

A Front End for Adaptive Online Listening Tests

Johan Pauwels
Centre for Digital Music
Queen Mary University of
London
j.pauwels@qmul.ac.uk

Simon Dixon
Centre for Digital Music
Queen Mary University of
London
s.e.dixon@qmul.ac.uk

Joshua D. Reiss
Centre for Digital Music
Queen Mary University of
London
joshua.reiss@qmul.ac.uk

ABSTRACT

A number of tools to create online listening tests are currently available. They provide an integrated platform consisting of a user-facing front end and a back end to collect responses. These platforms provide an out-of-the-box solution for setting up static listening tests, where questions and audio stimuli remain unchanged and user-independent. In this paper, we detail the changes we made to the webMUSHRA platform to convert it into a front end for adaptive online listening tests. Some of the more advanced workflows that can be built around this front end include session management to resume listening tests, server-based sampling of stimuli to enforce a certain distribution over all participants, and follow-up questions based on previous responses. The back ends required for such workflows need a large amount of customisation based on the exact listening test specification, and are therefore deemed out of scope for this project. Consequently, the proposed front end is not meant as a replacement for the existing webMUSHRA platform, but as starting point to create custom listening tests. Nonetheless, a fair number of the proposed changes are also beneficial for the creation of static listening tests.

1. INTRODUCTION

Creating a typical online listening test starts with preparing a set of audio files as stimuli and a set of questions to be asked. Then a listening test platform is configured with these inputs and deployed online. A number of such integrated platforms, consisting of a user-facing front end and a back end to store listeners' responses, are now available, including webMUSHRA [9], BeagleJS [4], the WebAudioEvaluationTool [3] and Go Listen¹. What all of these have in common is that they only allow one to define static listening tests. The experiment designer needs to define beforehand what stimuli to use and what questions to ask; no changes to the test can be made based on users' responses. The result is that the test will behave exactly the same way every time it is run, apart from some potential random shuffling. While this might be exactly the intended behaviour in some

¹<https://golisten.ucd.ie/>



Licensed under a Creative Commons Attribution 4.0 International License (CC BY 4.0). **Attribution:** owner/author(s).

Web Audio Conference WAC-2021, July 5–7, 2021, Barcelona, Spain.

© 2021 Copyright held by the owner/author(s).

cases, for some use-cases a more adaptive setup would be required. Examples of the latter include progress management such that partially completed sessions can be resumed, server-based stimulus sampling such that a distribution can be enforced over multiple participants, and follow-up questions triggered by previous answers.

In this paper, we discuss the changes we made to an existing tool for online listening tests, webMUSHRA, in order to make it suitable as a front end for adaptive listening tests. As part of these changes, some improvements were made that also benefit the creation of static listening tests. We start in section 2 by briefly explaining the current state of the webMUSHRA platform to the extent that is required to understand the rest of the text. In section 3, we will detail the general improvements to the front end that are beneficial for static and adaptive listening tests alike. We then move on to the architectural changes made to adaptively update the front end in section 4. We conclude with some remarks about back ends and future improvements in section 5.

2. CURRENT ARCHITECTURE OF WEB-MUSHRA

The front end of webMUSHRA is a single-page web application coded in JavaScript following the 5th edition of the ECMAScript standard. It is configured by means of a YAML file specifying the audio stimuli and questions for a number of tests, along with additional UI and UX options. The tests are presented on different “pages”, each with their associated user interface, and can be of the following types:

Generic page A non-audio page that simply displays the HTML content given in the YAML configuration, typically used for providing additional explanation about following experiments.

Volume test A volume calibration page that allows adjusting the overall volume to a comfortable level at the start and stores this level.

ITU-R BS.1116 (BAQ) test A Basic Audio Quality test as specified by the ITU-R standard BS.1116 [1, 12], wherein an audio condition under test and a hidden copy of a reference need to be compared blindly to that reference.

ITU-R BS.1534 (MUSHRA) test A Multiple Stimulus with Hidden Reference and Anchor test as specified by the ITU-R standard BS.1534 [2], where multiple audio conditions need to be compared to a reference together

with automatically degraded versions of the reference and a straight copy of it.

Forced or unforced paired test A test where the audio condition most similar to a reference needs to be chosen out of two options, potentially with a “don’t know” option.

Spatial localisation test A page where the origin of a 3D sound can be indicated in a variety of three-dimensional spaces with different attributes such as width, height, depth, apparent/auditory source width [6, 7] or listener envelopment [11].

Single-stimulus Likert test A page that presents a single audio stimulus with an arbitrary question that needs to be answered by selecting a point on a Likert scale [5].

Multi-stimulus Likert test A page with multiple audio stimuli, each having an associated Likert scale to respond to the same question.

Final page An obligatory final page to finish a set of one or more tests, optionally containing a questionnaire or feedback on the participant’s performance.

The front end makes extensive use of the jQuery library version 2.1 and uses jQuery Mobile v1.4 for its UI elements. Further details about the libraries used can be found in [8]. Communication-wise, only few exchanges are made between client and server. At the start of the webapp, the YAML configuration file is received from the webserver together with the client code and all assets. The URIs of the audio files are then read from the configuration, whereafter the files get requested from the webserver and loaded into the front end. The listening tests are now ready for user interaction and participants can make their way through all pages. In addition to their responses, the time spent on each page is recorded, such that fatigue effects can be monitored[10]. After completing the final page, all user responses are gathered and posted to the server as JSON. A diagram of this client-server interaction can be seen in fig. 1.

The accumulated responses can be configured to send to any back end that accepts JSON; it does not have to be the webserver that the front end is served from. A PHP-based back end that writes the JSON to a tabular format on disk is included as part of the webMUSHRA platform, but an alternative back end in Python is also available², which is built on the Flask microframework. This loose coupling between front end and back end was one of the main reasons why we chose webMUSHRA as a starting point for our modifications, together with its ease of configuration and high-quality audio engine built around the Web Audio API.

3. GENERAL FRONT END IMPROVEMENTS

The biggest general improvement made to the webMUSHRA front end is that the questionnaire creation capability was extracted from the final page and can now be used everywhere. Furthermore, the number of user interface elements that can be used in questionnaires has been significantly extended. In addition to the existing textboxes and

²<https://github.com/nils-werner/pymushra>

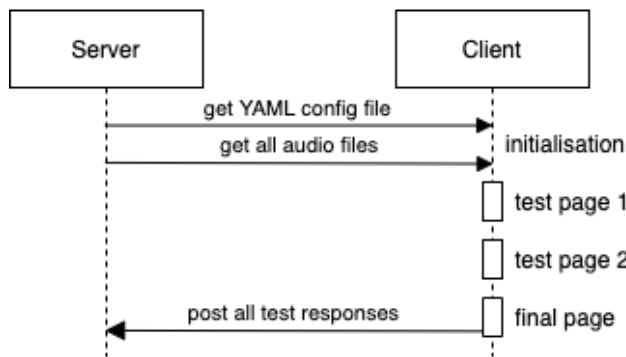


Figure 1: Client-server communication diagram of the current webMUSHRA version.

Likert scales (radio buttons), also toggle switches, checkboxes, date selectors, file selectors, email and password fields, sliders and dropdown lists have been added and number selection improved. In table 1, examples of each of the possible interface elements are displayed together with the YAML configuration that produces them. Some example use-cases for this generalised questionnaire include the creation of a consent form by adding a questionnaire to a generic page before the actual test starts, or the addition of optional free-form feedback to any listening test. These extensions make webMUSHRA comparable to general-purpose survey services, such as Google Forms, but with added audio capabilities.

The main consequence of the general availability of questionnaires, however, is for the single and multi-stimulus Likert tests. A questionnaire can consist of a single Likert scale, and therefore supersedes the page-specific Likert functionality. The latter is thus removed, leaving only the audio player to be rendered by default, but it can be recreated in a more general way as a questionnaire. For multi-stimulus tests, a general questionnaire is also possible, but in addition the stimulus-specific Likert scale is replaced by a stimulus-specific questionnaire which is replicated for every stimulus. This allows one to recreate the previously available functionality, and create more complex test layouts such as the one seen in fig. 2.

With the questionnaire capability comes the option to control for each input whether answering it is required or optional, unlocking the navigation to move to the next page when all required answers are given. Similarly, the audio players of the single and multi-stimulus pages have the ability to lock the questionnaire input elements until playback of the corresponding audio file has either been started or fully completed. Finally, further minor improvements have been made, such as the optional display of a waveform on single stimulus pages. Keyboard shortcuts have been added to all interface elements, such that listening tests can be completed using only a keyboard.

4. TOWARDS AN ADAPTIVE FRONT END

Whereas the changes proposed in the previous section are clear-cut improvements to static and adaptive listening tests alike, in this section we go over some changes that are necessary for adaptive listening tests, but are not necessarily en-

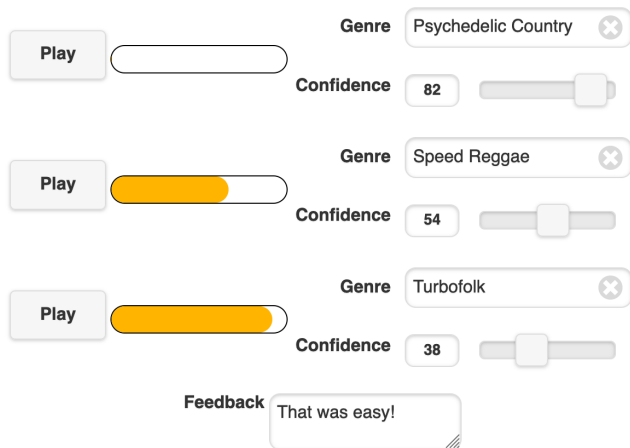


Figure 2: Multi-stimulus listening test demonstrating a more complex layout consisting of multiple UI elements per stimulus.

hancing static listening tests. More precisely, these changes are the lazy loading of audio files and immediate posting of responses.

Instead of loading all audio files specified in the YAML config files before the listening test starts, we modified the code to only request and load the files necessary for the next page. A noticeable drawback of this, is that navigating between pages is no longer as smooth as with the standard webMUSHRA front end. However, it has the advantage that the first page becomes available almost immediately, whereas the loading time at the start of the web app can be significant when tens of audio files are used.

Each type of page now has the option to post the user’s responses to the back end as soon as the page is exited, instead of waiting until the final page to accumulate and send all responses. This functionality can be activated by adding the key `send` to the page configuration. Whenever this key is present (and does not have a value `false`), all accumulated responses up to that point are sent, so it is possible to send responses at regular checkpoints after every N pages instead of after every page in order to reduce network communication.

To avoid sending the same trial data multiple times, the page configuration can be set to `send: forget`. In this case, the responses are deleted in the front end after they are successfully received by the back end. If communication with the back end fails, the responses are retained and included with the next posting of results such that they are not lost. Specifying `send: remember` keeps the responses after successful posting.

When `send: forget` is set for a generic page, it is only allowed to delete questionnaire responses of itself and previous generic pages, whereas the other pages (containing listening tests) are only allowed to delete responses to listening tests. This separation makes a variety of workflows possible that differ in their handling of sensitive data. For instance, it is common to collect some demographic data from listening test participants. When this data is obtained through a questionnaire on a generic page with `send: forget`, the sensitive data is sent to the back end before the actual listening test starts. In this case, none of the listening test responses can be traced back to the demographic data (provided the

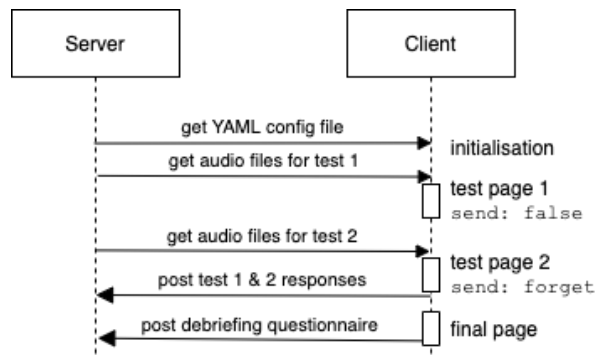


Figure 3: Client-server communication diagram with lazy loading of audio and selective posting of responses.

back end does not store the ip address of the client or the timestamp), whether the responses to the listening test itself are sent immediately or not.

Conversely, if the listening test responses need to be linked to participants’ data, this data can be requested via a questionnaire on a generic page with `send: remember` and will be sent alongside the listening test responses, regardless of them being sent after each page `send: forget` or accumulated together at the end `send: false`³. This collection of personal data can even be split into two parts, a first questionnaire in which sensitive data is asked and immediately dispatched to the server (`send: forget`), such as a participant’s real name for a consent form, followed by a questionnaire for less sensitive data that gets linked to subsequent test responses (`send: remember`), such as a username.

The added functionality provided by the immediate posting of responses can also be useful for static listening tests, but it would require a more advanced back end than the one currently included with webMUSHRA. Among other issues, responding to backwards navigation and the resulting multiple submissions poses a challenge, although backwards navigation can simply be disabled to avoid this problem. A potential advantage is that partial responses of users who withdraw or whose connection drops during an experiment are not lost.

The immediate posting of responses is independent of the lazy loading of audio files, but an example of client-server communication when both are activated can be seen in fig. 3.

The final change to the front end required to make adaptive listening tests, is to add the possibility of dynamically updating the configuration, more specifically the pages of the listening test. To this end, the front end checks for a JSON response from the back end whenever user input is posted. This JSON response can contain the configuration for additional pages, which are appended to the existing list of pages. It can also include the index of the following page, such that previously completed pages can be skipped and a session resumed where it was left off.

In order to start an adaptive listening test, a bootstrap YAML configuration should be present containing at least one page with a `send` option that is not `false`. The following pages are then requested from the server based on the user’s response so far. Multiple additional pages can be received

³For actual listening test responses, `send: remember` rarely makes sense, as it means that cumulated responses are posted which would need to be deduped in the backend

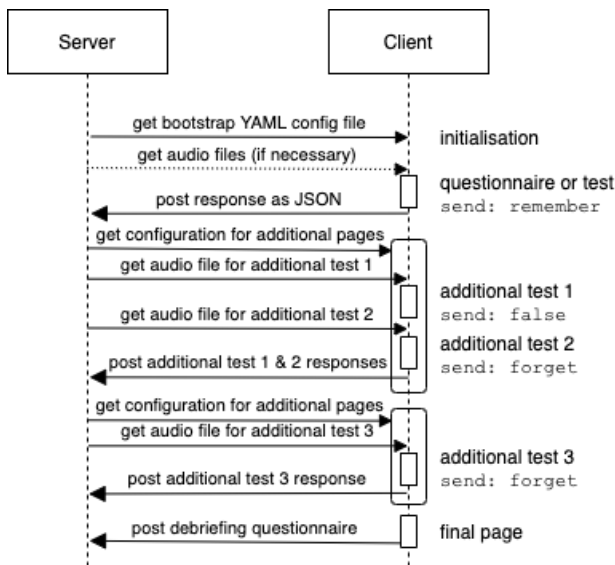


Figure 4: Client-server communication diagram for an adaptive listening test.

at once, and if one of the new pages also includes a `send` key, the process can continue indefinitely. This process is illustrated in fig. 4. If the goal is simply to provide the capability to resume a session, all pages can be configured in the initial YAML configuration and the update sent by the back end will then just be used to jump to the correct page in the sequence.

5. CONCLUSION

In this paper, we presented the changes we made to the webMUSHRA front end to turn it into a front end for adaptive listening tests. A sizeable number of these changes are clear-cut improvements that are beneficial for static listening tests as well as adaptive listening tests. These changes are currently being integrated into the next version of the main release of webMUSHRA.

Furthermore, a set of modifications has been made that are required for adaptive listening tests, but are not under all circumstances recommended for static listening tests. These modifications include lazy loading of audio stimuli, immediate posting of user responses and dynamically updating the test configuration. More discussion is required to decide on how these modifications can best be made available as part of the wider webMUSHRA platform. For the time being, they are available in our fork on Github⁴.

Adaptive listening tests require custom back ends adapted to the exact workflow that is desired. This allows for the construction of complex setups including session management, user personalisation and population balancing, but it also means that no back end can be provided that supports all possible scenarios out-of-the-box. This added complexity makes adaptive listening tests significantly harder to set up than static tests, for which the existing webMUSHRA platform remains a more suitable choice. Nonetheless, in order to lower this barrier to entry for adaptive listening tests, some proof-of-concept back ends illustrating various usage scenarios are available in our repository. These are writ-

ten in Python and use the FastAPI library⁵, but different languages and frameworks could potentially be used.

6. REFERENCES

- [1] International Telecommunication Union. *Recommendation ITU-R BS.1116-3: Methods for the subjective assessment of small impairments in audio systems*, 3th edition, February 2015.
- [2] International Telecommunication Union. *Recommendation ITU-R BS.1534-3: Method for the subjective assessment of intermediate quality level of audio systems*, 3th edition, October 2015.
- [3] N. Jillings, B. De Man, D. Moffat, J. D. Reiss, and R. Stables. Web audio evaluation tool: A framework for subjective assessment of audio. In J. Freeman, A. Lerch, and M. Paradis, editors, *Proceedings of the 2nd Web Audio Conference*, WAC '16, Atlanta, GA, USA, April 2016. Georgia Tech.
- [4] S. Kraft and U. Zolzer. BeagleJS: HTML5 and JavaScript based framework for the subjective evaluation of audio quality. In *Proceedings of the Linux Audio Conference*, 2014.
- [5] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 22(140):5–55, June 1932.
- [6] M. Morimoto and K. Iida. Appropriate frequency bandwidth in measuring interaural cross-correlation as a physical measure of auditory source width. *Acoustical Science and Technology*, 26(2):179–184, 2005.
- [7] A. M. Sarroff and J. P. Bello. Toward a computational model of perceived spaciousness in recorded music. *Journal of the Audio Engineering Society*, 59(7/8):498–513, September 2011.
- [8] M. Schoeffler, S. Bartoschek, F.-R. Stöter, M. Roess, S. Westphal, B. Edler, and J. Herre. webMUSHRA – a comprehensive framework for web-based listening tests. *Journal of Open Research Software*, 6(1):8, 2018.
- [9] M. Schoeffler, F.-R. Stöter, B. Edler, and J. Herre. Towards the next generation of web-based experiments: A case study assessing basic audio quality following the ITU-R recommendation BS. 1534 (MUSHRA). In *Proceedings of the 1st Web Audio Conference*, Paris, France, January 2015. IRCAM.
- [10] D. Schwarz, G. Lemaître, M. Aramaki, and R. Kronland-Martinet. Effects of test duration in subjective listening tests. In *International Computer Music Conference (ICMC)*, pages 515–519, Utrecht, Netherlands, Sept. 2016. ICMC, HKU University of the Arts Utrecht, HKU Music and Technology.
- [11] G. A. Soulodre, M. C. Lavoie, and S. G. Norcross. Objective measures of listener envelopment in multichannel surround systems. *Journal of the Audio Engineering Society*, 51(9):826–840, 2003.
- [12] W. C. Treurniet and G. A. Soulodre. Evaluation of the ITU-R objective audio quality measurement method. *Journal of the Audio Engineering Society*, 48(3):164–173, March 2000.

⁴<https://github.com/jpauwels/webMUSHRA/>

⁵<https://fastapi.tiangolo.com/>

Checkbox:	<input checked="" type="checkbox"/>	<pre> type: checkbox name: checkbox1 label: "Checkbox:" </pre>
Toggle:	<input type="checkbox"/> yes	<pre> type: toggle name: toggle2 label: "Toggle:" response: - value: "" label: no - value: yes label: yes selected: true </pre>
Text:	<input type="text" value="foo"/>	<pre> type: text name: text3 label: "Text:" </pre>
Email:	<input type="text" value="ben@trovato.com"/>	<pre> type: email name: email4 label: "Email:" </pre>
Password:	<input type="password" value="....."/>	<pre> type: password name: password5 label: "Password:" </pre>
File:	<input type="file" value="Browse... No file selected."/>	<pre> type: file name: file6 label: "File:" </pre>
Date:	<input type="text" value="05/07/2021"/>	<pre> type: date name: date7 label: "Date:" </pre>
Number:	<input type="text" value="5"/>	
Slider:	<input type="range" value="5"/>	
Likert scale:	<input type="text" value="12"/>	
Dropdown:	<input type="text" value="2"/>	
Number:	<input type="text" value="42"/>	<pre> type: number name: number8 label: "Number:" default: 42 </pre>
Slider:	<input type="range" value="42"/>	<pre> type: slider name: slider9 label: "Slider:" min: 0 max: 100 step: 1 default: 42 </pre>

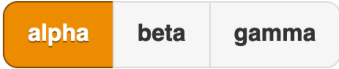
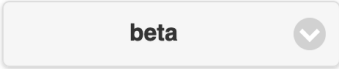

Likert scale:		<pre> type: likert name: likert10 label: "Likert scale:" response: - value: 0 label: alpha shortcut: q - value: 1 label: beta shortcut: w - value: 2 label: gamma shortcut: e </pre>
Dropdown:		<pre> type: dropdown name: dropdown11 label: "Dropdown:" response: - value: 0 label: alpha - value: 1 label: beta selected: true - value: 2 label: gamma </pre>
Longform text:		<pre> type: long_text name: long_text12 label: "Longform text:" </pre>

Table 1: User interface elements possible in questionnaires