Beatriz Neves Porto
Gustavo Azevedo Correa
Rodrigo Tei Dalmas

Gamificação aplicada ao processo de aprendizado de teoria musical

São Paulo, SP

2023

Beatriz Neves Porto
Gustavo Azevedo Correa
Rodrigo Tei Dalmas

Gamificação aplicada ao processo de aprendizado de teoria musical

Trabalho de conclusão de curso apresentado ao Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro.

Universidade de São Paulo – USP

Escola Politécnica

Departamento de Engenharia de Computação e Sistemas Digitais (PCS)

Orientador: Prof. Dr. Ricardo Nakamura

São Paulo, SP
2023

Catalogação-na-publicação

Porto, Beatriz Neves
Gamificação aplicada ao processo de aprendizado de teoria musical / B. N. Porto, G. A. Correa, R. T. Dalmas -- São Paulo, 2023.
77 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.APLICATIVOS MÓVEIS 2.TEORIA MUSICAL 3.JOGOS I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t. III.Correa, Gustavo Azevedo IV.Dalmas, Rodrigo Tei

A beleza do aprendizado
é que ninguém pode tirá-lo de você - BB King

Agradecimentos

Gostaríamos de agradecer ao Prof. Dr. Ricardo Nakamura que nos orientou e guiou com muita atenção e dedicação ao longo da jornada de desenvolvimento desde projeto. Também agradecer aos entrevistados que cederam seu tempo e tornaram possível o processo de validação das etapas do projeto. Agradecer também à todos que dedicaram seu tempo para responder ao formulário, nos auxiliando com dados quantitativos. E enfim agradecer à todos que apoiaram diretamente e indiretamente a conclusão desde projeto.

Resumo

A partir da perspectiva de estudantes de música, aos quais compõem o grupo de integrantes deste projeto, é notada a necessidade de incentivos e estímulos que acompanhem a evolução tecnológica no que tange o aprendizado de teoria musical. É visto em grande expressão soluções que proporcionam estudos práticos, com exercícios focados em técnicas de instrumentos musicais, entretanto, pouco se vê a respeito de soluções para estudo e exercício de teoria musical.

Integrando metodologias como Design Thinking e Desenvolvimento Ágil, este projeto visa propor uma solução de estudo *gamificado* que seja coerente com o contexto tecnológico e social dos estudantes, de modo que o estudo seja contínuo.

Palavras-chave: Gamificação. Teoria musical. Design Thinking.

Abstract

From the perspective of music students, who constitute the group of participants in this project, there is a discernible need for incentives and stimuli that align with technological advancements in the realm of music theory learning. While practical solutions abound for instrumental technique-focused exercises, there is a noticeable scarcity of solutions catering to the study and practice of music theory.

Integrating methodologies such as Design Thinking and Agile Development, this project aims to propose a gamified learning solution that aligns with the technological and social context of students, ensuring a continuous and engaging study experience.

Keywords: Gamification, Music Theory, Design Thinking.

Lista de ilustrações

Sumário

1 Introdução

Ao longo da trajetória do aprendizado na área da música, é comum existir um engajamento maior na prática musical através de um instrumento do que no estudo teórico. Porém, esta falta de conhecimento na parte teórica pode limitar a compreensão da prática musical e também o acesso a algumas técnicas importantes, como composição e improvisação. A partir desta perspectiva cria-se a necessidade de compreender as razões do desinteresse nos fundamentos teóricos do estudo musical e solucionar este problema de modo que exista harmonia nos estudos.

Dado este estudo inicial, é necessário buscar soluções que atendam as necessidades dos alunos e professores. Deste modo, é visto que um trabalho em conjunto com o público é necessário para que os requisitos sejam bem estruturados e solucionados ao longo do projeto.

1.1 Objetivos

O objetivo deste trabalho é implementar uma solução tecnológica capaz de estimular o processo de aprendizado de teoria musical; levando em consideração os motivos que levam ao desinteresse na área e utilizando recursos de *gamificação* para aumentar o engajamento. O produto final visado é um MVP construído em conjunto com professores e alunos de música, focando em sanar a dificuldade do estudo de teoria musical observada pelos professores e alunos.

1.2 Organização do Trabalho

O trabalho foi dividido em 8 seções, sendo elas:

1. Introdução;
2. Aspectos Conceituais;
3. Método do Trabalho;
4. Primeiro Ciclo de Desenvolvimento;
5. Segundo Ciclo de Desenvolvimento;
6. Terceiro Ciclo de Desenvolvimento;
7. Implementação;

8. Considerações finais.

Na primeira seção do trabalho, o capítulo 1, são definidos os conceitos motivadores do trabalho e o contexto acadêmico e comercial que cerca o assunto. Tendo isso definido, os principais conceitos utilizados na idealização do projeto são definidos no capítulo 2. A partir daí, é definido o método de trabalho que será utilizado no decorrer do projeto, o que é explicado no capítulo 3.

O método escolhido pelo grupo funciona em ciclos de desenvolvimento e *feedback*, buscando ao máximo entender e tratar as dores do público alvo. Por esse motivo, foram documentados os três ciclos principais realizados nos capítulos 4, 5 e 6. Depois da finalização dessas etapas, têm-se os requisitos mais robustos, o que se utiliza para fazer efetivamente a implementação do projeto, detalhada no capítulo 7.

Por fim, consolida-se os resultados e aprendizados extraídos do trabalho no capítulo 8 explicando as expectativas que foram atingidas, as que não foram, as contribuições geradas pelo trabalho e pontos de melhora para o futuro.

2 Aspectos Conceituais

2.1 Gamificação

A partir da popularização de jogos digitais nota-se a inserção deste ramo tecnológico em outras áreas. Este fenômeno, batizado de *gamificação*, abrange as mais diversas áreas de atuação, e será explorado como este conceito pode ser aplicado ao processo de aprendizagem.

Segundo Johan Huizinga em seu trabalho *Homo Ludens* (HUIZINGA, 2005), vemos que o conceito de jogo explora ideias de recompensa, onde o próprio jogar é a maior recompensa do jogador. A *gamificação* explora este conceito, buscando recompensar um usuário através de interações deste com uma aplicação. Entretanto, como o ato de jogar não é o único objetivo, não é possível classificar um produto *gamificado* como jogo.

De forma análoga, é possível extrair diversos outros conceitos do universo dos jogos, como *feedback*, narrativa, competição, entre outros. A partir destes conceitos criou-se o fenômenos conhecido como *gamificação*, que visa atrair a atenção de usuários através destes conceitos para outras finalidade que não sejam jogar.

Assim, entendemos que o conceito de *gamificação* pode ser aplicado ao contexto de ensino, uma vez que a dinâmica de resolução de exercício pode fornecer ao usuário recompensas dadas por pontuações e punições, criando uma estrutura onde o usuário se vê querendo estudar cada vez mais para atingir metas e maiores pontuações.

3 Método do trabalho

Este projeto visa a entrega de eficácia em soluções para o usuário final, e para atingir este objetivo foi optado pela integração de uma adaptação da metodologia Design Thinking com Metodologia Ágil (Kanban e Scrum) ao projeto.

3.1 Design Thinking

A metodologia *Design Thinking* possui foco no usuário final e tem como objetivo criar soluções criativas através de um processo colaborativo, entendendo as pessoas e suas necessidades (ARAÚJO et al., 2019). Esta metodologia é cíclica, sendo realimentada por testes e *feedbacks* de usuário, trazendo uma constante implementação focada nas necessidades das pessoas. Como a metodologia se dá numa estratégia cíclica, os requisitos são voláteis, podendo ser alterados a cada iteração uma vez novas necessidades surgem, e necessidades anteriores são alteradas. Assim, o projeto se concentra em um estado de constante alteração e evolução.

As etapas de cada ciclo do Design Thinking serão discutidas nas próximas sessões.

3.1.1 Empatizar

A etapa *Empatizar* é onde o trabalho referente a entender o usuário é realizado. Observando o que e como as pessoas interagem dentro de um contexto, podemos entender o que estas pessoas pensam e sentem, e assim perceber o que elas precisam.

Como instrumento para esta etapa temos as entrevistas, que pode ser considerada como uma pesquisa qualitativa, e formulários de público geral, classificados como pesquisa quantitativa.

3.1.2 Definir

O objetivo desta etapa é definir um ponto de vista, que é uma afirmação significativa a respeito do problema, a visão explícita do problema. A partir das informações reunidas na etapa anterior, podemos definir quais padrões foram observados e definir qual tipo de pessoa o usuário do sistema será, entendendo assim quais serão as necessidades que iremos satisfazer.

3.1.3 Idealizar

Enquanto as etapas anteriores são focadas em observações e interpretações, a geração de ideias em si é concentrada nesta etapa. Com um ponto de vista do problema definido na etapa anterior, na etapa *Idealizar* começamos a combinar o conhecimento adquirido sobre o problema com a criatividade para geração de soluções. O objetivo é definir o maior número possível de ideias diferentes, sem se preocupar com a definição de qual é a melhor solução, pois isso será tratado nas últimas duas etapas.

3.1.4 Prototipar

Na etapa *Prototipar*, o objetivo é gerar de forma iterativa uma série de protótipos que tem como foco ajudar a responder as perguntas que ainda existem sobre a solução final. Começamos respondendo perguntas mais gerais, através de protótipos simples e baratos, prosseguindo com o aumento da complexidade tanto da pergunta quanto do protótipo. As variáveis testadas em cada protótipo deverá ser identificada e documentada, assim como a pergunta que esperamos ser respondida por aquele protótipo.

3.1.5 Testar

Por último, temos a etapa *Testar*, em que os protótipos desenvolvidos na etapa anterior são realmente testados e geram um *feedback* sobre as ideias construídas na etapa *Idealizar*. Assim como na primeira etapa, essa etapa tem como foco ganhar mais empatia sobre o usuário, mas com a diferença de ter uma visão do problema definida.

3.2 Adaptações no Design Thinking

Foi optado pelo grupo por adaptar a metodologia vista anteriormente mantendo seus conceitos mas alterando a dinâmica das etapas. Deste modo, as cinco etapas foram reduzidas para três: Definição de requisitos, Desenvolvimento e Validação. A primeira contempla as duas primeiras etapas do Design Thinking. A etapa de Desenvolvimento visa refletir as etapas de Idealização e de Prototipação. Por fim, a etapa de Validação possui como foco substituir a etapa de Teste da metodologia de origem.

Esta redução nas etapas tem por objetivo reduzir a burocracia envolvida no Design Thinking e alocar tempo com maior eficiência, sem perder o contexto de colaboração com o usuário.

3.3 Metodologia ágil

A metodologia ágil visa oferecer entregas contínuas através de uma abordagem colaborativa entre os projetistas. Esta abordagem proporciona flexibilidade através de dinâmicas de organização utilizadas para contornar os desafios do projeto. Estas dinâmicas são definidas pelos métodos Kanban e Scrum explicados nas seções que seguem.

3.3.1 Kanban

O Kanban (do japonês, cartão) se concentra em uma proposta de uso de cartões (físicos ou virtuais) para organização de projetos. Esta organização é dada por colunas que representam estados de atividades, as quais os cartões navegam.

As colunas são voláteis e podem variar de acordo com as necessidades de cada projeto. No contexto deste projeto foram utilizadas como colunas: **A fazer**, **Fazendo** e **Feitas**. Os cartões, que representam atividades, Tem o início de sua trajetória na coluna **A fazer**. Quando um dos integrantes inicia a tarefa, o cartão é movido para a segunda coluna **Fazendo**. Enfim, quando a tarefa é concluída, o cartão é movido para a coluna **Feito**.

Esta proposta busca organizar e proporcionar uma comunicação eficiente e clara entre os integrantes do grupo, uma vez que o quadro onde as colunas estão contidas é exposto e de fácil acesso à todos. Assim, se torna simples obter informações de progresso de tarefas e completude do projeto.

3.3.2 Scrum

Embora o Kanban seja utilizado como forma de organizar as atividades, ele não estimula de maneira direta a comunicação entre os participantes. Assim, sabem-se as tarefas a serem feitas, mas não a prioridade de cada uma. O Scrum visa solucionar este problema, fornecendo uma proposta de organização de projeto consistida em *sprints* e reuniões diárias.

Sprints podem ser entendidos como períodos de trabalho real, onde são definidas metas de conclusão. As *sprints* são acompanhadas de reuniões diárias de atualização, onde o grupo planeja as tarefas do dia de maneira rápida e relata o andamento de tarefas do dia anterior.

O Scrum atua de forma cíclica, ou seja, a cada fim de uma *sprint* é planejada uma nova, de modo que o desenvolvimento do projeto seja contínuo

Assim, a combinação Kanban e Scrum é coerente com a proposta cíclica do Design Thinking, onde cargas definidas de trabalho são seccionadas em períodos de tempo real e o projeto possui uma evolução constante, de forma controlada e organizada.

4 Primeiro Ciclo de Desenvolvimento

4.1 Definição de requisitos

No primeiro ciclo de desenvolvimento, procuramos inicialmente entender as necessidades do mercado através de uma pesquisa de mercado e também através de outros trabalhos acadêmicos já realizados na área.

Na pesquisa de mercado, buscamos ferramentas de ensino relacionadas a teoria musical. Os aplicativos encontrados com funcionamento mais próximo do idealizado foram o Musicca, que pode ser acessado somente pelo navegador e o Music Theory Companion, que funciona mais próximo a um dicionário de conceitos do que a aulas com conteúdo explicado e uma jornada de aprendizado. Apesar de tangenciarem o assunto, nenhum deles tinha abordagem satisfatória da teoria musical, sendo seus pontos mais fortes o aprendizado de instrumentos, composição de música ou ferramentas para auxílio de música.

De acordo com a pesquisa acadêmica, validamos que o aprendizado de teoria musical é considerado necessário para o avanço do aluno na prática musical. A teoria permite a exploração de conteúdos práticos mais complexos, como composição e improvisação (FURLAN; FONTERRADA, 2005). Além disso, a notação musical em específico permite que a música seja compreendida de forma mais abrangente, ajudando a entender sua estrutura e organização, o que leva a uma execução musical mais completa (SOUZA, 2006).

Uma queixa presente nas escolas de músicas de vários níveis é a falta de domínio da leitura e escrita musical, que é considerado inerente ao aprendizado musical, comprometendo o avanço dos alunos (FURLAN; FONTERRADA, 2005).

4.2 Desenvolvimento

A partir da pesquisa efetuada, seguiu-se de acordo com a metodologia estipulada. Assim, segui-se para uma etapa de prototipação através de uma Matriz CSD (Certezas, Suposições, Dúvidas) de modo a entender o problema a partir de uma perspectiva prática. Esta matriz consiste em estabelecer três colunas: Certezas, Suposições e Dúvidas.

A primeira coluna conterá uma série de tópicos referentes as certezas que o grupo possui a respeito do problema analisado.

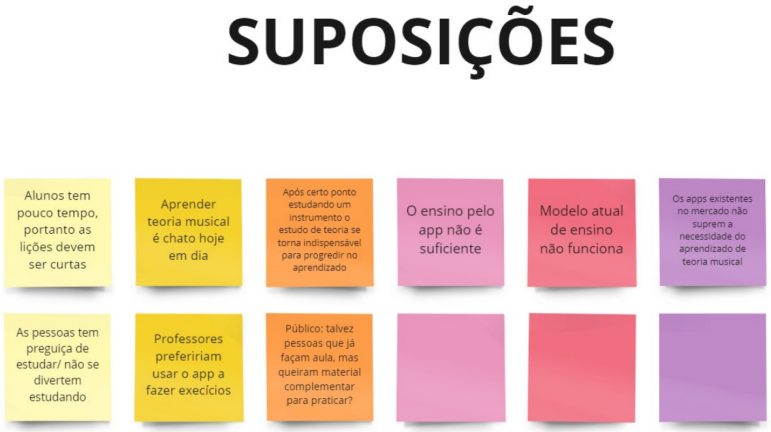
A segunda coluna irá conter informações alocadas em itens a respeito de suposições que o grupo faz a respeito do problema, ou seja, tópicos que fomentam questionamentos em cenários hipotéticos afim de gerar discussões sobre uma solução inicial.

Enfim, a última coluna é reservada para as dúvidas do grupo, ou seja, informações desconhecidas que também vão gerar discussões.

É visto que esta matriz possui importantes finalidades para a etapa inicial do projeto, as quais podemos citar: Compartilhamento da compreensão do problema de forma unânime; Análise de risco do projeto; Tomadas de decisão e fortalecimento do conjunto;

A matriz CSD produzida, com base na pesquisa, pode ser vista na figura 1.

(a) Certezas



(b) Suposições

(c) Dúvidas

Figura 1 – Matriz CSD

4.3 Validação

É seguido então para uma etapa de validação deste protótipo. Para isto, o grupo teve apoio de cinco entrevistados, que acompanharam o projeto ao longo do ano e através

das iterações do Design Thinking forneceram *feedbacks* e conversas para a manutenção de boas decisões, tanto de conteúdo exibido, quanto usabilidade do aplicativo.

Foi então realizada uma bateria de entrevistas com os cinco voluntários,dos quais continham dois professores de música popular, um estudante de música erudita, e dois estudantes de música popular.

As entrevistas seguiram um modelo de perguntas elaboradas previamente, entretanto conforme as conversas evoluíam novas dúvidas e apontamentos surgiam, tornando o modelo de entrevista altamente flexível. O modelo elaborado previamente pode ser visto na figura 2.

Perguntas - Aluno

1. Qual seu nome e idade?
2. Quantos anos você tinha quando começou a se interessar pela música? E estudar de fato?
3. O que te levou a se interessar pela área antes de começar a estudar?
4. Como você começou a estudar música?
5. Quanto tempo você estuda música por semana? Quanto desse tempo é dedicado ao estudo de teoria?
6. Você pratica o estudo de teoria musical hoje? Se sim, como? Se não, por que?
7. Quais exercícios você pratica?
8. Quais tópicos da teoria você mais tem/teve dificuldade em aprender?
9. Quais tópicos da teoria você considera mais fáceis?
10. Quais tópicos da teoria você mais gosta?
11. Quais tópicos da teoria você menos gosta?
12. Quais aplicativos você usa no seu celular?
13. Você já usou algum aplicativo para aprendizado?

Perguntas - Professor

1. Qual seu nome e idade?
2. Quando você começou a lecionar música?
3. Como era o ensino na época? O que mudou de lá para cá?
4. Como é seu relacionamento com seus alunos?
5. Como é a rotatividade dos alunos?
6. Dos que desistem de estudar, geralmente é por qual motivo?
7. Quantos deles se interessam por teoria musical?
8. Quais ferramentas você usa no seu ensino?
9. Seus alunos costumam ter acesso a internet e celular?
10. Se sim, você utiliza disso no aprendizado deles? Como?

Figura 2 – Perguntas roteirizadas

Dadas as conversas foram extraídas informações relevantes para as definições da próxima iteração, a citar:

- Professores sentem falta do desenvolvimento dos alunos em outras áreas da música (e.g. história da música);

- Alunos sentem dificuldade de ferramentas para ajudar com ritmo;
- Alunos que investem mais tempo no aprendizado de música (não apenas prática do instrumento) tendem a ter melhores resultados;
- Alunos sentem maior prazer ao tocar uma música conhecida ao invés de praticar exercícios;
- Alunos possuem pouco tempo para praticar.
- Os métodos de ensino são semelhantes há gerações.

Outro ponto de atenção foi referente aos aplicativos utilizados. Ao questionar os alunos a respeito de aplicativos utilizados no dia a dia, foi observada unanimidade em relação à redes sociais (Youtube, Instagram e Twitter). A respeito de aplicativos utilizados para aprendizado também foi observada unanimidade na resposta Duolingo, havendo também, citações à outros produtos como AnkiDroid utilizado para reforço de memorização.

Todos estes aplicativos possuem em comum uma identidade visual facilmente reconhecida, além de usabilidade facilitada, proporcionando benefícios na experiência do usuário.

5 Segundo Ciclo de Desenvolvimento

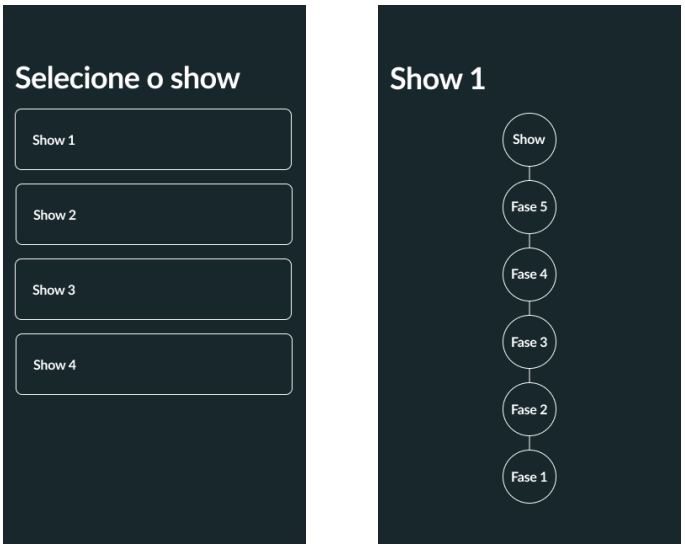
5.1 Definição de requisitos

Com os *feedbacks* dos entrevistados deu-se início à uma nova etapa de definição de requisitos. Foi concluído que modelos de exercício de múltiplas frentes é de grande valia no processo de aprendizado. Também foi definido que exercícios de curta duração são essenciais, uma vez que podem ser realizados em locais como transporte público ou no trabalho.

Assim, foi que o aprendizado será realizado através de uma série de trilhas, onde cada trilha conterá exercícios curtos a serem praticados. Cada exercício terá um tema diferente, de modo a estimular diversas áreas de conhecimento dentro do ramo musical.

5.2 Desenvolvimento

Com os *feedbacks* coletados na entrevista anterior foi proposto o *wireframe* apresentado na figura 3, desenvolvido no software Figma, de modo a sintetizar a solução proposta.



(a) Página principal (b) Fluxo de exercícios

Figura 3 – *Wireframe* da proposta de fluxo

Vemos na solução uma página principal, onde será possível escolher uma trilha de exercícios. As trilhas foram apelidadas de *shows*, e progridem em dificuldade ao longos

das aulas. O final da trilha corresponde a uma prova final, que também recebe o nome de **show**.

Foram modelados também exemplos de exercícios, apresentados na figura 4.

(a) Exemplo de exercício. 1 de 2

(b) Exemplo de exercício. 2 de 2

Figura 4 – *Wireframe* da proposta de exercícios

Assim, temos dois modelos principais de exercícios: Pergunta e Resposta, e Ritmo. Os exercícios visam ser procedurais, ou seja, randomizados a partir de um amplo banco de dados, de modo que a probabilidade de um exercício se repetir seja muito baixa ou próxima a nula.

5.3 Validação

Com o protótipo construído, é necessário seguir o fluxo da metodologia através de uma validação. Uma nova bateria de entrevistas foi realizada com os mesmos entrevistados para que seja avaliada a solução proposta. Além disso, foi construído um formulário para a obtenção de dados quantitativos, além de qualitativos a respeito da solução.

5.3.1 Formulário

O formulário consistia de perguntas de perfilamento gerais, visando categorizar as respostas e identificar preferências de cada público, e perguntas específicas de professores e alunos.

As perguntas dos professores e alunos foram formuladas de forma similar as entrevistas da etapa passada, para confirmar se a visão tirada delas não era enviesada.

Além de validar o formato dos exercícios idealizado até o momento. Sua divulgação foi feita em fóruns de música

5.3.1.1 Análise das respostas

Os respondentes consistiam em sua maioria de indivíduos de nível de escolaridade alta, que já tinham contato com o estudo de música de alguma forma e que já tiveram contato com ensino de música de forma formal, como aulas em sala de aula ao invés de aplicativos, isso provavelmente se deve ao canais de divulgação do formulário, que foi divulgado em fóruns relacionados a música.

Figura 5 – Nível de escolaridade dos respondentes

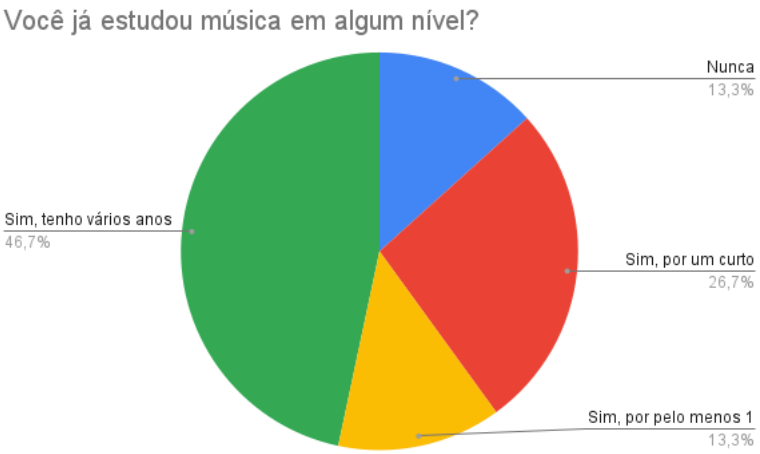


Figura 6 – Nível de estudo de música

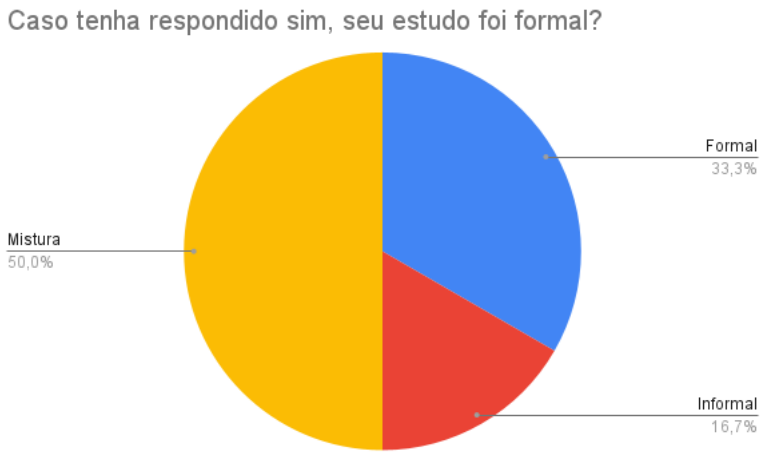


Figura 7 – Formalidade do estudo de música

Foi avaliado a utilização de aplicativos de aprendizado pelos alunos e professores tanto dentro quanto fora da sala de aula. Percebeu-se que os professores utilizam mais aplicativos na hora de ensinar o conteúdo do que os alunos utilizam para estudar fora do horário de aula.

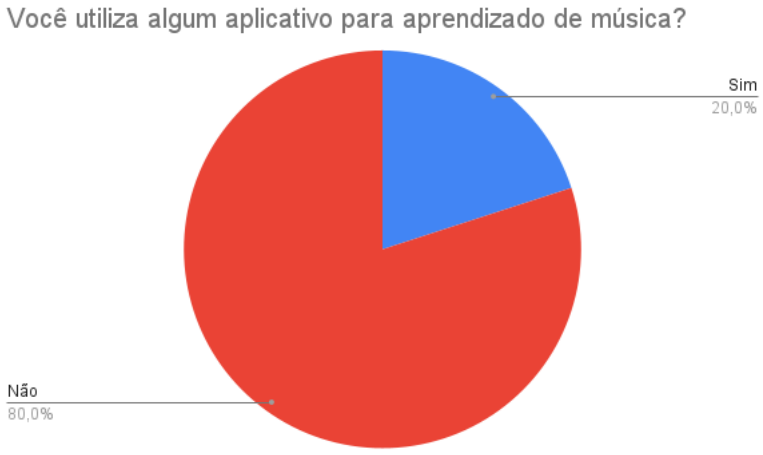


Figura 8 – Alunos que usam aplicativos no aprendizado

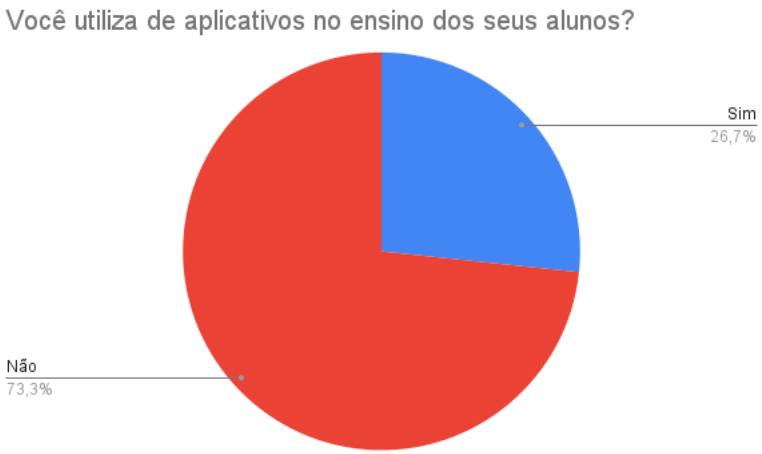


Figura 9 – Professores que usam aplicativos no ensino

Entre os aplicativos utilizados pelos professores, os mais comuns são os que servem como sala de aula, como Meets e Miro. Alguns professores incentivam seus alunos a usar aplicativos fora da sala de aula, seja para ensaiar cifras, testar afinação de canto ou até facilitar a leitura de partituras. Porém, o ensino de música ainda aparenta estar atrelado a sala e horário de aula.

Avaliou-se o tempo de dedicação que os alunos davam ao estudo, junto com a importância definida por eles e as expectativas dos professores nos mesmos âmbitos.

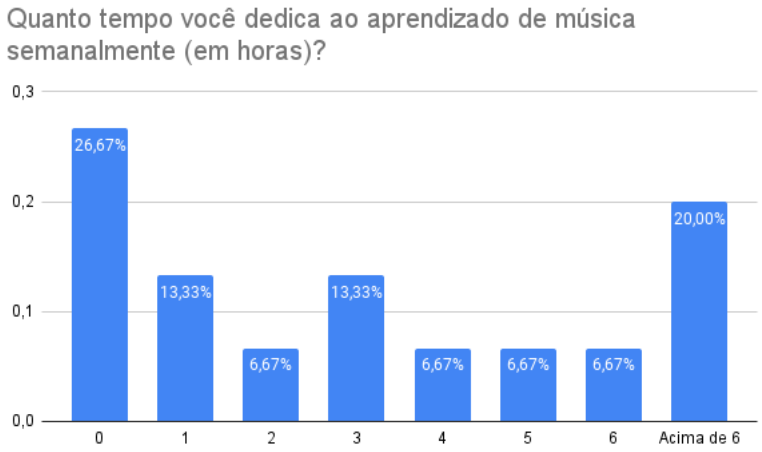


Figura 10 – Tempo dedicado pelo aluno ao aprendizado de música

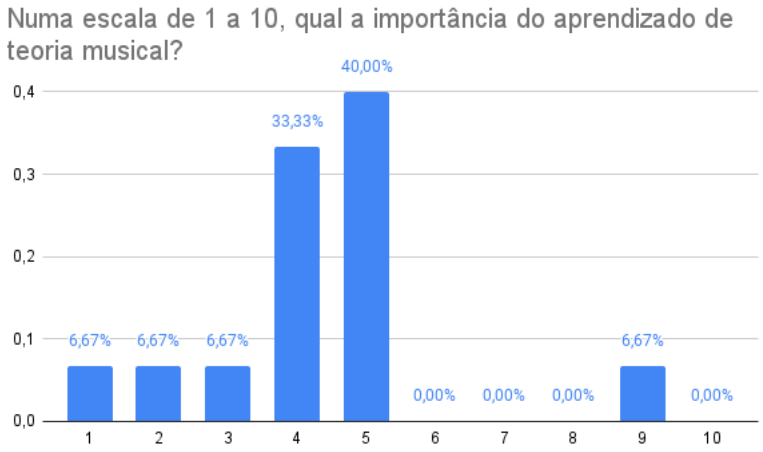


Figura 11 – Importância dada pelo aluno à teoria musical

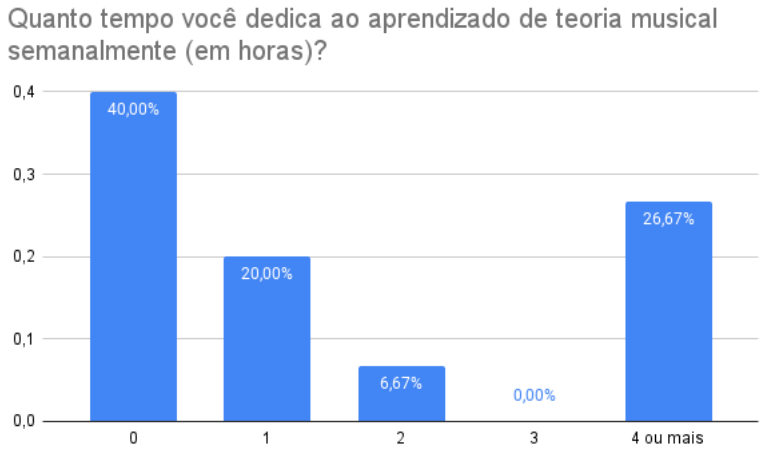


Figura 12 – Tempo dedicado pelo aluno ao aprendizado de teoria musical

Percebe-se que os alunos, apesar de muitas vezes dedicados ao estudo de música, não dão importância para se aprofundarem em teoria musical, chegando a mais de 70% a se dedicar 2 horas ou menos por semana. Já pelo lados dos professores as expectativas são mais altas.

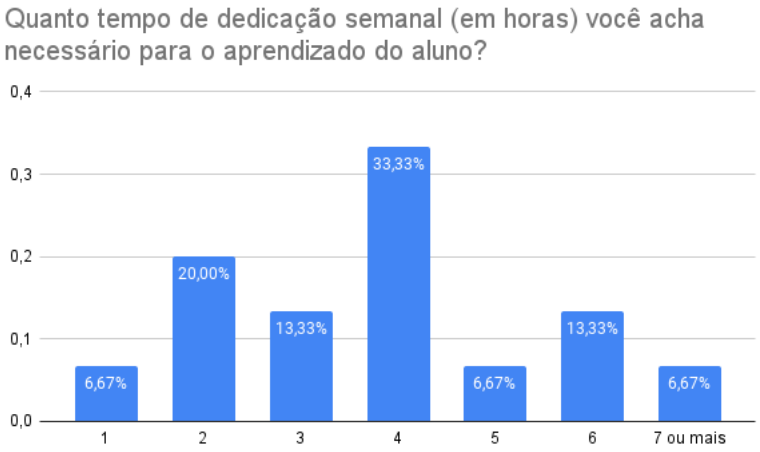


Figura 13 – Tempo esperado pelo professor de dedicação do aluno ao estudo semanalmente

Cerca de 70% dos professores esperam dedicação de 3 horas ou mais dos alunos. Identifica-se aí um ponto de possível melhora nos estudos dos alunos fora da sala de aula, parece ser necessário um incentivo maior nesses horários de estudo sem o professor. Uma lacuna que poderia ser preenchida pelo uso do aplicativo.

A partir disso, avaliou-se quais tipos de estudo fora da sala de aula são preferidos pelos alunos e professores, junto com o que os professores acham mais eficiente no aprendizado e o que os alunos acham mais incentivador para dedicar essa horas extras.

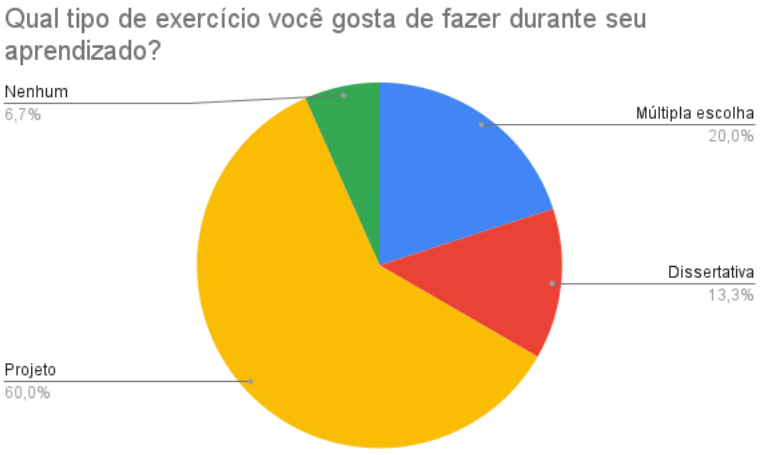


Figura 14 – Tipo de exercício preferido pelos alunos

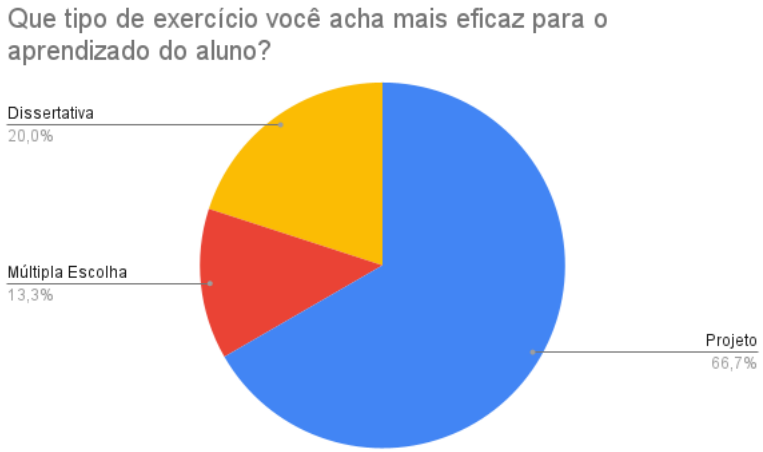


Figura 15 – Tipo de exercício preferido pelos alunos

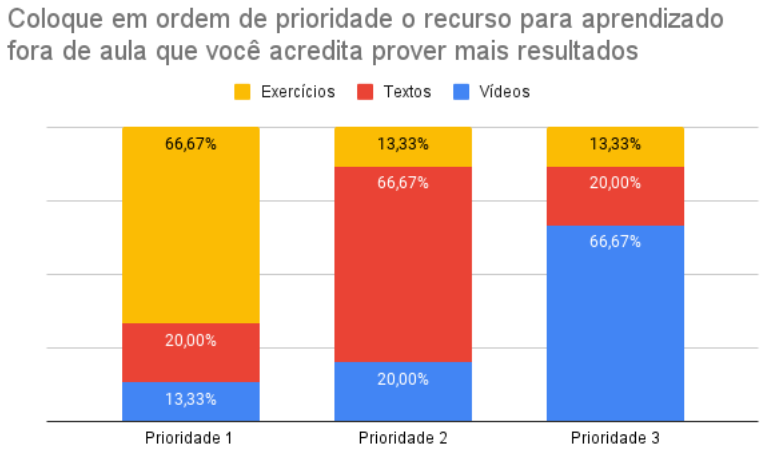


Figura 16 – Tipo de exercício preferido pelos alunos

Em geral, tanto os alunos quanto os professores acreditam que formas de aprendizado que fujam do modelo básico de questões e fazem os alunos pensarem mais em suas realizações são mais eficazes e incentivadoras no aprendizado. O que incentiva a produção de uma variedade de atividades diferentes no aplicativo.

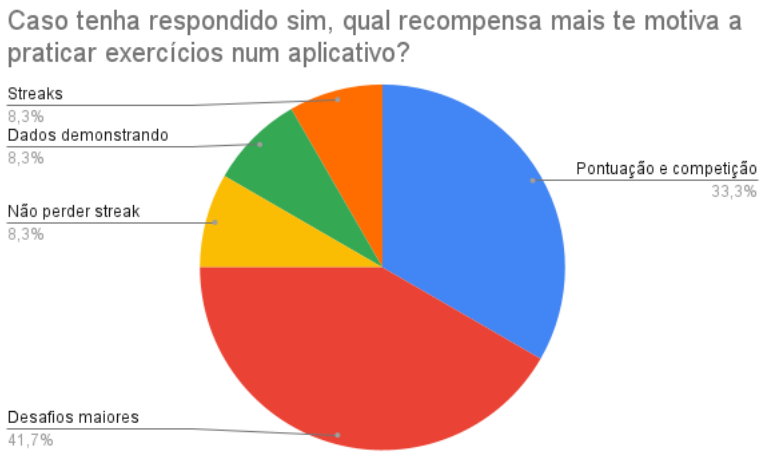


Figura 17 – Motivações providas por aplicativos preferidas pelos alunos

Percebe-se que os alunos se sentem mais amplamente incentivados pela competição entre seus pares, além do aumento de dificuldade nas tarefas realizadas. Isso segue paralelo a ideia da *gamificação* aplicada no projeto, uma vez que ambos são fatores que geram diversão em jogos.

Por fim, avaliou-se as definições do projeto até o momento, perguntando a ambos os públicos sua opinião sobre os exercícios formulados de ritmo e múltipla escolha.

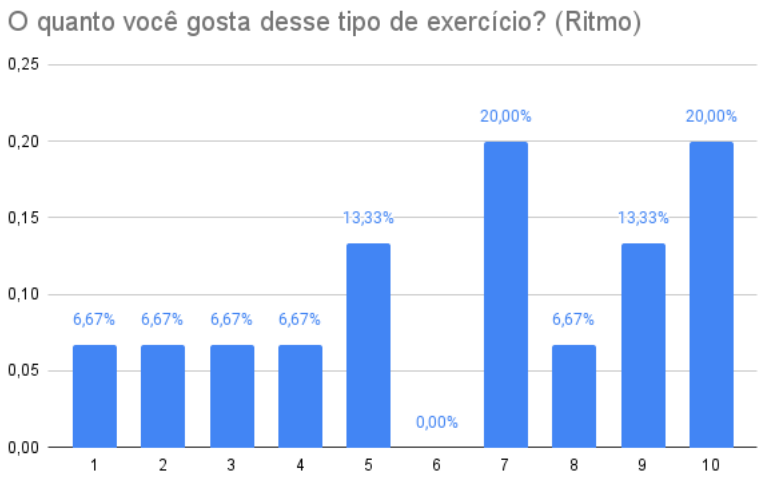


Figura 18 – Avaliação dos alunos sobre o exercício de ritmo

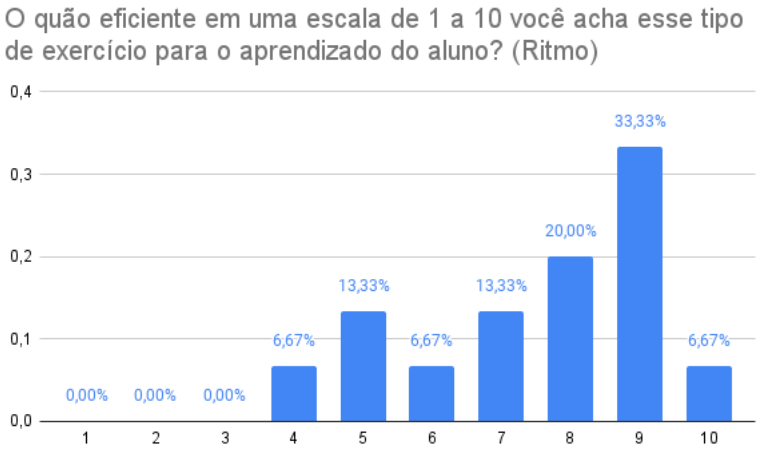


Figura 19 – Avaliação dos professores sobre o exercício de ritmo

Figura 20 – Avaliação dos alunos sobre o exercício de múltipla escolha

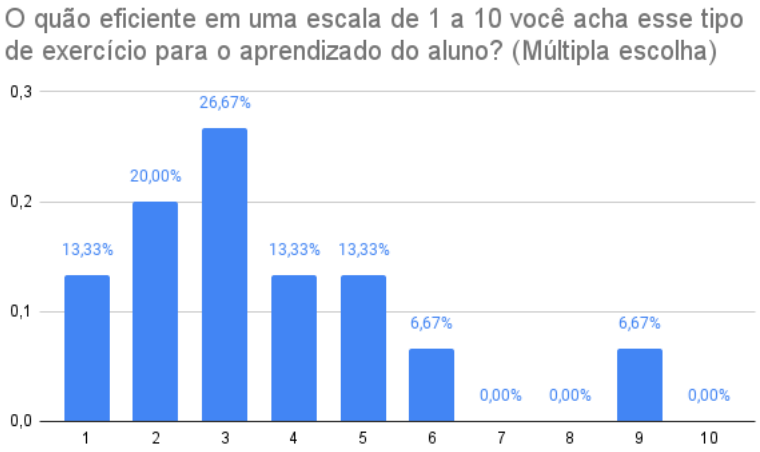


Figura 21 – Avaliação dos professores sobre o exercício de múltipla escolha

Os resultados obtidos reforçam a noção de que exercícios padronizados de múltipla escolha não são considerados divertidos e nem eficazes por ambos os públicos. Nota-se a demanda por exercícios que interajam com os alunos de formas além do simples texto.

5.3.2 Entrevistas

As entrevistas nesta etapa foram realizadas em formato de conversa livre, sem um roteiro. Foi coletado *feedback* onde vimos que a quantidade de modelos de exercícios era precária, e logo se tornaria maçante. As nomenclaturas também se mostraram confusas, não ficando claro o significado do *show*. O fluxo, entretanto, era agradável por assemelhar à soluções já vistas e consumidas pelos usuários (como o Duolingo).

Dada a comparação com o aplicativo Duolingo, um ponto abordado de maneira unânime entre os estudantes foi a importância de *features* como pontuações, *streak* de dias estudados e possibilidade de relacionamento entre conhecidos.

Deste modo, o fluxo principal foi validado, mas se vê necessário o acréscimo de alguns conteúdos para justificar o consumo frequente do aplicativo.

6 Terceiro Ciclo de Desenvolvimento

6.1 Definição de requisitos

Com o *feedback* da iteração anterior, deu-se início à definição de requisitos para uma nova prototipação. Visando adicionar mais elementos que proporcionem um sentimento de *gamificação*, foi adicionado um contador de *streak* de dias estudados. A cada dia consecutivo que o usuário realizasse uma lista de exercícios o aplicativo deverá incrementar uma unidade neste contador, e, caso o usuário passe um dia inteiro sem concretizar uma lista de exercícios, este contador será zerado. Este recurso visa estimular o estudo diário.

Para incentivar que os exercícios sejam realizados corretamente, serão adicionados outros dois recursos: vidas e pontuação.

Ao término de cada lista de exercícios o usuário irá receber uma pontuação referente aos acertos obtidos na lista, de modo que seja incentivado o maior número de acertos possíveis na lista.

Caso o usuário responda um exercício erroneamente, será descontado uma vida. As vidas são relacionadas à conta do usuário, tendo um limite máximo de 5. As vidas são restauradas por tempo, com o usuário recebendo uma nova vida a cada 24 horas.

Além disto, será adicionado também uma estrutura para seguir outros usuários e ser seguido, de modo que possam ser comparadas estatísticas entre usuários, proporcionando uma competição de estudos. Os usuários poderão acompanhar, além de seu progresso em pontuação e *streak*, o progresso de amigos.

Também devem ser adicionados novos modelos de exercícios, a fim de evitar interações monótonas, o que abaixaria a taxa de persistência do produto.

6.2 Desenvolvimento

O desenvolvimento iniciou-se através de telas Figma. Foram criados os elementos de *design* principais, os quais foram utilizados para a construção das telas. Em seguida, deu-se a implementação em paralelo do *frontend*, *backend* e infraestrutura para armazenar a aplicação.

Os detalhes de arquitetura e decisões tomadas serão vistos nas seções subsequentes.

6.3 Entrevistas

O foco das entrevistas nessa etapa foi abordar os pontos de melhoria implementados e procurar pontos de refino. As entrevistas foram novamente realizadas em formato de conversa livre, sem um roteiro. Nessa versão o *feedback* se mostrou mais positivo, os entrevistados demonstraram entender o fluxo de funcionamento naturalmente, algo que é comum quando se trata de jogos e é um sinal positivo de que foram seguidos padrões de mercado. Além disso, os exercícios implementados foram muito bem recebidos e, de acordo com os entrevistados, trouxeram uma variedade muito bem-vinda, validada nas diferentes etapas de empatia. Pôde-se fugir ao máximo do formato tradicional de exercícios observado na maioria dos aplicativos de teste de conhecimento. Deste modo, o aplicativo oferece uma gama de exercícios suficientemente diferente para não cansar o usuário, funcionando de forma similar a um conjunto de *minigames*.

7 Implementação

Para implementação, foi optado por separar a aplicação em duas frentes desacopladas: *frontend*, no aplicativo desenvolvido em Dart com Flutter (seção 7.4), e *backend*, uma API REST desenvolvida em Kotlin com Spring Boot (seção 7.6).

O *frontend* contempla a interface do aplicativo de dispositivo móvel, onde o usuário interage, concluindo os exercícios propostos, consultando suas pontuações, e utilizando todas as funções do aplicativo. O *backend* por sua vez é responsável por receber, tratar, e enviar os dados gerados e consumidos pela interação do usuário com o *frontend*.

A persistência de dados foi feita através de um banco de dados com MySQL (seção 7.5). Foi utilizado um serviço externo, o Google Firebase, para realizar a autenticação de usuários (seção 7.3).

A partir destas definições, foi modelada a seguinte arquitetura para a solução do projeto:

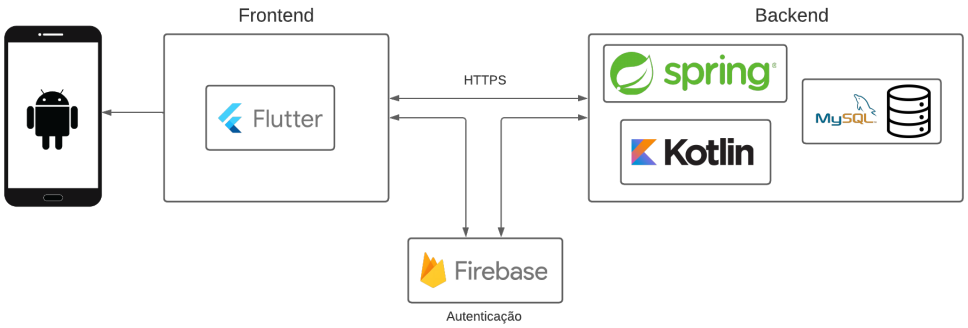


Figura 22 – Arquitetura da solução

A decisão de separar o *frontend* do *backend* em duas aplicações diferentes oferece vantagens significativas em termos de flexibilidade, escalabilidade e manutenibilidade. Com este modelo, o *backend* pode ser consumido por mais de uma aplicação diferente, permitindo que sejam implementados outros *frontends* como aplicações web ou *desktop*, sem ter a necessidade de alteração no servidor *backend*. Além disso, com as aplicações separadas o gerenciamento de recurso é facilitado, otimizando o consumo de acordo com as necessidades específicas de cada parte da aplicação.

7.1 Requisitos finais

De acordo com as especificação definidas nos ciclos de desenvolvimento, foram determinadas as funcionalidades essenciais para a completude do projeto. Os aspectos principais levantados são a diversidade de exercícios e o sistema de pontuação como

feedback positivo do usuário. Além desses, outros aspectos que contribuem para o sucesso da proposta são o *design* consistente, de acordo com a estética de um jogo, e a interação com outros usuários.

Para que essas funcionalidades sejam alcançadas, é necessário ter o sistema de pontuação e *streak*, o sistema de amizade entre usuários e funcionamento das aulas e exercícios separadas em trilhas pertencentes a diferentes unidades, que abordam tópicos diferentes dentro da área de teoria musical. Além disso, as aulas são separadas em quatro temas: percepção, ritmo, partitura e história.

No funcionamento das aulas e dos exercícios é preciso que os *assets* estejam guardados em algum serviço de armazenamento e que sejam fornecidos pelo *backend*, de acordo com o conteúdo acessado pelo *frontend* durante a utilização do aplicativo pelo usuário. É necessário também salvar os resultados das realizações dos exercícios para o sistema de pontuação.

Os resultados da interação do usuário com os exercícios devem ser atualizados, devolvendo as informações sempre que necessário. Isso é essencial para que o sistema de amizades cumpra seu objetivo, uma vez que a comparação dos scores na tela de estatística contribui para a competição entre os usuários e o aumento no processo de *gamificação* do aprendizado.

Algumas funcionalidades, apesar de não levantadas no ciclo de *Design Thinking*, são essenciais para o serviço, como disponibilidade e geração do instalador do aplicativo.

7.2 Design

O projeto visual do aplicativo possui como objetivo maximizar a quantidade de usuários que se identificam com o produto. Deste modo, foi abordada uma estética de fácil leitura, trazendo cores saturadas e formas simplificadas de fácil leitura.

7.2.1 Cores

O processo de desenvolvimento do design teve início com a definição de uma paleta de cores. Deste modo, temos o início da identidade visual, que possui como objetivo trazer cores contrastantes e de alta saturação, a lembrar um cenário *cartunizado*.

A primeira decisão tomada foi referente à escolha das cores de cada frente de ensino. Uma vez que o foco é contraste, foi decidido utilizar uma harmonização quadrada dentro do círculo cromático, de modo a maximizar o contraste via matiz. Assim, as cores foram definidas a partir do estudo apresentado na figura 23.

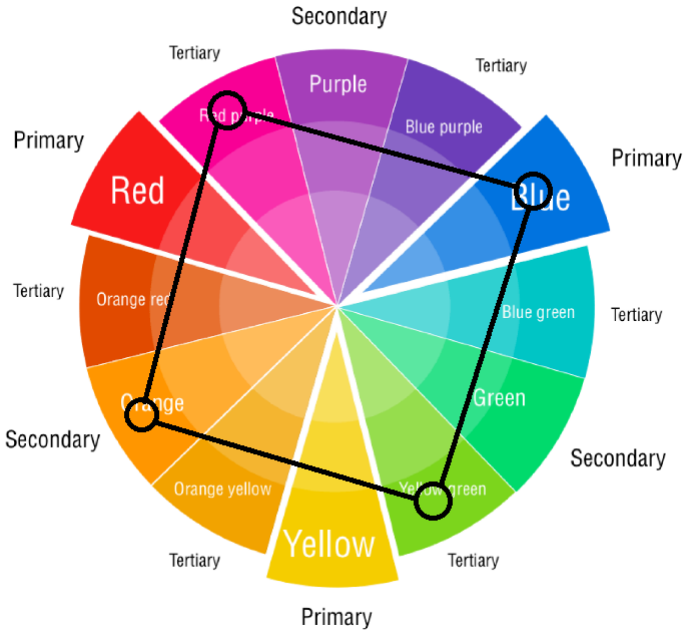


Figura 23 – Estudo de cores definidas no círculo cromático

Além da separação das frentes (percepção, ritmo, partitura e história), cada aula deverá ter uma segregação visual para definir o nível que o usuário está em uma aula. Assim, dentro de cada escopo de matiz foram definidos cinco graus de saturação, onde o nível com menor grau de saturação será utilizado como cor secundária para sua frente. Deste modo a paleta de cores para as frentes teve o resultado apresentado na figura 24.

Figura 24 – Paleta de cores definidas separadas por frente

Ao atingir o maior nível de uma lição o usuário deverá compreender visualmente que esta lição está finalizada. Para isto, foi definido uma cor de dourado para simbolizar o maior grau de completude de uma lição, conforme apresentado na figura 25.

Figura 25 – Paleta de dourado

Com as cores das frentes definidas, o aplicativo precisa de uma paleta de cores padrão. Ainda seguindo a ideia de contraste entre as cores, foi definida uma cor com matiz diferente de qualquer frente, mas que ainda concretizasse o ideal de destaque em meio às demais cores. Foi encontrado no vermelho tais qualidades. Assim, definimos a seguinte paleta apresentada na figura 26 para abrigar os componentes neutros do aplicativo.

Figura 26 – Paleta de cores padrão do aplicativo

Enfim, nota-se a necessidade de uma paleta de tons de cinza para usos variados na aplicação. Deste modo, definimos sete tons de cinza frio para que sejam usados por múltiplos componentes, finalizando a definição da paleta de cores da aplicação.

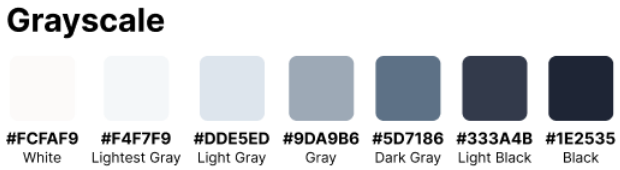


Figura 27 – Paleta de tons de cinza

7.2.2 Fontes

Com foco em acompanhar a estética apontada pela definição de cores, as fontes devem seguir um padrão de fácil legibilidade. Foram definidas então duas fontes principais: **Montserrat** e **OpenSans**. Ambas as fontes foram projetadas com foco em legibilidade em dispositivos móveis, o que trás como benefícios boa responsividade sem perder a estética visada.

O pacote de fontes definidas para o aplicativo pode ser visto na figura 28.

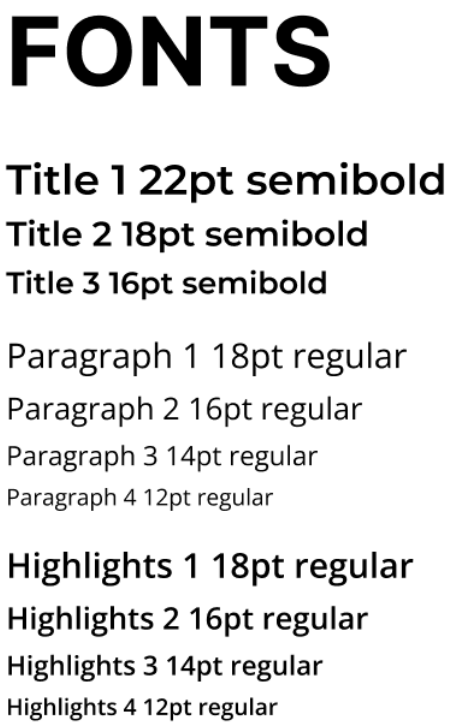


Figura 28 – Fontes definidas para o aplicativo

7.2.3 Componentes e Ícones

Os ícones utilizados na aplicação seguem a estética apresentada anteriormente, sendo arredondados e respeitando a paleta de cores definidas. O pacote de ícones pode ser

visto na figura 29.

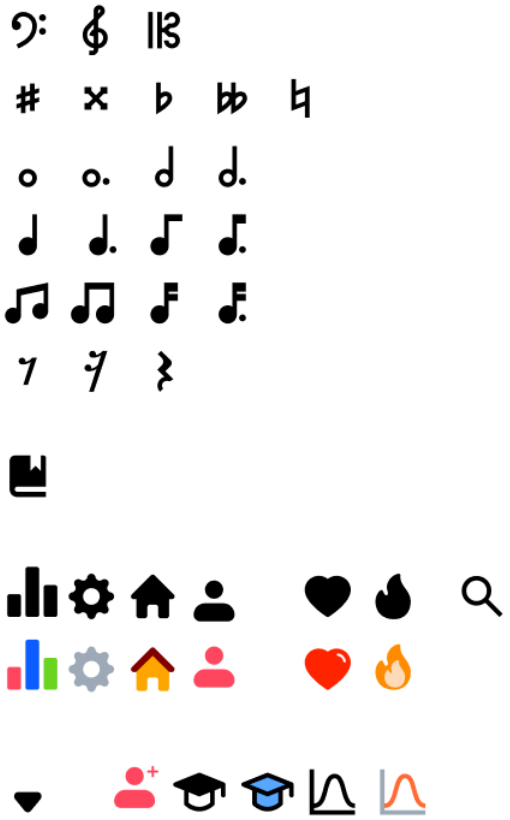


Figura 29 – Ícones selecionados para o aplicativo

Em seguida, foi definido os principais componentes visuais que compõe as telas da aplicação, apresentados nas figuras 30 e 31.

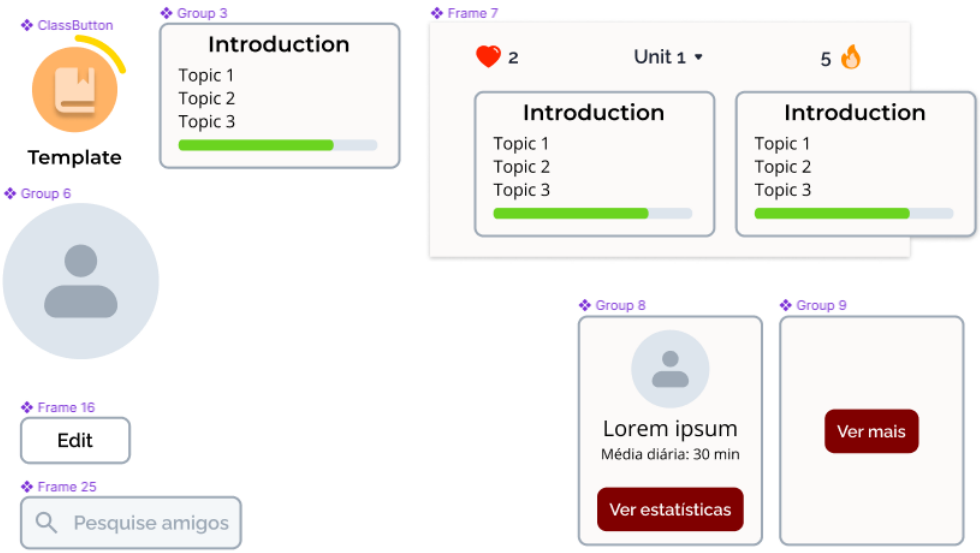


Figura 30 – Componentes criados para a composição das telas. 1 de 2

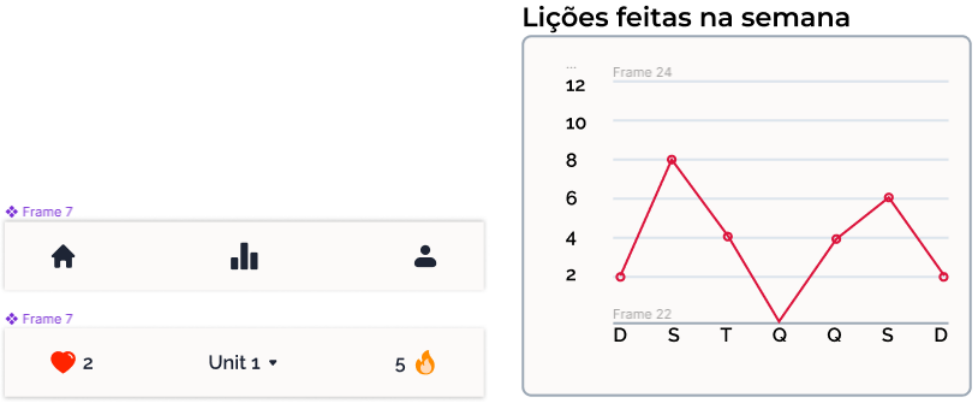


Figura 31 – Componentes criados para a composição das telas. 2 de 2

7.2.4 Personagens

Os personagens compõem uma banda, onde a estética respeita as formas e cores pré-estabelecidas pelas etapas de desenvolvimento anteriores. Assim, temos quatro personagens que para compor a banda, apresentados na figura 32.

(a) Baterista (b) Guitarrista

(c) Cantora (d) Baixista

Figura 32 – Personagens

É visto a presença de acessórios dourados em todos os personagens, visando uma identidade de conjunto entre os integrantes da banda. Além disso, cada integrante possui cores majoritárias referentes à frente de ensino à qual é designado, facilitando o entendimento do usuário da relação entre o personagem e sua frente.

Cada personagem possui uma personalidade própria, que é exposta ao longo dos conteúdos estudados onde o usuário terá diálogos com os personagens. Estas personalidades foram transpostas nas ilustrações, para enfatizar certos comportamentos e falas. Esta decisão foi tomada de modo a abraçar o maior público possível. Assim, o usuário terá maior probabilidade de empatizar com um personagem dada sua personalidade.

7.2.5 Fluxo de telas

O fluxo de telas não deve ser complexo, dado que o objetivo é atingir um maior público possível. Assim, o aplicativo é composto de três páginas principais: *Home*, Estatísticas e Perfil.

Após o *login* no aplicativo o usuário é redirecionado para a página *Home* que contém uma lista de unidades, onde cada uma possui uma lista de aulas. Cada aula contém um conteúdo, que pode ser lido de maneira opcional, e uma lista de exercícios referente ao conteúdo.

A tela de estatística fornece para o usuário uma coletânea de dados que podem ser comparadas com dados de amigos, de maneira a incentivar a competitividade e consequentemente aumento no ritmo de estudo.

O perfil do usuário contém acesso às configurações do aplicativo e às informações do próprio usuário, que incluem dados pessoais e amizades. As amizades podem ser consultadas de modo a serem feitas novas amizades, ou desfazer amizades presentes através do mecanismo de **Seguir** explicado na subseção 7.2.6.5.

O diagrama da figura 33 mostra o fluxo das telas.

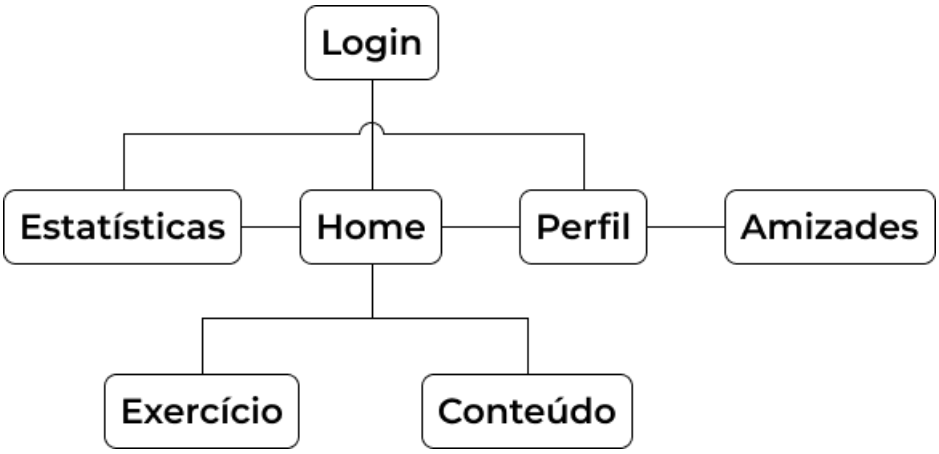


Figura 33 – Fluxo de telas do aplicativo

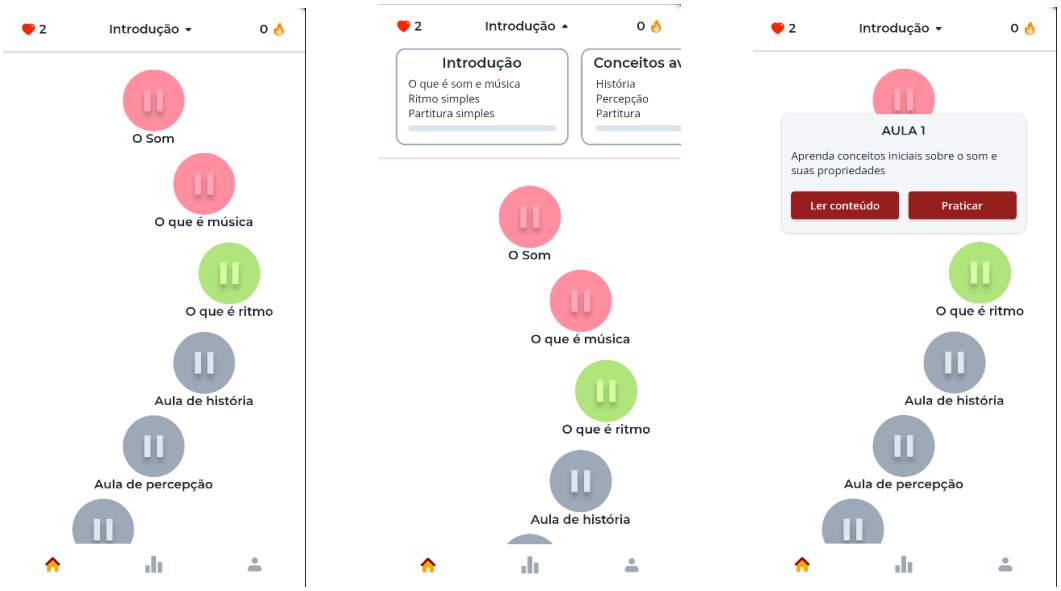
7.2.6 Implementação das telas

7.2.6.1 Home Page

O *header* da página foi definido de forma a abrigar os dados de vida do usuário, *streak* de dias para incentivar o estudo e um *dropdown* para que sejam acessadas as unidades. Dado que a troca de unidade acontece de maneira pouco frequente, não deve ser o conteúdo de mais fácil acesso na página.

Na figura 34 temos a trilha de lições. Estas foram desenhadas de modo a dar dinamismo ao longo do *scroll* pela página. Ao clicar em uma aula, um modal é aberto,

possuindo informações sobre a aula e dois botões, que podem redirecionar o usuário para o conteúdo da aula, ou para a lista de exercícios.



(a) Página principal (b) Dropdown de unidades (c) Modal para acessar conteúdos

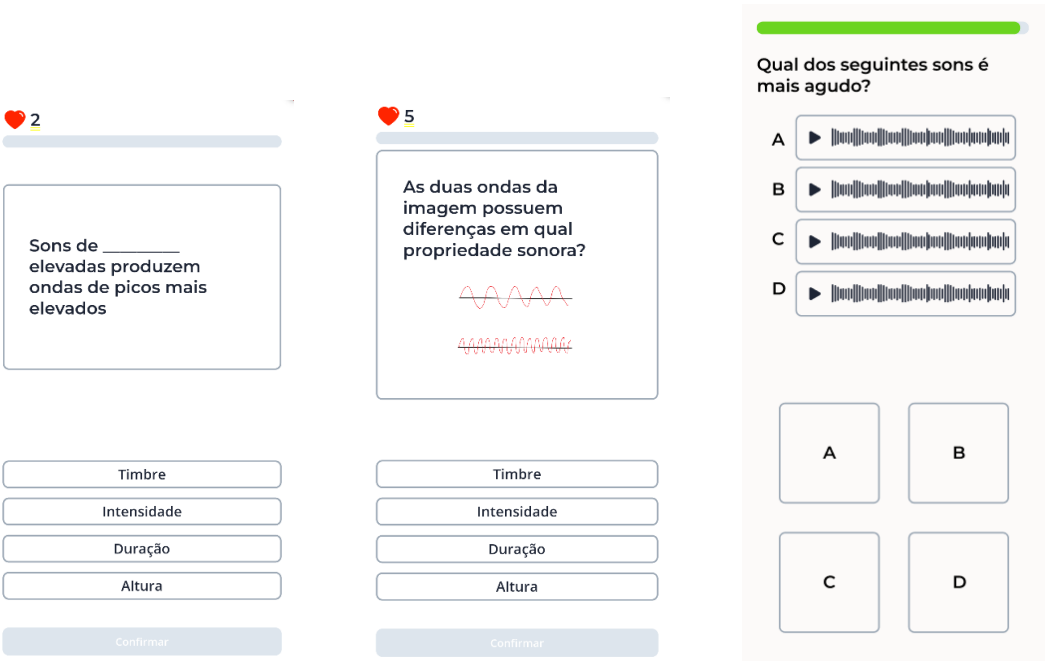
Figura 34 – *Home page*

7.2.6.2 Exercícios

Foram definidos cinco modelos principais de exercícios:

- Pergunta e resposta apenas com texto;
- Pergunta e resposta com texto e imagem;
- Pergunta e resposta apenas com áudio;
- Relacionar as colunas;
- Exercício de ritmo;

A modelagem das telas para cada tipo de exercício pode ser vista na figura 35.



(a) Exercício de texto (b) Exercício de imagem (c) Exercício de áudio

(d) Exercício de relacionamento (e) Exercício de ritmo

Figura 35 – Telas de exercício

7.2.6.3 Estatística

A página de estatística visa fornecer ao usuário a possibilidade de visualizar e comparar seu desempenho com de outros usuários através de métricas. Assim, é possível acompanhar a o andamento dos estudos a longo prazo.

Figura 36 – Página de estatística

7.2.6.4 Perfil

A página de perfil foi modelada para abrigar as informações pessoais do usuário, tais como nome, e-mail e amigos. Também é possível realizar operações sobre o aplicativo, como configurações e sair da conta. Esta página também deve dar acesso a uma página de busca de novas amizades, sendo possível buscar e seguir novas pessoas.

(a) Página principal (b) *Dropdown* de unidades

7.2.6.5 Busca de amigos

A busca de amigos é uma busca simples, onde traz uma lista de *tiles* que contém informações do usuário e a possibilidade de segui-lo. Ao seguir, é possível acompanhar seu progresso na página de estatísticas. Caso o usuário já siga a pessoa listada, é possível deixar de segui-la.

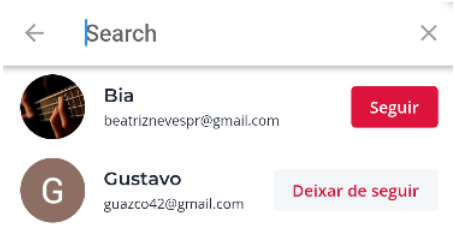


Figura 37 – Página de busca de amigos

7.3 Serviços externos

Com foco em facilidade de usabilidade, a autenticação da aplicação tinha foco em seguir o protocolo **OAuth2.0**. Deste modo, é necessário fazer uso de serviços de autenticação externos. A escolha foi direcionada para a integração do aplicativo com o Firebase. Este serviço é disponibilizado pela empresa Google e possui diversas aplicações, e, dentre elas, autenticação de acordo com o protocolo desejado.

A escolha do Firebase se deu pela facilidade de integração e ganho na usabilidade, uma vez que o usuário não tem necessidade de criar uma conta.

7.4 Frontend

7.4.1 Escolha da tecnologia

A escolha da tecnologia se dirigiu em torno dos seguintes tópicos:

- Desempenho;
- Facilidade de desenvolvimento;
- Documentação.

Dadas as necessidades do projeto, foi escolhida a linguagem **Dart** em conjunto com o *framework* **Flutter** para o desenvolvimento do *frontend*. Este *framework* permite agilizar o processo de desenvolvimento, uma vez que sua estrutura é simples e de fácil manuseio, além de auxiliar a escalabilidade do projeto devido sua estrutura de componentização por *Widgets* (INTRODUCTION..., 2023).

A integração com serviços externos é outro ponto vantajoso para este *framework*, que possui diversos recursos para que requisições HTTPS sejam realizadas de maneira simples, poupando tempo de desenvolvimento (NETWORKING, 2023); além de facilitar a integração com o Firebase, dado que ambos são mantidos pela Google.

Um último fator que contribui para a escolha do Flutter é a vasta documentação (FLUTTER..., 2023), que facilita o aprendizado e a resolução de problemas ao longo do desenvolvimento.

7.4.2 Arquitetura

A arquitetura escolhida, BLoC (Business Logic Component), é um padrão arquitetural utilizado no desenvolvimento de aplicativos que separa a lógica de negócios da interface do usuário e da camada de dados (SOARES, 2018). Esta arquitetura é especialmente popular no contexto do framework Flutter, mas pode ser aplicada em outras tecnologias. A estrutura fundamental da arquitetura BLoC envolve três componentes principais:

- **Eventos**: Representam as ações ou acontecimentos ocorridos no sistema que desencadeiam uma mudança de estado. Eventos são as entradas para o BLoC.
- **BLoC**: É o componente central que contém a lógica de negócios da aplicação. O BLoC recebe eventos como entrada, processa esses eventos e emite novos estados como resultado. Ele é responsável por gerenciar o estado da aplicação de acordo com as interações do usuário ou outros eventos do sistema.

- **Estados**: Representam as condições ou situações em que a aplicação pode se encontrar em resposta a eventos específicos. Os estados refletem a saída ou resultado da lógica de negócios contida no BLoC.

O fluxo de trabalho típico na arquitetura BLoC é acionado por eventos, comumente acionados pela interface, que são enviados ao BLoC. O BLoC processa esses eventos e, como resultado, emite novos estados. A interface do usuário reage a esses estados, atualizando a exibição de acordo com a lógica definida no BLoC.

Essa arquitetura promove a separação de preocupações e facilita a testabilidade do código, uma vez que a lógica de negócios está encapsulada no BLoC. Além disso, a utilização do padrão BLoC contribui para a manutenção do código, já que as diferentes camadas da aplicação estão claramente definidas e isoladas umas das outras.

Visualmente, podemos definir o fluxo de trabalho conforme o diagrama apresentado na figura 38.

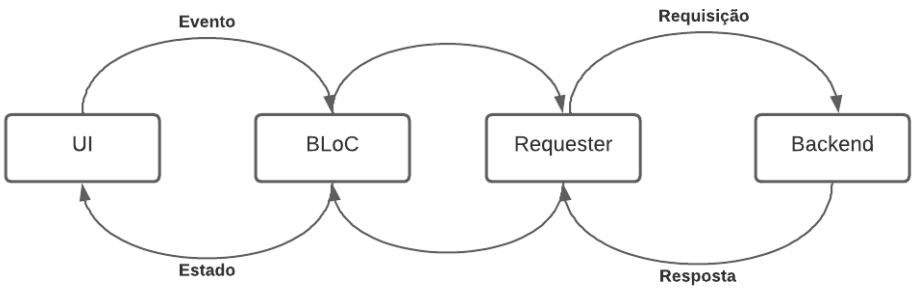


Figura 38 – Fluxo da arquitetura BLoC

Vemos na figura que o fluxo principal BLoC que compõe o *frontend* está contido nos dois primeiros blocos do diagrama, onde temos a interface que aciona o processamento do BLoC através de eventos, e esta responde alterando o estado da UI (*User Interface*).

Os dois blocos subsequentes (*Requester* e *Backend*) possuem como principal funcionalidade a integração de dados, analisada na subseção 7.4.4.

A estrutura do projeto foi separada nos seguintes BLoCs:

- **LoginPage**: Página de login que contém a lógica de integração com o Firebase para autenticação;
- **HomePage**: Página principal da aplicação que irá conter lista de lições, lista de unidades, e dados de vida e *streak*;
- **LessonPage**: Página que irá conter a lista de exercícios;

- **ContentPage**: Página de conteúdo multimídia de uma determinada lição;
- **ProfilePage**: Página de dados do usuário, incluindo dados pessoais e amigos;
- **StatisticsPage**: Página de estatísticas do usuário e de amigos;

7.4.3 Organização do código

Para a organização da arquitetura, o projeto seguiu a seguinte estrutura base:

```
/
├── common
├── models
├── modules
│   └── home_page
│       ├── ui
│       └── bloc
├── repositories
│   └── network
├── theme
├── app.dart
├── constants.dart
└── main.dart
```

Os arquivos na raiz do projeto (finalizados em **.dart**) são: **App**, **Constants** e **Main**. O primeiro arquivo contém a inicialização da aplicação. O arquivo **Constants** abriga um conjunto de constantes, como por exemplo os caminhos fixos dos *endpoints* consumidos pelos Requesters. O último inicializa o aplicativo juntamente com os *providers* de cada BLoC.

Os diretórios vistos na estrutura tem como responsabilidade agrupar os seguintes componentes:

- **Commom**: Contém elementos que são consumidos entre os múltiplos módulos.
- **Models**: Contém a declaração de todas as classes a serem instanciadas como objetos.
- **Modules**: Contém os módulos estudados nas seções seguintes. Cada módulo contém uma estrutura BLoC dentro de si.
- **Repositories**: Contempla duas estruturas principais: *repositories* e *requesters*. Os *repositories* agem como *hubs* por onde os BLoCs irão requisitar dados. Os *requesters*, por sua vez, são as estruturas por onde são feitas as chamadas HTTP vistas na subseção 7.4.4.

- **Theme**: Contém as configurações de tema do projeto como cores e fontes.

Cada módulo contempla dois diretórios: **ui** e **bloc**. O diretório *ui* é responsável por armazenar todos os componentes (e.g botão, *card*, *modal*) de interface que irão compor a(s) página(s) do módulo. o diretório *bloc* é responsável por armazenar três principais arquivos: state, event e bloc. Como visto no subseção de arquitetura (7.4.2), o arquivo bloc é responsável por gerenciar estados e eventos separados por lógicas de negócio. Cada evento disparado pela interface é capturado pelo bloc, que retorna estados que alteram variáveis para a manutenção da página. O bloc também é responsável por fazer chamadas HTTP para o *hub* Repository. Os arquivos *state* e *event* possuem declarações de classes de acordo com as necessidades do módulo. Estas classes são responsáveis por engatilhar as alterações citadas.

7.4.4 Integração

O *frontend* se vê integrado em dois serviços principais: *backend* interno e Firebase. A integração com o Firebase tem início nas configurações do projeto no Console do Firebase. Isso envolve a definição de parâmetros essenciais e a obtenção dos arquivos de configuração necessários para o *frontend* Flutter. A configuração inclui a criação de um aplicativo Flutter no console Firebase e a adição de arquivos no projeto como **google-services.json**.

Com as configurações iniciais realizadas e as devidas dependências instaladas, é possível acessar as classes do Firebase que realizam a autenticação através do protocolo OAuth2.0. Deste modo, foi implementada uma função que realizasse o login através de uma conta Google, fazendo com que o aplicativo recebesse um *token* de autenticação gerado pelo Firebase e dados pessoais do usuário como nome, e-mail e foto.

O *token* recebido é repassado para uma validação no *backend*, onde há registro de todos os usuários cadastrados na plataforma. Caso seja o primeiro *login* do usuário, seus dados são cadastrados no banco. De modo contrário, seus dados são resgatados para que seu progresso de estudos seja continuado adequadamente.

A integração do *backend* se deu através de requisições HTTP em *endpoints* aos quais o *backend* serve. Foi designado para cada página os dados necessários para o preenchimento de cada campo, e, disponibilizados os dados através dos *endpoints* (detalhados na seção 7.6), os dados foram alocados adequadamente.

Em cada requisição realizada, é passado no *header* da requisição o *token* do usuário disponibilizado pelo Firebase de modo que o *backend* valide o cadastro do usuário.

7.5 Persistência de dados

7.5.1 Escolha da tecnologia

Para o banco de dados, foi escolhido o MySQL devido seu fator OpenSource. Uma vez que o projeto é um MVP que não possui fins lucrativos, o grupo optou por um recurso gratuito que atenda as necessidades de demanda e escalabilidade do projeto, tornando o MySQL uma opção adequada.

7.5.2 Modelagem

A modelagem do domínio da aplicação foi pensada focando em explorar os conceitos de especialização e generalização, com o objetivo de permitir que o sistema seja flexível e escalável, tanto em termos de dados quanto em termos de funcionalidades. Essa necessidade vem do fato da aplicação implementar diversos tipos de conteúdos e exercícios, podendo haver ainda a necessidade no futuro de aumentar ainda mais essa diversidade.

O diagrama entidade-relacionamento da figura 39 descreve o modelo de banco de dados da aplicação. As responsabilidades de cada uma das tabelas e dos relacionamentos serão abordados nas próximas subseções.

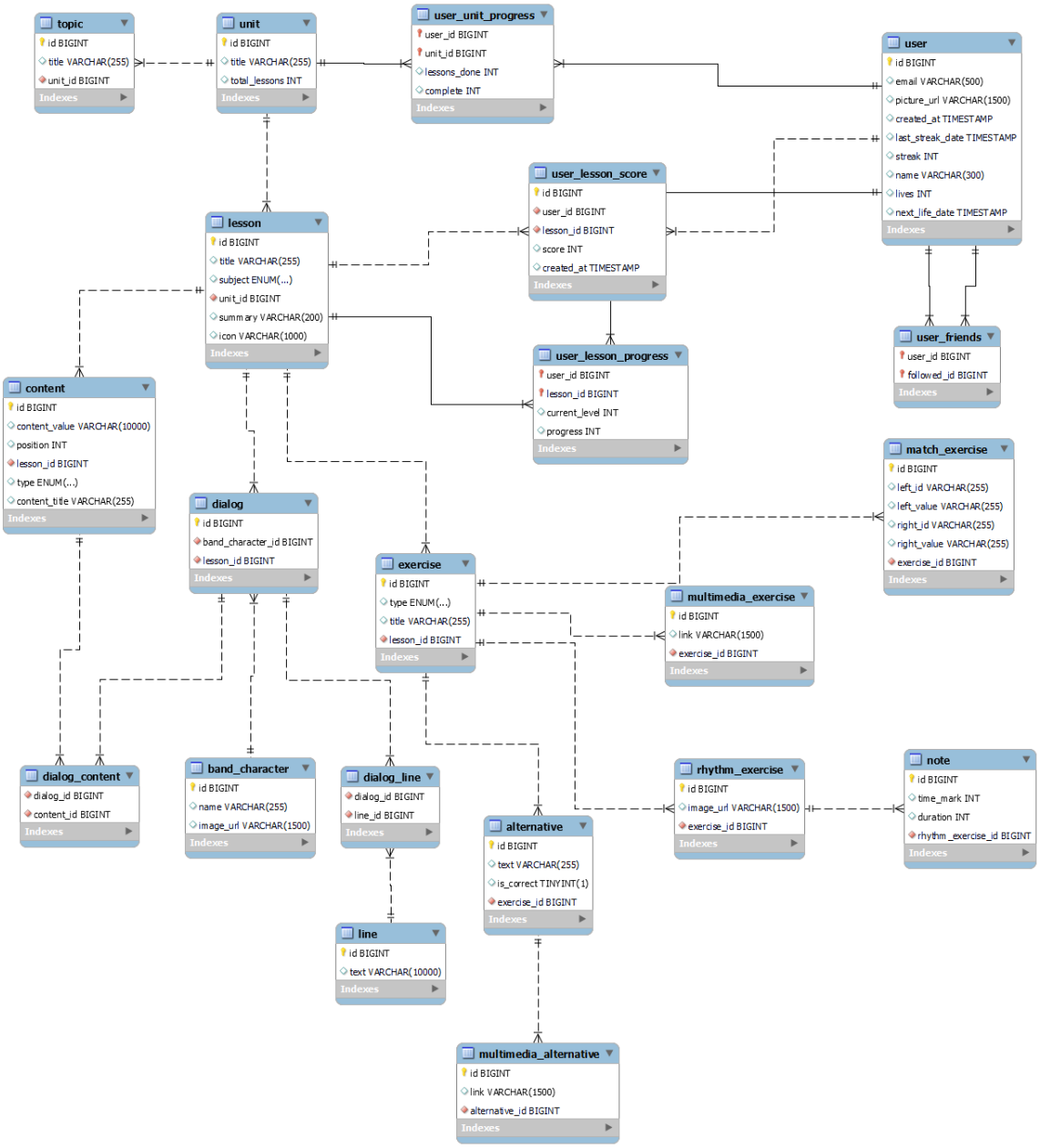


Figura 39 – Diagrama entidade-relacionamento

7.5.2.1 Usuário

A tabela *User* representa os usuários da plataforma. Cada usuário tem atributos para armazenar informações pessoais e de contato, além de manter registros de atividades, como datas de criação e última interação. A relação com *User_friends* representa a conexão entre os usuários, que podem seguir e serem seguidos por outros usuários.

7.5.2.2 Progresso do usuário

As entidades *User_unit_progress*, *User_lesson_score*, e *User_lesson_progress* rastreiam o progresso do usuário através das unidades e lições, armazenando informações

sobre a conclusão das unidades, pontuações nas lições e o progresso atual nas lições, respectivamente.

7.5.2.3 Aulas e conteúdos

As tabelas *Topic*, *Unit*, *Lesson* e *Content* formam a estrutura curricular da aplicação. *Unit* é a entidade de mais alto nível, representando unidades de ensino, que agrupa as aulas representadas pela tabela *Lesson*, e possui tópicos para descrever de forma resumida o conteúdo da unidade. Cada unidade pode conter múltiplas aulas, e cada aula pode ter diversos conteúdos (*Content*) associados, que são os materiais de aprendizado propriamente ditos, incluindo texto e recursos multimídia.

7.5.2.4 Exercícios

Esta entidade é ligada a *Lesson* e representa exercícios que os usuários devem realizar. As relações com *Multimedia_exercise*, *Match_exercise*, e *Rhythm_exercise* representam os diferentes tipos de exercícios disponíveis.

7.5.2.5 Diálogos

Os diálogos dentro das lições são representados na tabela *Dialog*, com personagens (*Band_character*) e falas (*Dialog_line*).

7.6 Backend

7.6.1 Escolha da tecnologia

Para o desenvolvimento do *backend*, foi escolhido utilizar Kotlin com o *framework* Spring Boot. O Kotlin tem como características principais sua concisão, expressividade e sintaxe clara, possibilitando o desenvolvimento de códigos mais claros e eficientes, o que também facilita a manutenção. Tem compatibilidade total com Java, o que permite integrar facilmente bibliotecas externas. O desempenho também é semelhante ao Java, mas conta com recursos mais modernos como como lambdas, funções de extensão e *null safety*.

O Spring Boot permite a implementação de APIs em nível de produção com necessidade mínima de configuração (WHY..., 2023), pois conta com recursos como configuração automática de servidor web e de bibliotecas externas adicionadas ao projeto, além de funcionalidades prontas para produção como métricas e *health checks*. Dessa forma, a utilização desse *framework* permite o desenvolvimento de um código mais conciso e eficaz, diminuindo a quantidade de código *boilerplate*.

Foi utilizado também o *framework* Hibernate, que tem como objetivo realizar o mapeamento objeto-relacional, ou seja, mapear as classes da aplicação com as tabelas do banco de dados. O Hibernate fornece uma camada de abstração que substitui a escrita de SQL puro, permitindo que toda a lógica de interação com o banco de dados seja feita através do código orientado a objetos. Isso garante uma robustez à aplicação, já que problemas na lógica de acesso ou escrita de dados podem ser detectados em tempo de compilação, enquanto se tivéssemos utilizando SQL diretamente, os problemas seriam notados apenas na execução. Além disso, o *framework* traz outras vantagens, como facilidade de troca de tecnologia do banco de dados, sem ter necessidade de alterar o código para isso; *lazy loading*, que permite que as relações entre os objetos sejam carregados de acordo com a necessidade do uso, otimizando o acesso ao banco de dados; entre outras.

7.6.2 Arquitetura

Foi definido o modelo de arquitetura MVCS para estruturar o *backend*. Este modelo (KUMAR, 2021) consiste na criação de quatro componentes base: Model, View, Controller e Service. O MVCS permite uma separação clara entre as camadas controladora e regras de negócio, deste modo se torna simples e coerente a integração com a arquitetura BLoC vista na seção 7.4.2. Além disto alterações na camada controladora não afetam as camadas de negócio, blindando a modelagem e as regras de negócio já definidas que não devem ser alterados conforme a expansão do projeto. A figura 40 é um diagrama que apresenta a troca de informações entre as quatro camadas do MVCS.

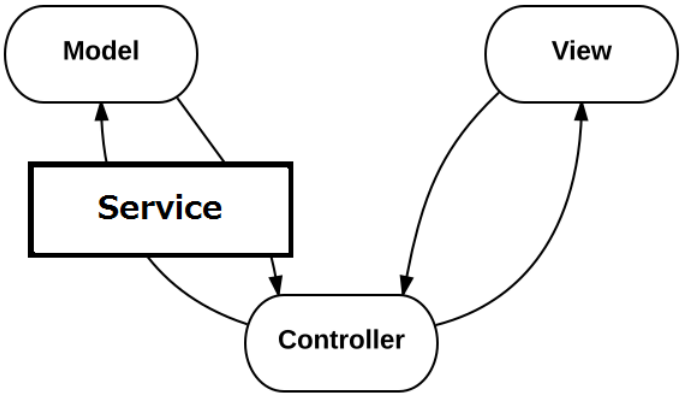


Figura 40 – Arquitetura MVCS

Podemos entender os quatro principais componentes desta arquitetura da seguinte maneira:

- **Model**: Responsável por determinar os modelos de dados do domínio;
- **Service**: Camada de serviço, onde as regras de negócio para o processamento das informações fica concentrada;

- **Controller**: Esta camada recebe os dados enviados pelo *frontend*, e responde com os dados tratados, utilizando os serviços para aplicar as regras de negócio necessárias.

- **View**: tipicamente representa a renderização dos dados para o usuário final. Com o *frontend* desacoplado, a própria aplicação do *frontend* assumirá o papel desta camada.

Juntamente com o padrão MVCS, foi utilizado o *design pattern* Data Mapper, em que a aplicação tem uma camada de dados separada da estrutura do banco de dados. De forma complementar a esse padrão de projeto, também foi utilizado o *design pattern* Repository, que é definido pelo isolamento da camada de acesso aos dados. Para a implementação destes dois padrões, foi utilizado o *framework* Hibernate, que permite a criação de classes com anotação de entidade, representando as tabelas do banco de dados na aplicação, e também fornece anotação para a criação facilitada de repositórios.

Esta arquitetura foi implementada com a separação das classes em pacotes, de acordo com a seguinte estrutura:

```
/
├── config
├── controller
├── dto
├── entity
├── enum
├── exception
├── filter
├── mapper
├── repository
├── service
└── util
```

Cada pacote agrupa classes com funcionalidades parecidas. Dentro dos pacotes, cada classe representa aquela funcionalidade para uma parte do modelo. Cada pacote tem como responsabilidade:

- **Config**: Classes de configuração da aplicação, como políticas de tratamento de erros, autenticação, etc;

- **Controller**: Representa a camada Controller do MVC, que são responsáveis por lidar com as solicitações HTTP e processar dados. No padrão MVC, o controlador age como um intermediário entre o modelo (dados) e a visualização (a apresentação, neste caso o aplicativo Android);

- **DTO**: Sigla de *Data Transfer Object*, este pacote contém as classes que são usadas para transferir dados entre processos, neste caso, entre a API e o aplicativo Android consumidor. Essas classes são usadas para encapsular os dados e separar a camada de serviço da camada de apresentação. Representa a camada Model do MVC;
- **Entity**: Representa os modelos de domínio no contexto da aplicação, através de classes de entidade;
- **Enum**: Classes de enumeração, utilizadas para criar variáveis que admitem como valor um conjunto de constantes predefinidas;
- **Exception**: Classes de exceção personalizadas;
- **Filter**: Filtros a serem utilizados na busca de informações no banco de dados;
- **Mapper**: Classes responsáveis por mapear entidades e objetos DTO, a fim de transformar os dados obtidos na camada de persistência para o formato esperado pela camada de transferência de dados;
- **Repository**: Os repositórios são interfaces utilizadas pelo Spring Data JPA para interagir com o banco de dados. Este pacote representa a camada de dados dentro da aplicação, permitindo operações SQL;
- **Service**: Representa a camada Service do MVCS, contendo as classes que encapsulam a lógica de negócios da aplicação.
- **Util**: Classes auxiliares que fornecem funcionalidades comuns que podem ser reutilizadas em diferentes partes do projeto.

Para ilustrar o fluxo de dados dentro da aplicação e entre os diferentes pacotes, pode-se observar no diagrama da figura 41 o tratamento de uma requisição HTTP do tipo GET:

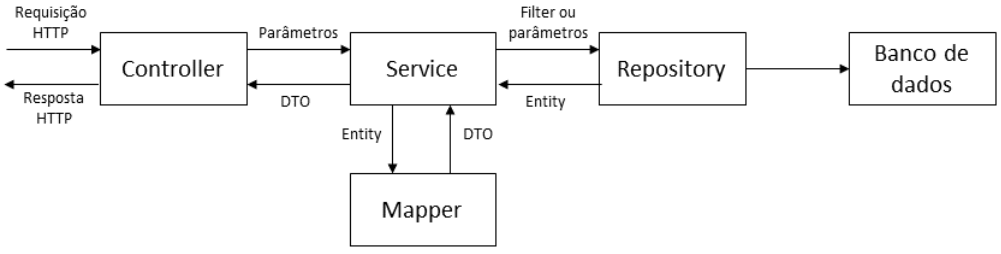


Figura 41 – Fluxo de dados para uma requisição HTTP GET

A Controller recebe a requisição HTTP e envia os parâmetros para a Service. A Service, por sua vez, utiliza os filtros ou os próprios parâmetros recebidos em caso de

consultas mais simples (como por *id*) para consultar os dados através do Repository, que faz de fato a conexão com o banco de dados. Como resposta, a Service obtém uma entidade, ou uma lista de entidades, e utiliza o Mapper para criar a resposta que será enviada para Controller. A Controller, enfim, recebe um DTO da Service e o envia como resposta da requisição.

No diagrama da figura 42, pode-se observar o funcionamento de uma requisição do tipo POST. A Controller recebe uma requisição e o Spring Boot transforma o corpo desta requisição na DTO definida pelo contrato da API, através das anotações em código. Então, a Controller envia esse DTO para Service, que utiliza o Mapper para convertê-lo em uma entidade. A entidade então é enviada para o Repository, que salva os dados da entidade no banco de dados.

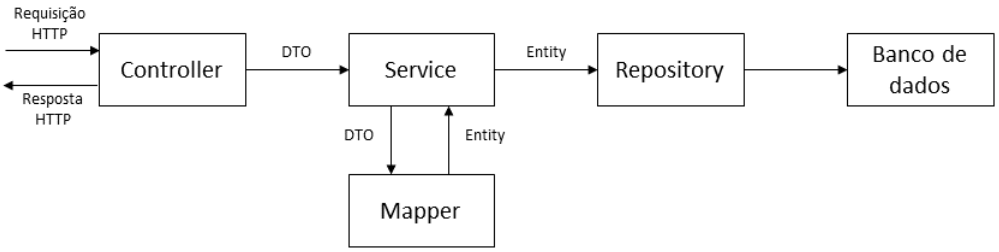


Figura 42 – Fluxo de dados para uma requisição HTTP POST

7.6.3 Tratamento de exceções

O Spring Boot fornece a anotação *ControllerAdvice* para criar classes de tratamento personalizado e global de exceções no contexto das Controllers. Nesta classe, cada método implementa um tratamento padrão para um exceção ou um grupo de exceções, e define uma resposta HTTP para cada caso.

A aplicação tem quatro principais tratamento de exceções, sendo eles:

- **Exceções de autenticação**: *token* expirado ou inválido;
- **Exceções de argumentos inválidos**: são lançadas quando um *endpoint* não recebe o corpo da requisição da forma esperada. Além disso, o mesmo tipo de exceção é lançada pelo SDK do Firebase quando o *token* de autenticação recebido não é válido, seja por estar nulo ou por estar em formato diferente do esperado.
- **Exceções de registros não encontrados no banco de dados**: exceções lançadas quando uma consulta não retorna dados do banco de dados;
- **Exceção de vidas insuficentes**: exceção lançada quando o *endpoint* de perda de vida é chamado com um número maior de vidas a serem perdidas do que o que o usuário possui.

7.6.4 Autenticação

Conforme discutido na seção de serviços externos (7.3) e na seção de integração do *frontend* (7.4.4), foi utilizado o Google Firebase como sistema de autenticação. O *login* e geração de *token* é realizado no *frontend*, que repassa o *token* para o *backend* nas requisições HTTP.

No *backend*, foi implementado um interceptor de requisições HTTP, a partir da interface *HandlerInterceptor* do Spring Boot. Utilizando a interface *WebMvcConfigurer*, adiciona-se esse interceptor na configuração da aplicação, com a opção de personalizar os caminhos da API que serão interceptados por essa classe. O interceptor verifica a existência do *token* no *header* da aplicação e passa esse *token* para o SDK do Firebase, que valida o *token* e recupera o usuário associado a este *token*. Caso o *token* não seja válido, esteja expirado ou ausente, o interceptador gera uma resposta de autenticação negada e encerra o processamento da requisição, evitando que a requisição chegue às Controllers.

7.6.5 Banco de dados

Como citado na seção de escolha de tecnologia (7.6.1), optou-se por utilizar o Hibernate como Object-Relational Mapping (ORM) para mapear as entidades da aplicação para tabelas no banco de dados. Isso abstrai a complexidade do gerenciamento direto das operações SQL, permitindo que a aplicação interaja com o banco de dados usando objetos em Kotlin.

A conexão com o banco de dados é gerenciada pelo Hibernate, utilizando o Driver JDBC correspondente ao MySQL. A conexão é encapsulada por objetos do tipo *Session*, que representam uma unidade de trabalho com o banco de dados, fornecendo métodos para executar consultas e outras transações. As sessões são criadas a partir de outra classe do Hibernate, a *SessionFactory*, que cria as conexões de acordo com as configurações fornecidas pela aplicação.

Para gerenciar o esquema do banco de dados, utilizou-se a biblioteca Liquibase, que fornece as funcionalidades de versionamento, rastreamento e gerenciamento do esquema de banco de dados. A utilização do Liquibase garante que qualquer mudança no esquema do banco de dados seja rastreada, versionada e aplicada de maneira consistente em todos os ambientes de desenvolvimento, teste e produção. As migrações do banco de dados são definidas em arquivos de configuração controlados pelo Liquibase, que são executados automaticamente durante a inicialização da aplicação.

7.7 Infraestrutura

Para a construção da infraestrutura da aplicação foram utilizados serviços da AWS, devido a sua grande documentação, tanto da Amazon quanto da comunidade e também seu baixo custo quando comparado a outros serviços como o Google Cloud, tendo a maioria dos serviços utilizados funcionando no *free tier* para a construção do MVP. Outro fator a ser citado é que seus serviços tem compatibilidade entre si, facilitando a implementação de mais de um serviço de uma vez só.

7.7.1 Serviços utilizados

- IAM

  Serviço de gerenciamento de permissões de usuários e ferramentas na AWS.

- CloudWatch

  Serviço de monitoramento e geração de logs para otimização de serviços.

- CodePipeline

  Serviço de automatização do processo de *pipelines* de lançamento para poder atualizar as aplicações de forma rápida e confiável.

- CodeCommit

  Serviço de versionamento. Foi escolhido por sua integração com o CodePipeline, sendo atualizações na *branch* master automaticamente traduzidas no produto final.

- CodeBuild

  Serviço de integração e compilação do código fonte. Foi escolhido por fazer parte do processo do CodePipeline, podendo ser usado para tanto para construção de APKs, quanto para atualização do funcionamento dos servidores.

- CodeDeploy

  Serviço de automatização de implantação de software. Utilizado para eliminar a necessidade de operações manuais e atualizar serviços utilizados. Foi escolhido por ser integrado ao CodePipeline, podendo prover a implantação do *backend* no Elastic Beanstalk.

- Elastic Beanstalk

  Serviço de implantação de aplicações *web*. Foi escolhido por sua facilidade de implantação, pela escalabilidade automática que o serviço provém, pela sua compatibilidade com outros serviços da AWS e a forma de cobrança baseada em recursos usados, o que é ideal para uma aplicação ainda pequena como a realizada no trabalho.

- S3

  Serviço de armazenamento de objetos da AWS com ampla gama de utilidades. É utilizado por padrão em vários dos outros serviços escolhidos, porém seu uso direto será destinado aos *assets* do aplicativo.

- RDS

  Serviço de gerenciamento de banco de dados. Sua escolha foi feita devido a sua fácil integração com o Elastic Beanstalk além de ser o principal serviço destinado a esse fim.

- ECR Serviço de registro de *containers* que oferece hospedagem para implantação de imagens. Sua escolha se deve ao ambiente necessário para construir o *frontend*.

7.7.2 Implementação

7.7.2.1 IAM

O primeiro passo para dar início ao processo de desenvolvimento na AWS foi a criação dos usuários de cada integrante do grupo, associando a cada conta as permissões necessárias dos demais serviços utilizados.

Além dos usuários, o serviço também é utilizado para configurar as permissões de outras ferramentas que terão que interagir entre si para o funcionamento completo da aplicação, como acesso do CodeBuild à imagens de ECR.

7.7.2.2 CloudWatch

O CloudWatch é o serviço que monitora o funcionamento de todos os outros, sendo responsável também pela criação dos *logs* de cada um. É a partir dele que o *pipeline* é avisado de alterações na *branch* master do projeto e que se inicia automaticamente o processo de *build*. Além disso, todas as vezes em que houve algum erro na implementação da aplicação ou construção do APK, foi por ele que se pode verificar os erros e corrigi-los.

7.7.2.3 CodePipeline

Para acelerar o processo de desenvolvimento, foram configurados dois *pipelines* de desenvolvimento tanto para *frontend* e *backend*. Esses *pipelines* consistem de repositórios do CodeCommit, projetos de compilação do CodeBuild e uma aplicação do Elastic Beanstalk para *deploy* do projeto.

7.7.2.4 CodeCommit

No CodeCommit são gerenciados os versionamentos, tendo um repositório para o *frontend* e outro para o *backend*. É importante para a fase de *build* que os arquivos

com as especificações de compilação estejam corretamente configurados nos respectivos repositórios.

No repositório do Frontend, o arquivo de configuração é um Dockerfile que contém as configurações do ambiente que será utilizado posteriormente na fase de *build*. No caso, é um ambiente em Debian que instala as bibliotecas necessárias, a versão do Flutter utilizada e suas dependências, e roda os comandos para construção de um arquivo APK.

No repositório do *backend*, o arquivo é um YAML com nome *buildspec* que define a versão do java utilizada pela aplicação no Elastic Beanstalk, os comandos de *build* e a localização dos artefatos utilizados.

7.7.2.5 CodeBuild

O CodeBuild é utilizado de maneiras diferentes para as diferentes frentes do serviço. No *frontend*, o código é extraído do repositório do CodeCommit e seu arquivo Dockerfile é utilizado para rodar um *container* que gerará o APK da aplicação, esse por sua vez que é guardado no *bucket* do S3 referente o *pipeline* em que se encontra essa etapa. A imagem do *container* utilizada para essa operação é guardada no ECR.

No *backend*, o código é extraído do repositório do CodeCommit e seu arquivo buildspec indica para o ElasticBeanstalk em as configurações em que a aplicação deverá ser rodada junto com os comandos de instalação dos pacotes.

7.7.2.6 ECR

No ECR é registrada a imagem utilizada na fase de implantação do *frontend*, onde será realizado a construção do arquivo APK.

7.7.2.7 CodeDeploy/ElasticBeastalk

Essa etapa do *pipeline* só é definida para o *backend*, uma vez que somente ele fica rodando continuamente em uma máquina na nuvem. Consiste em definir a aplicação com o que foi repassado pela etapa de *build*, além de configurar também o monitoramento e funcionamento dos *logs*.

7.7.2.8 S3

No S3 são guardados *assets* do projeto, além de artefatos de configuração dos ambientes e *logs* de erro.

7.7.2.9 RDS

O RDS é utilizado para hospedar o banco de dados do serviço. Ele é criado junto à criação do *pipeline* do *backend*, podendo ser acessado pelo código instanciado livremente.

7.7.3 Funcionamento dos *pipelines*

7.7.3.1 Frontend

O fluxo do *pipeline* do *frontend* segue com a utilização do código guardado no repositório do CodeCommit, especificamente na *branch* master, para construir o APK do aplicativo na fase do CodeBuild. Nesta fase, é utilizado o serviço ECR para obter a imagem do *container* necessária para execução do código do aplicativo e o Dockerfile. Terminado a fase de geração do arquivo, ele é guardado em um *bucket* no S3, de onde pode ser acessado, baixado e testado. Caso hajam mudanças no código, o *pipeline* se reinicia e um novo arquivo de instalação é gerado e salvo no bucket.

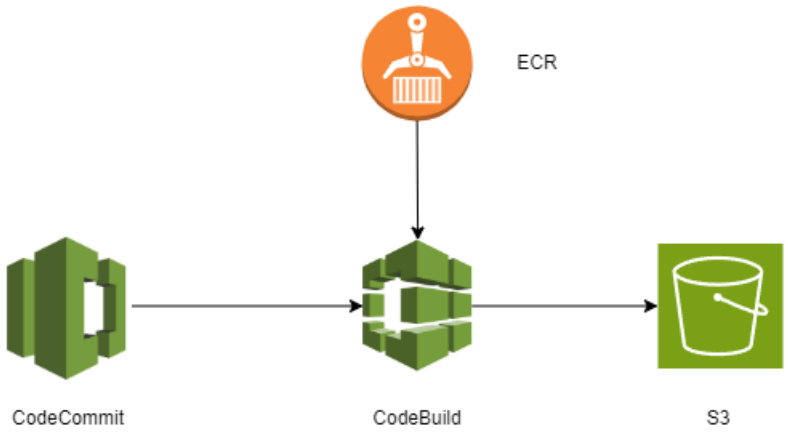


Figura 43 – Fluxo do *pipeline* do *frontend*

7.7.3.2 Backend

O fluxo do *pipeline* do *backend* segue com a utilização do código guardado no repositório do CodeCommit, especificamente na *branch* master, para configurar e determinar o funcionamento do aplicativo na fase do CodeBuild. Nesta fase, é utilizado o arquivo YAML para configurar o ambiente com os pacotes e versões necessários para o funcionamento. Terminado a fase de *build*, a aplicação no Elastic Beanstalk é configurada ou criada, caso não exista. O Elastic Beanstalk por sua vez realiza o *hosting* da aplicação, gerenciando seus recursos. Na sua criação, também é configurada uma instância do RDS com um banco de dados acessível pela aplicação vigente.

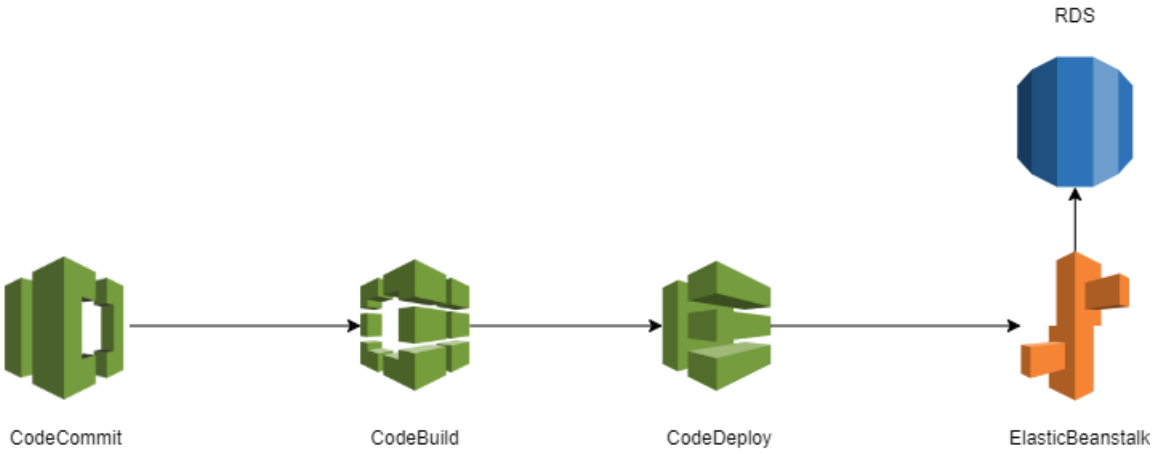


Figura 44 – Fluxo do *pipeline* do *backend*

Caso o código seja atualizado, o *pipeline* se encarga de atualizar a versão da aplicação rodando no Elastic Beanstalk.

8 Considerações Finais

Serão abordados abaixo os resultados finais do projeto, sendo citados os resultados atingidos e não atingidos. Também serão citados passos para a continuidade de solução.

8.1 Conclusões do Projeto de Formatura

Tinha-se como objetivo inicial do trabalho a criação de um MVP com algumas aula e diferentes tipos de exercício disponibilizados. Era necessário que o conteúdo fosse explicado de maneira simples e bem divida entre os módulos, de forma a evitar desânimo da parte do usuário geralmente associado ao aprendizado de teoria musical. Além disso, os exercícios elaborados deveriam cumprir seu papel de proverem o máximo de interatividade possível, fugindo do formato tradicional de perguntas e respostas.

Dito isso, pode-se argumentar que o projeto foi além das expectativas ao implementar efetivamente mecânicas de jogos visando o engajamento do usuário e chegar muito próximo do funcionamento de um produto final, sendo a escrita dos módulos de aula faltantes e uma possível monetização os únicos pontos faltantes para o lançamento de uma versão inicial simplificada do aplicativo.

Vale destacar ainda que com sua arquitetura e implementação atual, o projeto é altamente escalável, sendo possível servir um público inicial de jogadores de forma plena e expandir em conjunto com a evolução do público.

8.2 Contribuições

A contribuição do trabalho foi em grande parte a solução de uma dor de estudantes e professores de música que, até o momento, não tinham métodos de reforço de ensino fora da sala de aula eficazes. Além da possibilidade de difundir o interesse na área de teoria musical para uma gama maior de pessoas, conscientizando sobre a importância da área e deixando esse conhecimento mais acessível.

8.3 Perspectivas de Continuidade

O aplicativo foi projetado de modo que a escalabilidade fosse um ponto de atenção. Deste modo, o primeiro ponto de continuidade observado é a construção dos materiais didáticos, apoiado por profissionais qualificados para o ensino de música. É sugerido que estes materiais utilizem as ferramentas já inclusas na aplicação, entretanto, dado o alto

potencial de escalabilidade da aplicação de ponta a ponta, é possível adicionar novos modelos de exercícios sem prejudicar a estrutura existente.

Além disto, um sistema de pontuação mais robusto é visto como uma melhoria de grande valor. A distribuição de pontos por exercício pode ser reavaliada de modo a distribuir uma pontuação mais coerente com a dificuldade que cada modelo de exercício traz. Também é de grande valia proporcionar um ambiente competitivo para além das comparações estatísticas, recompensando usuários com maior taxa de pontuação em um intervalo de tempo e atribuindo *ranks* aos usuários.

Para a página de estatística é possível ampliar as métricas extraídas para auxiliar nas dificuldades do usuário. Assim, um modelo de recomendação de aulas pode ser de grande valor se entendido como uma necessidade do usuário.

No que tange a testes, a infraestrutura pode ser validada através de testes de estresse, assumindo o crescimento do aplicativo e aumento expressivo no número de usuários simultâneos. Também podem ser adicionados testes unitários no *backend*, testes de componentes no *frontend* e testes E2E(*End to End*).

Uma implementação desejada, mas não concluída é de notificações do aplicativo como lembrete diário para o usuário. Deste modo, o risco de perder o *streak* é reduzido, o que garante a motivação do usuário.

Por último, podemos citar uma melhora na validação dos dados recebidos pelo *backend*. Optamos por não priorizar este ponto neste trabalho pois o consumo dos *endpoints* do *backend* é feito apenas aplicação do *frontend*, e não tem contato com o usuário final, diminuindo a necessidade de mensagens de erro mais trabalhadas. Porém, ainda é algo que agregaria valor à aplicação, dado que a validação dos dados e implementação de mensagens de erro mais completas ajudam no *debug* em caso de problemas,

Referências

ARAÚJO, C. et al. Design thinking versus design sprint: A comparative study. In: _____. *Design, User Experience, and Usability. Design Philosophy and Theory*. [S.l.: s.n.], 2019. p. 291–306. ISBN 978-3-030-23569-7. Citado na página 21.

FLUTTER Documentation. 2023. Disponível em: <https://docs.flutter.dev/>. Citado na página 58.

FURLAN, L. P.; FONTERRADA, M. T. de O. O processo inicial do desenvolvimento da habilidade de leitura e escrita musical: Um paralelo com a psicogênese da língua escrita. *ANPPOM*, v. 15, 2005. Citado na página 25.

HUIZINGA, J. *Homo Ludens*. 3. ed. Rio de Janeiro: Editora Perspectiva, 2005. Acesso em: 21 ago 2013. Citado na página 19.

INTRODUCTION to Widgets. 2023. Disponível em: <https://docs.flutter.dev/development/ui/widgets-intro>. Citado na página 58.

KUMAR, A. *An Introduction To MVCS Architecture*. 2021. Disponível em: <https://quantiphi.com/an-introduction-to-mvcs-architecture/>. Citado na página 65.

NETWORKING. 2023. Disponível em: <https://docs.flutter.dev/development/data-and-backend/networking>. Citado na página 58.

SOARES, P. *Flutter / AngularDart – Code sharing, better together (DartConf 2018)*. 2018. Disponível em: <https://www.youtube.com/watch?v=PLHln7wHgPE>. Citado na página 58.

SOUZA, J. V. Sobre as múltiplas formas de ler e escrever música. In: _____. *Ler e escrever: compromisso de todas as áreas*. Porto Alegre: [s.n.], 2006. Citado na página 25.

WHY Spring? 2023. Disponível em: <https://spring.io/why-spring>. Citado na página 64.