INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO
CÂMPUS CAMPINAS

RAFAEL DUARTE DE MEDEIROS

SISTEMA DE GERENCIAMENTO DE ESTUDOS PARA DESENVOLVIMENTO E MANUTENÇÃO TÉCNICA DE GUITARRISTAS

CAMPINAS
2021

RAFAEL DUARTE DE MEDEIROS

SISTEMA DE GERENCIAMENTO DE ESTUDOS PARA DESENVOLVIMENTO E MANUTENÇÃO TÉCNICA DE GUITARRISTAS

Trabalho de Conclusão de Curso apresentado como exigência parcial para obtenção do diploma do Curso de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia Câmpus Campinas.

Orientador: Prof. Dr. Andreiwid Sheffer Corrêa

CAMPINAS

2021

Ficha catalográfica
Instituto Federal de São Paulo – Câmpus Campinas
Biblioteca
Rosana Gomes – CRB 8/8733

Medeiros, Rafael Duarte de
M488s Sistema de gerenciamento de estudos para desenvolvimento e manutenção técnica de guitarras / Rafael Duarte de Medeiros. – Campinas, SP: [s.n.], 2021.
101 f. il. :

Orientador: Andreiwid Sheffer Corrêa.
Trabalho de Conclusão de Curso (graduação) – Instituto Federal de Educação, Ciência e Tecnologia de São Paulo Câmpus Campinas. Curso de Tecnologia em Análise e Desenvolvimento de Sistemas, 2021.

1. Guitarra (Instrução e estudo – métodos). 2. Aplicações web. I. Instituto Federal de Educação, Ciência e Tecnologia de São Paulo Câmpus Campinas. Curso de Tecnologia em Análise e Desenvolvimento de Sistemas. II. Título.

ATA 9/2021 - TADS-CMP/DAE-CMP/DRG/CMP/IFSP

Ata de Defesa de Trabalho de Conclusão de Curso - Graduação

Na presente data, realizou-se a sessão pública de defesa do Trabalho de Conclusão de Curso intitulado **SISTEMA DE GERENCIAMENTO DE ESTUDOS PARA DESENVOLVIMENTO MANUTENÇÃO TÉCNICA DE GUITARRISTAS**, apresentado(a) pelo(a) aluno(a) **Rafael Duarte de Medeiros (CP3000117)** do Curso **SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS** (Câmpus Campinas). Os trabalhos foram iniciados às 10h00 pelo(a) Professor(a) presidente da banca examinadora, constituída pelos seguintes membros:

Membros	Instituição	Presença (Sim/Não)
Prof. Dr. Andreiwid Sheffer Corrêa (Presidente/Orientador)	IFSP	Sim
Profa. Dra. Bianca Maria Pedrosa (Examinadora 1)	IFSP	Sim
Prof. Me. Fábio Feliciano de Oliveira (Examinador 2)	IFSP	Sim

Observações:

A banca examinadora, tendo terminado a apresentação do conteúdo da monografia, passou à arguição do(a) candidato(a). Em seguida, os examinadores reuniram-se para avaliação e deram o parecer final sobre o trabalho apresentado pelo(a) aluno(a), tendo sido atribuído o seguinte resultado:

[X] Aprovado(a) [ ] Reprovado(a)

Proclamados os resultados pelo presidente da banca examinadora, foram encerrados os trabalhos e, para constar, eu lavrei a presente ata que assino em nome dos demais membros da banca examinadora.

Câmpus Campinas, 11 de junho de 2021

Documento assinado eletronicamente por:

- **Andreiwid Sheffer Correa**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 11/06/2021 11:39:47.

Este documento foi emitido pelo SUAP em 11/06/2021. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/autenticar-documento/ e forneça os dados abaixo:

Código Verificador: 192849
Código de Autenticação: 1a07b43c5e

ATA 9/2021 - TADS-CMP/DAE-CMP/DRG/CMP/IFSP

Dedico este trabalho aos meus familiares,
colegas de classe, professores e servidores do Instituto
que colaboraram em minha jornada formativa.

AGRADECIMENTOS

Agradeço a todos os professores e servidores do IFSP Câmpus Campinas, que contribuíram direta e indiretamente para a conclusão desse trabalho.
Agradeço também à minha família, que deu todo o apoio necessário para que eu chegasse até aqui.
Agradeço ao meu orientador que me auxiliou a solucionar as dificuldades encontradas no caminho.

"Se não fosse físico, acho que seria músico.
Eu penso em termos de música.
Vejo minha vida em termos de música".
Albert Einstein

"A música é o vínculo que une a vida do espírito à vida dos sentidos.
A melodia é a vida sensível da poesia".
Ludwig van Beethoven

"Sem a música, a vida seria um erro".
Friedrich Nietzsche

RESUMO

O desenvolvimento técnico na guitarra elétrica sempre foi um desafio enorme, cheio de obstáculos e batalhas pessoais. Entretanto, com o passar do tempo, tamanha dedicação presenteia aqueles que não resignam seus estudos com a sensação única e prazer de executar técnicas fascinantes com elevada proficiência. Para que se desenvolva um determinado nível técnico, necessita-se de dedicação contínua e uma organização de estudos criada de forma primorosa. Com o tempo um guitarrista tende a perder o senso de evolução e isso pode fazê-lo perder sua automotivação. Outro fato dificultador é em razão de grande maioria dos guitarristas não ter a música como principal atividade remunerada, fazendo que eles tenham uma rotina dupla. O trabalho desenvolvido tem como objetivo principal auxiliar guitarristas no gerenciamento dinâmico de seus estudos, se baseando no entendimento de seu universo através de entrevistas com um grupo experiente e buscar equiparação com teorias de aprendizado e retenção motora. Assim, consegue-se fazer com que guitarristas tenham um melhor e mais objetivo aproveitamento do tempo disponível para prática do instrumento. O sistema contempla também a visualização de aproveitamento em diferentes perspectivas, para que o guitarrista entenda melhor sua rotina e que isso possa refletir em sua automotivação. Tal trabalho consiste em um sistema web, tendo como principais ferramentas no seu desenvolvimento .Net Core e React, e foi desenvolvido utilizando os princípios de *Clean Architecture* proposto por Robert C. Martin.

Palavras-chave: Estudo instrumento musical. Desenvolvimento técnico guitarra. Organização de estudos musicais. Aprendizado motor. Sistemas web. Arquitetura Limpa. SOLID. Arquitetura de Software.

ABSTRACT

Developing technique on the electric guitar has always been a huge challenge, full of obstacles and personal battles. However, over time, such dedication presents those who do not resign their studies with the unique feeling and pleasure to performing fascinating techniques with high proficiency. Developing a certain technical level, requires continuous dedication and a well-designed study organization. Over time a guitarist tends to lose his sense of evolution, and this can make him lose his self-motivation, another difficult is because many guitarists do not have music as their main activity, making them to have a double routine. The main objective of this work is to assist guitarists in the dynamic management of their studies, based on the understanding of their unique universe through interviews of an experienced group and seeking to match theories of motor learning. Thus, being able to make guitarist to have a better and more objective use of the time available to practice the instrument. The system also includes the visualization of performance from different perspectives, so that the guitarist can better understand his routine and that can reflect on his self-motivation. This work consists of a web system, having as main tools .Net Core and React and was developed using the concepts of Clean Architecture proposed by Robert C. Martin.

Keywords: Study of musical instrument. Guitar technical development. Music Studies organization. Motor learning. Web systems. Clean Architecture. SOLID. Software Architecture.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

séc.	século
seg.	segunda-feira
ter.	terça-feira
qua.	quarta-feira
qui.	quinta-feira
sex.	sexta-feira
sáb.	sábado
dom.	domingo

LISTA DE SIGLAS

IFSP	Instituto Federal de Educação, Ciência e Tecnologia de São Paulo
BPM	Batidas Por Minuto
MIT	Massachusetts Institute of Technology
CLI	Command Line Interface
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
LINQ	Language INtegrated Query
ADO	ActiveX Data Objects
DDD	Domain-Driven Design
CQRS	Command Query Responsibility Segregation
GUID	Globally Unique Identifier
ORM	Object–Relational Mapping
ID	Identity Document
SGBD	Sistema de Gerenciamento de Banco de Dados
JWT	JSON Web Token
REST	REpresentational State Transfer
SPA	Single-Page Application
MVC	Model–View–Controller
UML	Unified Modeling Language
MIDI	Musical Instrument Digital Interface

LISTA DE SÍMBOLOS

ex	exemplo

SUMÁRIO

1 INTRODUÇÃO

Desde os primórdios da música, o ser humano foi fascinado pelo desenvolvimento técnico em um instrumento musical, seja pela busca de timbres mais refinados e de características únicas, que são consequência de uma execução precisa e segura, como também fazer o uso de melodias e sensações musicais apenas atingidas por execuções de uma sequência de escalas e arpejos em grandes velocidades, sem a degradação das características de uma boa execução.

No período Romântico, no séc. XIX, a diferença entre músicos com domínio técnico fica evidente, o intento técnico passa a ganhar mais seguidores e o mundo começa a apreciar o virtuosismo instrumental na música. Os violinistas e pianistas lideram esse movimento com seus representantes mais aclamados, Nicòlo Paganini e Franz Liszt. Este período coloca um novo limite para o que o estudo disciplinado pode alcançar por músicos instrumentistas (CAVINI, 2012).

Com a intensificação da difusão cultural que ocorreu após a Segunda Guerra Mundial, nas décadas de 60 e 70, emergem diversos segmentos na música popular, aos quais se destacaram o *Rock*, *Pop* e *Heavy Metal*. Segundo Flèchet (2007), os festivais criaram uma nova percepção e interação entre músicos e espectadores, assim originando uma nova forma de divulgação midiática. Tais eventos eram formados por diversas bandas e de formações semelhantes, normalmente compostas por vocalista, um ou dois guitarristas, contrabaixista, baterista, a possível adição de teclados e em alguns casos instrumentos comumente utilizados em música clássica. O foco passa a ser em bandas e todos os instrumentistas ganham espaço de expressão individual e destaque. Junto com esse movimento cresceu o interesse pelo estudo dos instrumentos contidos em tais formações, em grande parte a guitarra elétrica.

Os guitarristas desse período foram astutos quanto a fazer experimentos, buscar a expansão de seus conhecimentos e aplicá-los em suas composições. Muitos deles com formação clássica e vindos de conservatórios, fizeram o uso de elementos incomuns na música que emergia, chegando a adaptar trechos de peças clássicas, experimentação com novos modos, harmonias, melodias e principalmente velocidade com precisão na execução. Essa fusão ficou evidente com Ritchie Blackmore da banda Deep Purple, com diversas adaptações de trechos eruditos, como por exemplo na música *Highway Star (1972)*, que em seu solo é feita a adaptação de um trecho da peça de Antonio Vivaldi, *Violin Concerto in D minor (1725)*. Este período fez com que o intento técnico se intensificasse na guitarra e um novo tipo de virtuosismo contagiou músicos emergentes (WALSER, 1992).

No final da década de 70 e início de 80, influenciado pelo Blues e sua formação em piano e violino clássico, Eddie Van Halen encantou o mundo com execuções em níveis técnicos elevados e musicalidade intensa. Uma das suas técnicas mais marcantes, foi o *Tapping*, que na música *Eruption (1978)*, utilizando elementos harmônicos com repetição de padrões comumente presentes em peças clássicas, nasce um dos trechos mais icônicos e estudados por guitarrista até hoje. Seguindo esse movimento, emerge Randy Rhoads, fortemente influenciado por piano, violão clássico e suas marcantes influências de Blues. Em sua obra *Mr. Crowley (1980)*, em parceira com Ozzy Osbourne, ficou evidente em seu solo o uso de fraseado em Pentatônica proveniente de suas influências de Blues em magnifica conciliação com Arpejos sequencias e padrões de Escalas executados com excelência, ambos característicos de suas referências clássicas e impecável disciplina (WALSER, 1992).

Dentro dos muitos que influenciaram o estudo da guitarra, Yngwie Malmsteen mudou a percepção sobre técnica e limites que um guitarrista pode alcançar, o aclamado álbum *Rising Force (1984)* se torna um marco, onde a busca pelo virtuosismo da música erudita é expressa com Arpejos executados com a técnica *Sweep Picking* sem moderação de velocidade. Tal álbum repleto de harmonias inspiradas na música clássica não esconde suas principais influências resgatadas dos séculos passados. Sua conexão e disciplina incorporada é evidenciada nas dedicações de seu álbum, expressando gratidão a J. S. Bach, Nicolo Paganini, Antonio Vivaldi, Ludwig van Beethoven, Jimi Hendrix e Ritchie Blackmore (DILLARD, 2009; WALSER, 1992).

Depois dos acontecimentos da década de 70 e 80, chamados de *Rock-Neoclassical* e *Heavy-Metal Neoclassical*, a técnica apurada passa a ser um elemento comum, muitas vezes necessário no desenvolvimento de guitarristas que desejam se inserir nos estilos musicais que vieram a emergir, independentes da ligação direta com o *Rock* ou *Heavy-Metal*. Sendo assim, a busca pelo virtuosismo ou uma proficiência de satisfação pessoal, um guitarrista tem a necessidade de reservar tempo ao estudo de seu instrumento e ter regularidade na prática. Em seu estudo, Krampe e Ericsson (1996) observaram que uma rotina semanal de prática de um estudante de um instrumento varia de 5 a 35, horas dependendo do número e nível de técnicas que o mesmo mantêm, tamanho de repertório e objetivo de performance que cada músico possui.

Dentro do período de prática da guitarra, o músico encontra-se diante de um dos seus maiores desafios: definir o que vai aprender, o que vai aprimorar e o que vai manter no nível que está. Estudantes de guitarra que almejam atingir um determinado nível técnico, com o tempo, tendem a aumentar as aplicações técnicas em estado de excelência. Tal processo

demanda organização e exige a confecção de planos de estudos, normalmente feitos em papel, softwares de planilha eletrônica, impressos organizados em pastas, entre outas soluções pessoais (DILLARD, 2007; PETRUCCI, 2000).

O aprendizado motor é o princípio fundamental no desenvolvimento técnico em um instrumento musical, este consiste basicamente em aprimorar movimentos dentro de um intervalo de tempo, que varia conforme a complexidade e sequenciamento de movimentos para a obtenção de um resultado. Tal intervalo de tempo não deve ultrapassar o momento em que determinado sequenciamento começa a perder o nível de performance ao qual já estava estabelecido (MASAKI; SOMMER, 2012).

O objetivo deste trabalho, é oferecer aos guitarristas que necessitam de uma prática controlada, uma ferramenta que organize seu repertório de estudos, possibilitando adição, remoção, acompanhamento de cada estudo em relação a tempo dedicado e frequência de prática, como também oferecer uma análise estatística de dedicação geral. Com isso há o propósito de reduzir o tempo em que um guitarrista investe para gerenciar sua prática, e assim motivá-lo e estabelecer o foco no momento iminente e efêmero, que é tocar o instrumento. Os resultados foram avaliados com a validação dos modelos desenvolvidos utilizando os dados levantados através de entrevistas com um grupo de guitarristas, fazendo a sua equiparação com as funcionalidades implementadas.

2 JUSTIFICATIVA

O desenvolvimento técnico em um instrumento musical é um processo contínuo e longevo. Por se tratar, em sua essência de aprendizado motor e alta complexidade, demanda grande nível de concentração e constância. Muitos músicos que se dedicam a um instrumento musical, tendem a criar um foco elevado em desenvolvimento técnico, que basicamente é a busca pela execução precisa com movimentos curtos, rápidos e o mais confortáveis possíveis. Tal precisão pode ser definida como aproximação da execução de uma nota, com o efeito sonoro da fonte marcadora de andamento, seja um metrônomo ou outro instrumento. Outra característica é a divisão matemática do tempo mais exata possível, como por exemplo a execução de quatro semicolcheias em um compasso com formação 4/4, significa que as quatro notas tocadas devem ter a divisão mais uniforme de tempo dentro de um tempo de marcação, em termos subjetivos a sonoridade de execução é o timbre único de cada músico e também é definido com técnica (PETRUCCI, 2000; STETINA, 1990).

No estudo da guitarra, existem diversas fontes de informações, sendo: livros puramente técnicos com diversos exemplos, livros que trazem um método com desafios gradativos, revistas que expõem exemplos aleatórios, livros que desenvolvem uma técnica especifica dentro de um ou mais estilos, videoaulas com um guitarrista consagrado compartilhando uma fração de seu conhecimento, extração de trechos de músicas ou músicas completas como uma forma de desenvolvimento e a internet que reproduz todos elementos citados de forma digital. A guitarra é um instrumento inserido em diversos estilos musicais, entre os mais conhecidos se destacam o *Blues, Rock, Heavy Metal, Country, Jazz, Folk e Funk*, dentro de cada estilo existem diversas subdivisões, cada variante possui um grupo e modelos de aplicações técnicas, formas diferentes de execução e o fato de usarem equipamentos diferentes, muitas vezes limitam ou criam possibilidades dentro de cada universo. Com isso, podemos dizer que no estudo da guitarra não existe um método definitivo de aprendizado, cada guitarrista agrega aquilo que lhe convém e com o tempo tendem a ter um desenvolvimento único dentro das suas intensões individuais de estudo e nível técnico a ser alcançado, porém todos possuem a necessidade de prática, controle e se manterem motivados (MILARDI, 2004; PETRUCCI, 2000).

Com o tempo, um estudante de guitarra na busca pelo aperfeiçoamento técnico, tende a agregar inúmeros estudos que são coletados e adaptados das diversas fontes citadas anteriormente e desenvolvem fluência em assuntos que lhe convém no nível de interesse individual. Em um modelo simplificado, técnicas são geralmente estudos puramente motores

executados sobre uma complexidade musical simples, que envolvem precisão de movimentos, sincronismo entre as duas mãos e clareza de execução. Com o domínio motor básico estabelecido, ocorre a aplicação e evolução dos movimentos em estruturas musicais cada vez mais complexas, que podem ser: escalas, tríades, tétrades, acordes, entre outras. Em um terceiro estágio, essas aplicações de contextos musicais são transformadas em vocabulário e ocorre o aprofundamento musical. Dentro desse estágio a técnica passa a ser cada vez mais implícita e para que isso ocorra e não tenha barreiras, é necessária toda uma cadeia de desenvolvimento e manutenção motora, neste momento o guitarrista sentirá naturalmente a fluência técnica e será guiado por sua inspiração ao último estágio, a composição e improvisação, o objeto de desejo do guitarrista, onde a música acontece (PETRUCCI, 2000).

Guitarristas com foco em desenvolvimento técnico, organizam sua rotina de estudos através da confecção de um plano, no qual fracionam o tempo em grupos de estudos. Um guitarrista, em um modelo simplificado, pode dedicar uma hora de prática de Escalas de um determinado modo grego e uma hora de prática de Arpejos de um certo campo harmônico, sendo uma rotina de duas horas diárias e dois grupos compostos por quatro estudos, totalizando oito práticas ao dia. De acordo com os métodos de aprendizagem de instrumentos musicais expostos por Petrucci (2000) e Dillard (2007), um estudo após dominado jamais deve ser abandonado até a inserção dos elementos aprendidos em práticas mais complexas, como também, não necessitam prática constante após domínio e podem ser intercalados em níveis de complexidade. Dessa forma com organização é possível praticar muito mais do que oito estudos durante uma semana tendo apenas duas horas por dia.

Com a disponibilidade de tempo restrita, um guitarrista possui tempo limitado para praticar, sendo cada vez mais necessário e importante refletir sobre o plano de estudos. Quando o mal gerenciamento ocorre, duas adversidades são as mais frequentes: tocar sempre a mesma coisa e perder o domínio sobre algo que estava em estado de excelência. Foi observado por Masaki e Sommer (2012), que o aprendizado motor de forma intercalada e controlada pode ter uma evolução melhor em práticas dominadas, devido ao sistema nervoso manter maior nível de atividade e assim o aprendizado ser mais eficiente e menos tedioso, como também na fase de aprendizado de algo novo, foi observado que uma determinada sequência de movimentos pode ter um aprendizado melhor se inseridos em uma rotina fechada.

Segundo Silva (2017) o progresso da sociedade tem criado a redução de tempo para o desenvolvimento de habilidades unicamente de interesses pessoais. Muitos músicos não têm como atividade principal a música, como consequência surge a necessidade de dividir

a rotina de sua principal atividade com a prática do instrumento, isso faz com que cada minuto reservado tenha que ter o máximo de eficiência possível.

Sendo assim, com à necessidade de ter mais objetividade, eficiência, controle sobre o tempo e possibilitar a visualização empírica de dedicação, surge o propósito de automatizar o processo de organização de planos de estudos para guitarristas que tem como seu estilo um grande foco em desenvolvimento e manutenção técnica.

3 OBJETIVOS

Os objetivos deste trabalho, são apresentados em duas etapas. A primeira em que se sintetiza o objetivo principal e a segunda parte em que se busca detalhar objetivos específicos necessários para que o objetivo geral seja atingido.

3.1 Objetivo geral

Desenvolver um aplicativo web capaz de proporcionar o gerenciamento de estudos direcionados ao desenvolvimento e manutenção técnica de guitarristas, com base no tempo individual de cada estudo e técnicas sobre aprendizado e retenção motora.

3.2 Objetivos específicos

a) Obter dados sobre a rotina e organização de estudos em um grupo de guitarristas, buscando entender seu universo e necessidades.
b) Determinar quais e quantos são os estados e características que um estudo assume dentro do desenvolvimento técnico na guitarra.
c) Buscar equiparação de tais estados com os modelos de aprendizado e retenção motora existentes.
d) Definir o comportamento dos estados e características de desenvolvimento de um estudo dentro de um programa computacional, determinando os parâmetros para prioridades e ordenação de listas de estudos.

4 FUNDAMENTAÇÃO TEÓRICA

Para este trabalho a fundamentação teórica assume três aspectos: a primeira parte é exposta a teoria relacionada ao estudo e manutenção técnica na percepção musical, destacando tal comportamento no estudo da guitarra. Em segundo momento, é apresentada a fundamentação teórica relacionada ao desenvolvimento e aprendizado motor, que tem como objetivo justificar e demostrar por um aspecto técnico a importância da disciplina em habilidades que envolvam coordenação motora. Em um terceiro momento, é apresentada a teoria relacionada a área de tecnologia da informação, destacando os principais conceitos empregados no aplicativo desenvolvido.

4.1 Estudo em um instrumento musical

Durante este trabalho será muito comum a utilização do termo estudo ou Etude, listas de estudos, sequência de estudos, desenvolvimento de estudos, entre outros. Estudo, de uma maneira genérica, é uma unidade praticável que contempla ou exemplifica o desenvolvimento de alguma habilidade em um instrumento musical, este pode ser de diferentes níveis de complexidade, fórmula, quantidade de compassos, andamento, objetivo etc. Tais estudos podem até mesmo ser uma música inteira ou trechos extraídos, em livros na língua inglesa é comum o uso de exercise, example ou study, em material de foco erudito é comum se ver Étude que é de origem francesa (CAVINI, 2012; DILLARD, 2009; KRAMPE; ERICSSON, 1996; MACRITCHIE; MILNE, 2017; PETRUCCI, 2000).

4.2 Desenvolvimento técnico em um instrumento musical

Desenvolver habilidade em um instrumento musical demanda tempo, regularidade de prática e muitas abdicações, independentemente do universo musical inserido, seja ele na música erudita (clássica) ou qualquer outro atribuído como música popular. Músicos instrumentistas definem um objetivo de proficiência que desejam alcançar em um determinado tempo e quando alcançam este objetivo definem uma nova meta, assim criando um ciclo. Estudar um instrumento musical por muitas vezes se torna uma atividade sem um fim definido, podendo permutar por toda a vida de uma pessoa. Para isso elaboram um regime de prática que se encaixe em sua rotina, dividindo espaço com todos os outros elementos do cotidiano (PETRUCCI, 2000).

O gerenciamento de tempo torna-se um fator de importância para um músico instrumentista, se o mesmo tiver uma ideia previa do que precisa praticar, então terá um aprendizado melhor do que ficar repetindo de maneira desorganizada aplicações técnicas que já possui o domínio, sendo que muitas dessas aplicações necessitariam de frequências de estudos distintas, por isso além de reservar um tempo a prática do instrumento é importante gerenciar se vai aprender algo novo, evoluir um estudo existente ou apenas manter uma aplicação técnica em um constante nível de habilidade (PETRUCCI, 2000).

Para atingir proficiência técnica em um instrumento musical há a necessidade de entrar em um regime de dedicação contínuo, tal regime é dividido em grupos de aplicações técnicas a serem definidos conforme as particularidades de cada instrumento musical e objetivos individuais de cada músico. No livro técnico destinado a guitarra elétrica *Intelli-Shred*, Dillard (2007) reforça que o aprendizado feito com regularidade excede o aprendizado submetido a cargas grandes de tempo em momentos esporádicos, citando o exemplo de que uma pessoa que pratica seu instrumento durante uma hora por dia durante quatro dias seguidos terá melhor resultado do que quatro horas seguidas em um único dia.

Ao longo do tempo um estudante de um determinado instrumento musical tende a buscar material de estudo em diversas fontes, entre as mais comuns podemos destacar, livros que possuem um método de estudo gradativo, livros que desenvolvem uma determinada técnica e trazem diversas aplicações, livros puramente técnicos sem tantas aplicações musicais, videoaulas que trazem um *insight* por um instrumentista consagrado dentro de um estilo, materiais avulsos encontrados pela internet, trechos de músicas que interessam ao instrumentista e os próprios construtores de habilidade que o músico cria. Isso gera um grande repertório de estudos a ser organizado (PETRUCCI, 2000).

4.3 Técnicas e conceitos empregados na guitarra elétrica

Para um instrumentista, seja qual for o instrumento, as possibilidades de aplicações de conceitos musicais possuem uma grande diversidade, que se estendem quando se trata de estilos musicais e até mesmo da construção e limitações de cada instrumento. Para o guitarrista, as possibilidades se aumentam, devido a guitarra elétrica ser um instrumento eletroacústico, que utiliza captação e possibilita a mesma técnica ser aplicada de maneiras diferentes. Como por exemplo o controle do *Palm Mute*, que é uma técnica em que se abafa as cordas da mão em que se palheta (Comumente mão direita aos destros), tal técnica se alterou

com o tempo para acompanhar o aumento de ganho dos captadores e possui diferentes sensibilidades para estilos musicais distintos (DILLARD, 2009; PETRUCCI, 2000).

Para um guitarrista é crucial estudar palhetada (Em inglês *Picking*), técnica que consiste em palhetar as cordas do instrumento. Tal técnica possui diversos meios de ser executada e depende do que está sendo aplicado na mão em contato com a escala do instrumento localizada no braço da guitarra, que tem o objetivo de segurar as cordas até encostar no traste desejado. A palhetada possui dois ângulos de ataques, comumente chamados de palhetada para cima e palhetada para baixo (Em inglês *Up Stroke* e *Down Stroke*). A palhetada pode ser aplicada em apenas um ângulo de ataque, como pode ser aplicada de maneira alternada (Em inglês *Alternate Picking*), que consiste no revezamento entre os ângulos de ataque. Um guitarrista pode aplicar a técnica de palhetada varrida (Em inglês *Sweep Picking*), que tem o objetivo de subir ou descer as cordas mantendo o ângulo de ataque, também há inúmeras variações e padrões combinando ambos os tipos (DILLARD, 2007; PETRUCCI, 2000; STETINA, 1990).

As técnicas aplicadas a mão, cuja responsabilidade é segurar as cordas em direção aos trastes, encontrados no braço do instrumento (Mão esquerda aos destros), possuí inúmeras técnicas e deve estar em sincronia com a mão que executa a palhetada. Esta mão é responsável por aplicar estruturas musicais, que darão sonoridades ao que se está palhetando. Estas estruturas podem ser: escalas, arpejos, *tapping*, acordes, entre outras. Esta mão também aplica técnicas de interpretação de notas, que envolve arqueamentos e vibrações, como por exemplo: *bend*, *vibrato*, *tremolo*, *slide*, ligados, entre outras. É muito comum guitarristas buscarem ou criarem estudos para unicamente desenvolver o sincronismo entre as mãos e independência dos quatro dedos, como também é comum buscarem estudos que envolvam diversas técnicas e estruturas musicais em um único estudo (DILLARD, 2007; PETRUCCI, 2000; STETINA, 1990).

4.4 Aprendizado motor

O ser humano sempre foi fascinado por performances acima do comum, seja um jogador de futebol com dribles incríveis, um piloto de *Fórmula 1* disputando posições onde ações precisas são tomadas em frações de segundos, músicos instrumentistas de alta performance executando solos mágicos, entre outras muitas habilidades que o ser humano pode construir. Tais habilidades seguem definições pela ciência que estuda aprendizado motor e performance, são: habilidade **ambiental** que consiste em movimentos para atingir um

objetivo, **certeza** que é a habilidade de atingir o resultado final, a **economia** que tem como objetivo diminuir a energia gasta para atingir um determinado objetivo e **tempo** que tem como objetivo executar o mais rápido possível tal movimento. As habilidades também são definidas conforme sua previsibilidade em **abertas** e **fechadas**. Há também a definição de fluxo de movimentos em contraste com execuções com início e fim definidos, que são: **discretas**, **continuas** e **seriadas** (ALVES et al., 2016; KRAKAUER et al., 2019; SCHMIDT; LEE, 2011).

4.5 Aprendizado motor em instrumentos musicais

Desenvolver habilidade em um instrumento musical depende de vários elementos cognitivos para conseguir aprender e posteriormente executar uma sequência de movimentos involuntários com precisão rítmica que estão dispostos espacialmente e integrados com o corpo e audição para aquisição de timbre e melodia (ZATORRE; CHEN; PENHUNE, 2007).

Quando um músico toca um instrumento ele executa três funções motoras distintas: **tempo** que está ligada ao andamento da música que está tocando, que é definido por um conjunto de figuras musicais e suas divisões matemáticas, isso significa que o músico tem que executar movimentos sincronizados e estes devem fluir precisamente conforme a notação horizontal empregada na música; **sequência** que determina o que vai ser tocado dentro do elemento tempo, essa sequência tem variação entre os instrumentos, em instrumentos de teclas, seriam quais teclas serão pressionadas, em uma bateria seriam quais peças serão batidas, instrumentos de cordas temperados seriam quais casas e cordas a serem tocadas, não temperados seria o quanto haverá de deslocamento da mão e dedos e cordas a serem tocadas, entre outros, esse sequenciamento é feito de maneira involuntária sendo que inúmeros movimentos são feitos para a execução de um simples elemento deste sequenciamento; **organização espacial** além do sequenciamento das notas a serem tocadas, o músico busca as melhores estratégias para executar tais sequências, buscando movimentos mais precisos e objetivos (WYATT,; SCHROEDER, 1998; ZATORRE; CHEN; PENHUNE, 2007).

4.6 SOLID (Princípios de desenvolvimento)

Os princípios de desenvolvimento de software para programação orientada a objetos com o acrônimo de SOLID, foram primeiramente sugeridas por Martin (2000), revisados posteriormente em Martin (2008), até a última publicação em Martin (2017).

Destaca-se que tais princípios não são atrelados a nenhum *framework* ou linguagem de programação e possuem um caráter conceitual, sem maneira ou forma de implementação definidas. Sendo a ação de implementar feita pela análise e entendimento do sistema pelo desenvolvedor.

A letra S significa SRP, acrônimo para *Single Responsibility Principle (Princípio da Responsabilidade Única*). Este princípio tem como principal foco proporcionar a classes uma única razão para existir, uma única responsabilidade e uma razão para mudar. Para o autor, uma classe coesa é aquela com uma única responsabilidade, pois torna o software mais robusto para ser expandido e mantido, que ao contrário com classes entrelaçadas se tornaria custoso.

A letra O significa OCP, acrônimo para *Open/Close Principle* (Princípio do Aberto/Fechado). Este princípio tem como principal objetivo sugerir que classes, métodos, funções e outros, devem ser abertas para expansão e fechadas para modificações. Segundo o autor a implementação de um código deve ser pensada em expansão e manutenção, sendo uma nova funcionalidade sempre ser adicionada com a mínima modificação possível de código existente.

A letra L significa LSP, acrônimo para *Liskov Substitution Principle* (Princípio da Substituição de Liskov). Este princípio tem como principal característica determinar que classes derivadas devem ser sempre possíveis de assumir comportamentos de sua classe base. Significa que, quando ocorrer a implementação de Herança entre classes, as classes derivadas devem se comportar como a classe base se solicitado, muito comum para listagem de itens de subtipos diferentes e funcionalidades iguais aplicadas a diferentes derivações.

A letra I significa ISP, acrônimo para *Interface Segregation Principle* (Princípio da Segregação de Interfaces). Este princípio tem como principal característica evitar a implementação de interfaces com recursos desnecessários, chamada de interfaces “gordas”. Significa que classes clientes devem apenas implementar interfaces que vão utilizar e não devem ser obrigadas a implementar algo desnecessário, com isso o autor recomenda a utilização de interfaces especificas e menos genéricas.

A letra D significa DIP, acrônimo para *Dependency Inversion Principle* (Princípio da Dependência Inversa). Este princípio tem como ideia principal a dependência entre classes, em que o simples não deve depender do complexo e sim o complexo depender do simples. De outra maneira podemos dizer que abstrações não devem ser baseadas em detalhes e os detalhes devem ser baseados em abstrações. Este princípio é a base para a proposta de dependências da Arquitetura Limpa sugerida pelo mesmo autor. Martin (2017), destaca-se que

para este princípio ser atingido, há a necessidade de todos os outros anteriores serem respeitados.

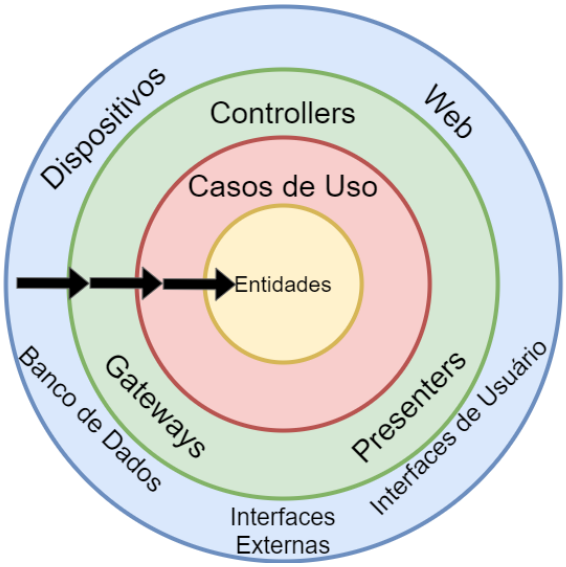
4.7 Arquitetura Limpa (*Clean Architecture*)

É proposto por Martin (2017) uma arquitetura para desenvolvimento e teste de software que tem como objetivos principais: tornar um software independente de framework, lógica de negócio individualmente testável, aplicativo independente de UI e aplicativo independente de banco de dados. Em resumo, o objetivo principal são as regras de negócio não depender de elementos exteriores.

Independente de *framework*, significa que *frameworks* devem no máximo serem utilizados apenas como ferramentas, sendo que não devem influenciar na concepção e design das regras de negócio. O autor acrescenta que a lógica de negócio deve ser possível de teste individual, sem depender de UI, banco de dados e outros serviços externos. Quanto a UI, sua implementação deve ser externa as regras de negócio, sendo possível a implementação de qualquer tipo de interface sem alteração de tais regras. Exemplificando, um mesmo aplicativo deve permitir diversos tipos de interfaces, no universo *web*, significa que deve suportar diversos tipos de clientes, podendo ser web, mobile, *command line interface* (CLI) e até mesmo clientes desktop.

Para tal feito, Martin (2017) propõem uma arquitetura em camadas estruturadas com regras de dependência e um fluxo de controle. A Figura 1, que foi adaptada do livro referente ao tema, demostra em um nível alto de abstração como um sistema deve ser pensando. Os círculos referem-se as camadas propostas e quanto mais ao centro, mais envolvidos com o domínio do aplicativo estão e quanto mais externos, mais estamos lidando com dispositivos. Há também a demonstração da regra de dependência, que ilustram que a camada exterior depende da próxima camada interior e o inverso não deve ocorrer. Destaca-se que não há limitações na quantidade de camadas, todavia a regra de dependência deve sempre ser respeitada.

Figura 1 – Arquitetura Limpa (*Clean Architecture)*



Fonte: Adaptado de Martin (2017)

A camada central chamada de Entidades, encapsula o domínio do aplicativo, contendo entidades e regras do negócio referente apenas ao universo da aplicação em desenvolvimento, estas podem ser unicamente entidades, como também podem conter comportamentos. Na camada Casos de Uso, chamada pelo autor de *Use Cases* ou *Application*, ocorre o encapsulamento de toda a lógica que é referente ao aplicativo. Significa que nesta camada são implementadas lógicas referentes ao aplicativo que não pertencem ao seu domínio, como por exemplo: em um sistema web temos a geração de *token,* políticas de acesso, segurança, como será feita a persistência de dados, gerenciamento e controle de erros, entre outros. Em resumo esta camada está relacionada a parcela referente a um sistema computacional e como serão atingidos os objetivos do aplicativo não pertinentes ao seu domínio. Nota-se na Figura 1, uma camada externa representada na cor verde, contendo *Gateways, Controllers e Presenters*, que é a camada responsável por conter toda a lógica referente a elementos externos ao aplicativo. Segundo Martin (2017), estes elementos não devem conhecer nada sobre a lógica do aplicativo, sendo estritamente responsáveis por fazer tal interconexão. Martin (2017) reforça que tal camada não deve fazer qualquer alteração de dados ou tomar qualquer decisão, sendo totalmente indiferente ao comportamento da camada interior, reforçando a regra de dependência proposta. A última camada representada em azul na figura 1, refere-se a *frameworks,* banco de dados, serviços externos (Como por exemplo um servidor de imagens), cliente (para sistemas web), entre outros. Martin (2017) reforça que tais elementos não devem interferir nas camadas centrais, respeitando sempre a regra de

dependência e reforça que tais elementos devem ser tratados apenas como ferramentas e não meios para implementação do aplicativo como um todo, possibilitando substituição, adição e modificação sobre estes. Acrescenta-se que o autor chama as camadas internas (Casos de Uso e Entidades) de núcleo.

4.8 ORM (*Object-Relational Map*)

Para que ocorra a interação entre dois tipos de sistemas distintos, foi criada a técnica de acrônimo ORM que significa *Object-Relational Mapping* (Objeto Mapa Relacional). Este conceito tem como objetivo fazer possível a interação entre banco de dados relacionais, que utilizam tabelas para armazenar dados e aplicativos desenvolvidos utilizando a programação orientada a objetos. Tal técnica faz possível a criação de um dicionário virtual intermediando tais sistemas com a utilização de *frameworks* desenvolvidos para diversas linguagens de programação. Estes *frameworks* criam associações entre as tabelas escalares do banco de dados e abstrações que são as entidades que serão manipuladas dentro do aplicativo, fazendo com que tais objetos estejam disponíveis ao aplicativo para manipulação e se solicitados para persistência de alterações, exclusão e novos registros ao banco de dados ((LORENZ et al., 2017; PRIBYL, 2000)).

4.9 *Mediator Pattern*

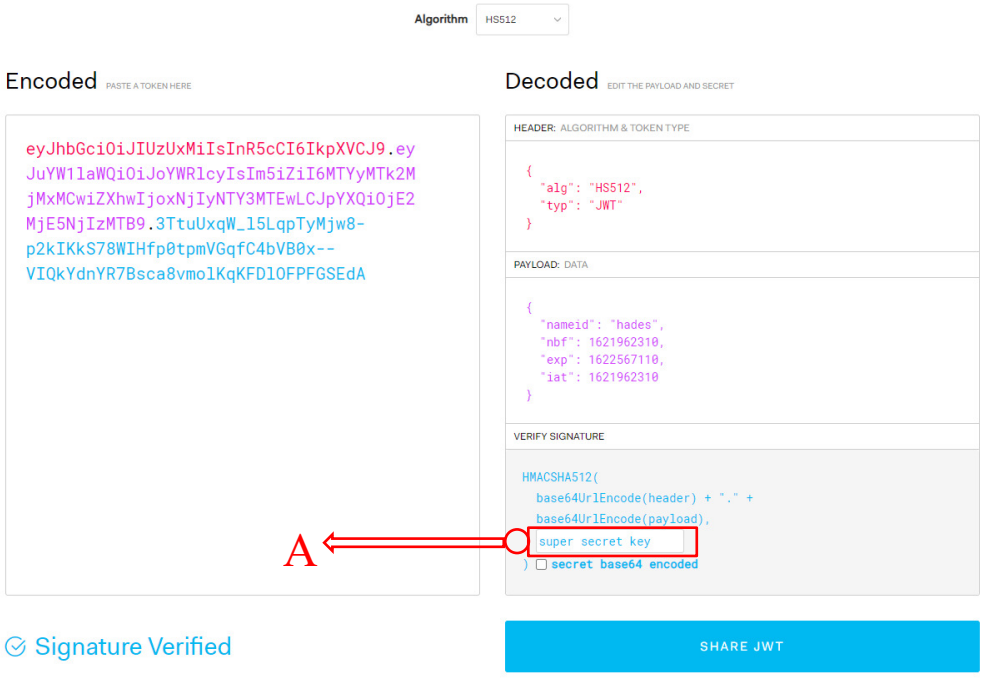
Entre os mais conhecidos padrões de design para desenvolvimento de software, temos o *Mediator Pattern,* um padrão comportamental que tem como objetivo principal sugerir o encapsulamento do comportamento entre duas classes distintas, fazendo com que elas interajam entre si desconhecendo o comportamento uma da outra, é deste conceito de mediação que surge a sua denominação. Este padrão de design foi sugerido com o objetivo de reduzir o caos entre classes fortemente acopladas que necessitam ser independentes, precisam manter regras de dependência e ter possibilidade de reuso dentro do aplicativo. Sendo assim, em sua implementação há o cessar de comunicação entre tais classes, que passa a ocorrer com classes mediadoras que chamam e retornam informações de forma incógnita. É comum ser aplicado quando classes estão entrelaçadas ao ponto de impossibilitar alterações e adição de novas funcionalidades e quando não é possível o reuso de um componente de tão dependente que é de tais classes. O *Mediator Pattern* tem como vantagem respeito ao *Single Responsibility Principle* (SRP), pois se torna possível fazer classes coesas e criar

responsabilidade única entre todos os envolvidos, *Open/Close Principle* (OCP) se torna possível, pois pode ocorrer a adição de novas mediações sem alterar código existente e possibilitar reforçar as regras de dependências estabelecidas (GAMMA et al., 1994; MARTIN, 2008).

4.10 *Jason Web Token* (JWT)

Entre os padrões criados para controle de acesso em aplicativos cliente-servidor, existe o *JSON Web Token* (JWT), que é utilizado para a criação de *web tokens*, que por sua vez é armazenado no dispositivo cliente em forma de cookies, no navegador, entre outros que são determinados pelo tipo de cliente, sendo seu uso muito comum em APIs REST e GraphQL API. *JSON Web Token* são utilizados em todos os *requests* que um cliente faz ao servidor que possua regras de acessos quanto a identificação de usuários, que por conseguinte o servidor determina sua validade e retorna o solicitado ou erros, caso o *token* não seja validado. Quando é gerado o *token*, ele possui uma assinatura digital que é determinada pelo desenvolvedor do aplicativo como uma chave secreta e utilizado algoritmos de criptografia, para geração do *token*, isso determina que JWT é uma chave assinada. A figura 2 representa a estrutura de um *JSON Web Token*.

Figura 2 – Estrutura *JSON Web Token*



Fonte: JWT IO <https://jwt.io/>, preenchido pelo autor (2021)

Na figura 2, observa-se a esquerda a representação do *JSON Web Token*, que é enviado no cabeçalho de autorização de todos os *requests* HTTP feitos pelo cliente utilizando a identificação *Bearer*, que decodificado é dividida em três partes, representados pela analogia entre as cores das colunas. A primeira parte chamada de *HEADER* (na cor vermelha), é composta pelo algoritmo de criptografia ("alg") e pelo tipo de *token* que carrega ("typ"). Em seguida há a estrutura do *PAYLOAD* (na cor rosa), que no exemplo é composta pelo usuário solicitante ("nameid") e informações ao tempo de expiração do *token*, observa-se que em *JSON Web Token* o *PAYLOAD* se adapta as necessidades de cada aplicação. E por fim temos a caixa de texto *VERIFY SIGNATURE* (na cor azul), que está em posse da chave criptografada de até 256 bits, representada pela marcação A, em que apenas o servidor do aplicativo tem conhecimento. (JONES, 2015; JONES; HILDEBRAND, 2015; NICKEL, 2016).

4.11 *Representation State Transfer* (REST API)

Foi proposto por Fielding (2000), um estilo de arquitetura baseado em diversos sistemas na época e baseado em seis princípios, ao qual foi nomeado com o acrônimo REST *RElational State Transfer* (*Transferência de estado representacional*). Esses seis princípios são:

- *Client-Server* (Cliente-Servidor): tem como ideia principal a separação de responsabilidade entre servidor e possíveis clientes, esta proposta tem como objetivo principal garantir que cliente e servidor evoluam de forma separada.
- *Stateless* (Sem estado de sessão): com a intenção de manter recursos livres, a seção é gerenciada pelo cliente, que deve possuir e enviar todas as informações necessárias em cada *request* para garantir acesso a recursos do servidor.
- *Cache*: com a intenção de melhorar performance e deixar o processamento menos custoso, foi criando o conceito de *cache*, em que o cliente terá a sua disposição informações recorrentes.
- *Uniform Interface* (Interface Padrão): possui padrões auto descritivos e amigáveis a linguagem humana para identificação e manipulação de

recursos no sistema, também faz a utilização de *hypermedia* como mecanismo.

- *Layered System* (Sistema em camadas): é um sistema hierárquico em camadas que garante que acessos a determinados componentes sejam restritos ao mesmo.
- *Code-on-Demand* (Código sob demanda): com a intenção de reduzir código para uma possível implementação de cliente, REST possibilita a execução de código com o uso de *scripts* ou *applets*.

Quando tratamos de REST, utilizamos muito o termo **recurso**, que é uma abstração para tudo que iremos manipular no sistema, seja informações, fotos, vídeos, serviços, listas de recursos etc. Para que isso ocorra, todo recurso deve ser **identificável** e o seu conjunto de informações é chamado de **representação**. Para que ocorra uma **transferência** de estado, são utilizados diversos **métodos de recurso** utilizando o HTTP como meio, sendo os mais comuns: GET, POST, PUT, PATCH, DELETE, COPY, HEAD, OPTIONS, LINK, UNLINK. Fielding (2000) e Fielding et al. (2017), não determina quais métodos devem ser utilizados para qual fim, mas recomenda que devem seguir uma padronização e buscar uma melhor representação para o que cada método será responsável.

4.12 Microsoft .NET Core

Sendo o Microsoft .Net Core escolhido ao desenvolvimento do aplicativo que contempla o objetivo geral deste trabalho, são apresentadas a sua origem e as principais bibliotecas utilizas. O .Net Core foi escolhido por possuir um pacote de bibliotecas projetadas para funcionarem em harmonia, que são essenciais para a realização deste trabalho e apresentadas abaixo.

Com a necessidade de revisar o pacote .NET e se adequar ao universo de tecnologias multiplataformas, em novembro de 2014 a Microsoft fez o anúncio de seu novo *framework* chamado de .NET Core, ao qual seria desenvolvido em formato de código aberto, mantido pela .NET Foundation e utilizadas como foco as linguagens de programação C#, F# e Visual Basic. Um conceito importante sobre este *framework* é a remoção de bibliotecas para sistemas legados, como por exemplo a biblioteca responsável por *Windows Forms* e a WPF (*Windows Presentation Foundation*), tornando o novo pacote mais leve. Em novembro de 2020 a Microsoft decidiu iniciar o processo de unificação de todos os pacotes .NET, tendo

como elemento principal dessa mudança o .NET Core, porém, passando a se chamar apenas .NET (PRICE, 2019, 2020).

Com o objetivo de integrar a possiblidade de consultas e manipulação de dados diretamente utilizando a linguagem de programação C#, foi criado o conceito LINQ (Acrônimo para *Language INtegrated Query*). Este conceito, possibilita utilizar operações familiares da linguagem de programação C# para operações de *query*, oferecendo todo o *IntelliSense*, correção de sintaxe e compilação que a linguagem de programação oferece. O conceito resulta em uma programação robusta ao qual o desenvolvedor está menos sujeito a erros causados por inconsistências de *queries*, implementadas com o uso de *strings*. Tal concepção é oferecido no .Net Core em forma da biblioteca System.Linq e é essencial para aplicação de *Object-Relational Map* (PRICE, 2019, 2020; WAGNER, 2017).

O Entity Framework Core é parte do conjunto de tecnologias ADO.NET, que oferece uma solução para desenvolvedores de aplicativos com necessidade de persistência. Ele tem objetivo de encapsular toda a complexidade ao se utilizar conceitos diferentes no desenvolvimento de um aplicativo, sendo: modelar entidades e definir suas relações, implementar suas lógicas e regras de negócios e interagir com sistemas de armazenamento de dados. Com isso, o Entity Framework Core fazendo o uso do conceito *Object-Relational Map* permite aos desenvolvedores trabalharem com maior abstração para manipulação de dados e oferece integração com as concepções LINQ e Entity SQL (PRICE, 2019, 2020).

Para gerenciamento de usuários, senhas, dados de perfis, *roles*, *tokens*, *claims*, confirmação de *email*, criação de contas através de serviços externos, entre outros, a Microsoft fornece a plataforma de identidade chamada Microsoft Identity Platform. Para .Net Core, a base de tal plataforma é implementada utilizando Microsoft Asp.Net Core Identity e tem o objetivo de fornecer ao desenvolvedor todos os recursos da plataforma principal. Como por exemplo: criação de *JSON Web Tokens*, definição da entidade usuário para ter sua persistência e relacionamentos gerenciada pelo Entity Framework Core, políticas de acesso, entre outros (ANDERSON, 2020; PRICE, 2019, 2020).

4.13 *Domain-Driven Design* (DDD)

Primeiramente para se entender o propósito do *Domain-Driven Design*, deve-se compreender o que significa domínio de um aplicativo. Para Evans (2003), domínio é o universo de conhecimento referente ao objeto de trabalho ao qual o software será aplicado, e este contém as entidades que são abstrações deste universo. Para Martin (2017), a camada de

entidades apresentada na arquitetura limpa remete ao domínio do aplicativo, sendo que tal camada pode compor mais de uma camada, possibilitando ser composta por logica de domínio e entidades de domínio, que são comumente chamadas de lógica de negócio (Em inglês *Business logic)*.

Para Evans (2003), o *Domain-Driven Design* consiste em desenvolver um aplicativo que evolua a partir de seu domínio, em que tal processo deve ser feito conectando peças mais complexas ao domínio e sugerindo três princípios: foco na lógica de domínio, problemas complexos devem ser baseados na lógica de domínio, explorar com especialistas o domínio do aplicativo, com o objetivo de revelar e resolver problemas relacionados.

Segundo Evans (2003), alguns pontos devem ser considerados para uma boa utilização do conceito *Domain-Driven Design*, sendo:

- *Context* (Contexto): Palavras que representam elementos computacionais do domínio, devem representar o contexto dele.
- *Model* (Modelo): O sistema deve ser pensando em abstrações do domínio, que devem solucionar os problemas enfrentados.
- *Ubiquitous Language* (Linguagem ubíqua): Modelos desenvolvidos, devem seguir uma linguagem estruturada para que todos os membros da equipe de desenvolvimento consigam entender o software.
- *Bounded Context* (Contexto Limitado): Deve estabelecer limites e responsabilidades dentro de subsistemas, dentro do qual um modelo específico pode ser aplicado.

Para Avram e Marinescu (2007), a aplicação do *Domain-Driven Design* pode ser dividida em camadas com responsabilidades distintas que são relativos a complexidade do aplicativo e sugerem a criação de regras entre elas, sendo essas camadas:

- *Presentation Layer*: Responsável por interfaces de usuários.
- *Application Layer*: Responsável pela lógica do aplicativo, não deve conter a lógica de domínio. Está relacionada ao tipo de aplicativo e funcionalidades necessárias.
- *Domain Layer*: Composta pela lógica e entidades de domínio.
- *Infrastructure Layer*: Deve conter bibliotecas utilizadas na comunicação entre camadas, segurança, serviços externos, entre outros.

5 MÉTODO PROPOSTO

O desenvolvimento deste trabalho foi realizado em três etapas. Na primeira adotou-se o levantamento de informações junto a um grupo de músicos escolhidos, buscando conhecer sua rotina de desenvolvimento e manutenção técnica, e com isso criar uma relação entre os modelos de aprendizado e retenção motora. Na segunda parte, recorreu-se a busca de padrões para elaboração de modelos aplicáveis a um software computacional. Na terceira etapa tratou-se da escolha de ferramentas e desenvolvimento do programa computacional.

5.1 Levantamento de dados

Esta etapa constituiu-se na coleta de informações, buscando compreender a rotina e necessidades de guitarristas de perfil pertinente à proposta. Foi utilizado o método presencial de entrevista e buscou-se extrair aspectos pessoais do desenvolvimento técnico de cada.

5.1.1 Seleção de guitarristas para levantamento de informações

O grupo foi composto por 12 pessoas, abaixo seguem suas informações mais relevantes para o projeto em questão. A média aritmética do grupo é 33 anos de idade.

Tabela 1 – Idade dos participantes

Idade	Quantidade	%
Até 25	2	16,7%
25 - 30	1	8,3%
30 - 35	3	25,0%
35 - 40	4	33,3%
Mais de 40	2	16,7%

Fonte: Elaborado pelo autor (2021)

A grupo foi composto, em sua grande maioria por guitarristas que não tem a música como principal atividade remunerada.

Tabela 2 – Participantes que têm a música como principal atividade

Resposta	Quantidade	%
Sim	2	16,7%
Não	10	83,3%

Fonte: Elaborado pelo autor (2021)

O grupo foi composto, em sua maioria por guitarristas que em algum momento foram professores de música.

Tabela 3 – Participantes que já foram instrutores de música

Resposta	Quantidade	%
Sim	7	58,3%
Não	5	41,7%

Fonte: Elaborado pelo autor (2021)

A tabela a seguir demonstra o tempo contínuo de contato com a música, este dado pode incluir contato com outros instrumentos e interrupções no desenvolvimento. A média aritmética ficou em 20 anos.

Tabela 4 – Tempo de contato com a música

Tempo	Quantidade	%
Até 8	2	16,7%
8 -16	2	16,7%
16 - 24	4	33,3%
24 - 32	3	25,0%
Mais de 32	1	8,3%

Fonte: Elaborado pelo autor (2021)

A grupo foi composto exatamente pela metade de guitarristas que estudam ou estudaram em conservatório.

Tabela 5 – Participantes que já estudaram em conservatório

Resposta	Quantidade	%
Sim	6	50%
Não	6	50%

Fonte: Elaborado pelo autor (2021)

A grupo foi composto em grande maioria por guitarristas que não possuem formação superior na área de música.

Tabela 6 – Participantes com formação superior

Resposta	Quantidade	%
Sim	2	16,7%
Não	10	83,3%

Fonte: Elaborado pelo autor (2021)

5.1.2 Entrevistas individuais com o grupo de pessoas

As entrevistas foram realizadas individualmente de forma presencial. Foi observado padrões em rotinas de estudos, semelhanças nos meios de se organizar tais estudos e necessidades.

5.1.2.1 Grupos de estudos

Foi observado que todos os entrevistados implementam em sua rotina grupos de estudos. Tais grupos facilitam a organização e intensificam o foco, pois são compostos de movimentos motores parecidos, estruturas musicais semelhantes ou harmonias afins. Isso ocorre por um motivo: a **habilidade espacial** é intensificada e trabalhada continuamente mesmo alterando o estudo, pois estes possuem disposição e movimentos parecidos. Tais grupos são de organização estritamente pessoal, porém foi levantando que todos seguem padrões. Há unanimidade em classificar tais grupos de desenvolvimento técnico com o elemento principal ao qual representam no universo de cada guitarrista, podendo ser a técnica principal, a estrutura musical ou harmonia ao qual pertencem. Assim, tornando a classificação de um grupo de estudos em um elemento subjetivo e consequente um guitarrista ter a quantidade de grupos que lhe convém.

Entre os entrevistados foi observado três formas diferentes de se definir tais grupos, cada abordagem possui uma percepção diferente. Foi observado que os entrevistados podem dividir grupos que podem ser exemplificados da seguinte maneira: Escalas, Arpejos, *Riffs*, *Tapping*, Fraseado e Repertório. Tal modo de organização não foca em organizar por técnica e não tem a intenção de organizar por harmonias, o foco são estruturas musicais e dentro de cada grupo há a adição de inúmeros estudos praticados com diferentes técnicas e harmonias. Foi observado que outros preferem organizar grupos exemplificados da seguinte maneira: Palhetada Alternada, Ligados, *Sweep Picking* e Repertório. Tal modo de organização busca criar cada grupo baseando-se em uma forma de execução e aplica-se dentro de estruturas musicais e harmonias. Por fim, foi observado que podem dividir de grupos que são

exemplificados da seguinte maneira: Aquecimento, Coordenação Motora Avançada, Fraseado em Mi Frígio, Fraseado em Lá Eólico, Fraseado em Ré Dórico, Repertório. Tal modo de organização cria grupos baseados em intenção musical, observa-se que este possui foco na aplicação de estruturas músicas e técnicas dentro de harmonias. Assim, formando grupos que desenvolvem percepção e fluência ao mesmo tempo que desenvolvem técnica. Repara-se também que possui dois grupos com desenvolvimentos puramente técnicos, um básico para aquecimento e outro que desenvolve técnica em estruturas musicais simplificadas.

Destaca-se que não existe uma regra para nomenclatura de grupos, cada guitarrista possui seu universo particular dinâmico de divisão de seus estudos e suas aplicações, alguns possuem foco em estudos puramente motores, enquanto outros se desenvolvem focando diretamente em estruturas musicais e há aqueles que trabalham elementos de percepção e harmonia junto ao desenvolvimento técnico.

5.1.2.2 Características de um estudo

Foi perguntado que elementos individuais de cada estudo são importantes em uma categorização deles. O objetivo desta questão é levantar as características para abstração de uma entidade chamada **Etude**. Tal entidade é o elemento a ser gerenciado pelo sistema ao qual este trabalho se destina, que é organizar estudos.

Todos os entrevistados atribuem um nome em cada estudo, porém não há padronização e muitas estratégias são adotadas, visto que, assim como em grupos de estudos o título é subjetivo, devido a percepção de cada guitarrista ser diferente quanto ao elemento principal contido e objetivo de tal estudo. Foi observado que existem atribuições de títulos aos estudos com características sistemáticas, como por exemplo: Ex-1-Penta-Am, Ex-2-Penta-Em-Pos12 etc. Outra observação foi de estudos com características criativas com nomes que melhoram a automotivação, como por exemplo: *Whirlwind Arpeggios*, *Tapping from the Halen* etc. Independente do nome, este é o elemento principal que se categorizam estudos dentro de uma rotina de desenvolvimento e manutenção técnica.

Foi observado que os entrevistados utilizam metrônomo para marcação de tempo e desenvolvimento de precisão rítmica em seus estudos. Tais estudos, quanto mais distantes de estruturas musicais e com objetivo de desenvolver movimentos mais refinados, intensifica-se o uso do metrônomo. O andamento específico de um estudo é dinâmico, ou seja, é aumentado conforme o nível de domínio. Quais estudos utilizam metrônomo e quais não usam são de escolha pessoal dentro da necessidade e objetivo de cada. Conclui-se que, é

importante ao guitarrista saber se o estudo deve ser feito com metrônomo e se sim, qual o BPM que tal estudo está.

Todos os entrevistados sentem a necessidade de fazer alguma observação sobre os seus estudos. Estas observações são feitas de diversas maneiras, sendo as duas mais frequentes: adjacente ao impresso do estudo ou na observação de suas planilhas de estudos. Entende-se que é importante o guitarrista ter um espaço para observações pessoais.

5.1.2.3 Rotina semanal de estudos

Foi questionado sobre a rotina semanal, com intenção de saber se existe um plano detalhado para determinada semana e em qual fidelidade é seguido. Tal questionamento tem o objetivo de identificar se os grupos de estudos são fixos ou flexíveis, se existe quantidade de horas semanais fixas, se existe um horário reservado para prática do instrumento e entender o ciclo de estudos de um guitarrista.

O aspecto principal sobre este ciclo é a constatação de que a rotina de um guitarrista não é constante, não existe um plano fixo para ser feito com regularidade. Dos entrevistados 75% tocam em ao menos uma banda, e destes 67% tocam em mais de uma banda. A rotina desses guitarristas é afetada diretamente por ensaios, shows e reuniões, os quais não seguem nenhum padrão e mudam de forma repentina. Um outro fator que reforça o não seguimento de um plano fixo, é o fato da grande maioria dos entrevistados (83,3%) não possuírem música como principal atividade, algo que impacta diretamente os estudos do instrumento musical ainda mais se for um trabalho com horários dinâmicos. Foi constatado que a rotina pode variar de zero horas, podendo passar em alguns casos de 12 horas por dia.

Segue abaixo um exemplo demonstrando a ausência de constância quanto a horas diárias da rotina aproximada de um dos entrevistados durante o período de 4 semanas. Estes dados foram possíveis de levantar pois o entrevistado faz acompanhamento em planilha eletrônica.

Tabela 7 – Amostra de quantidade de horas diária de prática durante quatro semanas.

	Dom.	Seg.	Ter.	Qua.	Qui.	Sex.	Sáb.
Semana 1	9	2	2	0	3	4	8
Semana 2	10	2	2	3	3	0	0
Semana 3	0	2	2	2	4	4	8
Semana 4	10	2	0	0	3	4	8

Fonte: Elaborado pelo autor (2021)

O que ocorre é o máximo aproveitamento de tempo disponível, buscando preencher com estudos da melhor maneira possível. Dentro desses **treinos** acontece o planejamento do que fazer para ter uma melhor manutenção e evolução técnica, é neste momento que o gerenciamento dos estudos é feito. Isso significa determinar qual sequência fazer, o que necessita de maior atenção e o que tem menos prioridade caso ocorra alguma interrupção e não consiga terminar o treino planejado.

Identificou-se que os guitarristas entrevistados possuem uma rotina dinâmica, assim não tendo um plano criteriosamente detalhado sobre em qual horário começar os estudos e o quanto de horas praticar. Para eles o mais importante é o que fazer quando se tem o tempo para praticar.

5.1.2.4 Estados de um Etude (Estudo)

Tentando entender como um estudo é percebido por um guitarrista, no ponto de vista de fluência e aprendizado, foram levantadas as seguintes questões: Quais as fases que um estudo apresenta quanto à intensidade de domínio? O que considera um exercício dominado? O que faz com tal estudo depois de dominado?

Na primeira questão, tentando entender em quantos estágios de desenvolvimento um estudo se encontra, 100% dos entrevistados disseram que primeiramente aprendem um novo estudo de forma lenta sem o uso de metrônomo. Este processo tem o objetivo de maximizar a perfeição de cada movimento, executando a sequência de notas devagar. Este aprendizado inicial é encarado de maneira pessoal, podendo o estudo ser fracionado e entendido por partes e posteriormente unidas, como pode entrar em um processo incremental, ao qual se domina o próximo fragmento e se incrementa ao conteúdo já entendido, em todo este processo o tempo é marcado no pé ou contagem de tempo mental.

Após esta etapa, o estudo passa para a fase de aprendizado dentro de um andamento definido (BPM), este andamento é escolhido com base no que o guitarrista consegue tocar de modo que se mantenha a correta divisão dos tempos entre as notas e a clareza de execução. Conforme vai aumentando o domínio sobre o estudo, o guitarrista tende a elevar o andamento até chegar ao momento que os movimentos ficam implícitos e a quantidade de erros perto de zero.

Neste ponto entra a segunda questão, o que é um estudo dominado? As respostas foram variadas, e por se tratar de um questionamento aberto e muito subjetivo não se obteve

uma resposta padrão. Porém, a síntese que melhor representa tais respostas é; quando o guitarrista se sente confortável, tem total confiança e executa o estudo como um fragmento único.

Foi observado que depois de cruzar este linear o estudo possuí diversos destinos, que são: continuar em crescente andamento, ser executado até a substituição por um outro estudo que incorpore as características dele, ser mantido por tempo indeterminado, como ocorre com muitos estudos de aquecimento e resistência muscular ou pode simplesmente ser abandonado por motivos pessoais.

Aponta-se que existem três estados que um estudo assume durante o processo de prática na guitarra, são eles: estado de entendimento, estado de evolução e estado de domínio.

5.1.2.5 Fluência e degradação técnica

Com o objetivo de sistematizar e entender melhor o domínio do aplicativo em uma aplicação computacional, os entrevistados foram questionados sobre o tempo (em dias) em que demoram para começar a esquecer um estudo e precisariam novamente praticá-los, nos três estágios sugeridos na questão anterior. Deve ser destacado que tal questionamento é com base na rotina individual mantida. Segundo PETRUCCI (2000), a degradação técnica é o ponto em que algo começa lentamente a ser esquecido e não consegue mais ser executado com a mesma proficiência, para novos estudos a sua prática deve ser mais constante do que de estudos dominados, por isso a disciplina se torna essencial no desenvolvimento técnico.

As respostas dos entrevistados reforçam tal argumento, sendo que para estudos em estado de entendimento as respostas foram de média dois (1,8 arredondado), mediana dois e moda dois. Para estudos em estado de evolução as respostas foram de média quatro (3,8 arredondado), mediana três e moda três. Para um estudo dominado as respostas foram de média seis (6,4 arredondado), mediana seis e moda sete. Foi também perguntado se os entrevistados conseguem dizer quantos dias fazem que não praticam um determinado estudo de seu repertório e foi observado que não há um rastreio.

5.1.2.6 Percepção do tempo de dedicação

Um ponto importante sobre a justificativa de desenvolvimento do software é entender se os guitarristas entrevistados têm uma real percepção ao tempo de dedicação ao

desenvolvimento técnico do instrumento dentro de um dia, semana ou mês. Foi constatado que ao citar quantas horas dedicam ao instrumento por dia, nenhum possui controle sobre este tempo, pois para obter tal tempo teriam que somar todos os tempos dos estudos praticado naquele dia. Foi observado também que o tempo ao qual citam como "Quantas horas toca por dia", é apenas uma percepção subjetiva que engloba todo o tempo em que reservou a manutenção e desenvolvimento técnico com o instrumento, não necessariamente praticando com eficiência.

5.2 Elaboração de modelos

Esta etapa constituiu-se em transformar as informações obtidas durante o processo de levantamento de dados em modelos aplicáveis a um sistema computacional. Em primeiro momento buscou-se por uma equiparação a teoria de aprendizado e retenção motora, em segunda parte desenvolveram-se a proposta de prioridade de um estudo, a terceira parte demonstra-se a proposta de aproveitamento de tempo e por fim elaborou-se um diagrama de Casos de Uso para se ter uma visão abstrata da interação com o aplicativo.

5.2.1 Equiparação de estudos com modelos de aprendizado e retenção motora

Os estados que um estudo assume durante a evolução técnica dentro do estudo da guitarra são: **entendimento**, **evolução** e **domínio**. Segundo Masaki e Sommer (2012) a percepção de evolução de movimentos motores se torna cada vez mais difícil de sentir com o aumento da fluência de sequências de movimentos motores. Outro fato importante levantado pelo mesmo autor é que quanto mais se tem domínio, mais difícil de esquecer tal sequência com o passar do tempo. Isso ocorre porque temos uma rápida capacidade de aprender e existe um determinado ponto em que a evolução ocorrerá de maneira lenta, porém consolidada e por mais que ocorra o exagero em repetições de tal sequência de movimentos não ocorrerá evoluções significativas.

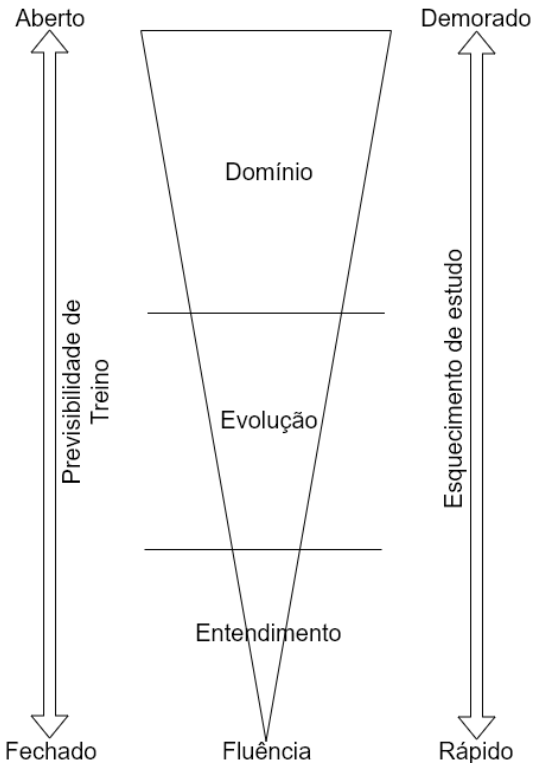
Quando um estudo está em estado de **entendimento**, ele necessita de constante prática, pois pode ocorrer rapidamente o seu total esquecimento e parecer como se nunca tivesse sido feito. Como citado por Krakauer et al. (2019), o reaprendizado antes do estado de **domínio** pode resultar em um aprendizado totalmente novo, como se nunca tivesse feito tal sequência motora. Em casos em que a sequência de movimentos já estava dominada, a recuperação pode ocorrer totalmente em tempos relativos. Adotou-se o conceito de que

quanto maior o domínio de um estudo, menor pode ser a sua frequência de prática, dentro de limites particulares de cada situação.

Um elemento importante sobre aprendizagem e aperfeiçoamento motor, é a forma de se organizar práticas de atividades motoras distintas, dentro de uma seção de treino, estes podem ser **abertas** e **fechadas**. Foi testado por Masaki e Sommer (2012), que quando se pratica de forma fechada atividades motoras parecidas ainda em fase de aprendizagem, se tem melhores resultados do que a prática aberta, este fato se reverte quanto a prática em atividades já dominadas. A sua justificativa é o aumento de atividade cerebral com a incerteza e variação de sequencias de atividades motoras. Segundo Krakauer et al. (2019) e Schmidt e Lee (2011), o treinamento aberto pode ser benéfico para a capacidade de adaptação motora. Pois a adaptabilidade é uma das principais habilidades que um instrumentista possui, tal importância é citada por Zatorre, Chen e Penhune (2007), como um dos principais elementos que um músico precisa para a habilidade em improvisar. Conclui-se que quanto maior o domínio de um estudo, mais ele poderá ser desconexo de sequencias em um treino.

Abaixo segue o modelo proposto:

Figura 3 – Modelo de aprendizado e retenção motora proposto



Fonte: Elaborado pelo autor (2021)

5.2.2 Prioridade de um Etude (estudo)

Para que o aplicativo faça a sugestão de estudos a serem praticados, precisa-se definir uma variável numérica de prioridade, números são elementos robustos para se criar ordenação e manipular listas em um sistema computacional. Tal variável por boa prática de programação e simplificação de código deve ficar entre zero e 100, pois elementos de interface que representam barras ou gráficos geralmente recebem valores de zero a 100. Acrescenta-se que os elementos gráficos terão o 100 (seu máximo) representando estudo em dia e o zero (seu mínimo) como um total afastamento do estudo, sendo o zero total prioridade de prática e 100 mínima prioridade.

Considerou-se os números obtidos sobre degradação técnica e esquecimento nas entrevistas realizadas e conciliou-se com os três estados em que um estudo se encontra e adotou-se três definições e valores: entendimento dois dias, evolução quatro dias e domínio seis dias. Conclui-se que quanto mais o tempo passa, os estudos definidos como entendimento devem ganhar prioridade em relação aos demais e evolução em relação ao domínio, sendo que ganhar prioridade significa ter um decréscimo proporcional em dias ao número cem. Outro ponto observado, é que o estudo técnico em um instrumento musical é algo pessoal e as variáveis de decréscimo em implementações definitivas do aplicativo devem ser configuráveis a cada usuário. A Figura 4 representa o cálculo de prioridade, em que na coluna "Decréscimo", atribui-se os valores sugeridos anteriormente, "Dias Passados" é uma simulação de quantos dias se passaram desde a última prática, "Total decréscimo" temos uma relação entre as colunas anteriores. Para que se tenha uma melhor percepção em relação ao nível máximo e mínimo de prioridade com o passar do tempo, adiciona-se um ajuste (*offset)* como uma constante com valor 25 dentro do sistema, que após multiplicar pelo Total decréscimo se subtrai do valor 100 para obter a prioridade. Este *offset* tornou-se necessário, pois a redução do valor e aumento de prioridade não causa problemas com ordenação, porém, não criaria uma boa sensação sobre prioridades, por sua vez, se o valor de *offset* for muito alto, teríamos em pouco tempo todas prioridade em zero fazendo o software perder sua capacidade de ordenação e proposta.

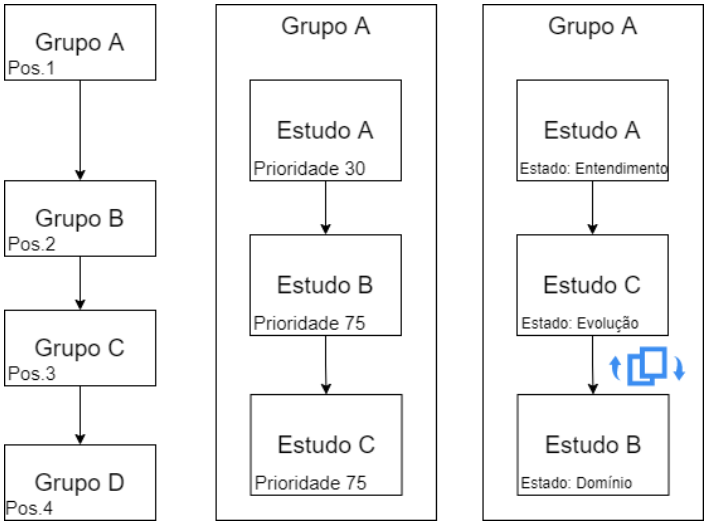
Figura 4 – Cálculo de prioridade para estudos

	Decréscimo	Dias Passados	Total decréscimo	*Offset*	Prioridade
Entendimento	2.0	**2.0**	1.0	25.0	**75.0**
Evolução	4.0	**2.0**	0.5	12.5	**87.5**
Domínio	6.0	**2.0**	0.3	8.3	**91.7**
Entendimento	2.0	**4.0**	2.0	50.0	**50.0**
Evolução	4.0	**4.0**	1.0	25.0	**75.0**
Domínio	6.0	**4.0**	0.7	16.7	**83.3**
Entendimento	2.0	**6.0**	3.0	75.0	**25.0**
Evolução	4.0	**6.0**	1.5	37.5	**62.5**
Domínio	6.0	**6.0**	1.0	25.0	**75.0**

Fonte: Elaborado pelo autor (2021)

Respeitando o modelo de aprendizado e retenção motora proposto, todos os estudos teriam sua prioridade calculada e seriam ordenados, todavia, observou-se que guitarristas utilizam grupos de estudos que tem sequência definida por eles e podem sofrer variação em sua ordem também estabelecida por eles, grupos também respeitam o conceito seriado de aprendizado motor, em que sequências parecidas podem ter um aproveitamento melhor se praticadas em um grupo. Levando em consideração tais elementos, sugeriu-se a ordenação de listas de estudos por grupos, dentro de cada grupo a ordenação por prioridade ascendente (o menor valor é o mais prioritário) e em empate o estado do estudo, sendo o nível de fluência ascendente. Destaca-se que o software irá sugerir tais sequencias e sempre manter aberta a escolha do praticante. A Figura 5 ilustra tal forma de ordenar, a primeira coluna representa a ordenação de grupos definida pelo usuário, a segunda coluna os estudos dentro de cada grupo ordenados por prioridade e na terceira coluna o resultado de um empate em que os grupos com estados diferentes possuem a mesma prioridade.

Figura 5 – Ordenação de estudos



Fonte: Elaborado pelo autor (2021)

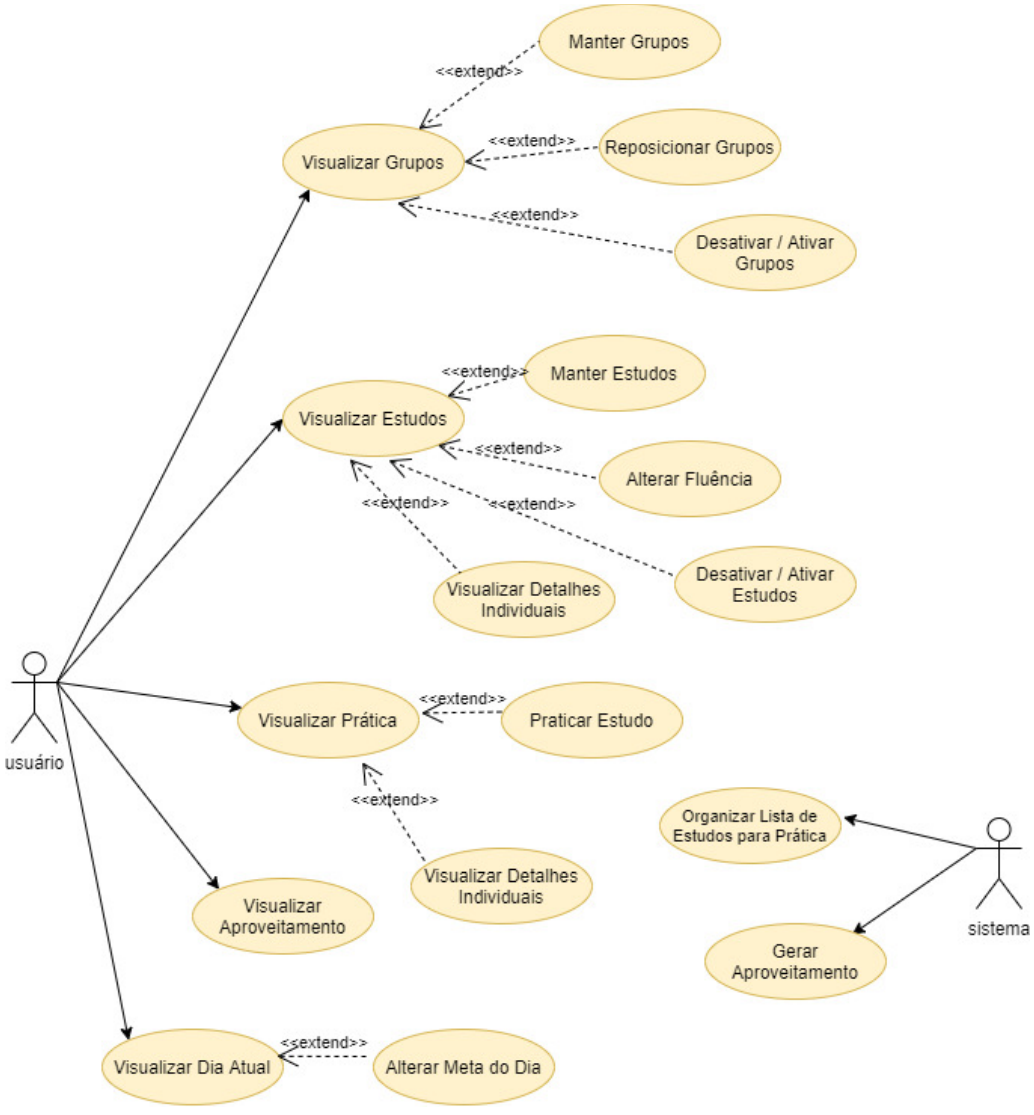
5.2.3 Tempo de aproveitamento de estudos

Foi observado que há a necessidade dos guitarristas entrevistados de contabilizar quanto tempo real foi dedicado ao estudo técnico em seu instrumento em um determinado intervalor de tempo, seja em um dia, semana ou mês. Para esta finalidade, existem muitas alternativas estatísticas possíveis de aplicação em modelos computacionais. Com isso, buscou-se uma alternativa em que o músico não fique focado observando gráficos estatísticos e painéis. Estes músicos precisam de facilidade em assimilar sua rotina, pois cada um possui uma meta diferente para o tempo de dedicação, por isso apostou-se em uma simbiose entre valores fixos dedicados e um aproveitamento dinâmico. Com este objetivo propôs-se um modelo em que este controle de dedicação é feito com base em um melhor valor dentro de um intervalo de tempo sugerido. Como exemplo, utilizando os últimos sete dias de estudos (uma semana), encontra-se o melhor dia e o relaciona com todos os outros dias com este valor, sendo obvio o melhor dia será 100% e os outros em relação com tal, acrescenta-se que quando um dia se passar, toda essa relação irá se alterar, criando um efeito contínuo de percepção de rotina, destaca-se que ao mesmo tempo que se observa o valor dinâmico o músico deve ter visualização de seu tempo total de prática em cada intervalo proposto.

5.2.4 Casos de uso

Casos de uso é uma técnica de modelagem de alto nível incorporada ao UML (*Unified Modeling Language*), segundo Sommerville (2011), esta técnica pode ajudar a descobrir quantos são os atores e em uma visão macro para entender melhor como tais atores irão interagir com o sistema. É representado pela Figura 6 o diagrama de casos de uso sugerido para o aplicativo deste trabalho.

Figura 6 – Diagrama de Casos de uso



Fonte: Elaborado pelo autor (2021)

5.3 Desenvolvimento do aplicativo

A etapa de desenvolvimento do aplicativo foi dividida em duas partes dominantes, a primeira que se consiste no desenvolvimento um servidor-API, utilizando .NET Core que utiliza a linguagem de programação C# e a segunda um cliente em React que foi desenvolvido utilizando a linguagem TypeScript no ambiente Node.js.

5.3.1 Desenvolvimento do Servidor API utilizando a tecnologia .Net Core 3.1

Adotou-se para o desenvolvimento do servidor do aplicativo o .Net Core 3.1 que é um *framework* desenvolvido pela Microsoft e possui licença MIT. A versão 3.1 foi escolhida por possuir suporte de longo termo e ser a última antes da implementação do .Net 5 que em resumo será uma padronização de todos os *frameworks* desenvolvidos pela Microsoft, sendo assim a escolha feita se torna uma opção segura.

Para ambiente de desenvolvimento foi escolhido o VS Code da Microsoft com uso da .Net CLI, por ser uma ferramenta muito utilizada pela comunidade de desenvolvimento web. Escolheu-se o uso do aplicativo Postman desenvolvido pela Postman Inc. por oferecer suporte a API RESTful e não necessidade do desenvolvimento de um cliente para testar o servidor do aplicativo. Tornando-se assim o desenvolvimento e teste da API mais robusto de característica procedimental e com separação de responsabilidades entre servidor e cliente, em resumo não há a necessidade se desenvolver um cliente para testes do servidor.

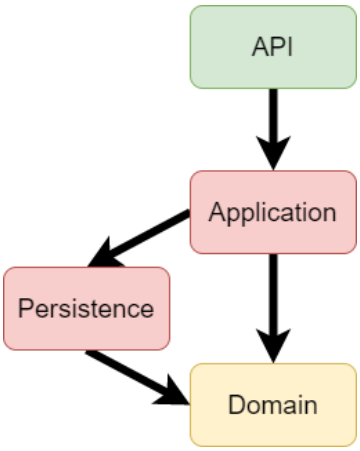
5.3.1.1 Arquitetura do Sistema

A arquitetura adotada é a *Clean Arquitecture* (Arquitetura Limpa), sugerida por Robert C. Martin. Segundo Martin (2017), o aplicativo deve ter como princípio de sua existência as entidades que formam a base de toda a lógica de domínio e por seguinte a lógica do aplicativo. No caso do programa em desenvolvimento, as entidades necessárias são: os grupos para organização de etudes, os etudes e os capítulos que são dias em que o guitarrista tocou e gerou dados. No caso deste aplicativo as entidades são apenas objetos sem comportamentos ou lógica específicos. Externa à camada de domínio, temos a camada de lógica da aplicação, chamada pelo autor de Use Cases, ela é responsável por tudo que é de responsabilidade do aplicativo e não deve interferir no domínio, esta camada depende da camada central e o inverso não deve ocorrer. Acima da camada de lógica da aplicação, existe

a camada de interfaces e conexões, que no caso dessa aplicação web é a API, esta camada depende da camada de lógica da aplicação, o inverso não ocorre e a camada de domínio desconhece sua existência. Por ser uma aplicação que necessite de persistência, entende-se a necessidade de separação de responsabilidade dentro da camada de aplicação para gerir tal incumbência, este novo serviço irá depender da camada de domínio e a camada de lógica de aplicativo depende desta. A camada de Persistência funciona como uma extensão da camada de aplicação para gerir a conexão e interação com banco de dados, que será externo e desconhecido pela camada de aplicativo e domínio.

Devido a Arquitetura Limpa ser de definições genéricas, Martin (2017) apresenta exemplos de aplicações da arquitetura proposta utilizando a terminologia direcionada a domínio, conhecida como DDD (Acrônimo para *Domain-Driven Design*). Sendo assim, jugou-se uma boa prática seguir tais conceitos aplicando-os em determinada terminologia. Portanto, as camadas e separação de responsabilidades empregadas receberam os nomes de *Domain* para domínio, *Application* para lógica do programa, API e *Persistence* para persistência, tornando o entendimento e implementação amigáveis. Na Figura 7, o diagrama ilustra a responsabilidade, dependência e camadas para o modelo de arquitetura proposto, observa-se que a seta aponta para qual camada há dependência.

Figura 7 – Diagrama arquitetura Servidor API



Fonte: Elaborado pelo autor (2021)

Destaca-se que a API nesta configuração fica com uma única responsabilidade, que é de receber *requests* e responder. Com isso, temos *controllers* pequenos sem responsabilidade ou conhecimento algum sobre a lógica do aplicativo. Na camada *Application*, temos a lógica do aplicativo responsável por lidar com os *requests* originários da

API e a camada *Domain* fica sendo o centro da aplicação com as entidades. Por fim, a camada *Application* requisita ou envia informações ao banco de dados usando a camada *Persistence* que tem como única responsabilidade interagir com o banco de dados que é um processo incógnito pela *Application*, fazendo o uso das entidades da camada de *Domain* como referência. Observou-se no desenvolvimento que mesmo com a criação de camadas, há a dependência transitiva entre canadas que não possuem dependência direta, reforçando assim a atenção ao desenvolver o aplicativo.

A implementação de tal arquitetura em .Net Core exige a criação de uma *Solution*, criação dos *Namespaces* que irão conter cada uma das responsabilidades e adição de dependências referentes a cada *Namespace*.

Utilizando a .Net CLI foram executados os seguintes comandos distribuídos em passos:

- Criando *Solution*.

```
dotnet new sln
```

- Criando *Namespaces*.

```
dotnet new classlib -n Domain
dotnet new classlib -n Application
dotnet new classlib -n Persistence
dotnet new webapi -n API
```

- Adicionando Projetos dentro da *Solution*.

```
dotnet sln add Domain/
dotnet sln add Persistence/
dotnet sln add Application/
dotnet sln add API/
```

Ressalta-se que ainda há a necessidade de se criar as referências entre os projetos que foram citadas anteriormente, deve ser lembrado que a camada central de entidades, não necessita de referências.

Utilizando a .Net CLI dentro de cada pasta de projeto foram executados os seguintes comandos distribuídos em passos:

- Referências da camada *Application*.

```
dotnet add reference ../Domain/
dotnet add reference ../Persistence/
```

- Referência da camada API.

```
dotnet add reference ../Application/
```

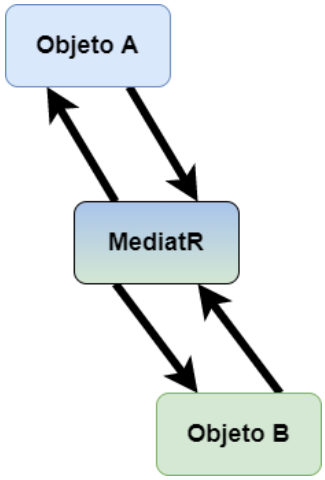
- Referências da camada *Persistence*.

```
dotnet add reference ../Domain/
```

Observa-se que como mencionado na revisão teórica deste trabalho, a Arquitetura Limpa proposta por Robert C. Martin possui também um fluxo de controle para que as camadas mantenham suas dependências propostas. Para isso adotou-se o *Mediator Pattern* com o uso da biblioteca MediatR, criada por Jimmy Bogard e de licença Apache 2.0. O MediatR tem como principal objetivo gerenciar interações entre diferentes objetos através de uma classe mediadora, com a intenção de reduzir o acoplamento entre elas. Com isso temos a manutenção de responsabilidade única entre classes e camadas diferentes, pois cada objeto não precisa conhecer a responsabilidade do outro.

Abaixo na Figura 8 segue diagrama representando o papel do MediatR como *Mediator Pattern*:

Figura 8 – Diagrama *Mediator Pattern* com MediatR

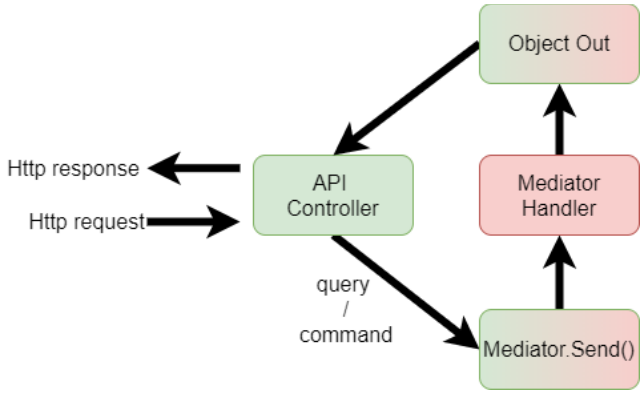


Fonte: Elaborado pelo autor (2021)

Aplicando-se o *Mediator Pattern* utilizando a biblioteca MediatR, temos um funcionamento que contempla o Fluxo de Controle proposto por Robert. C Martin. Pois a API sendo basicamente composta por *controllers* mínimos, se encarregarão de enviar *queries* ou *commands* para a camada *Application* através do método Mediator.Send(), que utilizará um *Mediator Handler* para executar o caso de uso ao qual é responsável, seja ele uma leitura de informações, inserção ou modificação no banco de dados. Logo após, ocorre o retorno de um objeto HTTP response para a API, finalizando assim o Fluxo de Controle proposto por Robert C. Martin.

A Figura 9, apresenta o diagrama demonstrando a aplicação do MediatR. Observa-se que a cor verde representa a API, a cor vermelha a camada *Application* e o papel do MediatR é um degrade entre as cores de ambas as camadas.

Figura 9 – Diagrama de aplicação do MediatR

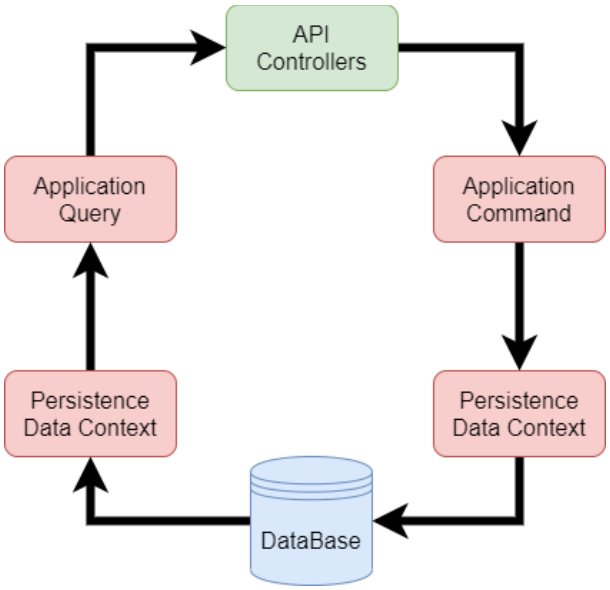


Fonte: Elaborado pelo autor (2021)

Seguindo boas práticas e o conceito da Arquitetura Limpa de Robert C. Martin, ao qual recomenda-se que um software deve crescer sem alteração do que já foi feito, foi atribuído o conceito de CQRS (*Command Query Resposability Segregation*). Que basicamente é tratar de maneira independente *commands* e *queries*, que resultará em melhor implementação de novos serviços, técnicas em que usam múltiplos banco de dados para escrita, leitura e *buffer*, entre outras necessidades. Com isso teremos um impacto mínimo ou nenhum na camada de *Application* quando ocorrer implementação de novos serviços. Destaca-se também que o aplicativo fica mais fácil de compreender e reforça a responsabilidade única de classes e métodos.

Abaixo na Figura 10 há a ilustração do diagrama demonstrando a aplicação do conceito CQRS no sistema desenvolvido com a utilização de um banco de dados, na figura a cor verde representa a Camada API, a vermelha a camada *Application*.

Figura 10 – CQRS aplicado ao sistema



Fonte: Elaborado pelo autor (2021)

Com isso, a arquitetura já se encontra definida, podendo ser implementada em código e executar testes. O primeiro fator a descrever é o ponto de entrada na API, que é feito com um *controller* que usando hierarquia de classes herda um *controller* base para evitar repetição de funcionalidades comuns, deixar o código mais limpo, expansível e de fácil entendimento, que são recomendações de Martin (2017) com os princípios do SOLID que neste caso são: SRP (*Single Responsability Principle* – Princípio da Responsabilidade Única), LSP (*Liskov Substituition Principle* – Princípio da Substituição de Liskov) e OCP (Open/Close Principle – Princípio do Aberto e Fechado). Pois foi atribuído única responsabilidade a um *controller*, todos *controllers* podem se comportar como o *controller* base se necessário e cada *controller* pode expandir sem alteração em funcionalidades existentes com o andar do desenvolvimento do software, pois as novas funcionalidades serão apenas adicionadas as classes *controllers*.

Na Figura 11, se encontra a implementação do *controller* base ao qual faz a injeção do serviço MediatR, que é inicializado na classe Startup do aplicativo, observa-se a definição de rota que será acrescida da responsabilidade de cada *controller* e repara-se que a classe herda ControllerBase que é parte da biblioteca Microsoft.AspNetCore.MVC, sendo um

controller com recursos suficientes para esta aplicação que não segue a arquitetura MVC, não necessitando de suporte a *views*. Também, há o atributo [ApiController] que especifica o comportamento da classe para funcionar com HTTP API Responses (Métodos de recursos da API REST).

Figura 11 – Classe BaseController

```
1   using MediatR;
2   using Microsoft.AspNetCore.Mvc;
3   using Microsoft.Extensions.DependencyInjection;
4
    ...
5   namespace API.Controllers {
        ...
6       [Route("api/[controller]")]
7       [ApiController]
8       public class BaseController : ControllerBase {
9           private IMediator _mediator;
10          protected IMediator Mediator => _mediator ??
11              (_mediator = HttpContext.RequestServices.GetService<IMediator>());
12      }
13  }
```

Fonte: Elaborado pelo autor (2021)

Com o BaseController criado, implementa-se os *controllers* que terão a responsabilidade de se comunicar com a camada *Application* para executar os casos de uso do sistema. Cada um desses *controllers* será responsável por um conjunto de funcionalidades que pertencem a uma mesma rota na API.

O *controller* representado pela Figura 12 possui duas funcionalidades e ilustra a implementação de todos os outros *conrollers* do sistema. A primeira funcionalidade chamada de Acquire, recebe o atributo [HttpGet] e tem como única responsabilidade aguardar o retorno de um objeto chamado PracticeEnvelope, que será criado dentro da camada *Application* na Classe Acquire do conteúdo relacionado utilizando uma estrutura de leitura (*Query*). A Segunda funcionalidade chamada de Done, recebe o atributo [HttpPost] e uma *string* na rota que será enviada como GUID (*Global Universal ID*) para à camada *Application* na classe Done do conteúdo relacionado utilizando uma estrutura de escrita (Command). Esta funcionalidade espera receber um objeto com o resultado da operação, pois todos os métodos nos *controllers* devem ter um HTTP Response e isso é feito utilizando um *struct* do MediatR chamado Unit.

Figura 12 – Classe PracticesController

```
using System.Threading.Tasks;
using Application.AppTrainer.Practices;
using Microsoft.AspNetCore.Mvc;
using MediatR;
using System;
...
namespace API.Controllers {
    ...
    public class PracticesController : BaseController {

        [HttpGet]
        public async Task<ActionResult<Acquire.PracticeEnvelope>> Acquire() {
            return await Mediator.Send(new Acquire.Query());
        }

        [HttpPost("{id}")]
        public async Task<ActionResult<Unit>> Done(Guid id) {
            return await Mediator.Send(new Done.Command { Id = id });
        }
    }
}
```

Fonte: Elaborado pelo autor (2021)

A lógica do aplicativo foi implementada na camada *Application*, para isso, há a necessidade do uso de *handlers* para executar as funcionalidades. O exemplo na Figura 13, ilustra a estrutura de uma funcionalidade com o objetivo de escrita ou alteração em que o serviço *Persistence* se encarregará de fazer. No caso desta funcionalidade, há o recebimento de um Id vindo do *controller* utilizando recursos do MediatR, o construtor do *handler* inicializa apenas os serviços pertinentes a funcionalidade e depois há a execução da tarefa de forma assíncrona (Ao qual se encontra minimizada na figura e será detalhada mais a frente).

Figura 13 – *Command handler* exemplo

```
namespace Application.AppTrainer.Collections {
    ...
    public class FluenceLearning {
        ...
        public class Command : IRequest {
            public Guid Id { get; set; }
        }
        ...
        public class Handler : IRequestHandler<Command> {

            private readonly DataContext _context;
            private readonly IUserAccessor _userAccessor;

            public Handler(DataContext context, IUserAccessor userAccessor) {
                _userAccessor = userAccessor;
                _context = context;
            }

            public async Task<Unit> Handle(Command request,
                CancellationToken cancellationToken) {...
        }
    }
```

Fonte: Elaborado pelo autor (2021)

A Figura 14 representa a funcionalidade mencionada anteriormente, servindo de ilustração para as implementações das tarefas *command* na camada *Application* do sistema. Nesta tarefa, temos primeiramente a aquisição de informações pertinentes, que no caso são: o usuário logado, que é encontrado através de uma interface em outro serviço (*Infrastructure*) e o Etude em questão que além de ser encontrado pelo Id, também leva em consideração o usuário logado para aumento de segurança além das políticas de autorização. Abaixo tem-se os tratamentos de exceções, que são importantes para teste enviando o código de falha ao cliente, garantindo melhor entendimento do erro, isso ocorre através de uma *middleware* implementada na própria API. Após a etapa de controle de exceções, o principal objetivo da tarefa é feito, alterar o campo do Etude que representa o nível de fluência. Se ocorrer alguma modificação, o contexto de dados é salvo, e como é salvo é responsabilidade da camada *Persistence*, pois ela encapsula o contexto de dados. Por fim, ocorre o retorno do resultado dessa operação ao *controller*.

Figura 14 – Tarefa *command handler* exemplo

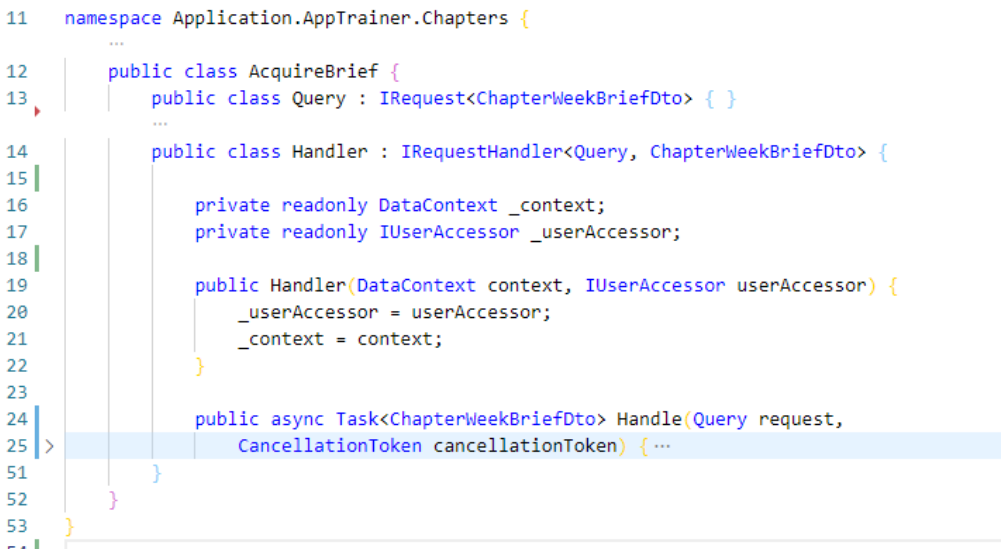
```
public async Task<Unit> Handle(Command request, CancellationToken cancellationToken) {

    var learningCode = 1;

    var user = await _context.Users.SingleOrDefaultAsync
        (x => x.UserName == _userAccessor.GetCurrentUsername());

    var etude = user.Etudes.FirstOrDefault
        (x => x.Id == request.Id && x.AppUserId == user.Id);

    if (etude == null)
        throw new Errors.RESTException(HttpStatusCode.NotFound,
        new { etude = "Not Found" });

    if (etude.Fluence == learningCode)
        throw new Errors.RESTException(HttpStatusCode.Forbidden,
        new { etude = "It´s already this state" });

    etude.Fluence = learningCode;

    var success = await _context.SaveChangesAsync() > 0;

    if (success) return Unit.Value;

    throw new Exception("Error saving Etude to learning");
}
```

Fonte: Elaborado pelo autor (2021)

Usando como exemplo de implementação de um *query handler*, a Figura 15 representa a classe AcquireBrief que tem como funcionalidade fazer um resumo dos últimos 7 dias de prática do usuário. A leitura através de *queries* tem estrutura de implementação

parecida com a de *command,* embora tenham objetivos completamente diferentes. *Queries* esperam um determinado tipo de objeto como retorno, podendo ser uma coleção (uma lista) ou não. No caso referido, ela aguarda um objeto que é uma instancia da classe ChapterWeekBriefDTO (Ao qual se encontra minimizada e será detalhada mais a frente).

Figura 15 – *Query handler* exemplo



Fonte: Elaborado pelo autor (2021)

Na Figura 16, temos a representação da tarefa executada pelo *query handler* descrito na Figura 15. Esta tarefa tem na sua parte superior a aquisição de informações pertinentes através dos serviços inicializados no construtor do *handler,* que no caso é o usuário logado e uma lista dos últimos sete dias. Com essa lista armazenada em uma variável, são somados os tempos de cada dia e quantos Etudes foram feitos através de uma estrutura de repetição (foreach). Após esse processo, os dados obtidos são utilizados para a criação de um objeto da classe ChapterWeekBriefDTO e retornado ao *controller*.

Figura 16 – Tarefa *query handler* exemplo

```
public async Task<ChapterWeekBriefDto> Handle(Query request,
    CancellationToken cancellationToken) {

    var user = await _context.Users.SingleOrDefaultAsync
        (x => x.UserName == _userAccessor.GetCurrentUsername());

    var queryable = _context.Chapters
        .Where(a => a.AppUserId == user.Id && a.Day > DateTime.Today.AddDays(-7))
        .AsQueryable();

    var chapters = queryable.ToList();
    var totalEtudes = 0;
    var totalTime = 0;

    foreach (var chapter in chapters) {
        totalEtudes += chapter.TotalEtudes;
        totalTime += chapter.TotalTime;
    }

    var chapterWeekBrief = new ChapterWeekBriefDto {
        TotalEtudes = totalEtudes,
        TotalTIme = Math.Round((double)totalTime / 60, 1),
        AverageDay = Math.Round(((double)totalTime / 7) / 60, 1)
    };

    return chapterWeekBrief;
}
```

Fonte: Elaborado pelo autor (2021)

O serviço de *Persistence*, é responsável por gerir persistência de dados e tem como elemento principal o Microssoft Entity Framework. Este foi escolhido por ser desenvolvido em um grande grau de compatibilidade para os *frameworks* .Net da Microssoft e por permitir manipular objetos no sistema utilizando o conceito de ORM (*Object-Relational Mapping*). Tal conceito permitiu criar o contexto de dados se baseando na camada de *Domain* que contém as entidades do sistema. A Figura 17 representa o contexto de dados criado para utilização no aplicativo.

Figura 17 – Contexto de Dados na camada *Persistence*

```
using Domain;
using Domain.App;
using Domain.AppTrainer;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

...
namespace Persistence {
    ...
    public class DataContext : IdentityDbContext<AppUser> {
        public DataContext(DbContextOptions options) : base(options) {
        }

        // Entities that wil be generated by Entity FrameworkCore
        public DbSet<Photo> Photos { get; set; }
        public DbSet<Tome> Tomes { get; set; }
        public DbSet<Etude> Etudes { get; set; }
        public DbSet<Chapter> Chapters { get; set; }

    }
}
```

Fonte: Elaborado pelo autor (2021)

Por se tratar de um framework ORM e a escolha do banco de dados não ter grau de importância no desenvolvimento para este aplicativo, foi escolhido para ser utilizado em modo de desenvolvimento o banco de dados SQLite. Porém, em modo de produção foi utilizado o banco de dados SQLServer. A utilização de dois bancos de dados possibilitou flexibilidade e melhor uso do ambiente de desenvolvimento, pois o SGBD SQlite não necessita de instalação no computador, que é feita via NuGet local ao aplicativo em desenvolvimento, sua criação e manipulação também é facilmente atingida utilizando comandos do Entity Framework Core, tornando o desenvolvimento inicial menos burocrático. Por sua vez o SGBD SQLServer foi utilizado em modo de produção e hospedado em servidores e atualizado utilizando *migrations* estáveis. Tal configuração é feita automaticamente com a criação de uma condicional no Startup.cs do aplicativo e aquisição de um plugin via Nuget que contemple a compatibilidade entre Entity Framwork Core e o SGBD escolhido, como é demonstrado na Figura 18. Os comandos para troca entre modo de desenvolvimento e produção são respetivamente: $Env:ASPNETCORE_ENVIRONMENT = "Development" e $Env:ASPNETCORE_ENVIRONMENT = "Production".

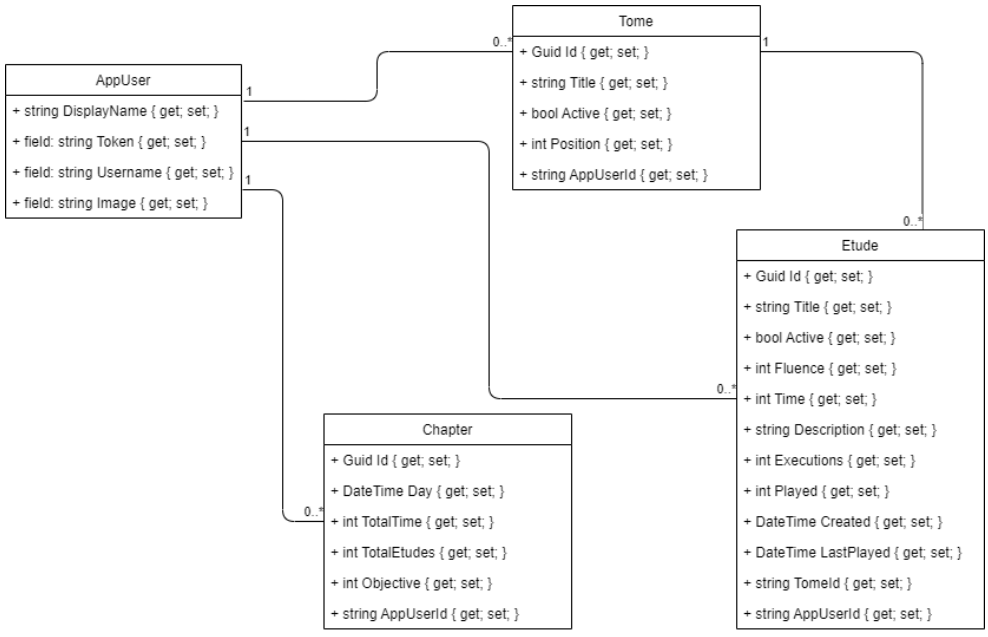
Figura 18 – Troca de SGDB no Startup.cs

```
public void ConfigureDevelopmentServices(IServiceCollection services) {
    services.AddDbContext<DataContext>(opt => {
        opt.UseLazyLoadingProxies();
        opt.UseSqlite(Configuration.GetConnectionString("DefaultConnection"));
    });

    ConfigureServices(services);
}

public void ConfigureProductionServices(IServiceCollection services) {
    services.AddDbContext<DataContext>(opt => {
        opt.UseLazyLoadingProxies();
        //opt.UseMySql(Configuration.GetConnectionString("DefaultConnection"));
        opt.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"));
    });

    ConfigureServices(services);
}
```

Fonte: Elaborado pelo autor (2021)

No processo de desenvolvimento de software, se torna necessário identificar as entidades através de um diagrama de classes UML em qual o *framework* ORM Entity Framework Core irá intermediar com qualquer que seja o SGBD. O diagrama de classes a seguir, representado pela Figura 19 representa tais entidades, onde AppUser é o usuário do sistema, Tome representa Grupos, Etude representa os estudos e Chapter, representa cada dia de prática registrado.

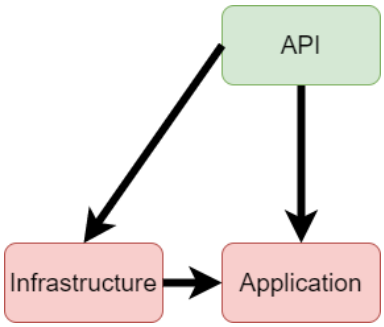
Figura 19 – Diagramas de classes de entidades



Fonte: Elaborado pelo autor (2021)

Destaca-se que sendo pertinente ao objetivo do trabalho, que é o desenvolvimento de um sistema web, se torna necessário garantir segurança do aplicativo. Para isso foi fragmentada a responsabilidade da camada de *Application* com um novo *namespace* chamado de *Infrastructure*, que tem responsabilidades sobre políticas de acesso, JWT (Jason Web Token), entre outras que possam ocorrer futuramente. A camada de *Infrastructure* foi colocada como dependência da API e dependendo da camada *Application* como demonstrado na Figura 20, pois se entende que acesso e segurança é pertinente ao aplicativo e não ao domínio. Observa-se na figura que a camada API está em verde e *Application* e *Infrastructure* em vermelho, pois ambas pertencem a camada de aplicação com responsabilidades diferentes. As setas apontam para qual há dependência.

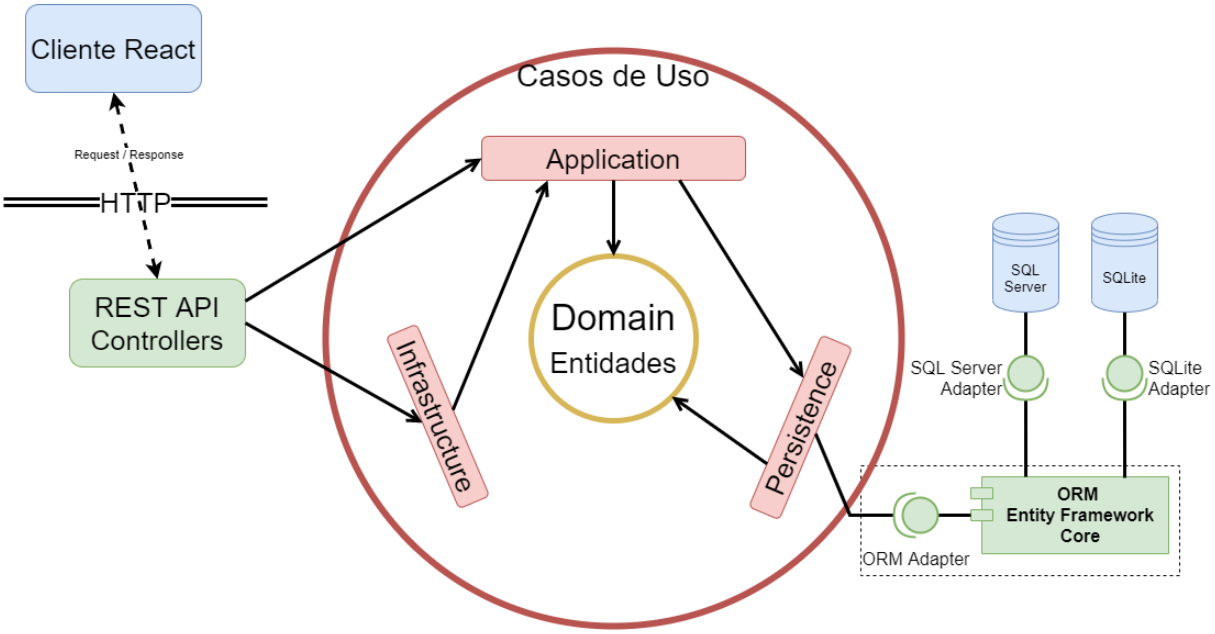
Figura 20 – Dependências do serviço *Infrastructure*



Fonte: Elaborado pelo autor (2021)

Após a definição de todos os papeis e responsabilidades dentro do sistema, a Figura 21 ilustra o modelo final obtido. Observa-se que esta ilustração já demostra em que lugar o cliente irá ficar. A figura representa o domínio do aplicativo no interior, ao qual não tem dependências, a camada de aplicação dividida em três responsabilidades, a API com *controllers* mínimos e o módulo responsável por gerenciar o banco de dados com o conceito ORM.

Figura 21 – Arquitetura limpa aplicada



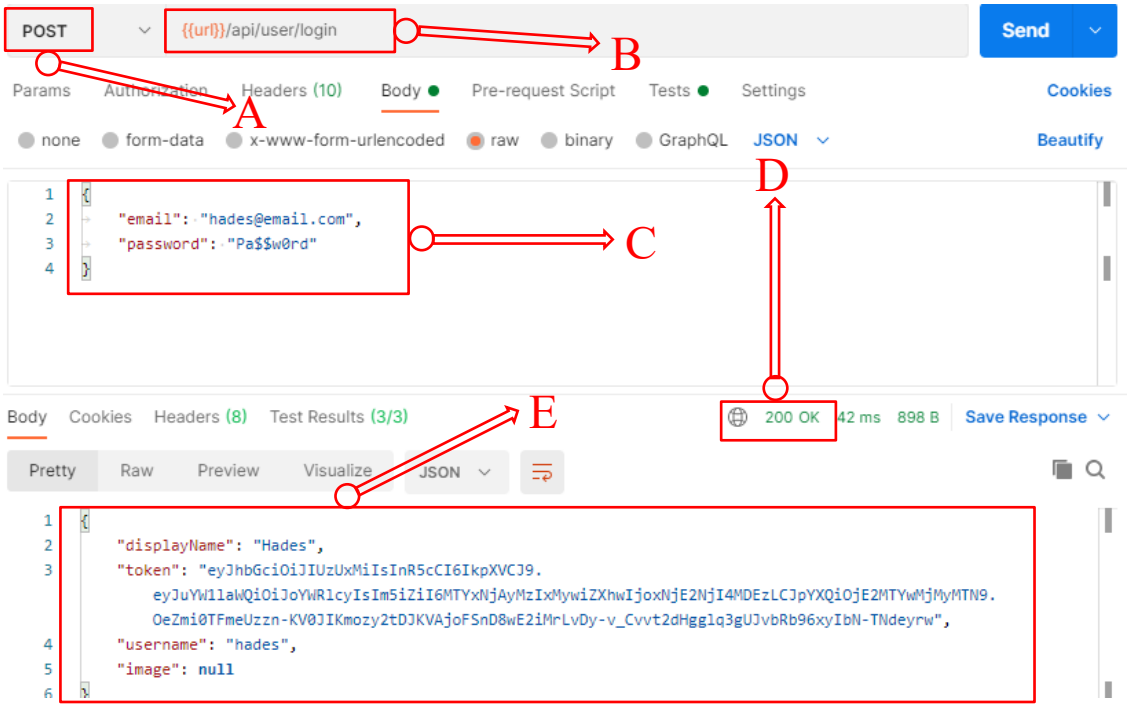
Fonte: Elaborado pelo autor (2021)

5.3.1.2 Teste do Servidor API

O teste foi feito utilizando a ferramenta Postman, que tem como principal objetivo fazer o papel de cliente. Nesta ferramenta podemos fazer o login no aplicativo, salvar o JWT (*JSON Web Token*), enviar *requests* com um método HTTP à API e obter respostas. No desenvolvimento do aplicativo, todos as funcionalidades foram testadas desta maneira, assim validando o funcionamento no servidor API e dando liberdade no desenvolvimento do cliente, sabendo exatamente em que parte do sistema as falhas estão ocorrendo.

A Figura 22 composta pelas marcações A, B, C, D e E, representa o resultado de uma tentativa bem-sucedida de login no sistema desenvolvido. Na marcação A, temos o método HTTP POST usado para tal operação. Na marcação B, temos o recurso que será acessado na API utilizando a rota user/login. A marcação C, há o corpo do *request*, que é composto pelo *email* e *password* do usuário tentando fazer o acesso. A marcação C visualiza-se a resposta do *request,* que no caso o 200 é um padrão para uma operação bem-sucedida. E por fim, na marcação E temos a resposta, que é composto por informações básicas sobre o usuário que serão utilizadas no cliente e o JWT que é armazenado no Postman de forma relativamente parecida com o que o navegador faz, que será utilizado para todos os recursos a serem requisitados na API.

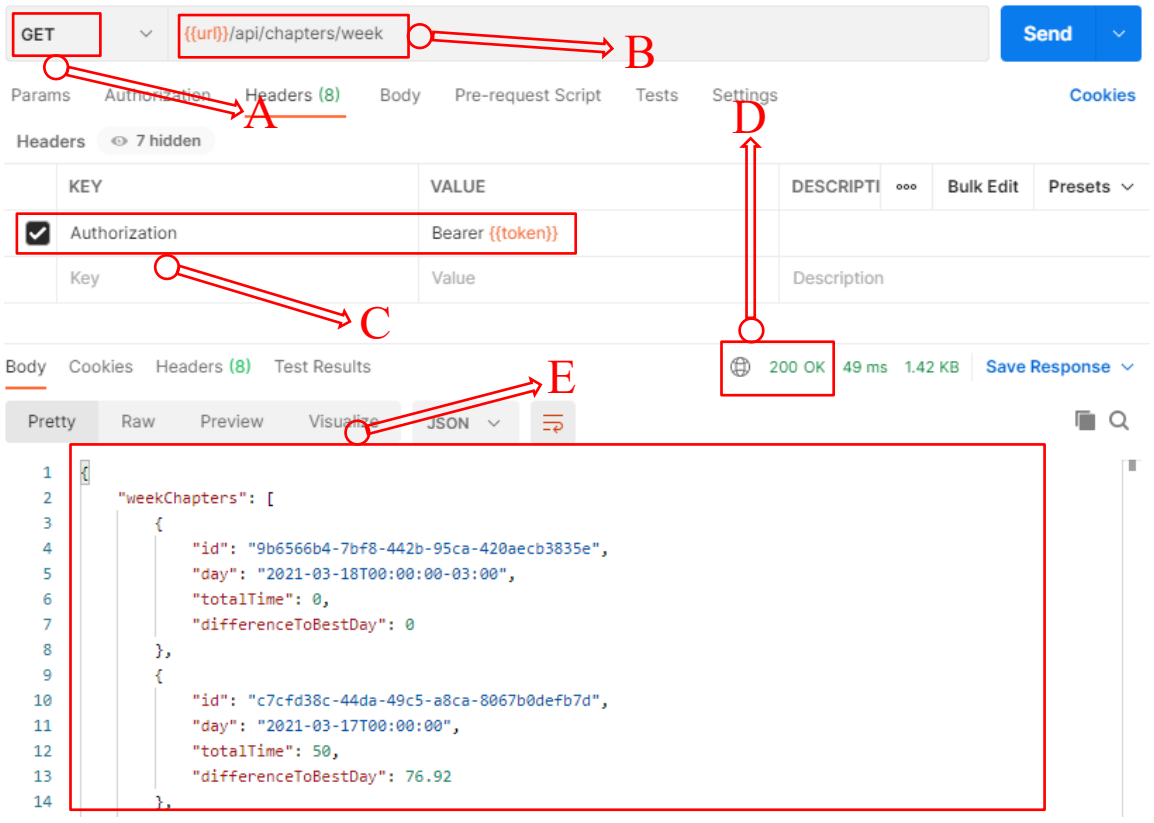
Figura 22 – Postman Login de Usuário



Fonte: Elaborado pelo autor (2021)

A Figura 23 compostas pelas marcações A, B, C, D e E, representa o resultado de uma tentativa bem-sucedida de requisição de uma semana de estudos, composta pelos 7 últimos dias em que o usuário registrou tempo de prática do instrumento. Neste *request*, há a utilização do método HTTP GET, identificado pela marcação A. Na marcação B, temos a rota /api/chapters/week para acessar tal recurso na API. Na marcação C, temos o envio de um cabeçalho que contêm o JWT que será estruturado pelo servidor. Na marcação C há o código 200 que é um padrão para uma operação bem-sucedida. E finalmente a marcação E, que demonstram em formato JSON o resultado da operação, que é uma lista de etudes com as informações pertinentes.

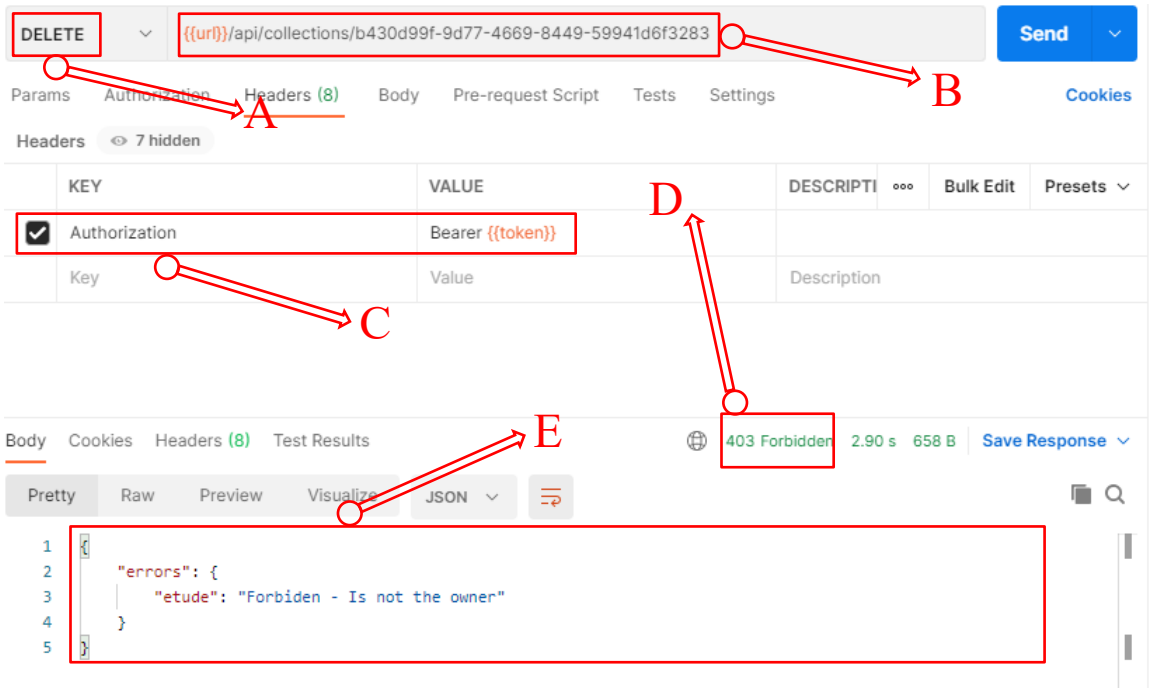
Figura 23 – Postman Método GET



Fonte: Elaborado pelo autor (2021)

A Figura 24 composta pelas marcações A, B, C, D e E, demonstra o processo de exclusão e uma tentativa malsucedida, em que um usuário logado pode tentar acessar o recurso da API diretamente em posse de um ID de um determinado etude que não lhe pertence, para isso é utilizado o método HTTP DELETE, demonstrado na marcação A. Na marcação B, observa-se o emprego da rota /api/collections/ sucedida pelo ID do etude. Na marcação D, observa-se o erro 403 apresentado, que ao contrário do 401 identifica proibição independente te estar logado. E por fim, a marcação E demonstra o erro apresentado de uma forma mais especifica, que no caso é uma proibição devido a não ser o dono do item que se busca deletar. Se o usuário fosse o dono do item, teríamos o código 200 retornado e um JSON vazio representado com {}.

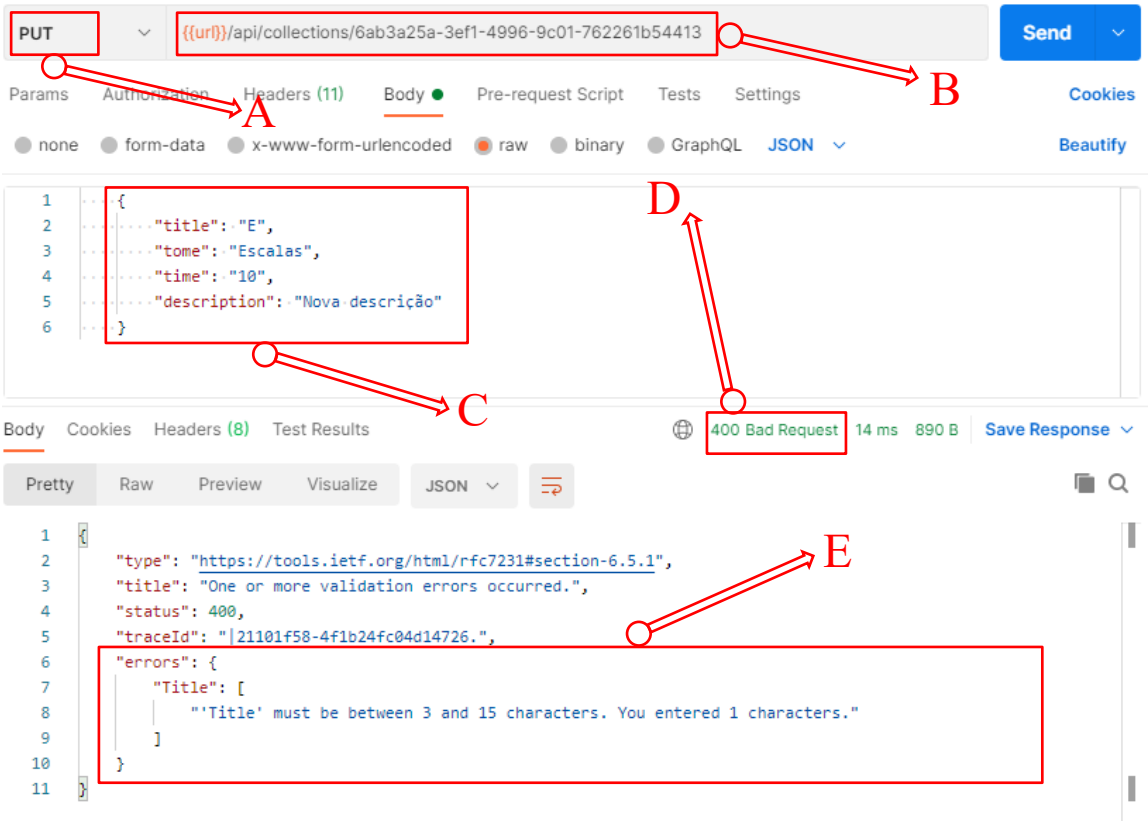
Figura 24 – Postman Método DELETE e erro



Fonte: Elaborado pelo autor (2021)

Por fim, a Figura 25 com as marcações A, B, C, D e E, ilustra o método HTTP utilizado para *requests* de atualização em uma operação malsucedida em que a validação do servidor identificou algo fora de suas atribuições, que no caso é o título de Etude menor do que 3 caracteres. Na marcação A temos o método HTTP PUT utilizado para atualização. Na marcação B, observa-se o emprego da rota /api/collections/ sucedida pelo ID do Etude que é apontada no cliente pelo usuário. Na marcação C, temos um corpo do *request*, que são as novas informações a serem atualizadas. Nota-se que, na ficha Headers no Postman faz o envio do cabeçalho como o token salvo no ato de login, semelhante a marcação C da Figura 18. Na marcação D temos o código 400, que identifica um erro que nesse caso fica evidente na marcação E, composto por informações detalhadas sobre o erro, que no caso é o campo *title* não ter caracteres suficientes. Se o *request* fosse bem-sucedido, teríamos o código 200 e um JSON vazio representado com {}.

Figura 25 – Postman Método PUT e erro



Fonte: Elaborado pelo autor (2021)

5.3.2 Desenvolvimento do Cliente Web utilizando React

Adotou-se para o desenvolvimento do cliente web do aplicativo a tecnologia React como principal elemento. O React não possui um tratamento para HTTP, necessitando uma biblioteca para tal função, no caso foi escolhido o Axios. Para armazenar, atualizar e observar estados da aplicação foi utilizado o MobX. Por fim, o Semantic-UI foi escolhido para criação dos elementos visuais da aplicação, assim formando os integrantes mais notáveis do aplicativo web cliente. Deve ser observado que todos citados são de licença MIT e a linguagem de programação utilizada é o TypeScript. Destaca-se que o cliente é desenvolvendo utilizando o Node.js e o gerenciador de pacotes npm.

5.3.2.1 Implementação do Axios

O Axios tem como responsabilidade fazer a comunicação com o servidor API, gerenciando *requests* e *responses*. Isto inclui tratar de maneira distinta cada um dos métodos

HTTP, gerenciar cabeçalhos e informações enviadas e recebidas em formato JSON esperados pelo servidor API. Em resumo o Axios fará a conexão com o servidor API e se encarregará de acessar recursos, semelhante ao que foi feito no Postman na sessão anterior de desenvolvimento do servidor API. Para implementação do Axios, foi utilizado um arquivo chamando agent.ts.

Primeiramente tratou-se do cabeçalho que é incluído em cada *request,* onde se integra o JWT que foi recebido pelo cliente quando o usuário efetua o login e armazenado temporariamente no dispositivo em que o aplicativo cliente encontra-se em execução. Para tal intenção o axios oferece a funcionalidade chamada *Interceptors* que podem acessar informações e configurá-las antes de *then* ou *catch*. Sendo assim, basta interceptar *requests* e adicionar o JWT a configuração de autorização do Axios. A Figura 26 ilustra tal implementação dentro do arquivo agente.ts. Primeiramente temos a configuração de um *Interceptor* que agirá em *requests*, fará o resgate do JWT no dispositivo do cliente e em seguida constituirá a configuração do cabeçalho de autorização do *request* executado.

Figura 26 – Axios configuração de *header* com JWT

```
axios.interceptors.request.use(
  (config) => {
    const token = window.localStorage.getItem("jwt");

    if (token)
      config.headers.Authorization = `Bearer ${token}`;
      return config;
  },
  (error) => {
    return Promise.reject(error);
  }
);
```

Fonte: Elaborado pelo autor (2021)

Com o cabeçalho efetivado, se torna possível a implementação dos métodos HTTP que o Axios irá executar. Para isso, deve-se definir os nomes dos métodos a serem utilizados, suas propriedades que deve incluir a URL do recurso da API e utilizando a forma de implantação *lambda* a definição dos métodos do Axios a serem utilizados. A Figura 27 ilustra como tal objetivo foi alcançado, em primeiro momento temos a definição do nome do método a ser utilizado dentro da aplicação cliente, suas propriedades que incluem URL com o possível adicional de um corpo para os métodos HTTP que necessitem. Deve ser lembrando que todos *requests* serão interceptados pelo *interceptor* do Axios e adicionado o cabeçalho.

Destaca-se também que todos os *requests* esperam uma resposta, que podem ser utilizados para interceptar erros diretamente no console do navegados ao qual o cliente é executado.

Figura 27 – Axios configuração de *requests*

```
77  const requests = {
78    get: (url: string) => axios.get(url).then(responseBody),
79    post: (url: string, body: {}) => axios.post(url, body).then(responseBody),
80    put: (url: string, body: {}) => axios.put(url, body).then(responseBody),
81    delete: (url: string) => axios.delete(url).then(responseBody),
82    postCommand: (url: string) => axios.post(url).then(responseBody),
83  };
```

Fonte: Elaborado pelo autor (2021)

Com a definição das propriedades dos métodos HTTP implementados com a utilização do Axios, torna-se possível a utilização de tais funcionalidades em casos de usos do aplicativo cliente a ser desenvolvido. A Figura 28 demonstra a implementação de tais métodos em um dos recursos da API, tal recurso é responsável por toda manipulação da coleção de Etudes de um único usuário. A primeira funcionalidade chamada de *get*, espera um objeto IUserCollection, que é composto por uma lista do modelo IEtude (em TypeScript podemos especificar que tipo de dado é esperado quando uma *promise* se efetua e é comum o uso de interfaces para modelagem de dados), repara-se que há a atribuição da URL do recurso da API que por sua vez se encarregará de detectar o usuário que faz o *request* através do JWT e irá retornar a lista de Etudes de propriedade deste usuário. Seguindo a implementação temos a funcionalidade *create*, responsável pela criação de um novo Etude, para isso deve ser enviado junto a URL do recurso API o corpo contendo os dados do novo Etude. Para editar foi criada a função *edit* que tem um funcionamento parecido com o processo de criação, exceto pelo fato de que aguarda um ID junto a URL para apontar qual Etude deve ser editado no servidor. A funcionalidade *details* serve exclusivamente para resgatar detalhes sobre um determinado Etude para edição e adiciona junto ao recurso da API o ID do Etude em questão. Foram criadas três funcionalidade para manipulação do estado do Etude, se está em processo de aprendizado, desenvolvimento ou fluência, tais funcionalidade são compostas pelo recurso da API, o ID do Etude e um acréscimo a URL que identifica tal intenção para não haver conflito, pois todos acessam o mesmo recurso da API com o mesmo método HTTP. Tal processo também é valido para a troca de estado do Etude, chamada de *changeActive.* Por fim a funcionalidade *delete* é composto pela URL e o ID do Etude a ser deletado.

Figura 28 – Axios acesso a recursos do Servidor API

```
131  const UserCollection = {
132    get: (): Promise<IUserCollection> => requests.get("/collections"),
133    create: (etude: IEtude) => requests.post("/collections", etude),
134    edit: (etude: IEtude) => requests.put(`/collections/${etude.id}`, etude),
135    details: (id: string) => requests.get(`/collections/${id}`),
136    fluenceLearning: (id: string) => requests.post(`/collections/${id}/fluencelearning`, {}),
137    fluenceEvolution: (id: string) => requests.post(`/collections/${id}/fluenceevolution`, {}),
138    fluenceFlowing: (id: string) => requests.post(`/collections/${id}/fluenceflowing`, {}),
139    delete: (id: string) => requests.delete(`/collections/${id}`),
140    changeActive: (id: string) => requests.post(`/collections/${id}/changeactive`, {}),
141  };
```

Fonte: Elaborado pelo autor (2021)

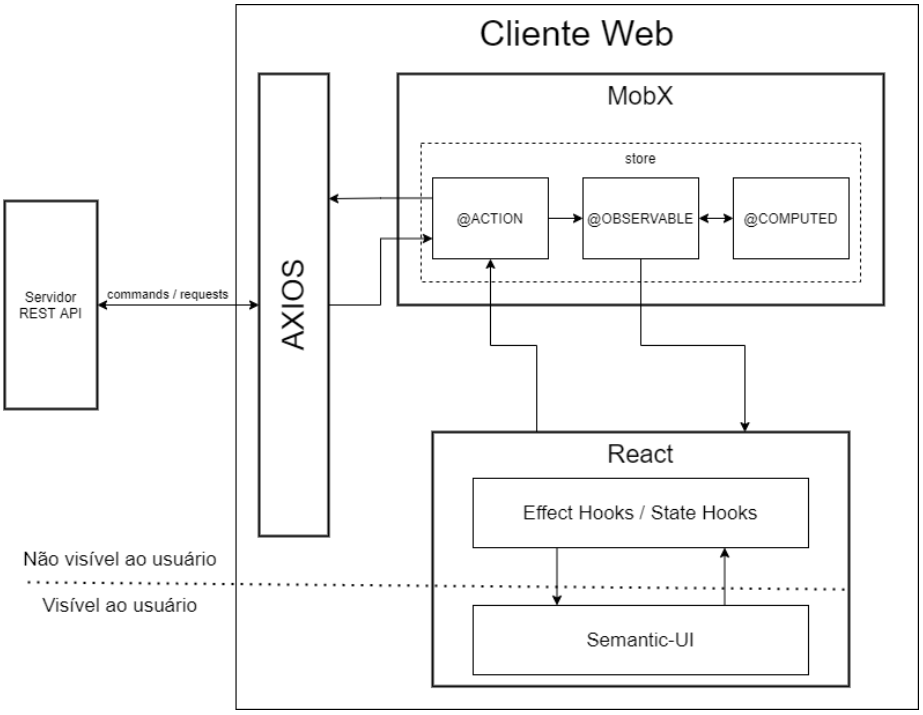
5.3.2.2 Atualização de estados, implementação MobX

A aplicação cliente *web* implementada com a utilização do React Native, não oferece uma forma de atualização de estados e gerenciamento de componentes observáveis, por este fato, se torna necessária a escolha e utilização de uma biblioteca destinada a tal fim. Para este trabalho foi escolhida a biblioteca MobX de licença MIT mantida por empresa de mesmo nome.

O MobX tem o conceito de criação de *stores*, que em síntese tem a intenção de remover lógica e estados de componentes para uma classe gerenciadora de estados que faz o intermédio entre modelos de tipo de dados, acesso ao *agent* HTTP e atualização de estados de elementos nos componentes observáveis do React. O princípio de funcionamento é baseado na interação do usuário com elementos da interface, em que através de ações o MobX altera estados de observáveis e atualiza em qualquer elemento gráfico ao qual foi implementado para fazer.

A Figura 29 representa o papel do MobX junto a outros do sistema desenvolvido neste trabalho, observa-se que o MobX faz toda a interseção e controle sobre todos os dados que são recebidos e enviados pelo usuário. Quando o usuário interage com um componente de interface ou o componente é programado para executá-la, há uma ação dentro do MobX, chamada de @action, essas ações têm o poder de modificar estados de variáveis que por sua vez atualizarão nos componentes de interface. Para derivações que são calculadas dentro de variáveis observáveis sendo alteradas pode-se utilizar java script *getters* com @computed.

Figura 29 – Arquitetura Cliente *web*



Fonte: Elaborado pelo autor (2021)

A Figura 30, demonstra a implementação da *store* responsável pelas funcionalidades relacionadas ao componente de listagem de estudos para prática. Na parte superior do código há a instância da *rootSotre,* seguindo encontra-se as variáveis que possuem seu estado observável. A ação setTargetDone tem o objetivo de armazenar na variável targetDone qual cartão foi clicado pelo usuário através de sua ID, isto se tornou necessário para o componente de carregamento, que substitui o conteúdo do botão ser executado apenas no cartão clicado. Seguindo o código a ação loadUserPractice tem o objetivo de requisitar do axios a listagem de estudos para prática e armazenar na variável userPractice que espera receber uma lista de estudos. Nota-se que toda alteração de estados feitas de forma assíncrona dentro de uma ação deve estar dentro da função runInAction do MobX. Destaca-se que as variáveis observáveis loading e loadingUserPractice são utilizadas para componentes de carregamento e tem seus estados alterado para falso e verdadeiro conforme necessário para transmitir respostas ao usuário na interface com uso de um componente padrão para carregamento. Repara-se que caso ocorra uma falha no bloco try catch, o usuário será alertado com um erro mencionando a falha através de um elemento gráfico *toast*.

Figura 30 – Exemplo MobX *store*

```
export default class UserPracticeStore {
  rootStore: RootStore;

  constructor(rootStore: RootStore) {
    this.rootStore = rootStore;
  }

  @observable userPractice: IUserPractice | null = null;
  @observable loadingUserPractice = true;
  @observable loading = false;
  @observable targetDone = "";

  @action setTargetDone = async (id: string) => {
    runInAction(() => {
      this.targetDone = id;
    });
  };

  @action loadUserPractice = async () => {
    this.loadingUserPractice = true;
    try {
      const userPractice = await agent.UserPractice.get();
      runInAction("loadUserPractice", () => {
        this.userPractice = userPractice;
        this.loadingUserPractice = false;
      });
    } catch (error) {
      runInAction("loadUserPractice error", () => {
        this.loadingUserPractice = false;
      });
      toast.error("👎 Erro carregando Prática.");
    }
  };
```

Fonte: Elaborado pelo autor (2021)

A Figura 31, demonstra uma ação que emprega a alteração de estados em duas *stores* diferentes, que são responsáveis por dois componentes diferentes, em que estarão visíveis ao usuário simultaneamente, sendo: o cabeçalho responsável pelo aproveitamento do dia e a funcionalidade que lista a prática recomendada de estudos. Esta ação, recebe do componente React o estudo (Etude) em que o usuário clicou no botão de “praticado”. O id deste estudo é enviado ao Axios que se encarregará de se comunicar com o servidor. Dentro da função runInAction, há uma condicional, que em primeiro momento soma ao total de tempo e quantidade de estudos praticados e caso não exista estudo no dia é iniciado um novo dia (capítulo). Destaca-se que mensagens para o usuário são enviadas em forma de *toast* com informações em cada um dos possíveis fins dessa ação.

Figura 31 – Exemplo MobX com duas *stores*

```
@action setEtudeDone = async (etude: IEtude) => {
  this.loading = true;
  try {
    await agent.UserPractice.done(etude.id);
    runInAction(() => {
      if (this.rootStore.userTodayChapterStore.todayChapter) {
        this.rootStore.userTodayChapterStore.todayChapter!.totalTime += Number(etude.time);
        this.rootStore.userTodayChapterStore.todayChapter!.totalEtudes++;
      } else {
        this.rootStore.userTodayChapterStore.loadTodayChapter();
        toast.info("👍 Começando um novo capítulo!");
      }

      this.loadUserPractice();
      this.loading = false;
    });
    toast.success("👍" + etude.title + " feito com sucesso.");
  } catch (error) {
    runInAction(() => {
      this.loading = false;
    });
    toast.error("👎 Erro ao registrar etude.");
  }
};
```

Fonte: Elaborado pelo autor (2021)

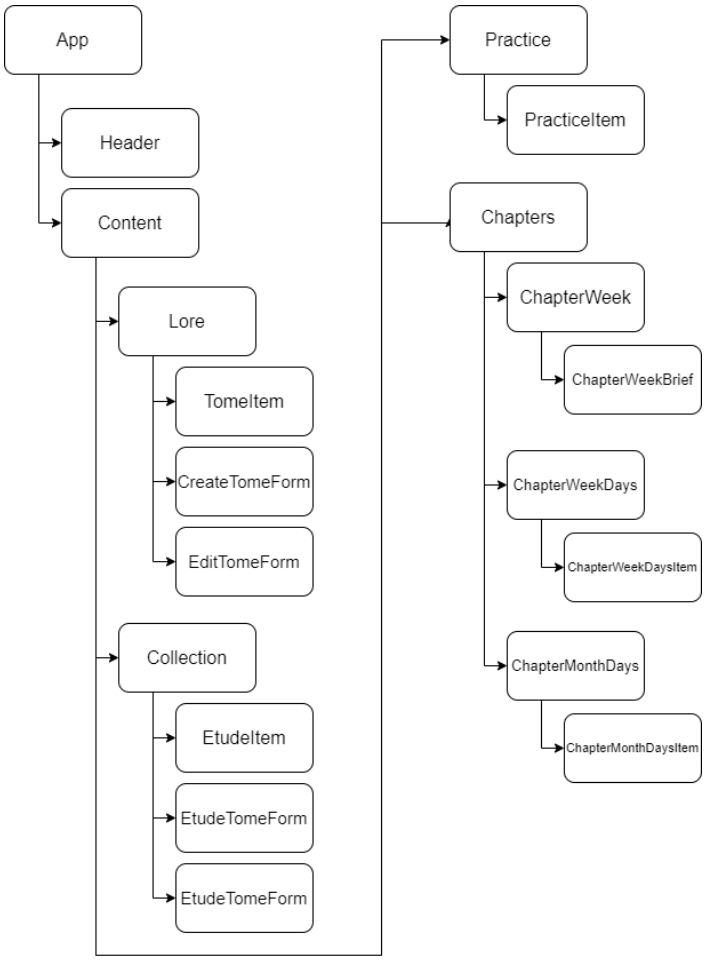
5.3.2.3 Desenvolvimento de cliente React

O elemento principal na etapa de desenvolvimento do cliente é o React, tal ferramenta é uma biblioteca escrita em JavaScript e *open-source* desenvolvido por uma grande comunidade e mantido pela Facebook. O React possibilita o desenvolvimento de um cliente *web* utilizando a linguagem TypeScript dentro do ambiente Node.js aplicando o conceito SPA, acrônimo para *Single-Page Application (*Aplicativo de página única). Este conceito tem como ideia principal o aplicativo *web* ter um comportamento parecido com um aplicativo nativo através de constantes rescritas de fragmentos (chamados de componentes) pelo aplicativo.

Para que se possa implementar o aplicativo SPA utilizando o React, componentes devem seguir uma ordem hierárquica em que muitos destes são criados apenas para encapsular outros componentes. Esta ordem irá definir também como propriedades (Props) são distribuídas entre eles. Observa-se que em fase de desenvolvimento, o conceito SPA aplicado em React facilitou refatoração e divisão de responsabilidade que se aumenta conforme o aplicativo cresce e sua proposta se solidifica. A Figura 32 representa a hierarquia ao qual os componentes do aplicativo desenvolvido seguem, deve-se ser observado que por convenção internacional o código é em inglês, em que *lore* representa coleção de grupos,

tome representa grupo, *collection* é coleção de *etudes*, e *chapters* representa períodos de elementos estatísticos.

Figura 32 – Componentes React



Fonte: Elaborado pelo autor (2021)

Para o desenvolvimento deste aplicativo, criou-se um componente React padrão, ao qual serviu de base para a implementação dos demais. A Figura 33 representa uma destas implementações e foi escolhido como modelo o componente Practice que é responsável por exibir dentro do container do componente Content a lista de todos os Etudes à se praticar através de reuso do componente PracticeItem.

Repara-se que para um componente React funcionar com React hooks, ele deve ser declarado como React.FC (*React Functional Component*). O código segue com a declaração da rootStore do MobX que permitirá acesso a qualquer *store*, que no caso é a que compões as funcionalidades empregadas no componente de exemplo. As ações que tal *store* declara são utilizadas pelo componente react como State Hook, estes estados são observados

pela *store* responsável, armazenando-os e os atualizando por ações. Vale lembrar que todas as *stores* têm acesso entre si através da *rootStore*, sendo assim, é possível alterar estados de observáveis entre componentes. Para que isso ocorra o componente React deve ser declarado como observável, como pode ser viso na última linha do código representado.

Seguindo o código, observa-se o React Effect Hook, representado por *useEffect*, que são ações ao qual deseja que aconteça em momentos definidos, que podem ser quando um componente é montado, atualizado ou desmontado. No caso deste componente temos, uma ação que ocorre quando o componente é montando e ocorre a execução da ação de carregar a lista de estudos do usuário, destaca-se que tal lista é carregada dentro da *store* na variável observável userPractice que é composta por uma lista de etudes.

Figura 33 – Componente React

```
const TrainerPractice: React.FC = () => {
  const rootStore = useContext(RootStoreContext);
  const { userPractice, loadUserPractice, loadingUserPractice } = rootStore.userPracticeStore;

  useEffect(() => {
    loadUserPractice();
  }, [loadUserPractice]);

  return (
    <Tab.Pane>
      <Grid>
        {/* LOADING COMPONENT */}
        {loadingUserPractice && <LoadingComponent content="Loading" />}
        {/* HEADER */}
        <Grid.Column width={16}>
          <Header floated="left" icon="pin" content={"Hora de Tocar!"} />
        </Grid.Column>
        {/* BODY */}
        <Grid.Column width={16}>
          {/* ETUDE LIST */}

          <Card.Group stackable itemsPerRow={4}>
            {userPractice?.etudes.map((etude) => (
              <TrainerPracticeItem key={etude.id} etude={etude} />
            ))}
          </Card.Group>

        </Grid.Column>
      </Grid>
    </Tab.Pane>
  );
};

export default observer(TrainerPractice);
```

Fonte: Elaborado pelo autor (2021)

O React não possui bibliotecas para componente de interface de usuário, sendo possível ser implementando praticamente qualquer solução desejada. Para este trabalho foi escolhida a biblioteca Semantic-UI, de licença MIT e mantida pela Semantic-Org. Na Figura 31 após o *return* o código demonstra a implementação de um componente de interface de

usuário utilizando o Semantic-UI. Repara-se que os componentes de tal biblioteca possuem nomenclaturas que buscam ao máximo demonstrar suas características através de seus nomes. Esta biblioteca possui centenas de componentes e propriedades para serem utilizados e é nativa para responsividade, tornando aplicativos *web* rapidamente implementados para qualquer tamanho de navegador.

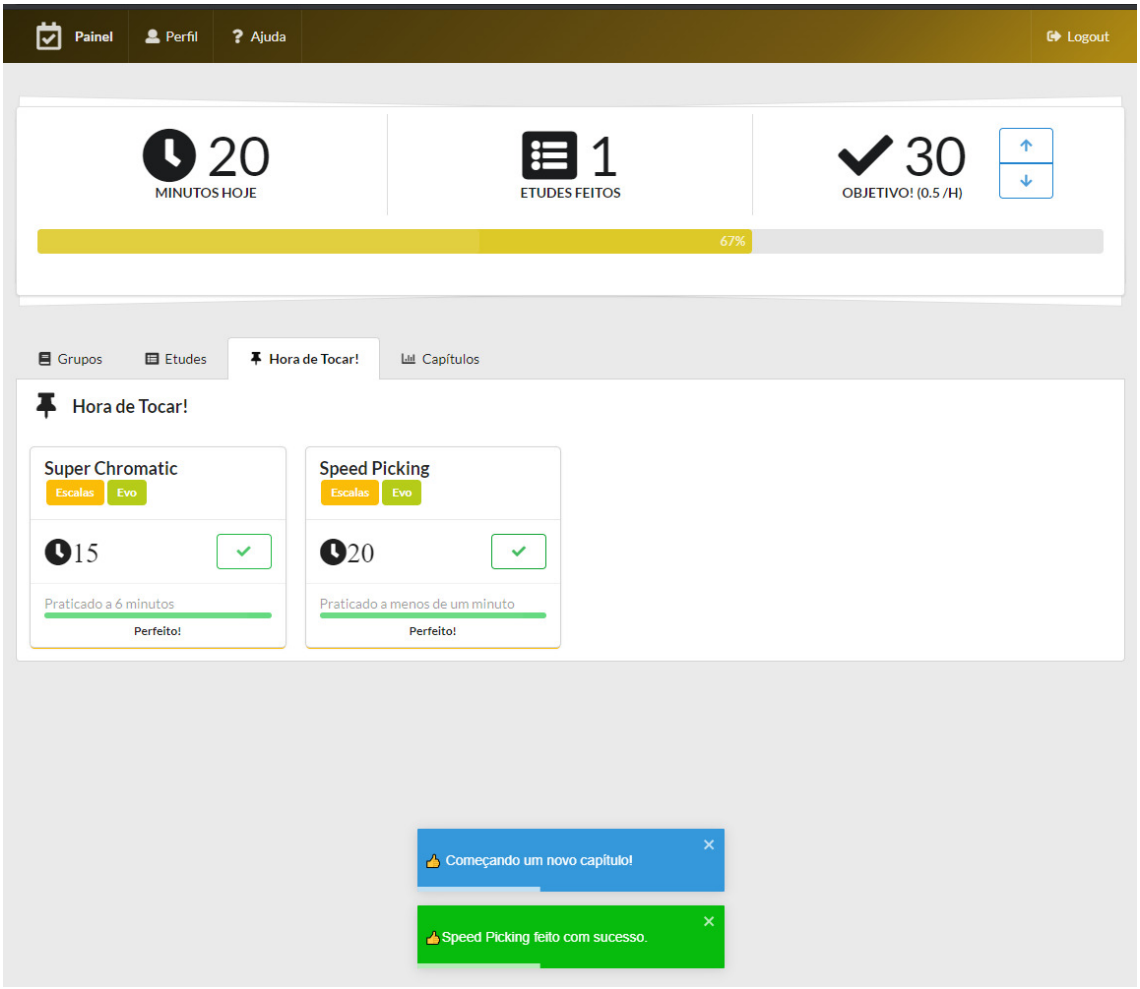
6 RESULTADOS

A etapa de resultados consiste em validar as funcionalidades em relação aos modelos propostos durante a primeira e segunda etapa do método, que são: as entrevistas individuais e a elaboração dos modelos. Esta etapa é dividia em sete partes, s primeira que tem como objetivo demonstrar o resultado do aplicativo de uma forma geral, a segunda que tem a intenção de demonstrar o processo de validação, a terceira busca demonstrar a organização quanto a grupos de estudos (Grupos), a quarta foca nos estudos (Etudes), a quinta foca na ordenação e sugestão de prática (Hora de Tocar!), a sexta parte busca demonstrar o resultado quanto a percepção de dedicação (Capítulos) e por fim o componente responsável pela percepção do dia iminente na prática do guitarrista. Destaca-se que a biblioteca utilizada para desenvolvimento dos elementos de interface de usuários (Semantic-UI) é responsiva, sendo assim tornou-se necessário representá-las em cada uma das subdivisões desta etapa para destacar o resultado de um aplicativo *web* responsivo.

6.1 Painel principal do aplicativo

A Figura 34 ilustra o painel principal do aplicativo no exato momento em que um novo estudo (Etude) é registrado. Acima observa-se a barra de navegação, logo abaixo o cabeçalho fixo e abaixo abas para as quatro funcionalidades principais. Destaca-se as mensagens em formato de *toast* que o usuário recebe para estudo feito e novo capítulo iniciado.

Figura 34 – Painel principal do aplicativo

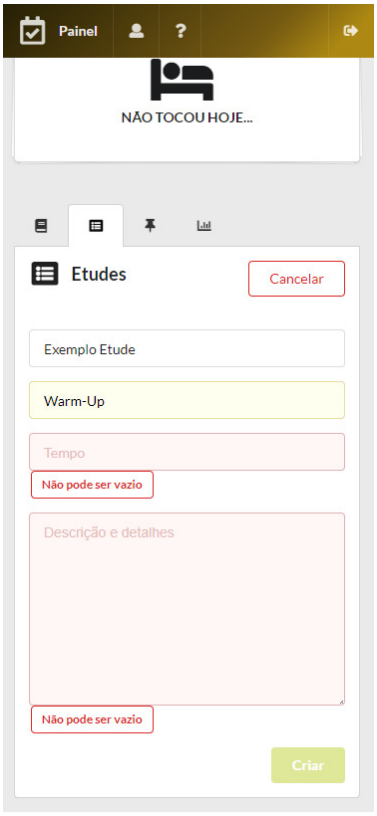


Fonte: Elaborado pelo autor (2021)

6.2 Validação de dados

O aplicativo foi desenvolvido com regras de validação, a Figura 35 demonstra tal validação no cliente junto a um processo de criação de novo estudo (Etude). Ressalta-se que todas as funcionalidades que dependem se inserção de dados do usuário utilizam o mesmo processo e regras de validação ilustrados na figura.

Figura 35 – Criação estudo (Etude) e validação no cliente

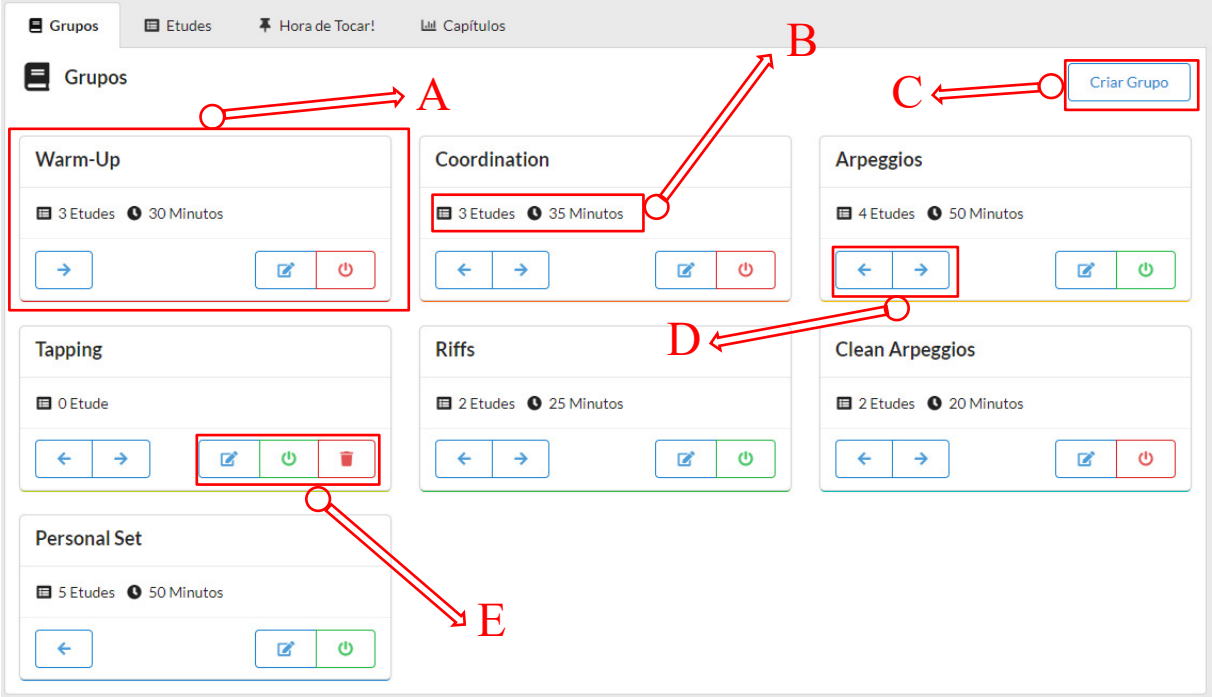


Fonte: Elaborado pelo autor (2021)

6.3 Componente de grupos de estudos

A interface de usuário para grupos de estudos representado pela Figura 36, possui os componentes necessários para o usuário manter seus estudos, definir a sequência para prioridades, visualizar dados importantes sobre a quantidade de estudos (Etudes) ativados e a possibilidade de ativar ou desativar tal grupo.

Figura 36 – Interface para Grupos

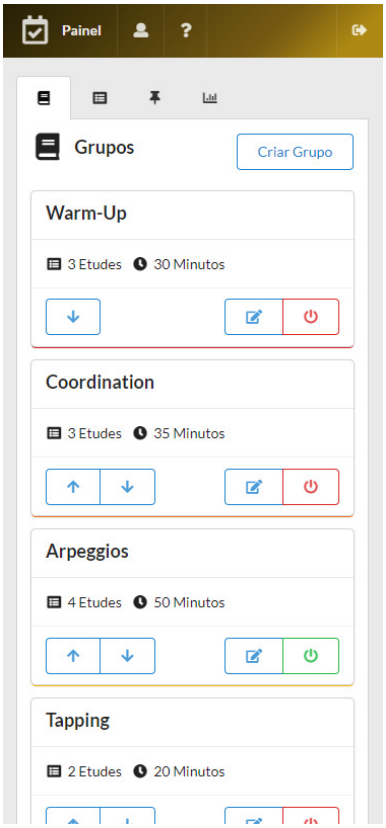


Fonte: Elaborado pelo autor (2021)

A tela é composta por todos os grupos de estudos criados pelo usuário e listados com um único componente através de reuso, este item é representado unitariamente pela marcação A. Além do nome do grupo o usuário tem acesso a dois elementos, a quantidade de estudos (Etudes) ativos e a soma do tempo destes em minutos, que são identificados pela marcação B. Para a criação de um novo grupo, foi implementado um botão de “Criar Grupo” para acesso ao formulário de criação que é representado pela marcação C. Para a ordenação pessoal de grupos, foi adicionado dois botões que apontam para a direção ao qual o grupo será movido e influenciará nas listagens de estudos e ordenação para prática. A marcação E, representa três funcionalidades, o botão em azul para edição e o botão que se alterna entre verde e vermelho para ativação e desativação do grupo, caso o grupo seja desativado, ele não será listado nos componentes “Etudes” e “Hora de Toca!”, e por fim, o botão de exclusão que somente será visível se o grupo não tiver nenhum Etude.

A Figura 37, demonstra a tela de estudos (Etudes) responsiva ao navegador *mobile.* Destaca-se o reposicionamento das setas, dando ao usuário a precisa posição ao qual o grupo será movido. Sendo esquerda e direta para navegadores grandes e para baixo e para cima em navegadores *mobile*.

Figura 37 – Tela de grupos em navegador *mobile*

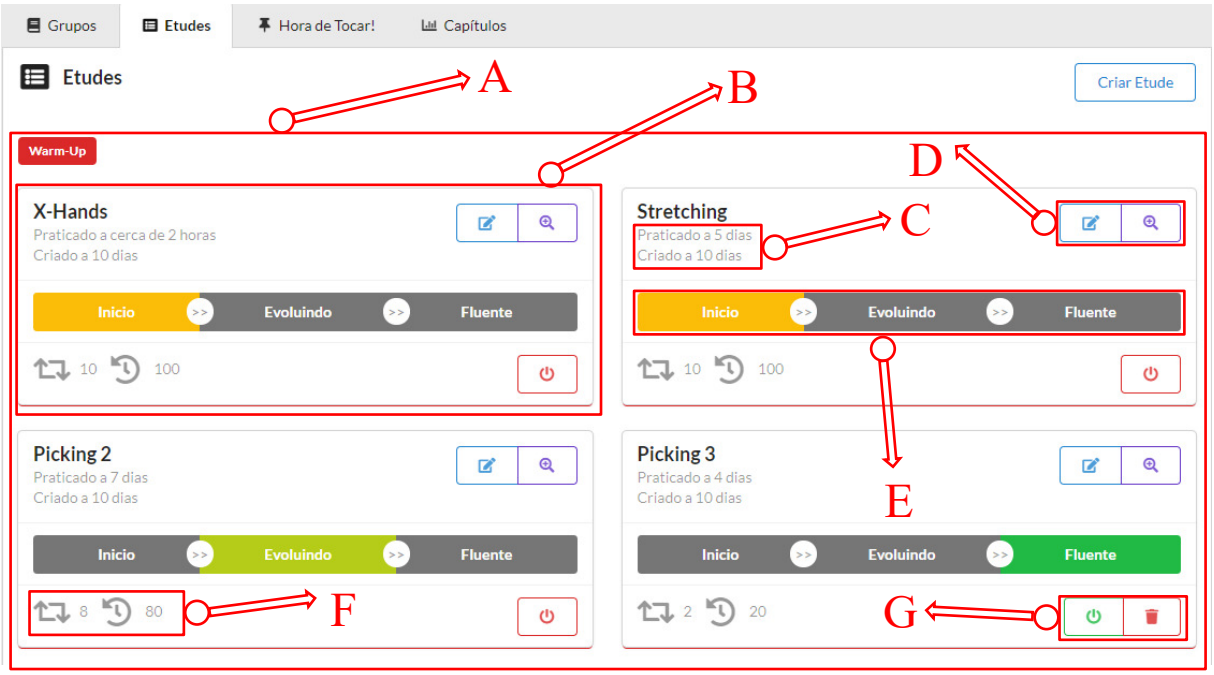


Fonte: Elaborado pelo autor (2021)

6.4 Componente de estudos (Etudes)

A interface de usuário para estudos Figura 38, foi desenvolvida nos mesmos princípios da anterior, em que temos reuso de um componente para fazer a listagem de um estudo, porém, os estudos estão dentro de um grupo.

Figura 38 – Interface para Estudos



Fonte: Elaborado pelo autor (2021)

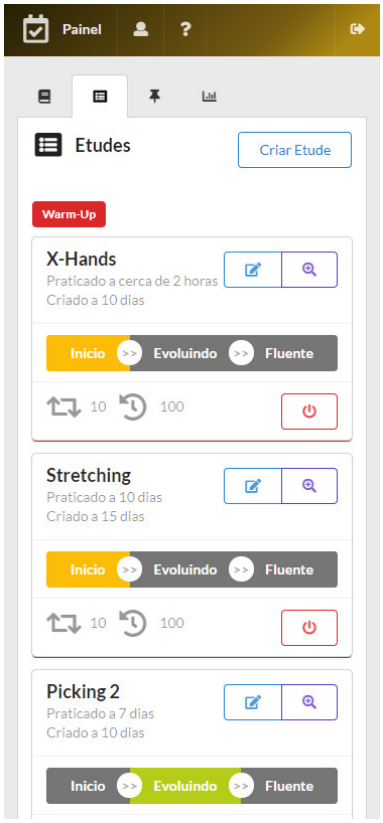
A tela é composta por todos os estudos (chamados de Etudes na aplicação final), que o usuário possui. Diferente da tela anterior, todos os estudos dentro de um grupo estarão distribuídos dento de um componente utilizado como container e tem seu rótulo destacado acima dos grupos, representado pela marcação A, por sua vez cada elemento dentro desse container é representado pela marcação B. Esta tela possui o desafio de ter que exibir diversas informações e ao mesmo tempo deve ser compressível, como visto em Barbosa (2010), os conceitos da teoria de Gestalt pode deixar esta tela com uma boa interatividade, se aproximarmos elementos comuns e for mantida uma boa simetria entre os elementos.

A última data de prática e quando o estudo foi criado, são demonstrados com a diferença para o dia atual, que não possuí limites e começa em segundos, caso nunca tenha sido praticado a mensagem “Nunca praticado” irá substituir tal diferença, demonstrado na marcação C. Seguindo mesmo padrão de ícones e funcionalidade da tela anterior, temos na marcação D, botão para edição e visualização de detalhes. O foco principal do componente de estudo é permitir ao usuário determinar o estado em que o estudo se encontra, se está em estado de entendimento, evolução ou domínio. Para isso os nomes foram alterados para Início, Evoluindo e Fluente, por questões de espaçamento e limitações da biblioteca de componentes gráficos, também foi escolhido três cores que vão do amarelo ao verde para representá-los. Para que o usuário tenha controle sobre quantas vezes praticou um estudo e a soma total de

tempo foi adicionado um componente, marcação F, em que possui a esquerda o total de vezes praticado e a direita o total de tempo, destaca-se que o total de tempo não é uma multiplicação de quantidade de vezes pelo valor de tempo atual do estudo, é uma soma feita a cada vez que é praticado, pois o usuário pode decidir em um momento alterar quantos minutos tal estudo é feito. Por fim, a marcação G demonstra a funcionalidade de ativar e desativar, que fará com que tal estudo não apareça na listagem de prática e a esquerda o botão de exclusão que somente irá aparecer caso o estudo esteja desativado.

A Figura 39, demonstra a tela de estudos (Etudes) responsiva ao navegador *mobile.*

Figura 39 – Tela de estudos (Etudes) em navegador *mobile*



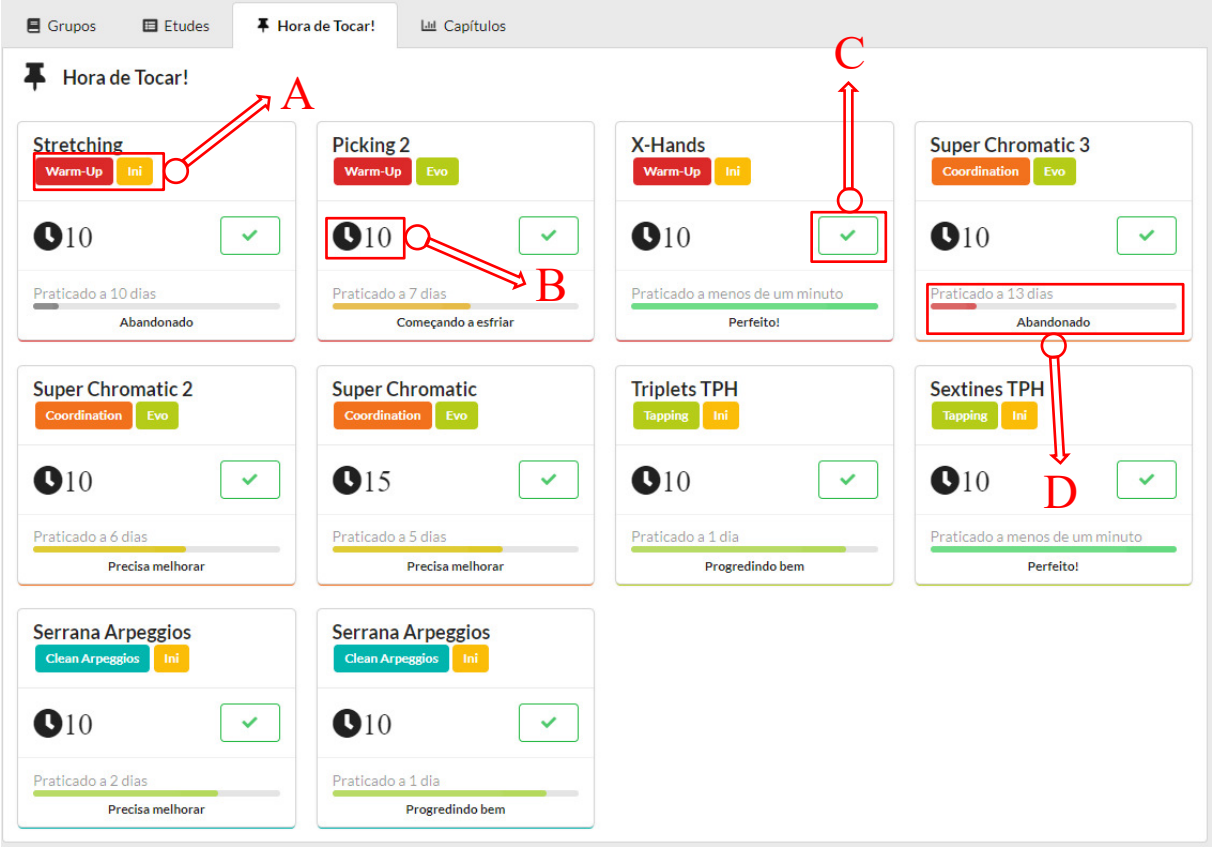
Fonte: Elaborado pelo autor (2021)

6.5 Componente para ordenação de prática

A tela de prática busca atingir o objetivo principal deste trabalho, é composta da lista sugerida de estudos seguindo o modelo de ordenação proposto. Em que grupos são mandatórios definidos pelo usuário e dentro de cada grupo os estudos são ordenados por

prioridade ascendente (menor valor tem mais prioridade) e feito desempate pelo seu estado. A figura 40, demonstra a janela em que se buscou colocar informações pertinentes ao momento da prática.

Figura 40 – Interface para prática



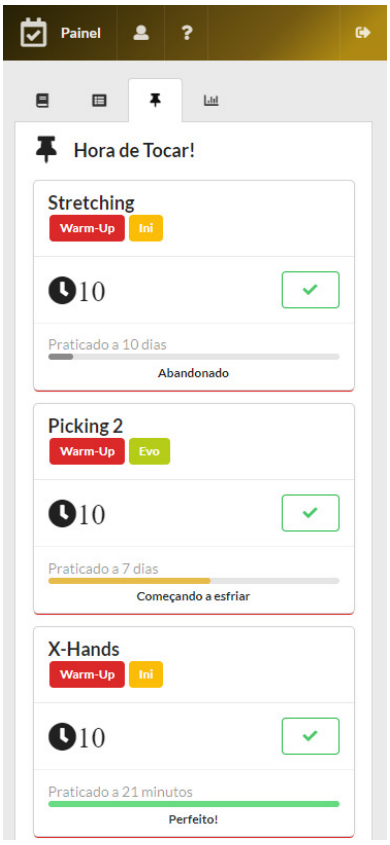
Fonte: Elaborado pelo autor (2021)

Assim como as anteriores, esta tela é composta por um componente de container que lista os componentes de estudo desenvolvidos para prática. Repara-se que a ordenação é respeitada e os grupos estão na sequência definida pelo usuário e são visualmente identificáveis por um rótulo e sobretudo a cor que herda da sequência de grupos, nota-se também, que o rótulo de grupo é seguido pelo rótulo de estado em que o estudo se encontra, ambos são demonstrados pela marcação A. Para que o usuário tenha uma rápida assimilação de quanto tempo aquele estudo possui, este valor foi colocado em evidência, marcação B. Quando o usuário termina sua prática em um determinado estudo, ele deve clicar no botão representado pela marcação C, que fará todo a atualização do estudo e da lista se baseando no novo estado em que todos os estudos se encontram. E por fim, cada componente possui identificadores de aproveitamento, representados pela marcação D, em que primeiro momento

observa-se a diferença de tempo para o dia atual em que tal estudo foi feito pela última vez, o valor de prioridade representado com uma barra que se altera entre zero a cem, seguindo um padrão definido de cores (cinza, vermelho, laranja, amarelo, e três tons de verdes), foi implementado mensagens amigáveis a linguagem humana que representam o nível em que se encontra a dedicação individual de cada estudo, sendo: para valores de prioridade 100 recebe "Perfeito!", maior ou igual a 80 recebe "Progredindo bem", maior ou igual a 60 recebe "Precisa melhorar", maior ou igual a 40 recebe "Começando a esfriar", maior ou igual a 20 recebe "Esfriando..." e "Abandonado" para valores menores do que 20.

A Figura 41, demonstra a tela de práticas (Hora de Tocar!) responsiva ao navegador *mobile*.

Figura 41 – Tela de prática em navegador *mobile*

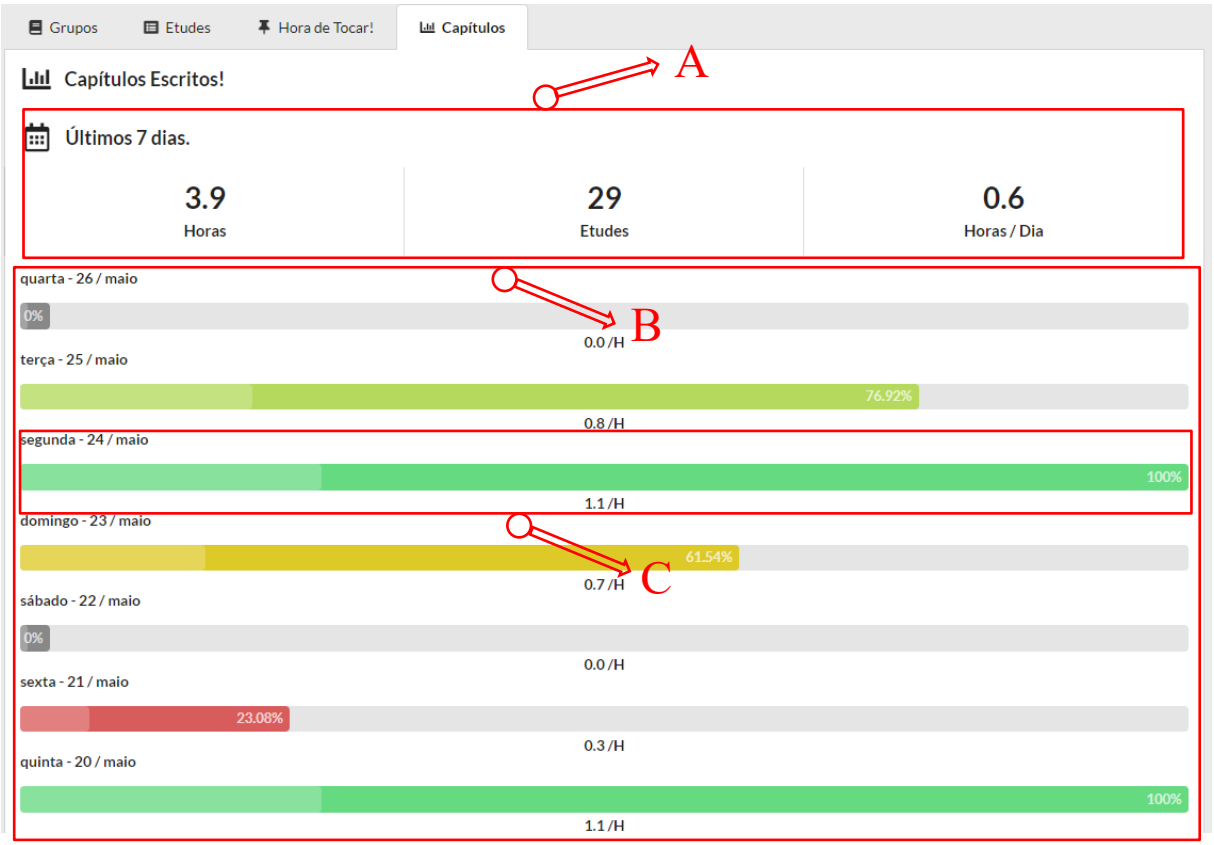


Fonte: Elaborado pelo autor (2021)

6.6 Componentes de aproveitamento

A tela que demonstra dados estatísticos sobre o aproveitamento de estudos do guitarrista, possui na parte superior o componente responsável por um resumo dos últimos sete dias e na parte inferior podendo receber componentes com diversos intervalor de tempos, foi recolhido por demonstrar no trabalho os últimos sete dias. A Figura 42 demonstra a janela em que tais componentes foram implementados.

Figura 42 – Interface para aproveitamento



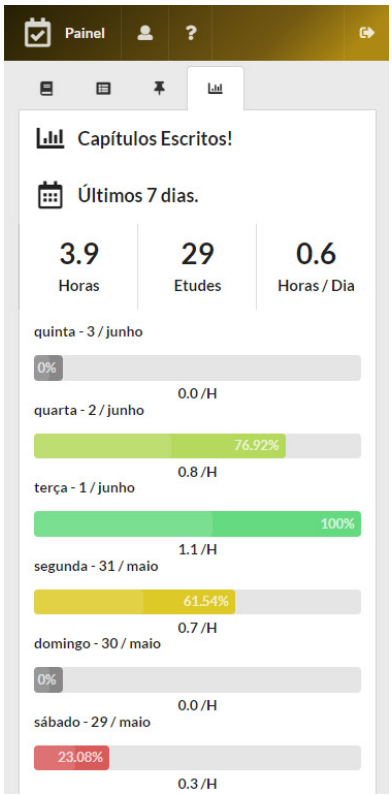
Fonte: Elaborado pelo autor (2021)

Assim como as anteriores, esta tela é composta por um componente de container, que neste caso irá comportar diferentes componentes estatísticos. O primeiro componente, marcação A, possui três valores que são computados com base nos últimos sete dias de prática, o primeiro valor é efetuada a soma de quantidade de horas, o segundo ocorre uma soma da quantidade de estudos e o terceiro faz uma média aritmética arredondada a uma cada decimal de quantidade de horas por dia. O próximo componente, marcação B, que integra o container, é composto por um componente que é reutilizado para cada um dos sete dias,

representado pela marcação C. Este componente possui a data, uma barra que representa o aproveitamento em relação ao melhor dia e quantidade de horas arredondada em uma casa decima de prática ao dia.

A Figura 43 demonstra o componente de Capítulos responsivo ao navegador *mobile*.

Figura 43 – Tela Capítulos em navegador *mobile*



Fonte: Elaborado pelo autor (2021)

6.7 Componente de aproveitamento diário

Se tornou necessária para fim de boa usabilidade do aplicativo, em que o usuário não precise ficar se deslocando entre telas, a implementação de um cabeçalho que demonstre o aproveitamento do dia atual e seja permitido ao usuário criar e acompanhar uma meta. A figura 44 e figura 45 demonstram o cabeçalho que sempre será exibido acima das telas anteriores, independente de qual esteja sendo visualizada.

Figura 44 – Interface para aproveitamento do dia (Vazia)

Fonte: Elaborado pelo autor (2021)

Figura 45 – Interface para aproveitamento do dia (Iniciada)

Fonte: Elaborado pelo autor (2021)

Este é um componente que é iniciado pelo primeiro estudo feito na tela de prática. Sendo composto por quatro componentes, o primeiro é uma soma de minutos feitos no dia, o segundo uma somatória de quantidade de estudos feitos no dia e o terceiro é a meta em que o usuário configura através dos botões (adiciona e subtraí dez minutos), representados pela marcação A, observa-se que este elemento também demonstra a quantidade de horas abaixo dos minutos para facilitar o ajuste de objetivo, e por último uma barra de porcentagem em relação ao tempo praticado do dia com objetivo. Vale ressaltar que a barra altera de cor conforme a porcentagem (cinza, vermelho, laranja, amarelo, e três tons de verdes).

7 CONCLUSÃO

Como conclusão, reflete-se como os objetivos foram enfrentados durante o desenvolvimento deste trabalho, apontando suas dificuldades e sintetizando seus resultados obtidos.

Para ser realizado o objetivo específico a), que de forma condensada consiste em entrevistar um grupo de guitarristas para entender seu universo e necessidades, adotou-se entrevistas como processo de indagação sobre o universo dos guitarristas. Verificou-se que a escolha por um grupo de guitarristas com experiencia musical em torno de 20 anos contribui positivamente no processo de entendimento, tal parte que se torna importante ao desenvolvimento de qualquer software computacional.

O objetivo específico b), que de forma condensada consiste em entender tais dados levantados no objetivo específico anterior e determinar os estados em que um estudo assume no processo de desenvolvimento e manutenção técnica e como estes são enfrentados por tais músicos. Compreende-se que de maneira pessoal, os músicos dividiam seus estudos em grupos das mais variadas formas, e tratavam de modo diferente cada estudo ao seu nível de domínio. Destaca-se que os músicos praticavam de maneiras semelhantes aos livros de desenvolvimento técnico que foram utilizados na parte introdutória e teórica deste trabalho. Percebe-se que havia uma certa dificuldade em rastrear tempo total de prática e intervalo de prática de cada estudo, reforçando a necessidade de um aplicativo para gerenciamento de estudos.

Sabendo-se que estudar técnica em um instrumento musical é um processo que envolve diversas habilidades, e no escopo de desenvolvimento técnico o aprendizado e retenção motora se torna um elemento importante, o objetivo específico c). que de forma condensada tem o objetivo de investigar a existência de um modelo de aprendizado e entender o processo de aprendizado e retenção motora em um instrumento musical. Explorou-se no universo de aprendizado de instrumentos musicais em uma percepção científica, na esperança de encontrar fórmulas que pudessem ajudar tais músicos a ter um melhor aproveitamento de estudos. Observou-se que são muitas variáveis empregadas para o processo de retenção e aprendizado motor e não foi encontrado respostas ao nível esperado, fazendo com que tal teoria passe a ser muito mais destinada a reforçar a ideia de disciplina na prática e entender como é o aprendizado e retenção motora humana. Porém, durante este trabalho, observou-se uma semelhança entre algumas teorias básicas em concordância com o desenvolvimento pessoal de como os músicos sentem sua degradação técnica em relação ao nível de domínio

de cada estudo. Isso abre a possibilidade de sugerir um modelo para ordenação de listas de estudos.

Com a finalização dos objetivos específicos anteriores, se torna possível realizar o objetivo específico d). Observou-se que para sugerir uma sistematização de práticas em um instrumento musical, deve ser sempre levado como elemento principal a liberdade que o músico tem de escolher e ao mesmo tempo demonstrar informações suficiente que o faça entender quais estudos estão precisando de atenção e poder definir quais estudos ou grupos não deseja praticar em determinado momento, visto que, muitos músicos não possuem uma rotina regular de tempo.

Com todos os objetivos específicos concluídos, se torna possível confirmar o objetivo principal deste trabalho. Para tal êxito, foi elaborado um sistema *web* utilizando como elemento principais .Net Core e React sendo o resultado demonstrado na sessão Resultados deste trabalho.

Durante o desenvolvimento do trabalho, observou-se diversas possiblidades para trabalhos futuros. Para um músico que acumula diversas fontes de estudos, um cliente *mobile* que se permite tirar fotos dos estudos, fazendo uma coleção de imagens para cada um. Indo além, este sistema poderia fazer a leitura da imagem e gerar um arquivo MIDI (acrônimo de *Musical Instrument Digital Interface* - Interface Digital de Instrumentos Musicais). Com esse arquivo MIDI, se torna possível a utilização de praticamente qualquer *software* de leituras de partituras, este *software* poderia ser implementado no cliente *web* e no cliente *mobile*, assim como algumas plataformas de distribuição de partituras já oferecem, com isso o guitarrista teria uma plataforma completa para estudos. Com esses elementos em funcionamento e tendo cada vez mais um mundo conectado, uma rede social de compartilhamento de experiências poderia ser implementada e poderia englobar escolas de músicas, grupos de desenvolvimento técnico, bandas etc.

REFERÊNCIAS

ALVES, S. M. et al. **Aprendizagem e Controle Motor**. 1. ed. Sobral: INTA, 2016.

ANDERSON, R. **Introduction to Identity on ASP.NET Core**. Disponível em: <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity>. Acesso em: 26 jun. 2021.

AVRAM, A.; MARINESCU, F. **Domain Driven Design Quickly**. United States of America: C4Media Inc, 2007.

BARBOSA, S. D. J. **Interação Humano-Computador**. 1ª edição ed. Rio de Janeiro: Elsevier, 2010.

CAVINI, M. P. **História da Música Ocidental**. São Carlos: EdUFSCar, 2012. v. 2

DILLARD, K. **Intelli-Shred**. 1. ed. Van Nuys: Alfred Publishing CO. INC., 2007.

DILLARD, K. **Arpeggio Madness**. 1. ed. Van Nuys: Alfred Publishing CO. INC., 2009.

EVANS, E. **Domain-Driven Design: Tackling Complexity in the Heart of Software**. Illustrated edição ed. Boston: Addison-Wesley Professional, 2003.

FIELDING, R. T. Architectural Styles and the Design of Network-based Software Architectures. p. 180, 2000.

FIELDING, R. T. et al. **Reflections on the REST architectural style and "principled design of the modern web architecture" (impact paper award)**. Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. **Anais**...: ESEC/FSE 2017.New York, NY, USA: Association for Computing Machinery, 21 ago. 2017. Disponível em: <https://doi.org/10.1145/3106237.3121282>. Acesso em: 26 maio. 2021

FLÉCHET, A. Por uma história transnacional dos festivais de música popular. Música, contracultura e transferências culturais nas décadas de 1960 e 1970. **Patrimônio e Memória**, v. 7, n. 1, p. 257–271, 4 ago. 2007.

GAMMA, E. et al. **Design Patterns: Elements of Reusable Object-Oriented Software**. 1st edition ed. Reading, Mass: Addison-Wesley Professional, 1994.

JONES, M. **JSON Web Algorithms (JWA)**. [s.l.] RFC Editor, maio 2015. Disponível em: <https://www.rfc-editor.org/info/rfc7518>. Acesso em: 25 maio. 2021.

JONES, M.; HILDEBRAND, J. **JSON Web Encryption (JWE)**. [s.l.] RFC Editor, maio 2015. Disponível em: <https://www.rfc-editor.org/info/rfc7516>. Acesso em: 25 maio. 2021.

KRAKAUER, J. et al. Motor Learning. In: **Comprehensive Physiology**. [s.l: s.n.]. v. 9p. 613–663.

KRAMPE, R. TH.; ERICSSON, K. A. Maintaining excellence: Deliberate practice and elite performance in young and older pianists. **Journal of Experimental Psychology: General**, v. 125, n. 4, p. 331–359, 1996.

LORENZ, M. et al. Object-Relational Mapping Revisited - A Quantitative Study on the Impact of Database Technology on O/R Mapping Strategies. 2017.

MACRITCHIE, J.; MILNE, A. J. Exploring the Effects of Pitch Layout on Learning a New Musical Instrument. **Applied Sciences**, v. 7, n. 12, p. 1218, 1 nov. 2017.

MARTIN, R. C. Design Principles and Design Patterns. p. 34, 2000.

MARTIN, R. C. **Clean Code: A Handbook of Agile Software Craftsmanship**. 1st edition ed. Upper Saddle River, NJ: Pearson, 2008.

MARTIN, R. C. **Clean architecture: a craftsman's guide to software structure and design**. Boston: Prentice Hall, 2017.

MASAKI, H.; SOMMER, W. Cognitive neuroscience of motor learning and motor control. **The Journal of Physical Fitness and Sports Medicine**, v. 1, p. 369–380, 25 set. 2012.

MILARDI, A. **The Electric Guitar: A History of an American Icon**. Baltimore: ohns Hopkins University Press, 2004. v. 6

NICKEL, J. **Mastering Identity and Access Management with Microsoft Azure**. Birmingham: Packt Publishing Ltd, 2016.

PETRUCCI, J. **Wild Stringdom**. 1. ed. Van Nuys: Alfred Publishing CO. INC., 2000.

PRIBYL, S. F. & B. **Oracle PL/SQL Programming, 2nd Edition**. Text.Monograph. Disponível em: <https://docstore.mik.ua/orelly/oracle/prog2/ch18_05.htm>. Acesso em: 23 maio. 2021.

PRICE, M. J. **C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development: Build applications with C#, .NET Core, Entity Framework Core, ASP.NET Core, and ML.NET using Visual Studio Code, 4th Edition**. Illustrated edition ed. Birmingham: Packt Publishing, 2019.

PRICE, M. J. **C# 9 and .NET 5 – Modern Cross-Platform Development: Build intelligent apps, websites, and services with Blazor, ASP.NET Core, and Entity Framework Core using Visual Studio Code, 5th Edition**. 5th ed. edition ed. Birmingham: Packt Publishing, 2020.

SCHMIDT, R. A.; LEE, T. D. **Motor control and learning: A behavioral emphasis, 5th ed**. Champaign, IL, US: Human Kinetics, 2011.

SILVA, L. L. C. F. DA. ASPECTOS DA VIDA COTIDIANA NA VIDA DO TRABALHADOR: O ESTRANHAMENTO DO TRABALHO E DA CIDADE. **Revista de Ciências do Estado**, v. 2, n. 1, 1 jul. 2017.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson, 2011.

STETINA, T. **Speed Mechanics for Lead Guitar**. Milwaukee: Hal Leonard Corporation, 1990.

WAGNER, B. **Language-Integrated Query (LINQ) (C#)**. Disponível em: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/>. Acesso em: 26 jun. 2021.

WALSER, R. Eruptions: Heavy Metal Appropriations of Classical Virtuosity. **Popular Music**, v. 11, n. 3, p. 263–308, 1992.

WYATT, K.; SCHROEDER, C. **Harmony and Theory: A Comprehensive Source for All Musicians**. 1. ed. Milwaukee: Hal Leonard Corporation, 1998.

ZATORRE, R. J.; CHEN, J. L.; PENHUNE, V. B. When the brain plays music: auditory-motor interactions in music perception and production. **Nature Reviews. Neuroscience**, v. 8, n. 7, p. 547–558, jul. 2007.

APÊNDICE A – Ferramentas**,** bibliotecas e *frameworks*

Este apêndice tem como objetivo, identificar e referenciar as ferramentas utilizadas durante o desenvolvimento do aplicativo deste trabalho. Ela é dividida em ferramentas utilizadas no desenvolvimento do servidor, posteriormente no desenvolvimento do aplicativo cliente e segue com as ferramentas de implementação de código e testes.

Abaixo segue os elementos utilizados no desenvolvimento do servidor:

- Microsoft .Net Core
  - Versão: 3.1.1
  - Link documentação: https://docs.microsoft.com/en-us/dotnet/fundamentals/
  - GitHub: https://github.com/dotnet/core
  - Licença: MIT
- Microsoft Entity Framework Core
  - Versão: 3.1.1
  - Link documentação: https://docs.microsoft.com/en-us/ef/
  - GitHub: https://github.com/dotnet/efcore
  - Licença: Apache License 2.0
- Fluent Validation
  - Versão: 8.6.1
  - Link documentação: https://fluentvalidation.net/
  - GitHub: https://github.com/FluentValidation/FluentValidation
  - Licença: Apache License 2.0
- MediatR
  - Versão: 7.0.0
  - Link documentação: https://github.com/jbogard/MediatR/wiki
  - GitHub: https://github.com/jbogard/MediatR
  - Licença: Apache License 2.0

Abaixo segue os elementos utilizados no desenvolvimento do cliente:

- Node.js
  - Versão: 12.15.0
  - Link documentação: https://nodejs.org/en/docs/

    - GitHub: https://github.com/nodejs/node
    - Licença: MIT
- Facebook React
    - Versão: 16.11.0
    - Link documentação: https://reactjs.org/docs/getting-started.html
    - GitHub: https://github.com/facebook/react
    - Licença: MIT
- TypeScript
    - Versão: 3.7.2
    - Link documentação: https://www.typescriptlang.org/docs/
    - GitHub: https://github.com/microsoft/TypeScript
    - Licença: MIT
- Axios
    - Versão: 0.19.0
    - Link documentação: https://axios-http.com/docs/intro
    - GitHub: https://github.com/axios/axios
    - Licença: MIT
- MobX
    - Versão: 5.15.2
    - Link documentação: https://mobx.js.org/README.html
    - GitHub: https://github.com/mobxjs/mobx
    - Licença: MIT
- React-Toastfy
    - Versão: 6.0.8
    - Link documentação: https://fkhadra.github.io/react-toastify/introduction
    - GitHub: https://github.com/fkhadra/react-toastify
    - Licença: MIT
- Semantic-UI React
    - Versão: 0.88.1
    - Link documentação: https://react.semantic-ui.com/
    - GitHub: https://github.com/Semantic-Org/Semantic-UI-React
    - Licença: MIT

- Date-FNS
    - Versão: 2.0.0
    - Link documentação: https://date-fns.org/
    - GitHub: https://github.com/date-fns/date-fns
    - Licença: MIT

Abaixo segue ferramentas utilizadas para desenvolvimento:

- Microsoft Visual Studio Code
    - Link Principal: https://code.visualstudio.com/
    - GitHub: https://github.com/microsoft/vscode
    - Licença: MIT
- Postman-Inc, Postman
    - Link Principal: https://www.postman.com/
    - Link documentação: https://www.postman.com/api-evangelist/workspace/wikipedia/overview
    - Licença: Paga em diversos pacotes, grátis com limitações
- DB Browser for SQLite
    - Link Principal: https://sqlitebrowser.org/
    - Link documentação: https://github.com/sqlitebrowser/sqlitebrowser/wiki
    - GitHub: https://github.com/sqlitebrowser/sqlitebrowser
    - Licença: GNU

APÊNDICE B – Questionário utilizado nas entrevistas

O questionário foi aplicado durante o método de desenvolvimento e como meio de realização do objetivo específico a). O questionário é composto por perguntas com respostas abertas e fechadas.

1. Qual sua idade?

   Repostas aberta.

2. Música é sua principal atividade?
   a. Sim.
   b. Não.

3. Em algum momento já chegou a dar aula de música?
   a. Sim.
   b. Não.

4. Quanto tempo (em anos) possui de contato com música? vale outros instrumentos.

   Resposta aberta.

5. Já estudou em conservatório ou escolas de músicas renomadas como por exemplo IG&T?
   a. Sim.
   b. Não.

6. Possui formação superior em música? Vale qualquer área da música.
   a. Sim.
   b. Não.

7. Em uma semana ideal, quantas horas aproximadamente costuma dedicar exclusivamente ao desenvolvimento e manutenção técnica? (Incluir manutenção de repertório e domínio de novas músicas e excluir tempo de composição, estudos teóricos e pesquisas).

   Resposta aberta.

8. Divide seu estudo em grupos? Processo de agregar estudos com características parecidas, por exemplo: “Exercícios de Resistência”, “Exercícios de Tapping”, “Exercícios de Escalas”, “Riffs em Semicolcheia em Mi Frígio”, “Meus licks de Pentatônica Am”, “Escalas para Shred”, “Repertório de Solos”, “Repertório da Banda x” etc.
    a. Sim.
    b. Não.
    c. Não, mas sinto que deveria.

9. Cite duas das fontes mais importantes para desenvolver seus próprios estudos:
    a. Livros.
    b. Videoaulas.
    c. Revistas.
    d. Fragmentos de músicas.
    e. Outros.

10. Considera desmotivante ficar montando listas de estudos, repertórios?
    a. Sim.
    b. Não.

11. Qual seu processo de aprendizado para um novo estudo?
    Resposta aberta

12. Em quantos estados de aprendizado considera que cada estudo possuí? Estado deve ser considerado nível de domínio, como por exemplo: aprendendo, dominado...
    Reposta aberta.

13. Já teve a experiência de “desaprender” um estudo? (Exemplo: ficou um tempo sem praticá-lo e sentiu a degradação da precisão, perca de segurança e não conseguia mais fazer no mesmo andamento).
    a. Sim.
    b. Não.

14. Se utilizarmos como base a questão anterior, você sente que este processo de “desaprender” é diferente para estados de aprendizado diferentes?
    a. Sim.
    b. Não.

15. Se considerarmos três estados, aprendendo, evoluindo e dominado para cada estudo. Em quantos dias sem praticar cada um deles você acredita que começa a “desaprendê-lo”?

    Resposta aberta.

16. Consegue dizer quantas horas por dia e semana se dedica para desenvolvimento e manutenção técnica?

    Resposta aberta.

17. Sobre a questão anterior, faz anotações?
    a. Sim.
    b. Não.

18. Sobre as duas questões anteriores, consegue dizer qual foi a última vez que praticou cada um os estudos em seu repertório?
    a. Sim.
    b. Não.

Documento Digitalizado Restrito

Versão final da monografia em PDF

Assunto: Versão final da monografia em PDF
Assinado por: Andreiwid Correa
Tipo do Documento: Outro
Situação: Finalizado
Nível de Acesso: Restrito
Tipo do Conferência: Documento Original

Documento assinado eletronicamente por:

- **Andreiwid Sheffer Correa**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 12/07/2021 12:39:09.

Este documento foi armazenado no SUAP em 12/07/2021. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifsp.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 718577
Código de Autenticação: 2689d3feab