UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

EASYSCORE: UM PROTÓTIPO PARA FACILITAR A LEITURA DE PARTITURAS EM DISPOSITIVOS MÓVEIS

GABRIELA CRISTINA PALUDO

BLUMENAU
2014

2014/2-10

GABRIELA CRISTINA PALUDO

EASYSCORE: UM PROTÓTIPO PARA FACILITAR A LEITURA DE PARTITURAS EM DISPOSITIVOS MÓVEIS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Aurélio Faustino Hoppe, Mestre - Orientador

BLUMENAU
2014

2014/2-10

EASYSCORE: UM PROTÓTIPO PARA FACILITAR A LEITURA DE PARTITURAS EM DISPOSITIVOS MÓVEIS

Por

GABRIELA CRISTINA PALUDO

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

	_____
Presidente:	Prof. Aurélio Faustino Hoppe, Mestre – Orientador, FURB
	_____
Membro:	Prof. Marcel Hugo, Mestre – FURB
	_____
Membro:	Prof. Dalton Solano dos Reis, Mestre – FURB

Blumenau, 11 de dezembro de 2014

Dedico este trabalho à minha família, que apoia minhas decisões pacientemente, incentivando nas horas mais necessárias, quando algumas coisas parecem inacessíveis.

AGRADECIMENTOS

A Deus, por todas as oportunidades.

À minha família, pela educação que me deram, pela assistência, apoio e incentivo em todos os momentos.

Aos amigos, pela sua presença, pelo amparo, pelas cobranças, dicas e conselhos.

Aos colegas, da faculdade e do trabalho, que se mostraram dispostos a auxiliar, compartilhando conhecimentos, experiências e informações de grande ajuda.

Ao professor Marcel Hugo, pela ideia de trabalhar com música e dispositivos móveis.

Ao meu orientador, Aurélio Hoppe por todo o acompanhamento do projeto, pelas assistências, pelo empenho e pela paciência necessária para a conclusão do projeto, com todas as dificuldades.

Nós nunca falhamos quando tentamos fazer nosso trabalho, nós sempre falhamos quando deixamos de fazê-lo.

Robert Baden-Powell

RESUMO

Este trabalho apresenta um protótipo de aplicativo que visa facilitar a leitura de partituras e o acompanhamento de partituras musicais em grupo, em dispositivos móveis com sistema operacional Android. O protótipo lê um arquivo `MusicXML` e através da API do `VexFlow`, escrita em JavaScript, apresenta visualmente a partitura na tela do dispositivo. Foi utilizado o *framework* Cordova, que possibilitou o uso de HTML5, Javascript e CSS na aplicação sem a necessidade de utilização de linguagens específicas do sistema operacional. Para a comunicação e sincronismo foi utilizado *WebSocket*, que guarda as informações das músicas e os arquivos `MusicXML` em um servidor e calcula o horário de início de execução. Os resultados obtidos a partir dos experimentos e dos testes de usabilidade demonstram que o trabalho alcançou o objetivo proposto. O protótipo facilita a leitura e o acompanhamento em conjunto de partituras em dispositivos móveis, podendo servir como inspiração para um produto que hoje está em falta.

Palavras-chave: Dispositivos móveis. Cordova. WebSocket. MusicXML. Partitura.

ABSTRACT

This work presents a prototype of an app which aims to facilitate the view of music scores and its accompaniment in a group, in mobile devices with Android operating system. The prototype reads a `MusicXML` file and through the `VexFlow` API, written in JavaScript, presents visually the music score on the device's screen. The Cordova framework was used, and it allowed the use of HTML5, Javascript and CSS in the application with no need of usage of the operating system's specific languages. For communication and synchronism WebSocket was used. It saves the music information and the `MusicXML` files in a server and calculates the time to start the performance. The results obtained through the experiments and the usability testing demonstrate that the work has reached its purpose. The prototype facilitates the read and the accompaniment in group of music scores in mobile devices, being able to serve as a inspiration to a product that is missing today.

Key-words: Mobile devices. Cordova. WebSocket. MusicXML. Music score.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

ADT – *Android Developer Tools*

API – *Application Programming Interface*

CSS – *Cascading Style Sheets*

FAQ – *Frequently Asked Questions*

HTML – *HyperText Markup Language*

HTML5 – *HyperText Markup Language revision 5*

HTTP – *Hypertext Transfer Protocol*

IDC – *International Data Corporation*

LAN – *Local Area Network*

MIDI – *Musical Instrument Digital Interface*

RF – Requisito Funcional

RNF – Requisito Não Funcional

UC – *Use Case*

URL – *Uniform Resource Locator*

UTF-8 – *Transforation Format–8-bit*

XML – *Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

A música é muito presente na vida das pessoas. Todos os dias as pessoas ouvem algum tipo de música, mesmo que sem querer, ao caminhar na rua ou por causa de um vizinho barulhento. Existem estudos que comprovam a influência da música no desempenho escolar e até ao fazer exercícios. Abordando-se a música como vetor motivacional, esta tem uma representação neuropsicológica extensa (HEILMAN et al., 1986).

Conforme Gandelman (2012), antes do século IX as músicas eram registradas apenas na memória. Por volta deste século, para auxiliar os que cantavam e tocavam instrumentos, foram criados símbolos que acompanhavam os textos apresentados. Gandelman ainda cita que até o século XIII, a música cantada era predominante e havia pouca música instrumental, por isso, existia uma forte ligação entre a música e a palavra. Foi após a criação do primeiro relógio na cidade de Londres, no século XIII, que surgiu a ideia de medir a duração dos sons.

Inicialmente as músicas eram gravadas de formas variadas, com símbolos representando as notas e, conforme citado por Rezende (2008), o ritmo musical era dado pelo ritmo das palavras. Cada região tinha uma forma diferente de notação. A padronização da escrita musical ficou mais forte com os copistas profissionais, que não estavam acostumados às peculiaridades da escrita de cada região, e as ordens religiosas tornaram obrigatória a *squore notation* para os livros de canto.

A necessidade de gravar o ritmo foi aumentando com a dificuldade percebida ao escrever os sons de várias vozes numa música. Franco de Cologne desenvolveu um sistema que permitia indicar os modos rítmicos. A partir disso, no século XIV, foram desenvolvidos quatro níveis principais de valores das notas (REZENDE, 2008).

Assim foi se desenvolvendo a partitura, que é uma forma de armazenar as informações de uma música. Os músicos normalmente carregam consigo pastas com várias partituras impressas em folhas de papel. Uma música pode ter várias páginas de partituras com a forma de reproduzir os sons criados para cada instrumento.

A música é bastante presente na sociedade e os dispositivos móveis também estão ocupando seu espaço. Conforme dados do IBOPE (2013), o número de pessoas com 10 anos ou mais que possuem *smartphones* com acesso à internet cresceu 42% de janeiro a julho de 2013. As pessoas dificilmente ficam longe de seus celulares e *tablets*. Estes dispositivos estão com uma capacidade de armazenamento e processamento cada vez maior. Para o conforto dos usuários, vários aplicativos foram e estão sendo desenvolvidos para estes aparelhos, permitindo rápido acesso a diversas funcionalidades.

Entre os sistemas operacionais utilizados nos dispositivos móveis, o Android tem quase 80% do mercado de *smartphones*, conforme pesquisa realizada pelo *International Data Coporation* (IDC) *Worldwide Mobile Phone Tracker* em 7 de agosto de 2013 (IDC – PRESS RELEASE, 2013). No mercado de *tablets*, cerca de 63% dos sistemas operacionais utilizados são Android (MARKETWATCH, 2013).

A maioria dos aplicativos disponíveis para leitura de partituras atualmente são somente para computadores pessoais, conforme Brandão (2013) e mesmo assim, não são específicos para leitura de partituras. Os aplicativos existentes que possuem funcionalidade para leitura de partituras são mais robustos e focam na edição e criação de partituras, que normalmente são geradas pelos aplicativos e depois impressas para leitura.

Os aplicativos para dispositivos móveis que auxiliam na criação de músicas e no aprendizado de instrumentos musicais costumam usar pouco a partitura pela dificuldade de visualização, ou utilização pelo usuário. Caso um músico queira acessar uma partitura pelo seu celular ou *tablet*, é comum ele ter que ficar manipulando o aparelho para visualizar a parte da música a ser reproduzida.

Para execução de músicas em grupo, com vários instrumentos musicais, não são conhecidas aplicações que sincronizem a leitura de partituras. Cada músico possui uma partitura diferente e precisa acompanhar atentamente o compasso em que a música está para não antecipar ou adiar a reprodução de sua parte. O desenvolvimento de um aplicativo que sincronize partituras em dispositivos diferentes poderia facilitar o acompanhamento da música.

Neste contexto, este trabalho visa criar em protótipo que facilite a leitura e visualização da partitura em um dispositivo móvel. Além da facilidade para ler e levar partituras no seu aparelho, o protótipo proposto também tem como objetivo sincronizar mais de um dispositivo para que ao executar uma música, todos estejam visualizando o mesmo compasso na partitura.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é disponibilizar um protótipo para dispositivos móveis na plataforma Android que facilite a leitura e o acompanhamento de partituras musicais em grupo.

Os objetivos específicos do trabalho são:

a) desenhar a partitura a partir do formato de arquivo de notação musical aberto baseado em *eXtensible Markup Language* (XML), o `MusicXML`;

b) possibilitar que o usuário acompanhe a execução das partituras sem a necessidade de manipular o dispositivo, após o início da execução, para visualizar o compasso a ser reproduzido;
c) garantir que os músicos estejam sincronizados no mesmo momento da música em aparelhos diferentes.

1.2 ESTRUTURA DO TRABALHO

O trabalho foi dividido em quatro capítulos: no primeiro, consta uma introdução sobre o assunto. O segundo capítulo apresenta a fundamentação teórica necessária para o desenvolvimento do protótipo, passando por métodos de comunicação entre os dispositivos móveis, a *Application Programming Interface* (API) `VexFlow` e o *framework* do Cordova/Phonegap. O terceiro capítulo traz informações sobre o desenvolvimento do trabalho, requisitos, especificação, implementação, operacionalidade e resultados. No quarto capítulo são relatadas as conclusões do trabalho e suas possíveis extensões.

2 FUNDAMENTAÇÃO TEÓRICA

Na seção 2.1, constam explicações sobre a prática musical em conjunto, contextualizando um dos objetivos do protótipo. Na seção 2.2, são apresentadas informações sobre como será feita a sincronização e comunicação entre os dispositivos, via *WebSocket*. A seção 2.3 é referente à interface do protótipo, trazendo os conceitos básicos do formato `MusicXML`, da biblioteca `VexFlow` e da plataforma Cordova. Por fim, a seção 2.4 descreve trabalhos correlatos ao protótipo proposto.

2.1 PRÁTICA MUSICAL EM CONJUNTO

Segundo Centro de Artes e Educação Física (2010), a formação de grupos instrumentais é bastante antiga. Os romanos já utilizavam grupos de cornetas e outros instrumentos em suas festas. Sempre que houver músicos próximos, existe a possibilidade de que eles formem conjuntos. A mistura dos sons de cada instrumento em diferentes momentos da música pode tornar a melodia ainda mais interessante do que numa apresentação solo.

Uma orquestra pode ter classes instrumentais de cordas, madeiras, metais, instrumentos de percussão e teclas. Antigamente entre os instrumentistas normalmente era escolhido um líder que conduzia a execução da música. Hoje as orquestras são conduzidas por um maestro (CENTRO DE ARTES E EDUCAÇÃO FÍSICA, 2010).

Existem termos diferentes para definir um conjunto de músicos, dependendo da quantidade de músicos no grupo, do tipo de música reproduzida, de como é feita a condução do grupo, entre outros. Comparando os grupos apresentados pelo Centro de Artes e Educação Física (2010), entretanto, é possível verificar que em todos os conjuntos o grupo precisa da marcação de tempos, andamento, etc. Além disso, quando não há um maestro ou regente, normalmente a condução é feita por um dos músicos, que busca garantir a harmonia e a sincronia entre o grupo.

Na prática musical em conjunto, é fundamental o sincronismo entre os músicos. Caso algum integrante do conjunto musical inicie ou reproduza as notas no tempo errado, toda a execução pode ser comprometida. Este sincronismo, porém, é natural. Conforme Schulz (2010), são comuns recitais em que os "instrumentistas sincronizam suas performances sem o recurso visual, em outros casos eles se olham somente no início de cada obra".

Clayton, Sager e Will (2004) fizeram estudos empíricos analisando o entrosamento dos músicos que tocam juntos. Um dos casos de estudo foi feito com dois músicos, que foram orientados a reproduzir uma música com uma fase inicial e três mudanças de padrão. Foi verificado que o ponto de referência para o sincronismo entre eles é o início de cada padrão.

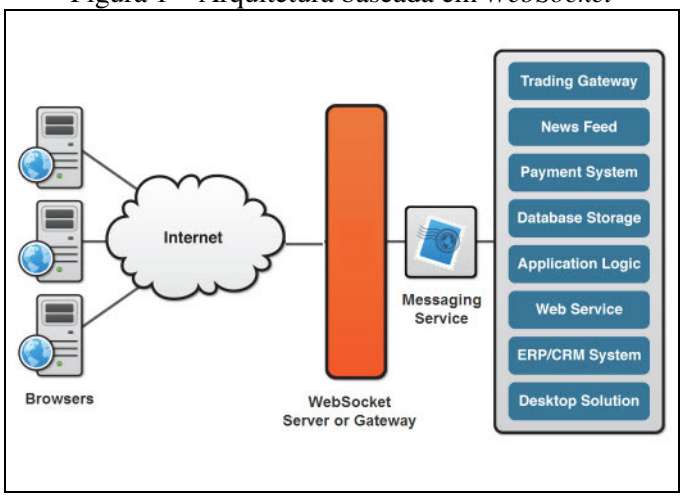
2.2 WEBSOCKET

O protocolo *WebSocket* permite comunicação bidirecional entre um cliente e um servidor. Segundo Fette e Melnikov (2011), o objetivo desta tecnologia é fornecer outra opção para aplicações baseadas em navegador, que necessitam comunicação bidirecional com os servidores que não dependem de múltiplas conexões *Hypertext Transfer Protocol* (HTTP) abertas.

Conforme Kaazing Corporation (2013), a especificação do *WebSocket* para HTML5 define um canal de comunicação *full-duplex* que opera através de um único *socket Transmission Control Protocol* (TCP) na rede. A especificação também aponta possíveis perigos na rede, como *proxies* e *firewalls*, com *streaming* através de qualquer conexão. Os servidores ficam menos carregados, portanto as máquinas existentes podem apoiar mais conexões simultâneas.

A Figura 1 mostra uma arquitetura baseada em *WebSocket* em que navegadores, ou *browsers* em inglês, usam uma conexão *full-duplex*.

Figura 1 – Arquitetura baseada em *WebSocket*



Fonte: Kaazing Corporation (2013).

Campanini (2012) explica que apesar de ter sido criado para implementação na web, o *WebSocket* "pode ser usado por qualquer cliente, servidor de aplicações, ou até em dispositivos móveis, desde que atenda aos requisitos para ser cliente ou para ser servidor, presentes na especificação do protocolo". A tecnologia exige menos infra-estrutura, permitindo trocas de mensagens em grandes volumes em cenários como *Service-Oriented Architecture* (SOA) e *cloud-computing*.

Para Campanini (2012), as aplicações que dependem de atualização em tempo real como bate-papos, jogos *multiplayers online*, mapas interativos e ferramentas de colaboração *online* podem utilizar *WebSockets*, evitando problemas de escalabilidade e desempenho.

Ainda segundo Campanini (2012), aplicações performáticas podem ser construídas sem "a complexidade de soluções anteriores".

Nos quadros 1 e 2 têm-se dois trechos de códigos onde Kaazing Corporation (2013) exemplifica como é feita a comunicação entre cliente e servidor via *WebSocket*. A conexão inicia em HTTP, depois o cliente solicita a alteração do modo e o servidor responde. No Quadro 1 o navegador manda uma requisição ao servidor, solicitando a alteração do protocolo utilizado de HTTP para *WebSocket*.

Quadro 1 – Exemplo de comunicação do cliente com o servidor

```
GET ws://echo.websocket.org/?encoding=text HTTP/1.1
Origin: http://websocket.org
Cookie: __utma=99as
Connection: Upgrade
Host: echo.websocket.org
Sec-WebSocket-Key: uRovscZjNol/umbTt5uKmw==
Upgrade: websocket
Sec-WebSocket-Version: 13
```

Fonte: Kaazing Corporation (2013).

O Quadro 2 mostra a resposta do servidor, caso ele tenha entendido e concordado com o protocolo *WebSocket*.

Quadro 2 – Exemplo de resposta do servidor para o cliente

```
HTTP/1.1 101 WebSocket Protocol Handshake
Date: Fri, 10 Feb 2012 17:38:18 GMT
Connection: Upgrade
Server: Kaazing Gateway
Upgrade: WebSocket
Access-Control-Allow-Origin: http://websocket.org
Access-Control-Allow-Credentials: true
Sec-WebSocket-Accept: rLHCkw/SKsO9GAH/ZSFhBATDKrU=
Access-Control-Allow-Headers: content-type
```

Fonte: Kaazing Corporation (2013).

Kaazing Corporation (2013), aponta que depois de estabelecida a conexão via *WebSocket* é possível enviar arquivos de texto e binários em qualquer direção, seja cliente para servidor ou servidor para cliente, ao mesmo tempo.

2.3 LEITURA E REPRESENTAÇÃO DE PARTITURAS

Nesta seção são apresentados o formato de entrada de dados `MusicXML`, a biblioteca `VexFlow` e a plataforma Cordova.

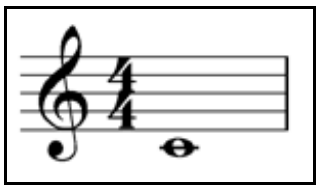
2.3.1 Formato `MusicXML`

`MusicXML` é um formato de arquivo de notação musical aberto baseado em XML, desenvolvido pela Recordare LLC. Conforme MakeMusic (2013a), o formato é utilizado por vários aplicativos e foi projetado para compartilhamento de arquivos de partituras musicais

entre aplicativos e para seu armazenamento. Este é um formato que complementa formatos de arquivo nativos utilizado por *softwares* de notação musical como o Finale e outros.

Para exemplificar, o desenho da Figura 2 contém uma semibreve no dó central em dó maior baseada em compasso 4/4.

Figura 2 – Desenho da partitura da nota semibreve no dó central em dó maior baseada em tempo 4/4



Fonte: MakeMusic (2013b).

O arquivo `MusicXML` correspondente ao desenho da Figura 2 é exibido no Quadro 3.

Quadro 3 – Arquivo em `MusicXML`

```
 1      <?xml version="1.0" encoding="UTF-8" standalone="no"?>
 2      <!DOCTYPE score-partwise PUBLIC
 3         "-//Recordare//DTD MusicXML 3.0 Partwise//EN"
 4          "http://www.musicxml.org/dtds/partwise.dtd">
 5      <score-partwise version="3.0">
 6        <part-list>
 7          <score-part id="P1">
 8            <part-name>Music</part-name>
 9          </score-part>
10        </part-list>
11        <part id="P1">
12          <measure number="1">
13            <attributes>
14              <divisions>1</divisions>
15              <key>
16                <fifths>0</fifths>
17              </key>
18              <time>
19                <beats>4</beats>
20                <beat-type>4</beat-type>
21              </time>
22              <clef>
23                <sign>G</sign>
24                <line>2</line>
25              </clef>
26            </attributes>
27            <note>
28              <pitch>
29                <step>C</step>
30                <octave>4</octave>
31              </pitch>
32              <duration>4</duration>
33              <type>whole</type>
34            </note>
35          </measure>
36        </part>
37      </score-partwise>
```

Fonte: MakeMusic (2013b).

Conforme MakeMusic (2013b), a linha 1 contém a declaração requerida para todos os documentos XML. O Unicode está em codificação *Universal Character Set* (UCS) *Transforation Format–8-bit* (UTF-8) e o *standalone* está com o valor `no`, que define que o documento tem uma definição externa em outro arquivo.

Nas linhas 2, 3 e 4 é definido que o `MusicXML` está sendo utilizado. A *Uniform Resource Locator* (URL) serve apenas como referência. A *tag* de DOCTYPE facilita a validação do `MusicXML` para todas as aplicações, baseadas em *Document Type Definition* (DTD) ou XML *Schema Definition* (XSD). A linha 5 define o tipo de documento `root`. Esta *tag* serve para identificar mais facilmente qual versão de `MusicXML` está sendo utilizada.

Segundo MakeMusic (2013b), a partitura pode ser dividida em partes ou medições: se a partitura for `timewise`, ela é dividida por medidas e cada medida é feita de partes; se a partitura for `partwise`, como no exemplo citado, as partes é que são dividas por medições. O cabeçalho que lista as diferentes partes na partitura neste exemplo se encontra nas linhas de 6 a 10.

Após definida a lista de partes da partitura, é definida cada parte do documento. O identificador (id) da *tag* que vai descrever cada parte `(<part id="P1">)` deve ser igual ao id informado no cabeçalho com a lista `(<score-part id="P1">)`. A linha 12 está definindo o primeiro compasso da parte 1, entre as linhas 13 e 26 são definidos atributos da nota a ser tocada, como o tom e a duração do compasso, por exemplo; e entre as linhas 27 e 34 são definidas as informações da nota.

No arquivo apresentado no Quadro 3 constam algumas das *tags* mais utilizadas no formato `MusicXML`. Essas e algumas outras *tags* podem ser vistas em uma lista no Apêndice A. A partir de um arquivo `MusicXML` exportado de outros aplicativos ou retirado da internet é possível geração da representação visual utilizando o `VexFlow`.

2.3.2 Vexflow

O `Vexflow` é uma Interface de Programação de Aplicativo, uma API, com código aberto escrita em JavaScript. A API renderiza notação musical, suporta *HyperText Markup Language revision 5* (HTML5) e pode suportar *Scalable Vector Graphics* (SVG) incluindo outra biblioteca Javascript (THE VEXFLOW TUTORIAL, 2014).

Como o código é aberto, existem várias implementações e funcionalidades dessa biblioteca disponíveis. A funcionalidade básica é o desenho da partitura na tela, com várias opções de articulações, porém a API possibilita edição da partitura na tela, destaque das notas

no compasso, reprodução de áudio da música com diferentes instrumentos, entre várias outras funcionalidades (THE VEXFLOW TUTORIAL, 2014).

O desenho da partitura é feito pelo elemento Canvas do HTML5. O Canvas permite a geração de gráficos dinamicamente. No Quadro 4 é possível visualizar um exemplo do código necessário para desenhar a pauta com a clave de sol:

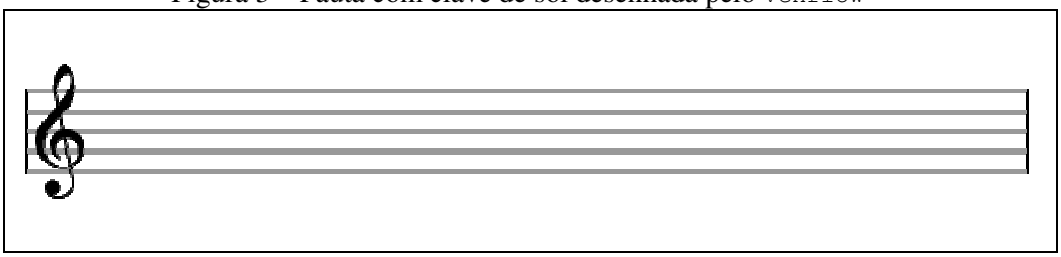
Quadro 4 – Pauta com clave de sol

```
1     <div class="description one">
2           <div class="example a" example="a">
3                 <canvas width=525 height=120></canvas>
4                 <script>
5                       var canvas = $("div.one div.a canvas")[0];
6                       var renderer = new Vex.Flow.Renderer(canvas,
7                                         Vex.Flow.Renderer.Backends.CANVAS);
8
9                       var ctx = renderer.getContext();
10                      var stave = new Vex.Flow.Stave(10, 0, 500);
11                      stave.addClef("treble").setContext(ctx).draw();
12                </script>
13          </div>
14      </div>
```

Fonte: The VexFlow tutorial (2014).

O desenho que aparecerá na tela, correspondente ao código do Quadro 4, é a Figura 3.

Figura 3 – Pauta com clave de sol desenhada pelo `Vexflow`



Fonte: The VexFlow tutorial (2014).

No código do Quadro 4, linha 10, é criada uma *stave*, que são as pautas, as cinco linhas da partitura. À pauta é adicionada a clave, *Clef*, na linha 11. *Treble* é a clave de sol, conforme a desenhada na Figura 3.

Segundo Ringwalt (2014) para adicionar uma nota, é necessário criar um *StaveNote*, que armazena uma nota simples, ou um acorde, e sua duração. Notas são agrupadas em *Voices* (ou compassos, em português), que têm uma duração de tempo definida. A soma da duração de cada nota nesse conjunto, incluindo as pausas, deve ser igual a duração que foi definida para o compasso.

As *voices* são agrupadas em *VoiceGroups*, que pode conter uma ou mais *Voices*. Depois disso o *Formatter* junta as *VoiceGroups* e alinha-as na partitura, para que as notas fiquem bem distribuídas, assim como palavras em um texto.

Segundo Ringwalt (2014), para fazer o *parsing* do `MusicXML` é necessário utilizar uma biblioteca desenvolvida paralelamente ao código original do `VexFlow`. Utilizando essa bilioteca, é criado um documento `Vex.Flow.Document` passando como parâmetro uma `String` com o XML. Depois disso é criado um `contextBuilder`, que é passado por parâmetro para o método `drawBlock` que vai desenhar a partitura. Um exemplo do código está no Quadro 5.

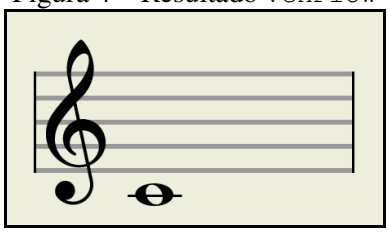
Quadro 5 – Exemplo de *parsing* do `MusicXML`

```
1. var docString = '<?xml version="1.0" encoding="UTF-8" standalone="no"?>
2.
3.
4.    <!-- aqui é definido o arquivo musicxml -->\
5.    </score-partwise>';
6.
7. var doc = new Vex.Flow.Document(docString);
8. var ctx = new contextBuilder(options.canvas_sel, 300, 120);
9. doc.getFormatter().setWidth(300).drawBlock(0, ctx);
```

Fonte: adaptado de Ringwalt (2014).

Ao aplicar o código do Quadro 5 basta substituir a variável `docString` do exemplo pelo arquivo `MusicXML` do Quadro 3 e o `VexFlow` gera a imagem da Figura 4.

Figura 4 – Resultado `VexFlow`



Se for comparado o resultado do `VexFlow` com a imagem da partitura da Figura 2, percebe-se que na representação da imagem com o `VexFlow` a partitura não apresenta a métrica (fração após a clave). Isso ocorre pois na API essa informação somente é utilizada para validar se a partitura está correta e para desenhá-la corretamente, mas não é apresentada na partitura.

2.3.3 Cordova

O Apache Cordova é um projeto de código aberto que permite a utilização de padrões web para o desenvolvimento de aplicativos multi-plataforma para dispositivos móveis. O *framework* evita o uso de linguagens nativas de cada plataforma com o desenvolvimento em *HyperText Markup Language* (HTML), Javascript e *Cascading Style Sheets* (CSS). (APACHE, 2013).

Existe uma confusão entre os termos Cordova e Phonegap. O blog do PhoneGap (ADOBE, 2012) explica que o PhoneGap, que era da Adobe, foi doado para a Apache. Nesta

transição o projeto mudou seu nome para Cordova, porém o PhoneGap ainda existe, com código aberto e é uma distribuição do Apache Cordova. É possível fazer uma analogia da relação do Cordova com o Phonegap pensando no Cordova como um motor que alimenta o PhoneGap, similar ao modo como o WebKit alimenta o Chrome ou Safari.

Conforme Prieto (2012), apesar de utilizar ferramentas de desenvolvimento web, o PhoneGap possibilita a criação de aplicativos híbridos. Ambros (2013) explica que esse tipo de aplicativo é parcialmente nativo, parcialmente *webapp*. *Webapps* acessam "funcionalidades semelhantes a um aplicativo nativo", porém vários recursos são inacessíveis pelo navegador. Aplicativos Cordova/PhoneGap híbridos, "podem aproveitar todas as funcionalidades do dispositivo".

Baixo custo de desenvolvimento, facilidade de manutenção e independência de plataforma são vantagens descritas por Ambros (2013) ao desenvolver um aplicativo com Cordova. As linguagens utilizadas no desenvolvimento podem ter sido previamente aprendidas e geram projetos para diferentes plataformas. Aplicativos nativos exigem conhecimentos específicos de linguagens e ferramentas do dispositivo e sistema operacional.

As APIs disponibilizadas pelo Apache Cordova fornecem acesso a várias funcionalidades nativas. Além delas, estão disponíveis *plugins* de terceiros. É possível ainda criar novas APIs para acesso a outras funcionalidades, nativas ou não, que os *plugins* existentes ainda não suportem (APACHE, 2014a). A Figura 5 mostra as ferramentas de desenvolvimento e APIs disponíveis para cada plataforma na versão 3.5.0 do Cordova.

Figura 5 – Suporte da Plataforma Cordova

	amazon-fireos	android	blackberry10	Firefox OS	ios	Ubuntu	wp7 (Windows Phone 7)	wp8 (Windows Phone 8)	win8 (Windows 8)	tizen
cordova CLI	✓ Mac, Windows, Linux	✓ Mac, Windows, Linux	✓ Mac, Windows	✓ Mac, Windows, Linux	✓ Mac	✓ Ubuntu	✓ Windows	✓ Windows	✓	✗
Embedded WebView	✓ (see details)	✓ (see details)	✗	✗	✓ (see details)	✓	✗	✗	✗	✗
Plug-in Interface	✓ (see details)	✓ (see details)	✓ (see details)	✗	✓ (see details)	✓	✓ (see details)	✓ (see details)	✓	✗
Platform APIs										
Accelerometer*	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Camera*	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Capture*	✓	✓	✓	✗	✓	✓	✓	✓	✗	✗
Compass*	✓	✓	✓	✗	✓ (3GS+)	✓	✓	✓	✓	✓
Connection*	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
Contacts*	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
Device*	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Events	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
File*	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗
Geolocation*	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Globalization*	✓	✓	✗	✗	✓	✓	✓	✓	✗	✗
InAppBrowser*	✓	✓	✓	✗	✓	✓	✓	✓	uses iframe	✗
Media*	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
Notification*	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
Splashscreen*	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗
Storage	✓	✓	✓	✗	✓	✓	✓ localStorage & indexedDB	✓ localStorage & indexedDB	✓ localStorage & indexedDB	✓

Fonte: Apache (2014a).

Segundo Prieto (2012), apesar de críticas sobre a performance de aplicativos desenvolvidos com a plataforma, um aplicativo PhoneGap feito de forma correta é “poderoso”.

A instalação do Cordova pode ser feita pela linha de comando do Node.js que já deve estar instalado na máquina. O Node.js é uma plataforma para criação de aplicações escaláveis que usa um modelo baseado em eventos sem bloqueio de *Input*/*Output* (JOYENT, 2014). Na linha de comando do Node.js, utiliza-se o comando `npm install -g cordova` para instalar o Cordova.

Depois da instalação do Cordova é possível criar projetos, gerenciar plataformas e *plugins* e até fazer o *deploy* do aplicativo no emulador ou no dispositivo móvel via linha de comando. A seguir são descritos alguns comandos da interface de linha de comando do Cordova, ou *Cordova command-line interface* (CLI).

2.3.3.1 Utilização do Cordova

Para criar um projeto Cordova, na linha de comando do Node.js deve ser chamado o `cordova create`, seguido pelos parâmetros da pasta onde a aplicação deve ser armazenada, de uma identificação de domínio reverso (opcional) e do nome da aplicação (também opcional). Exemplo: `cordova create hello com.example.hello HelloWorld` (APACHE, 2014b).

Após criado o projeto, é necessário incluir pelo menos uma plataforma. Para isso, deve-se acessar a pasta do projeto com o comando `cd nomePasta` e então adicionar as plataformas desejadas com o comando `cordova platform add` seguido pela plataforma: ios, amazon-fireos, android, entre outros. Exemplo: `cordova platform add android` (APACHE, 2014b).

Com pelo menos uma plataforma adicionada já é possível executar a aplicação, através de um emulador ou de um dispositivo da plataforma escolhida. Para visualizar uma lista de plataformas adicionadas ao projeto, utiliza-se o comando `cordova platforms ls` (APACHE, 2014b).

O Cordova traz por padrão um projeto `HelloWorld`, que mostra que o dispositivo está pronto. Dentro do projeto deve haver a pasta `www`. Nela consta o arquivo index.html que é chamado como página inicial por padrão ao executar a aplicação. As pastas de arquivos JavaScript, CSS e outros que podem ser utilizados também constam neste diretório (APACHE, 2014b).

O *build* é feito através do comando `cordova build`. É possível também gerar o *build* de uma plataforma específica adicionando o nome da plataforma no final do comando: `cordova build android` (APACHE, 2014b).

Para executar a aplicação, é utilizado o comando `cordova run` seguido pela plataforma que já foi adicionada ao projeto. Exemplo: `cordova run android`. Este comando faz o *deploy* da aplicação no emulador, caso nenhum dispositivo móvel da plataforma informada esteja disponível. Caso ele identifique um dispositivo móvel conectado, o comando fará o *deploy* no dispositivo (APACHE, 2014b).

Com o Cordova executando pelo dispositivo móvel, não é possível depurar o código. Para isso são utilizados aplicativos como o Weinre. Este é um depurador para páginas web que faz parte do projeto Apache Cordova, desenvolvido para trabalhar de forma remota e, em particular, para permitir a depuração de páginas web em dispositivos móveis (WEINRE, 2012).

2.4 TRABALHOS CORRELATOS

A maioria dos aplicativos existentes para leitura de partituras foram desenvolvidos para utilização em computadores ou *notebooks*, considerando os aplicativos citados por Brandão (2013). Ainda assim, normalmente aplicativos têm a geração de partituras como uma funcionalidade e não focam na leitura delas.

Foram selecionados, portanto, alguns trabalhos que apresentam ligação com funcionalidades propostas por este projeto: "Ferramenta para Criação de Composições Musicais para Android" (ALVARENGA, 2013), "*Goodfeel*" (DANCING DOTS, 2014), "*Midi Sheet Music*" (VAIDYANATHAN, 2013) e "*Towards the Implentation of a Generic Platform for Networked Music Performance: The DIAMOUSES Approach*" (AKOUMIANAKIS et al., 2008).

2.4.1 Ferramenta para Criação de Composições Musicais para Android

A ferramenta para criação de composições musicais para Android foi criada por Alvarenga (2013). O projeto desenvolvido permite carregar um arquivo de texto previamente salvo no aplicativo. A Figura 6 mostra a interface principal do aplicativo após ter selecionado uma música salva no dispositivo.

Figura 6 – Interface principal do aplicativo após abertura de música

Fonte: Alvarenga (2013).

A aplicação permite a escolha de instrumentos e a inclusão de *samples* dos diferentes instrumentos à música. Segundo Alvarenga (2013), após escolher o *sample* o usuário pode arrastá-lo até a *timeline* para atribuí-lo a um tempo de execução. Caso necessário, ainda é possível editar a posição do *sample* já inserido, através de um toque longo e depois arrastando

o componente até o local desejado. Para excluir o *sample* basta arrastá-lo para fora da *timeline*.

É possível aumentar, diminuir o volume, salvar as alterações feitas no arquivo e reproduzir a música. Ao reproduzir a música, o tempo da *timeline* que está sendo executado no momento é destacado.

2.4.2 *Goodfeel*

Este é um produto pago, criado pela Dancing Dots (2014), que permite a conversão de partituras impressas para um formato de música em braile, e de volta para o formato de partitura impressa. É possível revisar partituras em braile com interpretações verbais e musicais. Os músicos cegos podem criar músicas independentemente e criar partituras em tinta das próprias ideias.

O Goodfeel possibilita a importação de arquivos de diversos aplicativos utilizados na criação de partituras musicais, como o Finale e o Sibelius, através da exportação de arquivos com formato `MusicXML`. É possível importar e exportar arquivos `MusicXML`. Entre outras funcionalidades, também é possível incluir *plugins* que facilitam a utilização do sistema. A conversão das partituras permite a fácil colaboração de professores, alunos e colegas com visão e cegos (DANCING DOTS, 2014). A Figura 7 apresenta a tela principal do produto.

Figura 7 – Tela principal do Goodfeel

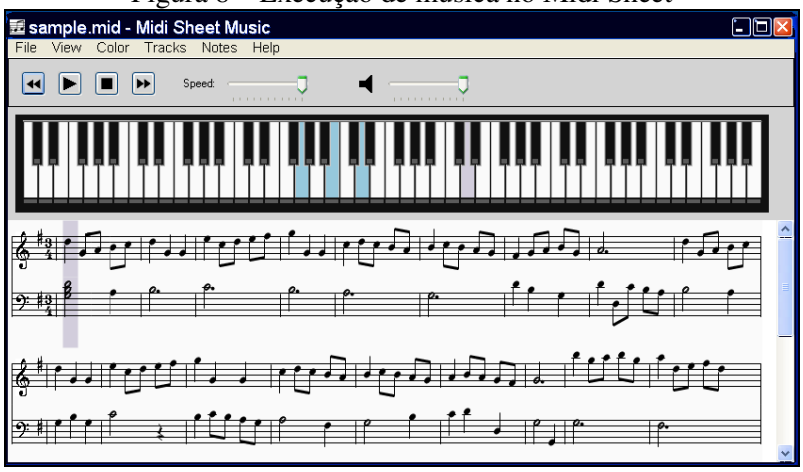
Fonte: Simon (2012, apud DANCING DOTS, 2005, p. 30).

O sistema pode ser executado sem limitações em computadores de 32 e 64 bits do Windows, desde o 98 até o 7. Testes com o Windows 8 estão sendo realizados. Para utilizar o produto, o usuário pode digitalizar uma partitura impressa e/ou utilizar o Lime, também disponibilizado pela Dancing Dots, para importar arquivos *Musical Instrument Digital Interface* (MIDI) e `MusicXML`. Após importada a partitura é possível corrigir e ouvir as composições, modelando e editando partituras em braile.

2.4.3 *Midi Sheet Music*

O *Midi Sheet Music* é um aplicativo gratuito que reproduz arquivos MIDI e destaca as notas do piano e da partitura. O aplicativo desenvolvido por Madhav Vaidyanathan pode ser utilizado nos sistemas operacionais Windows, Mac OS X e Ubuntu Linux, conforme informado pelo próprio Vaidyanathan (2013). A Figura 8 apresenta a interface do aplicativo sendo executado.

Figura 8 – Execução de música no Midi Sheet



Fonte: Vaidyanathan (2013).

É possível reproduzir arquivos de músicas MIDI, escolher destacar a nota a ser reproduzida na partitura, no piano, mostrar as notas musicais e pintar as notas de acordo com seu código. Além disso, o aplicativo permite escolher quais instrumentos utilizar e separar a partitura por clave (mão esquerda e direita no piano).

Ajustes de tempo, clave e duração da nota também são possíveis, além da transposição de notas para cima ou para baixo. No *playback* é possível ajustar a velocidade e selecionar um trecho para tocar continuamente. Na parte superior da tela, onde consta o piano, é possível visualizar as teclas que devem ser pressionadas para reprodução das notas que estão destacadas na partitura na parte inferior da tela.

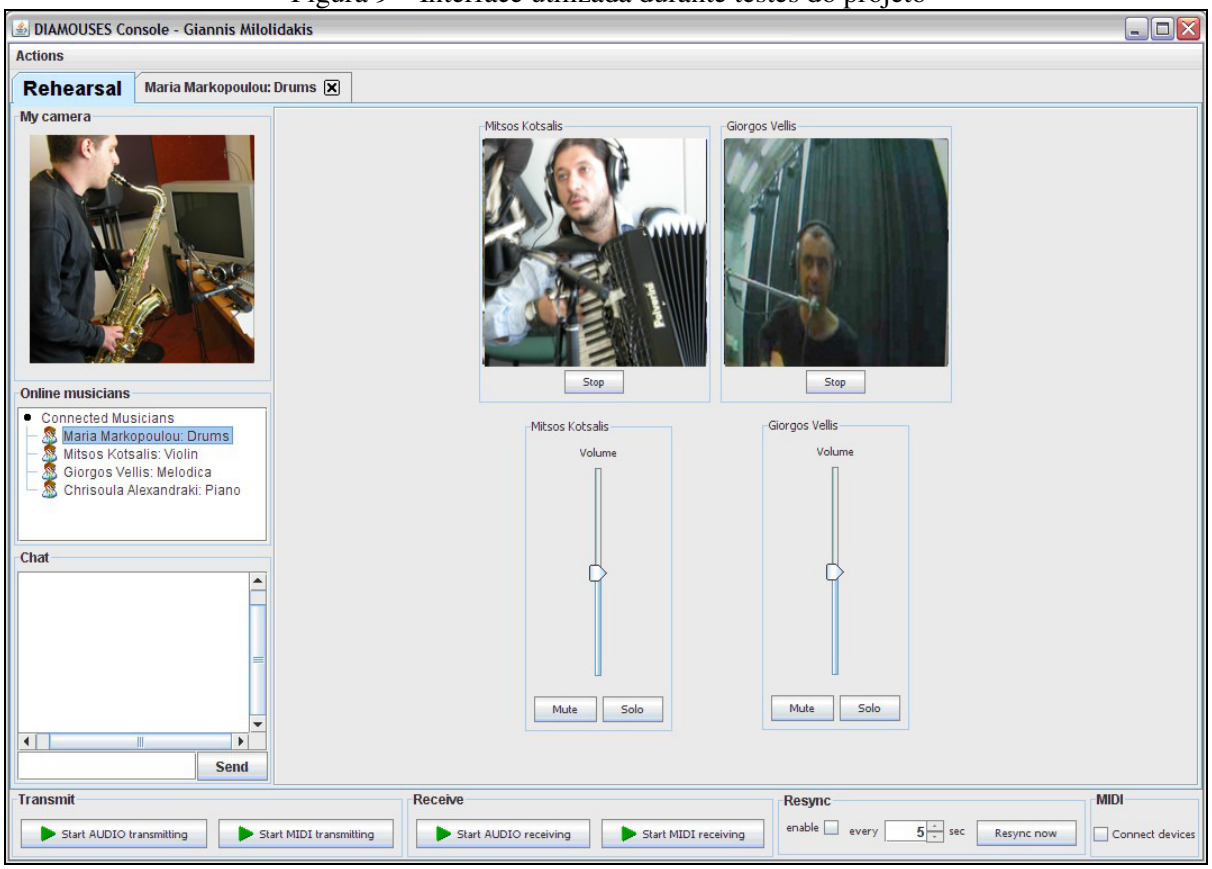
2.4.4 *Towards the implementation of a generic platform for networked music performance: the DIAMOUSES approach*

O trabalho de Akoumianakis et al. (2008) aborda o projeto DIAMOUSES, que visa permitir um grande número de cenários de aplicação para performance musical em rede. O principal objetivo, segundo os autores do trabalho, é fornecer uma plataforma genérica que suporte as especificidades de várias práticas de colaboração necessárias em diferentes cenários de performance musical.

No ambiente distribuído DIAMOUSES não há limitação de número de nós de redes conectados. Podem ser utilizadas como infraestruturas de rede *Local Area Network* (LAN), *Campus Area Networks* (CAN) ou *Wide Area Networks* (WAN). A arquitetura do sistema é definida para permitir que um ou mais nós da rede se comuniquem diretamente (*peer-to-peer*) ou por um servidor de *streaming* (cliente/servidor).

Conexões assíncronas também são possíveis utilizando um portal dedicado. Este portal permite o registro de usuários, suas competências e preferências musicais, além de permitir o compartilhamento de material audiovisual e outras informações de interesse. Uma API aberta fornece as principais funcionalidades, como a especificação dos endereços de rede, configuração dos dispositivos de *hardware*, captura de dados, transmissão e recepção de dados na rede e reprodução de dados. Foi desenvolvida uma plataforma para o sistema operacional Linux, com um servidor implementado na linguagem C++. Uma imagem da interface pode ser visualizada na Figura 9.

Figura 9 – Interface utilizada durante testes do projeto



Fonte: Akoumianakis et al. (2008).

O projeto possibilita que músicos utilizem microfones para captura de áudio, alto-falantes e fones de ouvido para reprodução, câmeras para captura de vídeo, dispositivos para captura e reprodução de MIDI e um computador consigam se comunicar. Cada músico pode tocar um instrumento musical diferente e passar o áudio, a imagem e outros dados em tempo real.

Akoumianakis et al. (2008) realizaram testes em infraestrutura LAN com 100Mbps. Durante a execução com os instrumentos musicais a latência de recebimento dos fluxos de dados não foi percebido pelos músicos. Ao reproduzir as músicas com os instrumentos MIDI foi observada a latência em detecção de tons e eventuais execuções de falsos eventos.

Outras partes deste projeto foram projetadas. Nelas seriam testadas outras infraestruturas e a avaliação de usuários proporcionaria a latência e perda de dados que podem ser toleradas. Ainda, segundo os autores, é esperado estimar a variação de tolerância em relação ao número de músicos participantes em uma sessão, as propriedades acústicas dos instrumentos e propriedades da música sendo reproduzida.

2.4.5 Comparação entre os trabalhos correlatos

O Quadro 6 apresenta de forma comparativa algumas características dos trabalhos apresentados na seção 2.4.

Quadro 6 – Comparação entre os trabalhos correlatos

características / trabalhos correlatos	Alvarenga (2013)	Dancing Dots (2014)	Vaidyanathan (2013)	Akoumianakis et al. (2008)
Plataforma	Android	Desktop	Desktop	Desktop
Arquitetura	Stand-alone	Stand-alone	Stand-alone	Cliente-Servidor
apresenta partitura	Não	Sim	Sim	Não
interação entre dispositivos	Não	Não	Não	Sim
acompanhamento da música	O tempo da *timeline* é destacado	Sim	Sim	Sim
permite importar ou exportar músicas em MusicXML	Não	Sim	Não	Não

Pode-se observar que dos trabalhos relacionados, apenas a ferramenta para criação de composições musicais para Android (ALVARENGA, 2013) é direcionada para usuários de dispositivos móveis. A partitura, porém, não é apresentada. No Goodfeel (DANCING DOTS, 2014), a partitura aparece para facilitar a utilização pelos músicos visuais. O *Midi Sheet Music* (VAIDYANATHAN, 2013) é o único dos trabalhos que foca na apresentação da partitura.

Sobre a interação entre os dispositivos, somente o *Diamouses approach* (AKOUMIANAKIS et al., 2008) possui uma funcionalidade para tal. O acompanhamento da música acontece em todos os trabalhos de alguma forma, evidenciando a importância de uma funcionalidade para esta finalidade. A importação ou exportação de arquivos `MusicXML` é destacada pela Dancing Dots na página da Goodfeel como uma forma de interação com outros produtos para composições musicais como o Finale e o Sibelius (DANCING DOTS, 2014). Entretanto, este foi o único trabalho correlato que possuía a funcionalidade.

Analisando as características dos trabalhos correlatos é possível perceber a falta de aplicativos com foco na apresentação da partitura em dispositivos móveis, principalmente quando se trata da apresentação simultânea para grupos de músicos. Daí surge a motivação para este trabalho.

3 DESENVOLVIMENTO

Neste capítulo são apresentadas as etapas do desenvolvimento do protótipo. A seção 3.1 traz os requisitos principais do trabalho. Na seção 3.2 consta a especificação, com modelos e diagramas. A implementação, com técnicas, ferramentas utilizadas e comentários sobre a operacionalidade da implementação estão na seção 3.3. A seção 3.4 detalha os resultados obtidos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O protótipo proposto deverá atender aos seguintes requisitos:

a) receber como entrada arquivos com o formato `MusicXML` (Requisito Funcional – RF);
b) identificar todos os itens do arquivo, como em uma partitura (RF);
c) apresentar a partitura na tela, conforme leitura do arquivo (RF);
d) acompanhar a execução da música pela partitura, centralizando na tela o compasso a ser executado (RF);
e) permitir aumentar ou diminuir o tamanho da partitura apresentada na tela (RF);
f) permitir que o usuário crie um estúdio e determine a hora de início (RF);
g) permitir o sincronismo fazendo com que todos os usuários conectados a determinado estúdio iniciem a visualização da partitura simultaneamente (RF);
h) ser implementado utilizando a linguagem de programação Java e JavaScript (Requisito Não Funcional - RNF);
i) utilizar o ambiente Eclipse com o *plugin Android Developer Tools* (ADT) para o desenvolvimento da aplicação (RNF);
j) disponibilizar o aplicativo para dispositivos com o sistema operacional Android (RNF);
k) utilizar a API do `VexFlow` (RNF);
l) utilizar a plataforma Cordova/PhoneGap (RNF);
m) utilizar o servidor Tomcat (RNF).

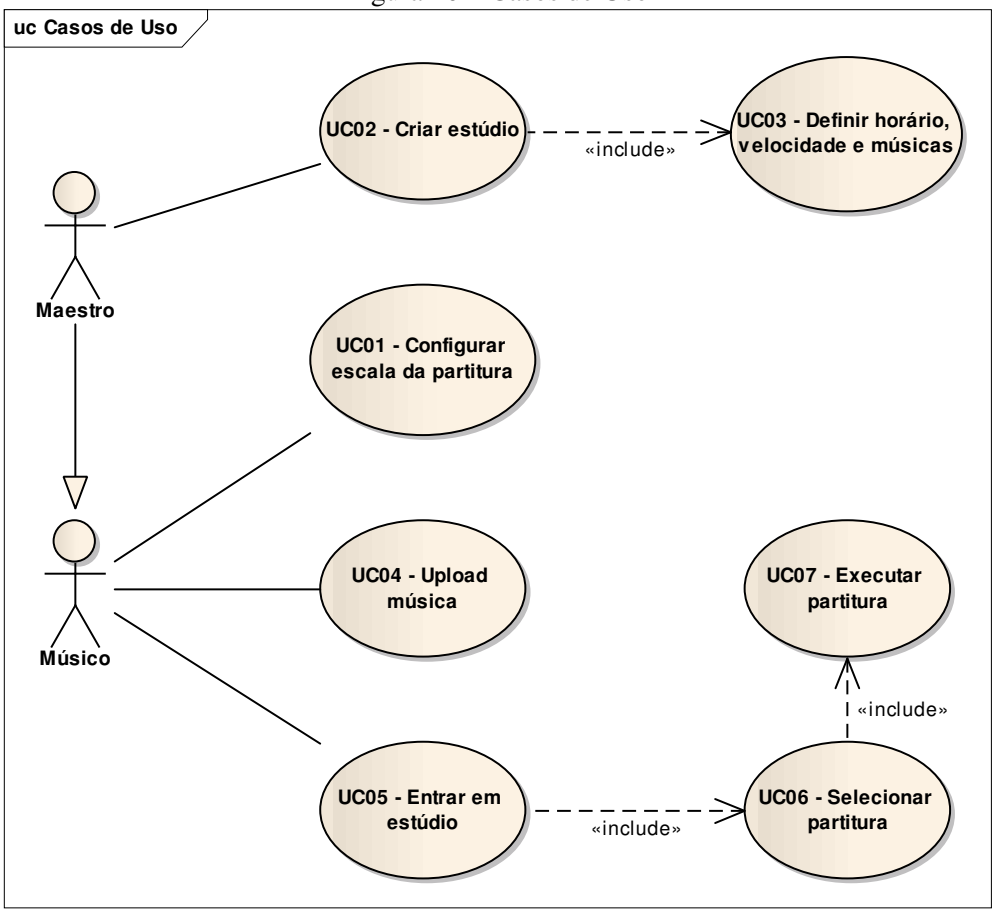
3.2 ESPECIFICAÇÃO

A especificação da aplicação foi criada utilizando a ferramenta Enterprise Architect (EA) versão 10.0. Com esta ferramenta foram desenvolvidos os diagramas de casos de uso, de classes e de sequência da *Unified Modeling Language* (UML).

3.2.1 Casos de uso

O diagrama apresentado na Figura 10 descreve os casos de uso da aplicação. Foram identificados dois papeis para um mesmo usuário, que pode ser chamado `Músico`. A partir do momento que ele cria um estúdio, ele assume o papel de `maestro` como um administrador da sala. Ao entrar num estúdio já existente ele assume o papel de `músico`.

Figura 10 – Casos de Uso



O caso de uso de `UC01 - Configurar escala da partitura` é utilizado quando o usuário desejar alterar o tamanho do desenho gerado pelo `VexFlow`. No caso de uso `UC02 - Criar estúdio`, o usuário cria um estúdio no protótipo para no caso de uso `UC03 - Definir horário, velocidade e músicas`, agendar um horário para execução, definindo uma velocidade para execução e as partituras que estarão disponíveis para os usuários.

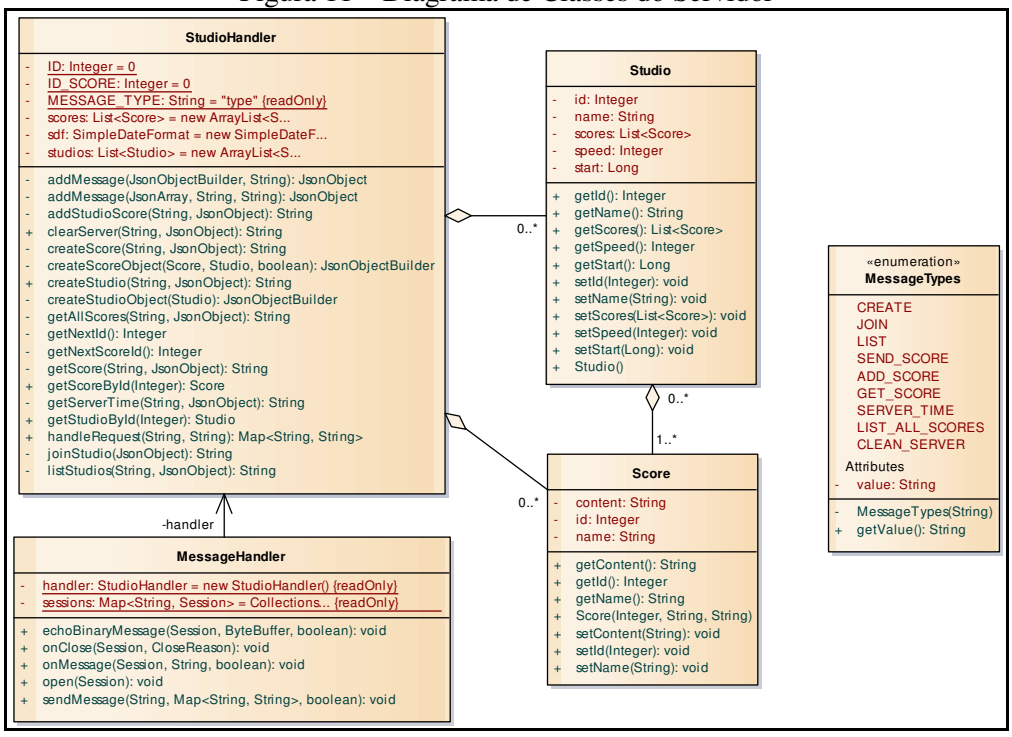
O caso de uso `UC04 - Upload música` é utilizado quando o músico enviar uma partitura do seu dispositivo para o servidor, disponibilizando-a para os outros usuários. O caso de uso `UC05 - Entrar em estúdio` é utilizado para que o usuário acesse um estúdio com horário de execução agendado e selecione a partitura a ser reproduzida no caso de uso `UC06 - Selecionar partitura`. Ao selecionar uma partitura, é agendada a execução da partitura, que é apresentada no `UC07 - Executar partitura`.

O detalhamento dos casos de uso da aplicação está no Apêndice B. Nele constam os objetivos, condições e cenários.

3.2.2 Diagrama de classes

Nesta seção constam os diagramas de classe do projeto, com relacionamentos e estruturas dos objetos. A Figura 11 mostra as classes do servidor, em Java e, na Figura 12 foram representadas as classes que constam no protótipo e a relação entre elas.

Figura 11 – Diagrama de Classes do Servidor



A classe `Score` é a partitura, que contém o arquivo `MusicXML`. Ela possui um `id` numérico, um nome e um `content`, que é o conteúdo do arquivo em forma de texto.

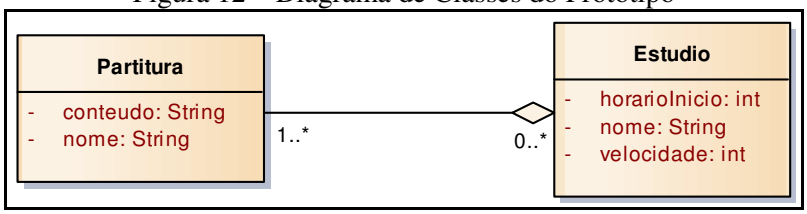
A classe `Studio` representa o estúdio. Esta classe possui um `id` numérico, um nome, uma lista de partituras, referente a uma única música, identificada pela `List<Score>` e dois atributos que guardam valores numéricos, da velocidade de execução da música e do horário (em milissegundos) do início da execução da partitura.

`MessageTypes` é um `enum` com os tipos de mensagens esperadas do cliente. `MessageHandler` é a classe que administra a comunicação via *WebSocket*, com funções como `open`, `onMessage`, `sendMessage` e `onClose`. Esta é a classe que tem as anotações da especificação do *WebSocket*. As sessões ficam gravadas em memória. Os dados não foram persistidos pois não fazem parte do objetivo principal do protótipo.

Por fim, a classe `StudioHandler` trata todas as requisições e decide para quais e quantos usuários a mensagem deve ser retornada. Assim como as sessões, os estúdios também

ficam gravados em memória sem persistência de dados. A Figura 12 apresenta de forma genérica como é o relacionamento entre os objetos principais do protótipo, desconsiderando o servidor.

Figura 12 – Diagrama de Classes do Protótipo



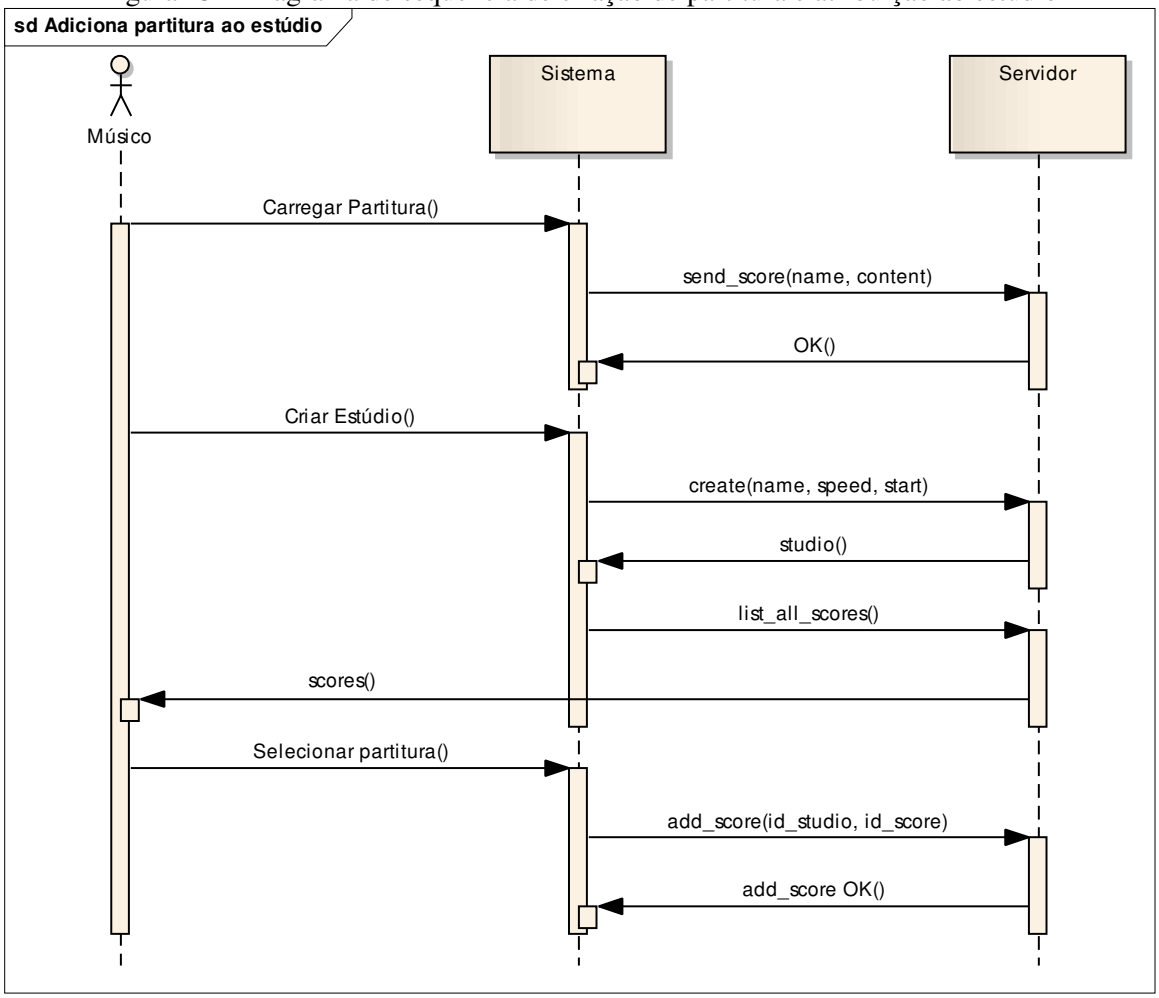
Uma partitura pode estar contida em zero ou mais estúdios. O estúdio tem as informações de horário de início, velocidade e nome. A partitura contém um nome e um `conteudo`, que é o arquivo `MusicXML` em formato de texto.

3.2.3 Diagramas de sequência

A seguir são apresentados os diagramas de sequência definidos para a aplicação. Foram criados dois diagramas, um para adicionar uma partitura a um estúdio e o outro para o acesso a um estúdio ou grupo de músicos e recebimento da partitura.

A Figura 13 demonstra o fluxo de atividades do `Músico` e sua interação com o servidor para a criação de uma partitura, com nome e conteúdo, e de um estúdio, com nome, velocidade e horário de início.

Figura 13 – Diagrama de sequência de criação de partitura e atribuição ao estúdio



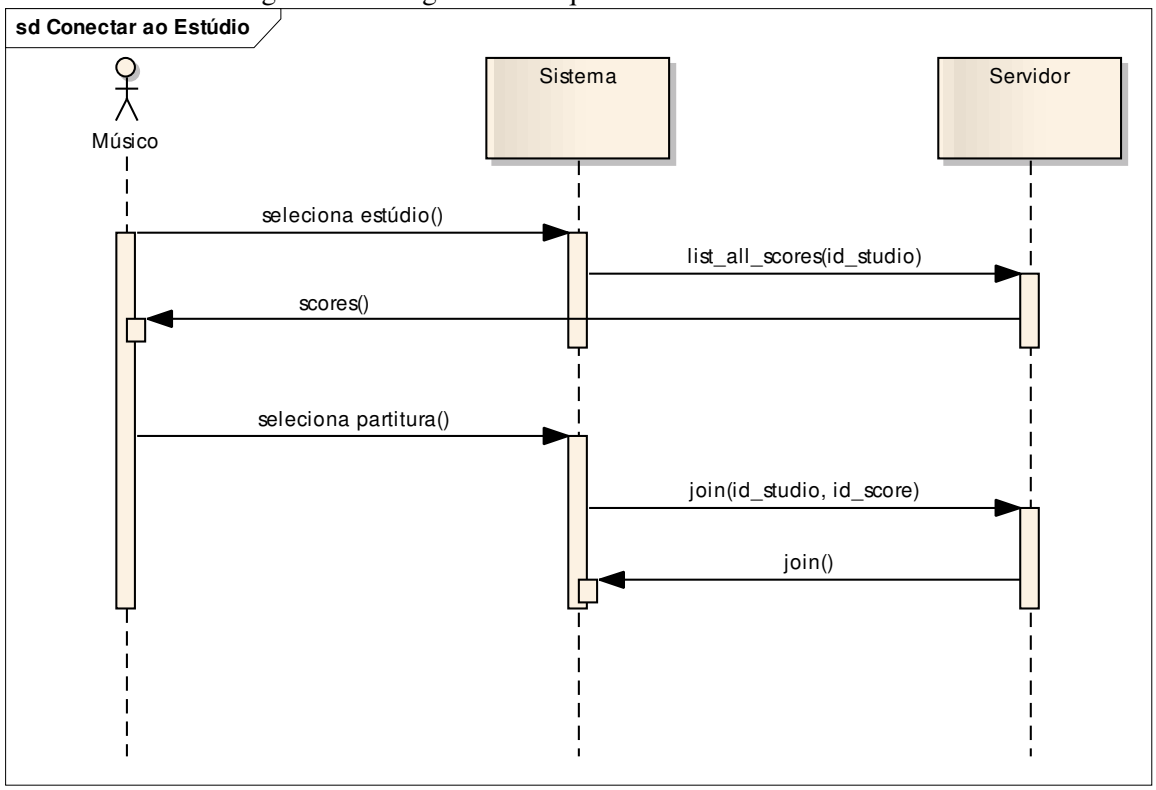
Um músico inicia a ação de carregar a partitura no protótipo, que a envia para o servidor através de uma mensagem com o comando `send_score` com o nome e conteúdo, `name` e `content`, respectivamente. O servidor retorna uma mensagem informando que a partitura foi enviada com sucesso ou não.

O músico também pode criar um estúdio. Para isso o protótipo envia uma mensagem ao servidor com o comando `create`, enviando o nome (`name`), a velocidade (`speed`) e o horário de início em milissegundos (`start`). Como ao criar um estúdio é necessário associar partituras para execução, o protótipo solicita ao servidor todas as partituras que foram cadastradas através do comando `list_all_scores`. O servidor responde enviando uma mensagem com todas as partituras, que são apresentadas ao usuário.

O músico então pode escolher entre as partituras retornadas pelo servidor as que ele deseja incluir no estúdio que está sendo criado. Ao selecionar uma partitura, o protótipo envia ao servidor uma mensagem com o comando `add_score`, o identificador do estúdio e o da partitura (`id_studio` e `id_score`, respectivamente).

Na Figura 14 consta o fluxo de atividade que o `Músico` executa para entrar em um estúdio. O músico escolhe a partitura referente à parte da música que ele deseja tocar, recebe o arquivo para leitura, o tempo restante para o início da execução e aguarda a contagem regressiva do protótipo, que dá início à execução da música.

Figura 14 – Diagrama de sequência de conexão ao estúdio



Para escolher uma música e agendar a sua execução, é necessário que o `Músico` acesse um estúdio, selecionando-o no sistema. Ao selecionar, o protótipo envia uma mensagem ao servidor com o comando `list_all_scores` e o identificador do estúdio (`id_estudio`). O servidor retorna para o usuário a lista de partituras escolhidas para aquele estúdio selecionado.

O `Músico` então seleciona uma partitura entre as que foram retornadas pelo servidor. Para receber as informações da partitura, o protótipo envia ao servidor uma mensagem com o comando `join`, o identificador do estúdio e o da partitura (`id_studio` e `id_score`, respectivamente). O servidor retorna uma mensagem com as informações da partitura e estúdio escolhidos e o protótipo agenda a sua execução.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas pesquisadas e utilizadas, detalhando algumas das principais rotinas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

Para o desenvolvimento foram utilizadas as linguagens Java e Javascript, recursos do Android *Software Development Kit* (SDK), do Cordova e da biblioteca `VexFlow`. A *Integrated Development Environment* (IDE) escolhida foi o Eclipse com ADT plugin, que permite o desenvolvimento de aplicações Android. O servidor utilizado foi o Apache Tomcat 7.0.55.

Para definir a comunicação entre os dispositivos, alguns métodos foram estudados e analisados. Primeiramente, as técnicas estudadas foram o Bluetooth e o WiFi-Direct. Após a optar pela utilização do Cordova, porém, foi necessário buscar outra forma de comunicação, já que não existem APIs Cordova disponíveis para utilização de WiFi-Direct, conforme Figura 5 já citada.

Foi cogitada a possibilidade de criar um *plugin* para acesso a essa funcionalidade nativa do WiFi-Direct, porém esta implementação seria inviável devido ao tempo disponível. Em algumas pesquisas, foram encontradas algumas APIs de terceiros para comunicação Bluetooth, porém com pouca documentação. Considerando que o protótipo é feito em HTML5, abandou-se a ideia de interação sem a necessidade de um servidor, optando-se pela tecnologia de *WebSocket*. Com esta tecnologia também é possível controlar de forma simples o sincronismo entre os dispositivos, necessária ao reproduzir a música em grupo.

Para desenhar a partitura na tela do dispositivo, foram pesquisados alguns *plugins*. O JFugue foi uma opção. Esta API Java de código livre permite criação e edição de músicas, interpretação e conversão de arquivos MIDI, *parsing* de arquivos `MusicXML`, entre outras funcionalidades, conforme Koelle (2012). Em sua página, Kolle sugere a API LilyPond para renderizar partituras musicais. Foram encontrados poucos exemplos para desenvolvimento de aplicativos Android com JFugue e pouca documentação.

Outra opção encontrada para o desenho foi o `VexFlow`. Esta API escrita em JavaScript retorna um HTML com a partitura e tem várias funcionalidades disponíveis, entre elas criação e edição de músicas, reprodução de áudio de alguns instrumentos, inclusão de articulações nas partituras e *parsing* de arquivos `MusicXML` através de um *plugin* desenvolvido com base no `VexFlow` (um *branch* do projeto principal). Utilizando Cordova/PhoneGap é possível renderizar a partitura na tela, conforme consta na seção de *Frequently Asked Questions* (FAQ) da API no GitHub, The VexFlow FAQ (2014). Por ter mais documentação e trabalhar com ferramentas já conhecidas, esta foi a API escolhida para o desenvolvimento do trabalho.

Para a interface do protótipo foi utilizado o JQuery Mobile, após algumas pesquisas e testes. O JQuery Mobile tem uma documentação extensa e bastante detalhada, além de ser simples e de rápido desenvolvimento.

3.3.2 Etapas da implementação

A seguir será feito o detalhamento de cada uma das implementações, que compreendem e contemplam a utilização do *Websocket*, o tratamento das requisições feitas via *WebSocket* ao servidor, o envio de dados para o servidor, o sincronismo e a leitura das partituras a partir do `MusicXML`.

3.3.2.1 Utilização do *WebSocket*

Para utilizar o *WebSocket* foi utilizada uma API para Android disponibilizada no GitHub por Knowledgecode (2014). Esta API permite a utilização das funcionalidades descritas na seção 2.2 com o mesmo código. O Quadro 7 contém trecho do código da conexão com o servidor.

Quadro 7 – Comunicação com *WebSocket*

```
1  document.addEventListener("deviceready", init, false);
2  var wsUri = "ws://campeche.inf.furb.br:8094/easyscore/websocket/easy";
3  function init() {
4      testWebSocket();
5  };
6
7   function testWebSocket() {
8       websocket = new WebSocket(wsUri);
9       websocket.onopen = function(evt) {
10          onOpen(evt)
11      };
12      websocket.onclose = function(evt) {
13          onClose(evt)
14      };
15      websocket.onmessage = function(evt) {
16          onMessage(evt)
17      };
18      websocket.onerror = function(evt) {
19          onError(evt)
20      };
21  };
```

O *plugin* foi adicionado ao projeto utilizando o comando `cordova plugin add https://github.com/knowledgecode/WebSocket-for-Android.git` com o Node.js, citado na seção 2.3.3. O código `new WebSocket(wsUri)`, na linha 8, abre a conexão definindo a URL do servidor onde a aplicação está hospedada, que no protótipo desenvolvido é: `ws://campeche.inf.furb.br:8094/easyscore/websocket/easy`. A função `websocket.onopen` que aparece na linha 9 é disparada ao abrir a conexão via *WebSocket*. A

função `websocket.onclose`, definida na linha 12, é disparada ao encerrar a conexão. Na linha 15 está definida a função `websocket.onmessage`, que é disparada quando o cliente recebe uma mensagem via *WebSocket*. A linha 18 define a função `websocket.onerror`, que é disparada ao ocorrer um erro no *WebSocket*.

3.3.2.2 Tratamento das requisições feitas via *WebSocket* ao servidor

A implementação do lado do servidor foi feita em Java. O Quadro 8 contém um trecho dessa implementação. O método `onMessage` recebe os parâmetros da classe `StudioHandler` na linha 2, verifica quais são as sessões que devem ser notificadas com o comando `handler.handleRequest(session.getId(),msg)` na linhas de 5 a 7 e na linha 6 chama o método `sendMessage` para cada sessão mapeada.

Quadro 8 – Notificação dos usuários

```
1  @OnMessage
2  public void onMessage(Session session, String msg, boolean last) {
3     Map<String, String> response =
4           handler.handleRequest(session.getId(), msg);
5     for (String sessionId : response.keySet()) {
6           sendMessage(sessionId, response, last);
7     }
8  }
9  public void sendMessage(String sessionId, Map<String, String> response,
10 boolean last) {
11    Session session = sessions.get(sessionId);
12    try {
13          if (session.isOpen()) {
14                session.getBasicRemote().sendText(
15                      response.get(sessionId), last);
16          }
17    } catch (IOException e) {
18          try {
19                session.close();
20          } catch (IOException ie) {
21          }
22    }
23 }
```

As requisições devem retornar somente para um dispositivo, portanto o `sendMessage`, definido na linha 9, é chamado somente uma vez. Na linha 11 o método pega a sessão do usuário. Posteriormente o método verifica se a seção está aberta na linha 13, com o comando `session.isOpen()` e então envia os retornos nas linhas 14 e 15 com o comando `session.getBasicRemote().sendText`.

Ao receber uma mensagem, o servidor a interpreta utilizando o método `handleRequest`, que consta no Quadro 9.

Quadro 9 – Tratamento das mensagens recebidas pelo servidor

```
1 public Map<String, String> handleRequest(String sessionId, String msg) {
2     Map<String, String> content = new HashMap<String, String>();
3     String returningMessage = "";
4     try {
5           JsonObject json = Json.createReader(new StringReader(msg))
6                       .readObject();
7           switch (MessageTypes.valueOf(json.getString(MESSAGE_TYPE)
8                       .toUpperCase())) {
9           case CREATE:
10                returningMessage = this.createStudio(sessionId, json);
11                content.put(sessionId, returningMessage);
12                return content;
13          case JOIN:
14                returningMessage = this.joinStudio(json);
15                content.put(sessionId, returningMessage);
16                return content;
17          case LIST:
18                returningMessage = this.listStudios(sessionId, json);
19                content.put(sessionId, returningMessage);
20                return content;
21          case SEND_SCORE:
22                returningMessage = this.createScore(sessionId, json);
23                content.put(sessionId, returningMessage);
24                return content;
25  //aqui irão constar todas as opções de mensagem recebida pelo servidor
26          default:
27                throw new EasyScoreException();
28          }
29    } catch (JsonParsingException | EasyScoreException jpe) {
30          returningMessage = Json.createObjectBuilder().add("message",
31                      "Formato Json Inválido").build().toString();
32          content.put(sessionId, returningMessage);
33          return content;
34    }
35 }
```

Nas linhas 5 e 6 o servidor lê a mensagem recebida. Entre as linhas 7 e 28 ele verifica qual comando foi recebido e o trata. Cada mensagem recebida pelo servidor é tratada por esse método.

3.3.2.3 Envio de dados para o servidor

Para enviar os dados para o servidor, o cliente envia mensagens via *WebSocket*. A conexão é aberta conforme o código do Quadro 7, citado anteriormente. Todos os métodos para envio de mensagens montam uma variável com o texto a ser enviado e depois chamam o método `doSend` que envia a mensagem para o servidor.

O Quadro 10 contém os métodos de criação de partitura, criação de estúdio, atribuição de partitura a um estúdio, e o método que envia a mensagem para o servidor.

Quadro 10 – Métodos de envio de dados para o servidor

```
1  function sendScore(nome, score) {
2     var vsendScore = '{"type": "send_score", "values": [{ \
3                         "name": "' + nome + '",\
4                         "content": "' + score + '"}]}';
5     doSend(vsendScore);
6     document.getElementById("nome_part").value = "";
7     document.getElementById("arquivo").value = "";
8  };
9  function createStudio(name, start, speed) {
10    var create = '{"type": "create","values": [{"name": "' + name +
11                '","start": "' + start + '", "speed":' + speed+ '}]}';
12    doSend(create);
13    document.getElementById('criaEstStart').value = "";
14    document.getElementById('criaEstName').value = "";
15  };
16  function doSend(message) {
17    websocket.send(message);
18  };
```

Na linha 1 inicia-se o trecho de código que envia a solicitação de criação de partitura ao servidor, com a função `sendScore`. O cliente passa como parâmetro o nome e o conteúdo lido do arquivo no dispositivo. Na linha 5 o cliente chama a função que envia a mensagem para o servidor e nas linhas 6 e 7 ele limpa os campos da tela onde o usuário criou o estúdio.

A função `createStudio`, definida na linha 10, é chamada ao pressionar o botão `Criar` na tela de criação de estúdio. Esta função recebe como parâmetro o nome (`nome`), o horário de início (`start`) e a velocidade (`speed`) que o usuário informou na tela. As linhas 14 e 15 estão apagando da tela os valores informados pelo usuário, que já foram enviados para o servidor.

Ao receber a mensagem, o servidor interpreta os comandos recebidos, conforme o código já citado no Quadro 9, e chama os métodos correspondentes. O trecho de código com os métodos de criação de estúdio e criação de partitura estão no Quadro 11. Após a criação, tanto o estúdio quanto a partitura são criados e adicionados a listas no servidor.

Quadro 11 – Métodos de criação de estúdio e de partitura no servidor

```
1  private String createScore(String sessionId, JsonObject json) {
2     JsonArray studioValues = json.getJsonArray("values");
3     String name = studioValues.getJsonObject(0).getString("name");
4     String content = studioValues.getJsonObject(0).getString("content");
5     Score score = new Score(getNextScoreId(), name, content);
6     scores.add(score);
7
8     return addMessage(Json.createObjectBuilder(), "OK").toString();
9  }
10 public String createStudio(String sessionId, JsonObject studioJson) {
11    JsonArray studioValues = studioJson.getJsonArray("values");
12    Studio studio = new Studio();
13    studio.setId(getNextId());
14    studio.setName(studioValues.getJsonObject(0).getString("name"));
15    String start = studioValues.getJsonObject(0).getString("start");
16    double dou = Double.parseDouble(start);
17    long lo = (long) dou;
18    studio.setStart(lo);
19    studio.setSpeed(studioValues.getJsonObject(0).getInt("speed"));
20    studios.add(studio);
21    JsonObjectBuilder responseBuilder = Json.createObjectBuilder();
22    responseBuilder.add("studio", createStudioObject(studio));
23
24    return addMessage(responseBuilder, "OK").toString();
25 }
```

As linhas de 1 a 9 estão definindo o método `createScore`, que é chamado quando o servidor recebe a mensagem do cliente solicitando a criação de uma partitura. Nas linhas 2, 3 e 4 o servidor está pegando os valores do `nome e conteúdo da partitura` contidos na mensagem recebida. A linha 5 cria um novo objeto `Score` com os valores recebidos e na linha 6 essa partitura `Score` é adicionada em uma lista com todas as partituras cadastradas no servidor. O retorno ocorre na linha 7, informando que a partitura foi criada com sucesso.

A criação do estúdio está definida entre as linhas 10 e 25. A linha 11 instancia uma variável com os valores enviados pelo protótipo. A linha 12 cria um novo estúdio, objeto `Studio`, e nas próximas linhas são definidos os atributos do objeto: `identificador`, `nome`, `horário de início` e `velocidade`.

O horário de início é recebido em milissegundos, portanto são necessárias algumas conversões. A variável é recebida em formato de texto, pois seu tamanho é maior do que o máximo de uma variável `Integer`. O texto recebido é transformado em um `Double` na linha 16 e depois em `Long` na linha 17 para ser salvo como valor de horário de início do estúdio.

Após criado o estúdio, este é adicionado a uma lista com os estúdios criados, na linha 20. A linha 22 adiciona as informações do estúdio criado na mensagem de retorno, que é enviada na linha 24.

3.3.2.4 Sincronismo

O sincronismo da execução é garantido ao determinar que todos os dispositivos iniciem a música ao mesmo tempo. Ao criar um estúdio, o usuário informa um horário de início, que será enviado ao servidor e armazenado no objeto incluído na lista de estúdios, conforme explicado na seção anterior.

Para consultar os estúdios disponíveis, o cliente envia ao servidor uma mensagem solicitando as informações dos estúdios. O servidor retorna a lista dos estúdios disponíveis e suas informações, incluindo `nome` e `identificador`. O cliente recebe essas informações pela função `websocket.onmessage`, citada na seção 3.3.2.1, e cria dinamicamente uma lista com os estúdios, conforme o trecho do Quadro 12.

Quadro 12 – Inclusão dos estúdios na lista

```
1 if (obj.hasOwnProperty('studios')) {
2     $("#listaEstudios").children("li").remove();
3     if (obj.studios.length <= 0) {
4           $('<li>').text("Não há estúdios disponíveis")
5                               .prependTo("#listaEstudios");
6     }
7     for (var i = 0; i<obj.studios.length; i++) {
8           if (obj.studios[i].name != "") {
9                 $('#listaEstudios').append($('<li/>', {
10                      //cria um item com as informações do estúdio
11                }));
12          }
13    }
14 }
```

Esse trecho da função `onMessage` verifica que o cliente recebeu os estúdios disponíveis na linha 1. Entre as linhas 7 e 12, o cliente verifica os estúdios que foram recebidos e os inclui em uma lista no protótipo.

Ao clicar em um estúdio, o cliente envia ao servidor uma mensagem solicitando as partituras atribuídas ao estúdio selecionado. O servidor retorna uma lista das partituras escolhidas para o estúdio com todas as informações da partitura, incluindo o identificador da partitura, nome e conteúdo. O cliente recebe essa mensagem e cria dinamicamente uma lista com as partituras recebidas, através do trecho de código no Quadro 13.

Quadro 13 – Inclusão das partituras na lista

```
1 if (obj.hasOwnProperty('scores')) {
2     $("#listaPart").children("li").remove();
3     if (obj.scores.length <= 0) {
4           $('#listaPart').append($('<li/>', {
5                 'text' : "Não há partituras disponíveis"}));
6     } else {
7           for (var i = 0; i<obj.scores.length; i++) {
8                 if (obj.scores[i].name != "") {
9                       $('#listaPart').append($('<li/>', {
10                            //cria um item com as informações do estúdio
11                      }));
12                }
13          }
14    }
15 }
```

Esse trecho da função `onMessage` verifica na linha 1 que o cliente recebeu as partituras. Entre as linhas 7 e 13, o cliente verifica as partituras que foram recebidas e inclui elas em uma lista no protótipo.

O usuário então seleciona uma partitura clicando sobre um item na lista que lhe foi apresentada, e o cliente chama a função `joinStudio`, que consta no Quadro 14, passando o identificador do estúdio e o da partitura.

Quadro 14 – Função que solicita as informações de determinada partitura

```
1 function joinStudio(idStudio, idScore) {
2     var joinGroup = '{"type": "join","values": [{"studio_id": ' +
3                               idStudio + ',"score_id":'+ idScore + '}]}';
4     doSend(joinGroup);
5 };
```

Quando um músico solicitar o recebimento das informações da partitura, o servidor interpreta a mensagem e chama o método `joinStudio`, que calcula o tempo que falta para o início da execução que foi agendada. A diferença de tempo é retornada para o cliente, que inicia uma contagem regressiva para o início da execução. Essa funcionalidade é implementada no método apresentado no Quadro 15.

Quadro 15 – Método para recebimento das informações da partitura escolhida

```
1 private String joinStudio(JsonObject json) {
2     JsonArray studioValues = json.getJsonArray("values");
3     Integer studioId=studioValues.getJsonObject(0).getInt("studio_id");
4     Integer scoreId = studioValues.getJsonObject(0).getInt("score_id");
5     Studio studio = getStudioById(studioId);
6     Score score = getScoreById(scoreId);
7     if (studio != null) {
8           JsonObjectBuilder objBuilder = Json.createObjectBuilder();
9           objBuilder.add("studio_id",studioId.toString());
10          objBuilder.add("studio_vel",studio.getSpeed());
11          objBuilder.add("score_id",scoreId.toString());
12          objBuilder.add("score_name",score.getName());
13          objBuilder.add("score_content",score.getContent());
14          Calendar calendar = Calendar.getInstance();
15          objBuilder.add("tempo", String.valueOf(studio.getStart() -
16                      calendar.getTimeInMillis()));
17          JsonArrayBuilder arrayBuilder = Json.createArrayBuilder();
18          arrayBuilder.add(objBuilder);
19          return addMessage(arrayBuilder.build(), "join",
20                "OK").toString();
21    } else {
22          return addMessage(Json.createObjectBuilder(),
23                      "Estúdio não encontrado").toString();
24    }
25 }
```

As linhas de 2 a 6 pegam as informações recebidas do servidor através da mensagem e instanciam os objetos de estúdio e partitura com os identificadores recebidos. Nas linhas de 8 a 13 são incluídas no retorno as informações do estúdio e da partitura: `identificadores`, `velocidade de execução`, `nome` e `conteúdo da partitura`.

O cálculo do tempo que falta para iniciar a execução é feito subtraindo o horário do servidor do horário de início da música, ambos em milissegundos. Para obter o horário do servidor, na linha 14 é instanciado um objeto `Calendar`. As linhas 15 e 16 incluem no retorno do método essa subtração utilizando o comando `objBuilder.add("tempo", String.valueOf(studio.getStart() - calendar.getTimeInMillis()))`. Nas linhas de 17 a 20 o servidor cria a mensagem com todas as informações incluídas anteriormente e retorna.

Os milissegundos restantes para iniciar a execução da música são retornados para o cliente, e através do trecho de código que consta no Quadro 16, o cliente inicia uma contagem regressiva para o início da execução. Este trecho está contido na função `onMessage`, citada na seção 3.3.2.1.

Quadro 16 – Inicia contagem regressiva no protótipo

```
1 if (obj.hasOwnProperty('join')) {
2     timestartvar = setTimeout(function(){
3           document.location.href='#exibePart';
4           velPart = obj.join[0].studio_vel;
5           partitura_lida = obj.join[0].score_content;
6           setDoc(partitura_lida);
7           drawLine();
8           var nBlocks = doc.getNumberOfMeasures() * 100;
9           document.getElementById('exibe_info').scrollLeft = -500;
10          scrollBlocksIntoView(nBlocks, velPart);
11    }, obj.join[0].tempo);
12    document.getElementById('btn_pause').style.display="block";
13    document.getElementById('btn_continue').style.display="none";
14    alert("Execução agendada");
25 }
```

Caso a mensagem recebida pelo cliente tenha o comando `join` e o argumento da linha 1 seja verdadeiro, o cliente identifica que o servidor está retornando as informações para agendamento da partitura. Na linha 2 é inicializada uma variável com a contagem regressiva através do comando `setTimeout`. O primeiro parâmetro recebido pelo comando é a função que deve executar ao finalizar a contagem regressiva. O tempo, em milissegundos, que inicia a contagem é definido na linha 11, como segundo parâmetro da função.

3.3.2.5 Representação visual das partituras

Conforme já mencionado na seção 3.3.1, a biblioteca `VexFlow` foi utilizada para fazer a leitura de arquivos `MusicXML`. Para construir a representação visual das partituras recebidas pelo cliente, é instanciado um documento `Vex.Flow.Document` passando como parâmetro em sua construção a `String` com o documento `MusicXML`. Depois de criado o documento, é criado o documento com formatação `Vex.Flow.DocumentFormatter`. Este documento define o tamanho do `canvas` e a escala em que o elemento irá aparecer. O Quadro 17 mostra o trecho de código da ferramenta que recebe o `MusicXML`, interpreta e desenha a partitura na tela.

Quadro 17 – *Parsing* do `MusicXML`

```
1     var doc = new Vex.Flow.Document(docString);
2     var formatter = doc.getFormatter();
3     formatter.zoom = escPart;
4     formatter.setWidth(largPart).draw($("#" + container)[0]);
```

A linha 1 recebe como parâmetro `docString`, que é o texto do arquivo `MusicXML` retornado do servidor. Ela cria um documento `VexFlow`. Na linha 2 é criado o documento de formatação do `VexFlow`. Na terceira linha é alterada a escala das imagens carregadas no `canvas`. É esta a linha que faz a partitura aumentar ou diminuir o tamanho. A linha 4 desenha o documento criado na tela.

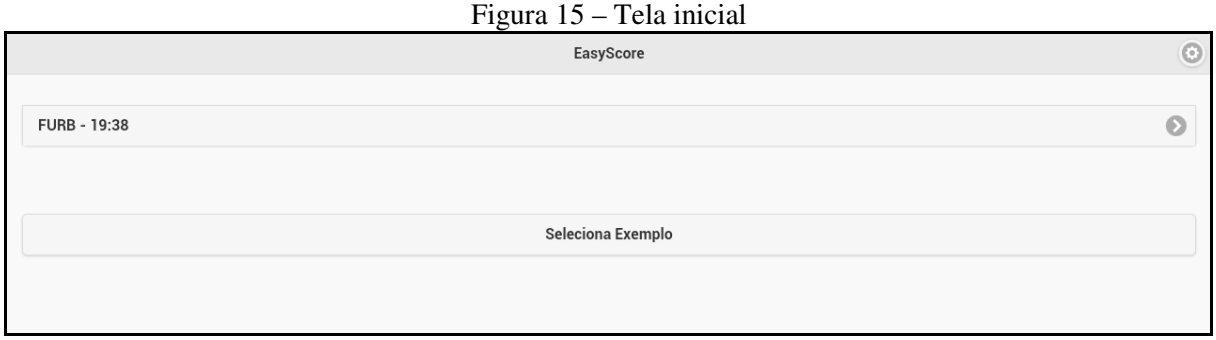
Conforme consta na seção 2.3.2, a biblioteca `VexFlow` tem várias funcionalidades, tais como: criar, editar e renderizar uma partitura a partir de um arquivo `musicXML`. Entretanto, ao criar o documento `Vex.Flow.Document`, utilizando a biblioteca criada por Ringwalt (2014), não é possível fazer o destaque das notas. Por este motivo, a ferramenta utilizada não possui uma funcionalidade que permita ao usuário destacar ou não a nota que deve ser reproduzida no momento.

3.3.3 Operacionalidade da implementação

Esta seção apresenta a operacionalidade da implementação em nível de usuário, mostrando as principais funcionalidades da ferramenta para Android. As imagens da execução foram obtidas com o dispositivo Samsung GT-P5110, *tablet* com a versão 4.1.2 (*Jelly Bean*) do Android. O protótipo não possui *login*.

Ao acessar a aplicação é apresentada a tela inicial, conforme a Figura 15, com uma lista de estúdios que já foram criados no servidor e estão disponíveis.

Figura 15 – Tela inicial



No canto superior direito da tela, pelo ícone no formato de engrenagem, o usuário pode acessar o menu do protótipo. Ao clicar no ícone, o menu aparecerá no lado esquerdo da tela, com as opções `Consultar Estúdios`, `Carregar Partitura`, `Criar Estúdio`, `Configurações` e `Exemplos de Partituras`, conforme Figura 16.

Figura 16 – Menu da ferramenta

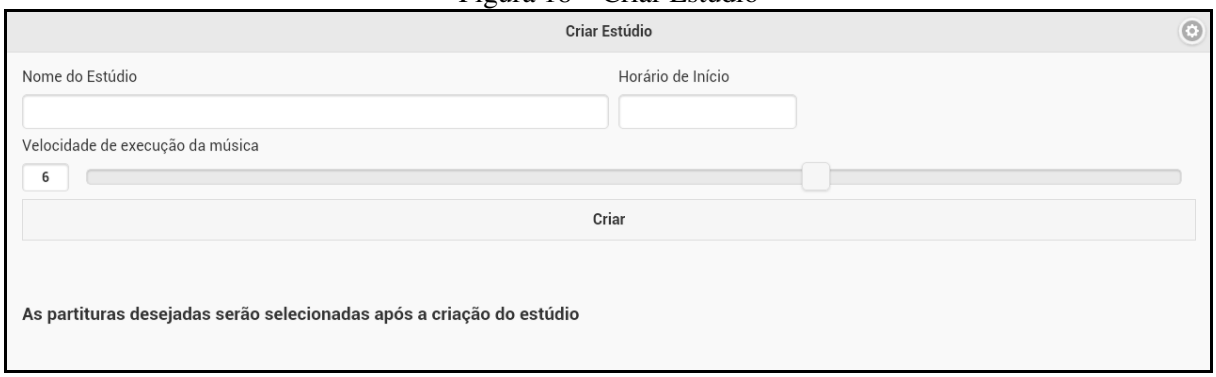
Para executar alguma música, é necessário que conste alguma partitura no servidor. Para fazer o *upload*, ou armazenar a música no servidor, o usuário deve acessar a opção `Carregar Partitura` no menu que se encontra no lado direito da tela.

Nesta página, o usuário deve informar o nome da partitura no campo `nome da partitura`, o caminho onde está o arquivo no campo `caminho do arquivo` e pressionar `Enviar`. Esta tela pode ser visualizada na Figura 17.

Figura 17 – Carregar Partitura

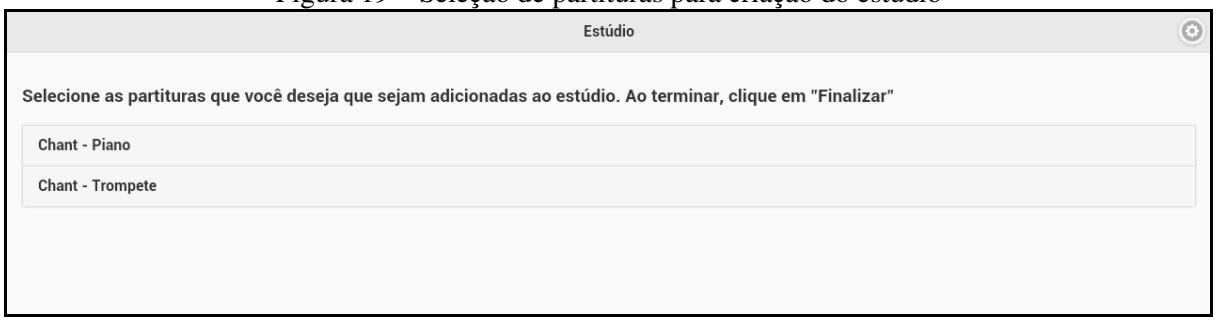
Após carregar uma partitura o usuário pode criar um estúdio selecionando a opção `Criar Estúdio` no menu, conforme Figura 18. Ao selecionar essa opção, o protótipo apresenta uma página onde o usuário pode preencher no campo `Nome do Estúdio` o nome escolhido e no campo `Horário de Início` o horário para iniciar a execução da partitura. No *range* do campo `Velocidade de execução da música` pode ser alterada a velocidade do *scroll* no momento da reprodução da partitura. Após definidas essas informações, ao clicar em `Criar` o protótipo cria um estúdio com o nome informado.

Figura 18 – Criar Estúdio



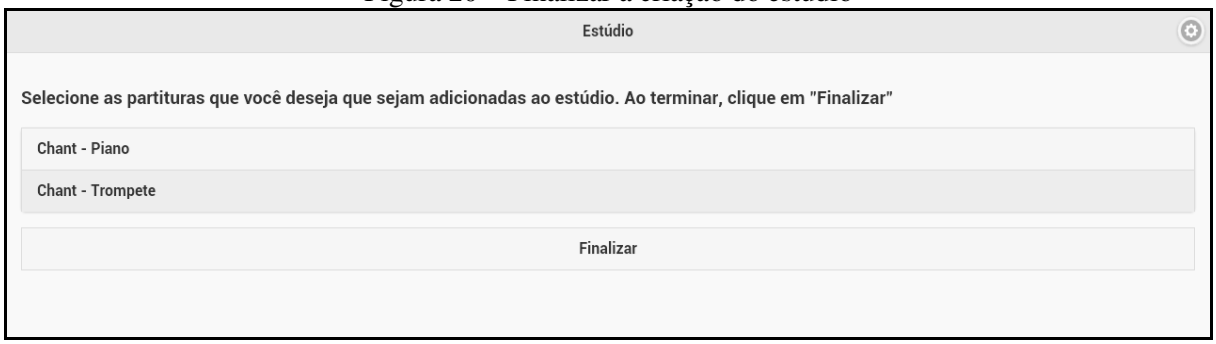
Ao pressionar o botão `Criar`, o usuário será redirecionado para uma tela para seleção das partituras que devem ser incluídas no estúdio, conforme Figura 19.

Figura 19 – Seleção de partituras para criação do estúdio



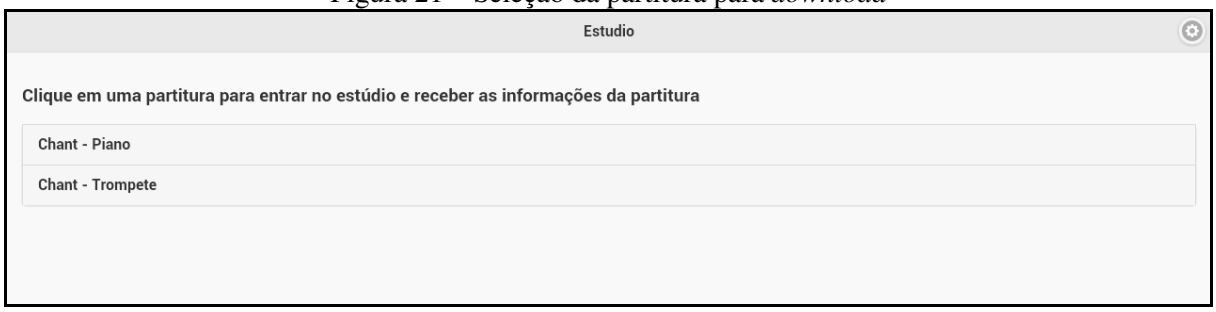
É necessário selecionar ao menos uma partitura para finalizar a criação do estúdio. Após selecionar todas as partituras desejadas, clique no botão `Finalizar` que aparecerá na tela após a seleção da primeira partitura, conforme a Figura 20.

Figura 20 – Finalizar a criação do estúdio



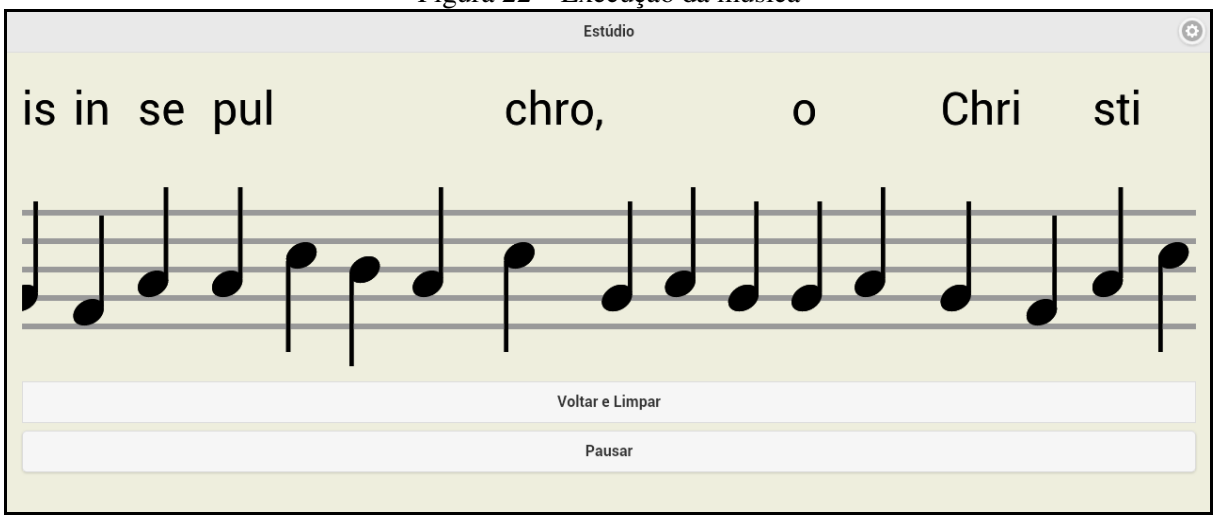
Para entrar em um estúdio, o usuário pode acessar a tela inicial ou selecionar no menu a opção `Consultar Estúdios`, correspondente à Figura 15. Ao selecionar um estúdio, o protótipo apresenta a tela apresentada na Figura 21. Ao clicar em qualquer partitura, o protótipo receberá as informações dessa partitura e a execução será agendada.

Figura 21 – Seleção da partitura para *download*



No horário em que a música foi agendada para iniciar, conforme a diferença de tempo recebida pelo servidor, a contagem regressiva do protótipo acaba e inicia a execução da partitura, conforme Figura 22.

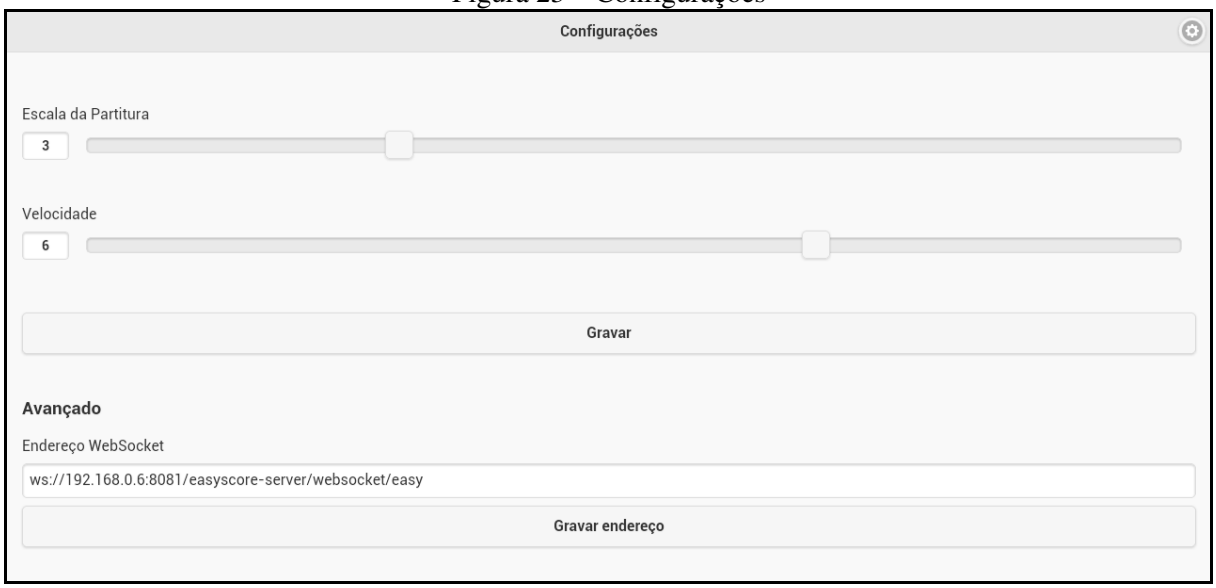
Figura 22 – Execução da música



Nesta tela ainda é possível pausar a execução da música, com o botão `Pausar` e continuar a execução com o botão `Continuar` que aparecerá ao parar a execução. O botão `Voltar e Limpar` pode ser utilizado para voltar à tela inicial do protótipo.

Na tela de configurações, através da opção `Configurações` no menu, é possível alterar o tamanho da partitura que vai aparecer na tela, além da velocidade em que os exemplos de partitura serão reproduzidos. Para isso, o usuário informa os valores nos campos indicados e clica em `Gravar`, conforme Figura 23.

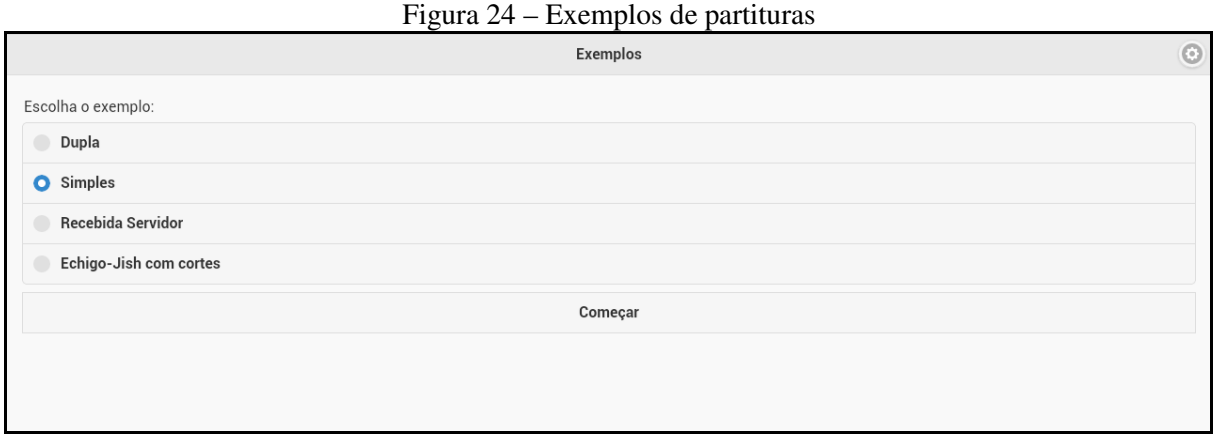
Figura 23 – Configurações



Na mesma tela ainda é possível alterar o caminho do *WebSocket* que está sendo acessado, informando o novo caminho no campo `Endereço WebSocket` e clicando em `Gravar endereço`.

Foram disponibilizadas algumas partituras para testes e para a ambientação do usuário com o protótipo. Estas partituras podem ser acessadas pelo menu através da opção `Exemplos de Partituras`. A mesma tela também pode ser acessada clicando no botão `Seleciona Exemplo` na página inicial, conforme Figura 15. Ao acessar a página de exemplos, o protótipo apresentará a tela apresentada na Figura 24. Para iniciar a execução, basta clicar no botão `Começar`.

Figura 24 – Exemplos de partituras



3.4 RESULTADOS E DISCUSSÃO

Nesta seção são apresentados os experimentos feitos com o protótipo. Na seção 3.4.1, detalha-se a avaliação do protótipo, o perfil dos usuários, a aplicação dos testes, o questionário de usabilidade, a análise e interpretação dos dados coletados. A seção 3.4.2

apresenta os testes de sincronismo. A seção 3.4.3 mostra os resultados obtidos nos testes de compatibilidade e, por fim, a seção 3.4.4 faz a comparação e discussão entre o protótipo desenvolvido e os trabalhos correlatos.

3.4.1 Experimento de usabilidade

O experimento de usabilidade foi realizado com nove usuários para avaliar a eficiência e a facilidade de utilização das funcionalidades implementadas no protótipo.

3.4.1.1 Metodologia

O experimento aconteceu durante o mês de novembro por meio de teste individual com os usuários. Para realizar os testes foram disponibilizados dois *tablets*, GT-P5110 e SM-T110, com sistema operacional Android nas versões 4.1.2 e 4.2.2 respectivamente. Nestes *tablets* também foram disponibilizados exemplos de partituras no arquivo `MusicXML` dentro de uma pasta EasyScore criada na raiz dos dispositivos. Foi fornecido a cada usuário um questionário de perfil, uma lista de tarefas e um questionário de usabilidade, que estão disponíveis no Apêndice C.

3.4.1.2 Aplicação do teste

Para iniciar a avaliação, os participantes foram orientados a preencher um questionário de perfil de usuário. Após preenchido o questionário, cada voluntário foi orientado sobre o objetivo dos testes de usabilidade e sobre o objetivo do protótipo. A lista de tarefas foi composta por seis tarefas e buscou contemplar todas as funcionalidades implementadas e disponíveis no protótipo. Ao finalizar cada tarefa, foi solicitado que o usuário informasse se a tarefa foi executada, se foi identificado algum problema ou se havia alguma observação a fazer.

O questionário de usabilidade foi composto por seis perguntas, cinco delas fechadas e uma aberta. Para quatro das questões fechadas foi disponibilizada a opção de colocar uma observação sobre a resposta escolhida. As perguntas procuraram obter as impressões do usuário sobre o protótipo, sobre a facilidade de utilização e a importância das funcionalidades disponíveis. Ao responder o questionário os voluntários também puderam deixar suas reclamações e/ou sugestões sobre o protótipo. Os resultados deste experimento são apresentados na próxima seção.

3.4.1.3 Análise e interpretação dos dados coletados

A primeira análise realizada foi a dos dados coletados através do questionário de perfil de usuário. No

Quadro 18 são exibidos os perfis dos usuários envolvidos no teste de usabilidade.

Quadro 18 – Perfil dos usuários envolvidos no teste de usabilidade

Sexo	67% masculino 33% feminino
Idade	11% menos de 18 anos 67% entre 18 e 25 anos 22% mais de 35 anos
Nível de escolaridade	11% ensino médio completo – 2º grau 67% ensino superior incompleto 22% ensino superior completo
Relação com a música	33% escuta música casualmente 22% é apreciador de música, consegue identificar ritmos e instrumentos 45% é músico amador, sabe tocar um instrumento

Com as informações de perfil dos usuários voluntários para os testes, prosseguiu-se para a avaliação dos resultados obtidos com a lista de tarefas.

3.4.1.3.1 Análise dos resultados da lista de tarefas

No Quadro 19 são apresentados resultados quanto às questões envolvendo o cenário do aplicativo.

Quadro 19 – Respostas quanto à lista de tarefas

Perguntas/Respostas	Sim	Não
1. Realize o cadastro de uma partitura, escolhendo qualquer arquivo `MusicXML` que conste no dispositivo	100%	
2. Realize o cadastro de um estúdio, informando nome, horário de início e velocidade da execução	100%	
3. Associe as partituras desejadas ao estúdio criado, selecionando-as na lista de partituras disponíveis no servidor	100%	
4. Consulte os estúdios disponíveis	100%	
5. Acesse um estúdio e escolha uma partitura para reprodução	100%	
6. Acompanhe a execução da partitura. Foi possível visualizar todos os compassos sem manipular o aparelho, a partir do momento de início de reprodução da partitura?	100%	

Considerando os resultados das questões da lista de tarefas exibidas no Quadro 19, percebeu-se que não houve grande dificuldade para execução das tarefas solicitadas. Além das

questões objetivas, o questionário também continha espaço para anotar as suas percepções sobre a tarefa solicitada. As observações feitas nesta parte do experimento apontaram alguns problemas como a instabilidade da conexão com o servidor e sugerem alterações para facilitar a visualização dos arquivos `MusicXML` disponíveis no dispositivo.

A instabilidade na conexão ocorre devido a falhas de conexão dos dispositivos móveis com a rede da universidade, onde a aplicação foi disponibilizada. Para a visualização dos arquivos, alguns usuários sugerem a inclusão de uma lista de arquivos disponíveis. Esta solução não foi implementada devido ao tempo limitado disponível, porém é possível incluir esta funcionalidade utilizando o mesmo *plugin* que já foi adicionado ao projeto para ler os arquivos do dispositivo.

Um dos voluntários colocou como observação o fato das descrições no menu não serem muito claras, como o item `Carregar Partitura`, que envia uma partitura do dispositivo para o servidor. Outra sugestão foi incluir uma mensagem informando que a execução das partituras somente ocorrerá após o horário informado na criação do estúdio.

Uma última observação feita por um dos voluntários foi a de que a velocidade máxima para execução da partitura ainda é baixa, já que existem músicas que são reproduzidas numa velocidade maior do que a velocidade máxima disponível pelo protótipo.

3.4.1.3.2 Análise quanto à usabilidade

Após analisados os resultados da lista de tarefas e perfil dos usuários, foram analisados os resultados obtidos do questionário de usabilidade aplicado. Estes resultados constam no Quadro 20.

Quadro 20 – Respostas quanto à usabilidade do protótipo

Perguntas	Respostas
1. Das tarefas solicitadas, quantas você conseguiu executar?	100% Todas
2. De um modo geral, você achou o protótipo intuitivo e fácil de usar?	100% sim
3. Você achou fácil visualizar as notas da forma que a partitura foi apresentada?	100% concorda totalmente
4. Você considera importante que o aplicativo tenha uma funcionalidade para sincronizar o início da reprodução da partitura?	78% concorda totalmente 11% concorda parcialmente 11% é indiferente
5. Qual é a sua avaliação do protótipo?	55% muito bom 45% bom

O resultado da primeira questão indica que é possível executar todas as funcionalidades testadas no experimento. Apesar das observações feitas na seção anterior, em

que foram apresentadas reclamações e sugestões para a melhoria do protótipo, todos os usuários consideraram o protótipo intuitivo e fácil de usar.

Nas observações da terceira questão, um voluntário comentou que seria interessante que as chaves (onde aparecem os acidentes, bemol e sustenido) aparecessem durante toda a execução. Atualmente essas informações aparecem somente no começo. A implementação dessa sugestão evitaria que o músico esquecesse essas informações no meio da execução. A quarta questão teve um comentário que observa que a importância do sincronismo no início da reprodução se dá somente ao reproduzir músicas em grupo.

Além dos resultados obtidos pelas questões fechadas, foram coletadas opiniões sobre o protótipo por meio de uma questão aberta sobre as dificuldades do protótipo. As maiores dificuldades identificadas pelos usuários foram a queda de conexão com o servidor, a busca do arquivo para carregar a partitura e fazer seu cadastro, a falta de uma página de ajuda, com instruções sobre a utilização do protótipo, o desconhecimento de partituras e uma interação baixa com o usuário, dificultando a utilização do protótipo por um usuário leigo.

Na questão cinco, que avalia o protótipo, houve um comentário dizendo que o protótipo é simples, mas promissor: “Resolve alguns problemas do usuário e torna sua atividade muito mais simples do que seria sem o aplicativo”.

Ainda em relação à usabilidade do protótipo, é importante ressaltar que a partir de testes unitários, verificou-se que a velocidade máxima que pode ser utilizada da forma que o protótipo foi implementado é a de 8.5. Após isso a partitura não faz mais a rolagem, ou *scroll*, da tela corretamente.

3.4.2 Experimento de sincronismo

Para avaliar o sincronismo dos dispositivos ao reproduzir as partituras, foram feitos dois experimentos com os dispositivos GT-P5110, SM-T110 e LG-E977, com as versões de Android 4.1.2, 4.2.2 e 4.1.2 respectivamente. No primeiro experimento o objetivo era verificar se a execução das partituras começaria ao mesmo tempo nos três dispositivos a partir de um mesmo estúdio. No segundo experimento, o objetivo era constatar se o protótipo permitia o uso/execução de vários estúdios ao mesmo tempo.

Para o primeiro experimento foi criado um estúdio que os três dispositivos acessaram, cada um escolhendo uma partitura. Os dispositivos tiveram uma diferença de até três segundos para iniciar a execução. O primeiro dispositivo iniciou no horário previsto, o segundo teve um atraso de um segundo e o terceiro atrasou três segundos.

Esta diferença de tempo para o início da execução se deve ao atraso do recebimento da resposta pelo cliente, que pode ser diferente para cada aparelho. Tal problema não foi tratado pois demandaria mais pesquisa e tempo.

No segundo experimento foram criados dois estúdios com um minuto de diferença entre os horários de início de execução. Um dispositivo acessou um dos estúdios e os dois outros dispositivos acessaram outro com horário de início de um minuto depois. Os dispositivos iniciaram a execução sem problemas, cada um no tempo definido pelo estúdio.

3.4.3 Experimento de compatibilidade

O protótipo foi instalado e experimentado em três dispositivos Android, visando testar a compatibilidade do protótipo em diferentes *hardwares* e versões do Android. Os dispositivos utilizados foram apresentados no Quadro 21.

Quadro 21 – Dispositivos utilizados no experimento de compatibilidade

Dispositivo	Versão do Android	Tamanho da tela
Samsung Galaxy Tab GT-P7300	4.0.4	8.9”
Samsung Galaxy Tab 2 GT-P5110	4.1.2	10.1”
Samsung Galaxy Tab 3 Lite SM-T110	4.2.2	7”
LG-E977	4.1.2	4,7”

No dispositivo GT-P7300, o *scroll* da partitura não funciona pois o método utilizado para esta finalidade não é compatível com a versão do Android. No dispositivo LG-E977, ao criar um estúdio, não foi possível informar o horário de início, pois o celular não abriu o teclado. Excetuando essas questões, todas as outras funcionalidades puderam ser executadas sem problemas. Nos outros dispositivos todas as funcionalidades foram concluídas.

A visualização das partituras naturalmente fica mais clara numa escala maior. Esta escala pode ser alterada na tela de `Configurações` (ver Figura 23), porém as partituras com duas pautas na maior escala possível, ultrapassam o tamanho da tela nos aparelhos com 7” e 4,7”. Na tela de 4,7”, a partitura simples ficou descentralizada. Quando o celular está na horizontal, ainda é possível ver a última linha na pauta, porém ela fica muito próxima do final. O ideal é fazer um *scroll* vertical no início da execução para visualizar a partitura completa.

3.4.4 Comparação com trabalhos correlatos e discussões

Após a avaliação dos resultados obtidos com os experimentos, o quadro de comparação dos trabalhos correlatos foi alterado, incluindo o protótipo EasyScore para comparação. O Quadro 22 descreve as diferenças e semelhanças entre o protótipo EasyScore e os trabalhos correlatos.

Quadro 22 – Comparação do protótipo com os trabalhos correlatos

características/ trabalhos	Alvarenga (2013)	Dancing Dots (2014)	Vaidyanathan (2013)	Akoumianakis et al. (2008)	Paludo (2014)
Plataforma	Android	Desktop	Desktop	Desktop	Android
Arquitetura	Stand-alone	Stand-alone	Stand-alone	Cliente-Servidor	Cliente-Servidor
apresenta partitura	Não	Sim	Sim	Não	Sim
interação entre dispositivos	Não	Não	Não	Sim	Sim
acompanhamento da música	O tempo da *timeline* é destacado	Sim	Sim	Sim	Sim
permite importar ou exportar músicas em `MusicXML`	Não	Sim	Não	Não	Sim

Comparando o protótipo aos trabalhos correlatos, é possível verificar que o diferencial do EasyScore é o uso da plataforma Android e na interação com os dispositivos, focando na leitura de partituras.

O `VexFlow` lê corretamente tanto arquivos simples quanto bastante complexos, com articulações e símbolos não muito comuns. Conforme Ringwalt (2014) a API foi desenvolvida para ajustar dinamicamente a música ao alterar o tamanho da tela, o que torna difícil a definição de medidas para incluir cursores para acompanhamento das notas. Para facilitar o ajuste, a API pode ser adaptada para utilizar um Espaço Normalizado (NDC).

Em outras bibliotecas encontradas sem suporte a `MusicXML`, funcionalidades para destaque das notas estão disponíveis ou são mais facilmente implementadas. O trabalho correlato *Midi Sheet Music* citado na seção 2.4.3, por exemplo, possibilita ao usuário o destaque da nota a ser reproduzida, além de mostrar as notas a serem tocadas no teclado.

Foi possível fazer a comunicação entre os dispositivos móveis, porém o sincronismo na execução da partitura ainda pode ser melhorado, estudando de técnicas de sincronização de relógios. A utilização de um servidor tornou a comunicação entre os dispositivos simples, sem necessidade de muitos outros recursos. Foi necessária somente a inclusão de um *plugin* no Cordova para utilização do *WebSocket* na aplicação e um servidor.

A utilização da plataforma Cordova auxiliou tanto na utilização da API do `VexFlow` quanto para a utilização do *WebSocket*. A possibilidade de utilização de ferramentas web para o desenvolvimento torna possível a utilização de várias APIs que não são disponíveis na linguagem nativa do Android, sem deixar de acessar importantes funcionalidades nativas. No protótipo desenvolvido foi utilizado um *plugin* para leitura de arquivos do dispositivo para carregar a partitura para o servidor, por exemplo. O desenvolvimento para outras plataformas

torna-se mais fácil e a plataforma também dispensa o conhecimento da linguagem nativa dos sistemas operacionais.

4 CONCLUSÕES

Os objetivos principais do trabalho, que eram facilitar a leitura e o acompanhamento em grupos de partituras foram alcançados. O protótipo possibilitou a leitura e acompanhamento de partituras musicais em grupo.

O objetivo de desenhar a partitura a partir do arquivo `MusicXML` foi atendido pela biblioteca `VexFlow`. Através dos testes de usabilidade é possível afirmar que a leitura das partituras foi facilitada com o protótipo, porém ainda poderiam ser criados cursores que acompanhassem a execução da partitura ou então algum destaque nas notas que possibilitasse visualizá-las de forma mais fácil.

A possibilidade de aumentar ou diminuir o tamanho da partitura apresentada na tela é outro objetivo que foi alcançado e um benefício para o usuário. Assim ele pode alterar a visualização da partitura conforme sua preferência, deixando ela maior e visualizando menos partes ou deixando ela menor e visualizando mais informações do resto da música. Esta funcionalidade também permite que o usuário adapte a escala da partitura de acordo com o tamanho da sua tela, para que a partitura inteira apareça.

Conforme os resultados dos experimentos e dos testes de usabilidade foi comprovado que é possível visualizar todas as notas de uma partitura sem manipulação do aparelho após a execução ter iniciado. Este foi outro objetivo atingido através de uma contagem regressiva com o tempo definido pelo usuário.

A alteração de velocidade com que a partitura passa na tela é interessante para o músico que está tocando a música sozinho e quer aumentar ou diminuir o ritmo de execução. Nos estúdios, porém, todos os músicos devem estar sincronizados tocando as notas ao mesmo tempo. Por esse motivo, foi incluída a informação da velocidade na criação do estúdio. Essa informação é recebida pelo protótipo e definida ao final da contagem regressiva, antes do início da execução.

Com a utilização de *WebSockets* e o servidor fazendo o cálculo da diferença de tempo do início da partitura menos o tempo atual do servidor, foi possível criar um sincronismo. Apesar do sucesso na implementação desta forma de sincronismo, poderiam ter sido verificadas formas de tratar os segundos de atraso que acabam acontecendo devido ao tempo de recebimento da resposta pelo cliente, para cada dispositivo. Este tratamento pode ser realizado aplicando uma técnica para determinar o atraso do recebimento da resposta.

Conforme as pesquisas realizadas, existem poucos aplicativos hoje que possibilitam a leitura de partituras em dispositivos móveis. O número cai ainda mais ao pesquisar sobre

aplicativos que tenham também alguma forma de comunicação com outros dispositivos. Alguns músicos conhecidos foram questionados e constatou-se que vários deles não têm conhecimento de aplicativos para leitura de partituras em dispositivos móveis. A implementação de um trabalho com os objetivos propostos aqui é válida diante da necessidade identificada pelos profissionais que atuam na área.

4.1 EXTENSÕES

Algumas das extensões possíveis para este trabalho são:

a) adaptar os códigos para disponibilizar o aplicativo para outros sistemas operacionais;
b) implementar de uma funcionalidade de reprodução do áudio do arquivo lido;
c) possibilitar o envio de mensagens com mais de 8.192 caracteres para o servidor, tornando possível o armazenamento de músicas mais extensas e com mais detalhes;
d) criar e possibilitar o uso de um cursor ou alguma forma de destacar cada nota que está sendo reproduzida conforme a música é executada;
e) controlar o acesso ao aplicativo;
f) utilizar uma técnica mais adequada para o sincronismo entre os dispositivos.

REFERÊNCIAS

ADOBE. **PhoneGap, Cordova, and what's in a name?** [S.l.], [2012]. Disponível em: <http://phonegap.com/2012/03/19/phonegap-cordova-and-what%E2%80%99s-in-a-name/>. Acesso em: 30 abr. 2014.

AKOUMIANAKIS, Demosthenes et al. **Towards the implementation of a generic platform for networked music performance:** the diamouses approach. 2008. 8f. Technological Educational Institute of Crete, Creta. Disponível em: <http://www.inf.ufrgs.br/~afhoppe/dodidx.pdf>. Acesso em: 30 out. 2013.

ALVARENGA, Gustavo G. **Ferramenta para criação de composições musicais para o Android.** 2013. 59 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://campeche.inf.furb.br/tccs/2013-I/TCC2013-1-16-VF-GustavoGAlvarenga.pdf>. Acesso em: 10 set. 2013.

AMBROS, Luisa. **Diferença entre aplicativos nativos, híbridos e mobile web apps**. [S.l.], [2013]. Disponível em: <http://luisaambros.com/blog/diferenca-entre-aplicativos-nativos-hibridos-e-mobile-web-apps/>. Acesso em: 02 jun. 2014.

APACHE. **About Apache Cordova**. [S.l.], [2013]. Disponível em: <http://cordova.apache.org/>. Acesso em: 30 maio 2014.

APACHE. **Apache cordova documentation**: platform support. Version 3.5.0. [S.l.], [2014a]. Disponível em: <http://cordova.apache.org/docs/en/3.5.0/guide_support_index.md.html#Platform%20Support>. Acesso em: 30 maio 2014.

APACHE. **Apache cordova documentation**: the command-line interface. Version 3.5.0. [S.l.], [2014b]. Disponível em: <http://cordova.apache.org/docs/en/3.5.0/guide_cli_index.md.html#The%20Command-Line%20Interface>. Acesso em: 26 abr. 2014.

BRANDÃO, Douglas. **Descubra os principais aplicativos para músicos existentes no mercado**. [S.l.], [2013]. Disponível em: <http://www.techtudo.com.br/noticias/noticia/2013/04/descubra-os-principais-aplicativos-para-musicos-existentes-no-mercado.html>. Acesso em: 02 nov. 2013.

CAMPANINI, André. **WebSocket no Java e Java EE:** da especificação a exemplos. [S.l.], [2012]. Disponível em: <http://www.infoq.com/br/articles/websocket-java-javaee>. Acesso em: 15 jun. 2014.

CENTRO DE ARTES E EDUCAÇÃO FÍSICA. **Conjuntos instrumentais**. [Porto Alegre], [2010]. Disponível em: <http://prolicenmus.ufrgs.br/repositorio/moodle/material_didatico/conjuntos_musicais_escolares/turma_abc/un01/conj_mus_esc_un01_conteudo.pdf>. Acesso em: 16 jul. 2014.

CLAYTON, Martin; SAGER, Rebecca; WILL, Udo. In time with the music: The concept of entrainment and its significance for ethnomusicology. **ESEM CounterPoint**, [S.l.], v. 1, p. 1 – 82, 2004. Disponível em: <http://www.open.ac.uk/Arts/experience/InTimeWithTheMusic.pdf>. Acesso em: 21 nov. 2014.

DANCING DOTS. **Goodfeel Braille music translator**. Valley Forge, [2014]. Disponível em: <http://www.dancingdots.com/main/goodfeel.htm>. Acesso em: 01 nov. 2014.

FETTE, Ian; MELNIKOV, Alexey. **The WebSocket Protocol RFC 6455**. [S.l.], [2011]. Disponível em: <http://www.rfc-editor.org/rfc/pdfrfc/rfc6455.txt.pdf>. Acesso em: 01 jun. 2014.

GANDELMAN, Salomea. **Breve história da notação musical.** [S.l.], [2012]. Disponível em: <http://musicaeadoracao.com.br/26041/breve-historia-da-notacao-musical/>. Acesso em: 16 set. 2013.

GURURANI, Sachin. **How is Wi-Fi Direct better than Bluetooth 4.0 for sending and receiving files?** [S.l.], [2014]. Disponível em: <http://gadgetstouse.com/gadget-tech/wi-fi-direct-vs-bluetooth-4-0/18555>. Acesso em: 20 mai. 2014.

HEILMAN, Kenneth M. et al. (1986) The right hemisphere neuropsychological functions. **Journal of Neurosurgery**, Department of Neurology, J. Hillis Miller Health Center, University of Florida, Gainesville, Florida, v. 64, n. 5, p. 693-704. maio 1986.

IBOPE. **Internet móvel avança no Brasil**. [S.l.], [2013]. Disponível em: <http://www.ibope.com.br/pt-br/conhecimento/artigospapers/Paginas/Internet-movel-avan%C3%A7a-no-Brasil.aspx>. Acesso em: 02 nov. 2013.

IDC – PRESS RELEASE. **Apple cedes market share in smartphone operating system market as Android surges and windows phone gains, according to IDC.** [S.l.], [2013]. Disponível em: <http://www.idc.com/getdoc.jsp?containerId=prUS24257413>. Acesso em: 17 set. 2013.

JOYENT. **Node.js**. [S.l.], [2014]. Disponível em: <http://nodejs.org/>. Acesso em: 22 abr. 2014.

JORDÃO, Fabio. **Tecmundo explica**: qual a diferença de Bluetooth para WiFi? [S.l.], [2013]. Disponível em: <http://www.tecmundo.com.br/wi-fi/44835-tecmundo-explica-qual-a-diferenca-de-bluetooth-para-wifi-.htm>. Acesso em: 20 maio 2014.

KAAZING COORPORATION. **About HTML5 WebSockets**. [S.l.], [2013]. Disponível em: <http://www.websocket.org/aboutwebsocket.html>. Acesso em: 01 jun. 2014.

KNOWLEDGECODE. **WebSocket for Android**. [S.l.], [2014]. Disponível em: <https://github.com/knowledgecode/WebSocket-for-Android>. Acesso em: 01 maio 2014.

KOELLE, David. **JFugue**: features. [S.l.], [2012]. Disponível em: <http://www.jfugue.org/features.html>. Acesso em: 11 abr. 2014.

MAKEMUSIC. **What is MusicXML?** [S.l.], [2013a]. Disponível em: <http://www.musicxml.com>. Acesso em: 14 set. 2013.

_____. **Hello world.** [S.l.], [2013b]. Disponível em: <http://www.musicxml.com/tutorial/hello-world>. Acesso em: 15 set. 2013.

_____. **Alphabetical index**. [S.l.], [2014]. Disponível em: <http://www.musicxml.com/for-developers/alphabetical-index/>. Acesso em: 10 maio 2014.

MARKETWATCH. **Apple iPad still no. 1, but market share shrinks.** [S.l.], [2013]. Disponível em: <http://www.marketwatch.com/story/apple-ipad-still-no-1-but-market-share-shrinks-2013-08-05-121032945>. Acesso em: 17 set. 2013.

PRIETO, Romain. **PhoneGap:** a misunderstood hybrid framework. [S.l.], [2012]. Disponível em: <http://www.asyncdev.net/2012/10/phonegap-a-misunderstood-hybrid-framework/>. Acesso em: 20 maio 2014.

REZENDE, Gabriel S. S. L. Música, experiência e memória: algumas considerações sobre o desenvolvimento da partitura a partir das obras de Max Weber e Walter Benjamin. **Revista Espaço Acadêmico**, [S.l.], v. 8, n. 85, jun. 2008. Disponível em: <http://www.espacoacademico.com.br/085/85rezende.htm>. Acesso em: 16 set. 2013.

RINGWALT, Dan. **MusicXML for VexFlow**. [S.l.], [2014]. Disponível em: <http://ringwa.lt/projects/musicxml/>. Acesso em: 8 abr. 2014.

SCHULZ, Sabrina L. **Acústico e eletroacústico:** a sincronia entre o piano e os sons pré-gravados em obras eletroacústicas mistas. 2010. 119 f. Dissertação (Mestre em Música) – Departamento de Artes, Universidade Federal do Paraná, Curitiba. Disponível em: <http://www.artes.ufpr.br/musica/mestrado/dissertacoes/2010/schulzme.pdf>. Acesso em: 20 nov. 2014.

SIMON, Marcos F. **Ferramenta para geração de partituras no sistema de musicografia braile.** 2012. 65 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://dsc.inf.furb.br/arquivos/tccs/monografias/TCC2012-2-20-VF-MarcosFSimon.pdf>. Acesso em: 15 out. 2014.

The VexFlow tutorial. [S.l.], [2014]. Disponível em: <http://www.vexflow.com/docs/tutorial.html>. Acesso em: 08 maio 2013.

The VexFlow FAQ. [S.l.], [2014]. Disponível em: <https://github.com/0xfe/vexflow/wiki/The-VexFlow-FAQ>. Acesso em: 31 mar. 2014.

VAIDYANATHAN, Madhav. **Midi Sheet Music.** [S.l.], [2013]. Disponível em: <http://midisheetmusic.sourceforge.net/index.html>. Acesso em: 15 set. 2013.

WEINRE. **Home**. [S.l.], [2012]. Disponível em: <http://people.apache.org/~pmuellr/weinre/docs/latest/Home.html>. Acesso em: 17 maio 2014.

APÊNDICE A – Exemplos de Tags utilizadas no `MusicXML`

Neste apêndice consta uma lista de tags que são bastante utilizadas no `MusicXML`.

Quadro 23 – *Tags* do `MusicXML`

`attributes`	Informação musical que normalmente muda nos limites do compasso
`Beats`	Quantidade de tempo para cada compasso (numerador)
`beat-type`	Define a figura (nota) que preenche um tempo (denominador)
`Clef`	Tipo de Clave (sol, dó, fá)
`direction`	Repete a direção. Valores: *backward* (para trás), *forward* (para frente)
`direction-type`	Tipo da direção, pode estar combinado
`Fifths`	Número de bemois e sustenidos em uma chave (key) tradicional
`Key`	Associação de sustenidos ou bemóis representados junto à clave (chave)
`Line`	Linha da pauta, numerada de baixo para cima
`measure`	Compasso
`notations`	Notação musical para uma nota
`Note`	Nota individual ou pausa
`octave`	Número de oitavas. Valor: de 0 a 9
`score-part`	Como uma parte é definida na partitura
`score-partwise`	Elemento do documento de uma partitura *partwise*
`Sound`	Define se um *offset* afeta o *playback*. Valores: *yes* (sim), *no* (não)
`Staff`	Atribuição de pauta para múltiplas pautas (sistema)
`Stem`	Direção da haste. Valores: *down* (para baixo), *up* (para cima), *none* (nenhuma) e *double* (dupla).
`Time`	Tempo
`Type`	Tipo do compasso. Valores: *start* (início), *stop* (parada)
`Voice`	Distingue a voz musical

Fonte: MakeMusic (2014).

APÊNDICE B – Detalhamento dos casos de uso

A seguir é apresentado o detalhamento dos casos de uso, com uma descrição, pré e pós condições e cenário.

O caso de uso `UC01 – Configurar escala da partitura` é utilizado para configurar o tamanho do desenho gerado pelo `VexFlow`. A alteração da escala na tela de configurações altera a escala da imagem gerada tanto na execução dos exemplos quanto na execução agendada pelo servidor. O detalhamento está descrito no Quadro 24.

Quadro 24 – Caso de uso `Configurar escala da partitura`

UC01 – Configurar escala da partitura	
Descrição	Configura a escala da partitura que será gerada pelo `VexFlow`
Pré-condições	Nenhuma
Cenário principal	1. No menu, através do ícone no formato de engrenagem situado no canto superior direito, o usuário seleciona a opção `Configurações`; 2. O usuário define no campo `Escala da partitura` a escala desejada para geração das imagens da partitura e clica em `Gravar`; 3. O protótipo salva as configurações.
Pós-condições	O protótipo apresenta uma mensagem informando que as configurações foram salvas.

No caso de uso `UC02 – Criar estúdio` o usuário cria um estúdio para execução da música. Mais detalhes constam no

Quadro 25.

Quadro 25 – Caso de uso `Criar estúdio`

UC02 - Criar estúdio	
Descrição	Cria um estúdio para execução de uma partitura no protótipo
Pré-condições	O usuário deve ter feito *upload* de pelo menos uma partitura
Cenário principal	1. O usuário acessa no menu, através do ícone no formato de engrenagem situado no canto superior direito, a opção `Criar Estúdio`; 2. O usuário preenche o nome do estúdio, o horário de início e a velocidade de execução e clica em `Criar`.
Pós-condições	O protótipo direciona o usuário para a tela de seleção de partituras

O caso de uso `UC03 – Definir horário, velocidade e músicas` atribui partituras ao estúdio criado, conforme detalhamento do Quadro 26.

Quadro 26 – Caso de uso `Definir horário, velocidade e músicas`

UC03 – Definir horário, velocidade e músicas	
Descrição	Seleciona as partituras que devem ser disponibilizadas no estúdio
Pré-condições	O usuário deve ter solicitado a criação de um estúdio
Cenário principal	1. O protótipo apresenta uma lista de partituras disponíveis no servidor; 2. O usuário seleciona as partituras desejadas; 3. O usuário clica em `Finalizar`; 4. O protótipo atribui as partituras informadas ao estúdio solicitado.
Pós-condições	Acesso à tela inicial do protótipo

O caso de uso `UC04 – Upload música` é necessário para enviar um arquivo de partitura ao servidor, para posterior execução. Mais detalhes foram apresentados no

Quadro 27.

Quadro 27 – Caso de uso `Upload música`

UC04 - Upload partitura	
Descrição	Faz o upload do arquivo `MusicXML` da partitura no servidor
Pré-condições	Nenhuma
Cenário principal	1. No menu, através do ícone no formato de engrenagem situado no canto superior direito, o usuário seleciona a opção `Carregar Partitura`; 2. O usuário informa um nome para a partitura no campo `nome da partitura` e no campo `caminho do arquivo`, o caminho em que foi gravado o arquivo `MusicXML` 3. O usuário clica em `Enviar`; 4. O servidor armazena as informações da partitura.
Pós-condições	Acesso à tela inicial do protótipo

O caso de uso `UC05 – Entrar em estúdio` associa o usuário ao estúdio solicitado, conforme detalhamento do Quadro 28.

Quadro 28 - Caso de uso `Entrar em estúdio`

UC05 - Entrar em estúdio	
Descrição	O servidor atribui o usuário a um estúdio
Pré-condições	O usuário deve ter consultado os estúdios
Cenário principal	1. O usuário seleciona o estúdio desejado; 2. O servidor retorna a lista de partituras atribuídas ao estúdio; 3. O protótipo apresenta a lista de partituras.
Pós-condições	O protótipo direciona o usuário para a tela de seleção de partituras

O caso de uso `UC06 – Selecionar partitura` faz o download da partitura escolhida, conforme detalhamento do Quadro 29.

Quadro 29 – Caso de uso `Selecionar partitura`

UC06 - Selecionar partitura	
Descrição	O usuário seleciona uma partitura disponível no estúdio e recebe todas as informações necessárias para execução do servidor.
Pré-condições	O usuário deve ter escolhido um estúdio
Cenário principal	1. O usuário seleciona a partitura desejada clicando em uma partitura na lista 2. O servidor retorna todas as informações da partitura escolhida: identificadores do estúdio e da partitura, nome da partitura, velocidade de execução, conteúdo da partitura com o arquivo `MusicXML` em forma de texto e o tempo em milissegundos que falta para o início da execução.
Pós-condições	Acesso à tela inicial, aguardando o início da execução.

O caso de uso `UC07 – Executar partitura` faz desenha a partitura escolhida, conforme detalhamento do Quadro 30.

Quadro 30 - Caso de uso `Executar partitura`

UC07 - Executar partitura	
Descrição	O sistema executa a partitura escolhida ao finalizar a contagem regressiva e faz o *scroll* com a velocidade definida na criação do estúdio.
Pré-condições	O usuário deve ter selecionado uma partitura
Cenário principal	1. O sistema agenda a execução da partitura com uma contagem regressiva 2. Ao final da contagem regressiva, a partitura é desenhada na tela e o *scroll* é iniciado, para que seja possível a visualização de todas as notas da partitura.
Pós-condições	Com o *scroll*, toda a partitura é mostrada na tela.

APÊNDICE C – Roteiro e questionário de avaliação de usabilidade

Neste apêndice constam o questionário e o roteiro de testes que os usuários seguiram. O questionário de perfil do usuário está no Quadro 31. O Quadro 32 contém a lista de tarefas que conduz o usuário, testando as funcionalidades do protótipo. No Quadro 33 consta o questionário de usabilidade.

Quadro 31 – Questionário de perfil de usuário

PERFIL DE USUÁRIO

Observação: as informações recebidas abaixo serão mantidas de forma confidencial.

Sexo: ( ) Masculino ( ) Feminino

Idade:
( ) Tenho menos de 18 anos
( ) Tenho entre 18 e 25 anos
( ) Tenho entre 25 e 35 anos
( ) Tenho mais de 35 anos

Nível de escolaridade:
( ) Ensino fundamental incompleto
( ) Ensino fundamental completo - 1º grau
( ) Ensino médio incompleto
( ) Ensino médio completo - 2º grau
( ) Ensino superior incompleto
( ) Ensino superior completo

Você possui algum dispositivo móvel (smartphone/tablet)?
( ) Sim ( ) Não

Indique seu grau de familiaridade com música:
( ) Sou leigo em música
( ) Escuto música casualmente
( ) Sou apreciador de música, consigo identificar ritmos e instrumentos
( ) Sou músico amador, sei tocar um instrumento
( ) Sou músico experiente, sei tocar um ou mais instrumentos e compor músicas para eles

Quadro 32 – Lista de tarefas

INSTRUÇÕES

Com este questionário buscamos avaliar a utilização do protótipo de aplicativo para leitura de partituras.

Um dos objetivos do protótipo é possibilitar a leitura de partituras em dispositivos móveis, sem a necessidade de manipulação do aparelho enquanto a música estiver sendo executada. Outro objetivo é disponibilizar a partitura para os usuários no mesmo momento, sincronizando o início da reprodução da partitura e facilitando o acompanhamento da música executada em grupo.

Você pode utilizar o protótipo livremente por um período de 5 a 10 minutos para se ambientar. Ao finalizar, solicitamos que prossiga nos testes conforme as orientações abaixo.

Lista de tarefas a serem executadas:

1) Realize o cadastro de uma partitura, escolhendo qualquer arquivo `MusicXML` que conste no dispositivo (foram disponibilizados arquivos dentro da pasta EasyScore dos dispositivos disponíveis para teste).

   A tarefa foi executada? ( ) Sim ( ) Não

Observação:_____
_____

2) Realize o cadastro de um estúdio, informando nome, horário de início e velocidade da execução.

   A tarefa foi executada? ( ) Sim ( ) Não

Observação:_____
_____

3) Associe as partituras desejadas ao estúdio criado, selecionando-as na lista de partituras disponíveis no servidor.

   A tarefa foi executada? ( ) Sim ( ) Não

Observação:_____
_____

4) Consulte os estúdios disponíveis.

   A tarefa foi executada? ( ) Sim ( ) Não

Observação:_____
_____

5) Acesse um estúdio e escolha uma partitura para reprodução.

   A tarefa foi executada? ( ) Sim ( ) Não

Observação:_____
_____

6) Acompanhe a execução da partitura. Foi possível visualizar todos os compassos sem manipular o aparelho, a partir do momento de início de reprodução da partitura?

A tarefa foi executada? ( ) Sim ( ) Não

Observação:_____
_____

Quadro 33 – Questionário de usabilidade

QUESTIONÁRIO DE USABILIDADE

1. Das tarefas solicitadas, quantas você conseguiu executar?
   ( ) Todas
   ( ) A maior parte delas
   ( ) Metade das tarefas
   ( ) Menos da metade das tarefas
   ( ) Nenhuma tarefa

2. De um modo geral, você achou o protótipo intuitivo e fácil de usar?

   ( ) Sim ( ) Não

3. Você achou fácil visualizar as notas da forma que a partitura foi apresentada?
   ( ) Concordo totalmente
   ( ) Concordo parcialmente
   ( ) Indiferente
   ( ) Discordo parcialmente
   ( ) Discordo totalmente

Observação:_____
_____

4. Você considera importante que o aplicativo tenha uma funcionalidade para sincronizar o início da reprodução da partitura?
   ( ) Concordo totalmente
   ( ) Concordo parcialmente
   ( ) Indiferente
   ( ) Discordo parcialmente
   ( ) Discordo totalmente

Observação:_____
_____

5. Qual é a sua avaliação do protótipo?

   ( ) Muito bom

   ( ) Bom

   ( ) Regular

   ( ) Insatisfatório

Observação:_____
_____

6. Qual foi a sua maior dificuldade utilizando o protótipo?

_____
_____

Muito obrigada pela participação!