville, se um requisito não funcional não for atendido, pode significar a inutilidade do sistema. O quadro 4 detalha os requisitos não funcionais do sistema.

Quadro 4 – Requisitos não funcionais do projeto.

Identificação	Requisito não funcional	Categoria	Prioridade
RNF001	Ser flexível quanto a inserção de datas	Usabilidade	Alta
RNF002	Ser intuitivo	Usabilidade	Alta
RNF003	Apresentar erros bem definidos com instruções claras	Confiabilidade	Média

RNF004	Ser responsivo, funcionando em telas de resoluções diferentes	Distribuição	Média
RNF005	O sistema deve informar quando a tela está carregando os dados	Desempenho	Baixa
RNF006	O sistema deve rodar em plataforma web, nos navegadores Google Chrome e Mozilla Firefox	Distribuição	Baixa

Fonte: Elaborado pelo autor

3.3 Planejamento

O planejamento consiste um modelo adaptado para esse projeto, usando metodologia ágil com kanban. As atividades do planejamento foram definidas segundo os requisitos do sistema e foram anotadas em um cronograma para servir de base e visão do andamento do projeto.

3.3.1 Recursos e Ferramentas

Esta seção detalha as ferramentas usadas no desenvolvimento do software, como linguagens de programação e banco de dados.

3.3.1.1 JavaScript

A plataforma Android Javascript é uma linguagem de programação interpretada, concebida inicialmente para que scripts pudessem ser rodados em páginas *web*, e agora é uma das linguagens mais utilizadas no mundo.

Javascript é uma linguagem bastante flexível e completa, “deixando espaço para várias técnicas que são impossíveis em linguagens mais rígidas” (HAVERBEKE.2018, p6). É uma linguagem de script, ou seja, seu código é lido em tempo de execução. É, ao mesmo tempo orientada a objetos e baseada em prototipagem, isso significa que os objetos em Javascript são herdados por protótipos ao invés de outros objetos ou classes.

O Javascript é usado tanto no *client-side* como no *server-side* atualmente. No *client-side*, ele altera o Modelo de Objeto de Documentos(DOM) do HTML, sendo usado para modificar o comportamento da página a determinados eventos. No *server-side,* ele é usado geralmente para programar um servidor capaz de lidar com várias conexões simultâneas.

O Javascript foi escolhido para esse projeto por sua flexibilidade para manipular dados no DOM do HTML, e por possuir *frameworks* mais completos.

3.3.1.2 Vue JS

Para manipular melhor uma linguagem, normalmente usa-se *frameworks*. *Frameworks* são um conjunto de regras complexas que moldam o código. *Frameworks* são extremamente comuns no mundo da programação, pois automatizam ou facilitam várias tarefas e módulos.

Já que Javascript é uma das linguagens mais populares do mundo, é fácil de se pensar que ela disponibiliza de *frameworks*. Os 3 mais populares são “React”, “Angular” e Vue.js.

O *framework* escolhido para esse projeto foi o Vue.js. Vue.js é um *framework* reativo de Javascript “capaz de suportar a construção fim-a-fim de aplicações *web* complexas” (FILIPOVA.2016, p.10). Ele executa as mesmas funções que outros *frameworks*, como eventos, ligação de dados, criação de componentes e etc.

Vue.js foi escolhido para esse projeto pois é mais simples do que os outros *frameworks*, além de ter um desempenho melhor.

3.3.1.3 MaterializeCss

Para tornar a interface mais agradável e intuitiva para o usuário, foi o usado uma metodologia de design “Material Design”. Esse padrão foi criado em 2014 pela empresa Google e implementado nos sistemas operacionais *mobile* Android e atualmente é uma das maiores tendências de *design*.

O material design se caracteriza por usar formas geométricas simples e cores sólidas. Seus elementos ocupam espaços únicos no ambiente e possuem luz e sombra, dando sensação de certa profundidade.

Para aplicar o “Material Design” no projeto, foi usado o *framework* css Materialize. Esse *framework*, criado por um grupo de estudantes da *Carnegie Mellon University*, aplica o “Material Design” em páginas, usando o sistema de *grids*, tornando a página responsiva. O Materialize também permite criar componentes responsivos, como botões, listas, tabelas, etc.

O Materialize foi escolhido para esse projeto pois além de apresentar um método consistente para aplicar o “Material Design”, seu uso torna fácil tornar as páginas responsivas, além de incrementar a usabilidade da interface como um todo.

3.3.1.4 FireBase DataBase

Para guardar os dados do projeto, foi usado o banco de dados *NoSql* do Firebase. O Firebase é uma plataforma na nuvem do Google que permite o desenvolvimento de aplicações web oferecendo um *back-end* completo, com “serviços comuns que um desenvolvedor pode precisar, como banco de dados, autenticação e etc.” (MORONEY, 2017, p. 1)

Para esse projeto específico, foi usada apenas a parte de banco de dados da plataforma. O banco de dados do Firebase é *NoSql*, ou seja, um banco de dados não relacional permitindo uma escalabilidade mais barata e menos cansativa.

3.3.2 Cronograma

O cronograma do projeto é responsável por localizar e organizar as atividades do projeto. Também é responsável por informar a sequência em que são executadas, o tempo que duram, com uma data inicial e uma data final e também a porcentagem concluída. A figura 1 apresenta o cronograma desse projeto.

Figura 7 – Cronograma do projeto.

Tipo de Tarefa	Nome da Tarefa	Inicio	Fim	Status
	Documentação	23/03/18	02/04/18	
Documentação	Definição de Escopo	24/03/18	29/03/18	100%
Documentação	Definição de requisitos	30/03/18	02/04/18	100%
	Planejamento/Modelagem	27/03/18	09/04/18	
Documentação	Definição das ferramentas	06/04/18	08/04/18	100%
	Diagramas			
Documentação	Diagramas de Caso de Uso	27/03/18	04/04/18	100%
Documentação	Diagrmas de Sequencia	04/04/18	09/04/18	100%
Programação	Montar Banco de dados	04/04/18	07/04/18	100%
	Protótipo	09/03/18	16/03/18	
Programação	Rascunhar prototipos	09/03/18	16/03/18	100%
	Desenvolvimento	24/04/18	27/05/18	
Programação	Montar Ambiente de Desenvolvimento	24/04/18	24/04/18	100%
Programação	Testar Conexão com o FireBase	25/04/18	25/04/18	100%
	Telas			
Programação	CRUD Escola	27/04/18	04/05/18	100%
Programação	CRUD Turma	04/05/18	11/04/18	100%
Programação	CRUD Alunos	12/05/18	14/05/18	100%
Programação	CRUD Planejamento	14/05/18	16/05/18	100%
Programação	CRUD Prova	16/05/18	18/05/18	100%
Programação	CRUD Presença	19/05/18	21/05/18	100%
Programação	CRUD Notas	22/05/18	24/05/18	100%
Programação	Pagina de Login	25/05/18	27/05/18	100%

Fonte: Elaborado pelo autor

3.4 Modelagem

A modelagem é uma parte do sistema quase tão importante quanto o seu desenvolvimento. Ela é responsável por abstrair o sistema, oferendo vários pontos de vista sobre o mesmo, como a sua estrutura e o seu funcionamento.

E para padronizar a modelagem, usa-se a UML (*Unifield Modelling Language* – *Linguegem* - Unificada de Modelagem) que é uma linguagem padrão que “através de sua semântica bem definida” (MELO,2004), define regras para a modelagem de

sistemas. Essa linguagem consiste em diagramas focando em aspectos diferentes do sistema.

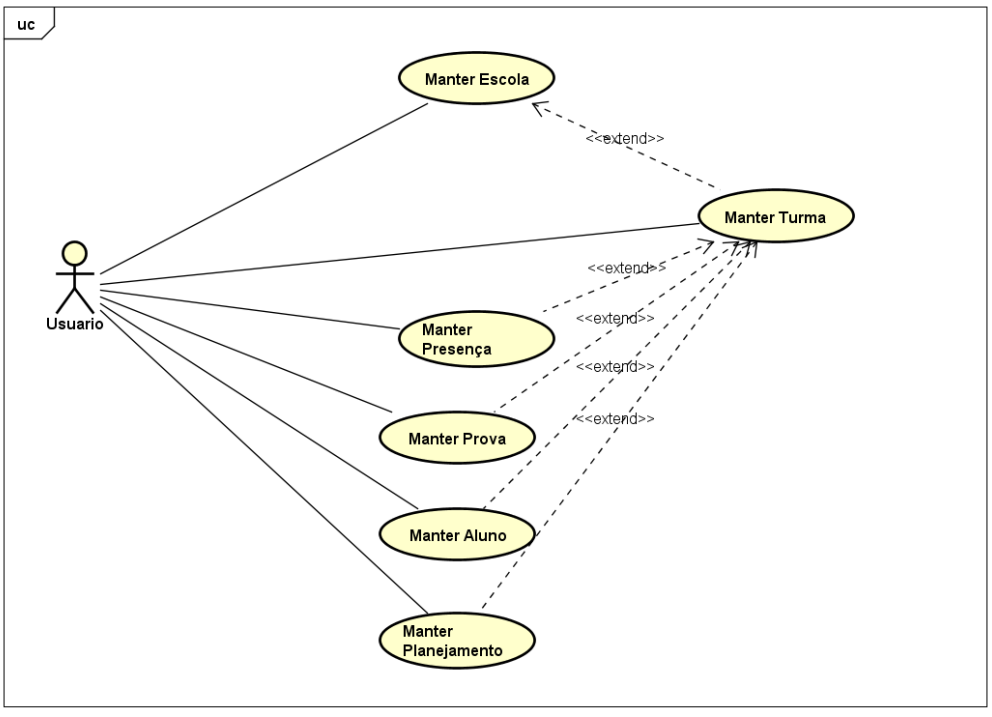
3.4.1 Casos De Uso

O diagrama de caso de uso, segundo Gilleanes (2009, p.55), “entre todos os diagramas UML, o mais abstrato e, portanto, o mais flexível e informal”. Esses diagramas oferecem uma visão geral das funcionalidades do sistema, sem detalhar muito a sua programação.

Os diagramas de caso de uso são muito ligados aos requisitos funcionais do sistema, que são representados por uma elipse, enquanto os atores (qualquer pessoa ou objeto exterior ao sistema que possua contato com o mesmo) são representados por um bonequinho.

A figura 8 representa o diagrama de caso uso de todas as funcionalidades do sistema, onde a palavra manter se refere às operações: criar, editar e excluir.

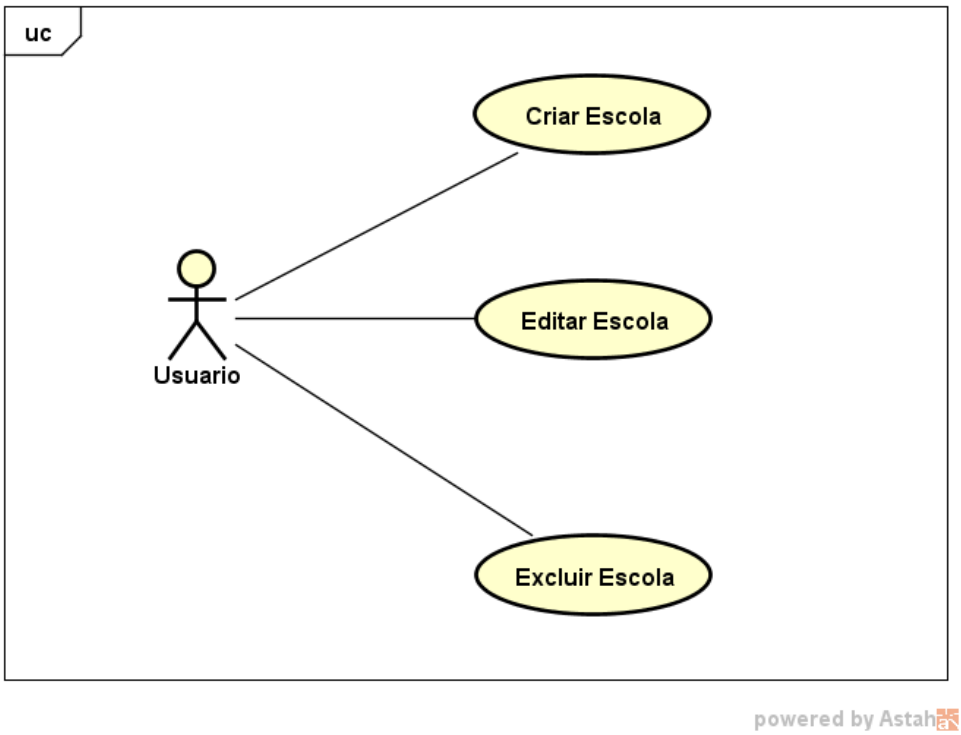
Figura 8 – Caso de uso de funcionalidades do aplicativo.



Fonte: Elaborado pelo autor

A figura 9 a presenta o caso de uso que representa as funcionalidades referentes ao cadastro de escolas.

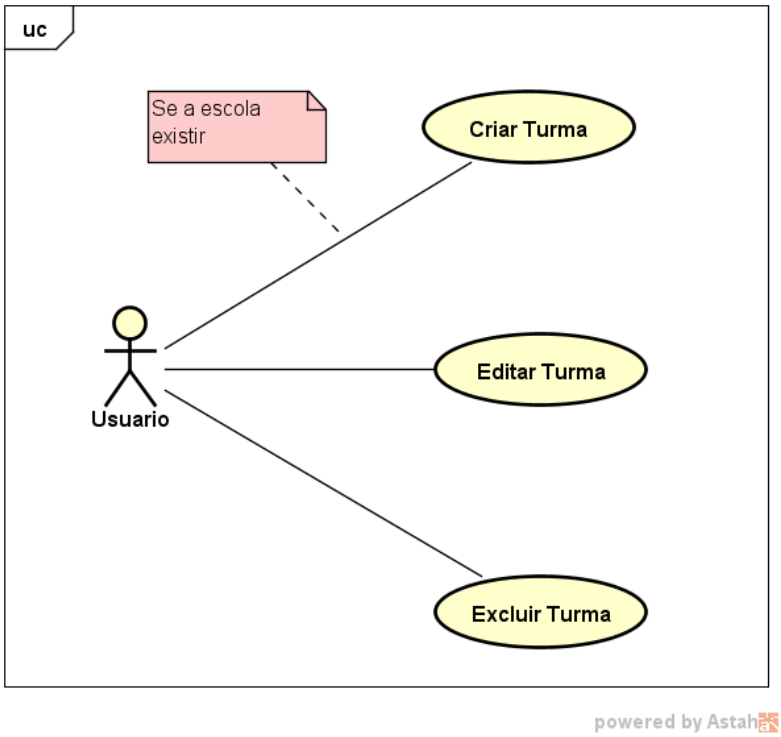
Figura 9 – Caso de uso de manutenção de escolas



Fonte: Elaborado pelo autor

A figura 10 a presenta o caso de uso que representa as funcionalidades referentes ao cadastro de turmas.

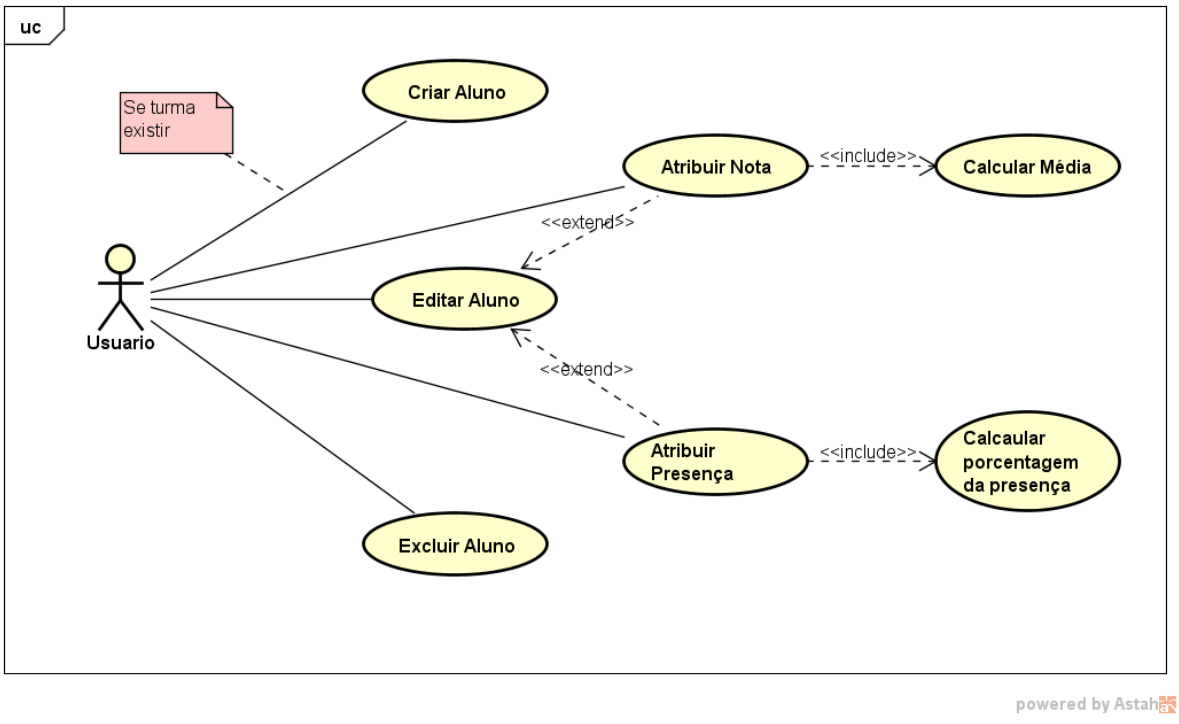
Figura 10 – Caso de uso de manutenção de turmas.



Fonte: Elaborado pelo autor

A figura 11 a presenta o caso de uso que representa as funcionalidades referentes ao cadastro de alunos.

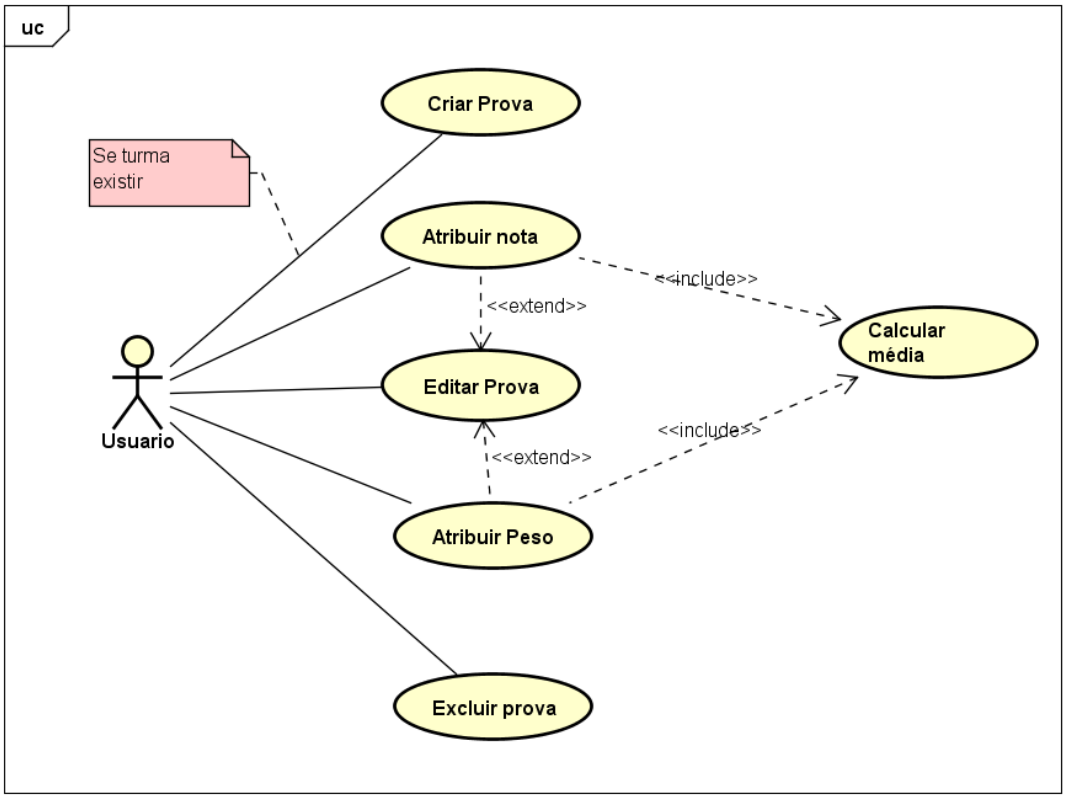
Figura 11 – Caso de uso de manutenção de alunos.



Fonte: Elaborado pelo autor

A figura 12 a presenta o caso de uso que representa as funcionalidades referentes ao cadastro de provas.

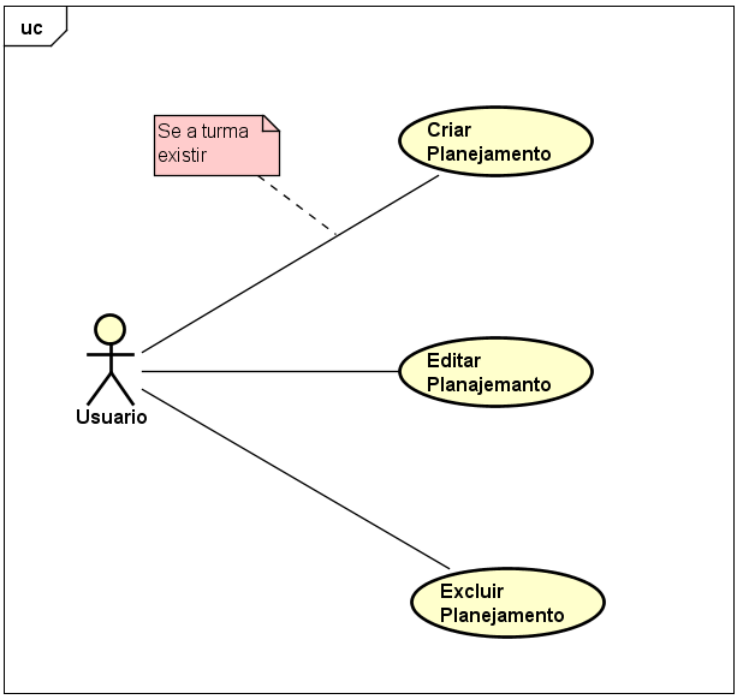
Figura 12 – Caso de uso de manutenção de provas.



Fonte: Elaborado pelo autor

A figura 13 a presenta o caso de uso que representa as funcionalidades referentes ao cadastro de planejamento.

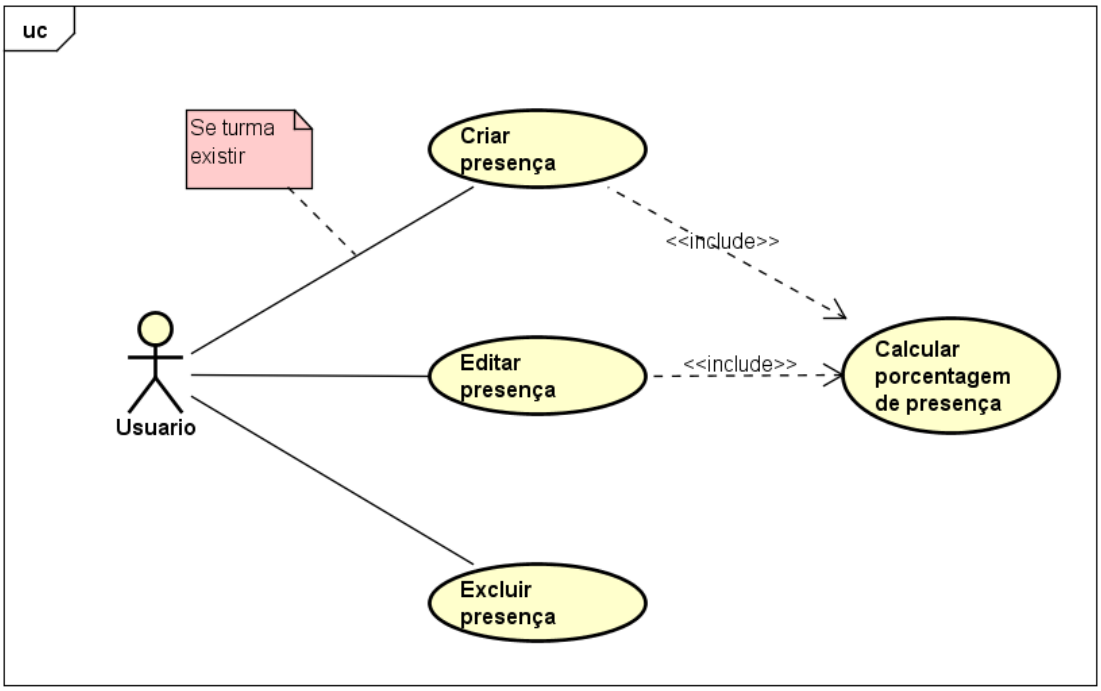
Figura 13 – Caso de uso da manutenção do planejamento.



Fonte: Elaborado pelo autor

A figura 14 apresenta o caso de uso que representa as funcionalidades referente ao cadastro de presença.

Figura 14 – Caso de uso de manutenção de presença.



Fonte: Elaborado pelo autor

Na próxima subseção, será apresentado a documentação dos casos de uso do sistema.

3.4.2 Documentação dos Casos de Uso

Cada caso de uso de sistema, ou funcionalidade, consiste em um conjunto de ações a ser executado, quer pelo sistema, quer pelo usuário, ou ambos. Para descrever essas etapas, se faz necessário a documentação dos casos de uso que “por meio de uma linguagem bastante simples” descreve as funcionalidades, as ações e os atores de cada caso de uso. (Gileannes, 2009, p.58).

Em todo formulário presente no sistema, com a função de criar, editar ou excluir alguma coisa, possui um botão Cancelar, que cancela todas as alterações e volta para a tela anterior.

A seguir, são apresentados a documentação de cada caso de uso. Os quadros 5 e 6 são referentes à escola; os quadros 7 e 8 são referentes à turma; os

quadros 9 a 15 são referentes aos alunos; os quadros 16 a 19 são referentes à prova; os quadros 20 e 21 são referentes ao planejamento e os quadros 22 e 23 são referentes à presença.

Quadro 5 – Caso de uso "Criar/ Editar Escola".

Nome do caso de uso	Criar/Editar Escola
Atores envolvidos	Usuário
Descrição	Criar/editar escola para armazenar turmas
Prioridade de desenvolvimento	Essencial
Ações do ator	**Ações do Sistema**
1. Solicita a criação/edição de escola	
	2. Apresenta o formulário para criação/ edição da escola
3. Inserir o nome da escola e confirma a criação/edição	
	4. Criar/editar a escola
Validações	Verificar se não há outra escola com o mesmo nome

Fonte: Elaborado pelo autor

Quadro 6 – Caso de uso "Excluir Escola".

Nome do caso de uso	Excluir Escola
Atores envolvidos	Usuário
Descrição	Excluir escola e suas turmas dependentes
Prioridade de desenvolvimento	Essencial
Ações do ator	**Ações do Sistema**
1. Solicita a exclusão da escola	
	2. Apresenta uma mensagem de confirmação

3. Confirmar exclusão	
	4. Excluir escola e turmas dependentes

Fonte: Elaborado pelo autor

Quadro 7 – Caso de uso "Criar/ Editar Turma".

Nome do caso de uso	Criar/Editar Turma
Atores envolvidos	Usuário
Descrição	Criar/editar Turmas, que vão armazenar alunos, planejamento, presença, provas e notas
Prioridade de desenvolvimento	Essencial
Ações do ator	**Ações do Sistema**
1. Solicita a criação/edição de escola	
	2. Apresenta o formulário para criação/edição da escola
3. Escolher qual escola a turma pertence	
4. Inserir Nome da turma	
5. Inserir Disciplina	
6. Confirmar criação/edição	
	7. Criar/editar turma
Validações	Verificar se não há outra escola com o mesmo nome

Fonte: Elaborado pelo autor

Quadro 8 – Caso de uso "Excluir Turma".

Nome do caso de uso	Excluir Turma
Atores envolvidos	Usuário
Descrição	Excluir Turma e seus dados (Alunos, Planejamento, Presença, Provas e Notas) dependentes

Prioridade de desenvolvimento	Essencial
Ações do ator	**Ações do Sistema**
1. Solicita a exclusão da Turma	
	2. Apresenta uma mensagem de confirmação
3. Confirmar exclusão	
	4. Excluir Turma e dados Dependentes

Fonte: Elaborado pelo autor

Quadro 9 – Caso de uso “Criar Aluno”.

Nome do caso de uso	Criar Aluno
Atores envolvidos	Usuário
Descrição	Criar Aluno em uma turma
Prioridade de desenvolvimento	Essencial
Ações do ator	**Ações do Sistema**
1. Solicita a criação do aluno	
	2. Apresenta o formulário para criação do aluno
3. Inserir código do aluno (número de chamada, RA, etc.)	
4. Inserir nome do aluno e confirmar criação	
	5. Criar aluno
Validações	Verificar se não há outro aluno com o mesmo código

Fonte: Elaborado pelo autor

Quadro 10 – Caso de uso “Editar Aluno”.

Nome do caso de uso	Editar Aluno

Atores envolvidos	Usuário
Descrição	Editar dados de um aluno específico
Prioridade de desenvolvimento	Essencial
Ações do ator	**Ações do Sistema**
1. Solicita a edição do aluno	
	2. Apresentar a janela de edição do aluno
3. Alterar código e/ou nome do aluno	
4. Confirmar alteração	
	5. Editar dados do aluno
Fluxo alternativo – Alterar nota	
	1.Excutar caso de uso Atribuir Nota
Fluxo alternativo – Alterar presença	
	1.Excutar caso de uso Atribuir Presença

Fonte: Elaborado pelo autor

Quadro 11 – Caso de uso “Excluir Aluno”.

Nome do caso de uso	Excluir Aluno
Atores envolvidos	Usuário
Descrição	Excluir aluno, suas notas e presenças
Prioridade de desenvolvimento	Essencial
Ações do ator	**Ações do Sistema**
1. Solicita a exclusão do aluno	
	2. Apresenta uma mensagem de confirmação

3. Confirmar exclusão	
	4. Excluir presença e notas do aluno
	5. Excluir aluno

Fonte: Elaborado pelo autor

Quadro 12 – Caso de uso "Atribuir Nota" - janela dos Alunos.

Nome do caso de uso	Atribuir Nota (Janela dos Alunos)
Atores envolvidos	Usuário
Descrição	Atribuir nota aos alunos em cada prova
Prioridade de desenvolvimento	Médio
Ações do ator	**Ações do Sistema**
1.Solicitar alteração de notas	
	2. Apresentar janela com notas dos alunos (ou aluno específico)
3. Alterar uma ou mais notas do aluno e confirmar edição	
	4. Confirmar edição das notas
	5. Executar caso de uso Calcular Média

Fonte: Elaborado pelo autor

Quadro 13 – Caso de uso "Atribuir Presença".

Nome do caso de uso	Atribuir presença
Atores envolvidos	Usuário
Descrição	Atribuir presença em dias já existentes
Prioridade de desenvolvimento	Médio
Ações do ator	**Ações do Sistema**

1.Solicitar alteração de presença	
	2. Apresentar janela com presenças dos alunos
3. Alterar uma ou mais presenças do aluno e confirmar edição	
	4. Confirmar edição das presenças
	5. Executar caso de uso Calcular Média Presença

Fonte: Elaborado pelo autor

Quadro 14 – Caso de uso "Calcular Média".

Nome do caso de uso	Calcular média
Atores envolvidos	
Descrição	Calcula a média de todas as provas
Prioridade de desenvolvimento	Médio
Ações do ator	**Ações do Sistema**
	1.Calcula a média das provas com base no peso que elas têm e nas notas que os alunos tiram

Fonte: Elaborado pelo autor

Quadro 15 – Caso de uso "Calcular Porcentagem de presença".

Nome do caso de uso	Calcular porcentagem de presença
Atores envolvidos	
Descrição	Calcula a porcentagem de presença dos alunos
Prioridade de desenvolvimento	Médio
Ações do ator	**Ações do Sistema**
	1.Calcula a porcentagem de presença dos alunos com base no total de dias inseridos na janela presença

Fonte: Elaborado pelo autor

Quadro 16 – Caso de uso “Criar/Editar Prova”.

Nome do caso de uso	Criar/Editar prova
Atores envolvidos	Usuário
Descrição	Criar/editar prova em uma turma
Prioridade de desenvolvimento	Essencial
Ações do ator	**Ações do Sistema**
1. Solicita a criação/edição da prova	
	2. Apresenta o formulário para criação/edição da prova
3. Inserir data da prova(Opcional)	
4. Inserir nome da prova	
5.Inserir conteúdo da prova (Opcional)	
6. Inserir Peso da prova	
	7. Confirmar criação/edição da prova
Validações	Verificar se não há outra prova com o mesmo nome
Fluxo alternativo – Alterar nota	
	1.Executar caso de uso Atribuir Nota
Fluxo alternativo – Alterar peso	
	1.Executar caso de uso Alterar Peso da prova

Fonte: Elaborado pelo autor

Quadro 17 – Caso de uso “Alterar peso da prova”.

Nome do caso de uso	Alterar peso da prova
Atores envolvidos	Usuário

Descrição	Altera o peso da prova
Prioridade de desenvolvimento	Médio
Ações do ator	**Ações do Sistema**
1. Inserir Peso da prova	
	2. Executar caso de uso Calcular Média

Fonte: Elaborado pelo autor

Quadro 18 – Caso de uso "Atribuir Nota" – janela das Provas.

Nome do caso de uso	Atribuir Nota (Janela das Provas)
Atores envolvidos	Usuário
Descrição	Atribuir nota aos alunos em cada prova
Prioridade de desenvolvimento	Médio
Ações do ator	**Ações do Sistema**
1.Solicitar alteração de notas	
	2. Apresentar janela com notas dos alunos em relação àquela prova específica
3. Alterar uma ou mais notas do aluno e confirmar edição	
	4. Confirmar edição das notas
	5. Executar caso de uso Calcular Média

Fonte: Elaborado pelo autor

Quadro 19 – Caso de uso "Excluir Prova".

Nome do caso de uso	Excluir prova
Atores envolvidos	Usuário
Descrição	Excluir prova e suas notas
Prioridade de desenvolvimento	Essencial

Ações do ator	Ações do Sistema
1. Solicita a exclusão da prova	
	2. Apresenta uma mensagem de confirmação
3. Confirmar exclusão	
	4. Excluir nota
	5. Executar caso de uso Calcular média
	6. Excluir prova

Fonte: Elaborado pelo autor

Quadro 20 – Caso de uso "Criar/ Editar Planejamento".

Nome do caso de uso	Criar/Editar planejamento
Atores envolvidos	Usuário
Descrição	Criar/editar entradas para planejamento
Prioridade de desenvolvimento	Essencial
Ações do ator	**Ações do Sistema**
1. Solicita a criação/edição de planejamento	
	2. Apresenta o formulário para criação/edição do planejamento
3. Inserir/Editar data	
4. Inserir conteúdo(Opcional)	
5. Marcar se o conteúdo já foi aplicado ou não	
	7. Criar/editar turma
Validações	Verificar se não há outra entrada de planejamento com a mesma data

Fonte: Elaborado pelo autor

Quadro 21 – Caso de uso “Excluir Planejamento”.

Nome do caso de uso	Excluir planejamento
Atores envolvidos	Usuário
Descrição	Excluir entradas no planejamento
Prioridade de desenvolvimento	Essencial
Ações do ator	**Ações do Sistema**
1. Solicita a exclusão da entrada do planejamento	
	2. Apresenta uma mensagem de confirmação
3. Confirmar exclusão	
	4. Excluir entrada do planejamento

Fonte: Elaborado pelo autor

Quadro 22 – Caso de uso “Criar/ Editar Presença”.

Nome do caso de uso	Criar/Editar presença
Atores envolvidos	Usuário
Descrição	Criar/editar entradas de presença dos alunos em uma turma
Prioridade de desenvolvimento	Essencial
Ações do ator	**Ações do Sistema**
1. Solicita a criação/edição de entrada de presença	
	2. Apresenta o formulário para criação/edição da entrada de presença
3. Escolher data	
4. Inserir presença e falta para os alunos	
	5. Criar/editar turma

	6. Executar caso de uso Calcular Porcentagem de presença
Validações	Verificar se não há outra entrada de presença com a mesma data

Fonte: Elaborado pelo autor

Quadro 23 – Caso de uso “Excluir Presença”.

Nome do caso de uso	Excluir Turma
Atores envolvidos	Usuário
Descrição	Excluir entrada de presença
Prioridade de desenvolvimento	Essencial
Ações do ator	**Ações do Sistema**
1. Solicita a exclusão da Turma	
	2. Apresenta uma mensagem de confirmação
3. Confirmar exclusão	
	4. Excluir Turma e dados Dependentes
	5. Executar caso de uso Calcular Porcentagem de presença

Fonte: Elaborado pelo autor

3.4.3 Documentação de Sequência

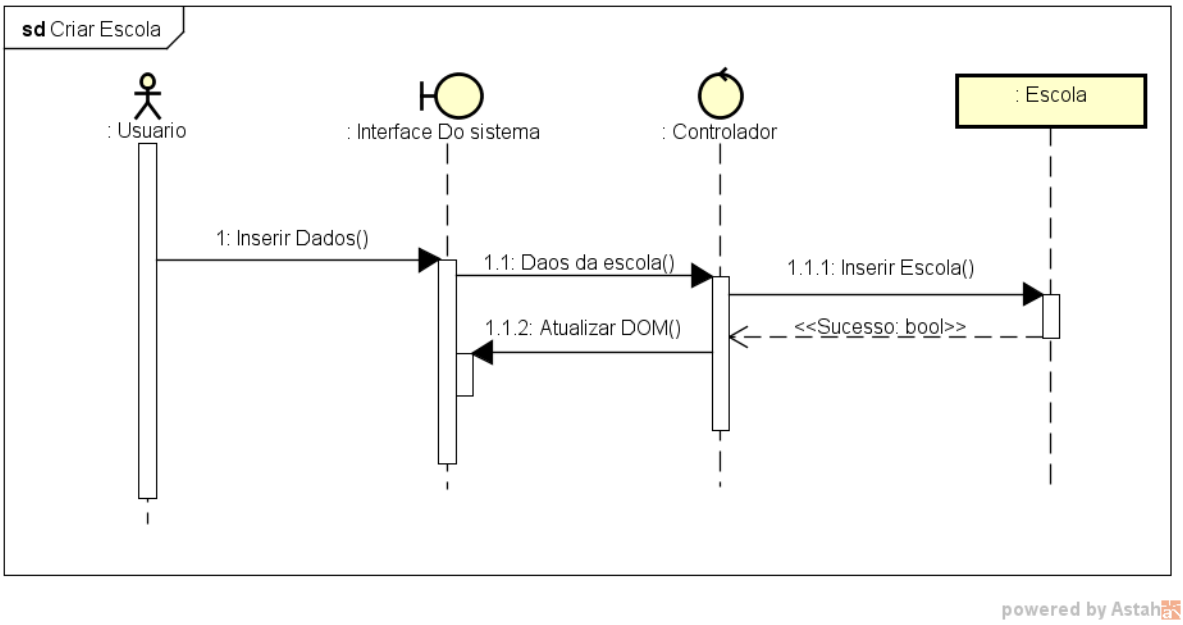
O diagrama de sequência é mais um dos diagramas da UML. Como o nome sugere, o diagrama de sequência mapeia a ordem que ocorrem os eventos quando um determinado processo ocorre. O diagrama de sequência determina a ordem dos ventos, as mensagens trocadas entre as entidades e os métodos chamados em um processo.

Gilleanes(2009, p. 200) aponta que o diagrama de sequência possui forte relação como diagrama de casos de uso, “havendo [...] um diagrama de sequência

para cada diagrama de caso de uso". Isso pode ser notado pela presença do ator(bonequinho) nos diagramas de sequência.

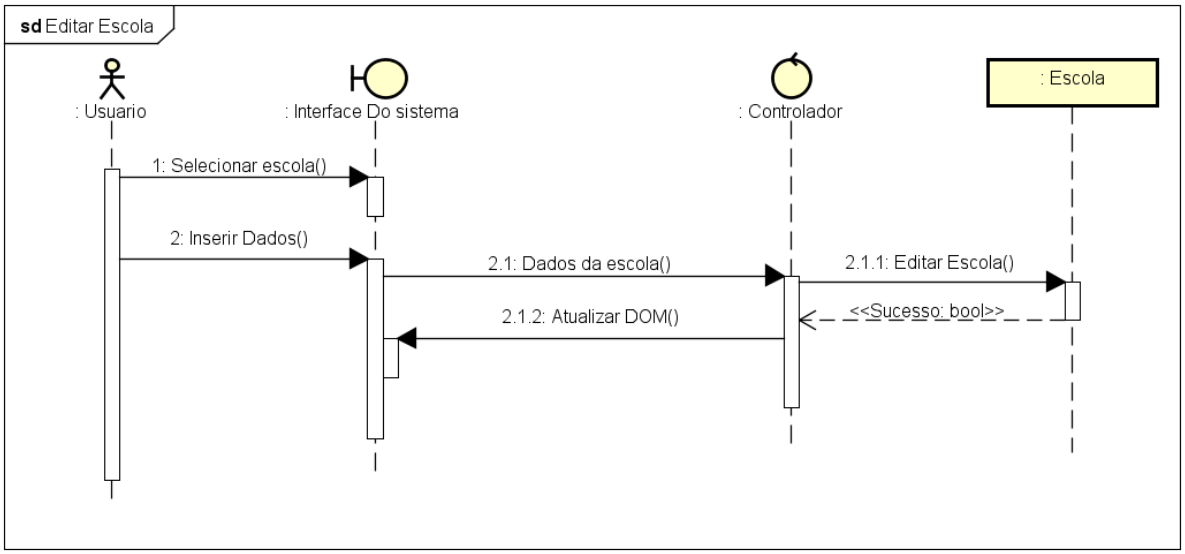
A seguir, são apresentados os diagramas de sequência para cada operação do sistema. As figuras 15 a 17 são referentes à escola; as figuras 18 a 20 são referentes à turma, as figuras 21 a 23 são referentes à aluno; as figuras 24 e 25 são referentes à nota; as figuras 26 a 28 são referentes à prova; as figuras 29 a 31 são referentes à planejamento; as figuras 32 a 34 são referentes à presença.

Figura 15 – Diagrama de sequência da operação criar escola.



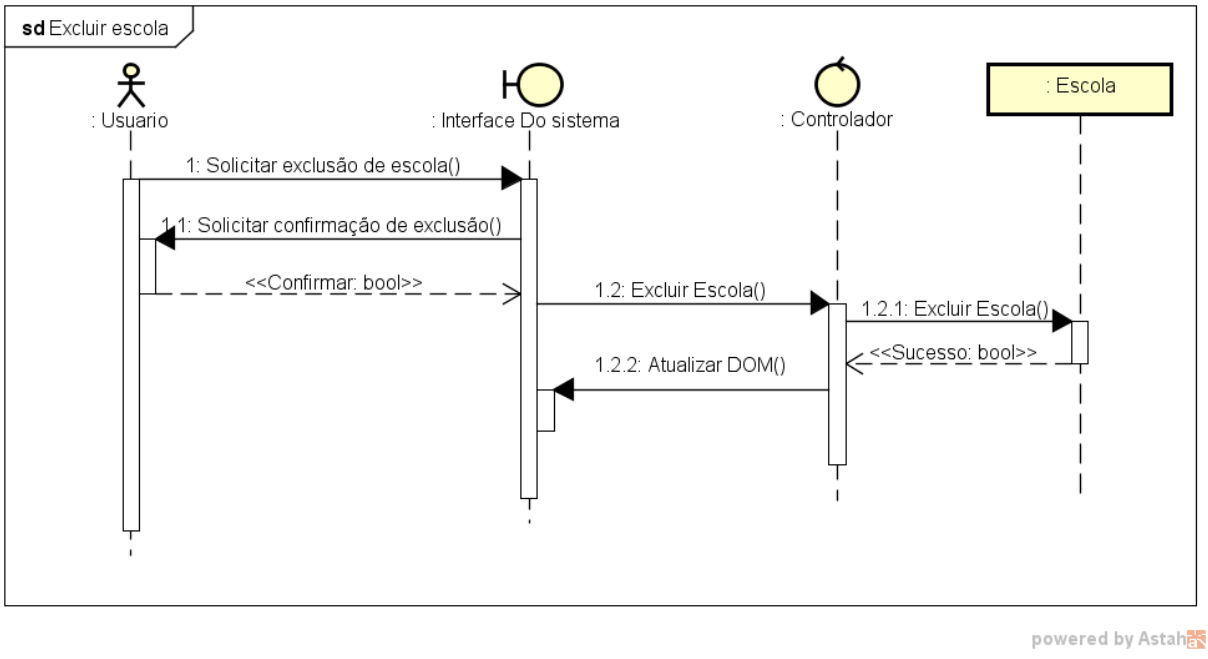
Fonte: Elaborado pelo autor

Figura 16 – Diagrama de sequência da operação editar escola.



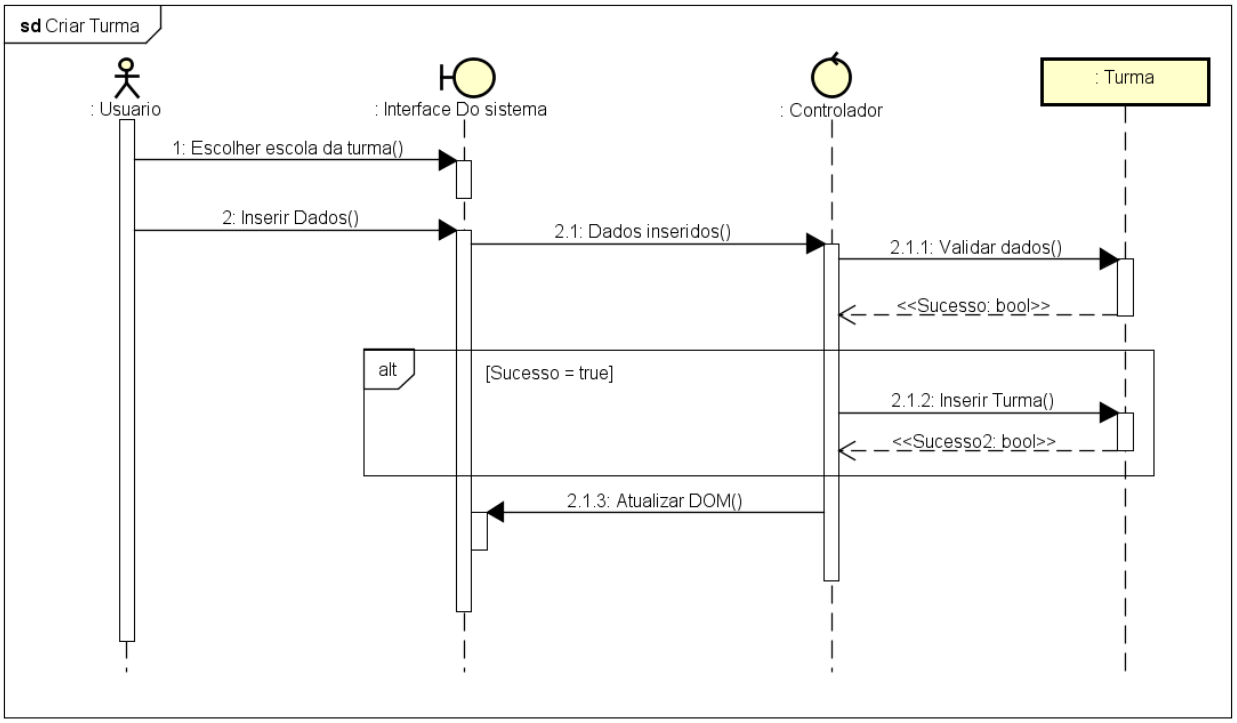
Fonte: Elaborado pelo autor

Figura 17 – Diagrama de sequência da operação excluir escola.



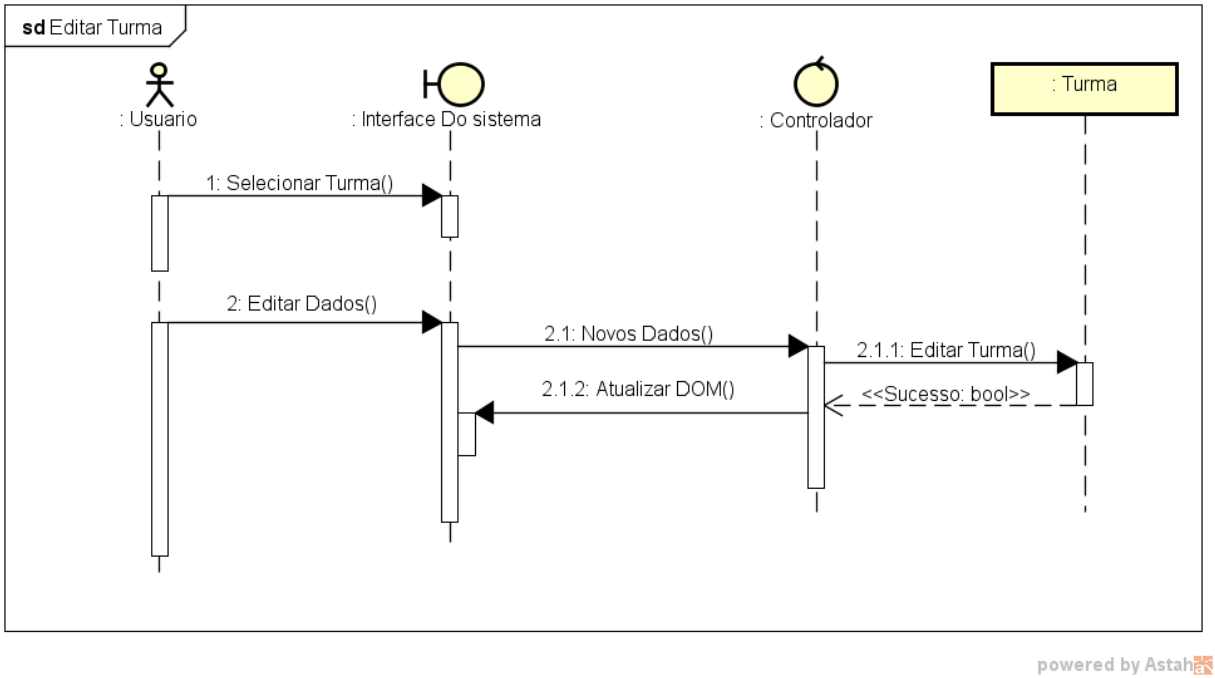
Fonte: Elaborado pelo autor

Figura 18 – Diagrama de sequência da operação criar turma.



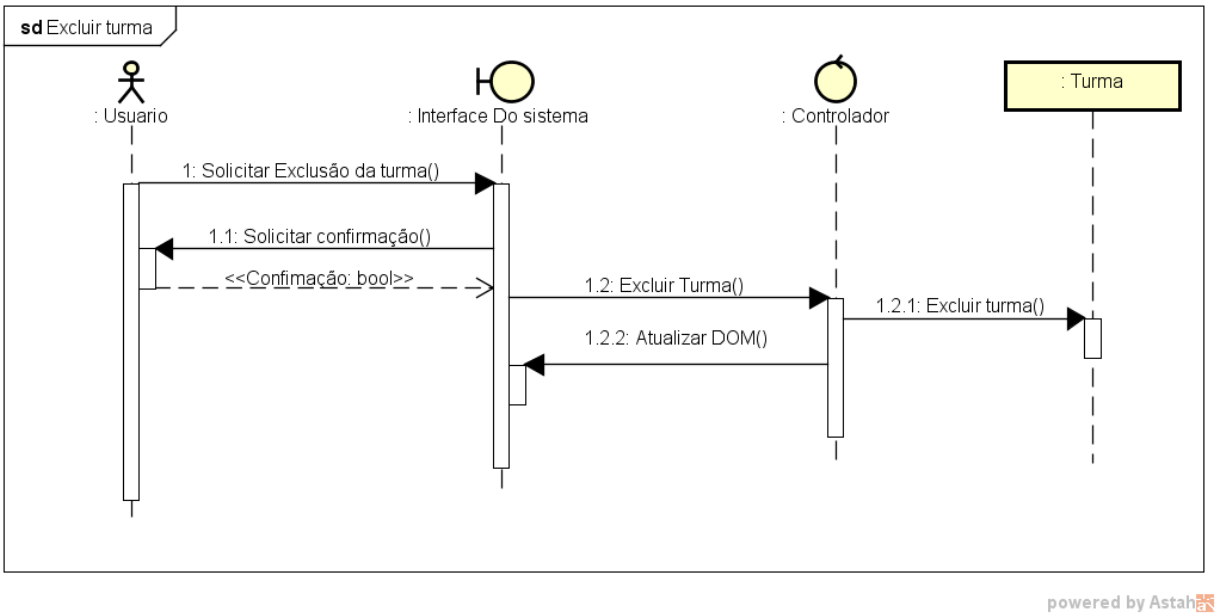
Fonte: Elaborado pelo autor

Figura 19 – Diagrama de sequência da operação editar turma.



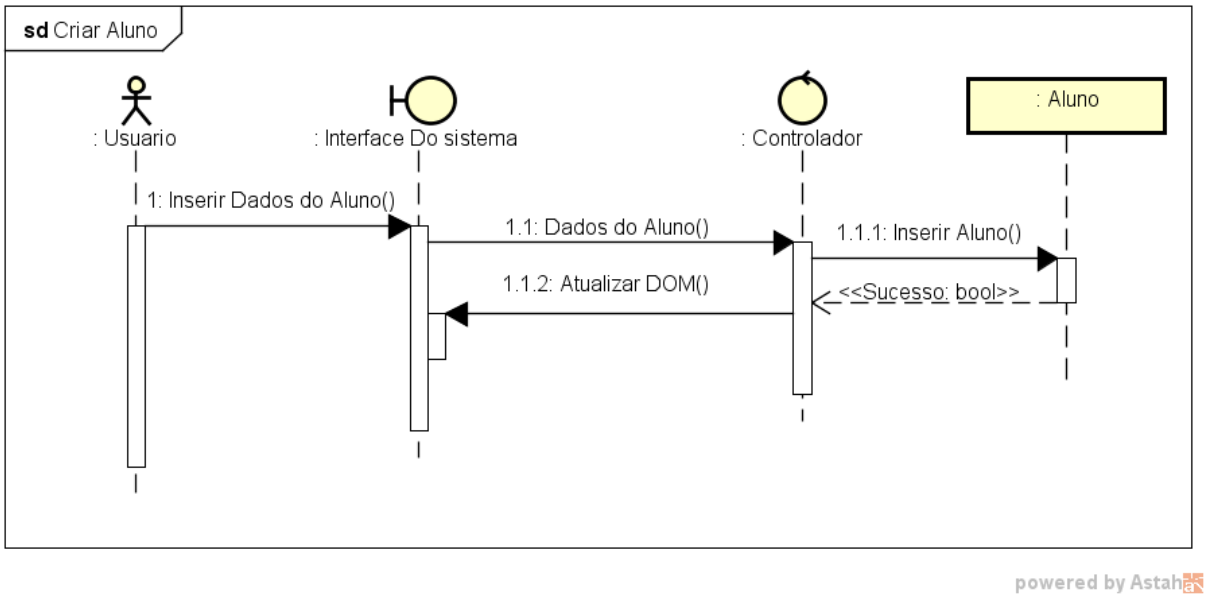
Fonte: Elaborado pelo autor

Figura 20 – Diagrama de sequência da operação excluir turma.



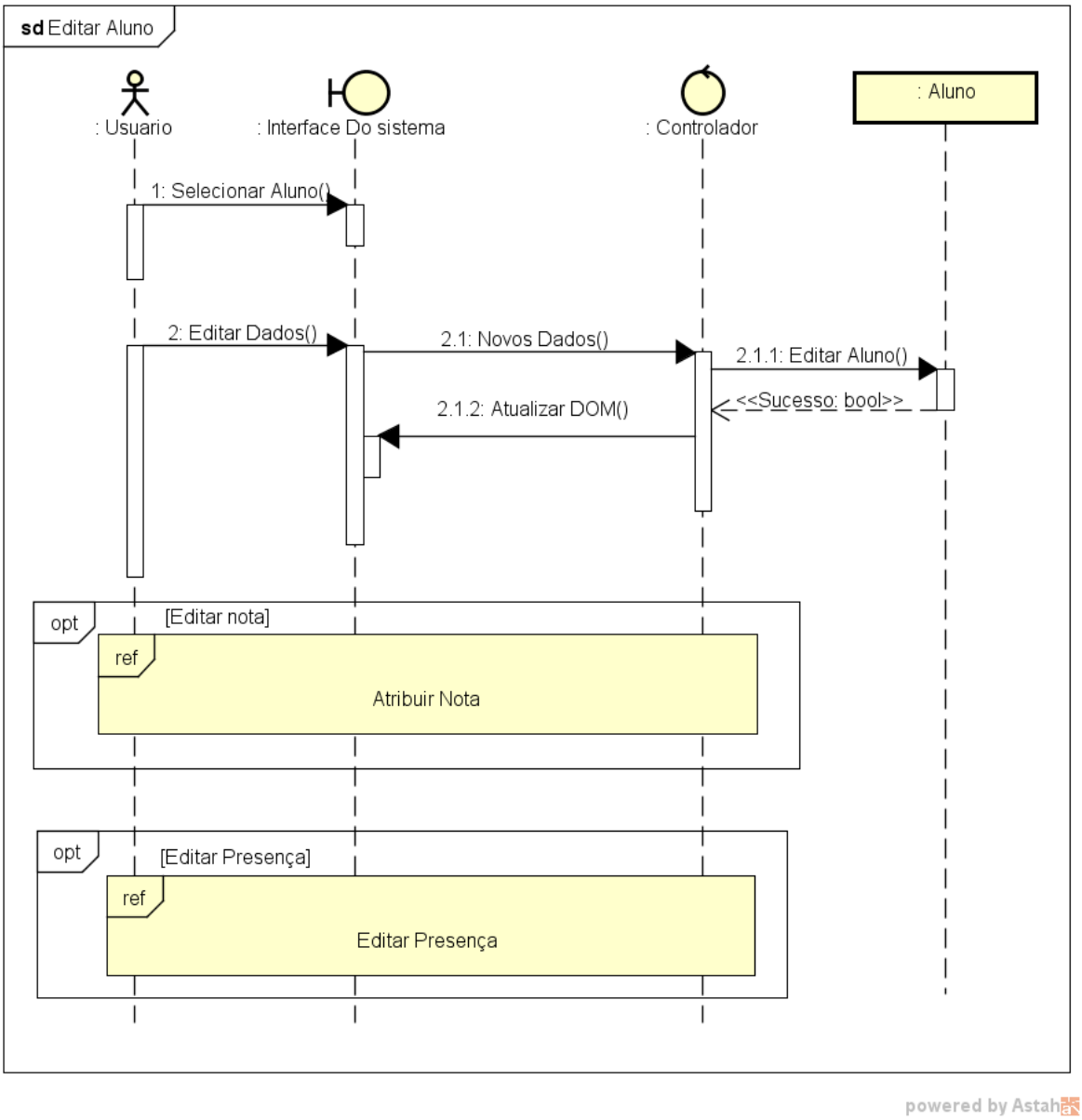
Fonte: Elaborado pelo autor

Figura 21 – Diagrama de sequência da operação criar aluno.



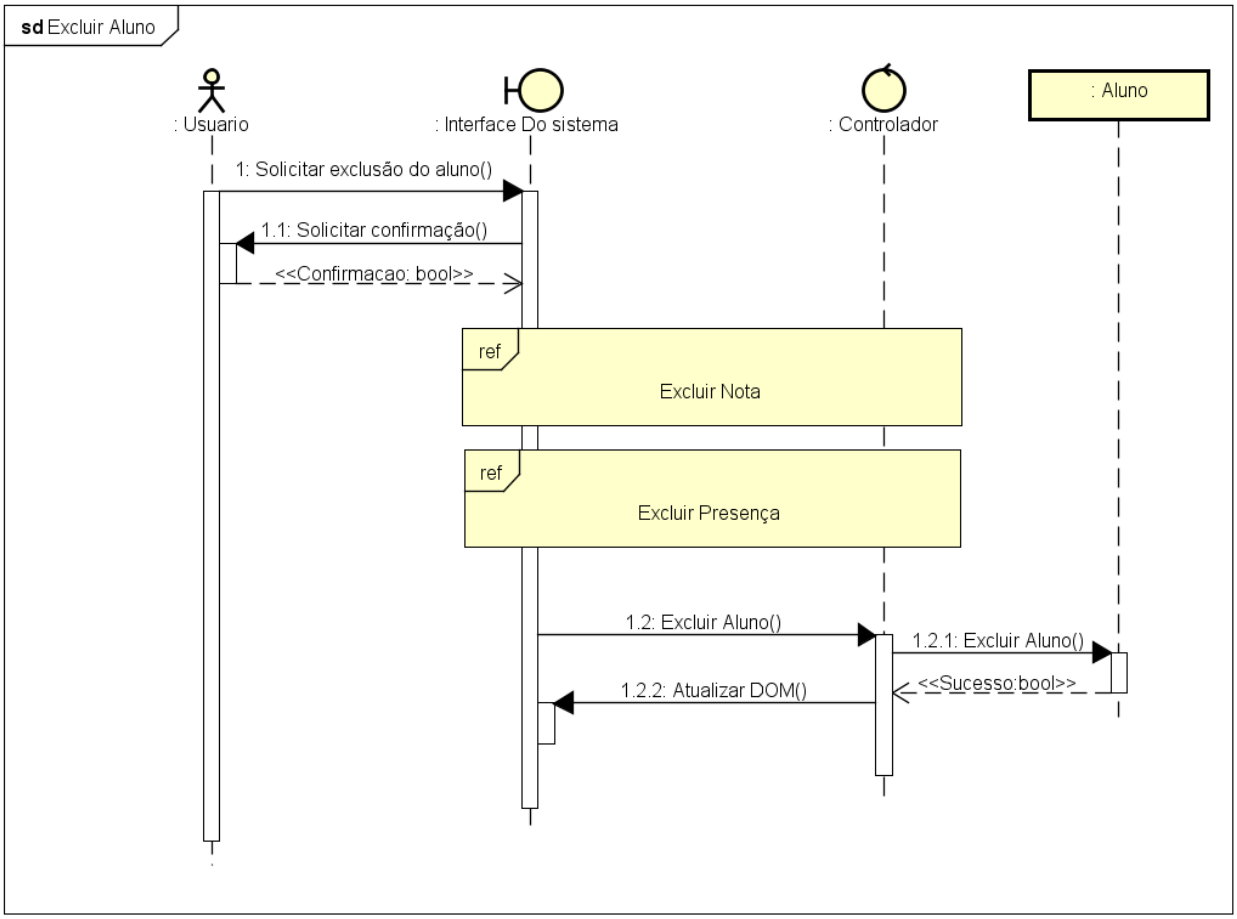
Fonte: Elaborado pelo autor

Figura 22 – Diagrama de sequência da operação editar aluno.



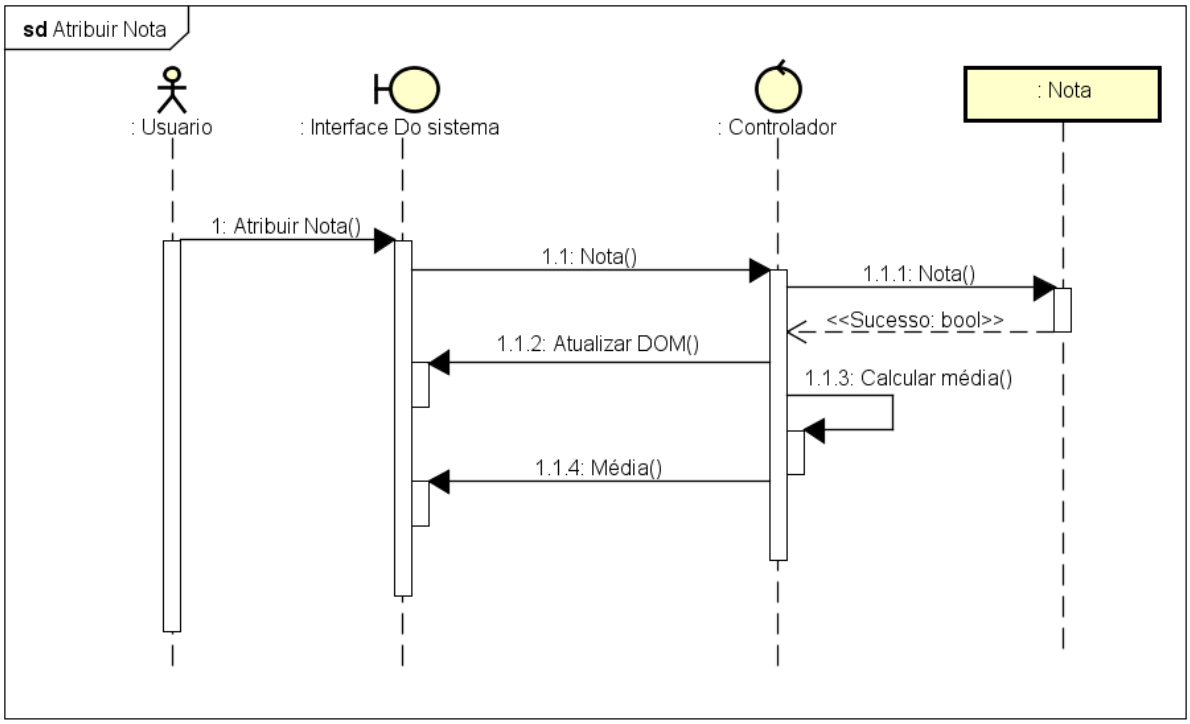
Fonte: Elaborado pelo autor

Figura 23 – Diagrama de sequência da operação excluir aluno.



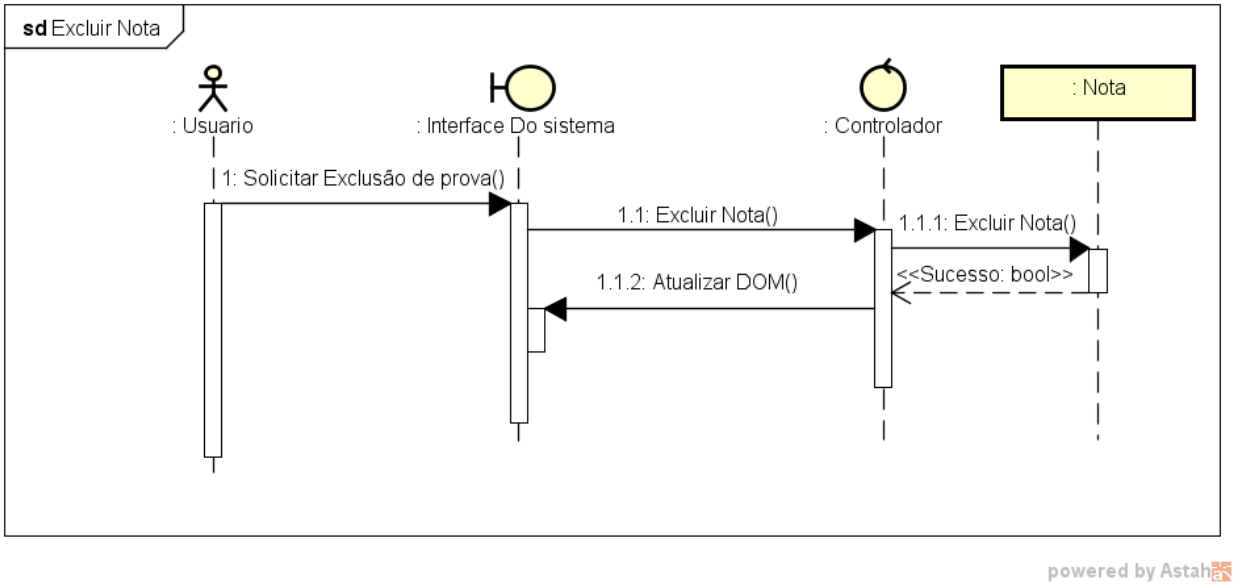
Fonte: Elaborado pelo autor

Figura 24 – Diagrama de sequência da operação atribuir nota.



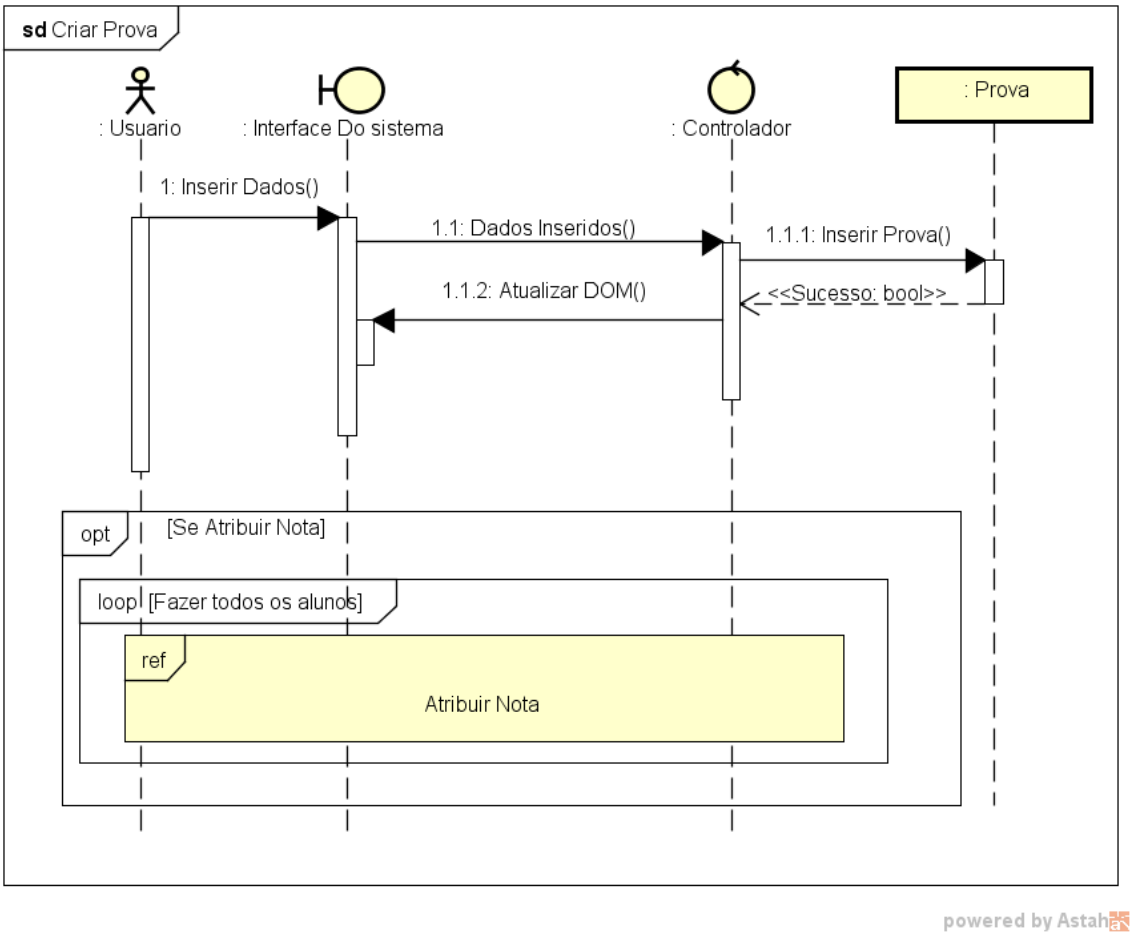
Fonte: Elaborado pelo autor

Figura 25 – Diagrama de sequência da operação excluir nota.



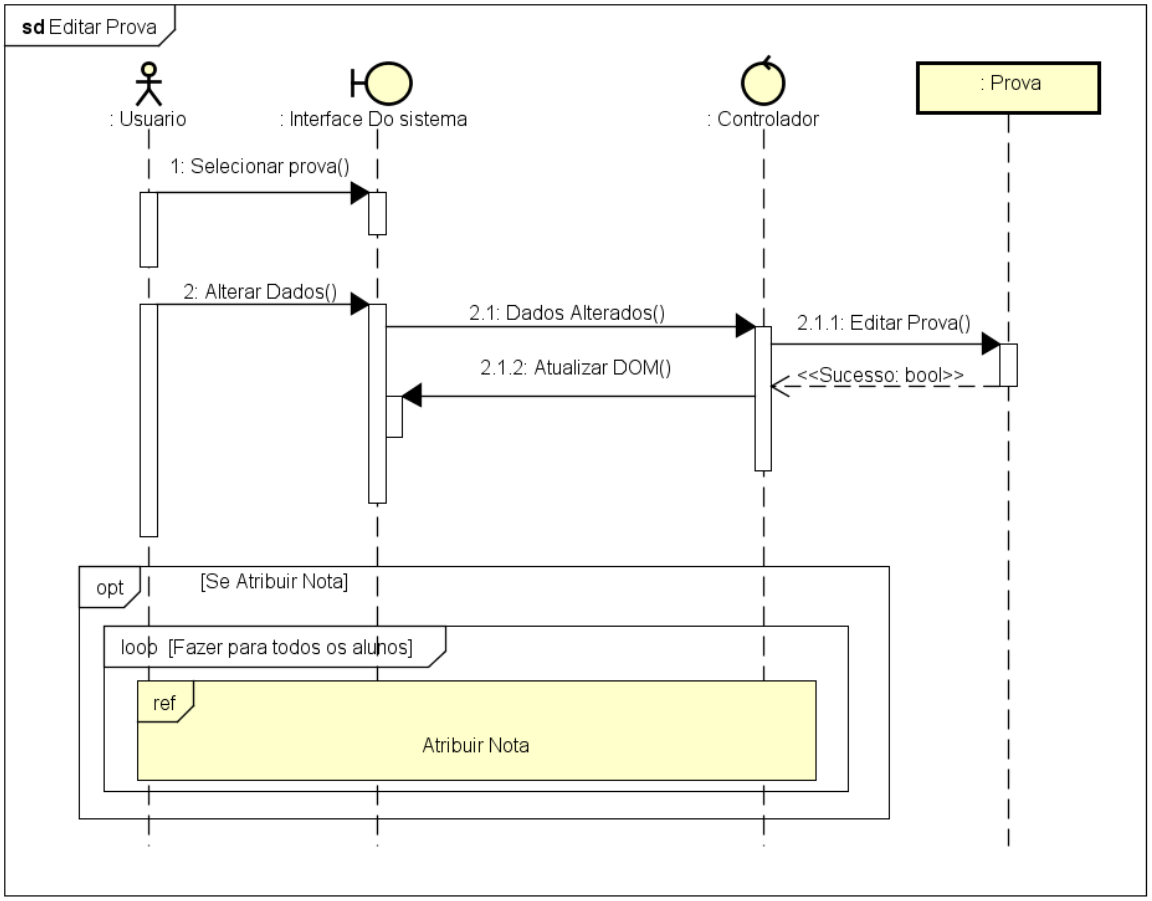
Fonte: Elaborado pelo autor

Figura 26 – Diagrama de sequência da operação criar prova.



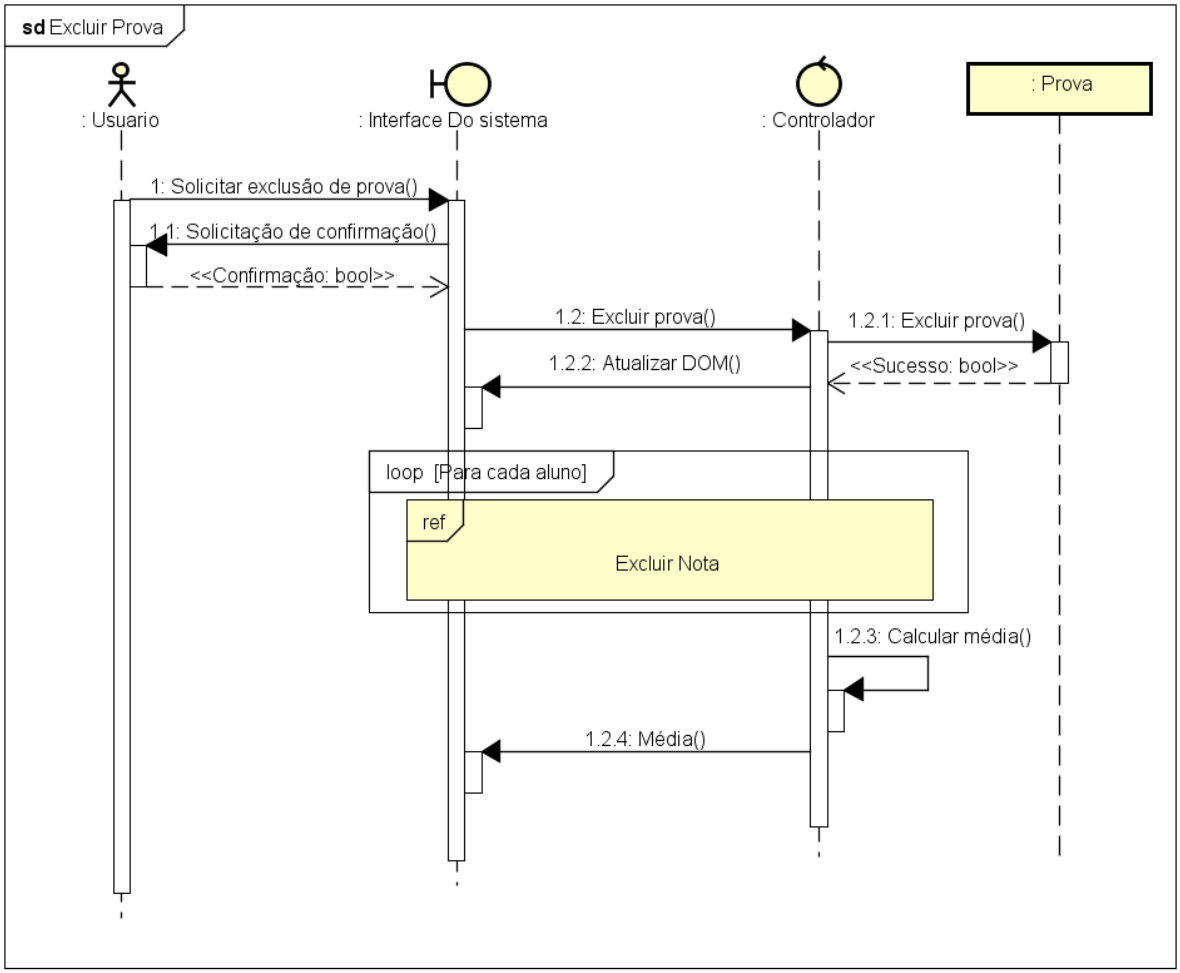
Fonte: Elaborado pelo autor

Figura 27 – Diagrama de sequência da operação editar prova.



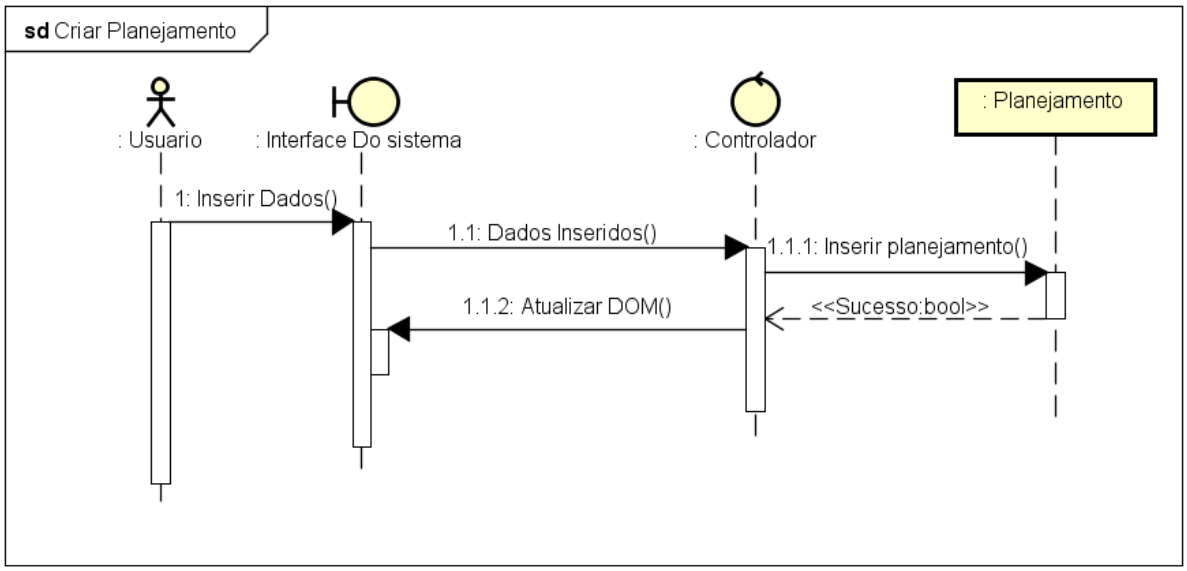
Fonte: Elaborado pelo autor

Figura 28 – Diagrama de sequência da operação excluir prova.



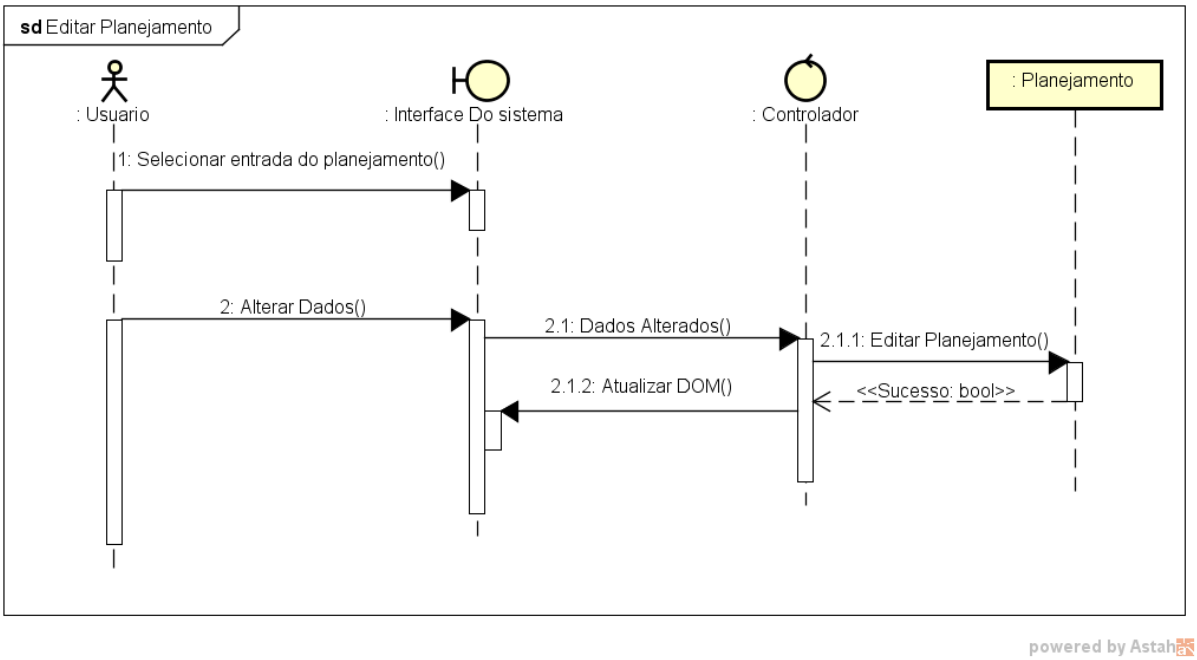
Fonte: Elaborado pelo autor

Figura 29 – Diagrama de sequência da operação criar planejamento.



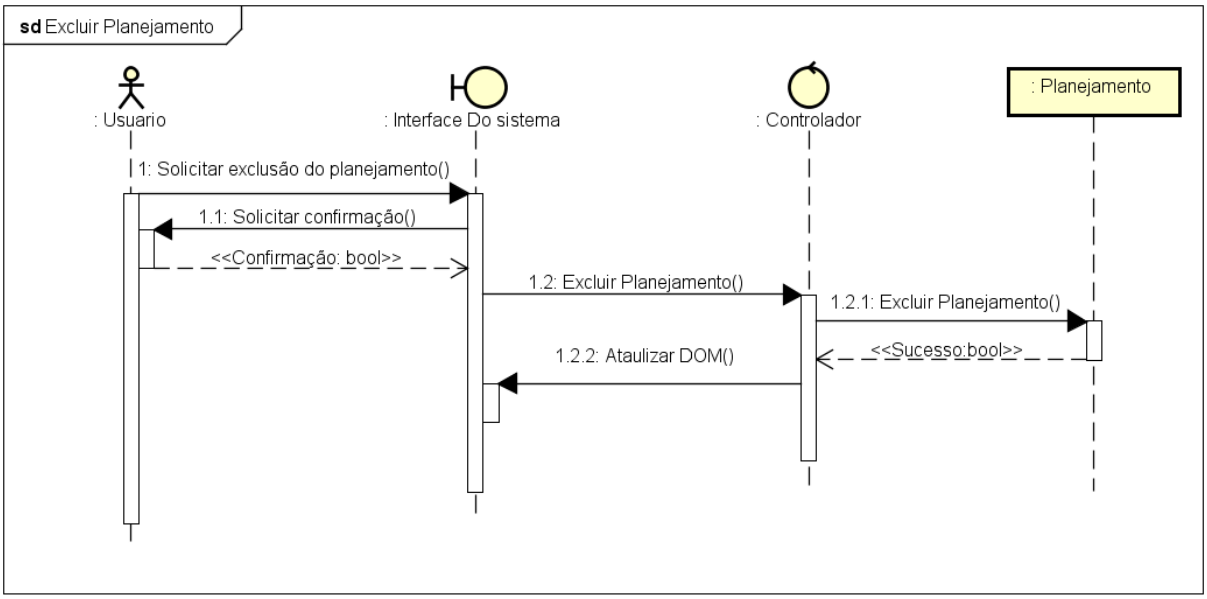
Fonte: Elaborado pelo autor

Figura 30 – Diagrama de sequência da operação editar planejamento.



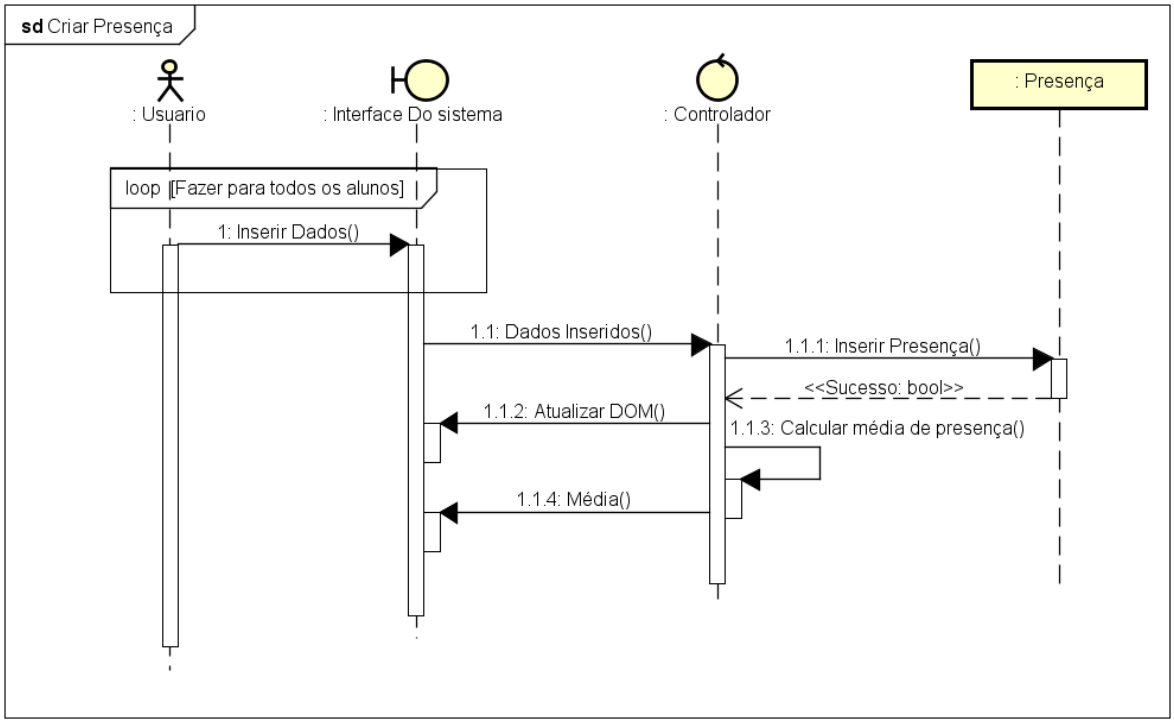
Fonte: Elaborado pelo autor

Figura 31 – Diagrama de sequência da operação excluir planejamento.



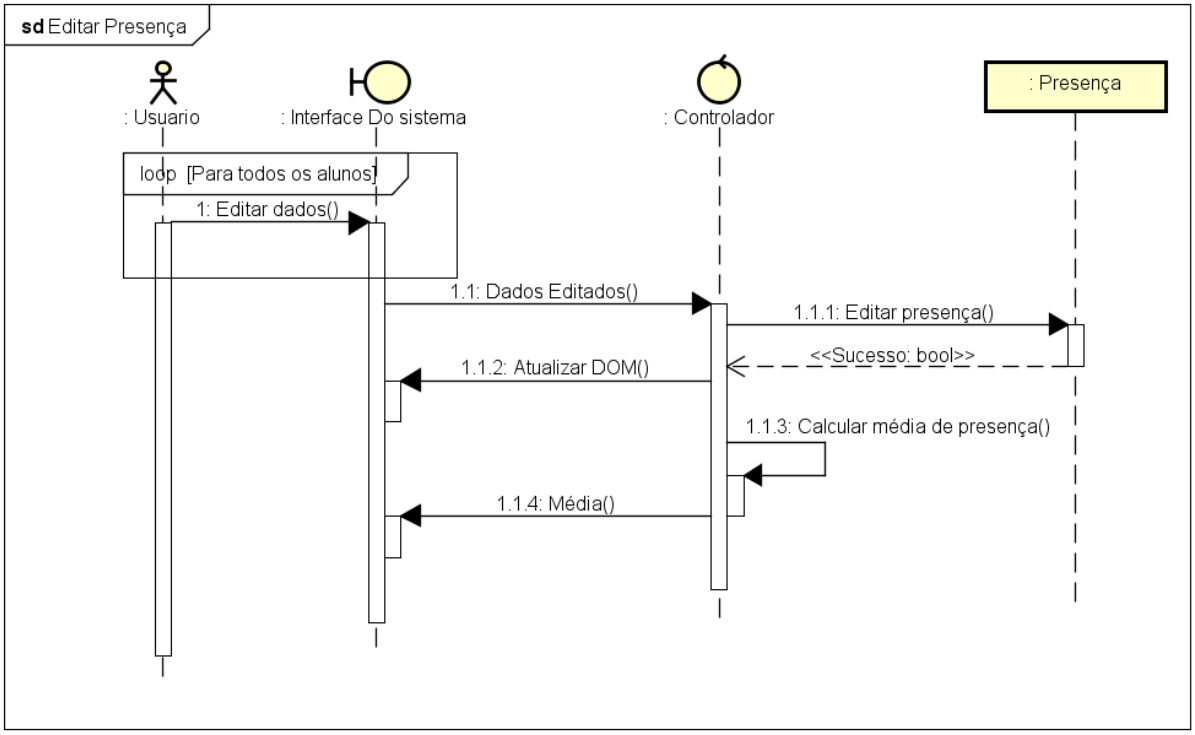
Fonte: Elaborado pelo autor

Figura 32 – Diagrama de sequência da operação criar presença.



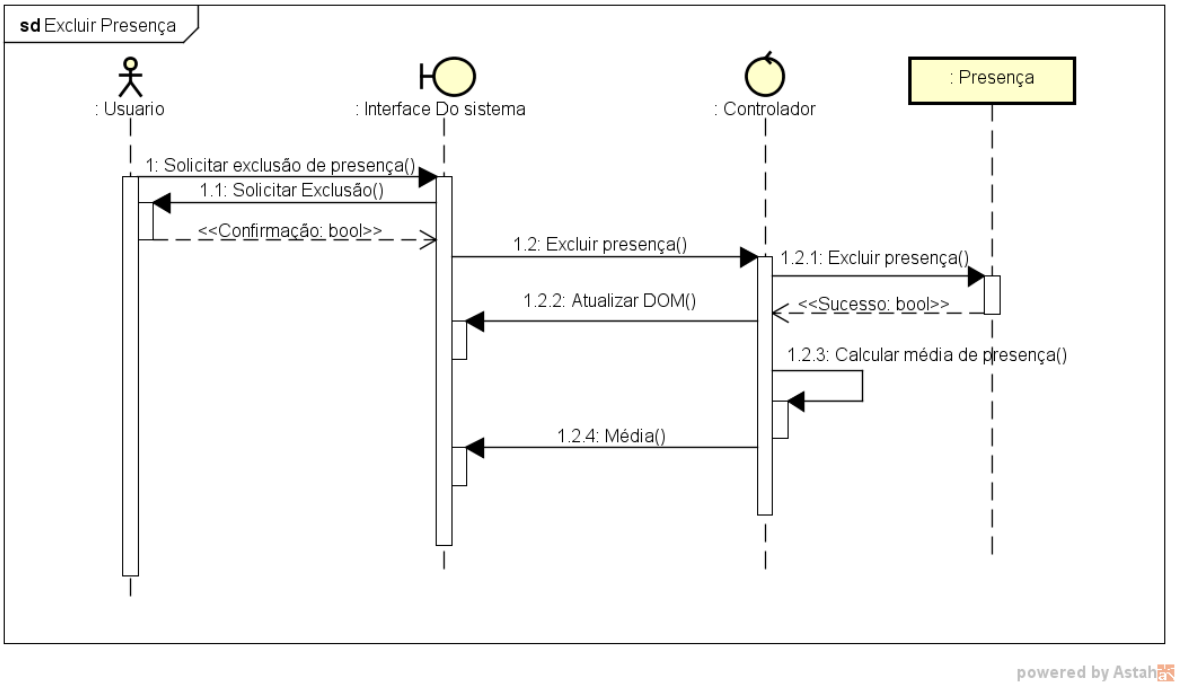
Fonte: Elaborado pelo autor

Figura 33 – Diagrama de sequência da operação editar presença.



Fonte: Elaborado pelo autor

Figura 34 – Diagrama de sequência da operação excluir presença.

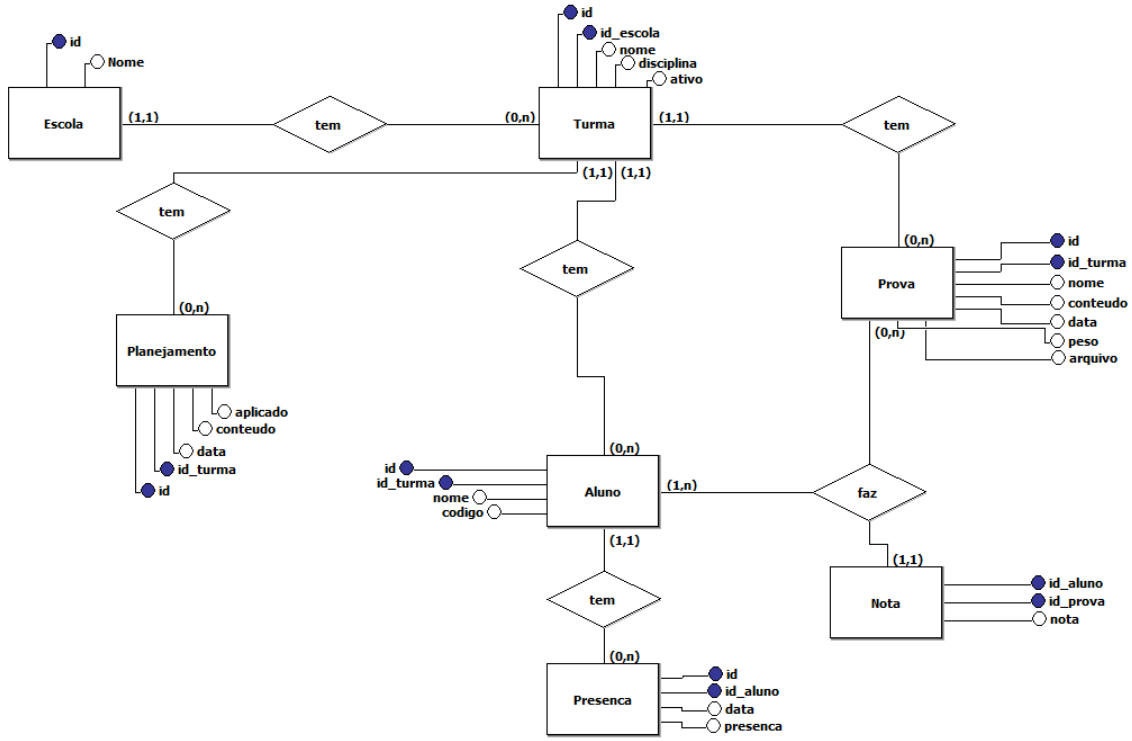


Fonte: Elaborado pelo autor

3.4.4 Modelo do banco de dados

O modelo de entidade e relacionamento é uma representação dos dados pertencentes ao sistema. Como o nome sugere esse modelo representa, “através de elevado grau de semântica” (MACHADO,2008, p.68) objetos no mundo real cuja informações são coletadas (entidade, representada por um retângulo), seus relacionamentos (representados por um losango) e seus atributos (representados por círculos)

Figura 35 – Diagrama de Entidade e Relacionamento.



Fonte: Elaborado pelo autor

A figura 35 representa o modelo de entidade e relacionamento do sistema. Nota-se que alguns atributos estão destacados, isso se deve por serem chaves primarias e estrangeiras do atributo, portanto, não podem ser repetidos em uma tabela e servirão para conectar as tabelas entre si, fornecendo uma semântica mais legível.

Porém, é importante notar que esse esquema seria válido apenas para um banco de dados relacional. O Firebase, banco de dados usado nesse projeto, é um banco de dados *NoSQL* (*Not Only SQL* – Não somente SQL). Um banco de dados

NoSQL possui características próprias, como por exemplo: Escalabilidade horizontal, esquema flexível ou ausência do mesmo, suporte nativo a replicação e consistência eventual. E, diferente do modelo relacional, preza a “alta disponibilidade e escalabilidade” (LÓSCIO, 2011), ao invés de uma estrutura interligada.

Os bancos de dados *NoSQL* possuem 4 maneiras principais de serem modelados: chave-valor, orientado a documentos, orientado a colunas e orientado a grafos.

Esse projeto usou o modelo chave-valor, por ser o mais condizente com as necessidades do projeto. O modelo chave-valor “é considerado bastante simples e permite a visualização do banco de dados como uma grande tabela *hash*” (LÓSCIO,2011) e é conveniente nesse caso, pois o Firebase, banco de dados usado no projeto trabalha com JSONs.

JSON (*Javascript Object Notation* – Notação de Objetos Javascript) é uma espécie de formatação de dados, contemplando uma chave e um valor, sendo ideal para troca de dados. Apesar de simples, o JSON “se prova útil o suficiente e simples o suficiente para ser usado em vários contextos” (Droettboom,2016, p5).

Esse modelo, como o nome sugere, é construído a partir do conceito que cada entidade ou objeto no banco possui uma chave e valor correspondente. Esse valor, pode conter outros objetos com chave e valor, que pode conter outros objetos e assim por diante. No Firebase, assim que um dado é inserido, a chave pode ser determinada pelo programa, ou ser gerada automaticamente.

As figuras a seguir detalham como o banco de dados está estruturado:

Quadro 24 – JSON geral

```
{
      "usuarios": {
            "-LCTda7F4qGRkasdijas6GZ": {
                  "nome": "Arnaldo"
            }
      },
      "LCTda7F4qGRkpaisdpasodA": {
            "userId": "-LCTda7F4qGRkasdijas6GZ",
            "escola": {},
            "turmas": {},
            "alunos": {},
            "planejamento": {},
            "provas": {}
      }
}
```

Fonte: Elaborado pelo autor

A figura 24 representa o JSON do Firebase de uma maneira geral. Os usuários ficam em uma entidade separada, para que, se for preciso recuperar apenas os usuários, não traga consigo dados desnecessários. A entidade abaixo contém as escolas, turmas, alunos, planejamento e provas. Essa entidade possui também o id do usuário como identificador, com isso, o usuário terá acesso apenas as informações dele. Todos os usuários contêm apenas uma entidade como essa.

Quadro 25 – JSON Escolas e turmas

```
{
      "usuarios": {
            "-LCTda7F4qGRkasdijas6GZ": {
                  "nome": "Arnaldo"
            }
      },

      "LCTda7F4qGRkpaisdpasodA": {
            "userId":"-LCTda7F4qGRkasdijas6GZ",
            "escola": {
                  "chaveEscola1": {
                        "nome": "ETEC"
                  }
            },
            "turmas": {
                  "chaveTurma1": {
                        "nome": "Analise e Desenvolvimento de Sistemas",
                        "disciplina": "Estrutura de Dados",
                        "idEscola": "chaveEscola1"
                  }
            },
            "alunos": {...},
            "planejamento": {...},
            "provas": {...},
            "presenca": {...}
      }
}
```

Fonte: Elaborado pelo autor

A figura 25 agora apresenta um olhar mais detalhado sobre o JSON da figura 24. A entidade escola possui uma lista das escolas que o usuário cadastrou. As escolas são identificadas por chaves únicas, essas chaves são geradas automaticamente pelo Firebase, ou seja, “chaveEscola1” e outras chaves representadas na figura 25 e nos quadros restantes nesse capitulo são apenas simbólicas. Quanto as turmas, para identificar as escolas que elas pertencem, essas entidades possuem um campo que guarda a chave da escola. Usando esse método é possível carregar as escolas e as turmas separadamente, mas mantendo um certo grau de semântica.

Quanto as entidades menores, como aluno, prova, planejamento, presença, elas seguem uma estrutura diferente como será analisado a seguir.

Quadro 26 – JSON turmas e alunos

```
{
      "usuarios": {...},
```

```
      "LCTda7F4qGRkpaisdpasodA": {
            "userId":"-LCTda7F4qGRkasdijas6GZ",
            "escola": {...},
            "turmas": {
                  "chaveTurma1": {
                        "nome": "Analise e Desnvolviemnto de Sistemas",
                        "disciplina": "Estrutura de Dados",
                        "idEscola": "chaveEscola1"
                  }
            },
            "alunos": {
                  "chaveTurma1": {
                        "chaveAluno1": {
                              "codigo": "001",
                              "nome": "Arnaldo Assis Ferreira",
                              "notas": {...},
                              "presencas": {...}
                        },
                        "chaveAluno2": {
                              "codigo": "002",
                              "nome": "Mateus Ferreira Da Silva",
                              "notas": {...},
                              "presenca": {...}
                        }
                  }
            },
            "planejamentos": {...},
            "provas": {...},
            "presenca": {...}
      }
}
```

Fonte: Elaborado pelo autor

O quadro 26 apresenta a estrutura dos alunos em relação as turmas. Dentro da entidade aluno há um campo com a chave da turma, e dentro dele está armazenada a lista de alunos daquela turma. Essa estrutura é diferente das anteriores, pois temos muito mais alunos do que escolas e turmas, então filtrar esse resultado seria trabalhoso, mas com essa estrutura, é possível selecionar os alunos de cada turma com rapidez. Mas a estrutura do aluno é a mais complexa das outras, pois ela apresenta 2 propriedades interessantes: notas e presenças. Essas propriedades se relacionam com as entidades prova e presença, como será explicado a seguir:

Quadro 27 – JSON Alunos e Provas

```
{
      "usuarios": {...},
      "LCTda7F4qGRkpaisdpasodA": {
            "userId":"-LCTda7F4qGRkasdijas6GZ",
            "escola": {...},
            "turmas": {
                  "chaveTurma1": {...}
            },
            "alunos": {
                  "chaveTurma1": {
                        "chaveAluno1": {
                              "codigo": "001",
                              "nome": "Arnaldo Assis Ferreira",
                              "notas": {
                                    "chaveProva1": {
                                          "nome": "prova1",
                                          "peso": 5,
                                          "nota": 10
                                    }
                              },
                              "presenca": {...}
                        },
                        "chaveAluno2": {...}
                  } },
            "planejamentos": {...},
            "provas": {
                  "chaveTurma1": {
                        "chaveProva1": {
                              "data": "04/05/2018",
                              "nome": "Prova 1",
                              "peso": 5,
                              "notas": {
                                    "chaveAluno1": {
                                          "codigo": "001",
                                          "nome":      "Arnaldo      Assis
Ferreira",
                                          "nota": 10,
                                    },
                                    "chaveAluno2": {...}
                              }
                        },
                  }
            },
            "presenca": {...} } }
```

Fonte: Elaborado pelo autor

O quadro 27 apresenta a estrutura dos alunos e sua relação coma estrutura de provas. Para começar, a entidade prova possui semelhanças com as outras entidades, é separada em uma chave da turma, e cada prova possui uma chave única gerada pelo Firebase. Essa estrutura guarda a data, o nome e peso da prova.

Porém, ela possui uma propriedade a mais, que é o mesmo nome da propriedade em Alunos: notas.

A figura 29 mostra como esse banco de dados seria estruturado se fosse relacional, e nele vemos que a tabela nota é resultado do relacionamento N para N entre as tabelas Aluno e Prova. Essa tabela foi gerada por causa da normalização, uma técnica no banco de dados relacional que visa aumentar a consistência e retirar a redundância de banco de dados. Porém o Firebase é um banco de dados *NoSQL*, e não ser relacional é uma das suas principais características. Isso quer dizer que a desnormalização é comum no FireBase.

Por isso temos 2 *nodes* notas, um em alunos e outro em provas. Assim quando uma entidade for carregada na tela, ela será carregada com todos os seus dados, assim, mesmo com redundância, não é preciso pegar dados em outra entidade, tornando a recuperação de dados mais rápida. Assim que um *node* nota é alterado, o outro é alterado na mesma hora e vice-versa, mantendo a consistência. O node notas guarda as informações da prova (na entidade alunos) ou as informações do aluno (na entidade prova), mas sempre guarda a nota nas duas entidades.

A entidade presença segue a mesma lógica:

Quadro 28 – JSON Alunos e Presença

```
{
      "usuarios": {...},

      "LCTda7F4qGRkpaisdpasodA": {
            "userId":"-LCTda7F4qGRkasdijas6GZ",
            "escola": {...},
            "turmas": {
                  "chaveTurma1": {...}
            },
            "alunos": {
                  "chaveTurma1": {
                        "chaveAluno1": {
                              "codigo": "001",
                              "nome": "Arnaldo Assis Ferreira",
                              "notas": {...},
                              "presenca": {
                                    "chavePresenca1": {
                                          "dia":"2018-05-01",
                                          "presenca": true
                                    }
                              },
                        "chaveAluno2": {...}
                  } },
            "planejamentos": {...},
            "provas": {...},
            "presenca": {
                  "chaveTurma1": {
                        "chavePresenca1": {
                              "data":"2018-05-01",
                              "presencas": {
                                    "chaveAluno1": {
                                          "codigo":"001",
                                          "nome":      "Arnaldo      Assis
Ferreira",
                                          "presenca": true
                                    },
                                    "chaveAluno2": {...}
                              }
                        }
                  }
            }
      }
}
```

Fonte: Elaborado pelo autor

O quadro 28 apresenta a estrutura dos alunos e a estrutura de presenças. Assim como as outras entidades, a presença é separada pela chave da Turma, e dentro dela, cada entrada de presença recebe uma chave única gerada pelo

Firebase. A entidade presença armazena a data e um node contendo as presenças. Apesar da entidade presença não ter uma relação N para N com a entidade alunos, se faz necessário esse tipo de consistência também. Então, a entidade alunos também tem um node de presença, contendo o dia e a presença do aluno representada por um valor booleano.

A entidade planejamento, no entanto, é mais simples:

Quadro 29 – JSON Planejamento

```
{
      "usuarios": {...},

      "LCTda7F4qGRkpaisdpasodA": {
            "userId":"-LCTda7F4qGRkasdijas6GZ",
            "escola": {...},
            "turmas": {
                  "chaveTurma1": {...}
            },
            "alunos": {...},
            "planejamentos": {
                  "chaveTurma1": {
                        "chavePlanejamento1": {
                              "data": "01/03/2018",
                              "conteudo": "Introdução a Filas",
                              "aplicado": true
                        },
                        "chavePlanejamento2": {
                              "data": "10/03/2018",
                              "conteudo": "Aplicação de Questionário",
                              "aplicado": false
                        }
                  }
            },
            "provas": {...},
            "presenca": {...}
      }
}
```

Fonte: Elaborado pelo autor

O quadro 29 apresenta a estrutura da entidade planejamento. A entidade, assim como as outras, é separada pela chave da turma, e dentro dela, cada entrada do planejamento é identificada por uma chave gerada pelo Firebase. Ela contem apenas a data, o conteúdo e uma propriedade booleana indicando se o conteúdo foi aplicado ou não. Como não tem relação com as outras entidades irmãos, ela é mais simples e enxuta.

3.5 Desenvolvimento

Este capitulo irá abordar as etapas de desenvolvimento do projeto, detalhando como cada uma foi construída e como elas se conectam.

A primeira etapa do desenvolvimento foi criar todas as telas, porém sem conteúdo, e coloca-las em suas respectivas rotas: /escola, /escola/turma, escola/turma/alunos e assim por diante. Depois disso as telas foram desenvolvidas gradativamente, cada uma em seus arquivos com extensão .vue.

3.5.1 Os arquivos .vue

Cada uma das telas do sistemas foram guardadas em arquivos com a extensão .vue. Esses arquivos são próprios do *framework* Vue, e são usados quando o aplicativo contem mais de uma pagina ou é muito complexo, como é o caso desse projeto. Os arquivos são chamados de componentes.

Cada componente possui o seu próprio código HTML, colocado em uma *tag* <*template*>; o seu próprio código Javascript, onde se inclui o código Vue ou outros códigos em Javascriprt como Jquery, colocado em uma *tag* <*script*>; e, se necessário, o seu próprio CSS, colocado em uma *tag* <*style*>.

Sobre a parte HTML, a *tag* <*template*> deve conter somente um elemento, e como não é prático ter um apenas um elemento por arquivo, normalmente se coloca o conteúdo da pagina dentro de uma *tag* <*div*>. Esse componente permite ter outros dentro dele, sem que isso altere a sua forma.

Quanto à parte de Javascript, dentro da *tag* <*script*> é onde contém o código Vue.

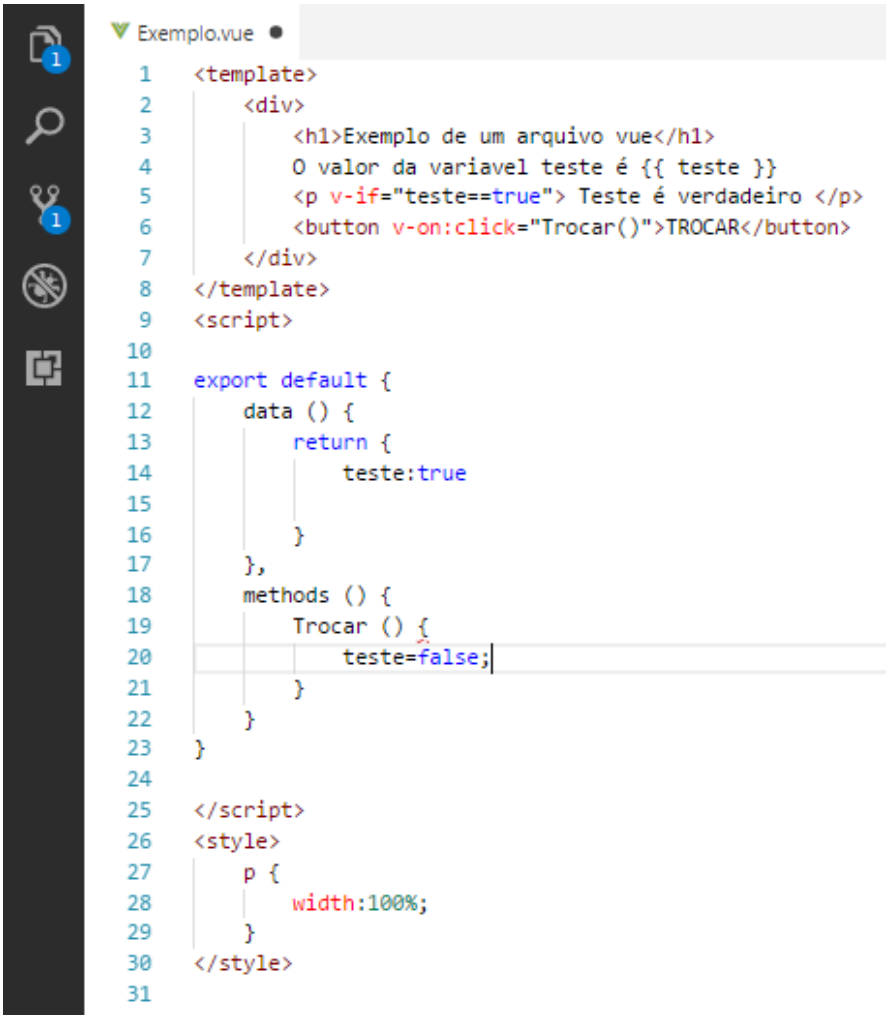
O código Vue é organizado de maneira que, suas variáveis, seus métodos, seus filtros etc, fiquem separados uns dos outros. Por exemplo, as variáveis ficam dentro de um campo chamado *data*, os métodos ficam dentro de *methods*, e assim por diante. Também podem ser separados os códigos que acontecem durante a linha de vida do Vue, como por exemplo, o código executado quando a página é criada,

quando a pagina é montada, quando a pagina é atualizada, quando a página é destruída, etc.

A interação entre o HTML e o Vue é feita da seguinte maneira: as variáveis do vue podem ser representadas no HTML quando são incluídas entre 4 chaves({{}}). O Vue também oferece outras ferramentas para que essa interação ocorra, como o *v-model,* que sincroniza uma variável com um componente do HTML; *v-on*, que associa um método do vue a um evento HTML, como o click de um botão; *v-if*, que renderiza elementos do HTML condicionalmente e *v-for* que renderiza um elemento HTML várias vezes, dentre outros.

A figura 36 descreve um exemplo simples de um componente Vue.

Figura 36 – Exemplo de um componente Vue.



Fonte: Elaborado pelo autor

A figura 39 apresenta um exemplo básico de um componente Vue. Nesse exemplo, a pagina está mostrando o valor da variável teste, que começa como *true*. O texto dentro da *tag* <*p*> aparece apenas se o valor da variável teste for *true*. E o botão aciona o método Trocar(), que troca o valor da variável teste de *true* para *false*. Depois desse evento, o texto dentro da *tag* <*p*> não irá mais aparecer, pois a condição não será cumprida.

Os arquivos em um projeto Vue são criados separadamente pois serão renderizados dinamicamente. O próximo capitulo vai explicar isso com mais detalhes.

3.5.2 O arquivo App.vue

Em sua essência, o Vue executa apenas um componente: o App.vue. Esse componente contem todo o HTML, Javascript e CSS do projeto. Mas como foi citado antes que o projeto tinha várias telas, as mesmas foram inseridas nesse arquivo.

Os componentes podem ser inseridos em outros componentes, através de *tags* especificas com seu nome. Onde essas *tags* aparecerem, no lugar será carregado o conteúdo do componente. A figura 37 mostra a parte do código do componente App.vue, onde é possível notar o uso de outros componentes.

Figura 37 – Trecho do Código App.vue.

Fonte: Elaborado pelo autor

A figura 37 traz o componente App.vue. Nota-se que há dois componentes sendo carregados nessa tela: NavBar.vue e BreadCrum.vue. Esses componentes são responsáveis pela navegação do aplicativo. E esses mesmos componentes são carregado na tela, onde as *tags* *<app-navbar>* e <appBreadCrumb> estão, respectivamente. Quanto à outra *tag*, a *<router-view>*, carrega as outras telas dependendo da rota atual.

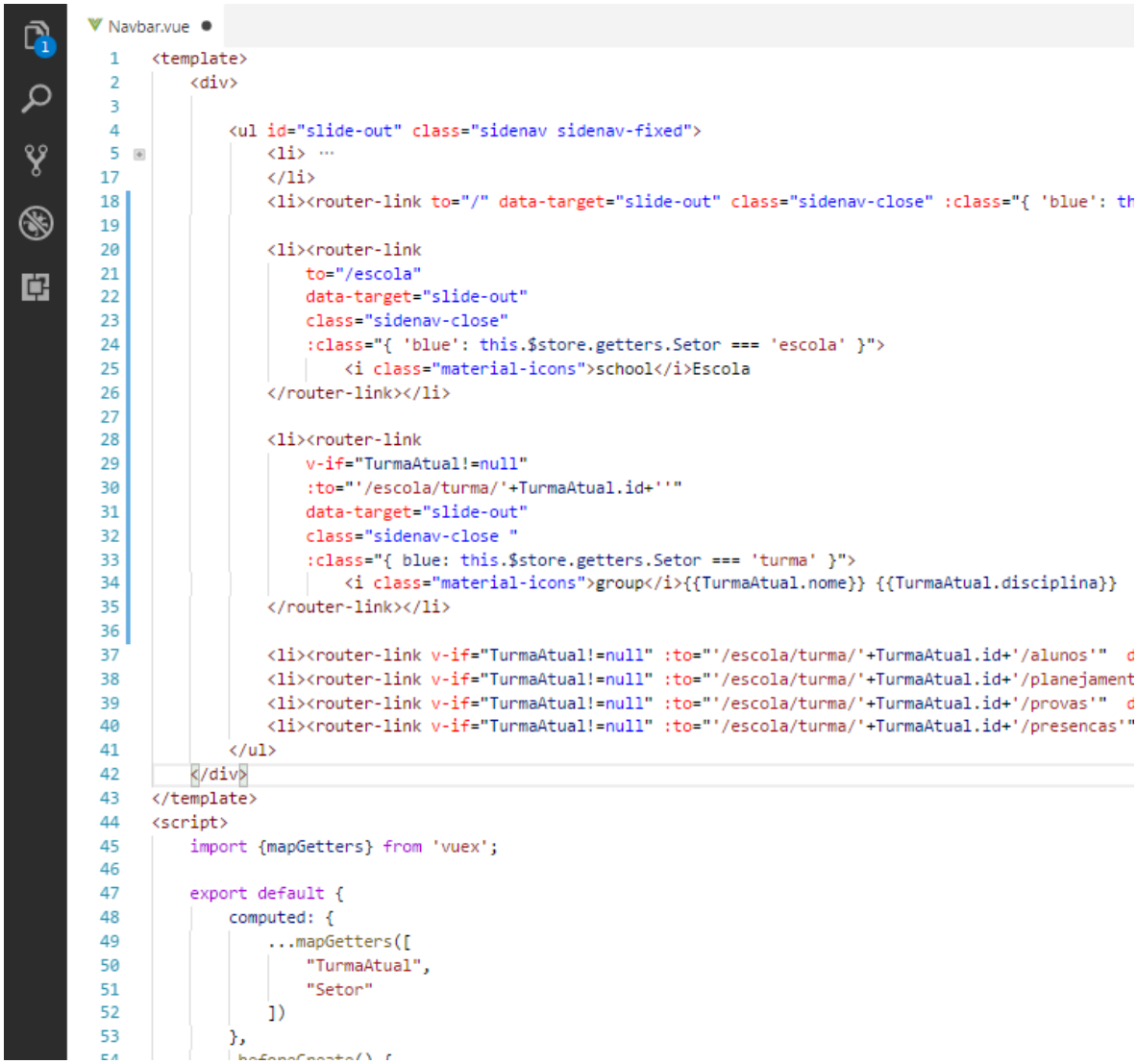
Mais detalhes sobre esses componentes serão dados à nas próximas seções.

3.5.2.1 Navbar e BreadCrumb

Para navegar pelas paginas do sistema, foram usados dois componentes do Materialise.css: *Navbar* e *Breadcrumb*. Esses dois componentes irão sempre aparecer no sistema, como mostrado na figura 36.

A *Navbar* é um componente do materialize que permite o usuário navegar pelas páginas do aplicativo através de uma barra lateral localizada do lado esquerdo da tela. Esse componente possui links para as páginas, que podem ser acessados a qualquer hora, e levam para qualquer lugar. E para o usuário se localizar, o link para a pagina atual muda de cor, indicando qual pagina está ativa. A figura 38 mostra um trecho do código do componente NavBar.vue.

Figura 38 – Trecho do código NavBar.vue.



Fonte: Elaborado pelo autor

O *Breadcrumb* é outro componente do materialize que auxilia na navegação. Porém, diferente da *Navbar*, o *Breadcrumb* mostra o caminho atual da tela, e a sua hierarquia em relação as outras telas. A navegação desse componente é limitada apenas às telas de nível superior, mas mostra ajuda o usuário a se localizar melhor que a *Navbar*. A figura 39 mostra o código do componente BreadCrumb.vue.

Figura 39 – Trecho do código BreadCrumb.vue.

Fonte: Elaborado pelo autor

Para salvar o caminho atual, ambas as telas utilizam uma técnica Vue chamada *state manegement*. Essa técnica permite salvar variáveis que podem ser acessadas e modificadas por outras telas. Ou seja, assim que um link para uma tela é ativado, essa variável é mudada para que os outros componentes possam vê-la. Essas variáveis são guardadas em no arquivo store.js, e é necessário importa-lo em todo componente que for utiliza-lo.

Nesse projeto, duas variáveis são armazenadas no store.js: setor e TurmaAtual. A variável setor guarda qual é a tela atual (escola, turma, aluno,

planejamento, prova ou presença), e é usada pelos *Navbar* e *BreadCrumb* para indicar a página atual. A TurmaAtual é um objeto que guarda dados da turma que está sendo usada, como nome, id e disciplina da turma. Esse objeto é usado para recuperar informações no banco de dados.

Figura 40 – Trecho do código store.js.

Fonte: Elaborado pelo autor

Para recuperar as informações do store.js, né necessário usar as funções chamadas *getters*, pois não é uma boa prática acessa-las diretamente. Do mesmo modo, para editar as informações, são usadas as *mutations*, uma para cada variável.

3.5.2.2 Router-view

Outro componente que apareceu na figura 37 foi o *router-view*, mas esse componente não foi importado como a Navbar.vue e o Breadcrumb.vue. Essa *tag* carrega um componente dependendo da rota atual.

Como mencionado antes, o Vue carrega apenas o componente App.vue, e os outros componentes são adicionados dinamicamente. Essa é a característica que o torna uma *Single Page Application* (SPA). Mas para ter a aparência de uma *Multiple Page Application* (MPA) é usado um recurso do Vue: o *vue-router*. As rotas são caminhos que estão na URL, ou barra de endereços. E o *vue-reouter* permite criar rotas e renderizar os componentes dinamicamente onde a *tag* <*router-view*> aparece. Essas rotas são guardadas no arquivo routes.js.

Figura 41 – Código do router.js.

router.js

```
import Escola from './telas/Escola.vue';
import DashBoardGeral from './telas/DashBoardGeral.vue';
import Turma from './telas/TurmaDashBoard';
import Alunos from './telas/ListaAluno.vue';
import NotFound from './telas/NotFound.vue';
import Planejamento from './telas/ListaPlanejamento.vue';
import Prova from './telas/ListaProva.vue';
import Presenca from './telas/ListaPresenca.vue'

export const routes = [
    { path: '', component: DashBoardGeral },
    { path: '/escola', component: Escola },
    { path: '/escola/turma/:id', component: Turma},
    { path: '/escola/turma/:id/alunos', component: Alunos },
    { path: '/escola/turma/:id/planejamento', component: Planejamento },
    { path: '/escola/turma/:id/provas', component: Prova },
    { path: '/escola/turma/:id/presencas', component: Presenca},
    { path: '/notfound', component: NotFound },

    //{ path: 'NotFound', component: NotFound },
    { path: '*', redirect: '/notfound' }
];
```

Fonte: Elaborado pelo autor

Como mostra a figura 40, as rotas são compostas por duas partes: o caminho, que é a URL para acessa a rota, relativa à pagina principal, e o componente a ser

carregado quando aquele caminho é acessado. Com esse método, assim que o usuário clica em um link, a pagina não carrega, em vez disso, ela muda o conteúdo do <*router-view*> dinamicamente, dando a impressão que são páginas diferentes.

O próximo capitulo detalha como foi o desenvolvimento dessas telas, que são geradas e disponibilizadas dinamicamente.

3.5.3 Componentes do *router-view*

Esse capitulo ira abordar o processo de desenvolvimento dos componentes que são renderizados pelo *vue-router*. Cada um desses componentes ocupa a mesma *tag* <*router-view*> do App.vue, e são gerados e apresentados segundo o *path* do router.js, ou seja, segundo a URL atual.

Cada uma dessas telas está ligada a uma ou mais entidades do banco de dados Firebase, e esse ligação é feita para os componentes poderem criar, ler, atualizar e excluir os dados dessas entidades. Esses processos recebem o nome de CRUD (*Create Read Update Delete*).

O Firebase se conecta com os componentes através de eventos. Ou seja, é possível escrever métodos para determinar o que acontece quando um dado do Firebase é criado, atualizado ou excluído. Cada componente possui um *array* de objetos, que é sincronizado com os dados do Firebase: se uma escola é adicionada no Firebase, é adicionada nesse *array*; se uma escola editada no Firebase, é atualizada no *array*; se uma escola é excluída no Firebase, é excluída no *array* e assim por diante.

As seções a seguir irão detalhar mais como foi o desenvolvimento desses componentes e com que entidades do Firebase eles se relacionam.

3.5.3.1 Componente Escolas

O primeiro desses componentes a ser criado foi o Escola.vue. Esse componente possui o CRUD completo de escolas, que só apresentam o nome. Também apresenta o CRUD de turmas que tem relação com escolas. O componente Escola.vue manipula apenas duas entidades do Firebase: escola e turmas.

Para disponibilizar esses dados na tela, foi usado o componente *Collapsible* do framework materialize. O *Collapsible* é uma lista de barras verticais que, quando clicadas, expandem o seu conteúdo embaixo da barra clicada. Nessa tela, as escolas são colocadas nas barras, e a lista de turmas de cada escola é colocada no conteúdo expansível.

Do lado do nome de cada escola e turma, foram colocados botões de ação para editar e excluir, no caso da escola também foi colocado um botão de adicionar turmas, enquanto no caso da turma, foi colocado um botão para acessar a turma.

Quando uma turma é selecionada, uma série de processos é iniciada: os dados dessa turma são colocados no store.js, para ser acessado por outros componentes; o *vue-router* coloca o *id* dessa turma na URL; a *navbar* apresenta mais cinco links: o nome da turma (direcionando para o *dashboard* da turma), alunos, planejamento, provas e presença. Por fim, o usuário é direcionado para a pagina *dashboard* da turma, explicada na próxima seção.

Se o usuário voltar a clicar em escolas, na *navbar*, o processo inverso ocorre: o store.js limpa os dados da turma, o *vue-router* elimina o id da turma na URL e os *links* da *navbar dashboard* da turma, alunos, planejamento, provas e presença somem. relacionam.

3.5.3.2 Componente DashBoard Turma

Das telas referentes à turma, o componente TurmaDashBoar.vue foi o primeiro a ser criado, porém foi um dos últimos a ser populado com informações. Esse componente apresenta as informações gerais da turma como o nome e a disciplina, mas também apresenta a quantidade de alunos, eventos no planejamento e provas recentes.

Esse componente foi um dos últimos a ser desenvolvido pois ele contém informações de outros componentes, como alunos, planejamento e provas, que precisavam ser desenvolvidos primeiro. Assim que cada componente desses era desenvolvido, o TurmaDashBoard.vue era incrementado.

Esse componente possui conexões com as entidades turmas, planejamento e provas do Firebase, mas não possui conexão com a entidade alunos, pois a

quantidade de alunos é uma informação armazenada na turma, mas manipulada pelo componente ListaAluno.vue, que será explicado na próxima seção.

3.5.3.3 Componente Lista Aluno

O componente ListaAluno.vue foi um dos mais complexos componentes do projeto. Esse componente possui o CRUD dos alunos, contendo seu código e nome, mas além disso, possui as notas dos alunos, assim como as presenças de cada dia.

A primeira implementação do componente foi o CRUD dos alunos, que manipulava apenas o código e o nome. Após isso, um sistema de filtragem por nome. As outras informações dependiam de outros componentes, que precisavam ser desenvolvidos.

Após o CRUD básico, o componente ListaProva.vue teve que ser desenvolvido, para que o sistema de notas pudesse se implementado. Após isso, o componente ListaPresenca.vue teve que ser desenvolvido para que o sistema de presença fosse implementado.

E por fim, depois de ser desenvolvido, foi implementado a alteração do número de alunos na entidade turmas do Firebase. Cada vez que um aluno era criado, a quantidade aumentava em um, e cada vez que um aluno era deletado, a quantidade diminuía em um.

Esse componente é o que possui mais ligações com entidades do Firebase: alunos, provas, presença e turma (para a alteração de quantidade).

3.5.3.4 Componente Lista Planejamento

O componente ListaPlanejamento.vue por outro lado, foi um dos mais simples a ser desenvolvido. Esse componente apresenta o CRUD das entradas do planejamento, manipulando data, conteúdo do planejamento, e se foi aplicado ou não. Ele também só possui conexão com a entidade planejamento no FireBase.

Após o desenvolvimento do CRUD, foi implantado as opções de filtragem, que era possível filtrar o planejamento por conteúdo, por mês de sua aplicação e pelo

estado de aplicação (se foi aplicado ou não). Os filtros podem combinar entre si para trazer a informação mais precisa possível.

Uma complicação do desenvolvimento foi a utilização do componente *Datepicker* do materialize, que, por razões de estática e funcionalidade, foi substituído por um *datepicker* padrão. Um *datepicker* é um componente web que permite a seleção mais precisa de datas, através de um calendário.

Após o desenvolvimento do ListaPlanejamento.vue, foi implementado no componente TurmaDashBoard.vue uma lista da entrada de planejamento que tinhas data até a próxima semana.

3.5.3.5 Componente Lista Prova

O componente ListaProva.vue possui o CRUD das provas do sistema, manipulando sua data, nome e conteúdo, mas, assim como o componente ListaAluno.vue, manipula também as notas dos alunos.

A primeira implementação foi o CRUD básico das provas, seguido da implementação do sistema de filtragem semelhando ao do ListaPlanejamento.vue: por nome e por mês de aplicação. Também apresenta o componente *datapicker* padrão ao invés do *datapicker* do materialize.

Após isso, foi implementado o sistema de notas. A primeira parte era atribuir notas no componente ListaProva.vue, e elas aparecerem no ListaAluno.vue. Para isso foi necessário resgatar a lista de alunos, e depois inserir na entidade aluno do Firebase os dados da prova e da nota em todos os alunos.

A segunda parte era fazer isso do ListaAluno.vue, ou seja, alterar as notas no ListaAluno.vue e elas aparecerem no ListaProva.vue. Para isso, foi necessário manipular a entidade prova do Firebase para incluir dados do aluno e sua nota em todas as provas.

Finalizado o desenvolvimento do componente ListaProva.vue e sua sincronização de notas com o ListaAluno.vue, foi possível a implementação de um quadro no componente TurmaDashBoard.vue, contendo as provas em um período de uma semana, apresentando o dia e o conteúdo das provas.

3.5.3.6 Componente Lista Presença

Por fim, o componente ListaPresenca.vue é o último das telas do *router-view*. Ele apresenta o CRUD das entradas de presença. Cada entrada contém uma data e uma lista de alunos com suas presenças, representadas por uma *checkbox*. Esse componente, assim como o ListaProva.vue possui uma relação com o componente ListaAluno.vue.

A primeira implementação foi o CRUD básico, manipulando apenas as datas, seguido de um sistema de filtragem, por mês, assim como os componentes ListaPlanejamento.vue e ListaProva.vue.

Após o básico foi necessário implementar o sistema de presença. A partir daí o desenvolvimento se assemelhou muito ao do ListaProva.vue, se dividindo e duas etapas.

Na primeira etapa, foi necessário manipular os dados na ListaPresenca.vue, e eles aparecem no ListaAluno.vue. Para isso, primeiro foi necessário carregar a lista de alunos, e os inserir na entidade aluno do Firebase os dados da presença, como data e se o aluno estava presente ou não.

Na segunda etapa, foi necessário fazer o caminho inverso: manipular os dados no ListaAluno.vue e eles aparecerem no ListaPresenca.vue. Para isso foi necessário manipular a entidade presença do Firebase, inserindo os dados do aluno e se ele estava presente ou não, para cada entrada da presença.

3.6 Avaliação

A avaliação consistiu em averiguar se o sistema está de acordo com o que se considera como um bom software. Para Sommerville (2007, p.9):

> “os produtos de software possuem [..] atributos associados que demonstram a qualidade de software. Esses atributos não estão relacionados com o software em sim”.

Esses atributos dependem da necessidade do software, mas Sommerville elencou quatro principais deles, descritos no quadro 30.

Quadro 30 – Atributos de um bom software

Característica do produto	Descrição
Facilidade de manutenção	O software deve ser escrito de modo que possa evoluir para atender às necessidades de mudança dos clientes. É um atributo fundamental, pois a mudança de software é uma consequência inevitável de um ambiente de negócios em constante mutação.
Confiança	O nível de confiança tem uma série de características, incluindo confiabilidade, proteção e segurança. Um software confiável não deve causar danos físicos ou econômicos no caso de falha do sistema.
Eficiência	O software não deve desperdiçar os recursos do sistema como memória e ciclos do processador. Portanto, a eficiência inclui tempo de reposta, tempo de processamento, utilização de memória, etc.
Usabilidade	O software deve ser usável, sem esforço excessivo, pelo tipo de usuário para o qual ele foi projetado. Isso significa que ele deve apresentar uma interface com o usuário e documentação adequadas.

Fonte: SOMMERVILLE (2007)

O sistema desenvolvido foi avaliado usando como base os atributos contidos no quadro 30:

- Facilidade de manutenção: O código foi comentado em suas áreas essenciais, e o desenvolvimentos das telas e outros atributos foi mantido separado. Além disso, há linhas de código que provém logs, ou registros, para informações de erros;
- Confiança: como o armazenamento dos dados é realizado em um ambiente fora do software (Google Firebase), os dados estão sempre protegidos. Há também o mínimo de contato possível com o banco, e sempre que houver contato, será limitado por validações de dados.
- Eficiência: O sistema foi desenvolvido usando o framework Vue.js que, como citado na seção 3.3.1.2, é um dos frameworks de Javascript mais rápidos, leves e eficientes. O código foi revisto varias vezes para garantir a eficiência.

- Usabilidade: como várias pessoas estão familiarizadas com o material design, a escola do framework Materilize.css foi acertada. O uso de uma interface agrave e com características familiares do Material Desing, deixa mais fácil para os usuários se guiarem no sistema.

Outra característica que demostra a qualidade do software são se ele atingiu ou cumpriu os requisitos do sistema. Os requisitos descritos no capitulo 3.2 fora todos cumpridos, tanto os funcionais como os não funcionais. O desenvolvimento dos requisitos foi descrito pelos capítulos 3.5.

O sistema web OClaRy cumpriu com os seus objetivos, entregando os requisitos e respeitando os atributos de um bom software descritos por Sommerville.

4 CONSIDERAÇÕES FINAIS

Esse trabalho teve como objetivo final desenvolver um software de diário de classe online e documentar as etapas do seu desenvolvimento. Através da metodologia kanban adaptada, o software foi desenvolvido seguindo os princípios da metodologia ágil e atingiu os resultados esperados.

O trabalho inicialmente explicou como o diário de classe funcionava, que informações eram armazenadas e geridas nele, além de detalhes sobre as suas limitações. Depois de explicar como funcionava um diário de classe, foi apresentado os softwares já existentes de gestão escolas, traçando um comparativo entre os eles.

Em seguida, foi apresentado o processo de desenvolvimento: o planejamento, apresentando o diagrama do projeto; uma breve explicação sobre a metodologia ágil kanban e como ela foi adaptada para o projeto; a descrição das ferramentas utilizadas para o desenvolvimento do projeto; os diagramas de sequência e caso de uso da engenharia de software; e finalmente o processo de codificação do projeto.

Durante a codificação do projeto, houve alguns problemas devido à falta de conhecimento prévio das ferramentas utilizadas. Foram necessárias algumas pesquisas nas documentações oficiais das ferramentas e em fóruns públicos online de programação para contornar os contratempos.

Uma característica notável no software é o foco na interface. O framework Materialize.css providenciou vários componentes web usando o padrão de design materialize. Como esse padrão de design, mais conhecido pelos usuários, o software se torna mais intuitivo para usar.

O software cumpre os requisitos definidos, automatizando tarefas comuns de um diário de classe, ao mesmo tempo que entrega essas funcionalidades em uma interface familiar ao usuário.

REFERÊNCIAS

ABOUT JAVASCRIPT.Disponível em: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript>. Acesso em: 05 abr. 2018.

AGILE DEVELOPMENT: XP E SCRUM EM UMA ABORDAGEM COMPARATIVA. . Disponível em: <https://www.devmedia.com.br/agile-development-xp-e-scrum-em-uma-abordagem-comparativa/30808>. Acesso em: 12 mai. 2018.

BOOCH, Grady; RUMBAUGH, James; , Ivar Jacobson. **UML**: guia do usuário. 1 ed. Rio de Janeiro: Elsevier, 2005. 6-7 p.

BUZAN, Tony. The ultimate book of mind maps. 1 ed. : Creative, 2005. 15 p.

COMPARAÇÃO COM OUTROS FRAMEWORKS. Disponível em: <https://br.vuejs.org/v2/guide/comparison.html>. Acesso em: 06 abr. 2018.

CONHEÇA O VUE.JS, UM FRAMEWORK JAVASCRIPT PARA CRIAÇÃO DE COMPONENTES WEB REATIVOS. Disponível em: <https://tableless.com.br/conheca-o-vue-js-um-framework-javascript-para-criacao-de-componentes-web-reativos>. Acesso em: 06 abr. 2018.

DEVELOPER SURVEY RESULTS. Disponível em: <https://insights.stackoverflow.com/survey/2017>. Acesso em: 06 abr. 2018.

DIARIO NOTA DEZ: APRESENTAÇÃO DO SISTEMA. Disponível em: <https://pt.slideshare.net/anselmosantos1/notadez-dirio-escolar-eletrnico-online>. Acesso em: 19 mai. 2018.

DIÁRIO DE CLASSE.. Disponível em: <https://www.santiagosiqueira.com.br/2013/03/diario-de-classe.html>. Acesso em: 17 mai. 2018.

DIÁRIO ESCOLAR DIGITAL DO PROFESSOR: VANTAGENS E DESVANTAGENS. . Disponível em: <http://www.proesc.com/blog/diario-escolar-digital-do-professor/>. Acesso em: 19 mai. 2018.

FILIPOVA, Olga. **Learning vue.js 2**. 1 ed. Birmingham UK: Packt Publishing Ltd, 2016.

FRAMEWORK. . Disponível em: <https://pt.wikipedia.org/wiki/Framework>. Acesso em: 06 abr. 2018.

GOOGLE FIREBASE FOR DUMMIES: O QUE É E COMO FUNCIONA A PLATAFORMA. . Disponível em: <https://blog.mastertech.tech/tecnologia/google-firebase-for-dummies-o-que-e-e-como-funciona-plataforma/>. Acesso em: 09 mai. 2018.

HAVERBEKE, Marijn. **Eloquent javascript**. 3 ed. [S.L.: s.n.], 2018.

INTRODUÇÃO AOS BANCOS DE DADOS NOSQL. . Disponível em: <https://www.devmedia.com.br/introducao-aos-bancos-de-dados-nosql/26044>. Acesso em: 09 mai. 2018.

JAVASCRIPT. . Disponível em: <https://pt.wikipedia.org/wiki/JavaScript>. Acesso em: 05 abr. 2018.

KANBAN. . Disponível em: <https://br.atlassian.com/agile/kanban>. Acesso em: 17 mai. 2018.

LEARN ABOUT MATERIAL DESING AND OUR PROJECT TEAM. . Disponível em: <http://materializecss.com/about.html>. Acesso em: 08 abr. 2018.

TEOREY, Toby J.; LINGSTON, Sam; NADEAU, Tom. **Projeto e modelagem de banco de dados**. 2 ed. Rio de Janeiro: Elsevier, 2007. 13-15 p.

MACHADO, Felipe Nery Rodrigues. **Projeto e implementação de banco de dados**. 2 ed. São Paulo: Érica, 2008. 68-70 p.

MATERIAL DESING: APRENDA TUDO SOBRE O DESING DO GOOGLE. . Disponível em: <https://marketingdeconteudo.com/material-design/>. Acesso em: 08 abr. 2018.

MELO, Ana Cristina. **Desenvolvendo aplicações com UML 2.0**: do conceitual à implementação. 2 ed. Rio de Janeiro: Basport, 2004. 33-34 p.

METASYS DIÁRIO DE CLASSE. . Disponível em: <http://www.metasys.com.br/produtos/ensino-aprendizagem/metasys-diario-classe/>. Acesso em: 19 mai. 2018.

METODOLOGIA DE DESENVOLVIMENTO DE SOFTWARE. . Disponível em: <https://www.devmedia.com.br/metodologia-de-desenvolvimento-de-software/1903>. Acesso em: 12 mai. 2018.

MORONEY, Laurence. **The definitive guide to firebase**: Build Android Apps on Google's Mobile Platform. 1 ed. Seattle Washington USA: Apress, 2017. 1 p.

MOST POPULAR AND INFLUENTIAL PROGRAMMING LANGUAGES OF 2018. . Disponível em: <https://stackify.com/popular-programming-languages-2018>. Acesso em: 06 abr. 2018.

ORIENTAÇÃO PARA PREENCHIMENTO DE DIÁRIOS, SEGUNDO NORMAS DO MEC. . Disponível em: <http://www.cdn.ueg.br/source/campus_sao_luis_de_montes_belos_243/noticias/25118/secretaria/diarios.pdf>. Acesso em: 13 mai. 2018.

PRESSMAN, Roger S.. **Engenharia de Software**. 8 ed. São Paulo: Pearson

Makron Books, 1995.

PRINCIPAIS RECURSOS. . Disponível em: <https://www.diarionotadez.com.br/recursos.php>. Acesso em: 19 mai. 2018.

SISTEMA DE GESTÃO ESCOLAR. . Disponível em: <http://www.tecsoft.com.br/produtos/sistema-gestao-escolar/>. Acesso em: 19 mai. 2018.

SOMMERVILLE, Ian. **Engenharia de Software**. 6 ed. São Paulo: Pearson Addison Wesley, 2003.

TEOREY, Toby J.; LINGSTON, Sam; NADEAU, Tom. **Projeto e modelagem de banco de dados**. 2 ed. Rio de Janeiro: Elsevier, 2007. 13-15 p.

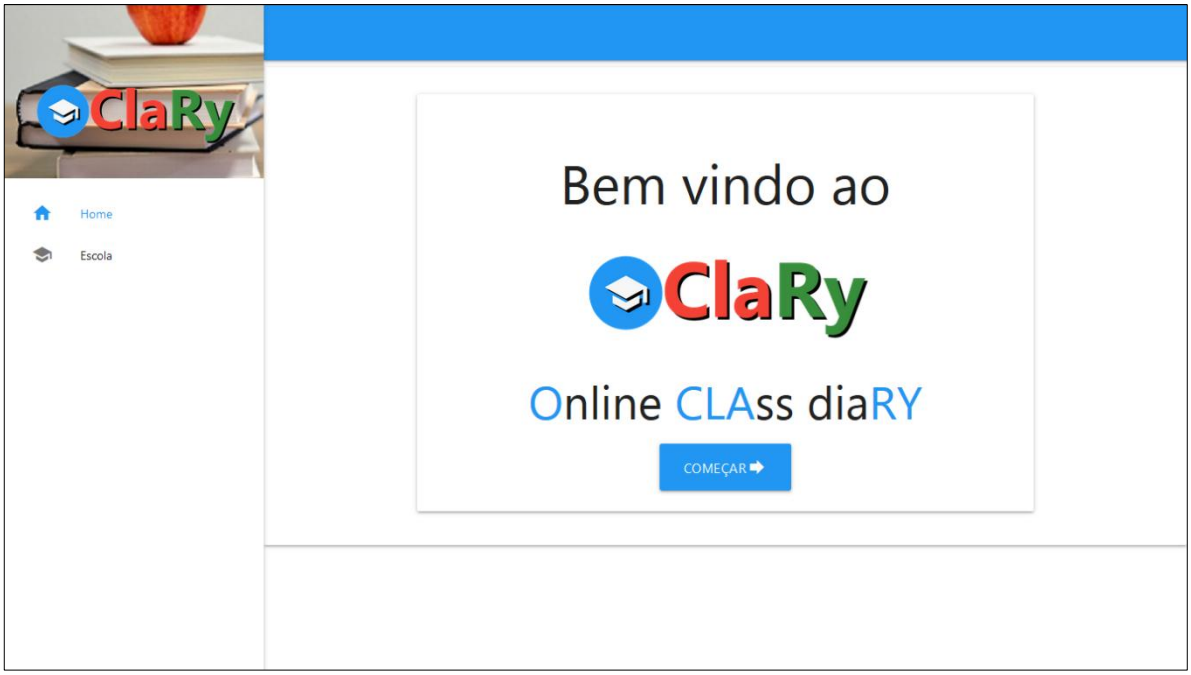
WHAT IS A MIND MAP?. . Disponível em: <http://www.mindmapping.com/mind-map.php>. Acesso em: 19 mai. 2018

APÊNDICE A – Telas do Aplicativo

A.1 Tela Inicial

Ao realizar login, o usuário se depara com a tela inicial do sistema. Essa tela contém apenas o nome do sistema e um botão que direciona o usuário para a tela de escolas. Essa tela está representada na figura 42.

Figura 42 – Tela de abertura do sistema.



Fonte: Elaborado pelo autor

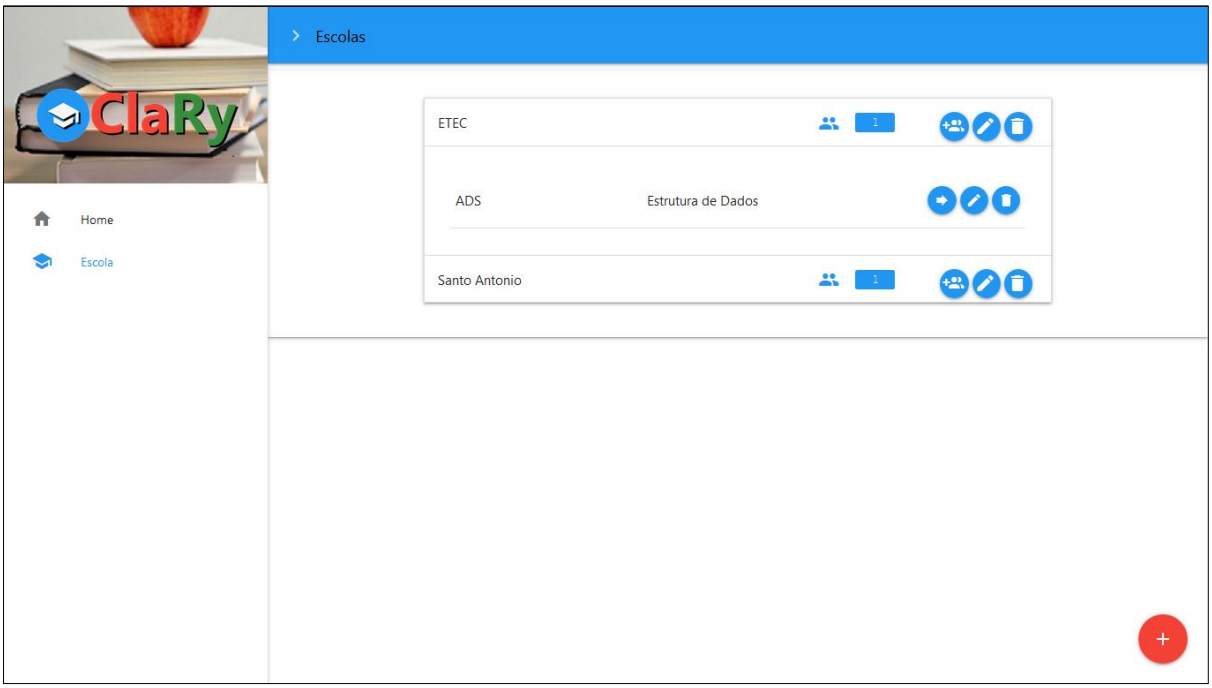
A.2 Tela Escolas

Ao clicar no botão da tela inicial, no link da barra de navegação à esquerda, ou no link escolas no *breadcrumb* na parte superior, o usuário é direcionado a tela de escolas. Essa tela contém as escolas cadastradas e as suas respectivas turmas, todos representados em um *collapsible*. Para sabe quais turmas uma escola tem, basta clicar sobre o nome da escola e a lista de turmas se expandirá.

O botão flutuante no canto inferior direito da tela tem a função de abrir o formulário de inserção de escolas. No *collapsible*, cada escola possui 3 botões: adicionar turma (que abre o formulário de inserção de turma), editar escola e excluir escola. Também há informações de quantas turmas há naquela escola, não havendo a necessidade de abri uma escola para saber quantas turmas há nela.

Quanto a lista de turmas, cada uma possui 3 botões também: de editar turma, excluir turma e acessar turma. O botão acessar turma leve o usuário à DashBoard da turma selecionada. A figura 43 apresenta a tela de escolas.

Figura 43 – Tela de escolas.



Fonte: Elaborado pelo autor

A.3 Tela Dashboard da Turma

A tela Dashboard da Turma é acessada pelo botão “acessar turma” na tela de escola. Essa tela apresenta várias informações gerais referentes à respectiva turma.

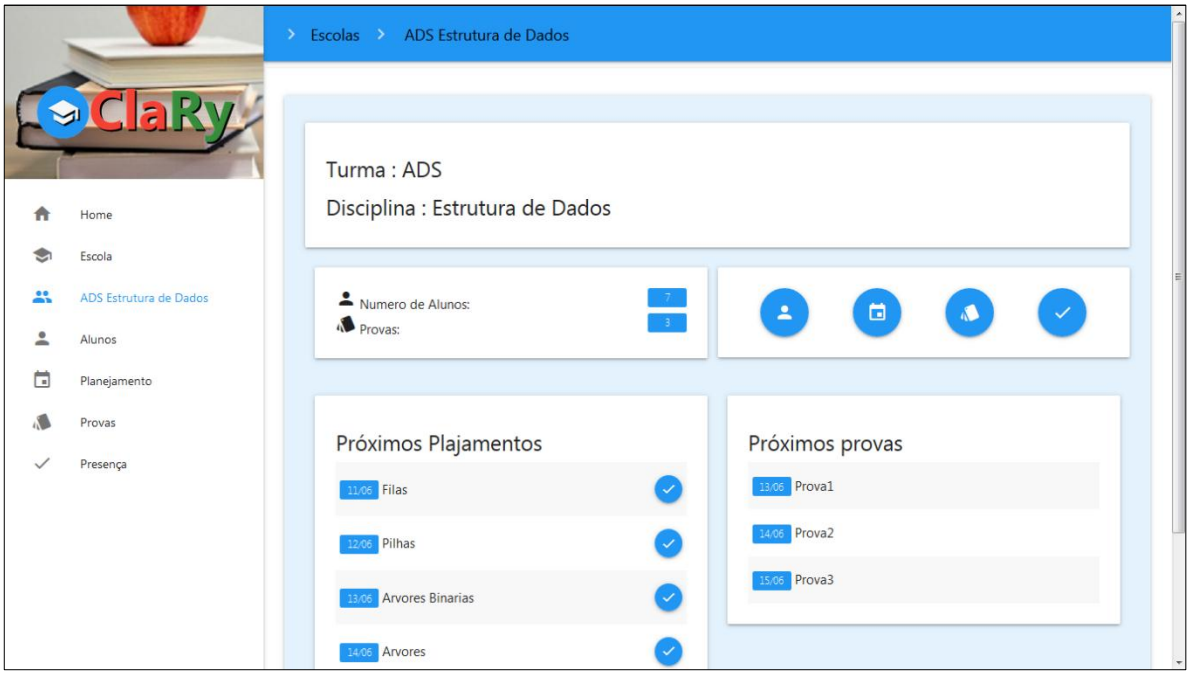
A primeira informação está no topo da tela, que é o nome e disciplina da turma. Logo abaixo estão duas seções: a primeira contém o número de alunos cadastrados naquela turma e abaixo, o número de provas, ou atividades avaliativas

que a turma possui. A segunda seção, ao lado da primeira, apresenta quatro botões, que dão acesso à outras telas da turma: alunos, planejamento, provas e presença.

E mais abaixo, há duas listas: a primeira apresenta a lista de eventos do planejamento que ainda não foram concluídos. Essa lista apresenta os planejamentos nos próximos 7 dias. Essa lista também possui um botão ao lado, para que o usuário possa indicar que aquela entrada do planejamento já foi concluída. A segunda lista é a de provas que estão próximas em um intervalo de 7 dias.

A figura 44 apresenta a tela Dashboard da turma.

Figura 44 – Tela Dashboard da Turma.



Fonte: Elaborado pelo autor

A.4 Tela Alunos

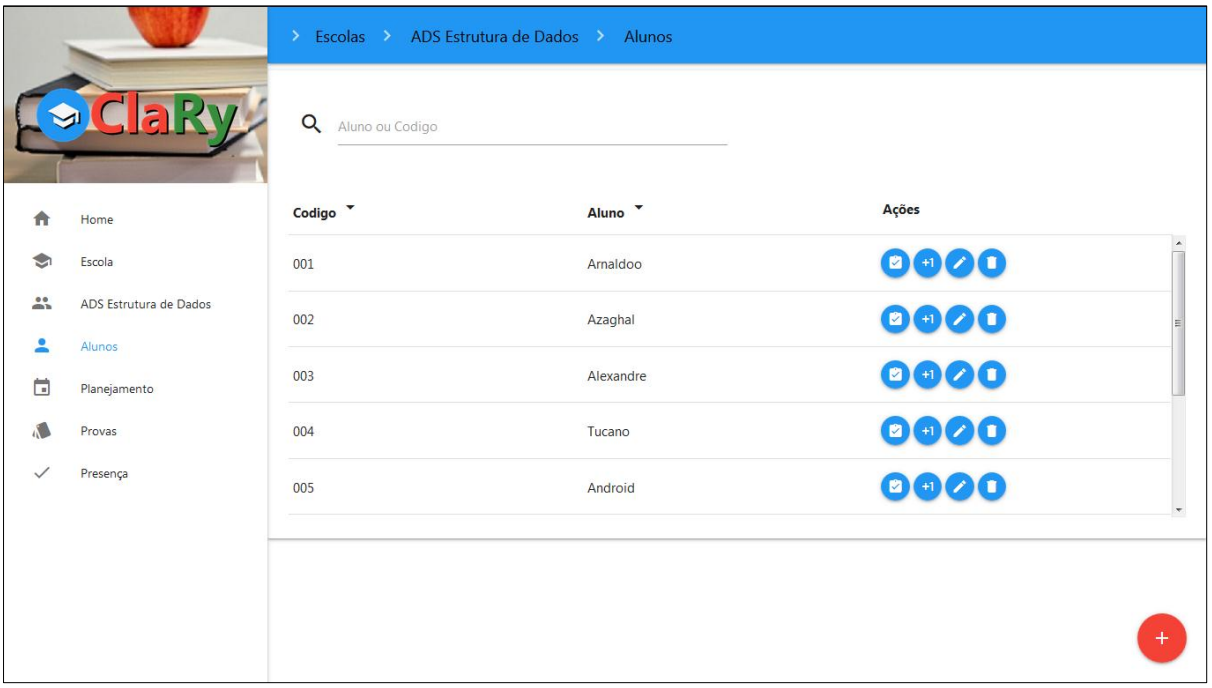
Ao clicar no link na barra de navegação do lado esquerdo ou no botão aluno do Dashboard Turma, ambos representados pelo ícone de uma pessoa, o usuário será redirecionado à tela de alunos. Essa tela disponibiliza os alunos em uma tabela,

com seu nome e seu código, além de botões que podem mostrar as presenças, as notas das provas, editar as informações dos alunos, e excluir o aluno.

No topo da tela, também está disponível um campo de texto para pesquisar os alunos por nome ou por código. No canto inferior direito, há um botão flutuante onde é possível cadastrar os alunos.

Ainda é possível organizar os alunos em ordem crescente e decrescente por nome ou por código, clicando nos títulos da tabela. A figura 45 apresenta a tela de alunos

Figura 45 – Tela de alunos.



Fonte: Elaborado pelo autor

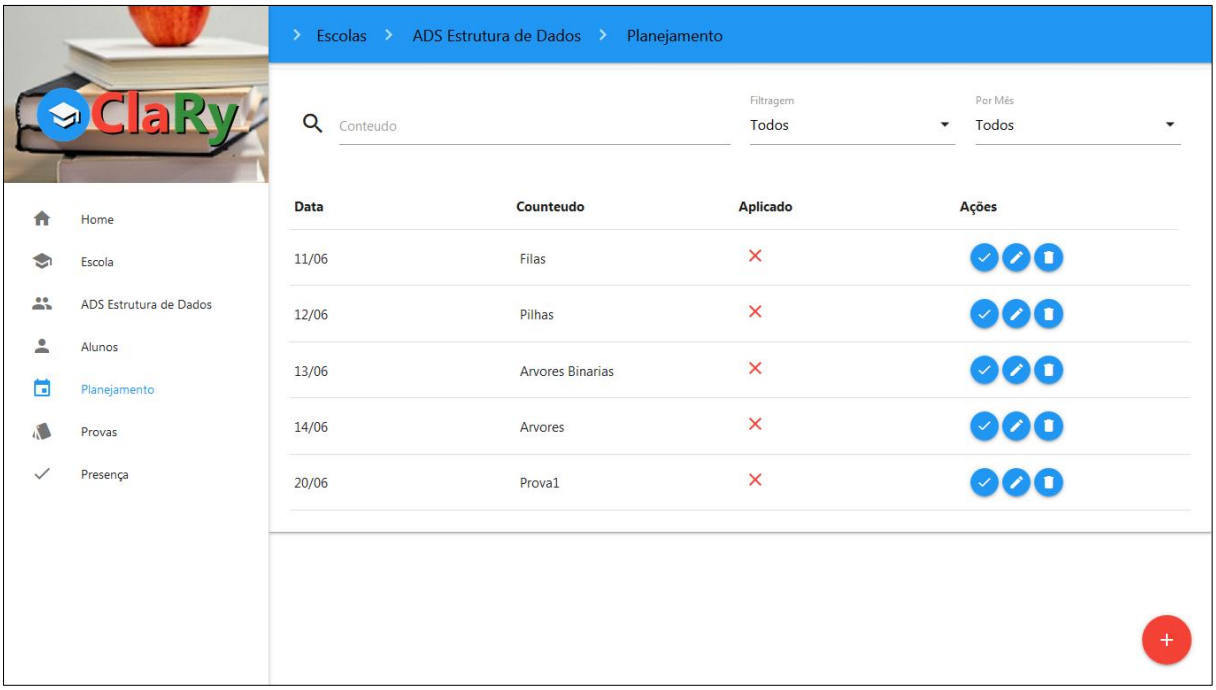
A.5 Tela Planejamento

Ao clicar no link na barra de navegação do lado esquerdo ou no botão planejamento do Dashboard Turma ambos representados por um ícone de calendário, o usuário será redirecionado à tela de planejamentos. Essa tela possui as entradas do planejamento disponibilizados em uma tabela, mostrando a data, o

conteúdo do planejamento, e botões que podem editar, excluir ou mudar o status da entrada do planejamento entre aplicado e não aplicado.

No topo da tela há opções de filtragem por conteúdo, por entradas aplicados ou não, e por mês de aplicação. Essas filtragens podem ser combinadas para trazer informações mais precisas. A figura 46 apresenta a tela do planejamento.

Figura 46 – Tela de planejamento.



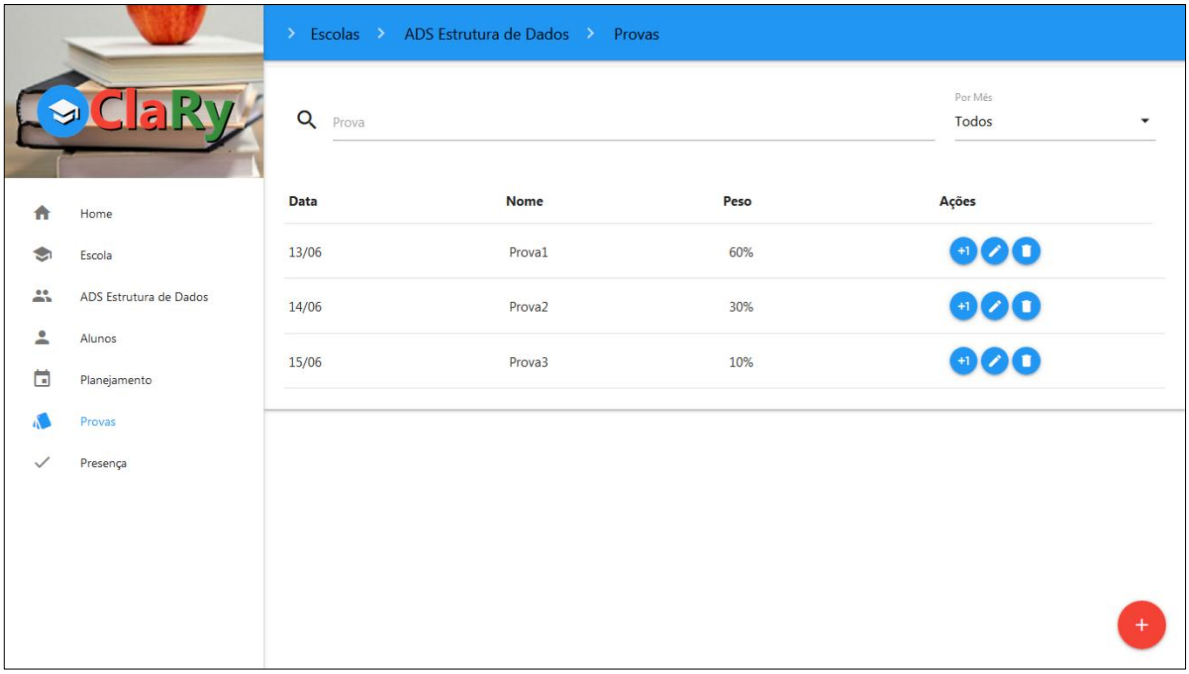
Fonte: Elaborado pelo autor

A.6 Tela Prova

Ao clicar no link na barra de navegação do lado esquerdo ou no botão planejamento do Dashboard Turma, ambos representados por um ícone de papeis empilhados, o usuário será redirecionado à tela de provas. A tela de prova dispões as atividades avaliativas da turma em uma tabela, contendo a data da aplicação, o nome da prova e o seu peso em porcentagem, assim como botões de ação para editar a prova (onde é possível ver também o conteúdo da prova), excluir e ver as notas que os alunos tiraram, como também atribuir notas aos alunos.

As opções de filtragem incluem por nome, conteúdo e mês de aplicação. A figura 47 apresenta a tela de provas.

Figura 47 – Tela de provas.

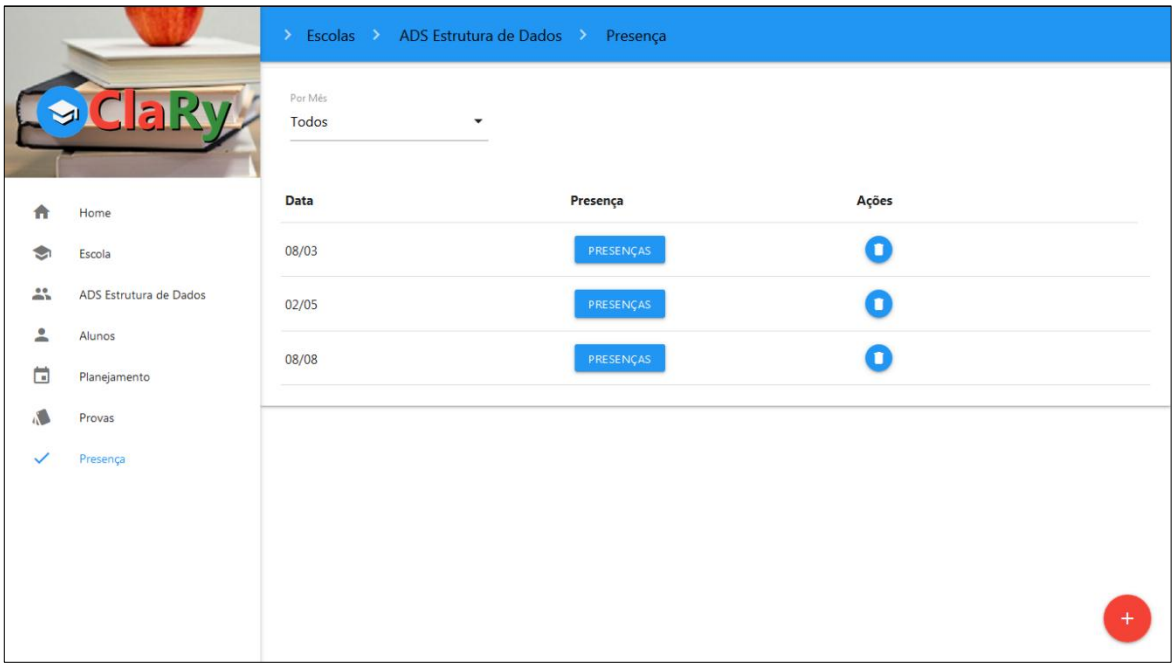


Fonte: Elaborado pelo autor

A.7 Tela Presença

Ao clicar no link na barra de navegação do lado esquerdo ou no botão planejamento do Dashboard Turma, ambos representados por um ícone de certo(*check*), o usuário será redirecionado à tela de presenças. A tela de presenças dispões as entradas de presença da turma em uma tabela, contendo a data da entrada, um botão para atribuir e visualizar as presenças dos alunos e um botão para excluir aquela entrada. As entradas de presença são disponibilizadas em uma pequena janela chamada modal, nela, há o nome dos alunos com uma *checkbox* ao lado, indicando se estavam presentes ou não naquele dia. A tela possui a opção de filtragem por mês de entrada. A figura 48 apresenta a tela de presença.

Figura 48 – Tela de presença.



Fonte: Elaborado pelo autor

APÊNDICE B – Eventos do Firebase

O banco de dados Firebase, diferente do SQL, é um banco de dados não relacional. O Firebase utiliza eventos para se comunicar com o Vue.js, esse por sua decide como o sistema vai se comportar perante esses eventos.

Os eventos são *child_added*, que é ativado quando um dado é inserido no banco de inserido; *child_removed*, que é ativado quando um dado é removido do bance; e *child_changed*, que é ativado quando um dado no banco é modificado. Cada um desse eventos possui um argumento chamado *snapshot.* Esse argumento é o dado que foi removido, inserido ou modificado, e ele é usado no Vue.js por causa do valores que possui: *snapshot.val().<nome_do_campo>* retorna o valor do campo informado; e *snapshot.key* retorna o id criado pelo Firebase.

O Clary tenta manter uma cópia do banco de dados na forma de uma lista de objetos. Toda vez que um dado é inserido no banco de dados, é inserido nessa array, a mesma coisa acontece se o dado é alterado ou deletado. A figura 49 apresena o código de como é feito a inserção. A figura 50 apresenta o código de como é feita a atualização e a figura 51 apresenta o código de como é feita a exclusão. Todos os códigos são referentes à escola.

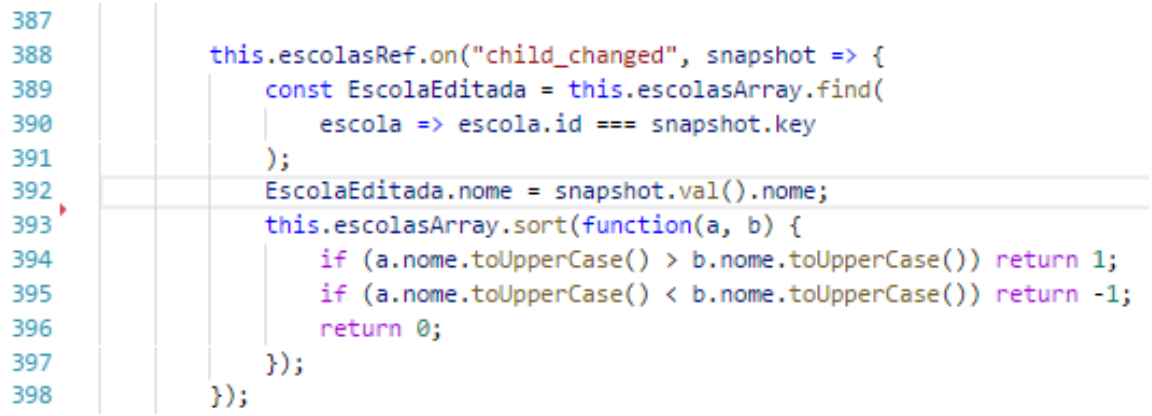
Figura 49 – Código do evento *child_added* do Firebase.

```
363
364     this.escolasRef.on("child_added", snapshot => {
365         this.escolasArray.push({ ...snapshot.val(), id: snapshot.key });
366
367         this.escolasArray.sort(function(a, b) {
368             if (a.nome.toUpperCase() > b.nome.toUpperCase()) return 1;
369             if (a.nome.toUpperCase() < b.nome.toUpperCase()) return -1;
370             return 0;
371         });
372     });
```

Fonte: Elaborado pelo autor

O código da figura 49 também é executado quando a página é carrada, trazendo todos os dados do Firebase e os colocando no escolasArray, que é o *array* utilizado como cópia do banco de dados. As linhas de código seguintes são para ordenar as escolas em ordem alfabética.

Figura 50 – Código do evento *child_changed* do Firebase.

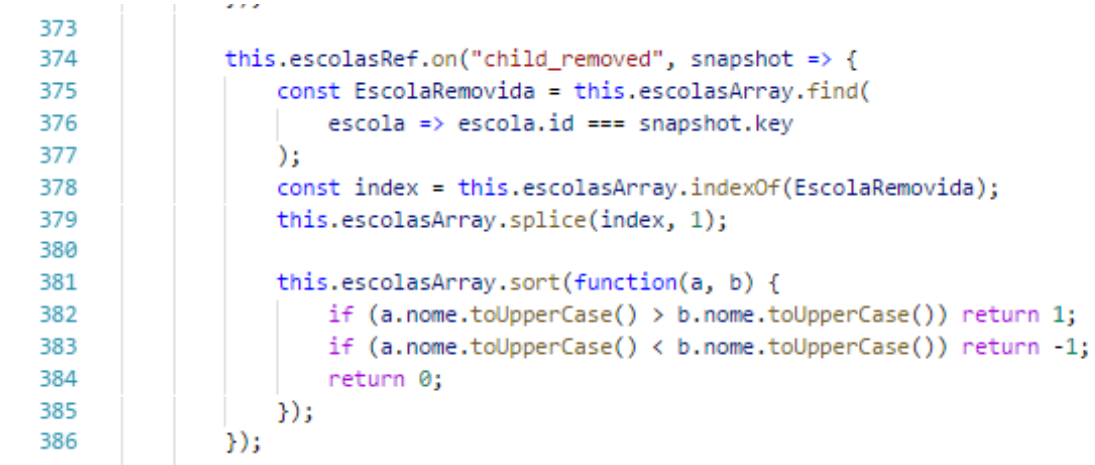


```
387
388         this.escolasRef.on("child_changed", snapshot => {
389             const EscolaEditada = this.escolasArray.find(
390                 escola => escola.id === snapshot.key
391             );
392             EscolaEditada.nome = snapshot.val().nome;
393             this.escolasArray.sort(function(a, b) {
394                 if (a.nome.toUpperCase() > b.nome.toUpperCase()) return 1;
395                 if (a.nome.toUpperCase() < b.nome.toUpperCase()) return -1;
396                 return 0;
397             });
398         });
```

Fonte: Elaborado pelo autor

O código da figura 50 consiste em procurar no *array* local a escola com o mesmo *id* gerado no Firebase. Com a escola encontrada, o nome é mudado e o *array* é ordenada mais uma vez.

Figura 51 – Código do evento *child_removed* do Firebase.



```
373
374         this.escolasRef.on("child_removed", snapshot => {
375             const EscolaRemovida = this.escolasArray.find(
376                 escola => escola.id === snapshot.key
377             );
378             const index = this.escolasArray.indexOf(EscolaRemovida);
379             this.escolasArray.splice(index, 1);
380
381             this.escolasArray.sort(function(a, b) {
382                 if (a.nome.toUpperCase() > b.nome.toUpperCase()) return 1;
383                 if (a.nome.toUpperCase() < b.nome.toUpperCase()) return -1;
384                 return 0;
385             });
386         });
```

Fonte: Elaborado pelo autor

No código da figura 51, antes de ser excluído, o dado é enviado para o Vue.js através do evento, assim, dando tempo de excluir a escola do *array* local antes de ser excluída no Firebase definitivamente.