UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ATILA ROCHA DA COSTA SILVA

SIGMA - Sistema de Gestão e Acompanhamento Móvel de Alunos Portadores de Necessidades Educacionais Especiais

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Leandro Krug Wives
Coorientador: Ms. Francisco Dutra dos Santos Jr.

Porto Alegre
2019

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
Reitor: Prof. Rui Vicente Oppermann
Vice-Reitora: Prof[a]. Jane Fraga Tutikian
Pró-Reitor de Graduação: Prof. Wladimir Pinheiro do Nascimento
Diretora do Instituto de Informática: Prof[a]. Carla Maria Dal Sasso Freitas
Coordenador do Curso de Ciência de Computação: Prof. Sérgio Luis Cechin
Bibliotecária-chefe do Instituto de Informática: Beatriz Regina Bastos Haro

RESUMO

Há alguns anos o Brasil possui uma Política Nacional de Educação Especial na perspectiva da Educação Inclusiva, onde as escolas devem oferecer infraestrutura e profissionais especializados para acompanhar e atender esses alunos - o chamado Atendimento Educacional Especializado. Dentro desse contexto, em especial nas Escolas de Ensino Fundamental do Município de Porto Alegre, salas de recursos foram implantadas para auxiliar esses profissionais. O propósito deste trabalho é a realização de um sistema de auxílio para os professores de salas de recursos em Escolas do Ensino Fundamental, focado em uma aplicação móvel para a plataforma iOS, de modo a explorar as capacidades de dispositivos móveis modernos com o intuito de agilizar o fluxo de trabalho do professor ao mesmo tempo em que oferece garantias de segurança para os dados de seus alunos. Neste documento são descritas e analisadas as técnicas utilizadas durante o processo de desenvolvimento dessa aplicação e suas interfaces. O aplicativo resultante foi utilizado por professores, os quais o avaliaram através do System Usability Scale, onde atingiu uma marca de 88,3 pontos, demonstrando que o sistema tem boa usabilidade ao executar as tarefas e requisitos propostos.

Palavras-chave: Engenharia de Software. Aplicação móvel. iOS.

SIGMA - Management System for Students with Special Educational Needs

ABSTRACT

A few years ago, Brazil established a National Policy on Special Education in the perspective of Inclusive Education, where schools must offer infrastructure and specialized professionals to accompany and attend these students - the so-called Specialized Educational Assistance. Within this context, in particular, the Elementary Schools of Porto Alegre, resource rooms were set up to assist these professionals. The purpose of this work is the realization of a helper system for the teacher of resource rooms, focused on a mobile application to explore the capabilities of mobile devices, securely streamlining the teachers' workflow, maintaining students' data private. In this document, the techniques used during the development process of this application and its interfaces are described and analyzed. The resulting application was used by teachers, who evaluated it through the System Usability Scale, where it reached a mark of 88.3 points, demonstrating that the system has good usability in performing the proposed tasks and requirements.

Keywords: Mobile Application, Software Engineering, iOS.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BaaS	Backend as a Service
MVC	Model-View-Controller
MVVM	Model-View-View Model
SQL	Structured Query Language
NoSQL	Not Only Structured Query Language
SUS	System Usability Scale
JS	JavaScript
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
SSL	Secure Socket Layer
TLS	Transport Layer Security
UX	User Experience
iOS	iPhone Operational System
PDF	Portable Document Format
AEE	Atendimento Educacional Especializado
NEE	Necessidades Educacionais Especiais
PDI	Plano de Desenvolvimento Individual

SUMÁRIO

1 INTRODUÇÃO

Em 2018, 13.7% do total de alunos de escolas públicas estado-unidenses receberam aulas de educação especial devido a algum tipo de deficiência (NCES, 2018). Esse número indica a importância de um atendimento de qualidade a esse grupo tão vulnerável.

No Brasil, esse atendimento, o Atendimento Educacional Especializado (AEE), dá-se nas Salas de Recursos, por profissionais que acompanham os alunos com algum tipo de Necessidades Educacionais Especiais (NEE). Tais educadores trabalham em conjunto com os demais professores do ensino regular para acompanhar e promover o crescimento intelectual saudável do aluno, acompanhamento que se dá através de uma série de registros e documentos que marcam o progresso do estudante nas mais diversas áreas do conhecimento.

Os registros do aluno feitos pelo professor são realizados muitas vezes através do próprio celular, com as mais diversas ferramentas e meios de captura, onde o registro fica armazenado até que, posteriormente, seja transferido ao seu computador, para que o mesmo possa eventualmente usá-lo para compor documentos capazes de explicar o progresso do aluno em questão. Esse processo é totalmente informal, e os dados do aluno, informações muitas vezes de um menor de idade, ficam expostos tanto a perdas quanto a roubos de usuários maliciosos. A informalidade do processo também traz uma falta de padronização entre o processo de cada professor, dificultando a integração das diversas fontes de informação, integração que por sua vez nos traz a possibilidade de analisar o comportamento dos alunos de modo progressivo e longitudinal. Tendo em vista tal processo, foi observado que existe a necessidade do desenvolvimento de uma maneira mais formal, centralizada e segura para ajudar professores de Salas de Recursos com esse fluxo de trabalho.

Para isso, este trabalho propõe o desenvolvimento de um sistema, focado em um aplicativo móvel para a plataforma iOS, que facilite o processo de captura, descrição, armazenamento de registros, e também possibilite a geração de documentos mais complexos, como pareceres e documentos de adequação curricular, pelo professor, de forma totalmente segura e controlada. O uso da plataforma móvel para a realização desta solução nos permite uma imensa vantagem nos quesitos de segurança e rapidez, já que o processo pode tomar parte inteiramente no dispositivo do professor.

O trabalho realizado está descrito em 7 capítulos. No próximo, serão exploradas soluções relacionadas e como elas ajudam a identificar pontos de melhora. Nos 2

seguintes trataremos do projeto e implementação do sistema como um todo, desde a especificação dos requisitos até as motivações por trás das tecnologias escolhidas. No quinto capítulo serão explicadas as interfaces de usuário desenvolvidas para o sistema. O sexto capítulo será composto pela avaliação dos resultados dos testes de usabilidade feitos com os usuários do sistema. No sétimo e último capítulo será discutido se o objetivo do projeto foi atingido, as dificuldades encontradas e as possibilidades de continuação do trabalho desenvolvido.

2 TRABALHOS RELACIONADOS

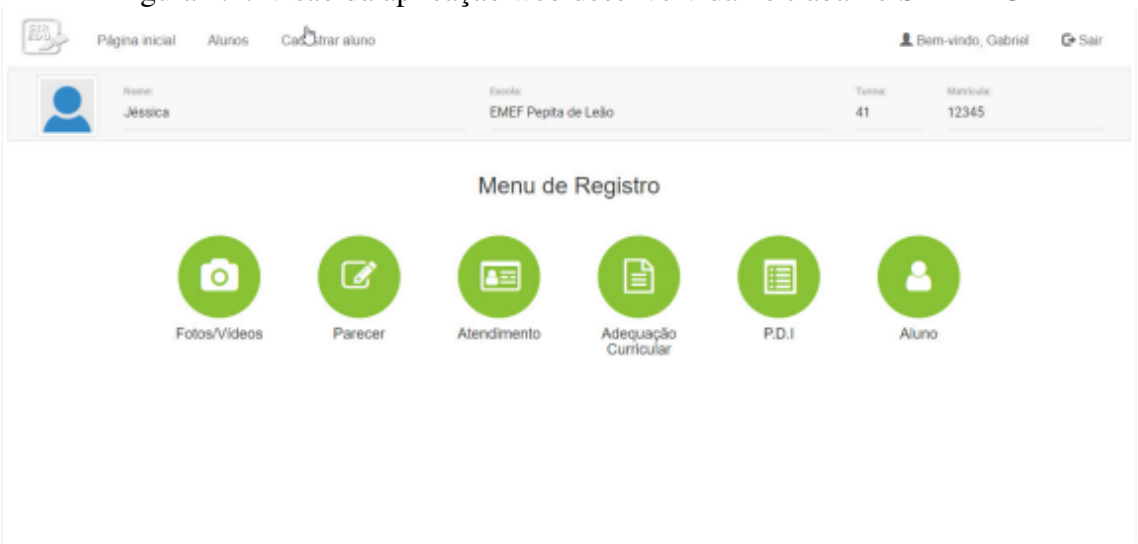
Este capítulo apresenta uma visão geral de alguns trabalhos relacionados, em especial aqueles que foram desenvolvidos anteriormente no Instituto de Informática da UFRGS para lidar com esse mesmo problema. Pode-se perceber que nenhum deles foi desenvolvido para plataforma iOS, sendo esse um grande diferencial deste trabalho.

2.1 SIR-EDU

SIR-EDU, o Sistema Integrado de Recursos Educacionais para a Gestão do Acompanhamento de Alunos com Necessidades Especiais (FERREIRA, 2017) propõe um sistema para gerenciar alunos de sala de recursos, o sistema é composto por uma aplicação móvel e outra web. A aplicação web auxilia no cadastro de alunos, eventos e pareceres. A aplicação móvel dá acesso a funcionalidades de visualização de eventos e pareceres e criação de eventos.

O resultado do sistema foi um protótipo funcional, o qual pode ser verificado na Figura 2.1, que comprova a utilidade das funcionalidades desenvolvidas para a resolução do problema. Porém o sistema não conseguiu garantir requisitos necessários de segurança dos dados e ficaram faltando funcionalidades como geração de documentos PDF e outras formas de documentos.

Figura 2.1: Visão da aplicação web desenvolvida no trabalho SIR-EDU



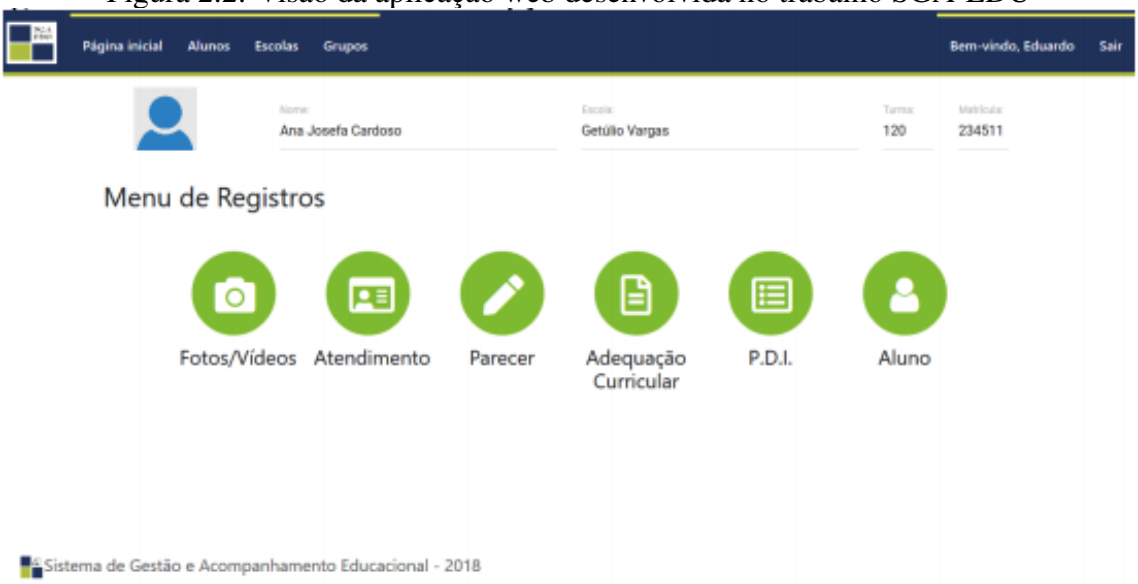
Fonte: (FERREIRA, 2017)

2.2 SGA-EDU

SGA-EDU, o Sistema de Gestão e Acompanhamento (LUCAS, 2018) é um protótipo de aplicação web que estende o trabalho desenvolvido no sistema descrito anteriormente, SIR-EDU, implementando cadastros de diferentes de tipos de documentos como PDI e adequações curriculares (JR., 2018).

Essa versão do SIR-EDU expande as funcionalidades da aplicação web desenvolvida previamente, principalmente na utilidade que fornece ao professor usuário (Figura 2.2), porém ainda não implementa uma versão móvel e os requisitos de segurança necessários para fazer a aplicação passar da fase de prototipação.

Figura 2.2: Visão da aplicação web desenvolvida no trabalho SGA-EDU



Fonte: (LUCAS, 2018)

2.3 Modelagem do processo de atendimento em salas de recursos para alunos com necessidades educacionais especiais

A modelagem do processo de atendimento em salas de recursos para alunos com necessidades educacionais especiais (MUNDEL, 2019) tem uma forte relação com a proposta de sistema desenvolvida neste trabalho, porém, diferentemente dos trabalhos já citados, ele aborda a modelagem do processo de atendimento para salas de recursos do Município de Porto Alegre, de certa forma definindo um padrão, o que é um movimento necessário para viabilizar a aplicação da solução desenvolvida neste trabalho num plano

nacional, mas foge do contexto atual de desenvolvimento de um produto de software.

2.4 Resumo do Capítulo

Pode-se perceber que os trabalhos anteriores comprovam o valor do projeto, porém não chegaram a atingir aplicações utilizáveis por professores fora do contexto de testes, devido tanto à falta de segurança das informações armazenadas na aplicação ou pela ausência de um meio de exportação das mesmas para serem usadas em outros contextos.

Essa falta de segurança dos sistemas mencionados se dá pela falta de políticas de proteção dos dados, além da autenticação por e-mail e senha, já que, no uso desses sistemas, um usuário autenticado detém privilégios de leitura sobre todo o banco de dados, deixando o sistema sujeito a ataque por um cliente malicioso. No Capitulo 4 será discorrido como foi mantida a segurança das informações no aplicativo desenvolvido.

Finalmente, na Tabela 2.1 pode-se verificar os requisitos implementados pelos trabalhos anteriores e sugeridos neste, nela observamos os objetivos do trabalho de propor uma experiência móvel segura, sem sacrificar na utilidade no quesito de geração de documentos.

Tabela 2.1: Tabela comparativa entre as implementações previas e a sugerida

Sistema	Web ou Móvel	Banco de dados	Seguro	Parecer e Eventos	Adequação Curricular	Exportar Documentos como PDF	Compartilhamento de Eventos
SIR-EDU	Web e Móvel (funcionalidade limitada)	No-SQL (baseado em documentos)	Não	Sim	Não	Não	Não
SGA-EDU	Web	No-SQL (baseado em documentos)	Não	Sim	Sim	Não	Não
SIGMA	Web (para visualização) e Móvel (independente)	No-SQL (baseado em documentos)	Sim	Sim	Sim	Sim	Sim

3 PROJETO

Este capítulo descreve as metodologias, recursos e artefatos utilizados tanto no projeto do sistema quanto no gerenciamento do seu desenvolvimento e a documentação gerada no mesmo.

3.1 Metodologia Ágil

Metodologia ágil é um termo "guarda-chuva" para diferentes metodologias de desenvolvimento de software que seguem os princípios e valores ágeis definidos no manifesto ágil (BECK MIKE BEEDLE; THOMAS, 2001b), esses princípios remetem a priorização da satisfação do usuário, flexibilidade a mudança e simplicidade.

Metodologias ágeis divergem de metodologias clássicas ao sacrificar o planejamento extensivo e documentação completa por adaptatividade, focando em iterações e incrementos e colaboração com o cliente (BECK MIKE BEEDLE; THOMAS, 2001a).

3.1.1 *Scrum*

Scrum é um *framework* de administração de processos incrementais. Ele é aplicável nas mais diferentes áreas, mas por sua natureza incremental e leve, é muito valioso para o desenvolvimento de produtos de software que envolvem validação constante (SUTHERLAND, 2017).

O Scrum, segundo o guia do seu criador Jeff Sutherland (SUTHERLAND, 2017), prevê um formato de time, eventos e artefatos. O primeiro sendo dividido em:

- **Product Owner** (dono do produto): uma pessoa designada para ser responsável por maximizar o valor do produto final elaborado pelo time de desenvolvimento a partir da administração e priorização das histórias no *backlog* do produto.
- **Development Team** (time de desenvolvimento): o time de desenvolvimento é um conjunto de profissionais, numa organização horizontal, que trabalha com o intuito de desenvolver e entregar incrementos ao produto no fim de cada *Sprint*. O time é o único responsável pelo incremento de produto e pela administração de seu próprio trabalho.
- **Scrum master** (mestre de *Scrum*): o mestre de *Scrum* é responsável por garantir

que os valores e práticas do Scrum sejam aderidos tanto pelo time de desenvolvimento como pelo dono do produto. Ele atua como líder/ajudante do time de *Scrum*, auxiliando na implementação e evolução do processo ágil, promovendo o aprendizado após cada interação.

Os eventos são prescritos no *Scrum* para minimizar a necessidade de reuniões e todos eles têm duração limitada e regular, para que haja o menor impacto possível na produtividade do time. Na figura 3.1 vemos a ordem de execução dos eventos. Eles são classificados em:

- **Sprints**: são as unidades de tempo de desenvolvimento previstas no *Scrum*, têm seu tempo fixado durante o planejamento da *Sprint*, sendo estipuladas normalmente com duração de 1-4 semanas. São realizadas sequencialmente e intercaladas com outros processos tanto de planejamento e revisão, tais como o de *Sprint Review* e *Sprint Retrospective* quanto de processos diários com o *Daily Scrum*. Uma *Sprint* não pode ter seu conteúdo alterado a ponto de mudar o objetivo da interação. A falha da *Sprint* também está prevista no *Scrum*. Seu tempo curto é necessário para que se, durante sua execução, haja mudanças nos requisitos do produto, não se perca muito tempo de desenvolvimento.
- **Sprint Planning** (planejamento da *Sprint*): o momento de planejamento de uma *Sprint* que precede sua execução. É um período de no máximo 8 horas, que é designado a fim de escolher um objetivo para a *Sprint*, definir os elementos do *backlog* que serão abordados e as tarefas necessárias para atingir cada um desses objetivos.
- **Daily Scrum** (*Scrum* diário): o *Scrum* diário é uma prática diária em período de Sprint, normalmente realizada no começo do dia em um horário fixo, com o intuito de reduzir complexidade - evento que tem duração curta e funciona para planejar o próximo dia de desenvolvimento. O *Scrum* diário faz três perguntas que promovem a visibilidade do processo para todo o time:

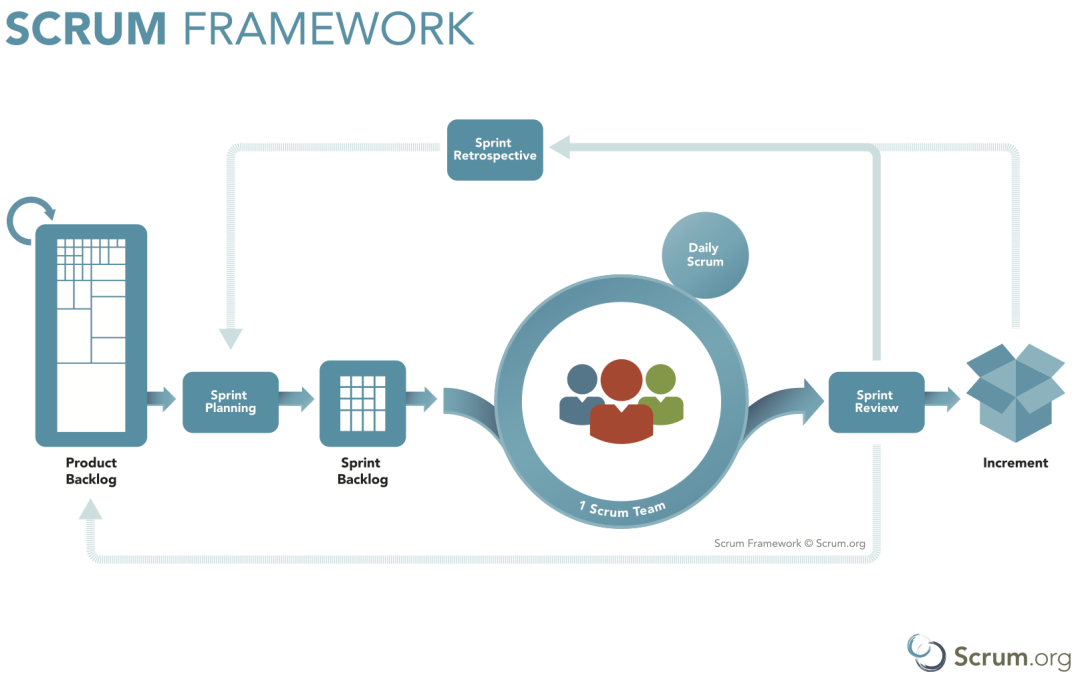
  1. O que foi feito no último dia?
  2. O que será feito hoje?
  3. Existe algum impedimento para o processo de desenvolvimento?

- **Sprint Review** (revisão da *Sprint*): A revisão da *Sprint* é uma reunião, de no máximo 4 horas, com os *stakeholders*, onde o time de *Scrum* recebe *feedback* sobre o resultado da última iteração, elicitando mudanças necessárias no *Product Backlog*

para aumentar o valor do produto.

- **Sprint Retrospective** (retrospectiva da *Sprint*): a retrospectiva *Sprint* é uma reunião que ocorre entre o *Sprint Review* e o próximo *Sprint Planning* e tem o propósito de refletir sobre o desempenho do time na última *Sprint*, ela possibilita que o time crie propostas para melhorar o processo e analise como corrigir problemas frequentes.

Figura 3.1: Visão ilustrativa dos eventos realizados no *Scrum*



Fonte: (SUTHERLAND, 2017)

Os artefatos do *Scrum* são classificados em:

- **Product Backlog** (backlog do produto): o *Product Backlog* é uma lista, ordenada por prioridade, de tudo que é necessário se ter no produto; é um artefato mutável que vai de acordo com novas prioridades ou obsolescência de outras.
- **Sprint Backlog** (backlog do Sprint): o *Sprint Backlog* é um conjunto de itens pertencentes ao *Product Backlog* que serão desenvolvidos durante o *Sprint* corrente. Ao entrar na *Sprint*, os itens também são mais detalhados para responder perguntas como "Quando esta funcionalidade está pronta?"
- **Increment** (incremento): o *Increment* é o conjunto de itens que foram completos durante o *Sprint*. Um incremento deve ser sempre usável e testável - quaisquer tarefas pendentes não entram no incremento. Ele é um passo em direção à compleição

do produto, sendo ou não entregue imediatamente.

3.1.1.1 ScrumBut

O uso de *ScrumBut*, ou "*Scrum* mas", implica que o uso de *Scrum* não foi completo, ou seja, que algumas práticas foram adaptadas devido a limitações do time ou características específicas do projeto que está sendo desenvolvido. Segundo (SUTHERLAND, 2019), a prática é definida pela sintaxe: "Nós usamos *Scrum* mas (motivo), (solução alternativa)".

Exemplos do uso do *ScrumBut*:

- "Nós usamos *Scrum* mas, *Daily Scrum* todo o dia atrasa nosso processo, então fazemos o *Daily Scrum* segundas e quintas."
- "Nós usamos *Scrum* mas, retrospectivas são uma perda de tempo, então não as fazemos".

Devido ao fato de o projeto ser desenvolvido por apenas uma pessoa, alguns aspectos e eventos do *Scrum* acabaram sendo inutilizados ou flexibilizados:

- Nós usamos *Scrum* mas, como só tem um desenvolvedor no processo, ele faz o papel de mestre de *Scrum* e de desenvolvedor.
- Nós usamos *Scrum* mas, como só tem um desenvolvedor no processo, não fazemos reuniões diárias.
- Nós usamos *Scrum* mas, como só tem um desenvolvedor no processo, não fazemos retrospectiva da *Sprint*.

3.1.2 Histórias de Usuário

As histórias de usuário são artefatos que descrevem funcionalidades que irão gerar algum valor, seja para o usuário, seja para o cliente ao qual o sistema está sendo desenvolvido (COHN, 2004). Elas foram estruturadas no projeto, levando em consideração o tipo do usuário, da seguinte forma:

Como [tipo de usuário] eu quero [funcionalidade] para que [valor agregado].

As histórias de usuário definidas para este projeto são as seguintes:

1. Como professor, quero adicionar, remover e editar alunos para poder adicionar eventos aos meus alunos

2. Como professor, quero fazer o envio de áudio, vídeo e imagens das realizações de um aluno para que eu possa registrar suas realizações.
3. Como professor eu quero fazer o acesso usando minhas credenciais (e-mail e senha) para que eu possa visualizar minhas informações seguras.
4. Como professor eu quero me registar usando minhas credenciais (e-mail e senha) para que eu possa manter minhas informações seguras.
5. Como professor, quero poder ver os eventos dos meus alunos para que eu possa acompanhar o progresso deles.
6. Como professor, quero sair da minha sessão para que outros professores possam usar o mesmo dispositivo.
7. Como professor, quero ver os eventos dos alunos em uma linha do tempo para que eu possa visualizar claramente quando cada evento ocorreu.
8. Como professor eu quero ver os conteúdos dos eventos para que eu possa ter uma melhor compreensão do evento.
9. Como professor, quero exportar opiniões em PDF para poder compartilhar dados importantes fora do aplicativo.
10. Como professor, quero fazer o mais rápido possível para adicionar eventos para que eu não perca as conquistas importantes.
11. Como professor, quero criar vários tipos de opiniões para que eu possa ter relatórios detalhados para cada situação.
12. Como professor eu quero excluir alunos e eventos para que eu possa organizar o meu trabalho.
13. Como professor eu quero ter informações precisas exibidas nos formulários para que as informações apresentadas nas opiniões sejam válidas.
14. Como professor, quero que a interface do aplicativo seja perfeita e rápida de usar, para que eu possa registrar eventos de alunos com eficiência.
15. Como professor, eu quero poder me referir a eventos com '@' enquanto edito opiniões para que eu possa citá-las sem esforço.
16. Como professor, quero poder ver eventos no meu navegador para poder visualizar eventos fora do dispositivo.
17. Como professor, quero poder compartilhar links para eventos através do aplicativo, para poder compartilhar o desempenho do aluno com pessoas autorizadas.
18. Como professor eu quero ser capaz de criar um parecer para que não precise fazer

isso fora do aplicativo.

19. Como professor eu quero ser capaz de criar adequações curriculares para que não precise fazer isso fora do aplicativo.

3.1.3 Quadro de *Scrum*

O quadro de *Scrum* é uma ferramenta de visibilidade utilizada para acompanhar e organizar as *Sprints*, para o desenvolvimento do sistema o software utilizado foi o Trello, uma ferramenta de colaboração que organiza projetos em quadros e listas de tarefas (TRELLO, 2017), o que se encaixa perfeitamente com o uso do quadro de *Scrum*.

Figura 3.2: Visão do quadro de tarefas *Scrum* no Trello

Fonte: Provido pelo autor.

3.1.4 Sprints

Nesta seção serão detalhadas as *Sprints* realizadas durante o processo de desenvolvimento do produto.

O tamanho da caixa de tempo destinada a cada *Sprint* foi de três semanas, e esse tempo permaneceu inalterado durante todo o processo de desenvolvimento. Durante o

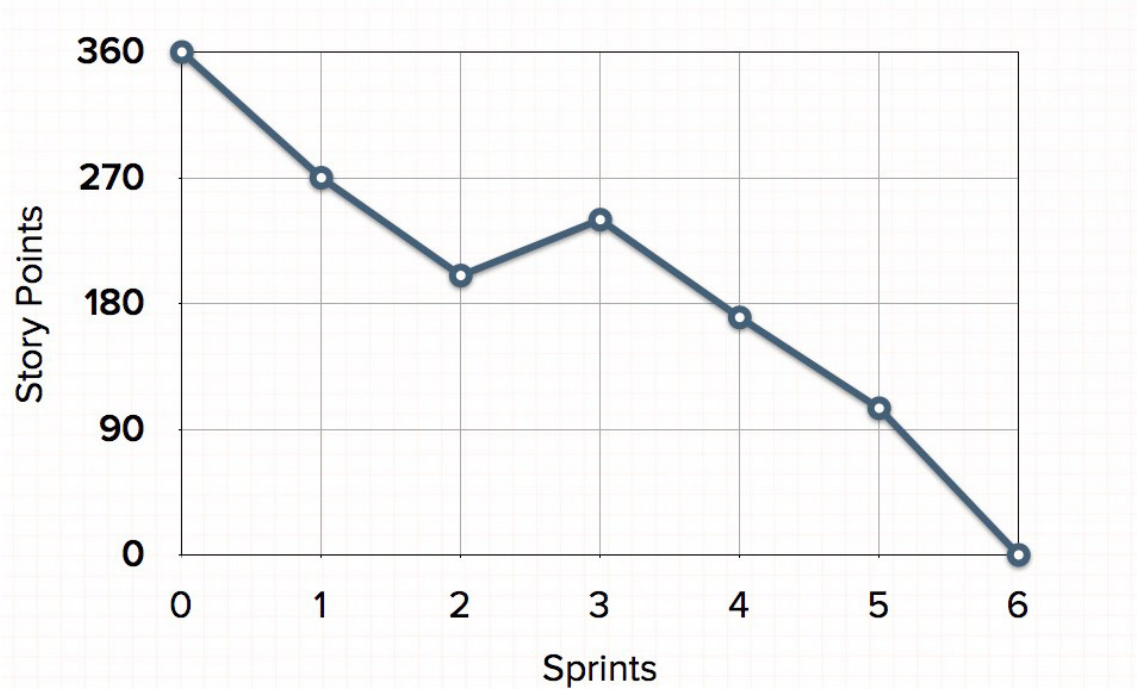
andamento foram realizados dois *Releases* e quatro *Sprints*:

- Sprint 1: protótipo inicial do cliente móvel com integração com o servidor e CRUD de estudantes, entradas e autenticação.
- Sprint 2: composição de eventos e documentos, com todos os tipos de mídias suportados e referências. Release 1.
- Sprint 3: composição de múltiplos tipos de documentos e implementação do design.
- Sprint 4: criação do cliente web de visualização e criação das regras de segurança. Release 2.

3.1.4.1 Artefatos gerados

Para ajudar na melhora do processo, a cada incremento foi usado um artefato chamado *Burndown Charts* 3.3, o qual é um gráfico que mapeia o progresso da *Sprint* no tempo estimado, avaliando os pontos de tarefas realizadas a cada dia de trabalho. A partir dele podemos inferir dicas sobre impedimentos que podem estar afetando o desenvolvimento (SOFTWARE, 2019).

Figura 3.3: Exemplo de um *Burndown Chart*



Fonte: Mountain Goat Software - https://www.mountaingoatsoftware.com/agile/scrum/scrum-tools/release-burndown

3.2 Arquitetura do Sistema

Para a escolha da arquitetura do sistema foi imprescindível a análise prévia das histórias de usuário. Como o foco do sistema é uma aplicação móvel, a opção mais simples para a arquitetura seria apenas um aplicativo autocontido, que armazenaria todos os dados no próprio dispositivo do professor. O tipo de aplicação citado é bastante utilizado por soluções utilitárias ou pessoais, como "Calculadora", "Lista de afazeres"e "Jogos (não multijogador)".

Porém, os requisitos e funcionalidades presentes nas histórias inviabilizam essa opção, já que um dos aspectos principais dos requisitos é o desenvolvimento de dois tipos de aplicativos, um móvel capaz de capturar e salvar diversos tipos de mídias, e um para a web, capaz de visualizar as capturas do aplicativo móvel no ambiente de um navegador, portanto seria necessário criar alguma forma de compartilhar dados entre os dois. Além disso um outro cenário possível é que um ou mais usuários possam querer ter acesso concorrente a um mesmo dado de um aluno, e inclusive funcionalidades mais básicas como, por exemplo, um usuário deve conseguir acessar sua conta por diferentes dispositivos. Essa relação $1-N$ entre dados e dispositivos descarta qualquer possibilidade de uma abordagem puramente no dispositivo. A abordagem que se relaciona muito bem com essa condição de $1-N$ e é capaz de atender aos demais requisitos é a cliente-servidor (KUROSE; ROSS, 2017).

3.2.1 Cliente-servidor

Na arquitetura cliente-servidor temos uma ou mais instâncias de uma aplicação Servidor executando (no caso de mais de uma, são usadas técnicas de virtualização para dar a ilusão de um servidor apenas) (KUROSE; ROSS, 2017), e zero ou mais aplicações-clientes. O servidor funciona com o intuito de servir informação e funcionalidades para as aplicações-clientes, informação essa que tende a ser correlacionada entre os mesmos. Um dos pontos mais importantes deste modelo é que clientes não mantêm comunicação direta, somente através de um servidor.

O modelo cliente-servidor foi escolhido pois se encaixa muito bem na especificação dos requisitos, provendo autenticação e dados de forma centralizada. Além disso, conseguimos mitigar muitos dos pontos negativos dessa arquitetura usando serviços de *BaaS*, como o servidor ser um ponto único de falha para o sistema, com uso de serviços

que garantem disponibilidade e segurança extremamente altas a custos baixos.

3.2.1.1 Servidor

O servidor deve prover serviços de autenticação, armazenamento de dados e arquivos da aplicação descritos da seguinte forma.

1. Autenticação
    1. Autenticar com e-mail e senha.
    2. Cadastrar com e-mail e senha.
3. Armazenamento de arquivos
    1. Upload de arquivos.
    2. Download de arquivos.
3. Banco de dados não relacional
    1. Criação de documentos representantes das entidades no formato JSON.
    2. Atualização de documentos representantes das entidades no formato JSON.
    3. Deleção de documentos representantes das entidades no formato JSON.
    4. Leitura de documentos representantes das entidades no formato JSON.

Para o servidor também há um requisito não-funcional de segurança dos dados muito importante, já que as informações dos alunos armazenadas no banco de dados do mesmo tem um caráter extremamente sensível.

3.2.1.2 Clientes

Os clientes precisam de uma camada de serviços, para que assim consigam consumir a API exposta pelo servidor, especificada na seção acima, este consumo deve ser feito de forma segura e através da internet.

3.3 Entidades

Nesta seção serão definidas as entidades persistidas no banco de dados de sistema. As entidades descritas apresentam também o tipo de dados básicos que as representam no banco, para a referência dos tipos foi usado o provedor de banco de dados não relacional

Cloud Firestore (CLOUD, 2019b). Foram tomadas liberdades de tradução para o nome de cada uma das entidades para deixar mais clara o papel semântico das mesmas.

3.3.1 Professor

Esta entidade representa o professor, e contém informações necessárias para a aplicação ser capaz de compor os documentos propostos. Seu identificador está associado ao correspondente do usuário nos registros de autenticação que contém sua senha cifrada. Todos os outros documentos refentes a alunos desse professor também contêm seu identificador.

- **id**: Identificador do tipo UUID (LEACH; MEALLING; SALZ, 2005) que garante a unicidade de cada documento do tipo *Professor* armazenado como String.
- **timestamp**: Data de criação do documento do tipo *Professor*, armazenado como *Number* que representa o intervalo de tempo a partir de uma data de referência. (APPLE, 2019)
- **name**: Nome completo do professor, armazenado como *String*.
- **email**: Endereço de e-mail do professor, armazenado como *String*.
- **phoneNumber**: Número de contato do professor, armazenado como *String*.

3.3.2 Aluno

Esta entidade representa o aluno e contém toda a informação necessária para a produção de documentos e eventos relacionados. Relaciona-se com a entidade *Professor*, que representa seu respectivo professor.

- **id**: Identificador do tipo UUID que garante a unicidade de cada documento do tipo *Aluno* armazenado como *String*.
- **timestamp**: Data de criação do documento do tipo *Aluno*, armazenado como *Number* que representa o intervalo de tempo a partir de uma data de referência.
- **name**: Nome completo do aluno, armazenado como *String*.
- **fatherName**: Nome completo do pai do aluno, armazenado como *String*.
- **motherName**: Nome completo da mãe aluno, armazenado como *String*.
- **birthDate**: Data de nascimento do aluno, armazenado como *Number* que repre-

senta o intervalo de tempo a partir de uma data de referência.

- **generalRegistry**: Número do RG do aluno, armazenado como *String*.
- **history**: Informações adicionais sobre o histórico do aluno que podem ser relevantes para análises futuras, armazenadas como *String*.
- **classNumber**: Turma do aluno em questão para referência e organização, armazenada com *String*.
- **series**: Ano do aluno com base nas diretrizes e bases da educação nacional brasileira (BRASILEIRO, 1996), armazenado como *String*.
- **shift**: Turno de atendimento do aluno, armazenado como *String*.
- **phoneNumber**: Número de contato do aluno, armazenado como *String*.
- **responsibleName**: Nome do responsável pelo aluno, armazenado como *String*.
- **relationship**: Tipo de relacionamento do responsável com o aluno, armazenado como *String*.
- **address**: Endereço do responsável pelo aluno, armazenado como *String*.
- **cid**: Palavra alfanumérica que representa o código internacional de doenças relacionada ao aluno (ORGANIZATION et al., 1988), armazenada como *String*.
- **specialNeeds**: Lista de condições que o aluno apresenta, armazenada como *Array de Strings*.
- **termsOfUse**: Indica se o aluno está de acordo com a liberação do uso de seus dados pelo professor, armazenado como *Boolean*.
- **schoolId**: Identificador da escola à qual o aluno está cadastrado, armazenado como *String*.
- **teacherId**: Identificador do professor ao qual o aluno está cadastrado, armazenado como *String*.

3.3.3 Documento

Esta entidade representa possíveis documentos gerados pelo professor, estes se diferenciam pelo atributo *content* fazendo com que seus campos sejam dinâmicos.

- **id**: Identificador do tipo UUID que garante a unicidade de cada documento do tipo *Documento* armazenado como *String*.
- **timestamp**: Data de criação do documento do tipo *Documento*, armazenado como

Number que representa o intervalo de tempo a partir de uma data de referência.

- **tags**: Conjunto de marcadores do documento, informando a que áreas do conhecimento ele está relacionado, armazenado como *Array de Strings*.
- **content**: Este atributo representa o conteúdo do documento e é representado por uma série de chaves e valores em um *Map*, que representam as seções e o conteúdo da mesma respectivamente, sua configuração muda de acordo com o tipo do parecer, permitindo que novos tipos de estruturações do conteúdo sejam configurados dinamicamente.

3.3.4 Evento

Esta entidade representa eventos de estudantes salvos por seu professor, podendo estar relacionada a eventos de vídeo, áudio, texto ou foto - este comportamento é informado pelo atributo *mediaType*.

- **id**: Identificador do tipo UUID que garante a unicidade de cada documento do tipo *Evento* armazenado como *String*.
- **timestamp**: Data de criação do documento do tipo *Evento*, armazenado como *Number* que representa o intervalo de tempo a partir de uma data de referência.
- **mediaType**: Tipo do conteúdo (vídeo, áudio, texto ou imagem) do evento, armazenado como *String*.
- **comment**: Comentário adicional para contextualizar o evento, armazenado como *String*.

3.3.5 Escola

Esta entidade representa as escolas as quais os estudantes estão associados, elas funcionam para facilitar tarefas de agrupamento dos estudantes e para o preenchimento de pareceres.

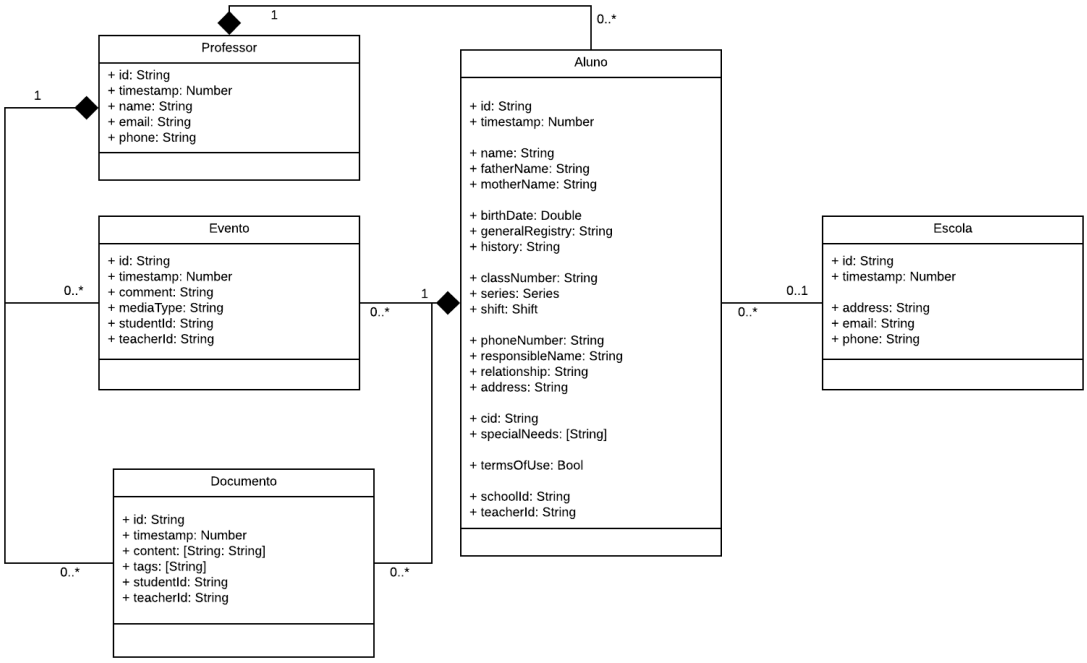
- **id**: Identificador do tipo UUID que garante a unicidade de cada documento do tipo *Escola* armazenado como *String*.
- **timestamp**: Data de criação do documento do tipo *Escola*, armazenado como *Number* que representa o intervalo de tempo a partir de uma data de referência.

- **email**: Endereço de e-mail da escola, armazenado como *String*.
- **phoneNumber**: Número de contato da escola, armazenado como *String*.
- **adress**: Endereço físico da escola, armazenado como *String*.

3.3.6 Diagrama de entidades e seus relacionamentos

Diagrama UML de entidade-relacionamento representativo das entidades do banco de dados definidas (Figura 3.4).

Figura 3.4: Diagrama UML das entidades



Fonte: Provido pelo autor.

4 IMPLEMENTAÇÃO

Neste capítulo será explicado o processo de desenvolvimento dos diferentes componentes do sistema: o servidor, o cliente móvel e o cliente web. Para cada serão apresentadas as técnicas utilizadas e escolhas de tecnologias.

4.1 Servidor

A implementação no Servidor elenca várias dificuldades como: onde ele estará hospedado, como faremos o gerenciamento deste hospedeiro, como configurar comunicação segura por HTTPS (RESCORLA, 2000). Além disso, também temos requisitos não funcionais importantes para podermos prover uma boa experiência de usuário:

- Escalabilidade: o sistema deve escalar facilmente, sem a necessidade de retrabalho futuro nesta condição.
- Segurança: devido à natureza sensível das informações contidas no servidor (dados educacionais e sociais de menores de idade), é imperativo que a privacidade dos usuários seja mantida.
- Disponibilidade: a natureza das situações de uso da aplicação é efêmera, então para não implicarmos em perdas para os usuários, o servidor não deve ter tempos de baixa significantes.

Isso tudo elenca um custo além do disposto para a realização do projeto, ainda mais considerando que as funcionalidades esperadas do servidor são comuns e não muito complexas. Como esse não é o foco deste trabalho, optou-se por salvar o esforço de desenvolvimento do mesmo, dessa forma, buscamos uma alternativa que conseguisse poupar tal trabalho sem comprometer a viabilidade do sistema final, conformando com os requisitos definidos acima.

4.1.1 Serverless Computing

O paradigma *Serverless Computing* (BALDINI et al., 2017), "computação sem servidor", vem sendo usado para definir implementações da arquitetura cliente-servidor onde a implementação do servidor não é feita pelos desenvolvedores da aplicação, e sim

por terceirizados para serviços externos na forma de um conjunto de microsserviços, que provêm tanto funcionalidades comuns a vários servidores quanto a possibilidade do desenvolvedor configurar novas funções para serem executadas se necessário, tudo sem se preocupar com o estado da aplicação. Este paradigma é uma evolução do *BaaS* em que além de prover infraestrutura, também são providas funcionalidades básicas de servidores.

Ao delegar a implementação do servidor a terceiros, times pequenos têm muito mais tempo de desenvolvimento para focar na experiência do usuário e robustez da aplicação.

4.1.1.1 Opções de provedores

Nesta seção iremos analisar comparativamente soluções possíveis de *BaaS* e apresentar a solução escolhida. AWS e Azure As plataformas de computação em nuvem Azure da Microsoft (MICROSOFT, 2019) e AWS da Amazon (AWS, 2019) oferecem uma gama incrível de serviços, *Serverless* que possibilitam escalabilidade total para o número de usuários do sistema, fazendo delas ótimas escolhas para a criação de aplicações modernas, entretanto o custo para configurá-las ainda é significativo.

Firebase No projeto SIR-EDU escolhemos a solução *Firebase* (CLOUD, 2019a), pelo seu foco em facilitar o processo de desenvolvimento e evolução de aplicações móvel. Ela é uma plataforma construída sobre o *Google Cloud Services*, dispondo do mesmo nível de confiabilidade da sua plataforma mãe, mas com o foco absoluto em desenvolvimento para plataformas móvel e web, trazendo uma gama de ferramentas nesse sentido, como testes A/B, notificações *PUSH*, etc.

Como seu foco é prover a maior facilidade possível aos clientes (nesse caso os desenvolvedores), *Firebase* também nos dá acesso a *SDK's* que permitem implementar as camadas de consumo das *API's* do servidor com muito mais facilidade. Dos serviços providos, utilizamos os seguintes:

- *Firebase Auth*: serviço de autenticação que oferece diversas opções para realizar o cadastro e credenciamento dos usuários. Para o desenvolvimento do sistema discutido, optou-se inicialmente pelo uso de e-mail e senha, de acordo com as histórias de usuário.
- *Cloud Firestore*: serviço banco de dados NoSQL de documentos, com suporte à sincronização de entradas em tempo quase real.

- *Cloud Storage*: serviço armazenamento de objetos, utilizado para armazenar os conteúdos dos eventos (arquivos .mp3, .mp4 e .jpg) gerados pelo cliente móvel.

4.1.2 Segurança

No quesito segurança, *Firebase* garante tanto que a transferência de dados seja segura ao realizar todas as comunicações entre o servidor e os clientes por HTTPS, que estende o HTTP com uma camada adicional de segurança SSL/TLS, quanto no armazenamento, já que as informações são mantidas em discos criptografados (NADA, 2019), fazendo com que, mesmo em um cenário onde esses dados caíssem na mão de um invasor, eles ainda tenham uma camada de proteção.

Para garantir o controle de acesso aos dados do sistema também são utilizadas regras de segurança, essas regras, que são definidas no servidor e aplicadas para todos os usos dos pontos de acessos das *API's* providas, garantem várias propriedades desejadas tanto para o banco de dados quanto para o armazenamento de arquivos independente do cliente, como:

- Somente usuários autenticados podem realizar consultas.
- Cada usuário tem acesso somente a informações de alunos que cadastrou.
- O usuário não pode realizar o cadastro de alunos sem tê-lo como professor.
- O usuário não pode sobrescrever o criador de qualquer documento.

Com o uso dessas medidas de segurança os pontos de acesso das *API's* de banco de dados e armazenamento de arquivos não ficam vulneráveis ao acesso indevido de terceiros.

4.2 Cliente móvel

Nesta seção vamos revisar em detalhes a implementação do cliente móvel, que compreende a maior parte do sistema.

4.2.1 Nativo x Multiplataforma

Antes de começar o processo de desenvolvimento de aplicações móveis temos uma decisão importante para tomarmos, desenvolver nativamente ou multiplataforma.

Desenvolver nativamente significa que para implementarmos o produto usaremos a linguagem própria do sistema operacional escolhido, isso implica no uso de *Java* e *Kotlin* para a plataforma *Android* ou *Swift* e *Objective-C* para o *iOS*.

Já para desenvolvimento multiplataforma (PALMIERI; SINGH; CICCHETTI, 2012) podemos escolher entre uma gama de *frameworks* que nos permitem desenvolver um projeto genérico para múltiplos sistemas operacionais usando linguagens intermediarias, muitas vezes a linguagem utilizadas nessas ferramentas é o *Javascript*, trazendo a força de ferramentas de programação para web ao ambiente móvel. Esses *frameworks* usam a API nativa de cada plataforma mapeando um conjunto de componentes para o uso do programador com a linguagem escolhida. Para fazer o uso de recursos dependentes do sistema operacional, como acessar a câmera ou a biblioteca de fotos, frequentemente ainda é necessário o uso das API's nativas.

Aplicações nativas fazem o uso completo da otimização da plataforma (CORRAL; SILLITTI; SUCCI, 2012), além de oferecer o controle completo da aplicação independente de suporte de componentes multiplataforma permitindo a implementação de interfaces mais complexas e customizadas, o que pode melhorar tanto aspectos de performance quanto de UX.

Aplicações multiplataforma fazem mais sentido quando a prioridade atingir o maior número de usuários na menor quantidade de tempo, sem ter que gastar com o desenvolvimento de duas aplicações.

Ambos nativo e multiplataforma têm seus pontos fortes e fracos, porém considerando a efemeridade dos momentos de uso da aplicação, como registrar a fala de um aluno, temos um requisito não funcional de desempenho alto com as funções de captura de mídia do sistema operacional. Como um dos objetivos da criação do cliente móvel é prover uma boa interface para o professor usuário gerencia seus alunos, para termos mais liberdade na criação da UX e manter o desempenho o mais alto possível prosseguiremos com uma solução nativa.

4.2.2 Android vs. iOS

Escolher a plataforma foi outra decisão de implementação bem importante, pois os respectivos sistemas operacionais restringem diretamente as possibilidades de funcionalidades e a experiência do usuário ao usar a plataforma.

4.2.3 iOS

A escolha acabou sendo iOS pois, embora o Sistema operacional Android tenha a maior fatia do mercado, no quesito de suporte e uso na área educacional, a maior gama de aplicações utiliza o iOS (FERNÁNDEZ-LÓPEZ et al., 2013). Isso se dá pelas garantias de privacidade e experiência de usuário que são asseguradas por políticas severas de aceitação de aplicações pela loja de aplicativos *App Store*.

4.3 *Swift*

Swift (INC., 2019a) é uma linguagem de propósito geral com código aberto. Disponibilizada em 2015, estando agora na versão 5.1, é uma linguagem relativamente nova. Por mais que sua idade signifique que ela ainda não é madura o suficiente, ela possibilitou *Swift* a incorporar várias novas tendências que vinham se popularizando na área de desenvolvimento de sistemas - tanto grandes quanto pequenos -, dando aos seus programadores maneiras modernas e familiares de lidar com segurança, performance e padrões de design. Segundo seus desenvolvedores, a maneira intuitiva de escrever programas em *Swift* deve ser:

1. Segura: a maneira mais óbvia de escrever *Swift* também deve ser a maneira mais segura. *Swift* é fortemente tipada e, permite liberdade ao programador para ignorar essas restrições, no entanto ele deve fazer isso de maneira consciente.
2. Rápida: a ideia dos criadores de *Swift* é que ela substitua linguagens baseadas em C (como *C++, C#* e o antigo *Objective-C*), portanto ela oferece performance, predicabilidade e consistência.
3. Expressiva: como expressado antes, *Swift* traz consigo décadas de conhecimento, trazendo uma sintaxe que se atualiza constantemente, oferecendo tudo que os programadores necessitam de diversas linguagens.

Dentre as funcionalidades da linguagem *Swift* estão presentes (INC., 2019d):

- Inferência de tipos.
- Tuplas.
- Funções de primeira classe.
- Estruturas.
- Extensões.
- Protocolos, artefatos similares a junção de interfaces e classes abstratas do Java, protocolos podem definir tanto assinaturas de funções e variáveis como fornecer implementações com a combinação de extensões.
- Funções de alta ordem, como map, $fitler$ e $reduce$.
- Tratamento de erros.
- Iteradores concisos.
- Controle de fluxo avançado, com instruções do tipo $guard$ e $defer$.
- Variáveis e constantes são inicializadas antes do uso.
- Memória é administrada automaticamente com o uso de contagem automática de referências.

Valores em Swift não podem ser nulos por padrão. Logo, para operar com o valor nulo precisamos usar o tipo $Optional$ representado pelo açúcar sintático '?'. Para obter os valores não nulos da variável temos diversas opções (INC., 2019c):

- Encadeamento de *Optionals*: acessa propriedades do valor condicionalmente ao próprio objeto ser ou não nulo.
- Valores padrão: utiliza o operador '??' para fornecer um valor não nulo padrão no caso do valor avaliar para nulo.
- Atribuição de *Optionals*: atribui o valor do *Optional* a uma outra variável condicionalmente ao seu valor ser nulo ou não.
- *Force-unwrap*: com o açúcar sintático '!' o $force-unwrap$, força um valor não nulo de um $Optional$, se por acaso o $force-unwrap$ avaliar para um valor nulo é lançado um erro de tempo de execução.

Essas características se mostraram muito úteis durante a implementação do cliente móvel, pois o uso da linguagem é natural, e erros são fáceis de detectar e corrigir devido a expressividade dos compromissos do programador em declarar variáveis que podem tomar valores nulos. Essa facilidade para programar de forma segura é muito valiosa para

desenvolvimento de interfaces de usuário, uma vez que falhas inesperadas não são um bom indicador de qualidade para o usuário final.

4.3.1 Arquitetura do Cliente

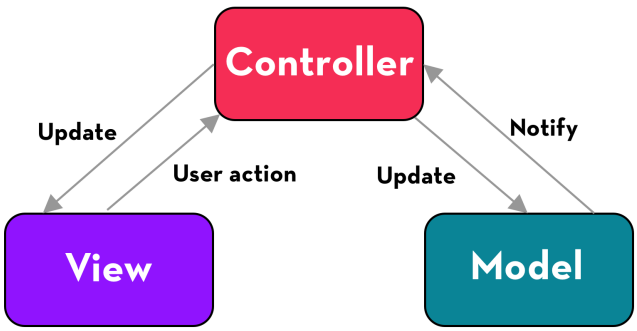
A arquitetura escolhida para o Cliente diretamente influencia a manutenibilidade da aplicação no futuro. Assim, nesta seção serão discutidas as opções de arquitetura disponíveis e como balanceamos a complexidade com a praticidade de cada uma.

4.3.1.1 MVC

A arquitetura MVC é a mais comumente utilizada para desenvolvimento de aplicações, ela divide a aplicação em três partes (REENSKAUG; COPLIEN, 2009):

- Model (Modelo): O Model, é a estrutura da aplicação que independe de qualquer interface que venha a ser implementado, é com ele que quaisquer dados da aplicação são acessados e alterados, e onde as regras de negócio são mantidas.
- Controller (Controlador): O Controller é quem reage à interação do usuário nas *Views* e atualiza as mesmas com informações provenientes do Modelo que são tratadas previamente.
- View (Visão): A camada de interface da aplicação; é por onde as interações do usuário como toques, gestos, repostas em texto, etc, são captadas para serem tratadas no *Controller*. Elas refletem as informações do Model, as quais são passadas pelo *Controller*.

Figura 4.1: Arquitetura MVC



Fonte: Elaborada pelo autor

O MVC foi uma das primeiras soluções existentes nesse sentido de isolar a in-

terface da aplicação do seu funcionamento interno, por isso ele é um padrão básico que é feito para ser alterado conforme as necessidades de cada aplicação, ditando apenas os princípios de separação de responsabilidades.

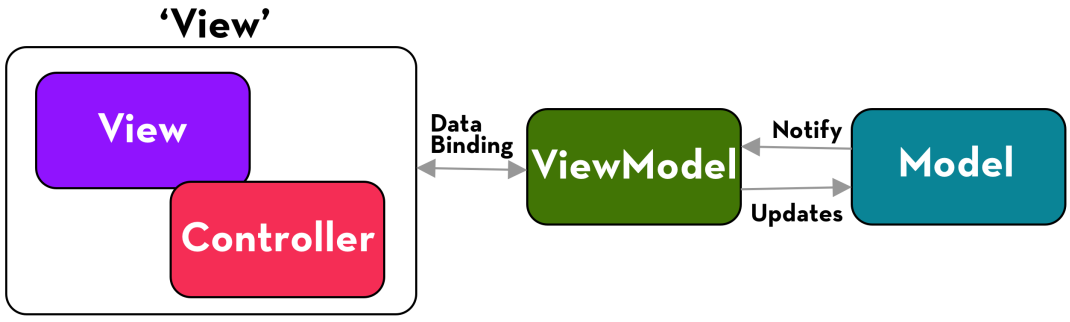
Seu uso é o previsto pelo ambiente de programação iOS, entretanto, com a maturidade do processo de desenvolvimento para a plataforma, foi sendo observado que o uso do MVC acaba que os controladores acabam ficando massivos, criando a terminologia informal *Massive View Controllers*, depois de um certo ponto, já que os mesmos acabam sendo responsáveis por: preparar dados do modelo para a UI, configurar os elementos da UI, realizar a chamada das camadas de rede, atualizar o modelo, realizar o tratamento de eventos de UI e realizar a amarração (*binding*) de dados da UI.

Esse acoplamento de tantas responsabilidades acaba criando monolitos gigantescos nos *controllers*, logo, como queremos dar facilidade a trabalhos futuros a partir desse projeto, essa não é a solução.

4.3.2 MVVM

Para resolver nossos problemas com a arquitetura, buscamos uma divisão maior do trabalho centrado nos *Controllers*. Criado pelos arquitetos da Microsoft, Ted Peters e Ken Cooper em 2005 (COOPER; PETERS, 2005), uma camada nova entre a *View* e o *Model* chamada *View model* foi adicionada. O *View model* realiza a amarração de dados do *Model* na *View*, sem uma necessidade externa de sincronização, para isso é bem útil o padrão *Observer*, que veremos em sequência. O *ViewModel* toma a responsabilidade de interagir com o modelo que era previamente do *Controller*, simplificando o mesmo. Além disso, também estabelece um padrão claro de comunicação, através da amarração.

Figura 4.2: Arquitetura MVVM



Fonte: Elaborada pelo autor

No seu uso aplicado para programação iOS ainda fazemos o uso de *Controllers*, porém agora ficam significativamente mais leves, uma vez que só tratam da formatação dos dados provenientes do *View model* e das interações do usuário, que serão transformadas em atualizações do *model*.

A função de amarração acaba causando um custo inicial alto, como aponta John Gossman (GROSSMAN, 2009), e também requer um conhecimento mais maduro da arquitetura para usar o padrão de forma correta em situações mais complexas.

4.3.3 Padrões de Design

Serão descritos agora os padrões de design de software (SOMMERVILLE, 2011) utilizados no projeto e o papel de cada um em atingir a aplicação final. Padrões de design não são implementações especificas a serem seguidas e sim comportamentos que com o tempo se mostraram uteis para resolver problemas que surgem no desenvolvimento de produtos de software.

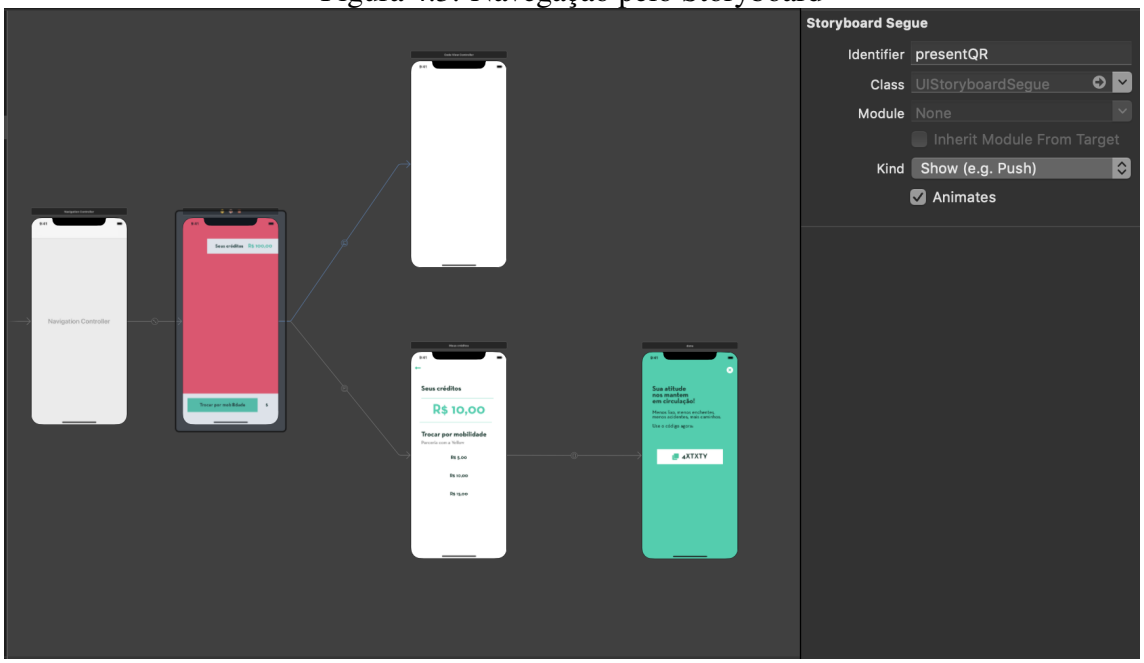
4.3.3.1 Padrão Coordinator

O comportamento nativo do ambiente de desenvolvimento iOS fornece um suporte gráfico para a programação de navegação e interfaces do usuário, o *Storyboard* (Figura 4.3), o que possibilita um tempo de desenvolvimento bem rápido - fator positivo para o desenvolvimento de protótipos. Essa facilidade vem com um custo alto, pois o uso dessa ferramenta acaba causando um acoplamento indesejado na programação da interface de usuário, além de dificultar o processo de injeção de dependências, que é bastante usado no projeto, pois o uso dessa ferramenta deixa transparente a instanciação das *controllers*.

O padrão de design *Coordinator* foi criado para dispensar o uso dessa ferramenta através da concepção da classe *Coordinator*, a qual toma as responsabilidades de navegar entre as diferentes cenas da aplicação, instanciando as controladoras quando necessário.

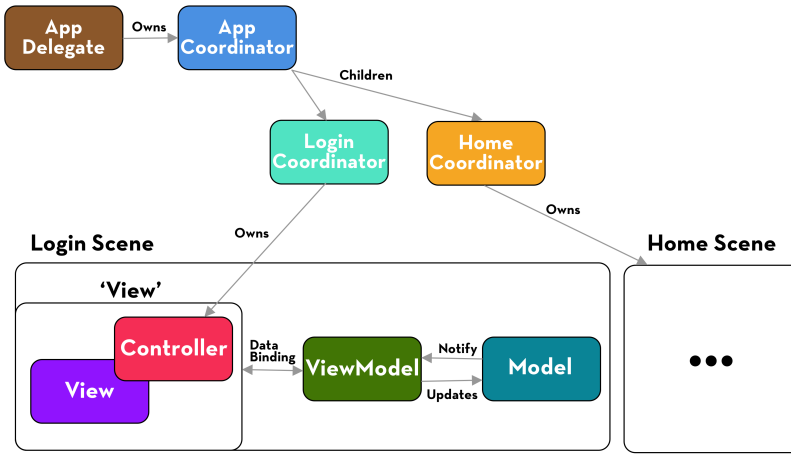
Como pode ser visto na Figura 4.4, ao iniciar a aplicação, é criado um *App Coordinator* que fica responsável por principiar a exibição do app, este já é injetado com o nosso provider e a janela do device. O *App Coordinator*, por sua vez, delega ao *Coordinator* de um cena inicial, esse Coordinator é responsável por criar e mostrar as interfaces relacionadas à cena na janela da aplicação. Os Coordinators são encarregados de apenas uma parte mínima da navegação, se alguma controladora não estiver estritamente relacio-

Figura 4.3: Navegação pelo Storyboard



Fonte: Elaborada pelo autor

Figura 4.4: Padrão *Coordinator*



Fonte: Elaborada pelo autor

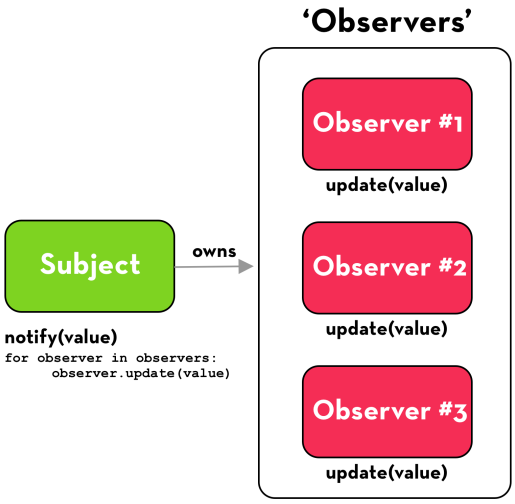
nada ao propósito de um coordenador, ela deve fazer parte de outro, para manter a coesão semântica de cada coordenador. Coordenadores se relacionam um ao outro através de nexos de pai/filho, assim que sua cena não estiver mais associada à pilha de navegação da aplicação ele é desativado para evitar vazamento de memória.

4.3.3.2 Padrão Observer

O padrão *Observer* (Observador) é um padrão onde um *Subject* (Sujeito) emite eventos para um ou mais *Observers*, que são mantidos pelo mesmo. Quando um evento é

expedido pelo *Subject* os seus *Observers* são notificados. Esse padrão é muito interessante quando usado em conjunto com a arquitetura, já que apresenta uma forma de viabilizar o vinculação de dados que acontece entre as camadas de *View model* e *View*.

Figura 4.5: Padrão *Observer*



Fonte: Elaborada pelo autor

4.3.3.3 RxSwift

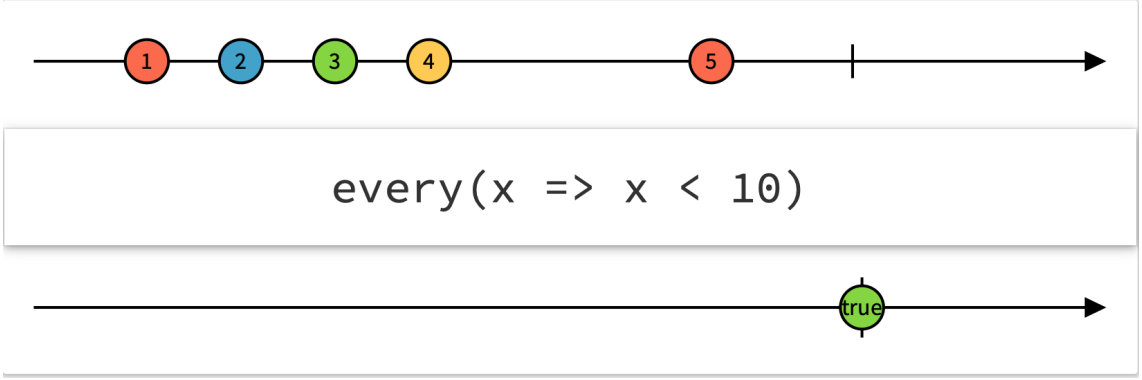
Para implementar o padrão *Observer* foi usada a biblioteca *RxSwift* (REACTIVEX, 2019), a qual nos fornece implementações de alto desempenho do padrão citado com várias ferramentas que permitem expandir o poder desse comportamento aparentemente simples.

Com *RxSwift* os eventos emitidos por um *Subject* tem o mesmo comportamento de uma sequência normal, porém com um detalhe adicional muito valioso: os elementos podem ser adicionados assincronamente. Todo o resto apenas expande nessa ideia.

RxSwift nos viabiliza um conjunto de ferramentas, chamados operadores, para utilizarmos na manipulação das cadeias de eventos, mitigando o peso de computação do observador em diversos operadores no caminho, funcionando como açúcar semântico para as amarrações. Os operadores previstos podem ser classificados nos seguintes tipos:

- Operadores de condição (Figura 4.6): operadores que alteram o resultado dependendo dos eventos que foram emitidos. Foram utilizados para o tratamento de erros de interface, para que no caso de erro, ou falha de uma requisição os mesmos continuassem transparentes para a interface.

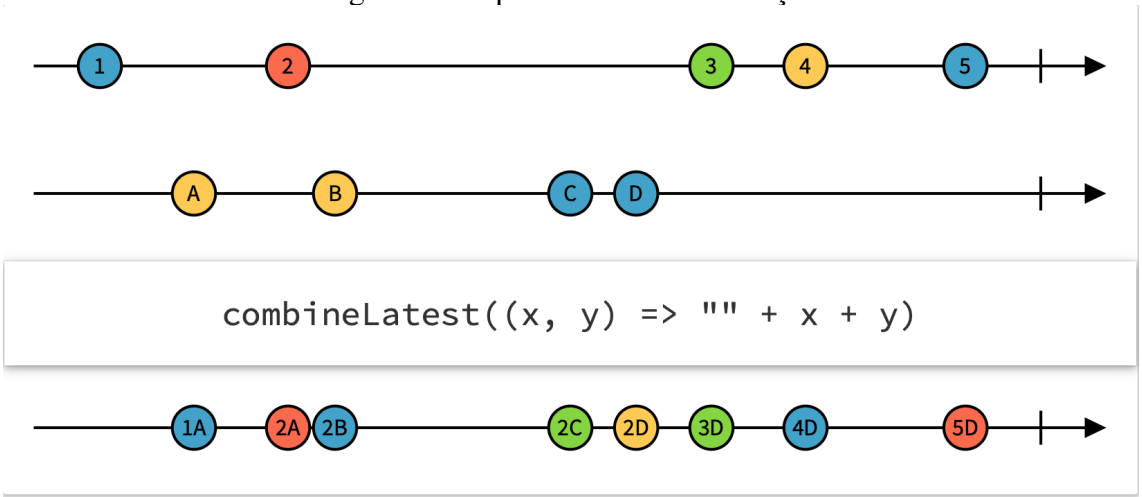
Figura 4.6: Operadores de condição



Fonte: https://rxmarbles.com

- Operadores de combinação (Figura 4.7): são operadores que fazem uso de uma ou mais cadeias de eventos, combinando-as de diversas formas. Foram utilizados para agregar inputs do usuário, como entradas de e-mail e senha durante o login, e informações dos alunos durante a edição e criação do aluno.

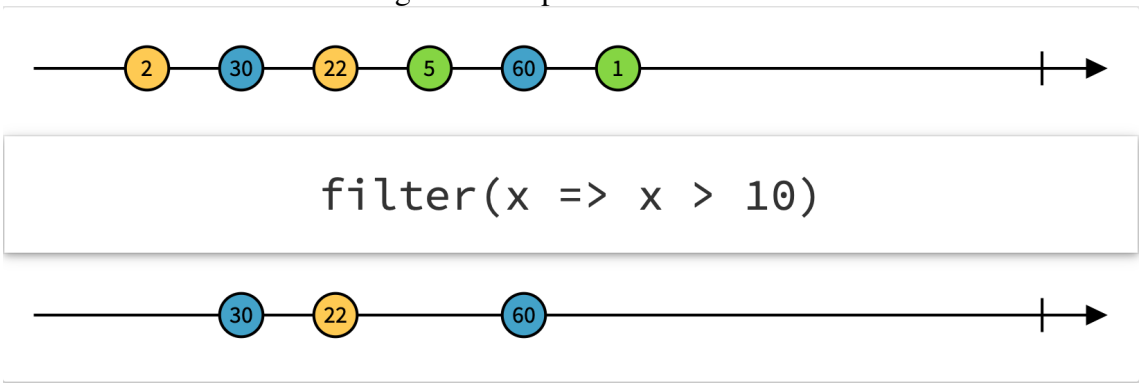
Figura 4.7: Operadores de combinação



Fonte: https://rxmarbles.com

- Operadores de filtro (Figura 4.8): são operadores que filtram eventos de acordo com certos critérios, podendo filtrar erros ou valores nulos. Foram utilizados para filtrar eventos que poderiam ser descartados previamente evitando um desperdício de computação, por exemplo filtrar eventos de senhas de comprimento menor que os seis dígitos requeridos pelo *Firebase*.
- Operadores de transformação (Figura 4.9): são operadores que modificam o valor de entrada, tais como *map* e *flatMap*. Foram utilizados para operarmos sobre os valores de forma sucinta, o uso mais representativo dessa classe de operadores é a transformação de parâmetros como e-mail e senha em uma requisição de autentica-

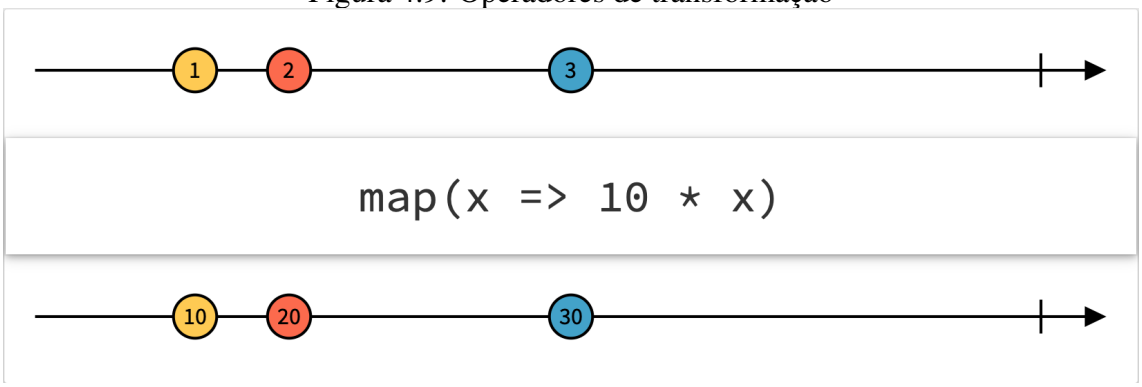
Figura 4.8: Operadores de filtro



Fonte: https://rxmarbles.com

ção.

Figura 4.9: Operadores de transformação



Fonte: https://rxmarbles.com

A seguir se vê um dos usos de RxSwift no projeto. O código chama o serviço de alunos do provedor para receber aqueles cadastrados para o professor “usuario”. Graças ao encadeamento de operadores, quando a informação dos alunos é recebida, ela é mapeada para uma estrutura conhecida da camada de apresentação, o *SectionModel*, para, por fim, ser vinculada a uma tabela a qual irá apresentar as informações.

```
provider
    .studentService
    .getSudents(for: provider.sessionService.user!.id)
    .map { SectionModel(items: $0) }
    .bind(to: tableData)
    .disposed(by: disposeBag)
```

4.3.4 Serviços

Para a realização de chamadas de rede para o servidor (chamadas de funcionalidades do sistema), foi criada uma série de serviços, os quais são reunidos por diferentes provedores através do uso de protocolos. Cada protocolo de provedor define um conjunto de serviços para cada fluxo, e o provedor da aplicação implementa todos os outros. Quando os protocolos são passados para cada fluxo individualmente é realizado um *upcasting* para limitar a visibilidade de cada fluxo ao seu próprio provedor de serviços.

```
1  protocol LoginProvider {
2      var loginService: LoginService { get }
3  }
4
5  protocol UploadProvider {
6      var sessionService: SessionService { get }
7      var uploadService: UploadService { get }
8      var studentService: StudentService { get }
9  }
10
11 protocol StudentProvider {
12     var sessionService: SessionService { get }
13     var studentService: StudentService { get }
14 }
15
16 class AppProvider: LoginProvider, UploadProvider,
     ↪ StudentProvider {
17
18     var sessionService: SessionService
19     var loginService: LoginService
20     var uploadService: UploadService
21     var studentService: StudentService
22     var pdfService: PDFService
23
24     init(window: UIWindow) {
25
```

```
26          FirebaseConfiguration.shared.setLoggerLevel(.error)
27          FirebaseApp.configure()
28
29          let database = DatabaseService()
30          let storage = StorageService()
31
32          sessionService = SessionService(database: database,
              ↪ window: window)
33          loginService = LoginService(database: database)
34          uploadService = UploadService(database: database,
              ↪ storage: storage, session: sessionService)
35          studentService = StudentService(database: database,
              ↪ storage: storage)
36          pdfService = PDFService(sessionService:
              ↪ sessionService, studentService: studentService)
37
38      }
39
40  }
```

4.3.4.1 Padrão Singleton ou injeção de dependências

Para conseguirmos utilizar os serviços de dentro dos fluxos do sistema, precisamos acessá-los de alguma forma e, já que o uso de variáveis globais é uma prática extremamente desencorajada, temos duas outras opções. O padrão Singleton define uma classe, no nosso caso o provedor da aplicação, que terá sempre apenas uma instância. Este padrão é o menos custoso de implementar, contudo acabamos nos prejudicando em outras áreas, uma vez que acabamos com um comportamento similar a uma variável global com um modo de acesso forçado.

A outra opção seria a injeção de dependências, onde os provedores são passados na criação de cada fluxo, o que normalmente causa um custo de implementação mais alto, porém, como já temos implementado o uso de coordenadores, podemos utilizá-los para este fim. Tendo em vista que foi utilizado o padrão *coordinator* que facilita a injeção de dependência, esse viés de utilização de serviços foi implementado na aplicação.

4.3.4.2 Cache

Para melhorar a experiência do usuário foi criada uma *cache* local, possibilitando a redução do custo de buscar imagens do servidor, fornecendo-as mais rapidamente. Sua implementação se deu usando o poder do padrão *Observer* com operadores do *RxSwift*. A sua implementação segue.

```
1     private func saveToCache(id: String, value: Data) {
2         self.cache[id] = value
3     }
4
5     private func loadFromCache(id: String) -> Single<Data>
        ↪ {
6         return Single.deferred {
7             if let data = self.cache[id] {
8                 return Single.just(data)
9             } else {
10                return Single.error(RxError.noElements)
11            }
12        }
13    }
```

A *cache* foi definida como um dicionário do tipo *Data* (tipo genérico para representação de *arrays de bytes*), indexado pelo caminho da imagem no servidor, armazenado como o tipo básico *String*, garantindo sua unicidade. Salvamos todas as buscas na *cache*, com uso do operador *do*{} para armazenar os resultados das requisições de arquivos.

4.3.5 Testing

Nesta parte serão descritas as práticas de testes empregadas no aplicativo *móvel*.

4.3.5.1 Snapshot testing

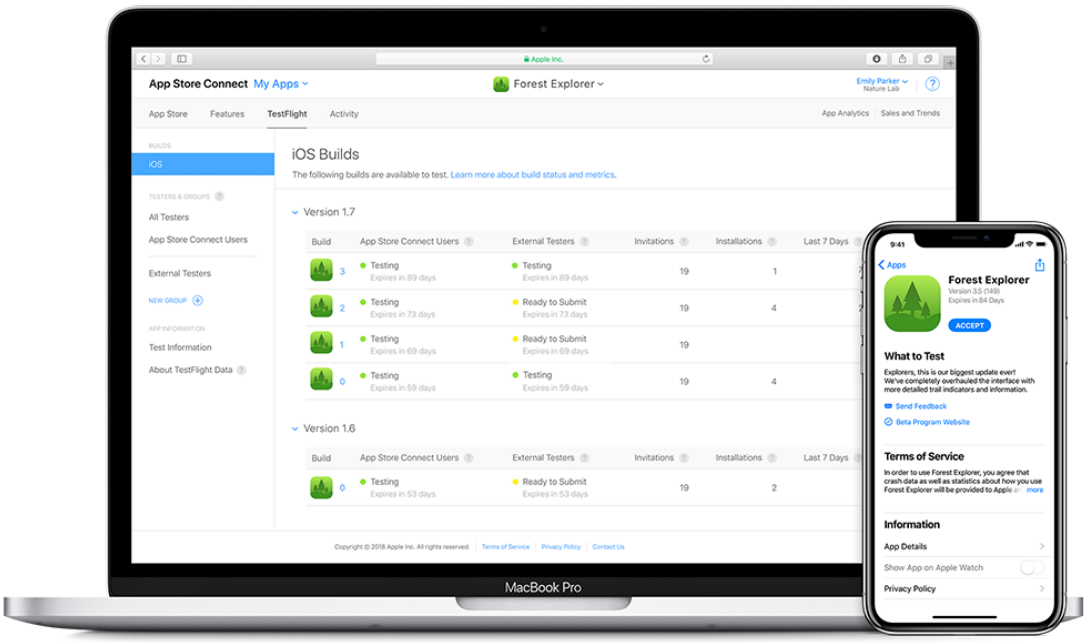
Snapshot testing é uma ferramenta de teste de *UI* que funciona a partir de uma comparação pixel a pixel de uma imagem salva da interface de usuário de uma versão estável com a versão que está sendo testada, o teste passa se as imagens forem idênticas.

Esse tipo de teste foi implementado na aplicação, porque é extremamente importante quando consideramos questões de experiência de usuário, já que ele detecta a mais leve alteração que pode ter ocorrido em alguma das interfaces da aplicação - o que é muito útil principalmente em projetos maiores, onde vários componentes são reusados.

4.3.5.2 Beta testing

Testes Beta são testes de Software conduzidos em uma ou mais versões do cliente pelo usuário final. Para realizarmos estes testes, tiramos proveito de uma ferramenta promovida pela *Apple* o *Testflight* (INC., 2019e). Com este recurso, ao compartilhar versões do cliente com testadores externos, podemos indicar quais funcionalidades devem ser testadas, permitindo-nos direcionar nossos testes, o que voltaremos a comentar no capítulo de Análise de Resultados. A plataforma também atualiza automaticamente as versões de testes para todos os clientes.

Figura 4.10: Painel de uso do Testflight



Fonte: Apple Inc. <https://developer.apple.com/testflight/images/testflight-overview-hero-large.png>.

4.4 Cliente web de visualização

Nesta seção serão descritas as tecnologias necessárias para a realização do cliente web e como elas foram utilizadas para atingir os requisitos.

4.4.1 Servidor

Um dos maiores benefícios da arquitetura do modelo cliente-servidor é a flexibilidade de tipos de clientes (KUROSE; ROSS, 2017). Esse atributo permite que o cliente Web seja integrado ao sistema sem nenhuma mudança necessária.

4.4.1.1 Garantia de segurança

Um aspecto importante do cliente web, assim como o do cliente *móvel*, é garantir a segurança e privacidade das informações. Neste quesito, o *Firebase* nos salva mais uma vez, já que a autenticação e as regras de segurança de acesso a arquivos e documentos continuam valendo no cliente web.

4.4.1.2 Hospedagem e domínio

A escolha feita pelo *Firebase* também nos traz, sem custo, dois aspectos vitais para aplicações web: hospedagem e domínio (*Firebase* oferece o serviço *Firebase Hosting*, que hospeda a página)

4.4.1.3 HTML

Hypertext Markup Language - HTML é uma linguagem de marcação multiplataforma que vem sendo usada pela *World Wide Web* desde 1990, devido à sua portabilidade e simplicidade. O HTML é usado para estruturar e dar significado para os diferentes componentes do cliente (BERNERS-LEE; CONNOLLY, 1995). Outras tecnologias que não o HTML são usadas para descrever tanto a apresentação (CSS) quanto funcionalidades e comportamentos (Javascript) dos elementos descritos pelo mesmo (MOZILLA, 2019a).

4.4.1.4 CSS

A fim de estilizar as interfaces do cliente web em prol da experiência do usuário, foi utilizada a tecnologia *Cascading Style Sheets*, o CSS, um linguagem declarativa de estilo que nos permite influenciar o desenho dos componentes de forma fácil e desacoplada da nossa linguagem estrutural, o HTML (MEYER, 2004).

Para a implementação final das interfaces também foi utilizada a biblioteca *Bootstrap* (BOOTSTRAP, 2019), que nos traz uma gama de componentes já estilizados para facilitar a prática da experiência desejada.

4.4.1.5 Javascript

JavaScript, que não deve ser confundida com *Java*, é uma linguagem interpretada com vários aspectos que permitem o uso de diversos paradigmas, como funções de primeira classe. A linguagem *JavaScript* é especificada pelo padrão *ECMAScript* que é suportado por todos os navegadores modernos, adicionando funcionalidades e comportamentos a elementos de páginas web (MOZILLA, 2019b).

4.4.2 ReactJS

Para a implementação do cliente web foi utilizada a biblioteca de *JavaScript React*, a qual permite a criação de interfaces declarativas através da definição de componentes que podem ser compostos formando *UI* mais complexas. Popularizado pelo ditado "*Learn Once, Write Anywhere*": aprenda uma vez, escreva em qualquer lugar (FACEBOOK, 2019b).

React faz uso de uma extensão de sintaxe do *JavaScript* chamada *JSX*, capaz de facilitar o desenvolvimento e a estruturação das interfaces com componentes mais poderosos devido a estarem inseridos na linguagem (FACEBOOK, 2019a) como vemos no trecho a seguir.

```
1     const element = {
2         <div>
3             <h1> Sou uma constante! </h1>
4         </div>
5     }
```

Como *React* é feito para o desenvolvimento de *SPAs, Single Page Applications)*, aplicações dinâmicas web que se comportam como aplicações de área de trabalho em oposição ao uso de roteamentos por *URL*, também foi usada a biblioteca *ReactRouter* para tratar dos caminhamentos corretamente, garantindo a privacidade de rotas críticas (REACTTRAINING, 2019).

5 INTERFACE DE USUÁRIO

Neste capítulo serão apresentadas as interfaces de usuário desenvolvidas para os clientes web e móvel do sistema e os conceitos de design utilizados para a realização dos mesmos.

5.1 UX

O termo UX (*User Experience*), experiência de usuário, vem sendo vastamente utilizado no contexto de design de interfaces. UX é definido por Marc Hassenzahl (HASSENZAHL, 2008) como o sentimento momentâneo do usuário ao utilizar o produto. Segundo este autor, boa experiência de usuário é o resultado da satisfação dos desejos humanos de:

1. autonomia: o usuário deve se sentir confortável em tomar decisões no produto.
2. competência: o usuário deve se sentir competente ao operar o produto.
3. estímulo: o produto deve responder a ações do usuário, encorajando a interação.
4. relacionamento: similaridade de experiência com outros usuários.
5. popularidade: efeito positivo em grupos.

Prover uma boa experiencia de usuário foi uma das prioridades durante a construção das interfaces. Já que o sistema é de uso rotineiro pelo professor a experiência deve ser o mais fluida possível.

5.2 Diretrizes de design

Como referência para a criação do design da aplicação, uma vez que a plataforma escolhida foi a iOS, fizemos uso das diretrizes de interfaces de usuário providas pela *Apple*. As chamadas *Human Interface Guidelines* (INC., 2019b) estabelecem seis princípios de design que serão seguidos durante o desenvolvimento das interfaces:

1. Integridade estética: o quão bem a aparência da aplicação auxilia na sua função, isso pode significar uma aparência mais leve e um comportamento padrão para interfaces mais sérias.
2. Consistência: uso de elementos conhecidos pelos usuários, como ícones e textos do

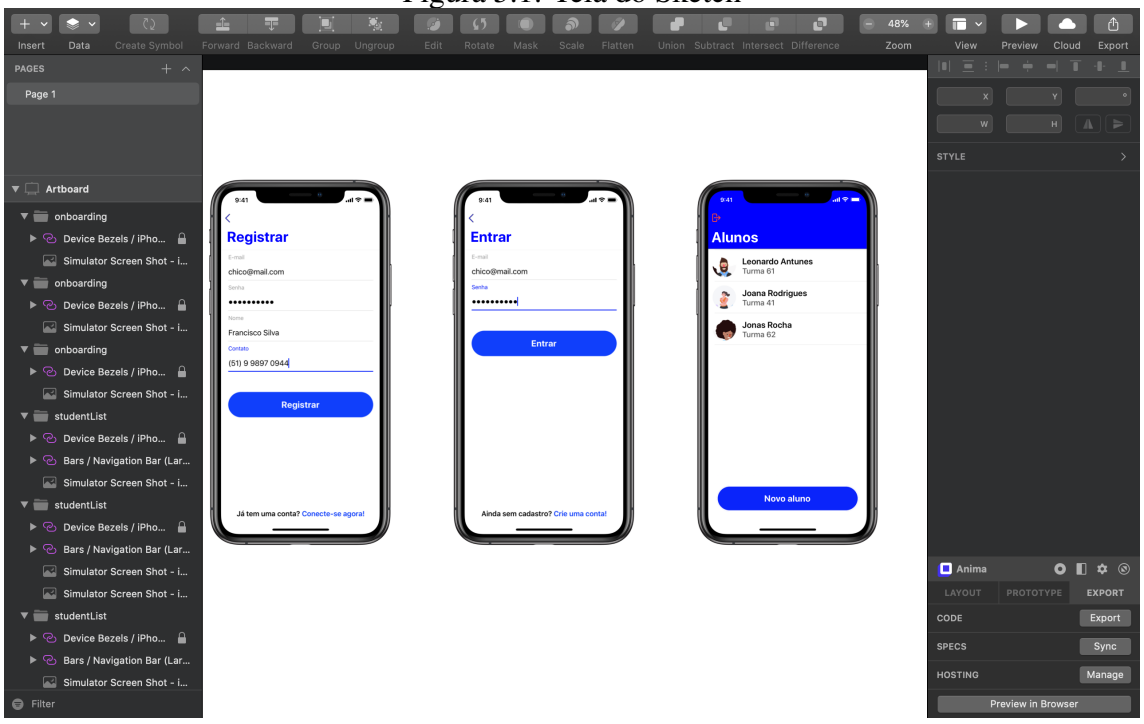
sistema, incorporando comportamentos já esperados.

3. Manipulação direta: o usuário deve poder manipular o conteúdo apresentado e usar gestos padrões de interação.
4. Resposta: ações do usuário devem ser respondidas pelo sistema de forma a confirmar a validade da ação do usuário. Elementos que contêm interações devem ter indicadores que refletem essa possibilidade.
5. Metáforas: o comportamento do conteúdo da tela deve apresentar ações que se relacionem ao mundo real, como arrastar e deslizar conteúdos.
6. Controle do usuário: a aplicação não deve tomar conta das tomadas de decisões, mas dar ao usuário este poder.

5.3 Sketch

A ferramenta utilizada para a construção das interfaces foi a aplicação *Sketch* (B.V., 2019), uma aplicação para *MacOS*, para a criação de interfaces, construída sobre *Metal*, a API gráfica do *OSX* alternativa ao *DirectX* da *Microsoft*. *Sketch* foi usado para fazer todas as telas da interface e os logos. Um dos aspectos de grande ajuda que o *Sketch* oferece são bibliotecas de elementos de *UI* nativos.

Figura 5.1: Tela do Sketch



Fonte: Provido pelo autor.

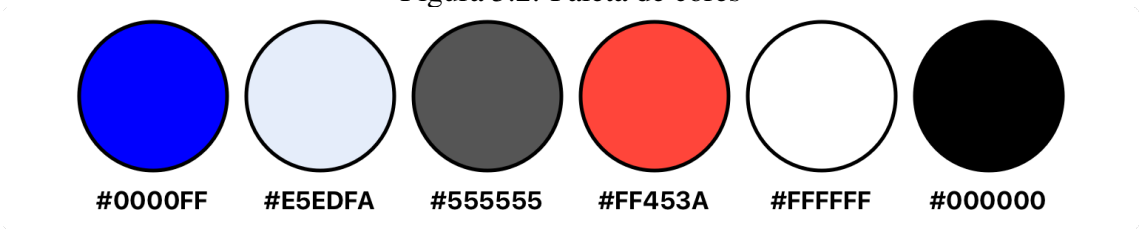
5.4 Interfaces do cliente móvel

Nesta seção será explicada a ideia por trás da interface do cliente, partindo da sua identidade visual e passando por cada interface desenvolvida com mais detalhes.

5.4.0.1 Cores

Devido à característica de grande densidade textual da aplicação, o uso de cores foi essencial para aumentar a legibilidade do texto. Para dar uma identidade reconhecível a ela, tons fortes foram usados nos elementos da interface, porém aplicados com parcimônia, respeitando o princípio de integridade estética.

Figura 5.2: Paleta de cores



Fonte: Provido pelo autor.

5.4.0.2 Logo

Para David Airey (AIREY, 2009) um produto sem um logo é análogo a um ser humano sem face. Partindo deste pressuposto, para a criação do logo, procurou-se trazer algo que imediatamente nos remetesse à funcionalidade da aplicação, desta forma optamos por explorar o nome do projeto, Sigma, utilizando a letra grega maiúscula de mesmo nome como identidade visual do projeto (ver figura 5.3). O uso deste símbolo, ademais, alude à ideia de somas, representação de um dos propósitos deste aplicativo, que é agregar informações - uma vez que o sigma representa somatórios em notação matemática.

Figura 5.3: Logo do sistema

Fonte: Provido pelo autor.

5.4.1 Navegação

A navegação entre interfaces se dá de duas maneiras: apresentações *push*, onde a nova interface vem da direita pra esquerda, tomando conta da janela; e apresentações modais, de baixo para cima, tomando conta parcial ou total da janela.

Apresentações modais são sucintas e autocontidas, elas têm o intuito de oferecer navegação a conteúdos que requerem uma ação explícita para retornar e não são necessariamente relacionadas ao conteúdo atual da interface - utilizamos este tipo de navegação para iniciar a composição de eventos e documentos.

Apresentações *push* nos ajudam a navegar pela hierarquia de dados e consistem em uma transição horizontal. Usamos este artefato para observar e alterar detalhes de estudantes a partir da lista de alunos.

5.4.2 Fluxo de Entrada/Cadastro

O ponto de entrada é a interface de apresentação (Figura 5.4), nela estão contidas a identidade visual do sistema e as opções de entrada no cliente móvel: "Entrar", para usuários já cadastrados, e "Cadastrar"para usuários novos. O principal propósito desta tela é trazer o usuário para dentro da aplicação sem bombardeá-lo com entradas de informação.

5.4.2.1 Preparo

O uso de interfaces de preparo (Figura 5.4) também está presente nas diretrizes de experiência de usuário. Esta tela é apresentada para o mesmo enquanto o aplicativo está sendo carregado pelo iOS e deve ser parecida com a interface que o usuário verá quando o cliente estiver pronto para exibição. Por isso, foi utilizado o logo da aplicação na mesma posição em que se encontra na tela de entrada.

Figura 5.4: Telas de apresentação (esquerda) e de preparo (direita)

Fonte: Provido pelo autor.

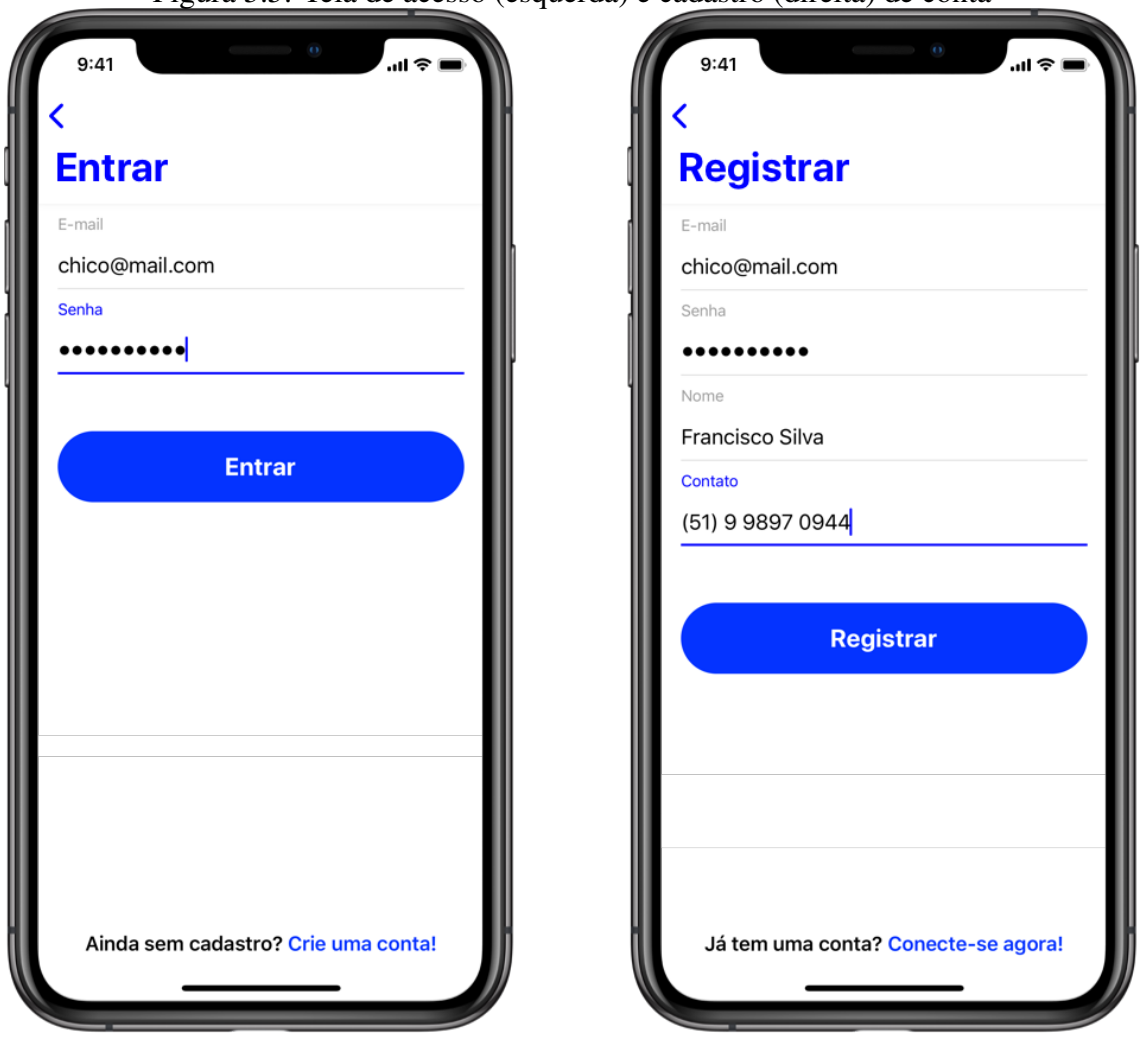
5.4.2.2 Cadastro

A interface de cadastro (Figura 5.5) contém as entradas de texto para as informações essenciais para o cadastro do professor: e-mail, nome, contato e senha. E, para a conveniência do usuário, um botão de navegação para a interface de acesso à conta.

5.4.2.3 Acesso a conta

A interface de acesso à conta (Figura 5.5) apresenta as mesmas características que a interface de cadastro, contudo exibe somente as entradas de texto relacionadas à autenticação do usuário.

Figura 5.5: Tela de acesso (esquerda) e cadastro (direita) de conta



Fonte: Provido pelo autor.

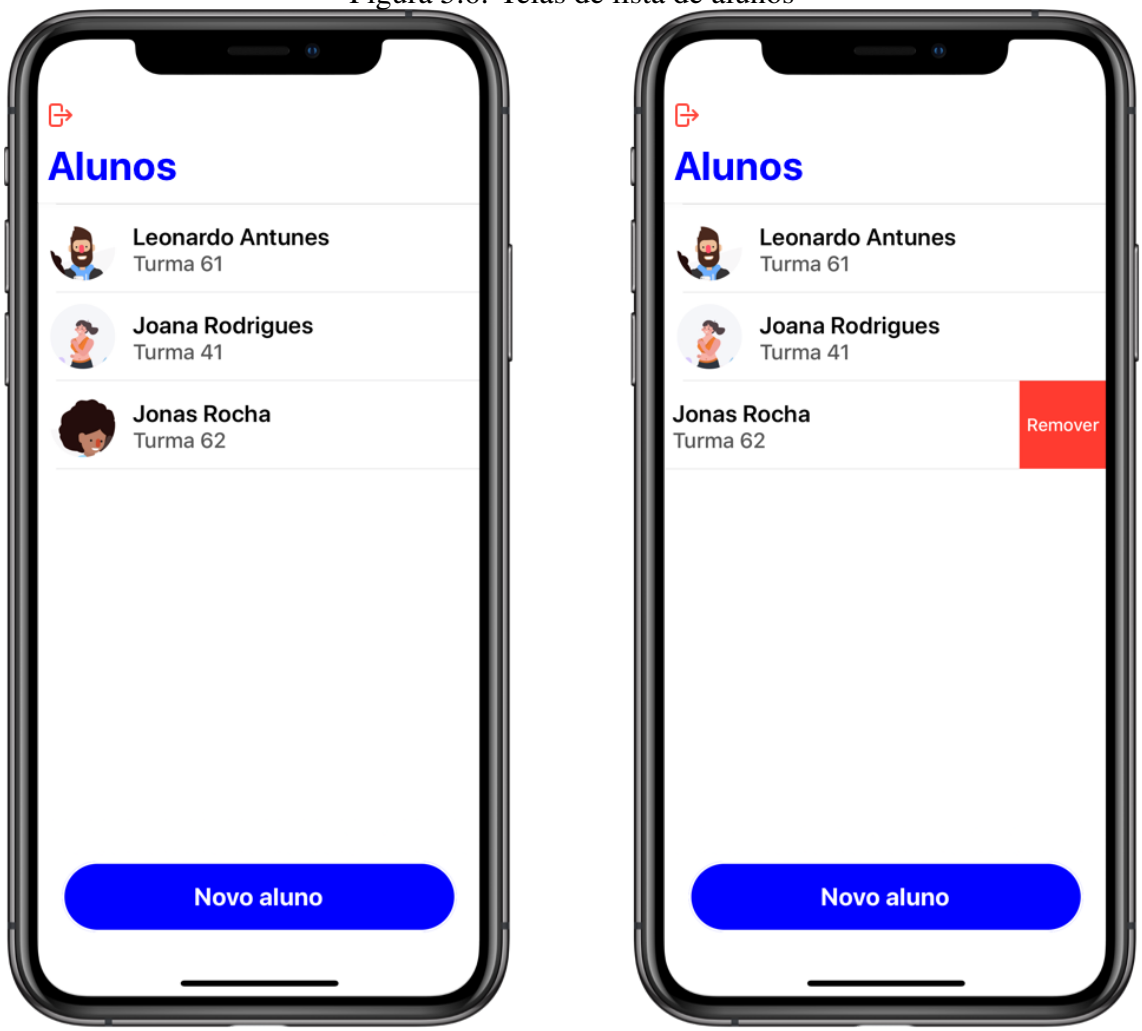
5.4.3 Fluxo principal

5.4.3.1 Lista de alunos

A tela de lista de estudantes (Figura 5.6) é a interface inicial à qual o professor é apresentado. Nela, o professor pode adicionar novos alunos com o uso do botão na parte inferior da tela, que apresenta de forma modal a interface de composição de aluno,

ou acessar detalhes de alunos específicos ao tocar na célula específica, apresentando a interface de detalhe do aluno com uma navegação de estilo *push*.

Figura 5.6: Telas de lista de alunos



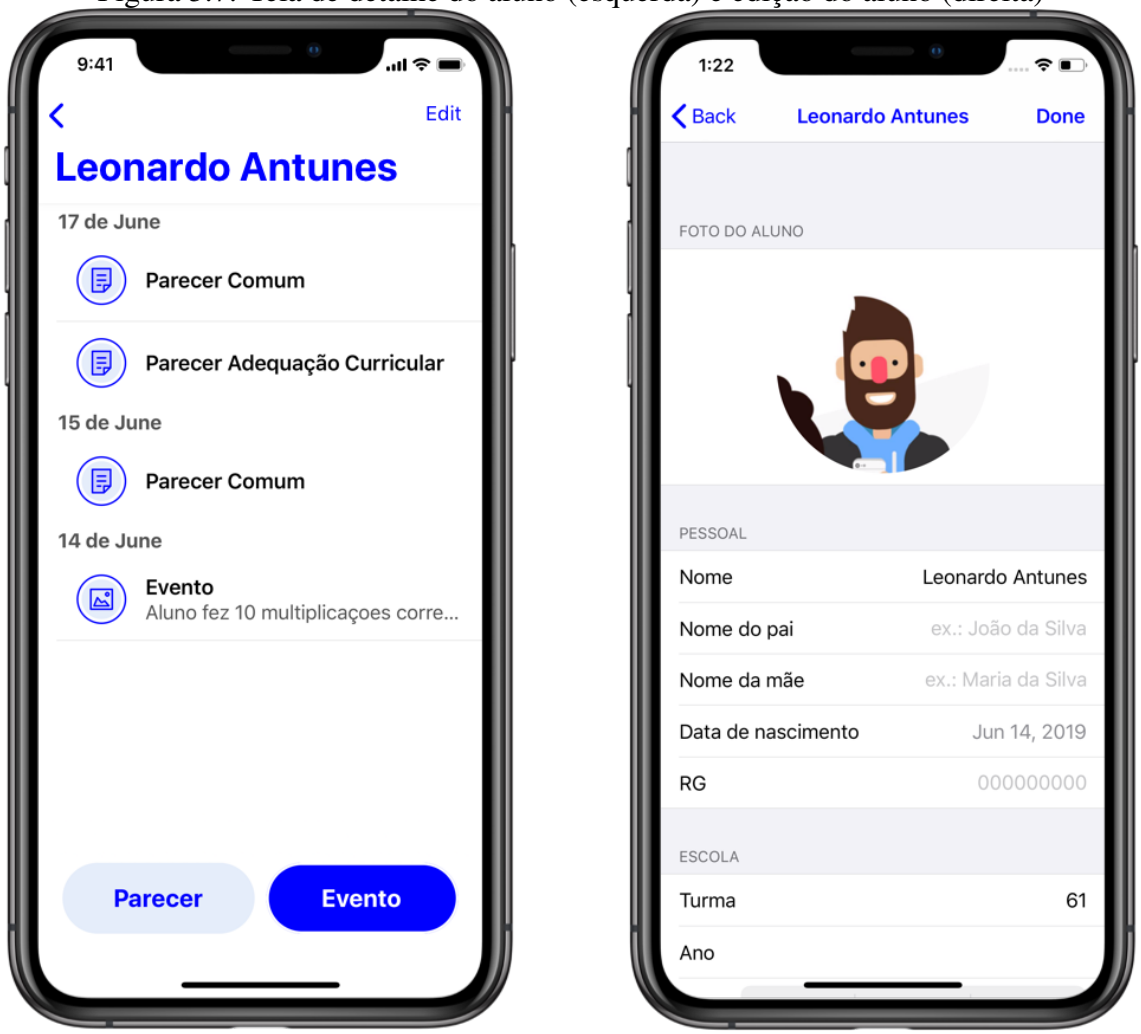
Fonte: Provido pelo autor.

5.4.3.2 Detalhe do aluno

A interface de detalhe do estudante (Figura 5.7) tem o propósito de agrupar todas as ações possíveis que o professor possa desejar realizar sobre o aluno, como visualizar eventos e pareceres prévios através da lista seccionada por dia, editar as informações do aluno a partir do botão "editar"no canto superior direito e criar novas instâncias de eventos ou pareceres com os botões na parte inferior da tela.

Os botões mantêm sua funcionalidade clara através do seu título exposto e são diferentes entre si, para não causar entradas indevidas.

Figura 5.7: Tela de detalhe do aluno (esquerda) e edição do aluno (direita)



Fonte: Provido pelo autor.

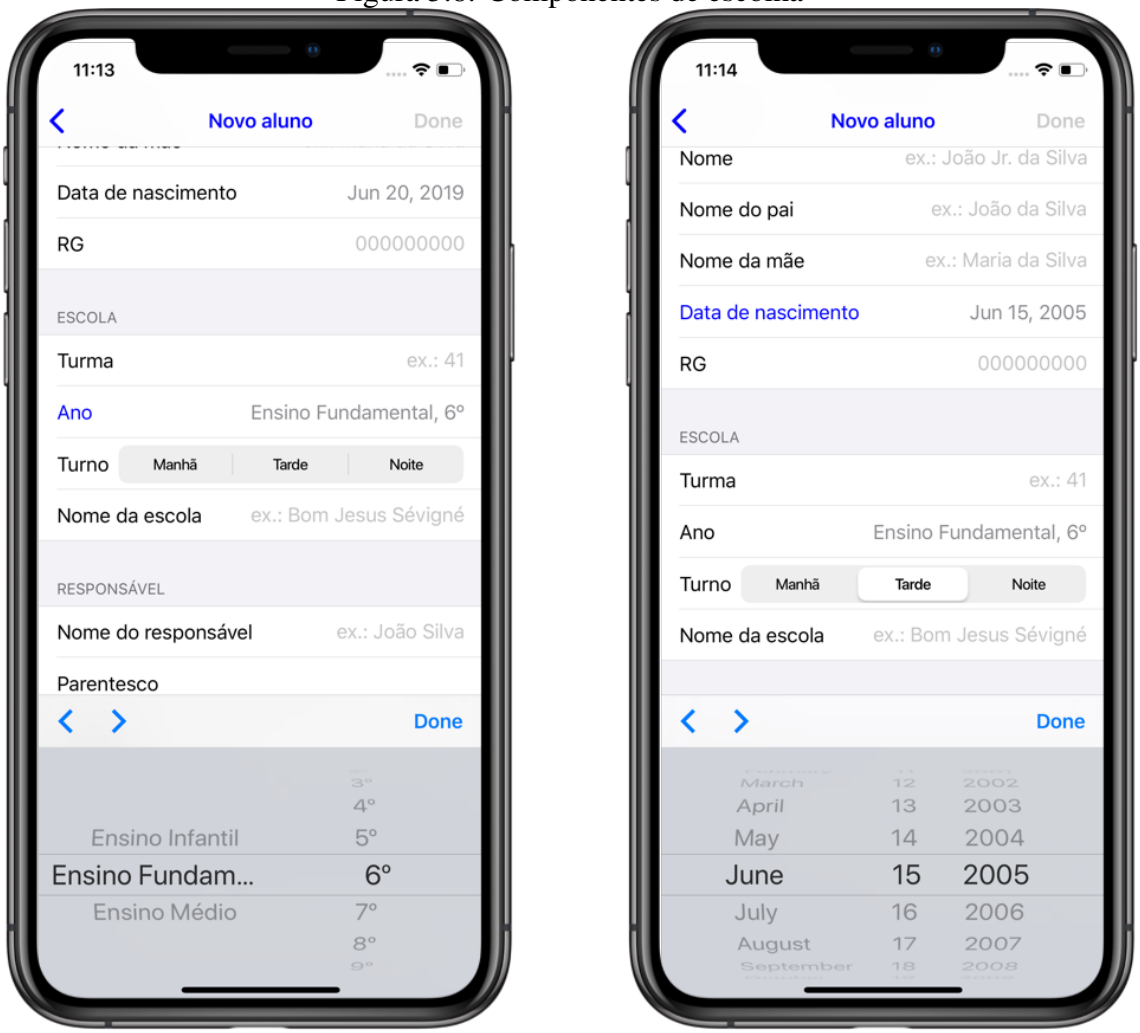
5.4.3.3 Composição de alunos

A tela de compor e editar estudantes (Figura 5.7) é um desafio para interfaces móveis, já que as informações do aluno são várias e não se consegue inferi-las com facilidade. Portanto, são necessárias muitas entradas por parte do professor usuário, então, para facilitar a visualização e preenchimento dos campos, foram usadas diversas técnicas de construção de interfaces, provendo uma experiência mais fluida (Figuras 5.8 e 5.9).

5.4.3.4 Composição de evento

A interface de composição de eventos (Figura 5.10) foi pensada para ser o menos intrusiva possível: exibe um comportamento de apresentação modal nativo, o nome e imagem do aluno que contextualiza o evento, um comentário opcional e uma mídia, que

Figura 5.8: Componentes de escolha



Fonte: Provido pelo autor.

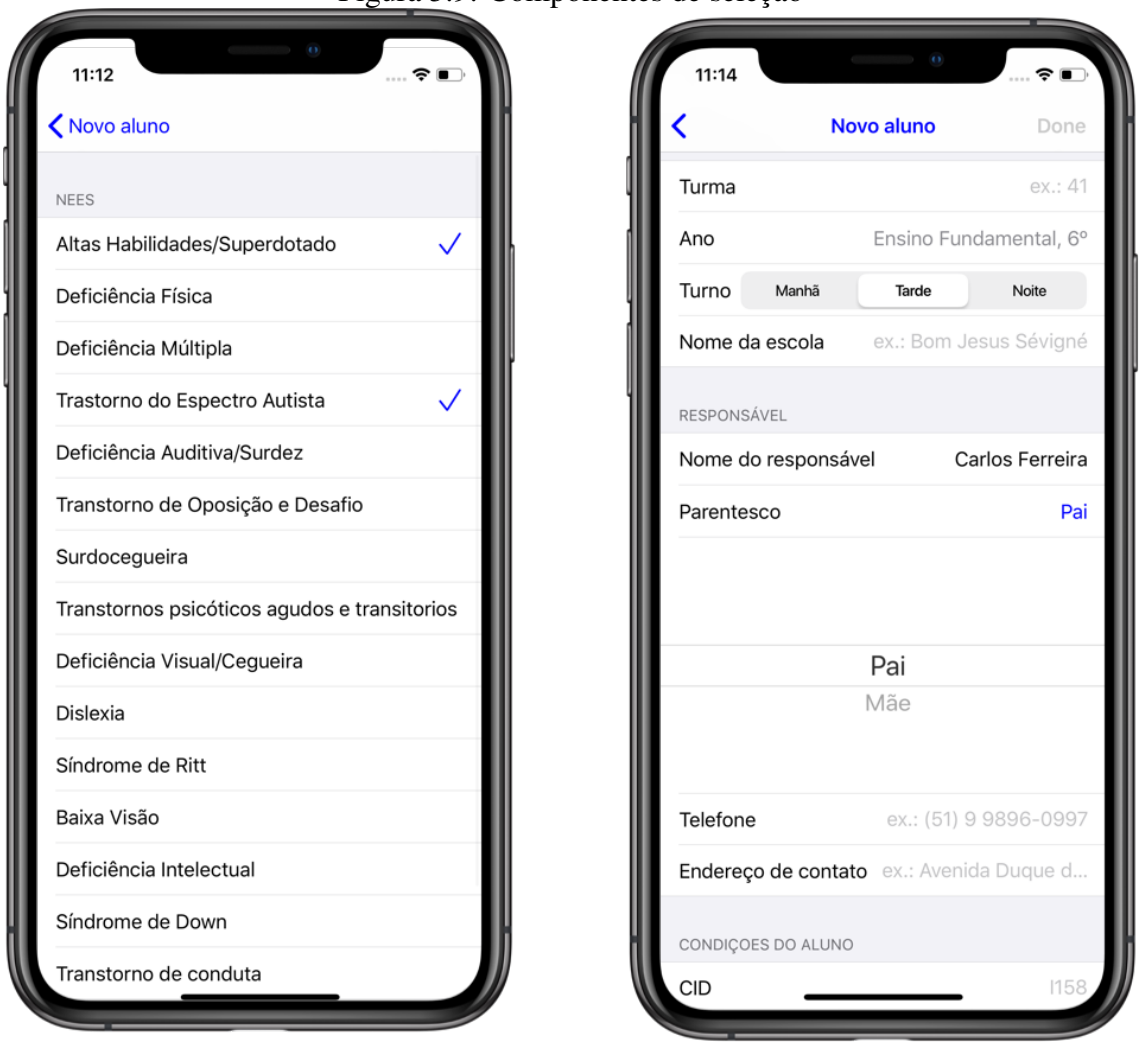
pode ser tanto áudio, vídeo, imagem ou texto, selecionada por uma interface de alerta modal.

5.4.3.5 Composição de parecer

A interface de composição de pareceres (Figura 5.10), apesar de muito similar, tem um comportamento mais dinâmico que a interface de composição de eventos vista previamente, pois existem múltiplos tipos de pareceres que requerem, por sua vez, diferentes tipos de entradas pelo usuário; para tanto, é usado um seletor de categoria de parecer que, quando atualizado, muda o conteúdo das próximas entradas.

Todas as entradas textuais de ambas interfaces de composição podem conter referências a outros eventos. A fim de facilitar este movimento, é usada uma interface acessória de entradas, em cima do teclado, para facilitar ao professor o estabelecimento

Figura 5.9: Componentes de seleção



Fonte: Provido pelo autor.

de relações entre eventos.

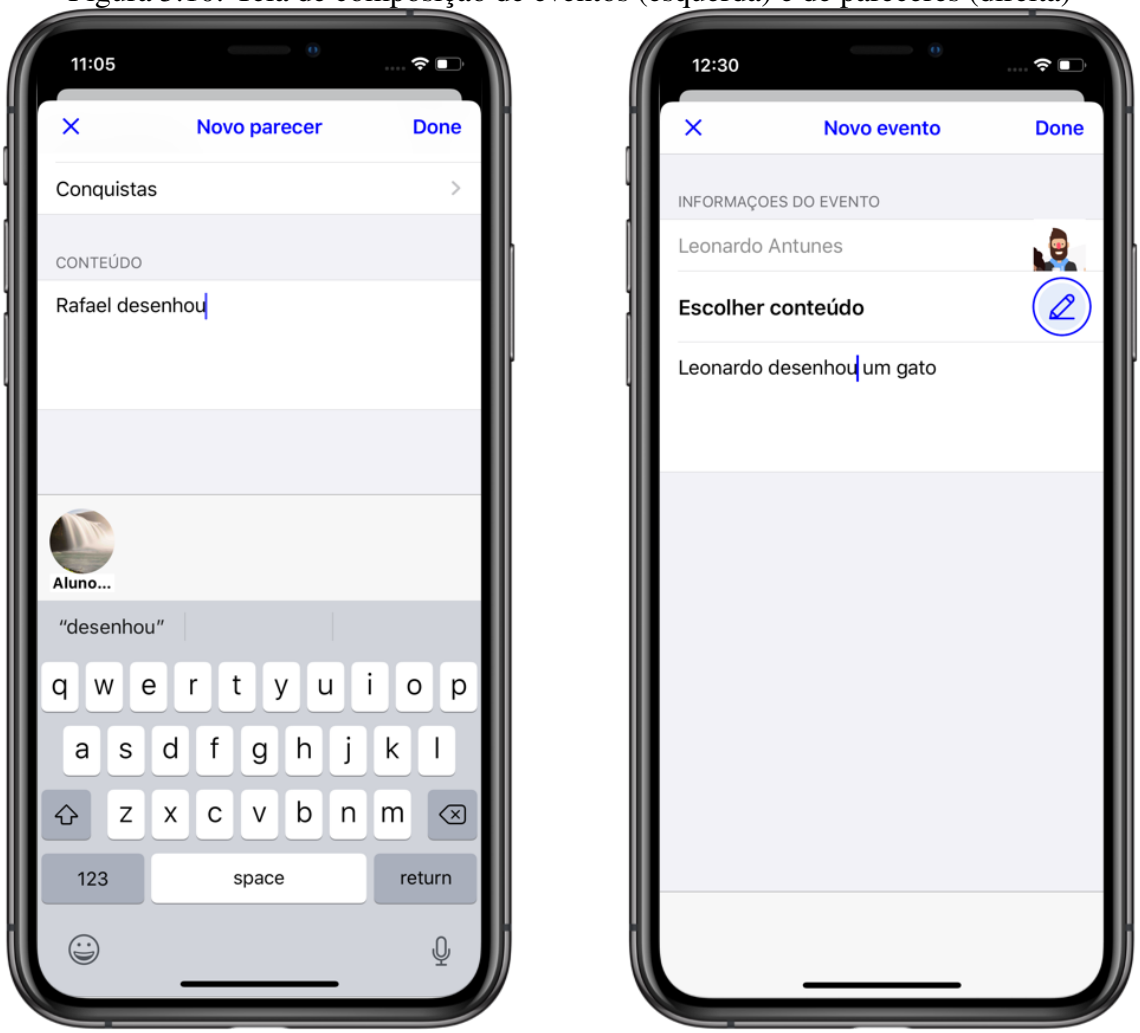
5.4.3.6 Detalhe do evento

Ao selecionar qualquer evento, o usuário é apresentado à sua mídia através de uma transição modal. Caso a mídia seja um áudio ou um vídeo, ela começa a ser executada imediatamente.

5.4.3.7 Detalhe do parecer

Na seleção de parecer é utilizada uma interface nativa chamada *Activity Controller* (Figura 5.13) que fornece ao usuário opções para exportar e compartilhar o mesmo em formato PDF.

Figura 5.10: Tela de composição de eventos (esquerda) e de pareceres (direita)



Fonte: Provido pelo autor.

Modelos no formato PDF foram pensados para possibilitar a exportação dos documentos criados pelo usuário de forma amigável. Vemos o exemplo do modelo feito para pareceres na figura 5.13.

5.5 Interfaces do cliente web

Nesta seção serão apresentadas as interfaces finais para o cliente web.

5.5.1 Autenticação do usuário

Esta interface é extremamente básica (Figura 5.14), e deve ser passada com facilidade pelo usuário. Para isso, foram utilizados componentes que o mesmo já deve estar

Figura 5.11: Componentes de texto e imagem

Fonte: Provido pelo autor.

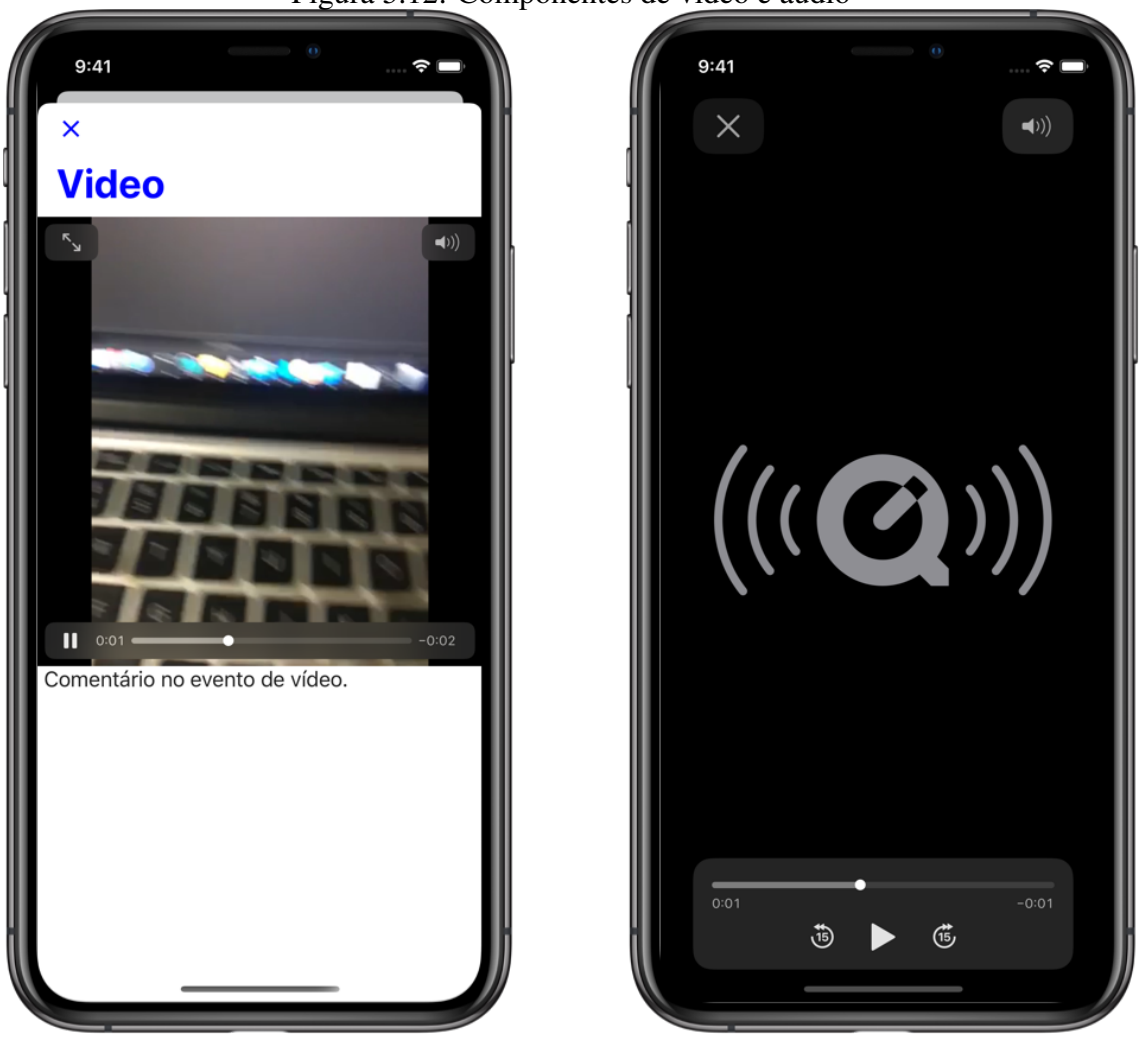
acostumado a usar: entradas de texto para login e senha agrupados no centro e um botão de confirmação.

Em caso de erro, o mesmo é apresentado na forma de um alerta padrão do navegador, além disso as mensagens de erros são tratadas para não revelar informações adicionais a um possível invasor.

5.5.2 Visualização do evento

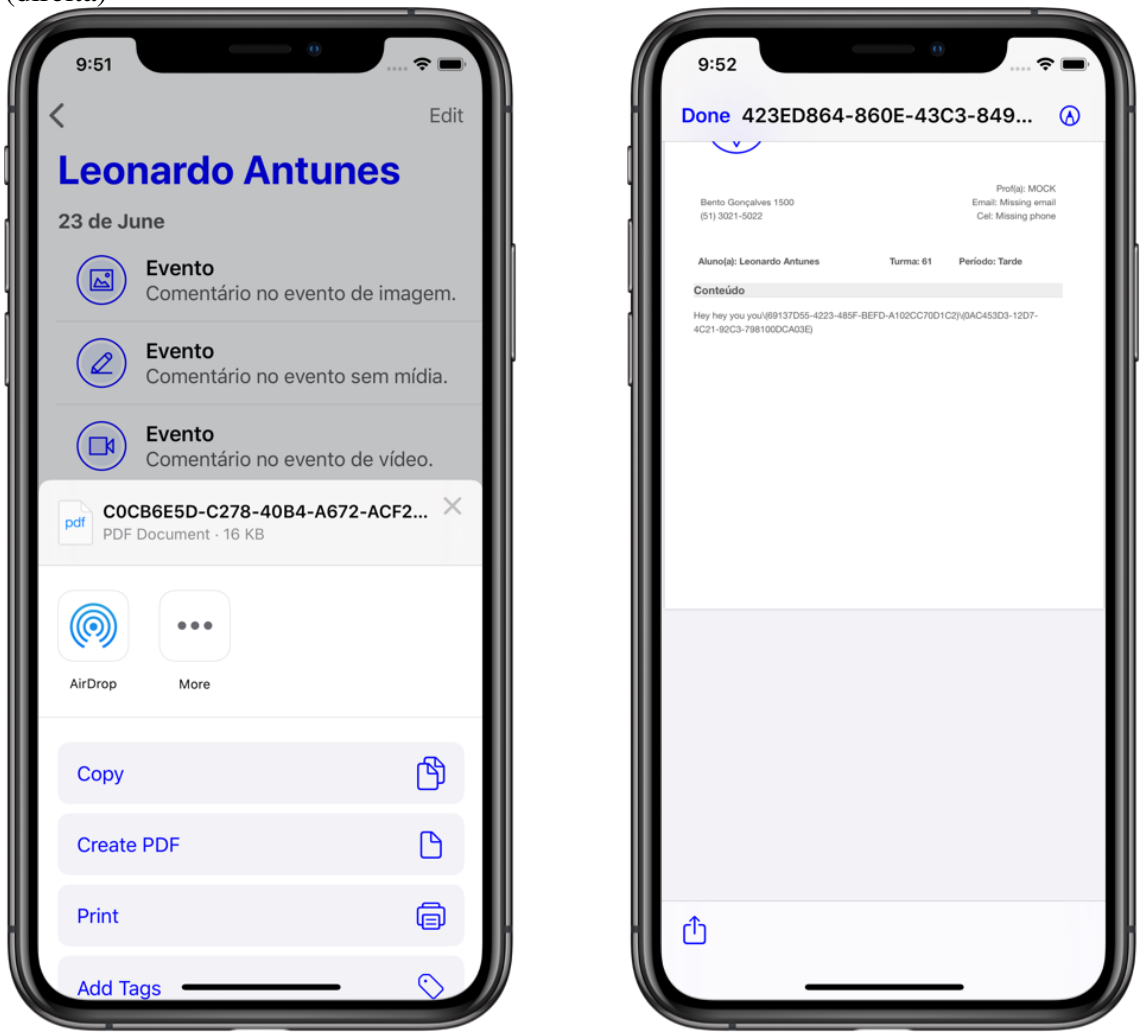
Para a visualização do evento temos uma tela padrão com suas informações principais e uma sessão central que contém a parte de mídia do evento, ela varia sua apresentação dependendo do tipo de mídia sendo descrita (Figuras 5.15 e 5.16).

Figura 5.12: Componentes de video e áudio



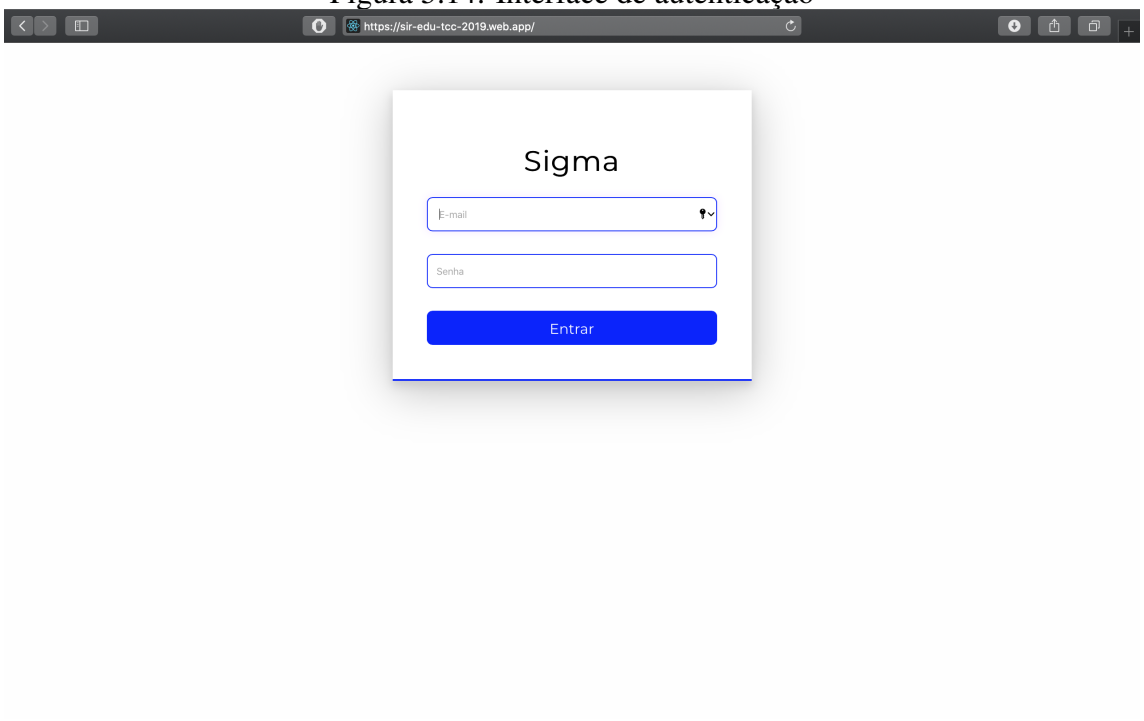
Fonte: Provido pelo autor.

Figura 5.13: Controlador de atividades (esquerda) e documento exportado como PDF (direita)



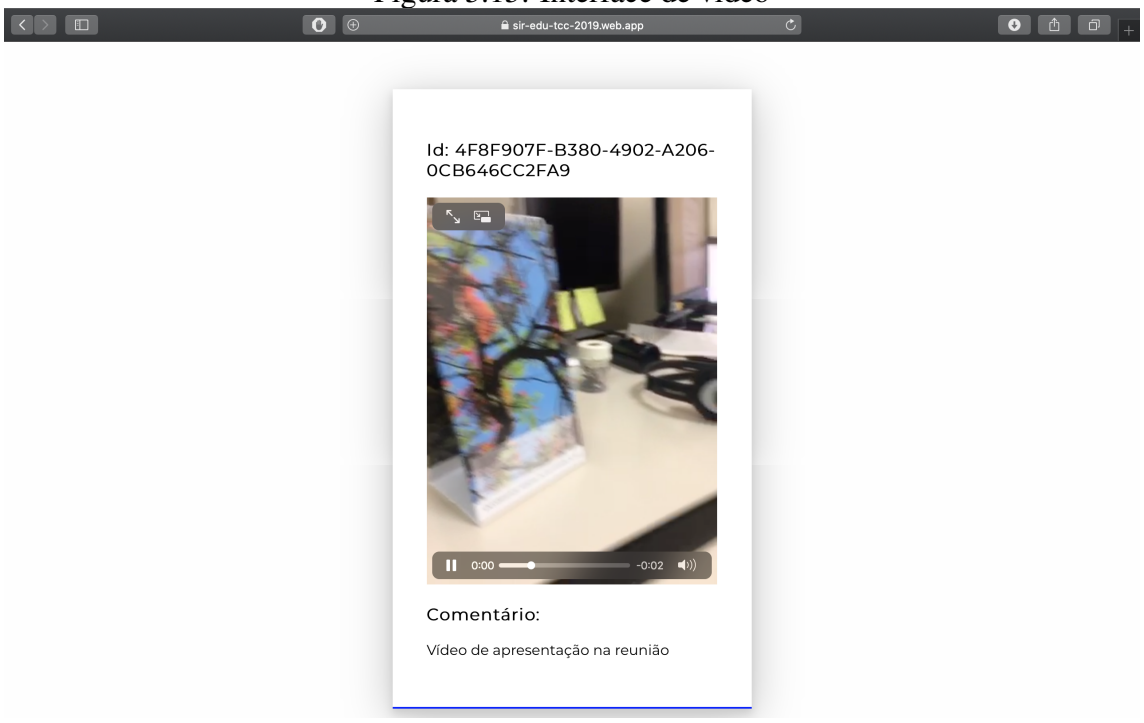
Fonte: Provido pelo autor.

Figura 5.14: Interface de autenticação



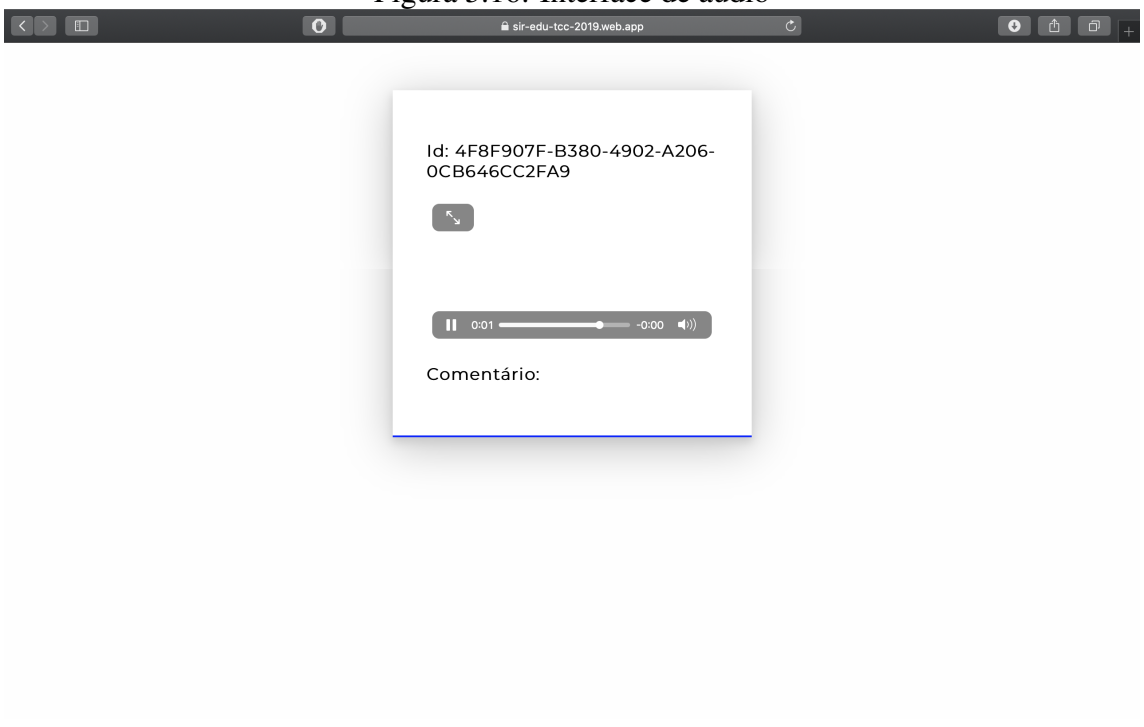
Fonte: Provido pelo autor.

Figura 5.15: Interface de video



Fonte: Provido pelo autor.

Figura 5.16: Interface de áudio



Fonte: Provido pelo autor.

6 AVALIAÇÃO DOS RESULTADOS

Neste capítulo avaliaremos a usabilidade do sistema. O método utilizado para esta avaliação foi o *SUS* (BROOKE et al., 1996), ou *System Usability Scale*.

Os testes foram feitos com seis usuários e, para a sua execução, cada um deles cumpriu uma lista de tarefas, a qual foi composta com base nas histórias de usuário mais relevantes no *backlog* do produto. Após a realização dessas tarefas, cada usuário respondeu a 10 questões da SUS.

As tarefas propostas para a execução do teste foram:

1. Cadastrar-se/entrar no cliente móvel.
2. Criar um aluno com foto.
3. Criar um evento com mídia (áudio/vídeo/imagem).
4. Criar um parecer com referência a um evento.
5. Visualizar o evento dentro da aplicação.
6. Exportar o evento (deslizar para a direita na tela de listagem dos eventos) como URL.
7. Visualizar o evento exportado no cliente web.
8. Exportar documento de parecer como PDF.

Dentre os usuários que testaram o sistema, cinco fazem uso regular de dispositivos iOS e um participa ativamente do ambiente de salas de recursos.

A Tabela 6.1 contém as respostas do questionário *SUS* relativo aos testes realizados com os usuários, tendo suas respostas simplificadas em: concordo, neutro e discordo.

Como o questionário SUS trabalha com questões explorando tanto concordância como discordância, a tabela 6.1 pode não mostrar de modo fidedigno a performance do sistema. Para aclarar, é possível ver na figura 6.1 o resultado positivo ou negativo para cada questão.

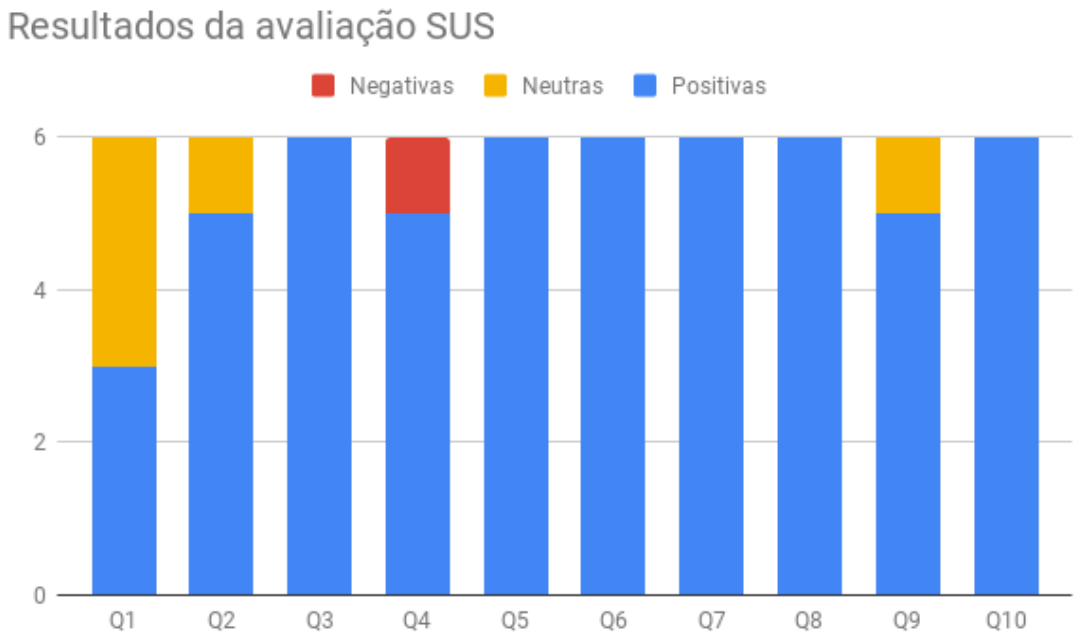
Como os resultados mostram, a avaliação do sistema apresentou uma nota média na escala *SUS* altamente satisfatória (BROOKE et al., 1996), com 88,3 pontos de média entre os 6 testes realizados, o que colocaria a usabilidade do sistema em um conceito equivalente a “A”.

A questão com maior discrepância nas respostas foi a questão 1 (Eu acho que gostaria de usar o sistema frequentemente), na qual metade dos respondentes se posicionaram de forma neutra. Isso pode ser analisado do ponto de vista de que uma parte significativa

Tabela 6.1: Respostas do questionário SUS

Questão	Afirmação	Concorda	Neutro	Discorda
Q1	Eu acho que gostaria de usar o sistema frequentemente.	50%	50%	0%
Q2	Achei o sistema desnecessariamente complexo.	0%	16,6%	83,3%
Q3	Achei o sistema fácil de usar.	100%	0%	0%
Q4	Eu acho que precisaria apoio de um técnico para poder usar esse sistema.	16,6%	0%	83,3%
Q5	Achei que as várias funcionalidades do sistema estão bem integradas.	100%	0%	0%
Q6	Encontrei muitas inconsistências no sistema.	0%	0%	100%
Q7	Imagino que grande parte das pessoas aprenderia a usar esse sistema rapidamente.	100%	0%	0%
Q8	Achei o sistema bastante complicado.	0%	0%	100%
Q9	Me senti bastante confiante ao usar o sistema.	83,3%	16,6%	0%
Q10	Precisei aprender muitas coisas antes de usar o sistema.	0%	0%	100%

Figura 6.1: Resultados da avaliação SUS



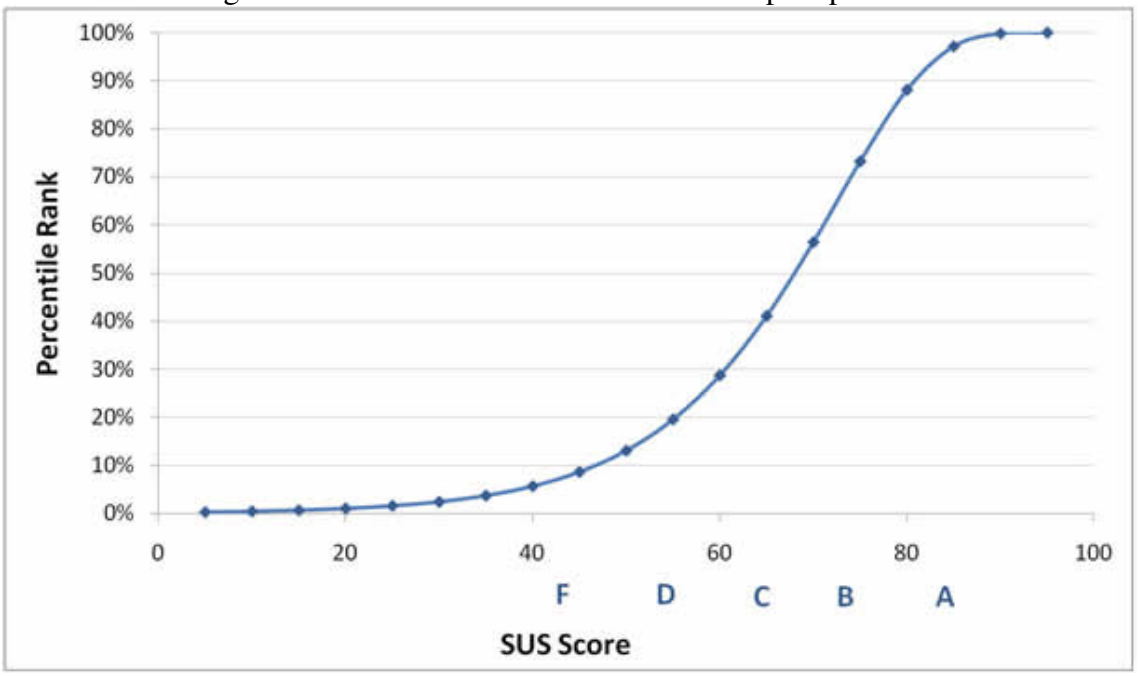
Fonte: Elaborada pelo autor

dos avaliadores não trabalha na área de atendimento a alunos com necessidades educacionais especiais. Sendo assim, alguns dos comportamentos do sistema podem não ser claros o suficiente para todos os respondentes, uma vez que não conseguem extrair benefícios

do uso da aplicação.

Além disso, nenhum dos respondentes achou o sistema desnecessariamente complexo, e todos os respondentes acharam que as funcionalidades da aplicação ficaram bem integradas.

Figura 6.2: Curva de conversão - score SUS para percentil



Fonte: Jeff Sauro - uxpamagazine.org/sustified

Analisando a curva de *ranking* de resultados da escala *SUS*, mostrada na Figura 6.2, e usando como base os pontos levantados por Jeff Sauro (SAURO, 2011), conclui-se que o resultado médio fica no percentil 88,3, ou seja, a aplicação conseguiu uma pontuação maior do que 80,3% das aplicações testadas, e usando um sistema de notas baseado em letras, sua nota seria equivalente a “A”.

7 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho se propôs a resolver o problema de falta de integração entre os diversos mecanismos que professores de salas de recursos utilizam para acompanhar os seus alunos, fornecendo uma solução rápida, segura e padronizada. Uma aplicação móvel que, através do dispositivo do professor, centraliza os diversos processos do seu fluxo de trabalho num único dispositivo, de maneira segura e intuitiva, foi a resolução encontrada.

Nos capítulos anteriores, apresentamos as técnicas de gerenciamento de projeto e engenharia de software utilizadas para completar o objetivo do trabalho, realizando a análise crítica das ferramentas usadas para atingir o mesmo. Com o uso destas, foi possível a criação deste sistema com padrões altos de segurança, disponibilidade e desempenho a um custo quase desprezível e com a mão de obra de um desenvolvedor.

O resultado obtido, tanto pelo desenvolvimento do projeto quanto na avaliação SUS do sistema, atendeu todos os requisitos estabelecidos nas histórias do usuário e atingiu uma avaliação de usabilidade de 88,3, resultando numa aplicação que pode ser entregue para uso nas salas de recursos e abrindo espaço para a expansão do sistema por trabalhos futuros.

7.1 Dificuldades

Nesta seção serão discutidos os pontos que geraram mais dificuldades no processo de desenvolvimento deste trabalho.

O design da interface foi um aspecto de grande importância no desenvolvimento da aplicação, pois está diretamente associado a dois critérios muito importantes: a facilidade de aprendizado do uso da aplicação e a rapidez de realizar a atividade principal, criar eventos e a fluidez do processo de preencher os diversos campos das entidades sendo criadas.

Durante a elaboração do sistema tivemos quatro iterações completas da aplicação móvel, que causaram mudanças na identidade visual e experiência de usuário provida, diminuindo consideravelmente a complexidade do sistema partir de cada uma.

A prática da criação de eventos apresentou problemas por tentar promover experiências similares no carregamento de diferentes tipos de mídias (principalmente para gravação de áudios, já que não se tem um componente nativo com essa função).

A variedade e complexidade dos pareceres foi uma grande complicação, pois em

uma linguagem fortemente tipada como *Swift* existe a necessidade de definir esse comportamento formalmente e, mais difícil ainda, representá-lo na interface de forma modular, com a finalidade de manter a coesão da tela de usuário sem quebrar a experiência intendida.

Embora tenhamos ganhado diversos aspectos de segurança já pertencentes às tecnologias utilizadas, como comunicação dos dados com HTTPS e bancos de dados criptografados, ainda se fizeram necessárias outras medidas para garantir a segurança das informações, como a configuração de regras de acesso ao banco de dados, limitando as consultas dos usuários ao banco para que mesmo autenticados os mesmos somente tenham privilégios de acesso aos seus alunos.

7.2 Continuidade

O trabalho, apesar de ter seu objetivo atingido, deixou ainda múltiplos aspectos abertos para expansão que poderiam ser melhorados, especialmente a funcionalidade de compartilhar eventos com pais e responsáveis, com o uso de uma autenticação provisória revogável para acessá-las, ou até mesmo uma aplicação inteira para o responsável monitorar o desenvolvimento do aluno.

Uma versão Android da aplicação para facilitar o acesso da aplicação a mais usuários.

A implementação de novos tipos de pareceres, para estender as funcionalidades da aplicação em conjunto com seus casos de uso.

A implementação de um modelo de classificação sobre os dados tanto para eventos, ajudando professores a fazer uma marcação mais significante nas entradas para cada aluno quanto para os próprios alunos, mapeando seu desempenho e dando dicas e estimativas sobre seu progresso. No caso do mapeamento dos alunos, diversas questões éticas sobre o uso dessas técnicas estão envolvidas.

REFERÊNCIAS

JEFF Sauro. 2019. <https://cloud.google.com/security/encryption-at-rest/#googles_default_encryption>. Accessed: 2019-05-15.

AIREY, D. **Logo design love: A guide to creating iconic brand identities**. [S.l.]: New Riders, 2009.

APPLE. **timeIntervalSinceReferenceDate:**. 2019. <https://developer.apple.com/documentation/foundation/date/1779647-init>. [Online: Acessado 15 de Junho de 2019].

AWS. **AWS Serverless**. 2019. <https://aws.amazon.com/serverless>. [Online: Acessado 15 de Junho de 2019].

BALDINI, I. et al. Serverless computing: Current trends and open problems. In: ____. **Research Advances in Cloud Computing**. Singapore: Springer Singapore, 2017. p. 1–20. ISBN 978-981-10-5026-8. Available from Internet: <https://doi.org/10.1007/978-981-10-5026-8_1>.

BECK MIKE BEEDLE, A. v. B. A. C. W. C. M. F. J. G. J. H. A. H. R. J. J. K. B. M. R. C. M. S. M. K. S. J. S. K.; THOMAS, D. **Manifesto for Agile Software Development**. 2001. <https://agilemanifesto.org/>. [Online: Acessado 15 de Junho de 2019].

BECK MIKE BEEDLE, A. v. B. A. C. W. C. M. F. J. G. J. H. A. H. R. J. J. K. B. M. R. C. M. S. M. K. S. J. S. K.; THOMAS, D. **Principles behind the Agile Manifesto**. 2001. <https://agilemanifesto.org/principles.html>. [Online: Acessado 15 de Junho de 2019].

BERNERS-LEE, T.; CONNOLLY, D. **Hypertext markup language-2.0**. [S.l.], 1995.

BOOTSTRAP. **Bootstrap**. 2019. <https://getbootstrap.com/>. [Online: Acessado 15 de Junho de 2019].

BRASILEIRO, G. **Diretrizes e bases da educação nacional.** 1996. <http://www.planalto.gov.br/ccivil_03/leis/L9394.htm>. [Online: Acessado 15 de Junho de 2019].

BROOKE, J. et al. Sus-a quick and dirty usability scale. **Usability evaluation in industry**, London–, v. 189, n. 194, p. 4–7, 1996.

B.V., B. **Sketch**. 2019. <https://www.sketch.com>. [Online: Acessado 15 de Junho de 2019].

CLOUD, G. **Firebase Products**. 2019. <https://firebase.google.com/products>. [Online: Acessado 15 de Junho de 2019].

CLOUD, G. **Supported Data Types**. 2019. <https://cloud.google.com/firestore/docs/concepts/data-types>. [Online: Acessado 15 de Junho de 2019].

COHN, M. **User Stories Applied: For Agile Software Development**. [S.l.]: Addison Wesley, 2004.

COOPER, K.; PETERS, T. **MVVM**. 2005. <https://github.com/jorgemht/Xamarin/wiki/MVVM-pattern>. [Online: Acessado 15 de Junho de 2019].

CORRAL, L.; SILLITTI, A.; SUCCI, G. Mobile multiplatform development: An experiment for performance analysis. **Procedia Computer Science**, Elsevier, v. 10, p. 736–743, 2012.

FACEBOOK. **Introducing JSX**. 2019. <https://reactjs.org/docs/introducing-jsx.html>. [Online: Acessado 15 de Junho de 2019].

FACEBOOK. **React**. 2019. <https://reactjs.org>. [Online: Acessado 15 de Junho de 2019].

FERNÁNDEZ-LÓPEZ, Á. et al. Mobile learning technology based on ios devices to support students with special education needs. **Computers & Education**, Elsevier, v. 61, p. 77–90, 2013.

FERREIRA, G. M. **Trabalho de Graduação: SIR-EDU: Sistema Integrado de Recursos Educacionais para a Gestão do Acompanhamento de Alunos com Necessidades Especiais**. 2017. <http://hdl.handle.net/10183/168934>.

GROSSMAN, J. **Advantages and disadvantages of M-V-VM**. 2009. <https://blogs.msdn.microsoft.com/johngossman/2006/03/04/advantages-and-disadvantages-of-m-v-vm/>. [Online: Acessado 15 de Junho de 2006].

HASSENZAHL, M. User experience (ux): towards an experiential perspective on product quality. In: **IHM**. [S.l.: s.n.], 2008. v. 8, p. 11–15.

INC., A. **About Swift**. 2019. <https://swift.org/about/>. [Online: Acessado 15 de Junho de 2019].

INC., A. **Human Interface Guidelines**. 2019. <https://developer.apple.com/design/human-interface-guidelines/>. [Online: Acessado 15 de Junho de 2019].

INC., A. **Optionals**. 2019. <https://developer.apple.com/documentation/swift/optional>. [Online: Acessado 15 de Junho de 2019].

INC., A. **Swift Language Guide**. 2019. <https://docs.swift.org/swift-book/LanguageGuide/TheBasics.html>. [Online: Acessado 15 de Junho de 2019].

INC., A. **Testflight**. 2019. <https://developer.apple.com/testflight/>. [Online: Acessado 15 de Junho de 2019].

JR., F. D. dos S. **Adequações curriculares em quatro matrizes**. 2018.

KUROSE, J. F.; ROSS, K. W. **Redes de computadores: uma abordagem top-down**. [S.l.]: Pearson, 2017. ISBN 978-85-430-1443-2.

LEACH, P.; MEALLING, M.; SALZ, R. Rfc 4122: A universally unique identifier (uuid) urn namespace. **Proposed Standard, July**, 2005.

LUCAS, E. A. **Trabalho de Graduação: Sistema de Gestão e Acompanhamento Educacional**. 2018. <http://hdl.handle.net/10183/190194>.

MEYER, E. **Cascading Style Sheets: The Definitive Guide**. O'Reilly, 2004. (Definitive Guide Series). ISBN 9780596005252. Available from Internet: <https://books.google.com.br/books?id=nU4EiJ9ZPf8C>.

MICROSOFT. **Azure Serverless**. 2019. <https://azure.microsoft.com/en-us/overview/serverless-computing/#explore-azure-serverless-apps>. [Online: Acessado 15 de Junho de 2019].

MOZILLA. **HTML: Hypertext Markup Language**. 2019. <https://developer.mozilla.org/en-US/docs/Web/HTML>. [Online: Acessado 15 de Junho de 2019].

MOZILLA. **Javascript**. 2019. <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript>. [Online: Acessado 15 de Junho de 2019].

MUNDEL, C. F. **Trabalho de Graduação: Modelagem do processo de atendimento em salas de recursos para alunos com necessidades educacionais especiais**. 2019.

NCES. **Children and Youth With Disabilities**. 2018. <https://nces.ed.gov/programs/coe/indicator_cgg.asp>.

ORGANIZATION, W. H. et al. International classification of diseases—ninth revision (icd-9). **Weekly Epidemiological Record= Relevé épidémiologique hebdomadaire**, v. 63, n. 45, p. 343–344, 1988.

PALMIERI, M.; SINGH, I.; CICCHETTI, A. Comparison of cross-platform mobile development tools. In: IEEE. **2012 16th International Conference on Intelligence in Next Generation Networks**. [S.l.], 2012. p. 179–186.

REACTIVEX. **RxSwift**. 2019. <https://github.com/ReactiveX/RxSwift>. [Online: Acessado 15 de Junho de 2019].

REACTTRAINING. **ReactRouter**. 2019. <https://reacttraining.com/react-router/>. [Online: Acessado 15 de Junho de 2019].

REENSKAUG, T.; COPLIEN, J. **The DCI Architecture: A New Vision of Object-Oriented Programming**. 2009. <https://www.artima.com/articles/dci_vision.html>.

RESCORLA, E. **Http over tls**. [S.l.], 2000.

SAURO, J. **SUStisfied? Little-Known System Usability Scale Facts.** 2011. <http://uxpamagazine.org/sustified>.

SOFTWARE, M. G. 2019. <https://www.mountaingoatsoftware.com/agile/scrum/scrum-tools/release-burndown>. [Online: Acessado 15 de Junho de 2019].

SOMMERVILLE, I. Software engineering 9th edition. **ISBN-10**, v. 137035152, 2011.

SUTHERLAND, J. 2017. <https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf>. [Online: Acessado 15 de Junho de 2019].

SUTHERLAND, J. 2019. <https://www.scrum.org/resources/what-scrumbut>. [Online: Acessado 15 de Junho de 2019].

TRELLO. **Trello Help - What is Trello?** 2017. <https://help.trello.com/article/708-what-is-trello>. [Online: Acessado 15 de Junho de 2019].