UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea in

Scienze e Tecnologie della Comunicazione Musicale

Tesi di Laurea Triennale

Render di documenti IEEE 1599 tramite API VexFlow

Relatore:	Correlatore:	Candidato:
Prof. Luca Andrea Ludovico	**Adriano Baratè**	**Edoardo Ciotto**

Indice

1- Introduzione

Le tecnologie dell'informazione e della comunicazione (TIC) influenzano notevolmente la vita di ogni individuo: nel lavoro, nella fruizione di infor-mazioni utili e nella cultura in generale. Questo grazie anche alla sempre più ampia diffusione di dispositivi in grado di collegarsi in rete e condividere contenuti con tutto il mondo [1].

Il continuo incremento di utenza ha incentivato le diverse software house, gruppi di ricerca e laboratori informatici a creare utility e applicazioni sempre più user friendly per la condivisione, o semplicemente, la lettura di file multimediali direttamente dal proprio browser senza l'installazione di ulteriori software.

L'interesse principale del seguente elaborato è stato proprio quello di progettare uno script in grado di realizzare graficamente uno spartito musicale traendo le informazioni utili da un file IEEE1599-2008, implementabile in una qualsiasi pagina Web senza la richiesta di plug-in specifici ma con il semplice utilizzo di risorse open source.

Le seguenti pagine vogliono descrivere l'intero progetto suddividendolo nelle diverse fasi di realizzazione dello stesso. In particolar modo, il secondo capitolo descrive lo stato iniziale dell'arte, soffermandosi sulla formulazione dell'idea finale e la ricerca di tecnologie già esistenti in grado di fornire un modello concreto e di confronto.

L'analisi dell'ambiente di lavoro e delle risorse offerte hanno portato alla scelta delle tecnologie utili alla realizzazione del progetto e allo studio delle stesse, alle quali è stato dedicato il terzo capitolo riguardante le tecnologie utilizzate. In particolar modo viene presentato Vex Flow [2], un API open-source sviluppato da 0xfe [3], in grado di manipolare e realizzare graficamente uno spartito musicale sfruttando l'elemento canvas del nuovo linguaggio di markup HTML5.

Il quarto capitolo tratta la realizzazione e la stesura del codice del progetto finale suddividendolo in tre principali versioni del codice:

- la prima, in grado di stampare a video le informazioni principali ricavate tramite parsing da un file IEEE1599-2008;
- la seconda, in grado di mettere in relazione le informazioni ricavate dalla prima versione con l'API VexFlow, realizzando graficamente le diverse parti e voci di uno spartito musicale;
- la terza, quella conclusiva, che ha permesso di migliorare la gestione dell'impaginazione dello spartito e risolvere alcuni dei limiti imposti dall'API VexFlow, utilizzando VexTab [4], un linguaggio più semplice da gestire.

Nel corso della trattazione, inoltre, sono presenti porzioni di codice opportunamente commentate. Queste mostreranno le parti più salienti del codice allegato interamente alla fine dell'elaborato.

Il quinto capitolo, infine, vuole descrivere e concludere l'intero progetto mettendo a fuoco i limiti imposti dal codice e dal progetto e quali saranno gli sviluppi futuri dello stesso.

2 – Stato dell'arte e formulazione dell'idea finale

2.1- EMIPIU: Valorizzazione dei beni culturali musicali

L'interesse principale dello studente nel realizzare il seguente progetto nasce dal volere valorizzare ed ampliare l'utilizzo del formato IEEE1599-2008 in ambito Web.

In fase di analisi è venuto incontro il Laboratorio di Informatica Musicale (LIM) del Dipartimento di Informatica e Comunicazione (DICO) dell'Università degli Studi di Milano, il quale ha presentato EMIPIU [5], acronimo di Enhanced Music Interactive Platform for Internet User, un progetto di Cooperazione scientifica e tecnologica fra l'Università degli Studi di Milano, il LIM, l'Università francese di Montpellier 2 e il Laboratorio di Informatica Robotica e di Microelettronica di Montpellier, in collaborazione con Didael KTS - Knowledge Technologies Services [6], co-finanziato da Regione Lombardia.

Il presente progetto è finalizzato alla condivisione delle conoscenze e valorizzazione di beni culturali musicali, facendo riferimento a tutti i protagonisti delle attività musicali, come musei, grandi archivi pubblici e privati, teatri, ecc., che possono trarre nuovo impulso e meglio valorizzare le risorse disponibili.

Specificamente, si tratta di:

- progettare ed implementare strumenti per la creazione, gestione e fruizione di documenti musicali in formato IEEE 1599-2008;
- automatizzazione dei processi di produzione di materiali IEEE 1599 mediante ottimizzazione e ingegnerizzazione della piattaforma MX attuale;
- creazione di una teca digitale di file in formato IEEE 1599 accessibile via Web, contenenti partiture codificate in XML e associate a oggetti digitali testuali, audio, video e grafici;
- realizzazione di strumenti di visualizzazione della musica e plug-in per il Web. In particolare di un player evoluto per la riproduzione sincronizzata e interattiva dei contenuti musicali inclusi nel formato IEEE1599;

- porting multipiattaforma del framework in relazione ai differenti sistemi operativi e architetture presenti nel panorama scientifico e tecnologico;

EMIPIU dispone un portale internet con interfaccia multilingue presentando il proprio progetto ed offrendo una sessione d'archivio chiamata "Music box" nella quale è possibile trovare numerosi riferimenti a documenti IEEE1599 precedentemente inseriti nell'intera sezione.

Fig. 1- Portale Internet EMIPIU, sezione Music Box

Ad ogni documento è stata dedicata una pagina Web nella quale è possibile trovare in primo luogo, una sezione informativa e descrittiva del brano musicale di riferimento, successivamente, la sezione dedicata interamente al player di sincronizzazione e riproduzioni dei contenuti presenti nel documento IEEE1599.

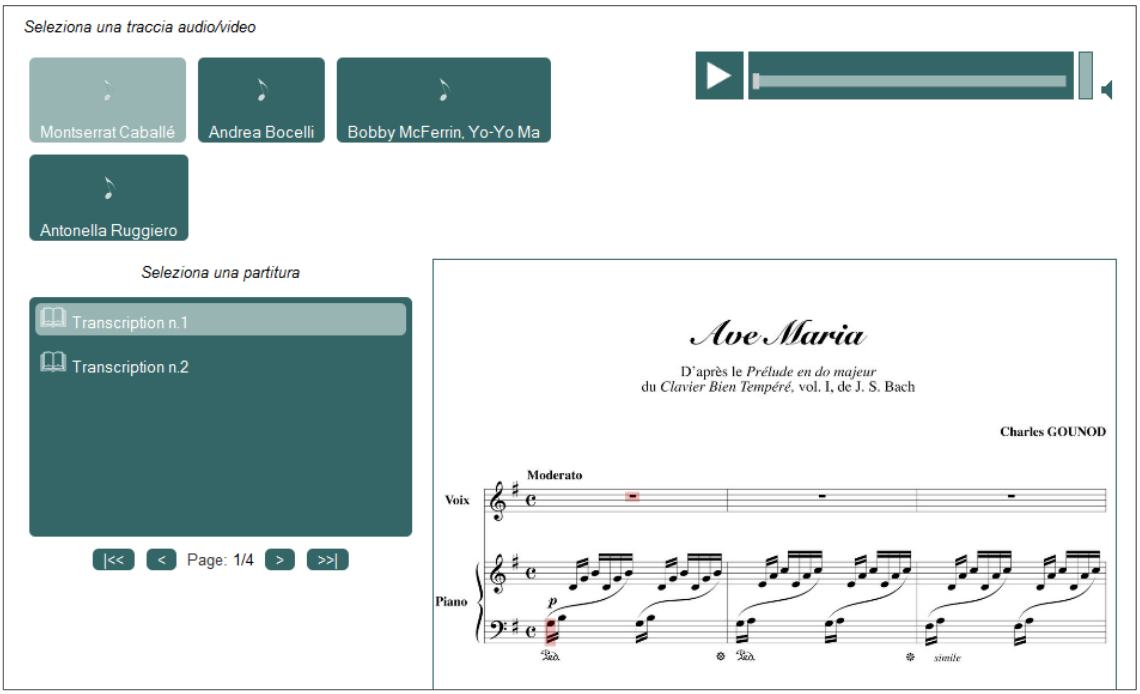


Fig. 2– Esempio sezione player documento: Ave Maria di Johann Sebastian Bach.

Come illustrato in **Fig. 2**, il player offre diverse funzionalità quali: selezione di una traccia audio/video e la riproduzione della stessa, selezione di una delle partiture disponibili e la loro visualizzazione sulla sinistra della pagina.

Tutti gli elementi disponibili nel player sono tutti gli elementi inclusi nel file IEEE1599, in particolar modo, le partiture sono immagine acquisite da pagine in formato cartaceo o create tramite software di notazione musicale. Questo legame tra documenti IEEE1599 e immagini impone di avere un file immagine di riferimento per la sincronizzazione degli eventi musicali e il conseguente suo caricamento nel player.

Nel caso di studio, sono state formulate diverse idee portando lo studente alla scelta di realizzare un render grafico di partiture musicali utilizzando unicamente le informazioni di un documento IEEE1599.

2.2- Render grafico di documenti IEEE1599: Analisi del panorama tecnologico

Il mercato dei software di notazione musicale offre una vasta scelta per tutti i sistemi operativi e tipologie di utenti: per chi è alle prime armi o per chi vuole utilizzare il proprio software a livello professionale. Negli anni, inoltre, sono nati software in grado di interagire con i propri strumenti MIDI in tempo reale migliorando la loro sensibilità e velocizzando la trascrizioni di partiture in documenti informatici[7].

Lo stesso non può essere affermato per software e applicazioni di notazione musicale per il Web, infatti, sono state trovate poche risorse disponibili e poca documentazione a riguardo.

In prima analisi sono stati presi in considerazione le applicazioni di Noteflight e VexFlow.

2.2.1 – Noteflight: "Your music, everywhere"

Il primo, è un software per il Web creato dal gruppo di lavoro Noteflight LLC di Boston, in Massachusetts, realizzato nel 2007 allo scopo di reinventare il modo con cui le persone creano, scrivono e condividono musica.

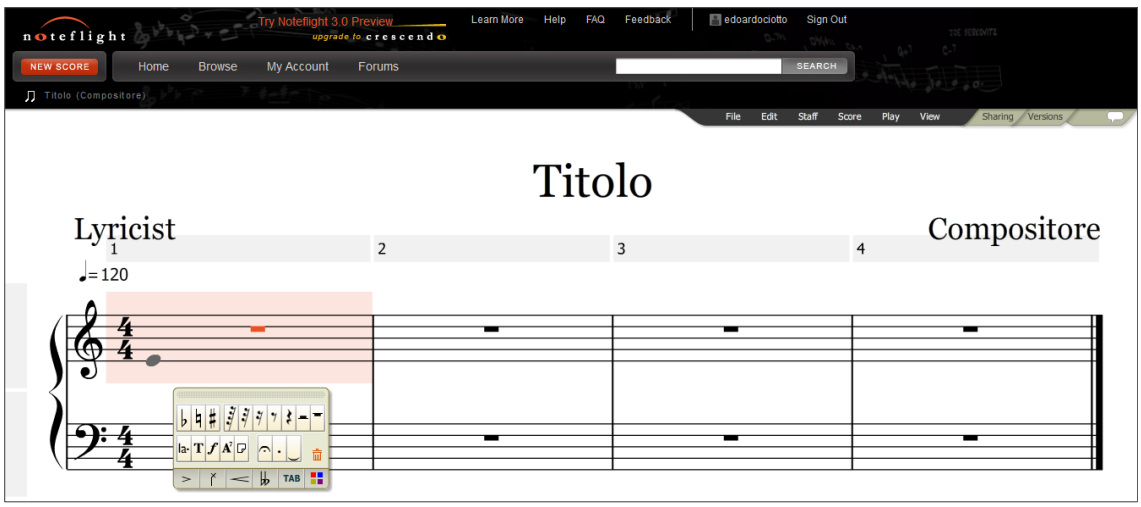


Fig. 3– Editor Noteflight di una nuova partitura.

Come è possibile osservare in **fig. 3,** l'applicazione offre numerose features, garantendo una buona personalizzazione della partitura. Si presenta come un vero e proprio software di notazione musicale, senza installazione richiesta e con la possibilità di condividere le proprie creazioni[8]. Inoltre, è possibile l'ascolto diretto della partitura avendo a disposizione un buon numero di timbri strumentali.

Il suo utilizzo necessita un'iscrizione iniziale al portale Web inserendo i consueti dati identificativi (indirizzo e-mail, password, tipo d'utilizzo del software) garantendo due tipi di iscrizioni: Free e Crescendo Membership.

	Free	Crescendo
Tipo di abbonamento	Free	Mensile o annuale
N. Score	10 possibili partiture	Numero illimitato di partiture
Timbrica	15 timbri base	Più di 50 timbri in alta qualità
Condivisione	Condivisione online	Gestione completa della condivisione dei contenuti: cosa fare vedere a chi
Stampa	Intero spartito	Possibilità di selezionare la parte interessata da stampare
Multipiattaforma	Il software è stato adattato anche per l'utilizzo multipiattaforma	
Conversione File	Import ed export di file MusicXML e MIDI	
Feaetures aggiuntive		Organizzazione dei file in cartelle, utilizzo di device MIDI per la scrittura della partitura, audio mixing, personalizzazione dell'aspetto visivo dei vari elementi presenti su partitura, utilizzo di templates.

2.2.2 – VexFlow: online music notation rendering API

Altro software in analisi è VexFlow, un API open-source online dedicata al rendering di notazione musicale, ideata da Mohit Muthanna sotto lo pseudonimo di 0xfe.

Scritto interamente in JavaScript, offre la possibilità di inserire all'interno di qualsiasi pagina Web HTML5 l'illustrazione grafica di uno spartito musicale implementata in un elemento canvas.

L'applicazione supporta la creazione di pentagrammi, chiavi musicali di base (violino, basso, alto), strutture di accordi, altezza, alterazioni, durata, punti di valore, misure, tempo, tonalità, legature, codette, tratti di unione, gruppi irregolari, tablature per chitarra[3] .

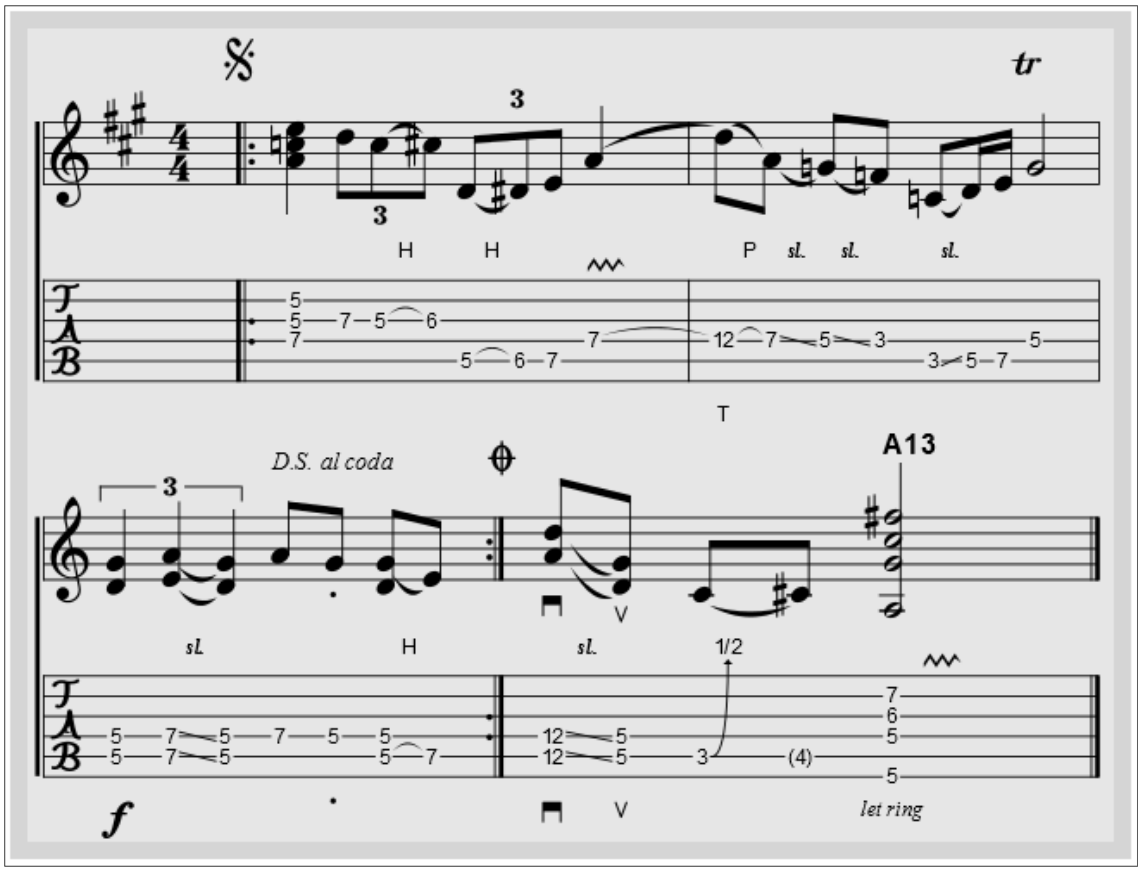


Fig. 4– Esempio proposto dall'homepage del sito www.vexflow.com renderizzato direttamente sulla pagina Web.

L'API non presenta ancora un interfaccia per la creazione diretta di uno spartito musicale come in Noteflight, impone dunque una conoscenza da parte dell'utente di programmazione JavaScript e HTML5. Tuttavia permette una maggior flessibilità nella personalizzazione dello spartito e della pagina ed offre il completo utilizzo del codice sorgente.

2.2.3 – Noteflight e VexFlow a confronto

Delineate le caratteristiche principali dei software in analisi, sono stati confrontati i criteri utili alla realizzazione del elaborato di Render grafico di documenti IEEE1599.

	Noteflight	VexFlow
Requisiti utente	Conoscenze di teoria e notazione musicale.	Conoscenze di teoria, notazione musicale, programmazione JavaScript e HTML5.
Requisiti di visualizzazione	Qualsiasi dispositivo con connessione di rete e browser.	Qualsiasi dispositivo con connessione di rete e browser che supporta HTML5.
Qualità grafica	Buona qualità e fedele alla concorrenza dei software di notazione musicale presenti in commercio.	Basilare e completamente personalizzabile ed editabile dal codice sorgente.
Condivisione	Condivisione dei contenuti Noteflight tramite link e finestre di condivisione.	Inserimento libero dell'intera applicazione all'interno di una qualsiasi pagina Web, includendo le librerie necessarie.
Interazione con file di formati diversi	Import ed Export di file MusicXML e MIDI.	L'importazione di un file è possibile attuando funzioni di parsing del documento interessato tramite JavaScript.
Licenza di utilizzo dei materiali	Ogni documento creato è limitato dalle condizioni d'uso di Noteflight[9].	Tutti i materiali e i codici sono sotto Licenza MIT open-source, reperibili da GitHub.com.

Le caratteristiche positive di una condivisione libera, un'interazione con file di diverso formato, la completa personalizzazione delle partiture e una licenza

open-source, hanno promosso VexFlow l'API adatta alla realizzazione dell'elaborato.

2.3– VexFlow come render di documenti IEEE1599: progetti simili

La scelta di VexFlow come applicazione necessaria alla realizzazione di un render di documenti IEEE1599, ha portato alla ricerca e all'analisi di progetti simili che utilizzano già VexFlow come render di documenti musicali.

Come è già stato scritto in precedenza, VexFlow è un'applicazione giovane e l'ambiente Web sta incominciando solo ora a movimentarsi per l'utilizzo di questa applicazione. Tuttavia, durante l'analisi del panorama dell'utilizzo di VexFlow come applicazione Web, è stato riscontrato una risposta positiva alla ricerca di un progetto simile al render di documenti IEEE1599: "HTML5 Guitar Tab Player - FireFox 4 Audio Data API".

Il presente progetto è stato realizzato da Greg Jopa, uno studente universitario laureato nell'Università Statale dell'Illinois, che offre la possibilità di acquisire informazioni da un file MusicXML e realizzare uno spartito musicale tramite API Vex Flow. Inoltre, utilizzando FireFox4 Audio Data API ha realizzato un lettore e player di tablature [10].

MusicXML è un formato di codifica di spartiti musicali basato su XML, realizzato nei primi anni del 2000 dall'azienda Californiana Recordare LLC. Il suo scopo è quello di favorire la condivisione di partiture codificandole in un formato di interscambio musicale, aperto a modifiche, semplice da utilizzare e in grado di rappresentare la maggior parte delle varianti del sistema di notazione utilizzato[11]. Molto simile al formato IEEE1599, che utilizza lo stesso linguaggio di markup per la codifica di informazioni musicali, ma limitato alla sola codifica delle partiture.

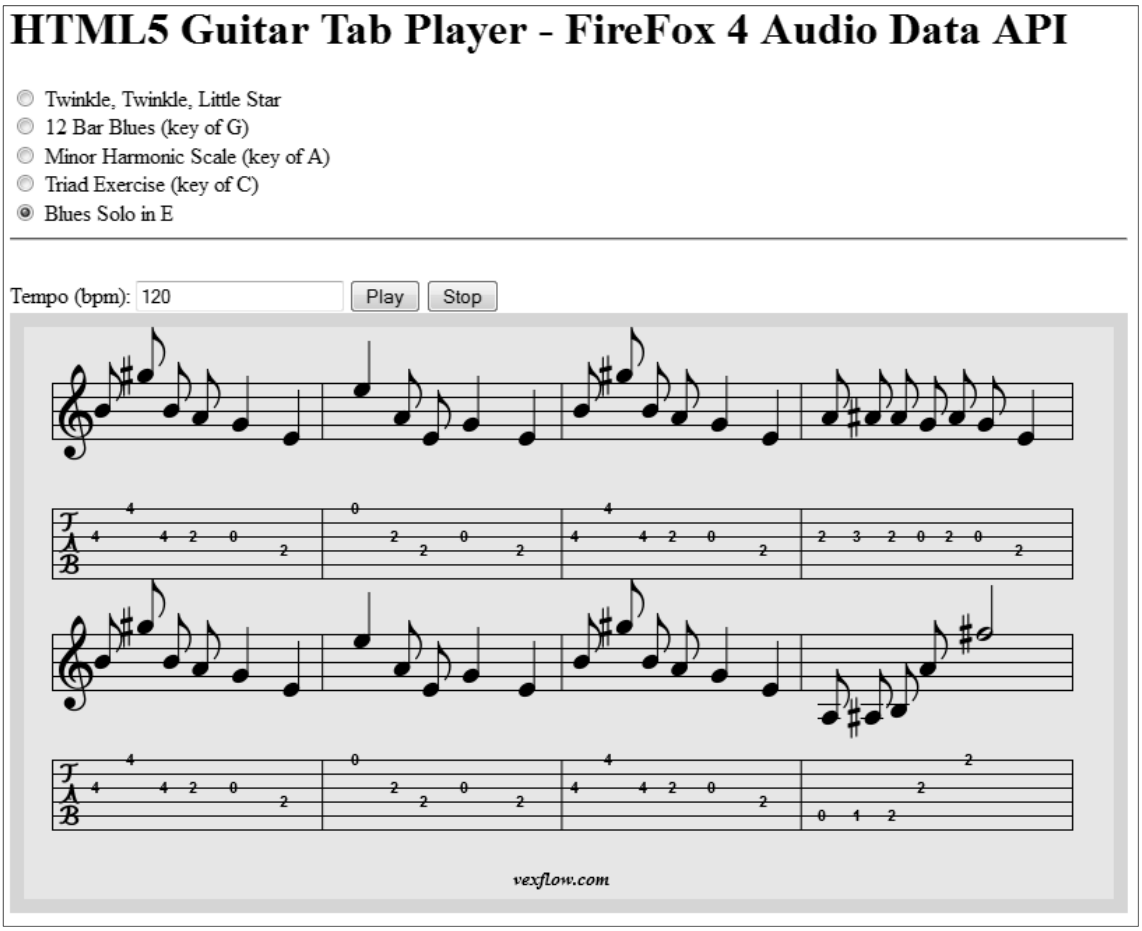


Fig. 5– Uno degli esempio proposto da Greg Jopa per mostrare il funzionamento del proprio player: "Blues Solo in E".

Analizzando il file sorgente della pagina sopra illustra in fig.5, è stata formulata l'idea finale dell'elaborato procedendo con lo studio dei diversi linguaggi di codifica da utilizzare e l'approfondimento dei temi necessari per la realizzazione del progetto.

3 – Analisi delle tecnologie e linguaggi utilizzati

3.1- Formato IEEE1599-2008

Il formato IEEE1599-2008 è uno standard internazionale approvato dall'Istitute of Electrical and Electronics Engineers (IEEE) il cui obiettivo è fornire una descrizione in XML di singoli brani musicali, prendendo in considerazione tutte le forme di descrizione collegabili al brano stesso[12].

3.1.1- Storia del formato

IEEE 1599 è il nome ufficiale di MX, acronimo di Musical application using XML. Il progetto di realizzare un formato in grado di rappresentare la musica tramite documenti XML, aprendo numerose modi per condividere la musica tra musicisti professionisti e non, è il risultato della ricerca guidata dal Laboratorio di Informatica Musicali (LIM), dell'Università degli Studi di Milano.

In torno al 2000, iniziano le prime fasi del progetto che consistevano nella valutazione delle caratteristiche di formati già esistenti per la descrizione di partiture musicali che utilizzavano NIFF, SMDL e XML: il primo è un formato introdotto nel 1994 basato su Resource Interchange File Format (RIFF) di Microsoft, un formato binario in grado di descrivere un file completo mediante una rappresentazione ASCII; il secondo, SMDL, acronimo di Standard Music Description Language, basato sul linguaggio SGML ma che non ebbe la sperata diffusione per via della complessità del formato rendendolo poco usabile; XML, invece, è un linguaggio di markup che utilizza delle "etichette" convenzionali per la codifica di testi elettronici[13].

Il progetto di standardizzazione IEEE inizia nel 2001 contattando altri gruppi di ricerca nel mondo accademico e commerciale.

La seconda fase inizia nel 2002, quando una versione prototipa del formato viene rilasciata, chiamata originariamente "Musical Application using XML" (MAX). Questo formato è stato discusso durante il MAX 2002, la prima conferenza internazionale sulle applicazioni musicali che utilizzavano XML, organiz-

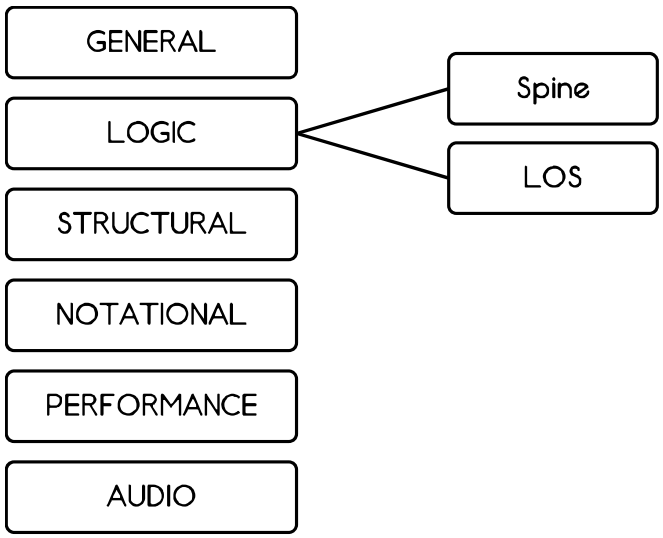
zato da IEEE CS TC on CGM, e fu analizzato e valutato da altri gruppi di ricerca per un'eventuale implementazione in applicazioni software.

Infine, nel luglio 2008, avviene l'ultimo processo di valutazione da parte di IEEE rendendo effettivo IEEE 1599 uno standard internazionale.[14]

3.1.2- Struttura dei documenti IEEE1599

Lo standard IEEE1599 utilizza tutti i vantaggi di un formato basato su XML: per esempio è libero, facile da leggere sia dall'uomo che da un computer ed è modificabile da comuni software di text editing. Inoltre XML è fortemente strutturato, come al maggior parte dei spartiti musicali, è può supportare nuovi simboli di notazione musicale[15].

La struttura di documento in formato IEEE1599 è caratterizzata da sei differenti livelli (layers).



Ad ogni layer è stata dedicata una specifica funzione:

- **General**, sono inserite tutte le informazioni relativi al brano di riferimento, tra cui titolo dell'opera, autori e genere;
- **Logic**, cuore del formato IEEE1599, suddiviso a sua volta in Spine e LOS, si occupa della descrizione logica dei simboli presenti su partitura;
- **Structural**, descrive ed identifica gli oggetti musicali del brano di riferimento, evidenziandone i mutui rapporti;

- **Notational**, contiene i riferimenti alle differenti rappresentazioni grafiche della partitura;
- **Performance**, dedicato a tutti i formati supportati che codificano i parametri delle note da eseguire e i parametri dei suoni da creare al fine di una produzione musicale sintetica da parte dell'elaboratore (es. Csound, SASL/SAOL);
- **Audio**, consente di legare le esecuzioni audio e video del brano di riferimento alla partitura.

Le relazioni tra i diversi layers sono schematizzati in **fig. 6**.

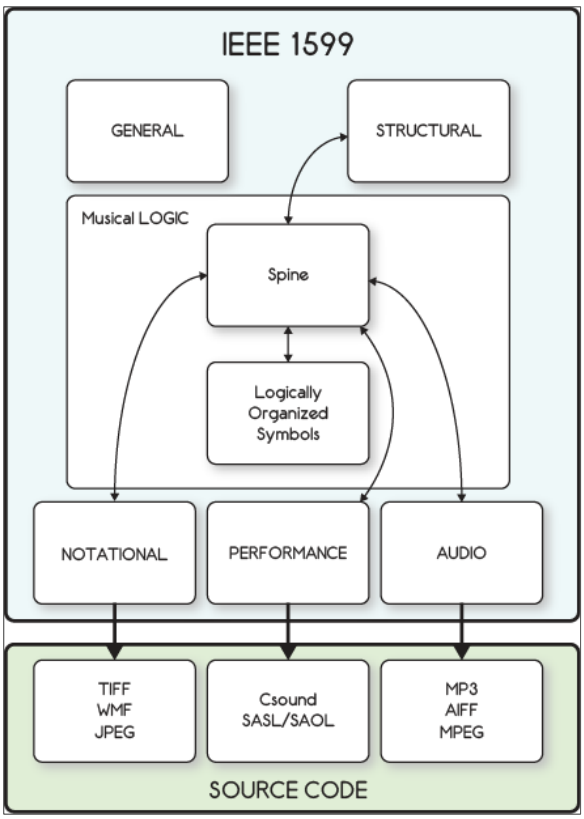


Fig. 6– Relazioni tra i layers di IEEE1599 e le risorse.

3.1.3- Applicazioni di IEEE1599

L'insieme dei layer del formato IEEE1599 descrivono a pieno le informazioni musicali acquisibili da un unico brano, pertanto le possibilità d'utilizzo sono notevolmente numerose: da software di intrattenimento multimediale a software puramente didattici, da software di analisi a editing di una partitura o semplicemente condivisione delle informazioni musicali in formato digitale.

A titolo di esempio, un'applicazione a fini didattici basata su IEEE 1599 è presente presso il Museo degli Strumenti Musicali del Castello Sforzesco di Milano offrendo al pubblico un'analisi accurata e complessa realizzata da Angela Ida De Benedictis su alcune opere di Bruno Maderna.

3.2- HTML5

HTML5 è la nuova evoluzioni di HTML, HyperText Markup Language, il linguaggio di markup utilizzato dal Web per eccellenza, offrendo un linguaggio moderno capace di soddisfare le nuove necessità di strutturazione del contenuto e sviluppo di nuove applicazioni.

3.2.1- Da HTML 4 HTML5

Nel 2004, con la creazione del gruppo di ricerca WHAT, acronimo di Web Hypertext Application Technology, il cui scopo era quello di elaborare nuove specifiche per lo sviluppo di un Web orientato alle applicazioni, incomincia un periodo di innovazione e ricerca di linguaggi sempre più all'avanguardia.

Questo distaccamento dal W3C, cioè il World Wide Web Consortium, comunità internazionale in cui le Organizzazioni Membro, momentaneamente 387, uno staff full time e gli utenti lavorano insieme per sviluppare standard per il Web[16], è determinato dal disaccordo in merito all'utilizzo di documenti XHTML2. Causa principale del disappunto era nella decisione di non mantenere la retro-compatibilità con la versione 1.1 imponendo, così, una maggior attenzione da parte dei sviluppatori Web.

Negli due anni successivi i due gruppi procedono separatamente con lo sviluppo dei propri lavori e specifiche. Il 27 ottobre 2006 Tim Bernes-Lee, presidente di W3C nonché inventore del Word Wide Web, scrive un post sul proprio blog intitolato "Reinventing HTML" annunciando la volontà di creare un nuovo gruppo di ricerca ammettendo alcuni sbagli commessi seguendo la filosofia XHTML2[17].

Passati altri due anni, nel luglio del 2009 Tim Bernes-Lee decide di direzionare gli sviluppi di W3C verso HTML5 a discapito di XHTML2. Come dalla pagina Web del W3C dedicata ad HTML5 si può notare (**fig. 7)** , lo sviluppo delle specifiche sono ancora in continua evoluzione e sostiene che la prima versione dello standard sarà pronta per fine 2014[18] .

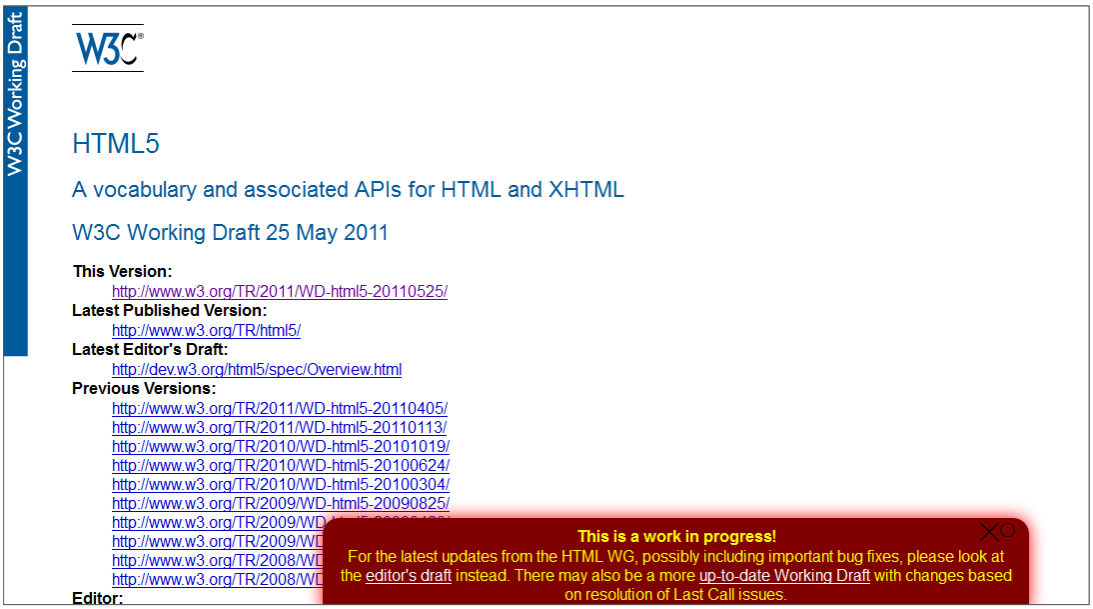


Fig. 7– Pagina Web del W3C dedicata ad HTML5

3.2.2- Novità da HTML5

HTML5 porta in campo Web numerose novità, sia a livello multimediale, come i tag `<video>` e `<audio>` progettati in modo che lo si possa usare senza nessuno script di controllo, sia a livello di utilità, come la possibilità di render utilizzabili delle applicazioni Web anche in modalità offline. Inoltre:

- altre novità riguardano la gestione degli elementi di testo, ad esempio le regole di strutturazione del testo in capitoli, paragrafi e sezioni sono più stringenti;

- vengono introdotti una vasta gamma di nuovi elementi rendendo la creazione della struttura di una pagina più semplice, come `<header>`, `<nav>`, `<section>`, `<article>`, `<aside>` e `<footer>`;
- nuovi tipi di input, come `type="search"` o `type ="url"`;
- sono stati eliminati, invece, alcuni elementi di scarso o minimo utilizzo ed è stato sostituito il lungo e complesso doctype con una dichiarazione più semplice `<!DOCTYPE html>`;
- Web Storage, un sistema di cookie alternativo più efficiente con un notevole risparmio di banda;
- La geolocalizzazione, cioè la capacità di localizzare il luogo in cui ci si trova ed eventualmente condividere l'informazione con altri.

3.2.3- Canvas HTML5

Una tra le novità più interessanti introdotte da HTML5 è il tag <canvas> definita come "una tela bitmap che dipende dalla risoluzione e che può essere usata per fare al volo il rendering di elementi grafici, immagini di giochi e altri elementi visivi"[19]. Si tratta di un rettangolo implementabile in una pagina Web all'interno del quale si può utilizzare il linguaggio di scripting JavaScript per realizzare graficamente qualsiasi forma e disegno in 2d e 3d.

L'elemento <canvas> viene introdotto da Apple nel 2004 con lo scopo di incrementare le funzionalità del suo MacOs X WebKit e disegnare in breve tempo un bitmap.

Metodo di rilevante importanza è `getContext` che, quando invocato, fa tornare un oggetto chiamato "contesto del disegno". In ambito 2d è possibile dividere le funzionalità in categorie differenti:

- **path**, cioè un insieme di punti uniti fra loro da definite primitive geometriche (arco, linea, ecc.);

- **modificatori**, tutte le funzioni che modificano il modo in cui i path, le immagini e i testi vengono disegnati;
- **immagine**, al quale appartiene un unico metodo chiamato drawImage, e permette di disegnare su un elemento di tipo immagine o video;
- **testo**, tutti i metodi che permettono l'inserimento di porzioni di testo all'interno di un canvas;
- **pixel per pixel**, tutti i metodi che gestiscono pixel per pixel di un canvas.

3.3- JavaScript

3.3.1- Introduzione e storia del formato

Il linguaggio JavaScript, che nulla ha a che vedere con il linguaggio Java di Sun Microsystems, è un linguaggio di scripting orientato agli oggetti, con il quale possono essere create sequenze complete di istruzioni da inserire nel codice di una pagina Web.

Nasce nel 1996 da Netscape, società creatrice di Netscape Navigator, il primo browser più diffuso nel mondo nella prima metà degli anni '90, e il protocollo crittografico SSL, Secure Sockets Layer, per assicurare una comunicazione online sicura.

Netscape voleva aumentare le capacità del proprio browser adottando un nuovo linguaggio di scripting, permettendo ai Web designer di interagire con i diversi oggetti della pagine, chiamato LiveScript le cui istruzioni potevano essere implementate all'interno del codice HTML e dare un aspetto dinamico ed interattivo alla pagina. La sintassi ricorda molto quella dei linguaggi di programmazione come C e Java.

A puro scopo di marketing e favorire la diffusione del nuovo linguaggio Netscape chiese a Sun Microsystems, multinazionale americana ideatrice del linguaggio di programmazione Java, di poterlo chiamare JavaScript in cambio di

render sempre più simili i nomi dei comandi, le sintassi e gli aspetti funzionali con quelli di Java.

Questo permise a Netscape di implementare all'interno del proprio browser Netscape Navigato 2.0 un interprete JavaScript.

Microsoft, pospose un nuovo linguaggio di scripting chiamato VBScript (Visual Basic Script), ma era compatibile solo con il browser di casa madre Internet Explorer. Affinché fosse possibile eseguire script sviluppati in JavaScript, Microsoft integrò un interprete per JScript, un linguaggio simile e compatibile con JavaScript, rendendo quest'ultimo utilizzabile e compatibile con tutti i browser [20].

3.3.2- Caratteristiche principali

JavaScript si presenta come un linguaggio semplice ed intuitivo, visualizzabile direttamente da un qualsiasi editor di testo o dal sorgente di una pagina Web.

Le sue principali caratteristiche sono:

- è un linguaggio case sensitive, quindi ha la capacità di distingue le minuscole dalle maiuscole;
- è universale, cioè gli script realizzati vengono interpretati all'interno di tutti i browser attualmente disponibili sul mercato. Inoltre, esistendo browser diversi per ogni sistema operativo, JavaScript viene definito un linguaggio cross-platform [20];
- è sicuro, cioè i progettisti hanno prestato attenzione a non fornire ai programmi scritti in questo linguaggio la possibilità di eseguire operazioni dannose. Tuttavia, in Netscape2, era possibile scrivere un codice JavaScript in grado di ottenere automaticamente l'indirizzo di posta elettronica dei visitatori[21]. Questi problemi sono stati risolti colmando le falle presenti nel linguaggio;

- è un linguaggio orientato agli oggetti, quindi prevede i meccanismi di incapsulamento, ereditarietà e polimorfismo;
- è facilmente implementabile in una qualsiasi pagina Web inserendo semplicemente i due consueti tag di apertura e chiusura dello script e selezionando il type uguale a text/javascript (come visibile nell'esempio qui sotto).

```
<script type="text/javascript">
document.write('Hello World') // stampa a video la scritta Hello World
</script>
```

3.3.3- DOM: Document Object Model

In molti credono che il DOM faccia parte di JavaScript, ma non è così. JavaScript è uno dei modi per accederci [22].

Un document object model (DOM) è un interfaccia di programmazione per le applicazioni (API) per rappresentare un documento e modificarne i vari elementi che i vari documenti che ne fanno parte.

DOM è uno standard ufficiale di W3C, progettato per l'utilizzo con documenti XML e HTML rappresentandoli a struttura d'albero.

Gli oggetti principali di tale struttura sono i nodi, cioè tutti i tipi di contenuto del documento. Ad esempio in un documento HTML i nodi principali saranno gli elementi `<body>`, `<head>`, `<p>`, ecc.

Si creano così strutture gerarchiche dove il nodo di livello superiore si chiamerà genitore, i nodi posti allo stesso livello fratelli mentre quelli posti ad un livello immediatamente inferiore si chiameranno figli.

3.4- SVG: Scalable Vector Graphics

Scalable Vector Graphics, in sigla SVG, standard W3C dal settembre 2001, è un'applicazione basata su XML per la rappresentazione di oggetti grafici in forma compatta e portabile. Nasce per l'esigenza forte di avere uno standard comune di grafica vettoriale da utilizzare sul Web. L'interesse di SVG è cresciuto rapidamente nelle maggiori compagnie software creando numerosi tools per la visualizzazione e la modifica di files SVG.

SVG ammette tre tipi di oggetti grafici:

- forme grafiche vettoriali, cioè le basilari linee, curve, linee, rettangoli, forme libere, ecc.;
- immagini (raster), cioè immagini rappresentate come una matrice i cui elementi sono detti pixel, descritti da un colore RGB o una paletta di colori. Questa matrice di pixel viene memorizzata in uno specifico formato in forma compressa;
- testo;

Un documento SVG inizia con le istruzioni standard XML e la dichiarazione `DOCTYPE`. In secondo luogo viene l'elemento `<svg>` con gli attributi `width` ed `height` che definiscono le dimensioni della tela. All'interno dei tag d'inizio SVG è possibile inserire gli elementi `<title>` e `<desc>` dedicati rispettivamente al titolo e alla descrizione dell'immagine. Infine viene inserito il tag rispettivo alla funzione interessata.

Ad esempio il tag `<circle />` inserirà un cerchio all'interno della tela di posizione, raggio, colore e riempimento dichiarati negli appositi attributi[23].

Esempio di un documento SVG:

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd>

<svg width="100" height="100">
```

```
  <title> Titolo Immagine </title>

  <desc> Descrizione dell'Immagine </desc>

  <circle cx="50" cy="50" r="25" style="stroke: black; fill:
  yellow"/>;

</svg>
```

Per osservare ciò che è stato creato da SVG può essere utilizzato un visualizzatore di illustrazioni grafiche vettoriali (es. Adobe® Illustrator®). Qui sotto (**fig. 8**) è rap-presentato il cerchio realizzato utilizzando il codice sopra descritto (il contorno quadrato nero dell'immagine non è creata da SVG).

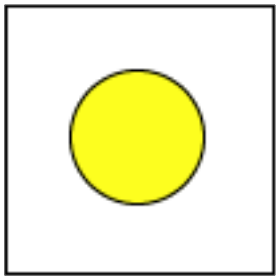


Fig. 8– Cerchio SVG visualizzato in Adobe® Illustrator®

3.5- API VexFlow

VexFlow, come già è stato scritto nel secondo capitolo, è un API open-source scritto interamente in JavaScript in grado di creare uno spartito musicale grafico in qualsiasi pagina Web. In questo terzo capitolo sono stati introdotti alcuni concetti fondamentali per il corretto funzionamento dell'API VexFlow.

In particolare HTML5 Canvas e SVG, elementi essenziali per il corretto funzionamento dell'applicazione.

3.4.1- Introduzione al codice

L'intero codice di VexFlow è distribuito sotto licenza MIT ed è reperibile online tramite il portale GitHub [24], una sorta di social network che permette la condivisione di codici per lo sviluppo di software. Questa soluzione ha reso possibile lo sviluppo del codice da parte non solo dell'ideatore 0xfe ma anche da altri utenti interessati in materia.

L'API VexFlow gestisce a livello di codice il Canvas HTML5, che dovrà essere obbligatoriamente implementato nella pagina Web tramite il tag `<canvas>`, disegnando su tela in grafica vettoriale simboli di notazione musicale dalle semplici note ai diversi glifi presenti su partiture cartacee.

Ogni funzione di VexFlow è descritta in una delle molteplici risorse rese disponibili dal codice in più piccoli file JavaScript, rendendo più fluido il suo utilizzo richiamandole semplicemente all'interno dell'`head` della pagina Web tramite il semplice tag `<script src=""></script>`.

3.4.2- Risorse essenziali: Raphaël JavaScript Library, jQuery

Alle risorse già presenti nell'intero codice di VexFlow bisogna aggiungere due librerie essenziali: Raphaël JavaScript Library e jQuery.

La prima è una libreria JavaScript, sotto licenza MIT, che si propone di semplificare il lavoro di chi utilizza la grafica vettoriale nel Web utilizzando SVG e VML, Vector Markup Language, cioè un linguaggio simile ad SVG che supporta il 3D. Momentaneamente è supportato da Firefox 3.0+, Safari 3.0+, Chrome 5.0+, Opera9.5+ e Internet Explorer 6.0+ [25]. Per essere utilizzato bisogna includere il file `raphael.js`, reperibile dal sito ufficiale www.raphaeljs.com, nella pagina HTML desiderata.

Fig. 9– Illustrazione offerta dal sito ufficiale come dimostrazione di Raphaël JavaScript Library

Seconda libreria essenziale da includere tra le risorse di VexFlow è la libreria JavaScript jQuery uno delle protagoniste della rinascita della tecnologia JavaScript per tre principali caratteristiche:

- offre una buona capacità di ovviare alcune delle peculiarità dei browser offrendo un'interfaccia e funzionalità omogenee;
- permette di accedere a elementi della struttura DOM attraverso selettori CSS;
- è di semplice utilizzo;

Si tratta di un framework, sotto licenza MIT, in grado di semplificare la programmazione lato client delle pagine HTML e si vanta di essere utilizzata anche da organizzazioni di primaria importanza, come Google, Dell, WordPress, Drupal, Mozilla.org, ecc.

jQuery è apprezzato molto nel Web per le animazioni. Infatti la libreria mette a disposizione una serie di animazioni predefinite e dispone le basi per la realizzazione di transizioni personalizzate.

Altro utilizzo di jQuery è quello di manipolare le classi CSS associate agli elementi HTML:

- per aggiungere una classe CSS ad un elemento è sufficiente selezionarlo e invocare il metodo `.addClass();`
- per rimuoverla il metodo è `.removeClass();`

La libreria jQuery definisce una singola variabile chiamata jQuery, che contiene una funzione capace di implementare tutte le funzionalità di base. Sono esposte come metodi sfruttando la funzione `.isNumeric()` per determinare se il parametro passato rappresenta un valore numerico [26].

Per compattare e rendere più ordinato il codice può esser utilizzato, al posto della sintassi jQuery, il simbolo `$`.

Come nel caso di Raphaël JavaScript Library, jQuery è scaricabile dal sito ufficiale `http://jquery.org/` ed è implementabile in una qualsiasi pagina Web utilizzando i tag `<script src=""></script>` .

3.4.3 – Elementi di notazione musicale con API VexFlow

Delineate le risorse necessarie al coretto funzionamento dell'applicazione riguardanti l'aspetto grafico e rappresentativo dell'informazione, in questa sezione saranno elencate le risorse presenti nel codice riguardanti gli elementi principali di notazione musicale.

Durante la trattazione dei seguenti argomenti saranno esposte piccole porzioni di codice e immagini esemplificative riguardanti le molteplici librerie presenti all'interno del progetto API VexFlow, scritte da Mohit Muthanna Cheppudira.

Il concetto di **pentagramma** è descritto nella libreria `stave.js` che gestisce la posizioni x e y nell'elemento Canvas, la larghezza e le diverse caratteristiche editabili (numero di linee, spazio tra una linea e l'altra, ecc.).

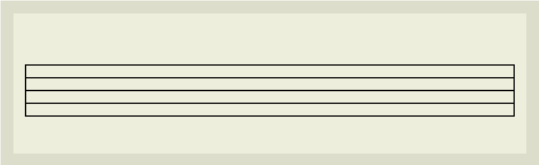


Fig.10 Pentagramma in VexFlow

Queste caratteristiche possono essere modificate nel codice nella sezione dedicata alle opzioni:

```
this.options = {
    vertical_bar_width: 10,
    glyph_spacing_px: 10,
    num_lines: 5,
    spacing_between_lines_px: 10, // in pixels
    space_above_staff_ln: 4,      // in staff lines
    space_below_staff_ln: 4,      // in staff lines
    top_text_position: 1,         // in staff lines
    bottom_text_position: 7       // in staff lines
  };
```

Inoltre in `stave.js` viene:

- impostata la chiave di violino come chiave default del pentagramma;
  `this.clef = "treble";`
- gestita la posizione sull'asse delle x di ogni nota e dei diversi elementi grafici, come glifi, segni di legatura ecc.
- controllata la creazione effettiva degli elementi grafici di inizio fine battuta;

Il concetto di **chiave** è gestita dalle librerie `clef.js` e `keysignature.js` al cui interno troveremo le regole di gestione delle stesse. In particolare:

- sono elencati i tipi di clef inseribili nel pentagramma (treble, bass, alto, tenor, percussion);
- il codice di riferimento al simbolo della tabella dei glifi (visionabile in Appendice B);
- posizione verticale, cioè la linea di riferimento di ogni chiave.

In `keysignature.js`, invece, è gestita la posizione delle note sul pentagramma, aggiungendo o togliendo un offset, in base alla chiave iniziale inserita:

```
switch (clef) {
  case "bass":
    offset = 1;
    break;
  case "alto":
    offset = 0.5;
    break;
  case "tenor":
    if (!isTenorSharps) {
      offset = -0.5;
    }
    break;
}
```

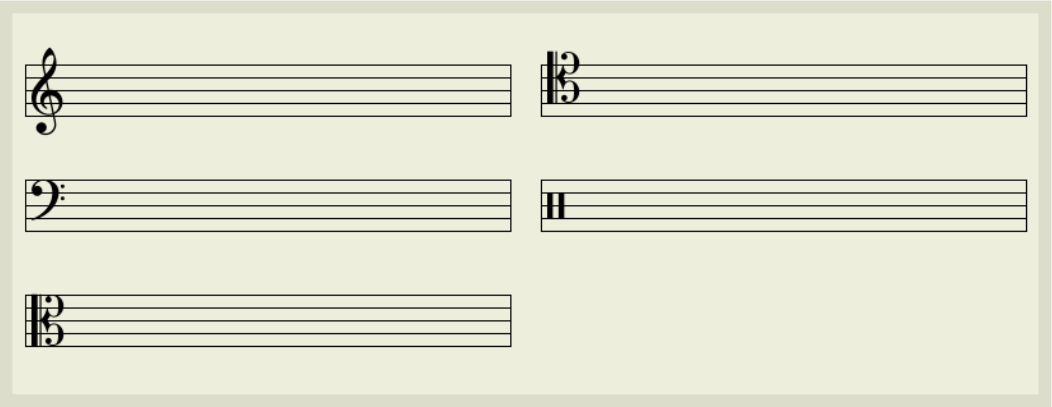


Fig.11 Chiavi disponibili in VexFlow

Concetto di **tonalità** è gestito principalmente nelle librerie `music.js` e `keymanager.js`.

Nella prima sono presenti le regole principali di teoria musicale, in particolare:

- numero totale delle note con e senza alterazioni;
- nomenclatura delle note in inglese;
- alterazioni e la loro gestione di ogni nota;
- nome e gestione degli intervalli diatonici;
- gestione di cinque tipi di scale (maggiore, minore, doriana, mixolydian);

In `keymanager.js` sono presenti le funzioni che riguardano la gestione delle armature di chiave e viene associato il carattere "`M`" per il modo maggiore e "`m`" per il modo minore.

```
Vex.Flow.KeyManager.scales = {
  "M": Vex.Flow.Music.scales.major,
  "m": Vex.Flow.Music.scales.minor
};
```

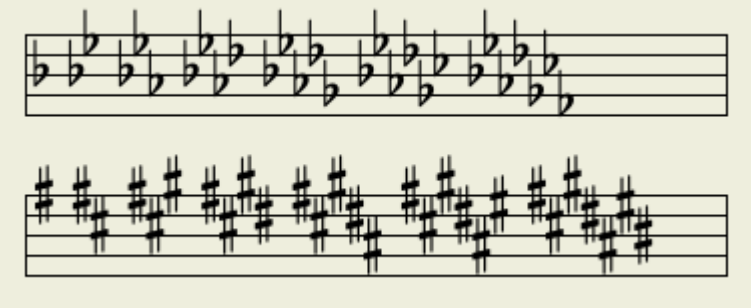


Fig.12 Armature di chiave

Il concetto di **tempo** è definito dalle librerie `tables.js` e `timesignatures.js`.

Nella prima sono gestite la rappresentazione della durata delle note (tiks):

```
Vex.Flow.durationToTicks.durations = {
  "1":    Vex.Flow.RESOLUTION / 1,
  "2":    Vex.Flow.RESOLUTION / 2,
  "4":    Vex.Flow.RESOLUTION / 4,
  "8":    Vex.Flow.RESOLUTION / 8,
  "16":   Vex.Flow.RESOLUTION / 16,
  "32":   Vex.Flow.RESOLUTION / 32,
  "64":   Vex.Flow.RESOLUTION / 64,
  "256":   Vex.Flow.RESOLUTION / 256
};
```

Inoltre è regolato il posizionamento sull'asse x delle note in base al numero di tiks, la durata delle pause e il loro posizionamento x, i glifi delle pause e del tempo di battuta.

In `timesignatures.js` sono gestiti invece i tempi di ogni battuta, associando un glifo per ogni tipologia.

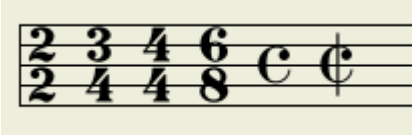


Fig.13 Glifi dei tempi su VexFlow

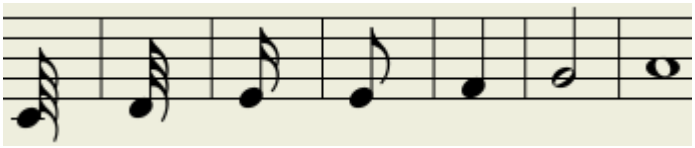


Fig.14 Durata delle note

Il concetto di **punto di valore** è definito dalla libreria `dot.js` che gestisce il suo utilizzo, la forma e la posizione in relazione alla nota.

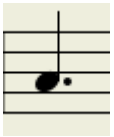


Fig.15 Punto di valore

La nozione di **legatura di valore** è definita dalla libreria `stavetie.js`. Al suo interno sono presenti le regole di gestione e quelle di render.

```
this.render_options = {
    cp1: 8,       // Curve control point 1
    cp2: 15,       // Curve control point 2
    text_shift_x: 0,
    first_x_shift: 0,
    last_x_shift: 0,
    y_shift: 7,
    tie_spacing: 0,
    font: { family: "Arial", size: 10, style: "" }
  };
```

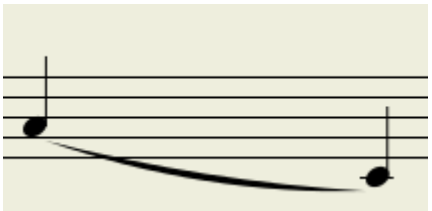


Fig.16 Legatura di valore

Qui di seguito saranno elencate le altre librerie `.js`, non citate in precedenza, con il relativo compito nell'API VexFlow.

accidental.js	Gestisce le funzioni riguardanti le alterazioni di nota
annotation.js	Implementa le annotazioni testuali
articulation.js	Gestisce i glifi e le loro posizioni delle articolazioni
barnote.js	Gestisce i glifi di misure, ripetizioni e fine
beam.js	Controlla le codette e stanghette di raggruppamento
bend.js	Implementa i glifi e le funzioni di bending per le tablature
flow.js	Definisce il "flusso", cioè il rhythm timing
formatter.js	Gestisce le regole di temporizzazione di una battuta
ghostnote.js	Implementa i glifi e regola l'utilizzo della ghostnote (corda muta)
glyph.js	Gestisce i glifi della tabella dei glifi (Appendice B)
header.js	Commento dell'autore e termini d'uso
modifier.js	Implementa diversi tipi di "modificatori di nota" (es. bends, posizione delle dita, ecc.)
renderer.js	Supporto per le differenze del rendering di contesto tra Canvas e la libreria Raphael
stavebarline.js	Implementa le barlines (singola, doppia, ripetizione e fine)
stavehairpin.js	Implementa il concetto di Crescendo o Decrescendo
stavemodifer.js	Classe base per la modifica dei pentagrammi
stavenote.js	Gestisce i glifi e il posizionamento delle note sul pentagramma
staverepetition.js	Implementa i glifi e i concetti di ripetizione(Coda, signo, D.C. ecc). Autore di questa libreria è Larry Kuhns
stavevolta.js	Implementa il concetto di Volta per le ripetizioni. Autore di questa libreria è Larry Kuhns
tabnote.js	Gestisce le note sul sistema di notazione tablature
tabslide.js	Implementa il concetto di slide nelle tablature
tabstave.js	Gestisce le tablature
tabtie.js	Gestisce il concetto di legature nelle tablature
tickable.js	Interfaccia della durata dei tick.
tickcontext.js	Gestisce il concetto di tick per le note, gli accordi e le tablature
tremolo.js	Implementa l'effetto tremolo e il relativo glifo
vibrato.js	Implementa l'effetto vibrato e il relativo glifo
voice.js	Gestisce il concetto di voce e accordo
voicegroup.js	Gestisce l'intera voce

3.4.4 – VexTab: un evoluzione "User Friendly"

Con il termine User-Friendly si vuole indicare un tipo di software di facile usabilità[27]. Mohit Muthanna definisce VexTab un "easy-to-use language to quickly quickly create, edit, and render standard notation and guitar tablature".

L'utilizzo e la scrittura di una partitura tramite VexTab risulta più semplice e intuitiva rispetto all'API VexFlow: mentre in quest'ultimo bisogna creare delle variabili, gestire il contesto del Canvas, conoscere notevoli nozioni di linguaggio JavaScript e le diverse funzioni offerte dal vasto numero di risorse, con VexTab viene semplicemente richiesto di gestire il contenuto di un elemento `<div>` di una pagina Web.

Qui sotto è inserito, a titolo esemplificativo, il contenuto di un elemento `<div>` e il suo relativo risultato:

```
tabstave notation=true

notes 2-4-2/3 10/4
```

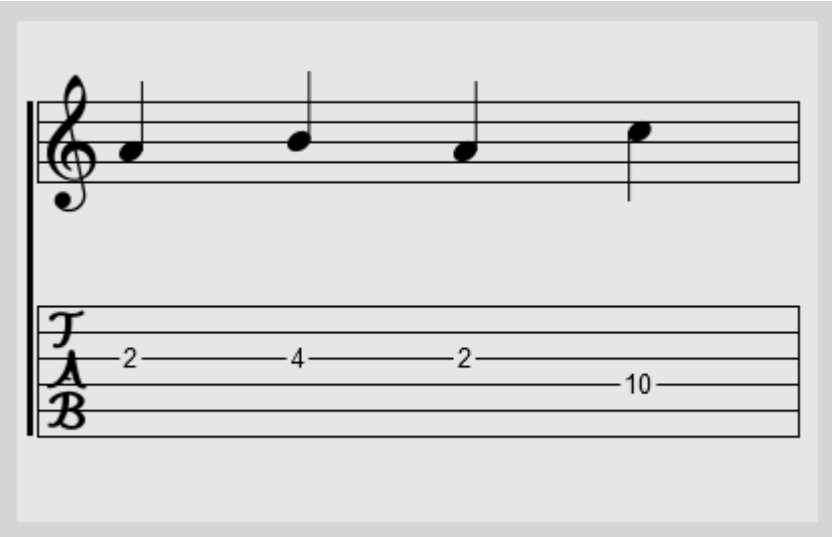


Fig.17 Esempio di VexTab

Analizzando il codice in esempio, si può intuire che il termine il comando notation=true indica se far visualizzare o meno il pentagramma con le relative note e i valori di notes si riferiscono alla posizione dei tasti (a sinistra di "/") e la corda (a destra di "/"). Inoltre si può affermare che i valori di default sono: chiave di violino, tablatura presente, durata di ¼ per tutte le note create.

Nel quarto capitolo saranno approfonditi i principali comandi utilizzati per la realizzazione del presente elaborato sia in ambito VexFlow che in ambito VexTab. Il sito di VexFlow (`www.vexflow.com`) offre un utile tutorial per i principali comandi di VexFlow e VexTab.

Il 9 gennaio 2013 viene annunciato il lancio di My VexFlow, una piattaforma che permette la condivisione e la pubblicazione di contenuti musicali basati su API VexFlow.

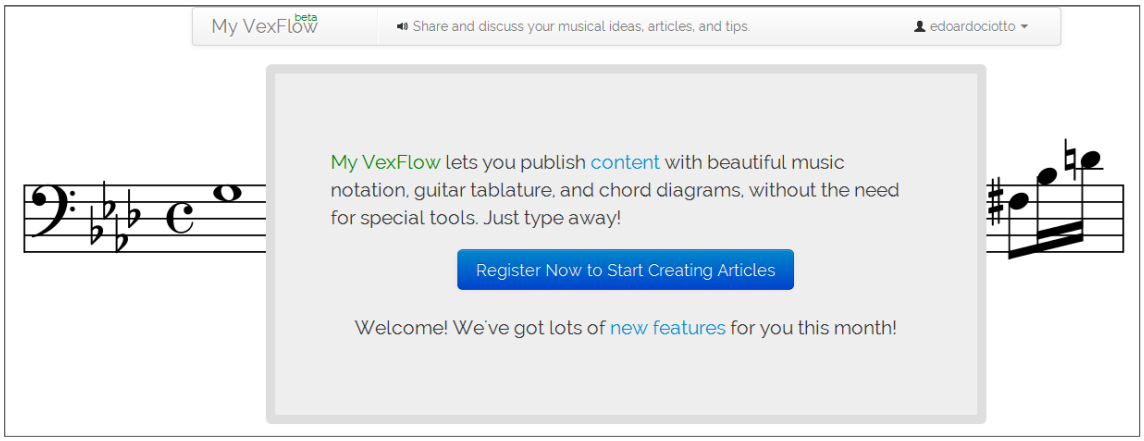


Fig.18 Homepage di My VexFlow

4 – Realizzazione del progetto

Il seguente capitolo vuole descrivere, passo a passo, le fasi di progettazione e stesura del codice finale (Appendice A). In corso di trattazione sono state inserite alcune righe di codice opportunamente commentate per descrivere i passaggi di rilevante importanza, accompagnate da immagini esemplificative allo scopo di mostrarne il risultato ottenuto.

In fase di progettazione ed analisi è stato utilizzato il file formato IEEE1699 "`beethoven_gottes_macht.xml`", codifica del celebre brano "Gottes Macht und Vorsehung", di Ludwig Van Beethoven.

Fig.19 Stesura automatica dei principali metadati presenti nel file `beethoven_gottes_macht.xml` da parte del progetto finale.

4.1- Tre diverse versioni del codice

Come già è stato scritto nel capitolo introduttivo, il progetto di "Render di documenti IEEE 1599 tramite API VexFlow" si è strutturato in tre principali passaggi che hanno portato miglioramenti nel progetto e nella strutturazione del codice.

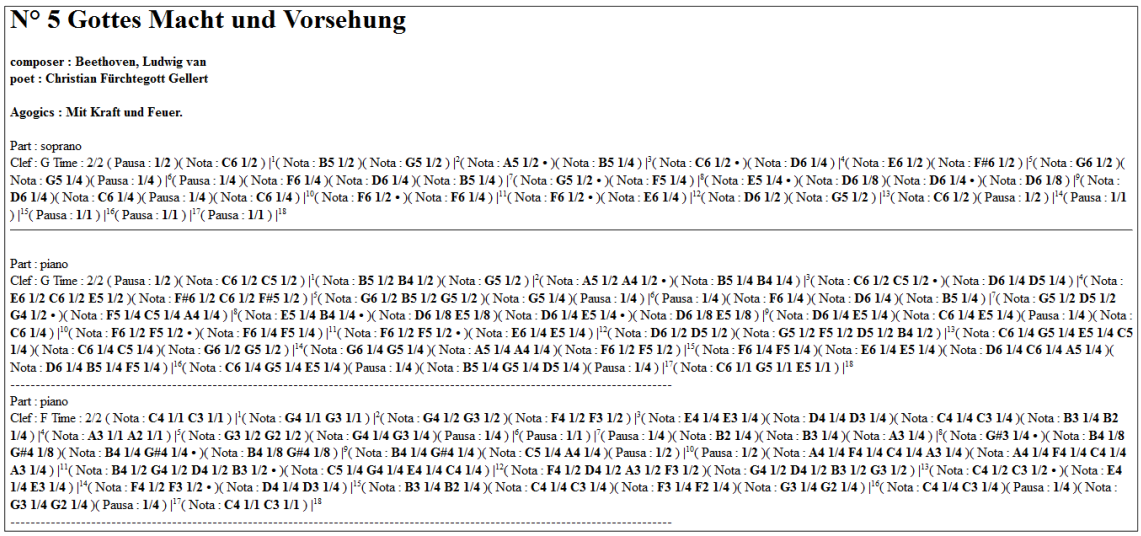
I motivi principali per cui sono stati creati tre differenti versioni del codice sono puramente di natura didattica e sperimentale. Infatti, in fase di progettazione:

- nel primo codice sono stati approfondite le conoscenze del linguaggio di scripting orientato agli oggetti JavaScript e la struttura a strati di IEEE1599;

- nel secondo codice viene introdotta l'API VexFlow con conseguente sperimentazione e studio delle diverse funzioni di rendering grafico;
- nel terzo codice, quello definitivo, viene approfondita la conoscenza dell'API VexFlow e, in particolare, l'utilizzo di VexTab come render di partiture in grafica vettoriale.

Inoltre sono stati approfonditi i temi di notazione musicale[28][29] e sviluppo di pagine Web basate su html, html5, CSS e JavaScript.

4.2- Prima versione: Render testuale di documenti IEEE1599


N° 5 Gottes Macht und Vorsehung

composer : Beethoven, Ludwig van
poet : Christian Fürchtegott Gellert

Agogics : Mit Kraft und Feuer.

Part : soprano
Clef : G Time : 2/2 (Pausa : 1/2)(Nota : C6 1/2) |1(Nota : B5 1/2)(Nota : G5 1/2) |2(Nota : A5 1/2 •)(Nota : B5 1/4) |3(Nota : C6 1/2 •)(Nota : D6 1/4) |4(Nota : E6 1/2)(Nota : F#6 1/2) |5(Nota : G6 1/2)(Nota : G5 1/4)(Pausa : 1/4) |6(Pausa : 1/4)(Nota : F6 1/4)(Nota : D6 1/4)(Nota : B5 1/4) |7(Nota : G5 1/2 •)(Nota : F5 1/4) |8(Nota : E5 1/4 •)(Nota : D6 1/8)(Nota : D6 1/4 •)(Nota : D6 1/8) |9(Nota : D6 1/4)(Nota : C6 1/4)(Pausa : 1/4)(Nota : C6 1/4) |10(Nota : F6 1/2 •)(Nota : F6 1/4) |11(Nota : F6 1/2 •)(Nota : E6 1/4) |12(Nota : D6 1/2)(Nota : G5 1/2) |13(Nota : C6 1/2)(Pausa : 1/2) |14(Pausa : 1/1) |15(Pausa : 1/1) |16(Pausa : 1/1) |17(Pausa : 1/1) |18

Part : piano
Clef : G Time : 2/2 (Pausa : 1/2)(Nota : C6 1/2 C5 1/2) |1(Nota : B5 1/2 B4 1/2)(Nota : G5 1/2) |2(Nota : A5 1/2 A4 1/2 •)(Nota : B5 1/4 B4 1/4) |3(Nota : C6 1/2 C5 1/2 •)(Nota : D6 1/4 D5 1/4) |4(Nota : E6 1/2 C6 1/2 E5 1/2)(Nota : F#6 1/2 C6 1/2 F#5 1/2) |5(Nota : G6 1/2 B5 1/2 G5 1/2)(Nota : G5 1/4)(Pausa : 1/4) |6(Pausa : 1/4)(Nota : F6 1/4)(Nota : D6 1/4)(Nota : B5 1/4) |7(Nota : G5 1/2 D5 1/2 G4 1/2 •)(Nota : F5 1/4 C5 1/4 A4 1/4) |8(Nota : E5 1/4 B4 1/4 •)(Nota : D6 1/8 E5 1/8)(Nota : D6 1/4 E5 1/4 •)(Nota : D6 1/8 E5 1/8) |9(Nota : D6 1/4 E5 1/4)(Nota : C6 1/4 E5 1/4)(Pausa : 1/4)(Nota : C6 1/4) |10(Nota : F6 1/2 F5 1/2 •)(Nota : F6 1/4 F5 1/4) |11(Nota : F6 1/2 F5 1/2 •)(Nota : E6 1/4 E5 1/4) |12(Nota : D6 1/2 D5 1/2)(Nota : G5 1/2 F5 1/2 D5 1/2 B4 1/2) |13(Nota : C6 1/4 G5 1/4 E5 1/4 C5 1/4)(Nota : C6 1/4 C5 1/4)(Nota : G6 1/2 G5 1/2) |14(Nota : G6 1/4 G5 1/4)(Nota : A5 1/4 A4 1/4)(Nota : F6 1/2 F5 1/2) |15(Nota : F6 1/4 F5 1/4)(Nota : E6 1/4 E5 1/4)(Nota : D6 1/4 C6 1/4 A5 1/4)(Nota : D6 1/4 B5 1/4 F5 1/4) |16(Nota : C6 1/4 G5 1/4 E5 1/4)(Pausa : 1/4)(Nota : B5 1/4 G5 1/4 D5 1/4)(Pausa : 1/4) |17(Nota : C6 1/1 G5 1/1 E5 1/1) |18

Part : piano
Clef : F Time : 2/2 (Nota : C4 1/1 C3 1/1) |1(Nota : G4 1/1 G3 1/1) |2(Nota : G4 1/2 G3 1/2)(Nota : F4 1/2 F3 1/2) |3(Nota : E4 1/4 E3 1/4)(Nota : D4 1/4 D3 1/4)(Nota : C4 1/4 C3 1/4)(Nota : B3 1/4 B2 1/4) |4(Nota : A3 1/1 A2 1/1) |5(Nota : G3 1/2 G2 1/2)(Nota : G4 1/4 G3 1/4)(Pausa : 1/4) |6(Pausa : 1/1) |7(Pausa : 1/4)(Nota : B2 1/4)(Nota : B3 1/4)(Nota : A3 1/4) |8(Nota : G#3 1/4 •)(Nota : B4 1/8 G#4 1/8)(Nota : B4 1/4 G#4 1/4 •)(Nota : B4 1/8 G#4 1/8) |9(Nota : B4 1/4 G#4 1/4)(Nota : C5 1/4 A4 1/4)(Pausa : 1/2) |10(Pausa : 1/2)(Nota : A4 1/4 F4 1/4 C4 1/4 A3 1/4)(Nota : A4 1/4 F4 1/4 C4 1/4 A3 1/4) |11(Nota : B4 1/2 G4 1/2 D4 1/2 B3 1/2 •)(Nota : C5 1/4 G4 1/4 E4 1/4 C4 1/4) |12(Nota : F4 1/2 D4 1/2 A3 1/2 F3 1/2)(Nota : G4 1/2 D4 1/2 B3 1/2 G3 1/2) |13(Nota : C4 1/2 C3 1/2 •)(Nota : E4 1/4 E3 1/4) |14(Nota : F4 1/2 F3 1/2 •)(Nota : D4 1/4 D3 1/4) |15(Nota : B3 1/4 B2 1/4)(Nota : C4 1/4 C3 1/4)(Nota : F3 1/4 F2 1/4)(Nota : G3 1/4 G2 1/4) |16(Nota : C4 1/4 C3 1/4)(Pausa : 1/4)(Nota : G3 1/4 G2 1/4)(Pausa : 1/4) |17(Nota : C4 1/1 C3 1/1) |18

Fig.20 Stesura automatica delle principali informazioni (altezza, durata, accordo, chiave, ecc.) in formato testuale.

Il primo approccio è stato quello di recuperare le principali informazioni di un brano musicale da un file IEEE1599, reso disponibile dal Laboratorio di Informatica Musicale, e stamparle a video tenendo conto della distinzione tra:

- parte e voce;
- nota e pausa;
- singola nota e accordo;
- durata e altezza della nota;

Inoltre l'obiettivo principale è stato quello di stampare le seguenti informazioni:

- titolo dell'opera;
- classificazioni degli autori con relativo nome e cognome;
- agogica;
- nome delle parti presenti nel brano;
- chiave, armatura di chiave e tempo;
- note e pause con relative durate e altezze;
- accordi e punti valore;
- misure numerate;

4.2.1 – Parsing di documenti IEEE1599

Comando essenziale per effettuare l'acquisizione delle informazioni necessarie per raggiungere l'obiettivo primario, è la funzione di parsing di documenti IEEE1599, in particolare di documenti XML.

Il parser è una funzione in grado di effettuare l'analisi sintattica di un testo rispetto ad un determinato linguaggio acquisendo, così, le informazioni da un documento di un certo tipo ad un altro. Nel caso in questione da un documento XML in uno script in JavaScript.

Al livello di codice, la funzione parser, si differenzia a seconda del browser utilizzato:

```
if(window.XMLHttpRequest){//IE7+, Firefox, Chrome, Opera, Safari
 xmlhttp=new XMLHttpRequest();}
else
{ xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");} // IE6, IE5
xmlhttp.open("GET"," beethoven_gottes_macht.xml ",false);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;

// Viene assegnato il documento alla variabile xmlDoc
```

Per motivi di sicurezza, i moderni browser non ammettono l'accesso tra domini, questo significa che sia la pagina Web che il file XML devono trovarsi sullo stesso server. Per poter visualizzare in locale il risultato del progetto, è stato utilizzato un altro metodo:

```
documentoXML = document.implementation.createDocument("", "", null);
documentoXML.async = false;
documentoXML.load("beethoven_gottes_macht.xml");
ElementiXML = documentoXML.documentElement;
```

dove:

- nella prima riga avviene l'implementazione del metodo `createDocument` alla variabile `documentoXML`;
- il comando `documentoXML.async = false` permette di caricare tutto il file prima di essere utilizzato;
- il metodo `.load("beethoven_gottes_macht.xml")`, assegna il documento indicato tra parentesi alla variabile `documentoXML`;
- nell'ultima riga, gli elementi del documento caricato vengono assegnati alla variabile `ElementiXML`;

Questo metodo funziona con i Browser Firefox e Opera, sia con documenti HTML e XML locali che remoti[30].

Browser utilizzati	XMLHttpRequest	createDocument
Local HTML e XML		
IE	no	no
Chrome	**si**	no
Safari	**si**	no
Firefox	**si**	**si**
Opera	**si**	**si**
Remote HTML e Remote XML		
IE	**si**	no
Chrome	**si**	no
Safari	**si**	no
Firefox	**si**	**si**
Opera	**si**	**si**

4.2.2 – Gestione degli oggetti acquisiti

Per capire cosa è stato inserito all'interno della variabili `ElementiXML` è stato utile osservare l'intero contenuto XML del documento caricato tramite il software di XML Copy Editor 1.2.0.6 [31], un software free rilasciato sotto GNU General Public License, orientato alla visualizzazione e la gestione di documenti XML.

Fig.21 Documento "`beethoven_gottes_macht.xml`" visualizzato tramite software XML Copy Editor 1.2.0.6

Ogni elemento presente nel documento è selezionabile ed attribuibile ad una variabile tramite il metodo `var a = variabile.getElementsByTagName('')` dove, tra parentesi, è richiesto l'inserimento del nome dell'elemento.

Esempio:

```
main_title = ElementiXML.getElementsByTagName('main_title');
```

in questo caso, la variabile `main_title` recupera tutti gli elementi del file XML aventi il nome (`'main_title'`).

In un documento IEEE1599 l'elemento `<main_title>` si trova nel livello General ed indica il Titolo dell'opera descritta.

```
<main_title> Gottes Macht und Vorsehung </main_title>
```

Ogni variabile creata tramite metodo `.getElementsByTagName` diventa automaticamente una variabile di tipo Array(). Ogni elemento caricato in una variabile

potrà essere utilizzato tramite il numero identificativo della posizione occupata all'interno dell'Array.

Il contenuto di un elemento, `Gottes Macht und Vorsehung` nel caso appena citato, è assegnabile ad una variabile o utilizzabile tramite il metodo: `main_title[0].childNodes[0].nodeValue;`

Dove:

- `main_title[0]` indica la variabile di riferimento e la posizione dell'elemento interessato;
- `childNodes[0].nodeValue` recupera il valore dell'elemento, cioè il suo contenuto;

Per assegnare il nome e il valore di un attributo viene utilizzato il metodo: `variabile = elemento[i].attributes;`

Esempio:

```
author_type = author[i].attributes;
```

Dove:

- `author_type` è la variabile di destinazione;
- `author[i]` è un Array creato tramite metodo `.getElementsByTagName;`
- `i` è una contatore utilizzata per scorrere il contenuto dell'Array `author`.

Nel corso della progettazione sono state create per ogni elemento utile, cioè rappresentabile su partitura, una variabile tramite i metodi sopra citati.

4.2.3 – Utilizzo delle informazioni musicali acquisite

Fig.22 Numero, titolo, artisti e agogica scritti automaticamente su pagina html

Uno dei primi obiettivi del progetto è quello di stampare a video titolo, autori e agogica di un brano codificato nel formato IEEE1599, come in **fig.22**.

Il codice utilizzato è il seguente:

```
number = ElementiXML.getElementsByTagName('number');
main_title = ElementiXML.getElementsByTagName('main_title');
author = ElementiXML.getElementsByTagName('author');
agogics = ElementiXML.getElementsByTagName('agogics');
//-------------------------------------------------------------
document.write('<div class="header"> <h1> N° ');
document.write(number[0].childNodes[0].nodeValue + " " );
document.write(main_title[0].childNodes[0].nodeValue );
document.write('</h1> <div class="main"> <b>');
//-------------------------------------------------------------
for (var i = 0; i < author.length; i++)
{
author_type = author[i].attributes;
document.write(author_type[0].value + " : ");
document.write(author[i].childNodes[0].nodeValue + "<br />");}
//-------------------------------------------------------------
document.write("<br />Agogics : " );
document.write(agogics[0].childNodes[0].nodeValue + "<br />");
document.write(' </b>  </div>');
```

Dove:

- la prima sezione gestisce i diversi elementi del documento associandolo ad una variabile;
- la seconda sezione stampa e gestisce il titolo, aggiungendo l'elemento `number`;
- la terza sezione presenta il ciclo degli autori, cioè un ciclo in grado di gestire e stampare la tipologia dell'autore e il suo nome;
- la quarta sezione stampa il contenuto dell'elemento `agogics`.

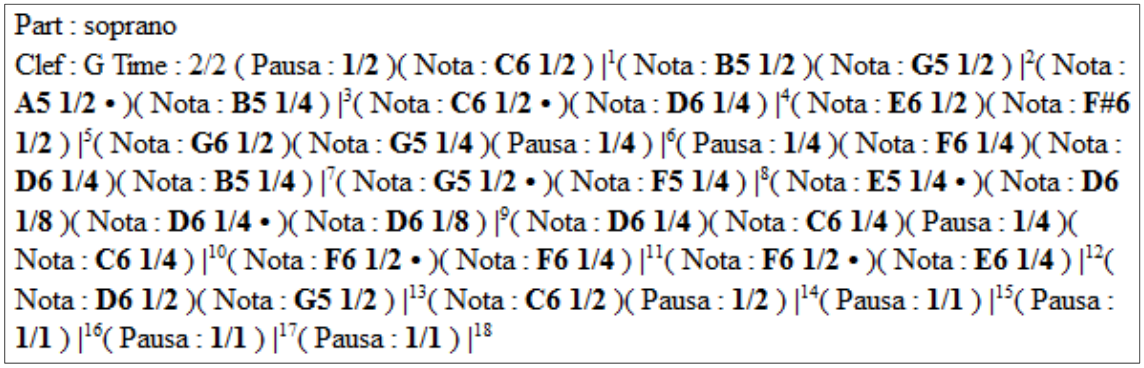

Part : soprano
Clef : G Time : 2/2 (Pausa : **1/2**)(Nota : **C6 1/2**) |[1](Nota : **B5 1/2**)(Nota : **G5 1/2**) |[2](Nota : **A5 1/2** •)(Nota : **B5 1/4**) |[3](Nota : **C6 1/2** •)(Nota : **D6 1/4**) |[4](Nota : **E6 1/2**)(Nota : **F#6 1/2**) |[5](Nota : **G6 1/2**)(Nota : **G5 1/4**)(Pausa : **1/4**) |[6](Pausa : **1/4**)(Nota : **F6 1/4**)(Nota : **D6 1/4**)(Nota : **B5 1/4**) |[7](Nota : **G5 1/2** •)(Nota : **F5 1/4**) |[8](Nota : **E5 1/4** •)(Nota : **D6 1/8**)(Nota : **D6 1/4** •)(Nota : **D6 1/8**) |[9](Nota : **D6 1/4**)(Nota : **C6 1/4**)(Pausa : **1/4**)(Nota : **C6 1/4**) |[10](Nota : **F6 1/2** •)(Nota : **F6 1/4**) |[11](Nota : **F6 1/2** •)(Nota : **E6 1/4**) |[12](Nota : **D6 1/2**)(Nota : **G5 1/2**) |[13](Nota : **C6 1/2**)(Pausa : **1/2**) |[14](Pausa : **1/1**) |[15](Pausa : **1/1**) |[16](Pausa : **1/1**) |[17](Pausa : **1/1**) |[18]

Fig.23 Trascrizione automatica delle principali informazioni della parte di soprano.

Passaggio successivo: acquisizione e trascrizione delle informazioni principali di una parte, come in fig.23.

In un primo momento, vengono dichiarate le variabili contententi le informazioni di parte, chiave e tempo:

```
part = ElementiXML.getElementsByTagName('part');
clef = ElementiXML.getElementsByTagName('clef');
time = ElementiXML.getElementsByTagName('time_indication');
```

All'interno di ogni parte viene caricata la lista delle voci e le misure presenti all'interno di una voce.

```
192 <part id="soprano">
193   <voice_list>
194     <voice_item id="soprano_voice1" staff_ref="staff_1"/>
195   </voice_list>
196   <measure number="1">
```

Questa caratteristica impone la creazione di due cicli di controllo: uno per la selezione della parte e l'altro per la selezione della voce.

```
for (var t = 0; t < part.length; t++) {
      voice_item = part[t].getElementsByTagName("voice_item");
            for (var g = 0; g < voice_item.length; g++) {
```

Il metodo `.length` permette di calcolare la lunghezza, cioè il numero massimo di contenuti presenti in un determinato Array.

`voice_item = part[t].getElementsByTagName("voice_item")` permette di inserire all'interno della variabile `voice_item` gli elementi di nome "`voice_item`" della parte numero t. In questo modo avremo un controllo su tutte le parti e le voci presenti nel documento: verrà analizzata una parte per volta.

Successivamente vengono stampate le informazioni acquisite:

```
document.write('Part : ' + part[t].getAttribute("id") + "<br />");
document.write("Clef : " + clef[rigo].getAttribute("shape") + " ");
document.write("Time : " + time[rigo].getAttribute("num") + "/" + ti-
me[rigo].getAttribute("den") + " ");
rigo++;
```

dove `rigo++` è un puntatore che permette di tener conto del pentagramma di riferimento senza tener conto della parte o della voce. Questo perché le variabili `clef` e `time` non sono legate alla parte ma al numero di elementi chiamati rispettivamente `clef` e `time`.

All'interno dell'elemento parte sono presenti le misure che saranno legate alla voce di riferimento presente nella `voice_list` tramite gli elementi voice. Ogni elemento voice si riferisce al proprio pentagramma tramite un nome identificativo.

```
196 <measure number="1">
197  <voice voice_item_ref="soprano_voice1">
198   <rest event_ref="v1_e0" staff_ref="staff_1">
199    <duration num="1" den="2"/>
200   </rest>
201   <chord event_ref="v1_e1">
202    <duration num="1" den="2"/>
203    <notehead staff_ref="staff_1">
204     <pitch step="C" octave="6"/>
205    </notehead>
206   </chord>
207  </voice>
208 </measure>
```

All'interno di ogni voce sono presenti gli elementi della misura (note, pause, articolazioni, ecc.).

Per recuperare tutte le informazioni appena descritte è stato creato un nuovo ciclo con il compito di controllare il numero di misura:

```
measure = part[t].getElementsByTagName('measure');
for (var x = 0 ; x < measure.length ; x++) {

voice = measure[x].getElementsByTagName("voice");
rest = voice[g].getElementsByTagName("rest");
chord = voice[g].getElementsByTagName("chord");

numero_misura = x + 1;
var n_rest = 0;
var n_chord = 0;
```

dove:

- `numero_misura` è utilizzato per scrivere il numero di ogni misura;
- `n_rest` e `n_chord` sono puntatori utilizzati per indicare gli elementi chord o rest presenti nella misura.

La gestione degli elementi presenti all'interno della misura di una voce avviene tramite un ciclo di controllo:

```
for (var m = 1; m < voice[g].childNodes.length; m += 2) {
```

In questo modo, il ciclo si concluderà ogni volta che verrà raggiunto il numero massimo di elementi contenuti all'interno di una misura. Anche se sembra atipica la gestione del contatore `m = 1` con incremento `m += 2`, risolve un problema del browser Firefox nell'acquisizione dei vari elementi di un file XML. Infatti gli "a capo" vengono calcolati come ulteriori oggetti creando elementi inesistenti di particolare disturbo [32]. In questo modo, saltando sempre un elemento, l'indicatore `m` punterà sull'elemento "giusto".

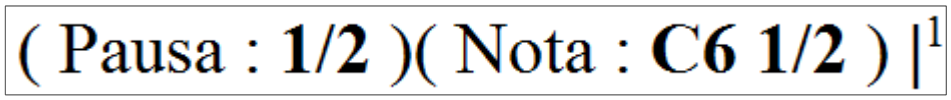


Fig.24 Due elementi stampati dall'applicazione

I concetti di pause e note vengono gestiti nel ciclo appena descritto tramite un sistema di `if () { } else { }`. Essendo una parte di codice considerevolmente complicata e lunga, non verrà inserita qui di seguito, tuttavia, può essere riassunta in passaggi:

- viene caricato l'elemento [m] e analizzato il suo nome;
- se si tratta di un elemento rest sarà una pausa, se si tratta di un elemento chord sarà una nota singola o un accordo;
- per le pause vengono controllati i valori degli attributi num e den dell'elemento duration e stampata la stringa: (Pausa: num/den);
- per gli accordi vengono contati gli elementi notehead presenti all'interno dell'elemento chord tramite un ciclo;
- vengono rilevati i valori degli attributi di pitch di ogni elemento notehead;
- viene rilevata la presenza del punto di valore o di un alterazione;

- viene rilevato il valore degli attributi num e den dell'elemento duration;
- vengono poste due condizioni: una riguardo la presenza o meno del punto di valore, l'altra riguardo la presenza o meno di un alterazione;
- in ogni condizione verrà stampato il relativo accordo inserendo, inoltre, l'ottava di riferimento della nota;

Fig.25 Posizionamento dei diversi simboli negli elementi

4.2.4 – Gestione delle voci e delle parti

L'aspetto grafico di questa applicazione si presenta ancora molto lontano dal progetto finale. Tuttavia, l'impaginazione di una pagina può restituire altre informazioni utili per la rappresentazione di uno spartito musicale.

Ad esempio la distanza tra uno spartito testuale e un altro può indicare l'appartenenza o meno alla stessa parte, oppure la presenza di linee divisorie possono gestire i concetti di voci e parti.

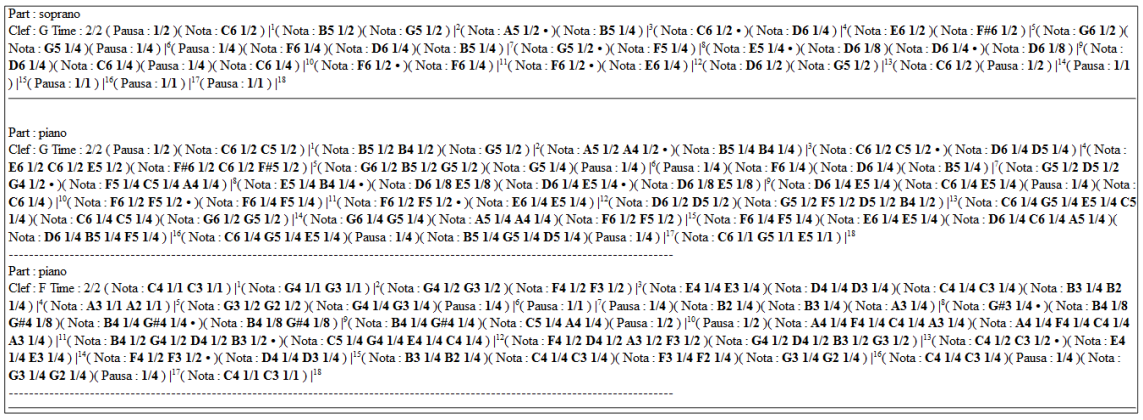


Fig.26 Linee Parti e Voci

Sfruttando la struttura a ciclo del codice per ogni fine parte è stato aggiunto il comando `document.write("<hr />")`, cioè che crea una linea retta orizzontale all'interno della pagina, mentre ad ogni fine voce viene stampata una linea tratteggiata come in **fig.26**. In questo modo possono essere distinte le parti e le voci presenti nel pentagramma.

4.2.5 – Prima analisi dell'applicazione

Per il controllo del corretto funzionamento dell'applicazione, sono stati presi in considerazione gli spartiti grafici in formato `.tif` collegati al file IEEE1599.

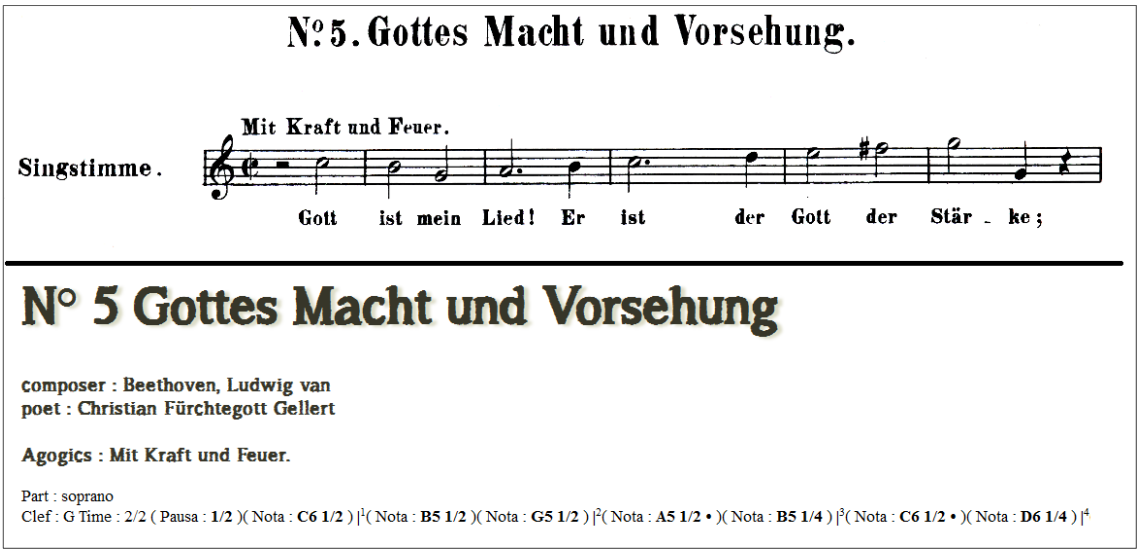


Fig.27 Confronto tra spartito grafico (sopra) e trascrizione automatica (sotto)

Confrontando gli elementi stampati con le prime quattro misure, come mostrato in **fig. 27**, risultano coincidenti senza presentare errori o sbagliate interpretazioni di decodifica. Lo stesso vale per le misure restanti affermando la prima versione del codice "scheletro" del progetto finale.

4.3- Seconda versione: Render grafico con API VexFlow

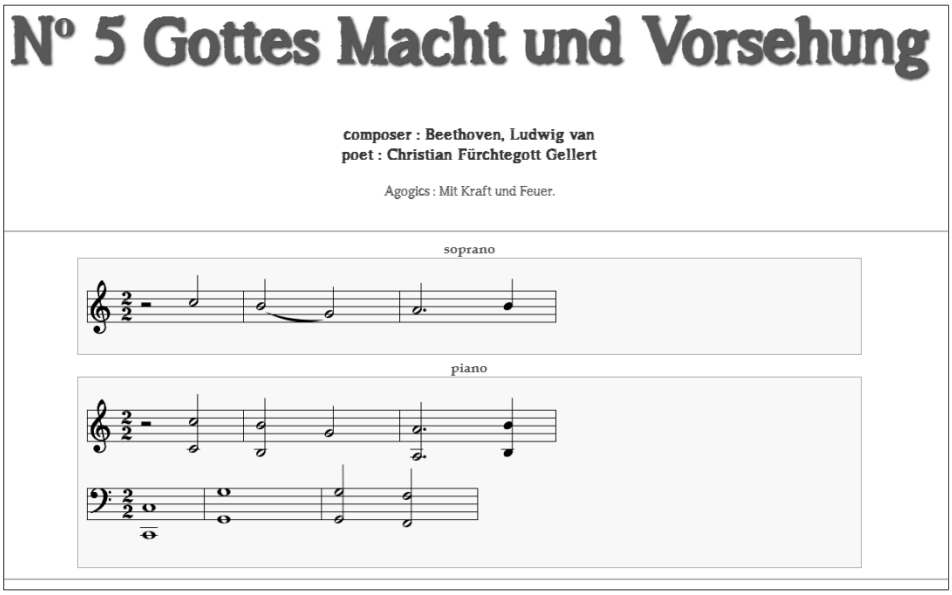


Fig.28 Pagina realizzata tramite la seconda versione del progetto.

Creato lo scheletro logico dell'intero progetto e testatone l'affidabilità sono state apprese le nozioni necessarie per l'utilizzo e l'implementazione dell'API VexFlow.

4.3.1- Implementazione API VexFlow

L'implementazione dell'API VexFlow impone il caricamento di tutte le librerie necessarie per il suo corretto funzionamento.

Tramite la linea di comando `<script src="nomelibreria.js"></script>` sono state inserite tutte le librerie descritte nel sotto-sottocapitolo 3.4.2. e presenti all'interno della cartella src, scaricabile da GitHub nella pagina dedicata all'API VexFlow.

Inoltre, per render più fluido e ordinato il codice, è stata creata una nuova libreria dal nome `RenderIEEE1599.js`, dove sono state inserite tutte le funzioni di questa seconda versione del progetto.

Elemento principale per il corretto funzionamento dell'API VexFlow è il Canvas HTML5.

Per implementarlo bisogna inserire il tag `<canvas> </canvas>` e stabilire attraverso gli attributi di height e width la dimensione della tela:

```
<canvas width=700 height=100"></canvas>
```

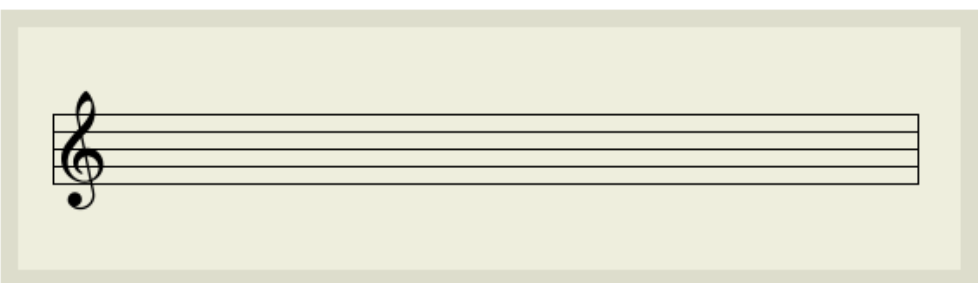
L'elemento `<canvas>` deve essere gestito a livello di script JavaScript:

```
//--Sezione 1
var canvas = $("div.one div.a canvas")[0];
var renderer = new Vex.Flow.Renderer(canvas,Vex.Flow.Renderer.Backends
.CANVAS);
//--Sezione 2
var ctx = renderer.getContext();
var stave = new Vex.Flow.Stave(10, 0, 500);
stave.addClef("treble").setContext(ctx).draw();
```

Dove:

- nella prima sezione vengono inizializzate le variabili `canvas` e `renderer` richiamando le funzioni che gestiscono il renderer dell'elemento `<canvas>`;
- nella seconda sezione viene inizializzata la variabile `ctx`, assegnando il contenuto del renderer di VexFlow al "contesto della tela" al `<canvas>`, e la variabile `stave`, creando un nuovo pentagramma, di posizione `(10,0)` e di larghezza `500px` e con chiave di tipo `"treble"` (SOL).
- infine viene assegnato il contenuto della variabile `stave` al contesto `ctx` e renderizzata nel `<canvas>` con il comando `.draw();`

Il risultato ottenuto è il seguente (**fig. 29**):



L'inserimento di **note**, **accordi** e **pause** è possibile tramite il seguente codice:

```
//-- Sezione 1
var notes = [
//-- Sezione 2
new Vex.Flow.StaveNote({ keys: ["c/4"], duration: "q"}),
new Vex.Flow.StaveNote({ keys: ["d/4"], duration: "q"}),
//-- Sezione 3
new Vex.Flow.StaveNote({ keys: ["b/4"], duration: "qr"}),
//-- Sezione 4
new Vex.Flow.StaveNote({ keys: ["c/4", "e/4", "g/4"], duration: "q"}];
//-- Sezione 5
var voice = new Vex.Flow.Voice({
    num_beats: 4,
    beat_value: 4,
    resolution: Vex.Flow.RESOLUTION
});
//-- Sezione 6
voice.addTickables(notes);
voice.draw(ctx, stave);
```

Dove:

- nella prima sezione viene inizializzata l'Array `notes`;
- nella seconda sezione vengono mostrati due comandi esemplificativi per l'inserimento di due note da 1/4;
- nella terza sezione viene mostrato il comando per l'inserimento di pause;
- nella quarta sezione viene mostrato il comando d'inserimento di un accordo;
- nella quinta sezione viene creata una voce da 4/4 ;
- nella sesta sezione vengono assegnate le note alla voce `voice` e renderizzate nell'elemento `<canvas>`.

Come è possibile notare nell'esempio appena descritto, i comandi di inserimento nota e inserimento pausa sono molto simili. Differenza sostanziale è il valore `duration` che per le pause sarà sempre seguito dalla lettera `"r"` di Rest. Di default le pause occupano la posizione del SI4.

I valori inseribili in `duration` sono:

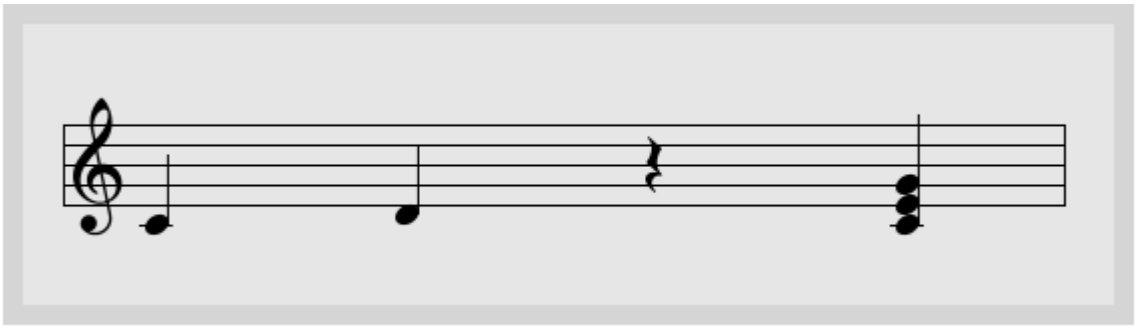
Forma letterale

- `w` per la semibreve (4/4);
- `h` per la minima (2/4);
- `q` per la semiminima (1/4).

Forma numerica

- `1` per la semibreve (4/4);
- `2` per la minima (2/4);
- `4` per la semiminima (1/4);
- `8` per la croma (1/8);
- `16` per la semicroma (1/16);
- `32` per la biscroma (1/32);
- `64` per la semibiscroma (1/64)

Il risultato ottenuto è il seguente (**fig. 30**):



Apprese le nozioni principali per la creazione di un pentagramma e la sua compilazione tramite gli elementi musicali base con VexFlow, sono state studiate soluzioni ottimali per l'implementazione dell'API all'interno del progetto d'elaborato.

Sono state quindi create delle funzioni minimali in grado di svolgere un compito preciso per ogni procedimento necessario al funzionamento del progetto finale.

4.3.2- Principali funzioni realizzate

Di seguito verranno elencate e descritte le principali funzioni realizzate e incluse nella libreria `RenderIEEE1599.js`.

1. **Funzioni di intestazione e caricamento:**

```
function Intestazione(){}
```

Funzione dedicata all'intestazione iniziale della pagina web. All'interno sono presenti i comandi di caricamento dei metadati del documento IEEE1599 e la stampa degli stessi tramite comando `document.write("")`.

```
function CreateDocument(){}
```

Funzione dedicata alla creazione e il caricamento per il parsing di documenti XML.

```
function LoadDocumentIEEE1599(docName){}
```

Funzione dedicata al caricamento di documenti IEEE1599 di nome `docName`, inserito dall'utente tramite comando `prompt("")`. Utilizzato per velocizzare la fase di test.

```
function LoadStaff(staffXML) {…}
```

Sono presenti 13 funzioni di Load destinate al caricamento degli elementi tramite funzione `getElementsByTagName("")`. In questo caso viene caricato l'elemento Staff.

```
function IEEE1599toVexFlow(){}
```

Funzione dedicata al richiamo del render di documenti IEEE1599 tramite API VexFlow all'interno della pagina web.

2. **Funzioni Render VexFlow:**

```
function MakeMeasure(){}
```

Funzione dedicata alla creazione di misure. All'interno sono presenti i comandi di caricamento delle misure, la creazione della prima misura inserendo chiave, armatura di chiave e tempo, la creazione delle misure successive, la gestione delle legature di valore e il richiamo della funzione `function ChordVX(Voiceg).`

```
function ChordVX(Voiceg){}
```

Funzione dedicata alla gestione di accordi, note, pause, legature e punti di valore. I diversi elementi vengono gestiti a livello generico (posizionamento ed inserimento nella misura) e saranno gestiti a livello specifico (altezza, durata, alterazione) dalle funzioni `NoteChord(chord) , DurationChord(chord) e RestVX(rest).`

```
function NoteChord(chord){}
```

Funzione dedicata alla gestione delle note e degli accordi. In particolare della loro altezza e della presenza o meno di punti di valore e alterazioni.

```
function DurationChord(chord){}
```

Funzione dedicata alla durata degli accordi.

```
function RestVX(rest){}
```

Funzione dedicata alla gestione delle pause, in particolare della loro posizione sullo spartito e la loro durata.

```
function ReTie() {}
```

Funzione dedicata al rilevamento di legature di valore all'interno di un elemento <notehead>.

```
function RivActualAccidental(chord) {}
```

Funzione dedicata al rilevamento di alterazioni negli elementi <notehead>.

4.3.3- Gestione delle risorse e delle funzioni

Il legame tra una funzione e l'altra è di rilevante importanza per il corretto funzionamento del progetto. Qui di seguito saranno esposti piccoli stralci di codice per descrivere la gestione delle funzioni all'interno della seconda versione del codice.

All'interno della pagina Web viene richiamata la funzione `IEEE1599toVexFlow()` per richiamare l'intera applicazione. Per aver maggior interazione tra utente e pagina Web è stato inserito il comando `prompt()` in grado di inserire il nome del documento interessato da renderizzare:

```
var documentname = prompt("Inserire il nome del file IEEE1599", "beethoven_gottes_macht.xml");
var ElementiXML = LoadDocumentIEEE1599(documentname);
```

E' stato inserito il documento `beethoven_gottes_macht.xml` come risposta di default per facilitare il collaudo dell'applicazione.

Successivamente viene creata la variabile `ElementiXML` che conterrà tutti gli elementi del documento di nome `documentname`.

Una volta caricato il documento, viene richiamata la funzione `Intestazione()` per la stampa dei principali metadati del documento IEEE1599.

Vengono caricati successivamente i principali elementi XML:

```
part = LoadPart(ElementiXML);
clef = LoadClef(ElementiXML);
time = LoadTime(ElementiXML);
key = LoadKeySignature(ElementiXML);
staff = LoadStaff(ElementiXML);
```

Successivamente è stato creato un ciclo in grado di stampare ogni tre misure le parti e le voci presenti in partitura. Si tratta di un ciclo puramente estetico creato per l'impaginazione della pagina.

Segue poi il ciclo delle parti e il caricamento delle voci di ogni parte:

```
for (var t = 0; t < part.length; t++) {
                    var VoiceItem = LoadVoiceItem(part[t]);
```

In questo ambito viene stampato il nome della parte e creato l'elemento `<canvas>`:

```
document.write("<b>");
document.write( part[t].getAttribute("id") );
document.write( "</b><br/>" + " <canvas width=1000 height=" );
document.write( VoiceItem.length * 120 + "></canvas><br/>");

var canvas = $("canvas")[Canvas];
var renderer = new Vex.Flow.Renderer(canvas,
Vex.Flow.Renderer.Backends.CANVAS);
var ctx = renderer.getContext();
Canvas++;
```

Dove:

- l'attributo `height` dell'elemento `<canvas>` è direttamente proporzionale al numero di voci presenti all'interno di una parte. Questo stratagemma viene utilizzato per avere un canvas dinamico ed evitare pentagrammi nascosti;

- la variabile `Canvas` tra parentesi quadre indica il numero di canvas creato e il numero identificativo della variabile `canvas` (in minuscolo);

Segue il codice:

```
for (g = 0 ; g < VoiceItem.length ; g++) {
      var stave = new Array();
      Measure = LoadMeasure(part[t]);
      MakeMeasure();
      ReTie();
}
```

Dove:

- inizia il ciclo delle voci;
- viene creato l'Array stave, nel quale saranno inserite le diverse voci;
- viene caricata la misura della parte di riferimento;
- viene richiamata la funzione `MakeMeasure()` e `ReTie();`

La gestione delle misure avviene similarmente alla prima versione del codice, ma verranno caricate tre misure alla volta:

```
for (m = 0; m < 3; m++) {

Voice = LoadMVoice(Measure[m + Imp]);
wmc = WidthMeasureCalculator(Voice[g]);
widthmeasure = 100+wmc;
```

Dove:

- Imp è il puntatore della misura d'impaginazione, cioè il numero reale di misure già renderizzate;
- wmc e la funzione `WidthMeasureCalculator(Voice[g])` servono per render dinamico l'elemento `stave` in larghezza. In particolare, verranno calcolati gli elementi presenti all'interno della misura e, di conseguenza, moltiplicati per la larghezza della misura creata.

Per ogni misura viene creato un elemento stave. In pratica ogni misura è un nuovo pentagramma:

```
if (m == 0) {
stave[g] = MakeStaff(g);
stave[g].setContext(ctx).draw();
f = StaffRef(VoiceItem[g].getAttribute("staff_ref"));

ClefVF(clef[f].getAttribute("shape"));
KeySignature(key[f]);
TimeSignature(time[f].getAttribute("num"),
time[f].getAttribute("den"));

}
```

La condizione `m == 0` indica che nella prima misura dovrà :

- essere creato un elemento `stave` ed assegnato al contesto;
- renderizzati i glifi delle chiavi, armature di chiave e tempi corrispondenti alla voce di riferimento `f`.

Segue il codice:

```
else if (g < 1)
stave[m] = new Vex.Flow.Stave(stave[m - 1].width + stave[m - 1].x,
stave[m - 1].y, widthmeasure).setContext(ctx).draw();

else
stave[m + g] = new Vex.Flow.Stave(stave[m].width + stave[m].x,
stave[m].y, widthmeasure).setContext(ctx).draw();
```

Dove :

- la condizione g < 1 si avvera quando siamo nella prima voce della parte di riferimento. In questo modo la misura sarà renderizzata di seguito alla precedente;
- il puntatore g viene utilizzato come controllo per renderizzare le voci successive alla prima ad un'altezza differente.

Successivamente viene richiamata la funzione `ChordVX(Voice[g])` associando il contenuto di ritorno ad una `Array() notes`.

La funzione `ChordVX(Voice[g])` si compone di numerose righe di codice. Qui di seguito verranno mostrati i principali passaggi gestiti:

- vengono inizializzate le variabili necessarie per la strutturazione del contenuto (nota, durata, pausa, accordo, altezza, alterazione, indicatore di pausa e indicatore di accordo);
- inizio ciclo voci;
- controllo elemento: pausa o accordo;
- viene assegnato il valore `b/4` al pitch degli elementi pausa e rilevata la durata prelevandola dagli attributi `den` e `num` dei rispettivi elementi `<duration>;`
- se è un elemento nota, vengono rilevate le alterazioni presenti nell'accordo di riferimento. Successivamente viene richiamata la funzione `NoteChord(chord[n_chord])` per l'assegnazione dei rispettivi valori di `pitch` e `duration` di ogni nota che compone l'accordo;
- infine viene strutturata la variabile da restituire ed aggiunti punti valore e alterazioni se presenti;
- la variabile sarà strutturata in questo modo: `outputnote[outputcount] = new Vex.Flow.StaveNote({ keys: note, duration: duration });` dove `note` è la variabile che descrive il pitch della nota/pausa e `duration` è la variabile che descrive la durata e il tipo di elemento (nota o pausa).

Infine, restituiti tutti gli elementi di una misura alla variabile notes, vengono gestite le legature di valore tra gli elementi di una misura, e tutti gli elementi vengono renderizzati nell'elemento `<canvas>.`

4.3.4- Impaginazione ed elementi di CSS

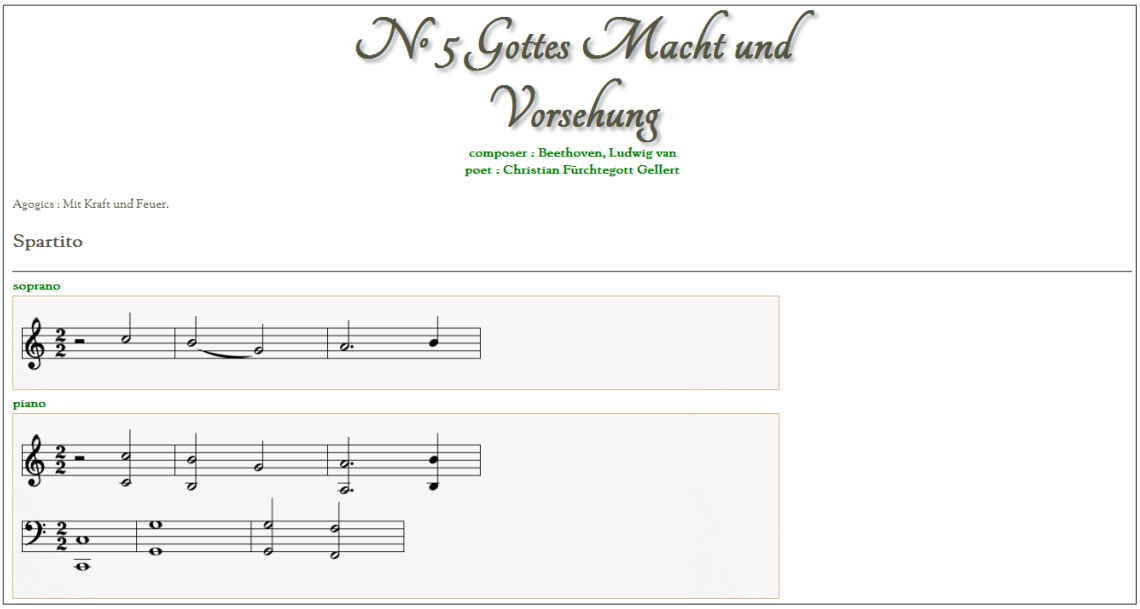


Fig.31 Impaginazione secondo progetto.

Nel corso del capitolo 4.3.3 sono stati già accennati elementi d'impaginazione all'interno del codice:

- vengono renderizzate tre battute per volta;
- viene creato un nuovo elemento canvas per ogni parte;
- l'altezza dell'elemento canvas è direttamente proporzionale al numero delle voci presenti in una parte;
- la larghezza delle misure è direttamente proporzionale al numero di elementi presenti al suo interno.

Queste scelte sono state fatte dopo numerosi collaudi dell'applicazione con diversi documenti IEEE1599. In molti casi, se all'interno di una battuta erano presenti numerosi elementi, si verificava un errore di rendering dovuto al "sovraffollamento" delle battute. Per questo motivo si è scelto di utilizzare una variabile in grado di allargare le battute più "affollate" e restringere quelle più spoglie.

Questa soluzione, però, ha portato un altro problema: le battute più affollate spingevano le battute spoglie fuori dal canvas, nascondendo parti di partitura. Per questo motivo, è stato scelto di renderizzare solo tre battute per volta.

Il risultato è stato quello illustrato in **fig.31**.

Non avendo completo controllo sull'impaginazione degli elementi stave all'interno del canvas, il risultato d'impaginatura non è ottimo. Tuttavia, i problemi che portavano il malfunzionamento dell'applicazione, sono stati risolti.

Inoltre, sono stati utilizzati elementi di CSS per distaccarsi dallo stile utilizzato da 0xfe nel sito www.vexflow.com:

Fig.32 Stampa metadati.

È stato utilizzato il font "`a song for jennifer`" per la parte dei metadati, aggiunto l'effetto ombra al titolo dell'opera e diversificato lo stile dei caratteri degli autori con quelli dell'agogica.

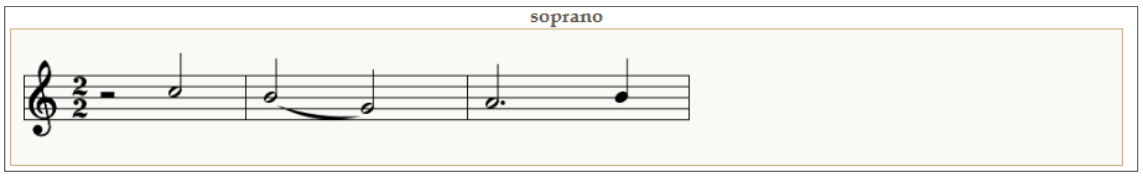


Fig.33 Stampa canvas.

L'elemento canvas è stato personalizzato con background e contorni.

4.3.5- Test dell'applicazione

Fig.34 Messaggio iniziale dell'applicazione

Per testare la bontà dell'applicazione sono stati presi a campione diversi file IEEE1599 con caratteristiche e particolarità diverse. Per semplificare questa fase è stato inserito all'apertura della pagina il metodo prompt(). In questo modo, inserendo il nome del documento da testare, l'applicazione caricherà il relativo file automaticamente.

Di seguito verranno mostrati i risultati dell'applicazione utilizzando diversi documenti IEEE1599 illustrando i problemi riscontrati in fase di analisi:

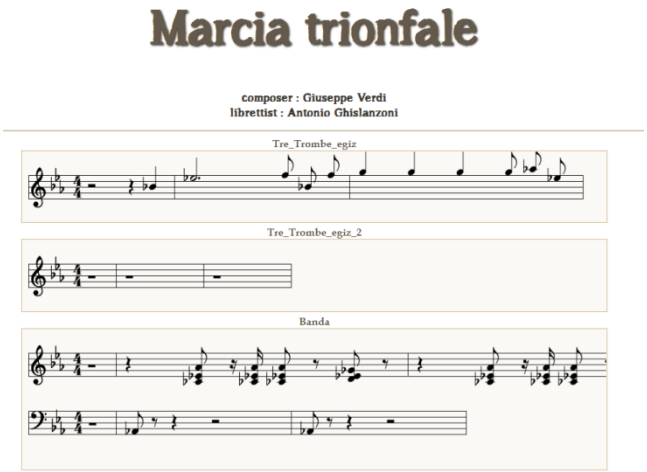


Fig. 35 Risultato del documento: "Aida_Marcia_trionfale.xml".

L'applicazione funziona bene, ma la presenza numerosa di elementi all'interno di ogni battuta, porta, ancora, fuori tela la partitura (come visibile nell'immagine qui a fianco).

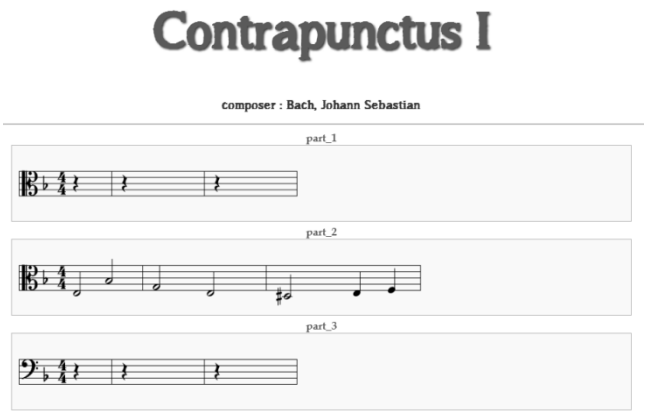


Fig. 36 Risultato del documento: "bach_artefuga_01.xml".

Buona risposta dell'applicazione ma stessi disturbi già citati nel caso precedente.

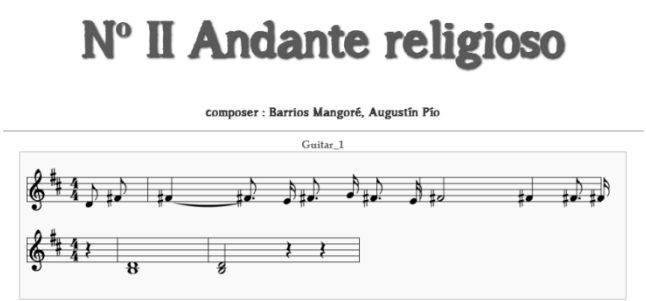


Fig. 37 Risultato del documento: "barrios_catedral.xml".

Il disturbo persiste ma gli elementi renderizzati sono fedeli alla partitura cartacea.

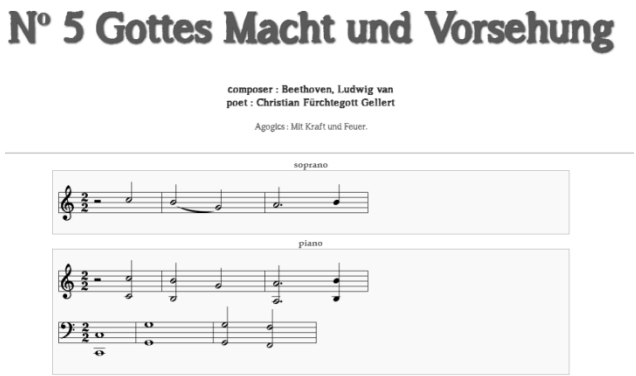


Fig. 38 Risultato del documento: "beethoven_gottes_macht.xml".

L'applicazione è fedele alla partitura originale. Il funziona-mento è ottimale.

4.3.6- Problemi riscontrati durante la fase di collaudo

I risultati ottenuti sono stati soddisfacenti: l'applicazione rappresentava graficamente in modo fedele gli elementi della partitura tralasciando gli elementi più caratteristici di uno spartito stampato (testo cantato, segni di crescendo e decrescendo, glifi forte, medio forte, ecc.). Tuttavia, i numerosi problemi di impaginazione delle partiture non hanno garantito il buon funzionamento dell'applicazione a livello visivo.

Inoltre, l'impossibilità di creare legature di valore tra le battute e la mancata renderizzazione di gruppi irregolari, hanno portato alla rivisitazione dell'intero codice utilizzando VexTab come renderer di documenti IEEE1599.

4.4- Terza versione: da VexFlow a VexTab

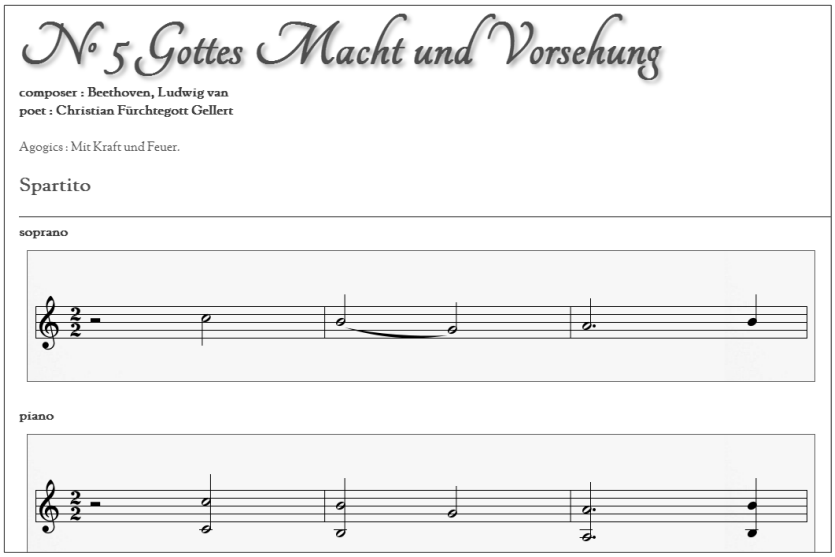


Fig.39 Versione finale dell'Applicazione

Qui di seguito saranno illustrate le principali differenze tra la seconda e l'ultima versione del progetto. L'utilizzo di VexTab ha implicato la completa riscrittura del codice di gestione del renderer. Tuttavia è stato utilizzato lo stesso scheletro delle precedenti versioni.

4.4.1- Principali differenze tra VexFlow e VexTab

L'utilizzo di un interfaccia più semplice ed intuitiva ha permesso una programmazione ed una gestione degli elementi più semplice e veloce:

- mentre in VexFlow bisogna utilizzare metodi e funzioni per la creazione di elementi grafici su Canvas, in VexTab bisogna semplicemente gestire il contenuto di un elemento .

```
document.write("<div style='width:700;'>");
document.write('<div  class="vex-tabdiv"  width=1000  scale=1.0  editor="false" editor_width=680 editor_height=300>');
```

Inoltre, è possibile utilizzare la funzione editor="true" per visualizzare il contenuto dell'elemento come in **fig. 40**.

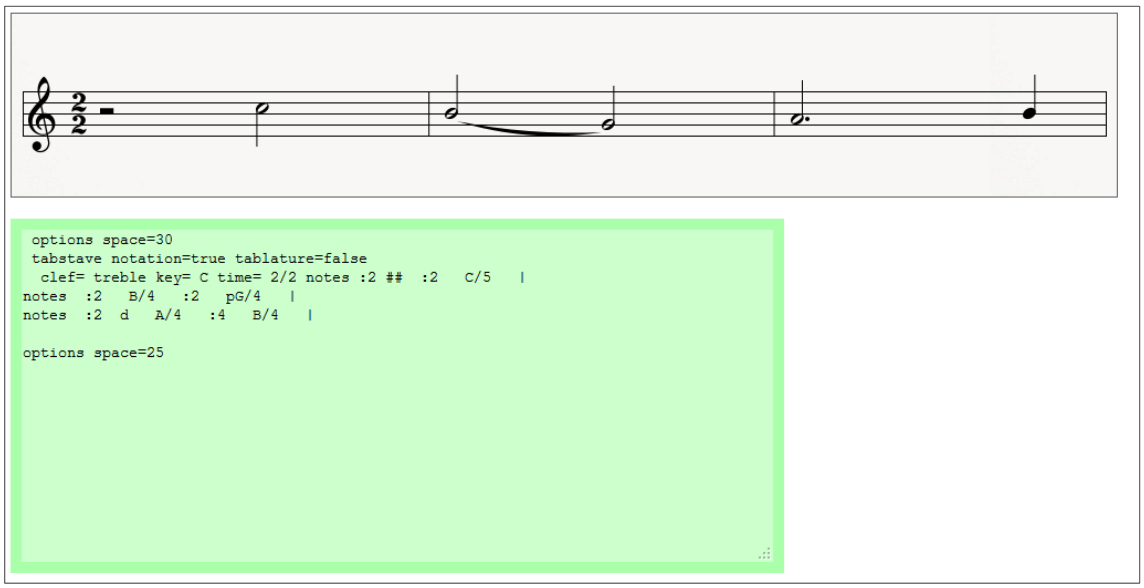


Fig.40 Editor contenuto elemento

Questo è permesso dalla libreria tabdiv2.js che gestisce il contenuto di un elemento includendo un Canvas HTML5 e gestendo i comandi VexFlow tramite dei comandi ASCII.

Inoltre, con l'attributo `tablature=true` è possibile vedere la tablatura corrispettiva dello spartito inserito all'interno degli elementi .

Dovendo gestire dei contenuti testuali, l'idea della prima versione del codice si è rivelata molto utile. Utilizzandolo sempre come scheletro, è stata riscritta una nuova libreria con funzioni in grado di soddisfare le nuove esigenze:

```
function MakeStaff(g) {
document.write(" options space=30 &#x0D tabstave notation=true tabla-
ture=false &#x0D "); }
```

A titolo d'esempio, la funzione MakeStaff(g) non crea più una variabile stave ma stampa all'interno degli elementi l'inizio di un nuovo spartito.

Inoltre, come si può notare dall'esempio appena proposto, abbiamo una attributo `options space=30` con il quale poter gestire meglio lo spazio occupato da una voce, oppure il carattere `&#x0D` (a capo) deve essere necessariamente inserito per il corretto funzionamento dell'applicazione, ma permette di posizionare al meglio ogni pentagramma.

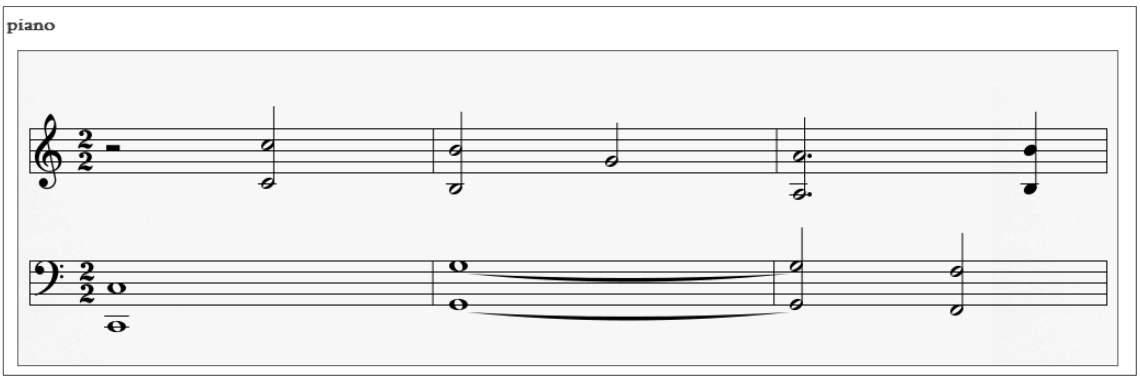


Fig.41 Partitura renderizzata dall'applicazione finale

Il codice dell'intero progetto finale è stato inserito in Appendice A.

4.4.2- Miglioramenti apportati

Oltre alla semplicità nel gestire molteplici informazioni musicali inserendo semplici comandi letterali, l'utilizzo di VexTab ha migliorato l'aspetto grafico dell'elaborato, garantendo una impaginazione ordinata e pulita.

Inoltre, l'implementazione di nuove funzioni, come le legature di valore tra note di diversa misura, impossibili nella seconda versione, oppure l'inserimento della funzione MakeTuplet(); che risponde bene alla creazione di terzine presenti all'interno di uno spartito.

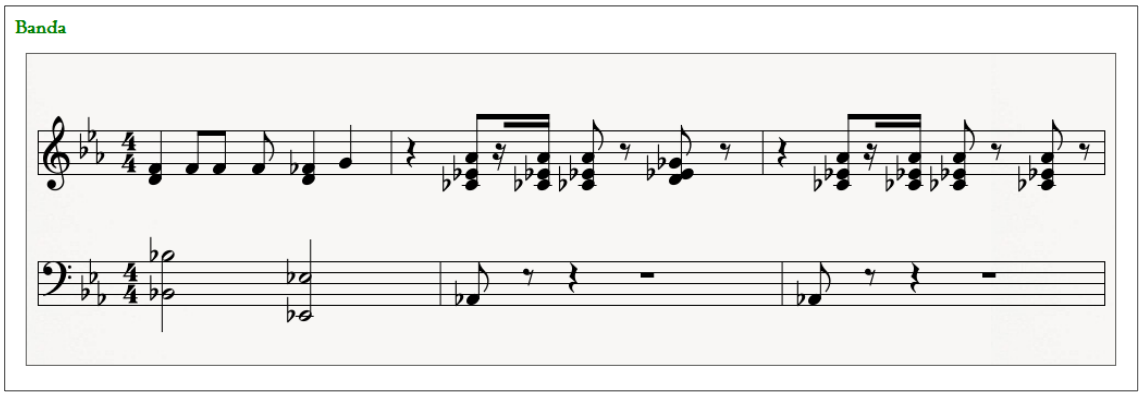


Fig.42 Inserimento ordinato degli elementi

La presenza di tanti elementi in una misura non crea più problemi a livello grafico, permettendo la completa visione della partitura.

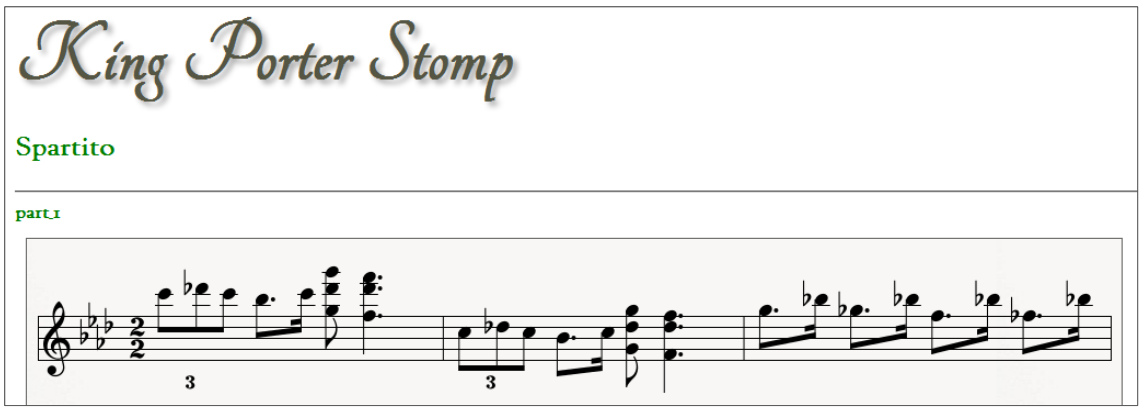


Fig.43 Esempio render di gruppi irregolari

5- Conclusioni e sviluppi futuri

La terza versione del codice, migliora gli aspetti della seconda ed incrementa le possibilità della prima permettendo di utilizzare un'interfaccia semplice e di facile modifica.

Tuttavia, sono presenti ancora qualche lacune nella gestione dell'informazione musicale offerta da documenti IEEE1599: ad esempio, la funzione dedicata alla renderizzazione di gruppi irregolari, funziona solo con gruppi di terzine.

Per questi motivi, gli sviluppi futuri si concentreranno al miglioramento del software sia a livello di codice che di risorse, garantendo una maggior stabilità con lo standard IEEE1599. In un momento successivo, sarebbe interessante implementare la sincronizzazione dello spartito musicale renderizzato con i relativi file musicali collegati al documento IEEE1599. Altra strada percorribile è quella di realizzare un applicazione in grado di creare un documento IEEE1599 utilizzando un editor basato su VexFlow direttamente dal Web.

APPENDICE A

Terza Versione del Codice

Documento HTML

```
<html>
<head>
  <title>Render IEEE1599 - VexTab</title>
  <link-
href='http://fonts.googleapis.com/css?family=OFL+Sorts+Mill+Goudy+TT|Yanone+Kaffe
esatz|Tangerine'
   rel='stylesheet' type='text/css'>
  <link href="tabdiv.css" media="screen" rel="Stylesheet" type="text/css" />
  <link href="style.css" media="screen" rel="Stylesheet" type="text/css" />

  <!-- Support Sources -->
  <script src="../src/RenderIEEE1599-VexTab.js"></script>
  <script src="../support/vexflow-min.js"></script>
  <script src="../support/underscore-min.js"></script>
  <script src="../support/jquery.js"></script>
  <script src="../support/tabdiv-min.js"></script>

</head>

<body> <div style=" text-align:left ">

<script type="text/javascript">
RenderIEEE1599();
</script>

</div>
</body>

</html>
```

Libreria: RenderIEEE1599-VexTab.js

```
function RenderIEEE1599() {
    var documentname = prompt("Inserire il nome del file IEEE1599", "beetho-
ven_gottes_macht.xml");
    var ElementiXML = LoadDocumentIEEE1599(documentname);
    Intestazione();
    part = LoadPart(ElementiXML);
    clef = LoadClef(ElementiXML);
    time = LoadTime(ElementiXML);
    key = LoadKeySignature(ElementiXML);
    staff = LoadStaff(ElementiXML);
    var MeasureCount = 0;
    var MeasureLengthControl = LoadMeasure(part[0]);
    for (Imp = 0; Imp < MeasureLengthControl.length; Imp += 3) {
        var RevTie = false;
        var SecondRevTie = false;
        var ChordTie = 0;
        var MeasureTie = 0;
        var NoteTie = 0;
        var revtuplet = false;
        var ntuplet = 0;
        var Note;
        stuplet = 0;
        notes = Array();
        var f = 0;
        document.write("<hr/>");
        for (var t = 0; t < part.length; t++) {
            var VoiceItem = LoadVoiceItem(part[t]);
            document.write("<b>" + part[t].getAttribute("id") + "</b><br/>");
            document.write("<div style='width:700;'>");
            document.write('<div class="vex-tabdiv" ');
            document.write('width=1000 scale=1.0 editor="true");
            document.write("editor_width=680 editor_height=300>');
            for (g = 0; g < VoiceItem.length; g++) {
                Measure = LoadMeasure(part[t]);
                MakeMeasure();
                document.write("&#x0D");
            }
            document.write("options space=25");
            document.write('</div>');
            document.write('</div>');
            document.write("<br/>");
        }
    }
}

function Intestazione() {
    var number = ElementiXML.getElementsByTagName('number');
    var main_title = ElementiXML.getElementsByTagName('main_title');
    var author = ElementiXML.getElementsByTagName('author');
    var agogics = ElementiXML.getElementsByTagName('agogics');
    document.write('<h1>');
    if (number.length > 0) {
        document.write("N&#186 " + num - ber[0].childNodes[0].nodeValue);
    }
    if (main_title.length > 0) {
        document.write(" " + main_title[0].childNodes[0].nodeValue + '</h1> <b>');
    }
    if (author.length > 0) {
        for (var i = 0; i < author.length; i++) {
            var author_type = author[i].attributes;
```

```
            document.write(author_type[0].value + " : " + au-
thor[i].childNodes[0].nodeValue + "<br />");
        }
    }
    document.write('</b>');
    if (agogics.length > 0) {
        document.write("<br />Agogics : ");
        document.write( agogics[0].childNodes[0].nodeValue + "<br />");
    }
}
//Funzioni creazione documento

function CreateDocument() {
    var documentoXML = document.implementation.createDocument("", "", null);
    documentoXML.async = false;
    return documentoXML;
}
// Funzioni gestione elementi VexFlow

function MakeTuplet() {
    if (stuplet > 0) {
        document.write(" ^" + ntuplet + "^ ");
        revtuplet = false;
        ntuplet = 0;
        stuplet = 0;
    }
}

function StopTie() {
    SecondRevTie = false;
    RevTie = false;
    MeasureTie = 0;
    ChordTie = 0;
    NoteTie = 0;
}

function RestVX(rest) { //Gestione pause
    var restvx = LoadRest(rest);
    var duration = ":" + rest.childNodes[1].getAttribute("den");
    return duration;
}

function DurationChord(chord) //Gestione Durata Accordi
{
    var DurationChord = LoadDurationChord(chord);
    var tuplet = DurationChord[0].getElementsByTagName('tuplet_ratio');
    if (revtuplet == true) {
        if (tuplet.length > 0) {
            revtuplet = true;
            ntuplet++;
        }
        else {
            revtuplet = false;
            stuplet = ntuplet;
        }
    }
    if (ntuplet == 0) {
        if (tuplet.length > 0) {
            revtuplet = true;
            ntuplet++;
        }
    }
    var Duration = " :" + DurationChord[0].getAttribute("den") + " ";
    return Duration;
}
```

```
function NoteChord(chord) {
    var Notehead = LoadNotehead(chord);
    var Pitch;
    var Octave;
    for (var nl = 0; nl < Notehead.length; nl++) {
        tie = Notehead[nl].getElementsByTagName('tie');
        if (NoteTie == nl) {
            if ((ChordTie != n_chord) || (MeasureTie != m)) {
                if ((RevTie == true) && (tie.length > 0)) {
                    SecondRevTie = true;
                }
            }
        }
        if (RevTie == false) {
            if (tie.length > 0) {
                ChordTie = n_chord;
                MeasureTie = m;
                NoteTie = nl;
                RevTie = true;
            }
        }
        Pitch = LoadPitch(Notehead[nl]);
        Octave = Pitch[0].getAttribute("octave") - 1;
        if ((SecondRevTie == true) && (RevTie == true)) {
            StopTie();
            if ((nl == 0) && (Notehead.length > 1)) {
                if (Pitch[0].getAttribute("actual_accidental") == "sharp") {
                    Note = "p(" + Pitch[0].getAttribute("step") + "#/" + Octave+ ".";
                }
                else if (Pitch[0].getAttribute("actual_accidental") == "flat") {
                    Note = "p(" + Pitch[0].getAttribute("step") + "@/" + Octave+ ".";
                }
                else {
                    Note = "p(" + Pitch[0].getAttribute("step") + "/" + Octave+ ".";
                }
            }
            else if ((nl == 0) && (Notehead.length == 1)) {
                if (Pitch[0].getAttribute("actual_accidental") == "sharp") {
                    Note = " p" + Pitch[0].getAttribute("step") + "#/" + Octave+ " ";
                }
                else if (Pitch[0].getAttribute("actual_accidental") == "flat") {
                    Note = " p" + Pitch[0].getAttribute("step") + "@/" + Octave+ " ";
                }
                else {
                    Note = " p" + Pitch[0].getAttribute("step") + "/" + Octave + " ";
                }
            }
            else if (nl < (Notehead.length - 1)) {
                if (Pitch[0].getAttribute("actual_accidental") == "sharp") {
                    Note =" "+Note+Pitch[0].getAttribute("step")+"#/" + Octave + ".";
                }
                else if (Pitch[0].getAttribute("actual_accidental") == "flat") {
                    Note =" "+Note+Pitch[0].getAttribute("step")+"@/" + Octave + ".";
                }
                else {
                    Note = " "+Note+Pitch[0].getAttribute("step")+"/" + Octave + ".";
                }
            }
            else {
                if (Pitch[0].getAttribute("actual_accidental") == "sharp") {
                    Note =" "+Note+Pitch[0].getAttribute("step")+"#/"+ Octave + ") ";
                }
                else if (Pitch[0].getAttribute("actual_accidental") == "flat") {
                    Note =" "+Note+Pitch[0].getAttribute("step")+"@/"+Octave + ") ";
                }
```

```
                else {
                    Note =" "+Note+Pitch[0].getAttribute("step")+"/" + Octave + ") ";
                }
            }
        }
        else {
            if ((nl == 0) && (Notehead.length > 1)) {
                if (Pitch[0].getAttribute("actual_accidental") == "sharp") {
                    Note = "(" + Pitch[0].getAttribute("step") + "#/" + Octave + ".";
                }
                else if (Pitch[0].getAttribute("actual_accidental") == "flat") {
                    Note = "(" + Pitch[0].getAttribute("step") + "@/" + Octave + ".";
                }
                else {
                    Note = "(" + Pitch[0].getAttribute("step") + "/" + Octave + ".";
                }
            }
            else if ((nl == 0) && (Notehead.length == 1)) {
                if (Pitch[0].getAttribute("actual_accidental") == "sharp") {
                    Note = " " + Pitch[0].getAttribute("step") + "#/" + Octave + " ";
                }
                else if (Pitch[0].getAttribute("actual_accidental") == "flat") {
                    Note = " " + Pitch[0].getAttribute("step") + "@/" + Octave + " ";
                }
                else {
                    Note = " " + Pitch[0].getAttribute("step") + "/" + Octave + " ";
                }
            }
            else if (nl < (Notehead.length - 1)) {
                if (Pitch[0].getAttribute("actual_accidental") == "sharp") {
                    Note =Note + Pitch[0].getAttribute("step") + "#/" + Octave + ".";
                }
                else if (Pitch[0].getAttribute("actual_accidental") == "flat") {
                    Note =Note + Pitch[0].getAttribute("step") + "@/" + Octave + ".";
                }
                else {
                    Note =Note + Pitch[0].getAttribute("step") + "/" + Octave + ".";
                }
            }
            else {
                if (Pitch[0].getAttribute("actual_accidental") == "sharp") {
                    Note =Note+ Pitch[0].getAttribute("step") + "#/" + Octave + ") ";
                }
                else if (Pitch[0].getAttribute("actual_accidental") == "flat") {
                    Note =Note+ Pitch[0].getAttribute("step") + "@/" + Octave + ") ";
                }
                else {
                    Note = Note+ Pitch[0].getAttribute("step") + "/" + Octave + ") ";
                }
            }
        }
    }
    return Note;
}
```

```
function ChordVX(Voiceg) {
    var note = Array();
    var duration;
    var outputnote = Array();
    var outputcount = 0;
    var rest = LoadRest(Voiceg);
    var chord = LoadChord(Voiceg);
    var Pitch;
    var Ac = Array();
    var n_rest = 0;
    n_chord = 0;
    for (var Vl = 1; Vl < Voiceg.childNodes.length; Vl += 2) {
        var aug_dot = false;
        if (Voiceg.childNodes[Vl].nodeName == "rest") {
            if (clef[f].getAttribute("shape") == "F") {
                duration = RestVX(rest[n_rest]);
                document.write(duration + " #5# ");
            }
            else if (clef[f].getAttribute("shape") == "C") {
                duration = RestVX(rest[n_rest]);
                document.write(duration + " #5# ");
            }
            else {
                duration = RestVX(rest[n_rest]);
                document.write(duration + " ## ");
            }
            n_rest++;
        }
        else if (Voiceg.childNodes[Vl].nodeName == "chord") {
            note = NoteChord(chord[n_chord]);
       for (var punto_c = 0; punto_c < chord[n_chord].childNodes.length; punto_c++) {
                if (chord[n_chord].childNodes[punto_c].nodeName == "augmen-
tation_dots") {
                    aug_dot = true;
                }
            }
            duration = DurationChord(chord[n_chord]);
            MakeTuplet();
            if (aug_dot == true) {
                duration = duration + " d ";
            }
            document.write(duration + " " + note + " ");
            n_chord++;
        }
        outputcount++;
    }
}

function MakeStaff(g) {
    document.write(" options space=30 &#x0D ");
    document.write("tabstave notation=true tablature=false &#x0D ");
}

function StaffRef(idStaff) {
    var id = 0;
    for (var ContID = 0; ContID < staff.length; ContID++) {
        if (idStaff == staff[ContID].getAttribute("id")) {
            id = ContID;
        }
    }
    return id;
}

function MakeMeasure() {
    StopTie();
    for (m = 0; m < 3; m++) {
```

```
        Voice = LoadMVoice(Measure[m + Imp]);
        if (m == 0) {
            MakeStaff();
            f = StaffRef(VoiceItem[g].getAttribute("staff_ref"));
            ClefVF(clef[f].getAttribute("shape"));
            KeySignature(key[f]);
            TimeSignature(time[f].getAttribute("num"), time[f].getAttribute("den"));
        }
        document.write("notes ");
        ChordVX(Voice[g]);
        document.write(" | ");
        document.write("&#x0D");
    }
}

function TimeSignature(TsNUM, TsDEN) {
    document.write("time= " + TsNUM + "/" + TsDEN + " "); //Tempo
}

function ClefVF(Cf) {
    switch (Cf) {
        case "G":
            document.write(" clef= treble ");
            break;
        case "F":
            document.write(" clef= bass ");
            break;
        case "C":
            document.write(" clef= alto ");
            break;
    }
}

function KeySignature(Ks) { // Armatura di chiave
    var flat = Ks.getElementsByTagName("flat_num");
    var sharp = Ks.getElementsByTagName("sharp_num");
    if (flat.length > 0) {
        switch (flat[0].getAttribute("number")) {
            case "0":
                document.write("key= C ");
                break;
            case "1":
                document.write("key= F ");
                break;
            case "2":
                document.write("key= Bb ");
                break;
            case "3":
                document.write("key= Eb ");
                break;
            case "4":
                document.write("key= Ab ");
                break;
            case "5":
                document.write("key= Db ");
                break;
            case "6":
                document.write("key= Gb ");
                break;
            case "7":
                document.write("key= Cb ");
                break;
        }
    }
```

```
 else if (sharp.length > 0) {
        switch (sharp[0].getAttribute("number")) {
            case "0":
                document.write("key= C ");
                break;
            case "1":
                document.write("key= G ");
                break;
            case "2":
                document.write("key= D ");
                break;
            case "3":
                document.write("key= A ");
                break;
            case "4":
                document.write("key= E ");
                break;
            case "5":
                document.write("key= B ");
                break;
            case "6":
                document.write("key= F# ");
                break;
            case "7":
                document.write("key= C# ");
                break;
        }
    }
}
//-- Funzioni di caricamento elementi

function LoadDocumentIEEE1599(docName) {
    var documentoXML = CreateDocument();
    var ElementiXML;
    documentoXML.load(docName);
    return ElementiXML = documentoXML.documentElement;
}

function LoadStaff(stafXML) {
    return stafXML.getElementsByTagName('staff');
}

function LoadPart(partXML) {
    return partXML.getElementsByTagName('part');
}

function LoadClef(clefXML) {
    return clefXML.getElementsByTagName('clef');
}

function LoadTime(timeXML) {
    return timeXML.getElementsByTagName('time_indication');
}

function LoadKeySignature(ksXML) {
    return ksXML.getElementsByTagName('key_signature');
}

function LoadVoiceItem(viXML) {
    return viXML.getElementsByTagName('voice_item');
}

function LoadMeasure(measureXML) {
    return measureXML.getElementsByTagName('measure');
}
```

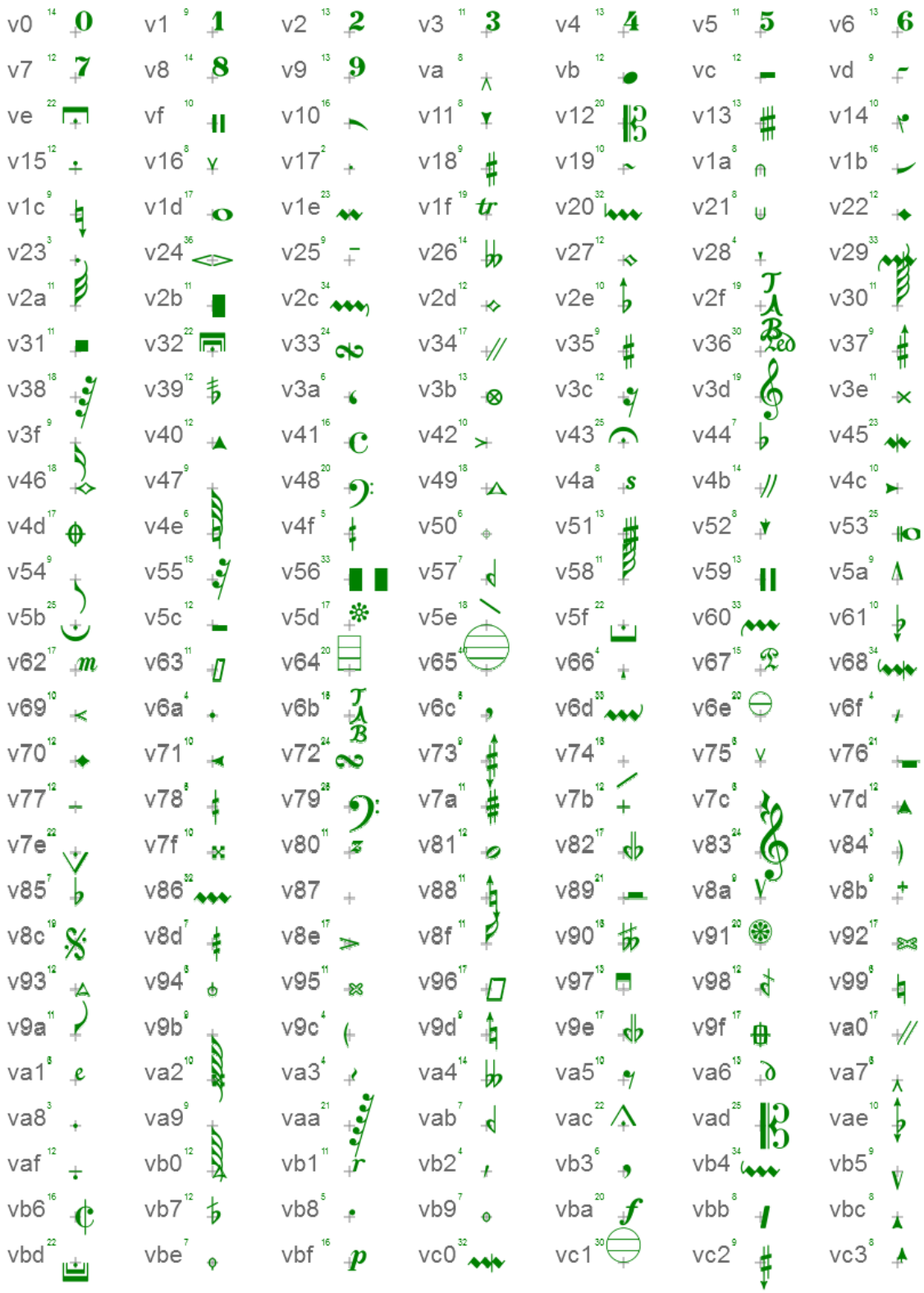
```
function LoadMVoice(MVoiceXML) {
    return MVoiceXML.getElementsByTagName('voice');
}

function LoadRest(restXML) {
    return restXML.getElementsByTagName('rest');
}

function LoadChord(chordXML) {
    return chordXML.getElementsByTagName('chord');
}

function LoadDurationChord(dcXML) {
    return dcXML.getElementsByTagName('duration');
}

function LoadNotehead(nhXML) {
    return nhXML.getElementsByTagName('notehead');
}

function LoadPitch(pitchXML) {
    return pitchXML.getElementsByTagName('pitch');
}
```

APPENDICE B

Tabella dei glifi presenti in VexFlow

Vex Glyphs

Cross indicates render coordinates.



Bibliografia

[1] Statistiche condotte da Eurostat sulla società dell'informazione: "Level of Internet access – households" 2006/2012 -. Sono state considerate le età comprese tra i 16 e i 72 anni:dalla fine del 2006 al 2012, la media delle famiglie aventi almeno un accesso internet di tutti i paesi dell'UE aumenta del 24%, raggiungendo il 74% di tutte le famiglie europee.

[11] Pessa, G.: "MusicXML, Nascita di un nuovo Standard Musicale. Prospettive per un futuro incerto" - Università di Bologna 2008

[12]Baratè, A., Ludovico, L.A., Mauro, D.A.: "Tecnologie basate su XML per la fruizione avanzata dei contenuti musicali". In: Tanlongo, F., Vario, M., Volpe, C. (eds.) Conferenza GARR_09

[13] Haus, G., Ludovico, L. A.: "Music Segmentation: an XML-Oriented Approach". Proc. Computer Music Modeling and Retrieval - CMMR 2004

[14]Ludovico, L.A.: "IEEE 1599: a Multi-layer Approach to Music Description". Journal of Multimedia 4(1), 9–14 (2009)

[15]Baratè, A., Haus, G., Ludovico, L.A.: "Music representation of score, sound, MIDI, structure and metadata all integrated in a single multilayer environment based on XML". In: Cui, B., Liu, L., Shen, J., Shepherd, J. (eds.) Intelligent music information systems : tools and methodologies. pp. 305–328. Information Science Reference, Hershey (2008)

[19] Pilgrim, M.: "HTML5: Guida operativa. Immergersi nel futuro dello sviluppo Web" - Tecniche Nuove (2011)

[20] Rubini, S.: "JavaScript" - APOGEO 2008

[21]Flanagan, D.: “JavaScript Versione 1.5 - La Guida”- APOGEO 2004

[23]Eisenberg, J.d.: “SVG Essenzials, Producing Scalable Vector Graphics With XML” - O’Reilly & Associates 2002

[26] Amedeo, E.: “JQuery. La potenza di JavaScript in poche eleganti righe di codice” - APOGEO 2012

[28] Mancini, F.: “Metologie e Tecniche per l’Editoria Musicale” - dispense per il corso di “Tecnologie Informatiche per l’Editoria Musicale” 2010

[29] Piston, W.: “Armonia” - EDT s.r.l., 1989

Sitografia

[2]http://www.vexflow.com/

[3]http://0xfe.blogspot.it/

[4]http://vexflow.com/vextab/

[5]http://emipiu.di.unimi.it/

[6]http://www.didaelkts.it/

[7]http://music-notation-software-review.toptenreviews.com/

[8]http://www.noteflight.com/info/about

[9]http://www.noteflight.com/info/terms_of_use

[10]http://www.gregjopa.com/

[16]http://www.w3c.it/it/3247/informazioni-sul-w3c.html

[17]http://www.html.it/pag/19263/da-html-4-ad-html5/

[18]http://www.netmagazine.com/news/w3c-finish-html5-2014-122239

[22]http://www.html.it/pag/15174/introduzione14/

[24]https://github.com/0xfe/vexflow

[25]http://raphaeljs.com/

[27]http://www.pc-facile.com/glossario/user-friendly/

[30]http://www.steveborn.com/codenotes/LoadingXML.htm

[31]http://xml-copy-editor.sourceforge.net/

[32]http://sviluppare-in-rete.blogspot.it/2007/11/javascript-processare-documenti-xml.html