LEADSHEETJS: A JAVASCRIPT LIBRARY FOR ONLINE LEAD SHEET EDITING

Daniel Martín
Sony CSL
dmartinmartinez@gmail.com

Timotée Neullas
Sony CSL
tneullas@gmail.com

François Pachet
Sony CSL
pachetcsl@gmail.com

ABSTRACT

Lead sheets are music scores consisting of a melody and a chord grid, routinely used in many genres of popular music. With the increase of online and portable music applications, the need for easily embeddable, adaptable and extensible lead sheet editing tools is pressing. We introduce LeadsheetJS, a Javascript library for visualizing, editing and rendering lead sheets on multiple devices. LeadsheetJS provides lead sheet editing as well as support for extensions such as score augmentation and peer feedback. LeadsheetJS is a client-based component that can be embedded from arbitrary third-party websites. We describe the main design aspects of LeadsheetJS and some applications in online computer-aided composition tools.

1. INTRODUCTION

A lead sheet is a specific type of music score consisting of a monophonic melody with associated chord labels (*Figure 1*). Lead sheets are routinely used in many styles of popular music such as songwriting, jazz, pop or bossa nova.

With the rise of online music communities using performance or pedagogical applications, there is an increasing need for tools for manipulating music scores. In this context, music notation takes an important role, and in particular lead sheets, which are the main form of score for popular music. There is also a need for web-based tools for visualizing, playing, and editing lead sheets collaboratively. Such tools should also work on various devices, following the trend in using web applications on mobiles and tablets. Finally, these tools should intercommunicate easily with other tools, e.g. by being embeddable in third-party websites.

The most popular score editors, *Finale* and *Sibelius*, are designed as desktop applications. As such they cannot be used online, even though cloud features can be added, e.g. to share scores by exporting them to the web [9]. The open-source desktop-based editor *MuseScore* [1] provides features for sharing scores but does not provide directly online editing. There are many online tools to edit and view scores, but they do not rely on web standards, and often require the installation of a plugin on the web-browser. Some tools, such as *NoteFlight*[2], *Scorio*[3] or *Flat.io*[4], do follow standards and produce machine-readable scores, but they are not designed specifically for lead sheets. For instance, they do not support chord notations, an important feature of a lead sheet.

Besides offering basic score editing services, online lead sheet tools should provide features for *augmented editing*, e.g. to be tailored to pedagogical or social contexts. The ability of adding heterogeneous graphic objects such as colored layers, text or images, is crucial to enable collaboration between users as a way for giving feedback on certain parts of the score. *INScore* [4] supports various graphical objects, but is not easily embeddable in an online application and it is more focused on real-time rendering of interactive music scores [6] for new forms of composition and performance.

This paper presents LeadsheetJS a Javascript library for storing, visualizing, playing, editing and making graphical annotations on lead sheets. In the following section we describe the main features of the library. Then we give some hints about its implementation. We finally describe tools built on top of this library.

2. LEADSHEETJS

LeadsheetJS is a Javascript library for lead sheets. It enables the edition and visualization of lead sheets under conventional formats, as well as rendering, playing and storing lead sheets in a database. *Figure 2* shows how LeadsheetJS interfaces with the player, the menu for editing and the rendered leadsheet.

LeadsheetJS provides tools for users to collaborate and give feedback to each other by highlighting certain parts of the lead sheet and commenting or suggesting modifications. LeadsheetJS has been implemented in Javascript, the main programming language for web browsers. This makes LeadsheetJS web-friendly and easily embeddable in third-party sites, as well as adaptable to several devices.

In the next sections we describe the main features of LeadsheetJS and we give a detailed explanation about the main design and implementation aspects.

[1] http://musescore.org/

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

[3] http://www.scorio.com/

[4] https://flat.io/

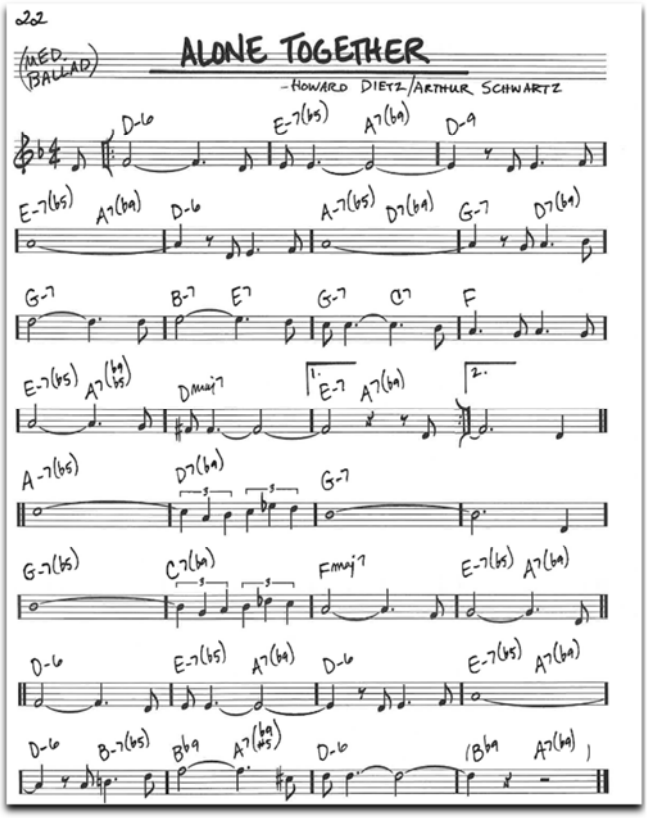

Figure 1. The lead sheet of "Alone together" by Dietz & Schwartz, as found in a typical Fake Book.

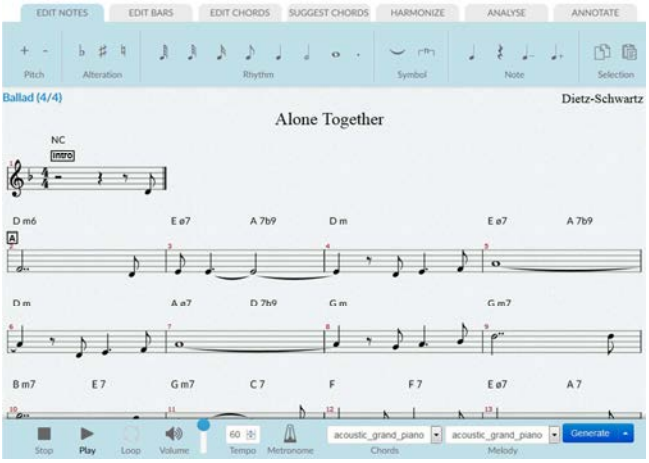

Figure 2. "Alone together" by Dietz & Schwartz, rendered in a browser with LeadsheetJS.

2.1 Peer feedback on lead sheets

"The one true comment on a piece of music is another piece of music", Stravinsky [17].

Music composition, as well as music learning, is a domain in which *feedback* on pieces being composed plays a major role. Feedback is traditionally provided by a teacher. Nowadays, on-line learning websites provide tools for peer-feedback in which learners can produce and review feedback made by peers.

The possibility of giving feedback on the audio representation of a piece of music has been addressed in previous works, e.g. [19, 20]. However, by commenting on pure audio, i.e. on a rendered waveform, users are limited to commenting on given time spans, whereas by commenting on a lead sheet, users can refer directly to the musical elements making up lead sheets, such as notes, chord labels, chord transitions, bars or structural elements (see *Figure 3*).

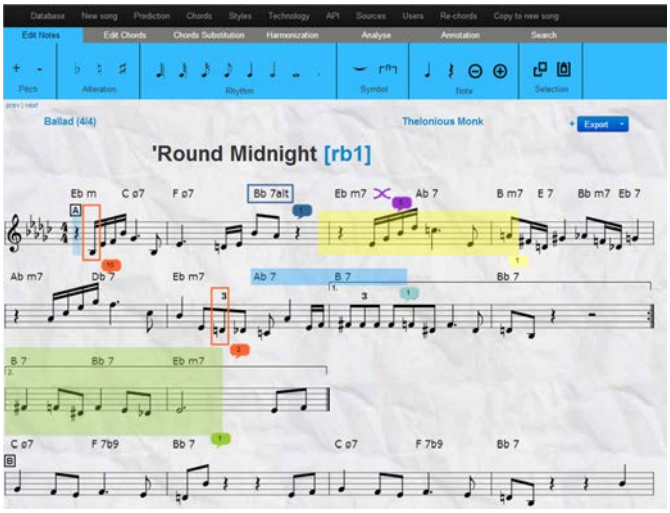

Figure 3. Examples of annotations on specific parts of a lead sheet.

In LeadsheetJS, feedback can be given at three levels:

a) Musical feedback: the basic level of feedback is musical. That is, a suggestion of a modification of a certain part of the lead sheet, such as changing certain notes, or certain chord labels,

b) Text feedback: musical suggestions can be explained with an explanation in the form of text comment,

c) Audio feedback: sometimes a musical idea is better expressed by being played in an instrument. Users can record a musical snippet, upload it and associate it to a specific metrical location in the lead sheet.

2.2 Embeddability

Arbitrary websites can render lead sheets by importing the LeadsheetJS library in the HTML source code. New lead sheets can be created or imported and rendered and edited from the site. As an example we show a website in the MusicCircle platform [19], displaying the lead sheet "Blue Room" by Rodgers & Hart (see *Figure 4*).

First, the LeadsheetJS library is imported in the HTML page. Then, the lead sheet of "Blue Room" is imported from a database (LSDB, described later) in our JSON lead sheet format through the LSDB API, which allows external sites to retrieve lead sheets. Finally, the JSON text is converted to a LeadsheetJS object and displayed in the page (see *Figure 5*).

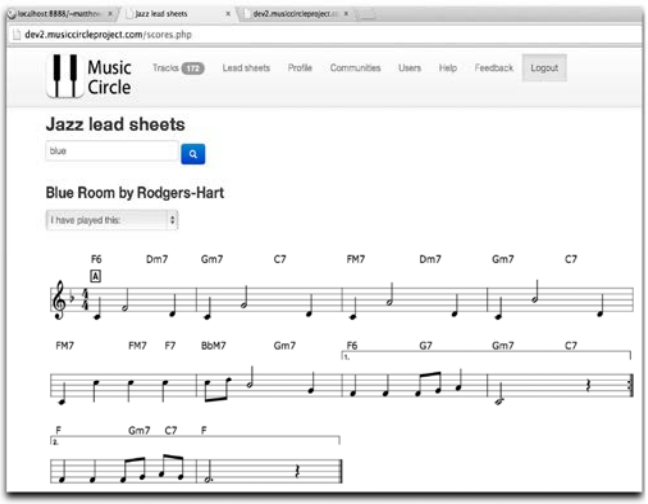

Figure 4. A lead sheet view embedded in a third party site.

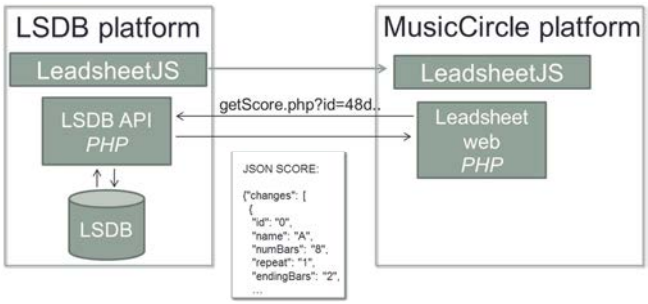

Figure 5. Architecture for embedding LeadsheetJS.

2.3 Multi-device

Web applications are not accessed only from a desktop computer but also from tablets and mobile phones: responsive web design has become essential for designing web applications. To that aim, LeadsheetJS resizes automatically scores depending on the width of the screen. This way it can be visualized in devices with different screen sizes such as tablets or mobile phones (see *Figure 6*).

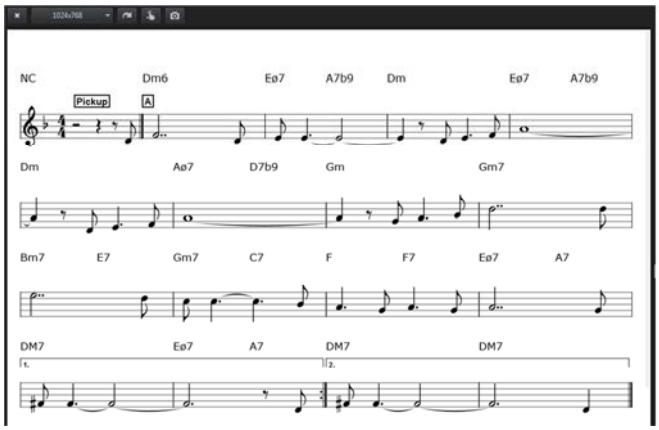

Figure 6. LeadsheetJS on a 1024x768 tablet.

2.4 Audio wave visualization

LeadsheetJS does not handle only symbolic information. Recordings of the performance of a lead sheet can also be associated to the lead sheet. LeadsheetJS provides visualization of the recording's waveform synchronized with the lead sheet, so that on top of each measure, the waveform of the recording part corresponding to that measure is displayed (see *Figure 7*). This feature is useful for musicians who record themselves performing a given lead sheet. They can then listen to their performance and see at the same time the lead sheet and the audio representation.

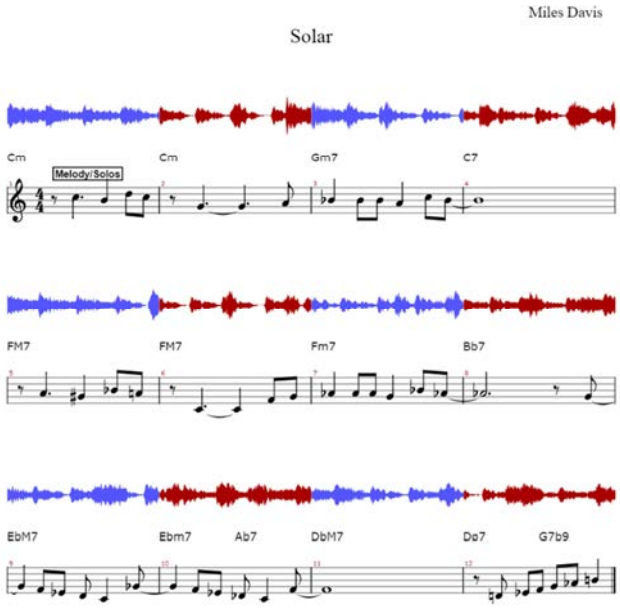

Figure 7. LeadsheetJS visualizing Solar, by Miles Davis, and audio recording displaying

2.5 Design

LeadsheetJS is a complex library that provides many functionalities (editing, visualizing, playing, storing). From an architectural point of view, it needs to be maintainable, scalable and extensible. Furthermore, modularity is required as users may need to use only certain features of LeadsheetJS. For example, a music blogger may want to visualize and play lead sheets in her blog without allowing edition or audio visualization.

The design of LeadsheetJS is module-based. It is inspired by Zakas' architecture [21] in which every module is an independent unit that does not need any other module to work. Zakas' architecture is based on the MVC (Model-View-Controller) architecture. Every module has its own model, view and controller classes. Each module is composed of a set of classes. There is one file per class. In total LeadsheetJS contains about 150 classes.

LeadsheetJS is a client-based Javascript library, i.e. it runs in the browser. However, certain functionalities require communication with a server or a database, such as storing or retrieving lead sheets. Databases and servers are not part of LeadsheetJS, yet it provides modules to communicate with them.

The architecture scheme is shown in *Figure 8*. The central module is *Leadsheet Model*. All modules depend on it since they need it in order to work. Modules *Viewer*, *Player* and *Interactor* provide visualization, playing and edition functionalities respectively. The *Annotation* module provides graphic annotation for peer feedback purposes. The *Format exporter/importer* modules is a converter to various formats so that the represented lead sheet can be sent to (or received from) other applications. The *Ajax* module facilitates the communication to a server. Therefore, it is used by the modules that depend on a server: the *Data Base* module, which is in charge of storing the lead sheet to a database in a given format, and the modules that are analysis tools which we describe in section 3.

Thanks to its modular nature, LeadsheetJS can be easily extended by adding modules that communicate with the existing ones.

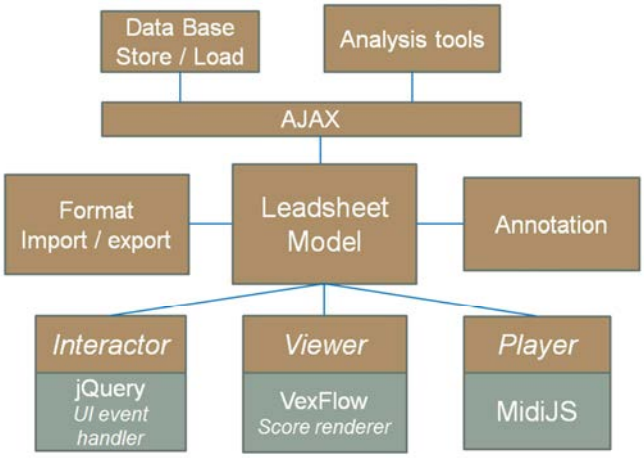

Figure 8. Module architecture of LeadsheetJS.

In *Figure 9* we show an example of LeadsheetJS embedded within a complete system with a client/server database system where LeadsheetJS is the client part, and PHP is the language on the server side that manages user sessions and persistence (saving lead sheets into a MongoDB database). The Ajax module is in charge to send requests to the server. For example, in order to store a lead sheet in a database the Database module will send the data to the server as an HTTP request through the Ajax Module.

The core module, *Leadsheet Model*, represents a lead sheet. A lead sheet consists of a melody that is in most cases monophonic, and a chord label grid representing the harmony. From a structural point of view, a lead sheet is a hierarchical structure composed by sections, which are composed of bars, which in turn are formed by a list of notes (a melody), and a list of chord labels. Each of these levels defines specific attributes: at the top level, the lead sheet has a composer, a title, a style as well as musical attributes such as global key and time signature. Section related information attributes are section name, number of bars, number of repetitions and number of endings. Bars may also have specific time or key signature changes, as well as structure labels like coda or segno. Finally, the lowest levels of the hierarchy are notes and chord labels.

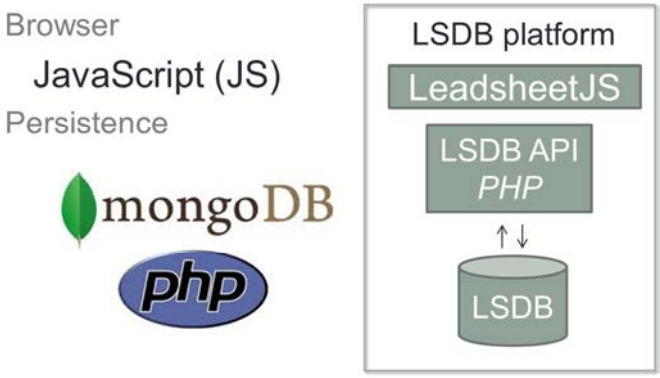

Figure 9. Example of a client-server database structure using LeadsheetJS.

The example in *Figure 1* shows a lead sheet as found in a typical Fake book, with its attributes such as title, “Alone Together”, composer “Howard Dietz and Arthur Schwarz”, style “Medium Ballad”. This lead sheet has two sections: the first one contains 14 bars and two endings; the second one has 12 bars.

The Leadsheet Model module enables applications to store and retrieve information about a lead sheet such as its structure, a specific bar, a chord label, or a group of notes, as well as metadata associated to it such as its title, composer, style, time signature or key signature. Typical queries include *get the notes of the first bar*, *get the number of sections*, etc. The Leadsheet Model also enables creation of new lead sheets or copies.

2.5.1 Viewer

The Viewer renders lead sheets on the web browser through an HTML5 canvas API, which allows generating graphics dynamically. The Viewer uses *Vexflow*[5], a low level score rendering Javascript library. *Vexflow* addresses low level rendering of notes and staves, whereas LeadsheetJS specifies what to draw in each bar as well as other higher level tasks such as determining how many bars to display per line.

2.5.2 Interactor

The Interactor component provides the editing part by using the library *JQuery*[6] which, among many other things, takes care of event handling. Keyboard and mouse events are caught by the Interactor to perform desired transformations on an edited lead sheet. We introduce three levels of edition: notes, chord labels and bars. Note edition works like in any traditional score editor. Chord label edition provides specific interaction schemes such as completion to suggest the most relevant chord types in a given context (see *Figure 10*). LeadsheetJS contains a comprehensive database of over 300 chord types, collected during the process of a lead sheet database compilation described in section 3.1.

[5] http://www.vexflow.com
[6] http://jquery.com/

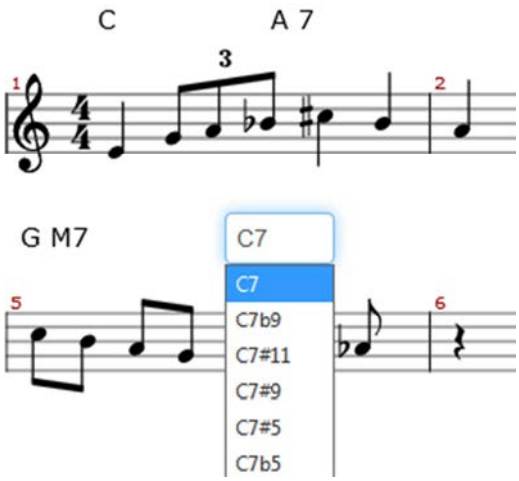

Figure 10. Chord label completion to speed up edition.

2.5.3 Player

LeadsheetJS provides a MIDI Player which uses the library *MidiJS*[7] to play a lead sheet, i.e. both the melody and the chord labels. The chord labels are transformed into MIDI chords.

The chord labels are represented by a pitch and a chord type. E.g.: in C# maj7, *C#* is the pitch and *maj7* the chord type. The chord type database provides information about the note degrees for each chord type. For instance for maj7 the degrees are I, III, V and VII.

In order to play chords, LeadsheetJS transforms chord labels into sets of MIDI notes by calculating the notes degrees of the chord type relative to the root pitch. E.g.: for C# maj7, notes are C#-E#-G#-B#. The player plays them arbitrarily in the 4th octave, so MIDI notes are 61-65-68-72. Other more refined MIDI players can easily be defined by the user.

2.5.4 Javascript Module Management

As a client-based application, LeadsheetJS runs on the browser, so each Javascript file needs to be imported in the HTML source code through the *script* tag. This may be an issue as we need to include explicitly each file and there are around 150 classes, while not all classes are always needed. For example, an instance of LeadsheetJS could only show a lead sheet and play it: in that case there is no need for editing, so the Interactor module does not need to be loaded. To optimize loading time, and ensure only needed modules are loaded, LeadsheetJS uses *RequireJS*[8], a tool to manage dependencies in Javascript.
In order to provide communication between modules in an uncoupled way we make an intensive use of the *Mediator* design pattern [12]. The Mediator pattern encapsulates the way different modules interact. It enables a module to *subscribe* to an action of another module which *publishes* it.

For example, when the Leadsheet Model module changes the pitch of a note, it *publishes* that action; that is, it sends a message to a *mediator* telling that the note's pitch has changed. The *mediator* checks which modules are interested in the action of *note pitch changed*; that is, which modules are *subscribed*, and informs them. This way, the Viewer module, which is subscribed to *note pitch changed*, knows it must redraw the score.
The advantage of using this pattern is that Leadsheet Model and Viewer do not communicate directly, which brings to uncoupled code, thus, more scalable and maintainable.

2.5.5 Javascript implementation

Javascript is a prototype-based language rather than a class-based one like C++ or Java. In order to define classes, there are mainly two approaches: to use Object literals or to use prototypes. By using object literals to define classes one can use private variables by using the *Module Pattern* [12]. The Module Pattern takes advantage of closures to simulate private variables, which are not natively supported in Javascript. On the other hand, using prototypes to define classes one cannot emulate private variables, but this approach has the advantage that it is less memory consuming, since all the methods of all instances of a class share the same memory. We have mainly used the Prototype approach as we are using multiple instances of many classes such as NoteModel or ChordModel.

2.6 JSON lead sheet format

LeadsheetJS provides a format to store lead sheet data in a database. The most common format for representing music scores is MusicXML [7]. LeadsheetJS does not use MusicXML for the following reasons: first, in MusicXML, chord labels' information is associated to a note, so the start beat of the chord is the same as that of the associated note. This makes it difficult to represent chords whose start beat does not match with the start beat of a note. This might not be a problem for other kinds of scores, but in lead sheets chord labels are crucial. That is why in our lead sheet format each chord label has its start beat information. Second, MusicXML provides exact formatting: it saves both musical and visualization information; e.g. for each note it saves the stem direction and the exact position in which it will be shown. LeadsheetJS only needs the musical information to render the lead sheet. The visualization aspects (stem directions, position of each element…etc.) is decided by Vexflow.

There are other human-readable music notation formats like ABC [3] and Lilypond [11]. Both are designed to let users create easily scores by writing text which is compiled by a software that produces a rendered score as an output. Therefore, they are not designed to be used in WYSIWYG[9] editors. The Guido Music Notation format [5], designed to be rendered by the Guido Engine Library [2] is similar to them, but is not only a representation format; it also supports 'functions' as instructions for transforming the score (e.g. transposing a melody). In our case, readability is not a priority as we do have a WYSIWYG editor. Instead, we have designed a JSON

[7] http://mudcu.be/midi-js/
[8] http://requirejs.org/

[9] What You See Is What You Get

(JavaScript Object Notation) based format [1], as JSON is a popular lightweight format which is widely used in web APIs. For example, the GUIDO API web-service is based in JSON [18]. Further, a lead sheet has a hierarchical structure which can be very well represented by the JSON format (see *Figure 11*). The decision of using JSON has distanced us from using other formats like MEI[16], a notation encoding standard based on XML similar to MusicXML.

However, LeadsheetJS is compatible with MusicXML as it provides a parser to transform MusicXML to our JSON lead sheet format, and it will eventually support other formats too (Lilypond, Guido and ABC).

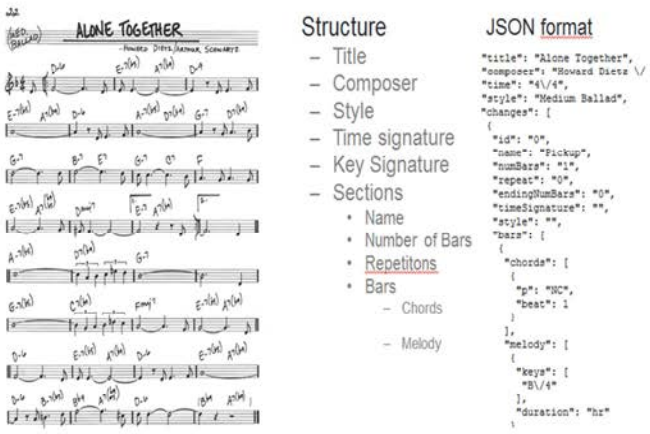

Figure 11. The lead sheet "Alone together" represented in JSON.

3. OTHER APPLICATIONS

This section describes applications using LeadsheetJS in various ways.

3.1 Lead sheet Database (LSDB)

The Lead sheet Database (LSDB) [15] is a comprehensive, on-line database of lead sheets for jazz and Brazilian music. Currently LSDB contains over 10,000 songs from 76 different song books, and over 300 different chord types.

Songs are entered by professional musicians using LeadsheetJS. Average time for entering songs is about 3 minutes, thanks to the availability of many short-cuts for fast editing. An LSDB API stores/retrieves lead sheets from the database, as described in section 2.2. This database is used for musicological analysis and music generation applications such as the tools described in section 3.2

The LSDB database uses MongoDB[10], a non-relational database (NoSql). NoSql databases are based on collections that contain JSON documents, which are structures of nested arrays and objects (objects are set of key-values). The biggest drawback of using a NoSql is that some important features of SQL databases such as *joins* or referential integrity cannot be performed at the database level, and have to be managed from the code of the server that produces the queries. This can be an issue in applications with complex databases, but in our case it is not, because the database structure is quite simple: there is a main collection of lead sheets, and then other related collections like sources and composers, so integrity is not as crucial as in other more complex systems. Joins are managed from the server language's code. Moreover, the JSON structure on NoSql databases is ideal to represent tree-based structures like lead sheets, whereas representing a tree in a SQL is quite more complex.

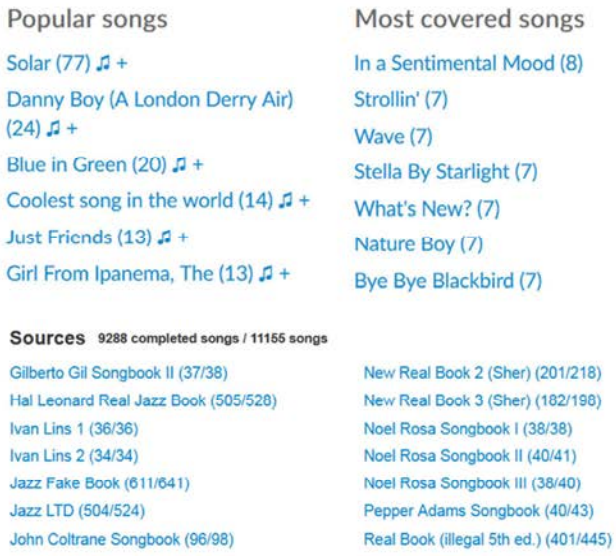

Figure 12. Part of LSDB content as shown in the web.

3.2 Automatic Feedback on lead sheets

Feedback can sometimes be provided automatically. LeadsheetJS provides various tools that produce automatic feedback to users who are trying to compose a song. This feedback can be either in the form of an analysis of the lead sheet, or in the form of generations and transformations of a lead sheet.

For instance, a *Chord Sequence Analyzer* tries to find which *style* or styles a sequence of chords expresses. A style is defined here by a *corpus* of songs, corresponding to a given composer; e.g. the style of Miles Davis [8]. The Chord Sequence Analyzer identifies the longest subsequences that can be analyzed in the style of a given set of key composers. This analysis is performed by computing the similarity of the chord sequence with several different composers' models. These models are statistical models generated from the LSDB.

Such a tool may be used to get information about how original or similar a lead sheet is, with regards to the LSDB database. *Figure 13* shows such an analysis for the chord sequences "Solar" with a map showing a time-line of the song and each composer (Pepper Adams, Charlie Parker, Duke Ellington and Michel Legrand)

[10] http://mongodb.com/

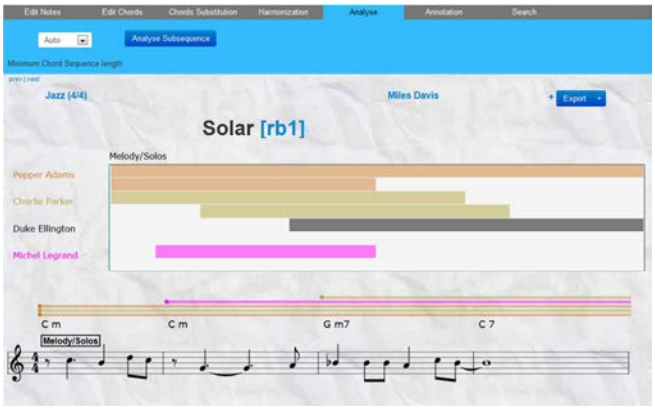

Figure 13. A chord sequence analyzer grafted on top of LeadsheetJS.

Another example is the *Harmonic Analysis* tool that finds the local tonalities of a lead sheet given its chord label sequence [13]. *Figure 14* shows two examples of analysis: Gm7 – C7 has been analyzed as *F Major* chords, whereas Fm7 – Bb7 are analyzed as *Eb Major*. These chords are part of "Solar", by Miles Davis.

Other automatic feedback tools have been defined, such as a *Chord Substitution* tool which, from a given chord or chord sequence, suggests alternatives based on chord substitution rules that are learnt from a specific corpus.

The *Harmonizer* tool, given a monophonic melody, proposes a multi-voice harmonization in a given style. E.g.: one can harmonize the melody of Coltrane's jazz standard Giant Steps in the style of Wagner or Bill Evans [14].

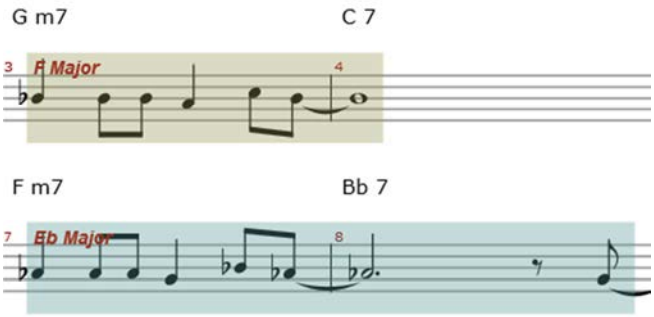

Figure 14. Harmonic analysis displayed on parts of "Solar", by Miles Davis.

Figure 15 shows the architecture of these tools and illustrates the process for the Chord Sequence analyzer tool: The user clicks on a button 'Analyze chord sequences'. LeadsheetJS catches the user action and requests the chord sequence analysis of *Solar*, sent in JSON format through the Ajax module. The request is sent to the server where the Leadsheet Web API, which is a server extension of LeadsheetJS, computes the chord sequence analysis. The response is sent to the client, where LeadsheetJS presents it in the User Interface as a time-line map.

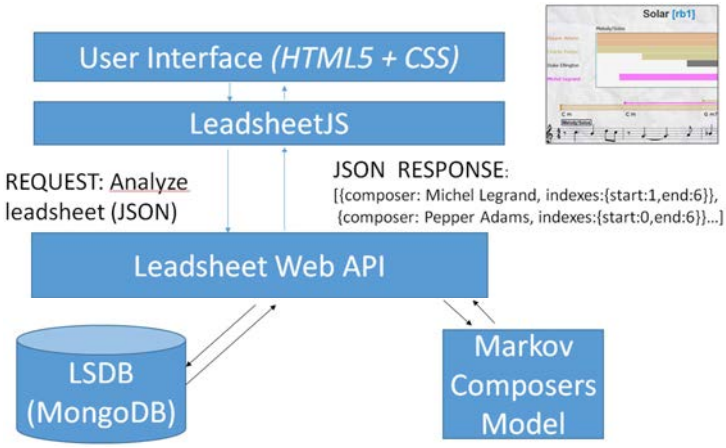

Figure 15. LeadsheetJS architecture and the data flow of chord sequence analyzer.

3.3 Flow Composer

In the context of the Flow Machines[11] project about style imitation, an online composition tool called Flow Composer was designed, to help a composer generate a lead sheet using different "styles". Again, styles are defined by corpus of songs taken from the Lead sheet Database.

The main idea is that a composer can start to create a song and leave some empty measures in which there will be only silences. Then, he queries the system to fill those blanks in a given style. Those blanks can be on the melody, represented by silences, or on the chord grid, represented by *No Chords* (NC). The system will generate a melody or chord labels to fill them taking into account the style chosen by the user, and also constraints of continuity. Composers usually don't want a whole new random song; they rather want the system to help them with certain parts of their composition. The composer can accept or reject all or part of the system's proposition. Flow Composer tools allow composers to have at any moment a full control on the lead sheet: there is a *history* feature in which every step is saved, so they can go back to a previous state.

Flow Composer is built on top of LeadsheetJS and uses the same modular approach. LeadsheetJS is used in Flow Composer to listen, view and edit lead sheets. We show in *Figure 16* how Flow Composer works. In the first image (on the top) a user is composing a bossa-nova. In the song there are two parts. The second part starts at measure 7 (with note *F* and chord *F7*) and is not shown in the figure. The second part is ok, but the composer does not know how to finish the first part so that it transitions well to the second part. So he leaves it empty with silences and *no chords* (NC), and queries Flow Composer to fill the empty part in the *bossa-nova style*. The second image (on the bottom) shows the result proposed by Flow Composer: it has filled the empty part by proposing a melody and a chord grid. Interaction may then proceed by accepting parts of the suggestions and/or querying other solutions.

[11] http://www.flow-machines.com

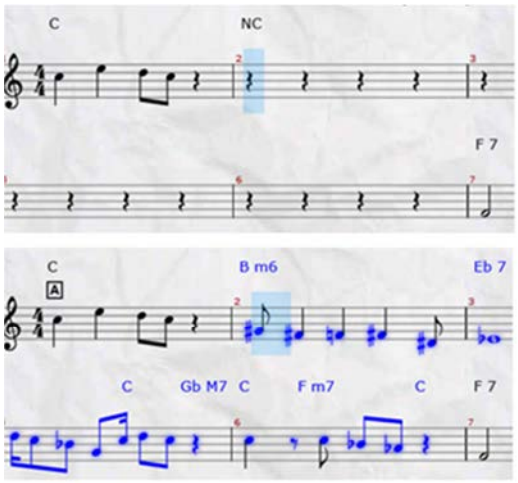

Figure 16. Flow Composer completion in blue.

3.4 Experiment on feedback in composition

PRAISE[12] (Practice and Performance Analysis Inspiring Social Education) is a social network for music education with tools for giving and receiving feedback in online communities. In the context of PRAISE we have built a tool for feedback in composition in which composers can compose a lead sheet and share it with other composers who can then provide feedback. This tool is based on the annotation module of LeadsheetJS.

In the PRAISE project, we designed an experiment to determine the impact of feedback in lead sheet composition [10]. We evaluate whether musical peer feedback, just like in the example explained in section 2.1, may actually improve or not the musical *quality* of a composition. In a first phase, participants are asked to compose a short song (8 bars). In the second phase they are invited to suggest modifications of other participants' compositions. Then participants are asked to reconsider their original song and try to improve it. The point is that a group of subjects will have received feedback whereas another group will have not. We then evaluate to which extent the quality of the improved composition of those subjects who received is better than that of those who did not. The quality evaluation is estimated from a listening panel. LeadsheetJS was used to implement this experiment, including modules for editing and playing for the composition phase and the Annotation module for the feedback phase.

The composer of the lead sheet can later review suggestions and accept them or not.

The feedback process is illustrated as follows. First, user *Bruno* composes a song and edits it with LeadsheetJS. Later, user *Silvia* looks at it and plays it. She decides to make some suggestions on certain notes. As shown in *Figure 17* once she has saved the suggestion, she can perform other actions, shown in the contextual menu:

- Add Comment: add an explanation of her musical suggestion,
- Upload sound: upload a sound recording related to the suggestion,
- Modify: she can decide to modify the suggestion she just saved,
- Remove: remove the suggestion.

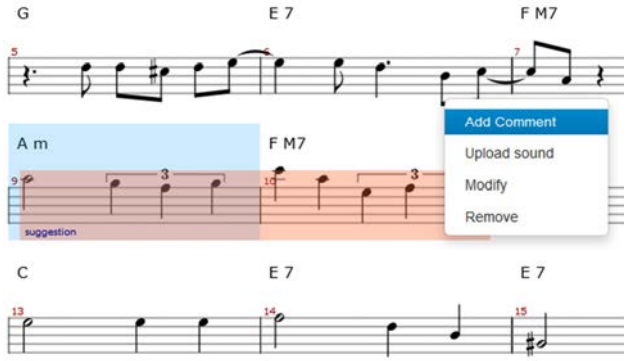

Figure 17. A user makes a suggestion on a specific part of a lead sheet.

Later on, Bruno can review all suggestions by switching between the *original* elements and *suggested* ones and listen to them. *Figure 18* shows a lead sheet with three suggestions. Bruno clicks on one of them to see the associated explanation.

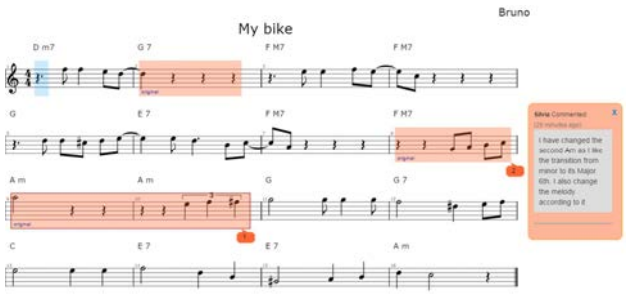

Figure 18. A user checks the suggestions received.

Finally, if Bruno likes the suggestion he can *accept* it so that the suggestion is merged with the whole song by right-clicking on the suggestion (see *Figure 19*).

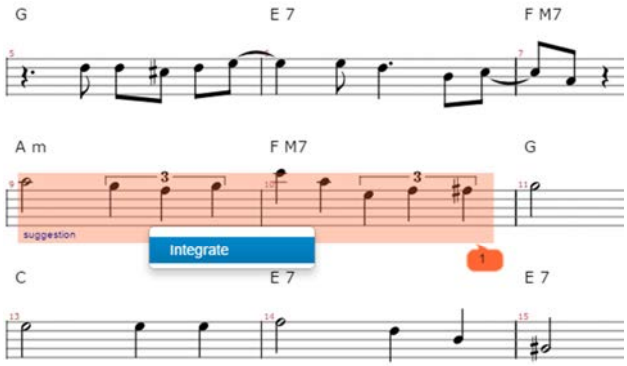

Figure 19. The user accepts a suggestion of modification.

[12] http://www.iiia.csic.es/praise/

4. CONCLUSION

We have presented LeadsheetJS, a Javascript library for lead sheets. By design, LeadsheetJS is compatible with multiple devices and easily embeddable. LeadsheetJS also provides various tools for music composition such as automatic analysis and peer feedback. We have illustrated how LeadsheetJS is used in several online music applications.

LeadsheetJS addresses the needs of online applications for composing, generating, sharing or teaching music online. New features are currently investigated such as multiple voices management, lyrics, audio based player, as well as rendering lead sheets using style-based accompaniment generation systems.

5. ACKNOWLEDGEMENTS

This work is supported by the Praise project (EU FP7 number 388770), a project funded by the European Commission under program FP7-ICT-2011-8.

6. REFERENCES

[1] Crockford D. 2013. “The json data interchange format”. Technical report, ECMA International, October.

[2] Daudin, C., Fober, D., Letz, S., and Orlarey, Y. “The guido engine a toolbox for music scores rendering.” In *Proceedings of the Linux Audio Conference* (2009), pp. 105–111.

[3] Dyke, G., & Rosen, P. (2010). abcjs–Project Hosting on Google Code.

[4] Fober, D., Letz, S., Orlarey, Y., & Bevilacqua, F. (2013, July). “Programming Interactive Music Scores with INScore”. In *Proceedings of the Sound and Music Computing Conference* 2013 (pp. 185-190).

[5] Fober, D., Orlarey, Y., & Letz, S. (2012). “Scores level composition based on the Guido Music Notation”. Ann Arbor, MI: MPublishing, University of Michigan Library.

[6] Fober, D., Orlarey, Y., & Letz, S. (2014, October). “Augmented Interactive Scores for Music Creation.” In *Korean Electro-Acoustic Music Society's 2014 Annual Conference* (pp. 85-91).

[7] Good, M. (2001, December). “MusicXML: An internet-friendly format for sheet music.” In *XML Conference and Expo* (pp. 3-4).

[8] Hedges, T., Roy, P., & Pachet, F. (2014) “Predicting the Composer and Style of Jazz Chord Progressions.” *Journal of New Music Research*, 43(3), 276-290.

[9] Kuzmich, J. “The two titans of music notation.” (2008, September) *School Band & Orchestra magazine*.

[10] Martín D., Frantz. B, Pachet, F. (2015, October) “Assessing the impact of feedback in the composition process: an experiment in lead sheet composition.” In *Tracking the Creative Process in Music*, Paris.

[11] Nienhuys, H. W., & Nieuwenhuizen, J. (2003, May). “LilyPond, a system for automated music engraving.” In *Proceedings of the XIV Colloquium on Musical Informatics* (XIV CIM 2003) (pp. 167-172).

[12] Osmani, A. (2012). “Learning JavaScript Design Patterns”. *O'Reilly Media, Inc.*

[13] Pachet, F. “Surprising harmonies.” (1999, February) *International Journal of Computing Anticipatory Systems 4*.

[14] Pachet, F. Roy, P. (2014) “Non-conformant harmonization: the real book in the style of take 6.” In *International Conference on Computational Creativity* Ljubljiana.

[15] Pachet, F., Suzda, J., & Martín, D. (2013). “A Comprehensive Online Database of Machine-Readable Lead-Sheets for Jazz Standards”. In *ISMIR* (pp. 275-280).

[16] Roland, P. (2002, September). The music encoding initiative (mei). In Proceedings of the First International Conference on Musical Applications Using (Vol. 1060, pp. 55-59).

[17] Stravinsky, I. and Craft, R. “Dialogues” (London: Faber and Faber 1982).

[18] Solomon, M., Fober, D., Orlarey, Y., & Letz, S. (2014, March). “Providing Music Notation Services over Internet”. In *Proceedings of the Linux Audio Conference*.

[19] Yee-King M., d'Inverno M., Noriega P., (2014, May) “Social machines for education driven by feedback agents”, in *Proceedings First International Workshop on the Multiagent Foundations of Social Computing, AAMAS-2014*, Paris, France.

[20] Yee-King M., d'Inverno M. (2014, May) “Pedagogical agents for social music learning in Crowd-based Socio-Cognitive Systems”, in *Proceedings First International Workshop on the Multiagent Foundations of Social Computing, AAMAS-2014*, Paris, France.

[21] Zakas, N. Scalable “Javascript Application Architecture”. Slides: http://cern. ch/go/Cl6S.