Beat Byte Bot: A Chatbot Architecture for Web-based Audio Management

Gil Panal, Luís Arandas

▶ **To cite this version:**

Gil Panal, Luís Arandas. Beat Byte Bot: A Chatbot Architecture for Web-based Audio Management. Proceedings of the 11th Workshop on Ubiquitous Music (UbiMus 2021), Sep 2021, Matosinhos, Portugal. pp.72-82. ⟨hal-03398729⟩

HAL Id: hal-03398729

https://hal.science/hal-03398729v1

Submitted on 23 Oct 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons CC BY-NC-SA 4.0 - Attribution - Non-commercial use - ShareAlike - International License

Beat Byte Bot: A Chatbot Architecture for Web-based Audio Management

Gil Panal[1], Luís Arandas[2]

[1]Independent Researcher
Málaga, Spain.

[2]Faculty of Engineering, University of Porto – INESC-TEC
Porto, Portugal.

gilpanal@gmail.com, luis.arandas@inesctec.pt

Abstract. *The use of chatbots for content management and creation is pervasive. From the collaboration promoted by the web and its community, this paper presents research done on chatbots, content management architectures and web audio. Building on the universal use of social platforms for audio recording and transmission, as well as the robustness of chat platforms to handle microphones and audio uploading, a new architecture for sharing music content using the cloud is proposed. It promotes a path of sharing, inclusion and above all fosters a new way to establish audio communities with the help of chatbots. A functional multi-server chatbot-based architecture is presented with a multitrack audio editor as web application.*

1. Introduction

The use of chatbots promotes a new paradigm in human-computer interaction [Følstad and Brandtzæg 2017]. These define software that simulates human behaviour conversationally, and through the way they exist in programming interfaces eventually help people in certain tasks. Its use allows users to ask questions about a specific topic, ask for references or news, as well as to assist in the treatment of the most varied types of media files [Brandtzæg and Følstad 2018]. There have been research proposals from conversation-based music recommender systems [Jin et al. 2019], serverless computing [Yan et al. 2016] and even in routine teaching [Bii et al. 2018]. As management systems, chatbots can be programmed to undertake almost any task in the platform they exist, which offers multiple opportunities for the digital music community. Building on previous research [Adamopoulou and Moussiades 2020], this paper presents a modular and functional architecture of a system entitled *Beat Byte Bot*. A web-based system composed of several servers that in turn allows users to make music together on a multitrack browser application with the assistance of a chatbot. Its structure as a computational system is described, why some paths were taken in the development, iterations are proposed as well as discussed directions for other platforms of this type.

2. Web-based chatbots for music

Given the existing services that allow the creation of chatbots, it is accessible for all developers to create quickly and from scratch [Adamopoulou and Moussiades 2020]. There are even platforms – as services – dedicated to providing specialised infrastructure for this purpose,

which is the case of *Microsoft Azur*e [Tajane et al. 2018] and *IBM Watson* [Godse et al. 2018]. The number of chat services that enable the connection with these platforms is also large e.g. *Facebook* and *Slack* [Kozhevnikov et al. 2017]. More and more chat services evolve and provide dedicated application programming interfaces (APIs) with multiple methodologies for developers to interact with [Shakhovska et al. 2019], and this demonstrates functional plurality and a greater and greater web ecosystem. In the particular case of chatbots in web-based audio applications, we propose a conceptual division into two major areas important for the present research: 1) audio recognition and analysis, where we have e.g. *AudD* bot and *Tadam* bot[1] made with the *ACRCloud* music recognition toolkit [Medina et al. 2017]; and 2) music recommendation [Kozhevnikov and Pankratova 2018], where we have e.g. *Groovy* bot [Sari et al. 2020], *Jukebot*[2] and *MusicBot* [Jin et al. 2019][Yucheng et al. 2019]. There are also a large number of web services regarding music production workstations, given the potentialities of the browser – e.g. *Soundtrap* [Lind and MacPherson 2017]. This is a fertile field to the application of chatbots, and that is where part of the proposed architecture focuses, featuring a connection point to the audio managed by the chatbot from the various users on the system.

3. *Beat Byte Bot* architecture

Today, almost all chat platforms have audio upload and record functionality and this demonstrates an opportunity for file processing, orchestration and management [Zimmermann et al. 2004]. This opportunity, thanks to the public access to source code and APIs [Stallman 2002], allows creative communities to create new networks and systems that deal with audio on the web. From the ubiquitous possibilities provided and the potential scientific contributions, the *Beat Byte Bot* system was developed as an example of a possible architecture that can be used in the future of web-based digital music services. A centralized architecture [see Kim 2021 for decentralization] that connects different cloud services, built to work with web technologies, with five distinct components [Samizadeh et al. 2021]. These have been developed to provide system modularity in order to coexist together – one of which belongs to the chosen chat platform, *Telegram* and its instant messaging servers[3] [Hasyim and Pramono 2021]. The four remaining components are divided in: 1) the chatbot; 2) a middleware API; 3) an external storage database (DB); and 4) a multitrack audio editor as a web application. All research development is publicly hosted on *GitHub*[4] interfacing with the other needed services [Daniel et al. 2019]. As shown in Figure 1, there are several links between modules, and the same is done given the specifics of the platform on which they run. In the following sections the functionalities of each part of the system are detailed.

[1] https://audd.io/, https://telegram.me/Tadam_bot.
[2] https://groovy.bot/., https://getjukebot.com/.
[3] *Telegram* was chosen due to its openness and potential practical applications given the access to chatroom content by bots. It allows us to work with e.g. files, albums and live locations [Kozhevnikov et al. 2017].
[4] https://github.com/gilpanal/beatbytebot.

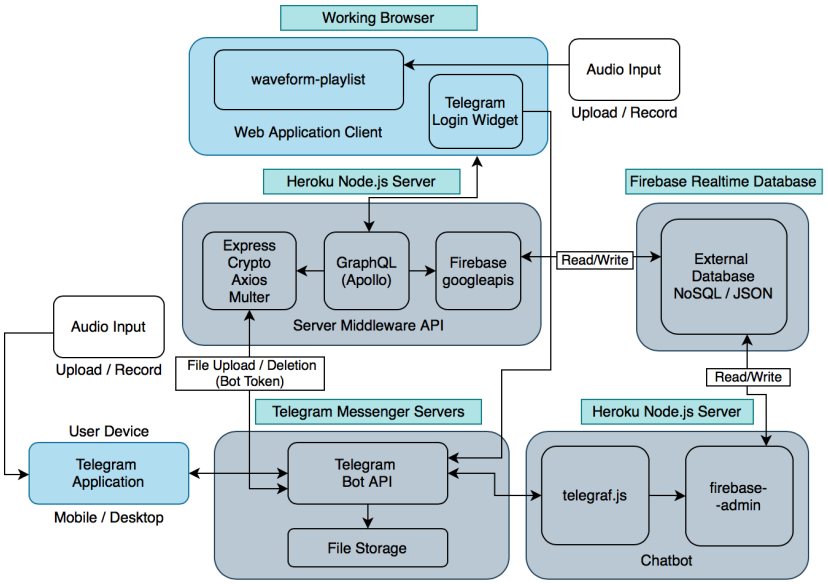

Figure 1. *Beat Byte Bot* architecture. Outlined in grey are the four major parts of the system, in blue are the connection points – via mobile or desktop, using the web app or *Telegram* itself – and in green the infrastructure.

3.1 The *Telegram* servers and the chatbot

Telegram's message servers are the main point of data management, as they are the ones that originate and record the data in the first place. The developed chatbot exists completely interconnected with the provided API. It is implemented in *Node.js* [Jannat et al. 2018] and deployed using the *Heroku* platform, a cloud service that provides Unix machines [as in Arandas et al. 2019]. When a user is connected to a functional chat with the bot and uploads audio – either in direct recording or by the file system – it is connected with the *Telegram Bot* API and hosted on its servers. From there, and whenever functional within a chat room, the bot continuously receives messages and then analyses them to know if they are audio messages. This is done real-time through a communication established on *Heroku* using the library *telegraf.js*[5] [Mardan 2018]. Based on chat administrator permissions, a metadata-based analysis is then made to the content [Saribekyan and Margvelashvili 2017]. Here we directly identify and transmit the uploads of the users to other modules of the architecture[6]. When successful, a connection is made with the next module, an external DB (see section 3.2). Using *firebase-admin* software development kit, the chatbot has instant control to this DB for reading and writing, allowing it to orchestrate the data from the detected audio files. The *Telegram* API maximum limit file size is of 20Mb for downloads, and the server can handle a maximum of 2Gb independently of the number of files, which makes *wav* format and others without compression or lossless not suitable to exchange due to its large size [Harchol-Balter et al. 2003]. The chatbot is the fundamental connection point between the users in the chat, what they type and record, and the rest of the architecture.

[5] https://telegraf.js.org/.

[6] At this point, there is no instant access to the audio buffer, only data about the post and specificities of the file. Something based on the type of response obtained by *Telegram*, not defined by our code.

3.2 *Firebase* realtime DB storage

As mentioned, every time the chatbot detects an audio file, information is retrieved from the message – e.g. file identification (ID), name of the *Telegram* channel or group and file format – in *JavaScript Object Notation* (JSON) format [Smith 2015]. Here, the nested data structure is automatically parsed and attributes are checked such as its duration, file size, title and *mime type* (e.g. *mpeg* and *ogg*) [Freed and Borenstein 1996]. After parsing, it is saved in a container in *Firebase Realtime* DB [Moroney 2017]. An external NoSQL DB hosted by *Google* services [Khawas and Shah 2018]. By keeping this module dedicated to recording data from the various audio files – given the capabilities of the platform in question – we can quickly expand the system without worrying about computing power [Tanna and Singh 2018]. In order to structure the data with the collaborative use of *Beat Byte Bot* in mind, when creating a new group or channel in *Telegram*, adding the chatbot means that we can start creating a *song*. That *song* is composed of multiple audio files, recorded or uploaded, that will be detected by the chatbot to follow the flow of the proposed architecture (see Figure 2 for the data structure).

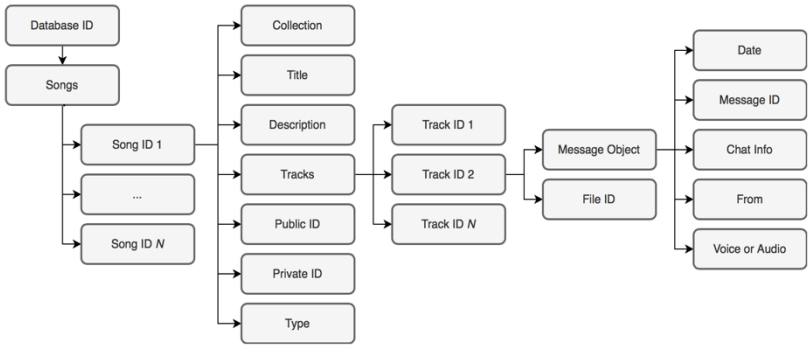

Figure 2. Data structure parsed by the chatbot to distil the message received from *Telegram* servers.

3.3 Server middleware API

In order to integrate the DB with the web application (see section 4), a dedicated middleware API was developed. Also made in *Node.js* and hosted on *Heroku* [Andersson and Chernov 2016] with a different instance than the chatbot, containing libraries such as *Express.js* [Liang et al. 2017], *Crypto, Axios* and *Multer* [Wittern et al. 2016], *GraphQL* (Apollo)[7] [Porcello and Banks 2018], *Firebase* and *Google* APIs, allows data management between the *Telegram* servers and the DB with the ability to write and erase. It extends possible connections to the system being an independent module. This middleware ends up being a bridge to the chatbot, also being able to deal with *collections*. This is an implemented feature offered to increase the relationship between *songs* users make. As a *song* is composed of several audio files, a *collection* is composed of several *songs* and can be initiated in the chat using the user command – *collection: <collection name string>*[8]. It is a specific server connected to the other modules that can establish links with

[7] https://www.apollographql.com/.

[8] Collections are presented as a functionality although no spam control structure has been made.

other web applications. Some of its functions – providing bidirectionality – are listed in the following table.

Table 1. Proposed API used to link the middleware with *Telegram* and the DB.

Programmed Function API	Result
GetSongs()	Returns all songs from the DB with the parameters: ID, title and collection.
GetSongsByCollection("collectionName")	Returns all songs belonging to a specific collection. Same kind as the *GetSongs()* callback.
GetTracks("songID")	Given a song ID (long integer) it returns all tracks included in the song.
GetUserPermission("songID", "userINFO")	When a user is logged in, using *Telegram* widget, it returns whether the user has admin access to the given song or not. The ID is similar to *GetTracks()* and the *userINFO* is a JSON object, also used in *DeleteTrack()*.
fileDownload("fileID")	Given a unique file identifier it will send to the client the buffer of the file. This option has been implemented in order to not expose the bot token used at file URL. Currently the any way to access the audio buffer is in the front-end.
fileUpload("chatID", audio, "userINFO")	For a registered user with admin permissions to the song, it uploads a file automatically after drag-and-drop on the input box, or when a new track is recorded using the microphone – access permission must be granted to the web browser. This method internally calls *sendAudio()*, a *Telegram* API function which has a limit of 50Mb size for files.
DeleteTrack("trackINFO", "userINFO")	Erases a track from the external DB by passing its ID – by a user with permission. The current *Telegram* Bot API does not provide an update to receive notifications when a message (e.g. text or audio) is deleted from the chat. Currently, the provided solution is triggered by the *edit* event on a message, if that *edit* has a *delete* caption, then the chatbot removes the track from the DB.

4. The web browser application

As a practical example of the application of this chatbot, a web-based multitrack audio editor is presented. Extending the presented functionalities, we promote a collaborative ecosystem [Lazzarini et al. 2015], as well as a use case for music management, production and listening. A type of web application that uses the various functionalities provided, in which several users can collaborate, listen and compose. The client side is built with standard web languages, *JavaScript,* HTML and CSS [as in Werner et al. 2017] with the library *Bootstrap* 4 [Krause 2016]. For the multitrack editor the library *Waveform Playlist*[9] v.3.2.4 [Pauwels and Sandler 2018] is used, with features such as: 1) a timeline-based interface; 2) the ability to solo or mute each track; 3) time control and visualization; and 4) track markers [Mahadevan et al. 2015]. Using the automatic decision taken by the various modules, this multitrack editor presents a way for multiple users distributed around the world to contribute to the same end, adding content to a *song*. There are three pages: 1) the *landing* page with several buttons for each *song*; 2) the *collection* page which offers the possibility to see to which *collections* a specific *song* belongs to; and 3) the *detail* page which has the multitrack editor and a *drag-n-drop* zone allowing the web user to add audio files to the session then subsequently added to the DB. It is possible to record or upload audio but it is not possible to create a *song* project or a *collection*, that is just done in the *Telegram* application through the chatbot. Also, on each page there is an implementation of the *Telegram* login widget[10] that allows each user to connect – if administrator of the chat group in question, enabling file upload or delete. When using the widget, the *bot token* (illustrated in Figure 1) is used in the

[9] https://github.com/naomiaro/waveform-playlist.
[10] https://core.telegram.org/widgets/login.

middleware to verify the hash obtained from the widget and it is also possible to locally search for *songs* or *chat groups* in the webpage – see Figure 3 for a diagram and two screenshots of the web application.

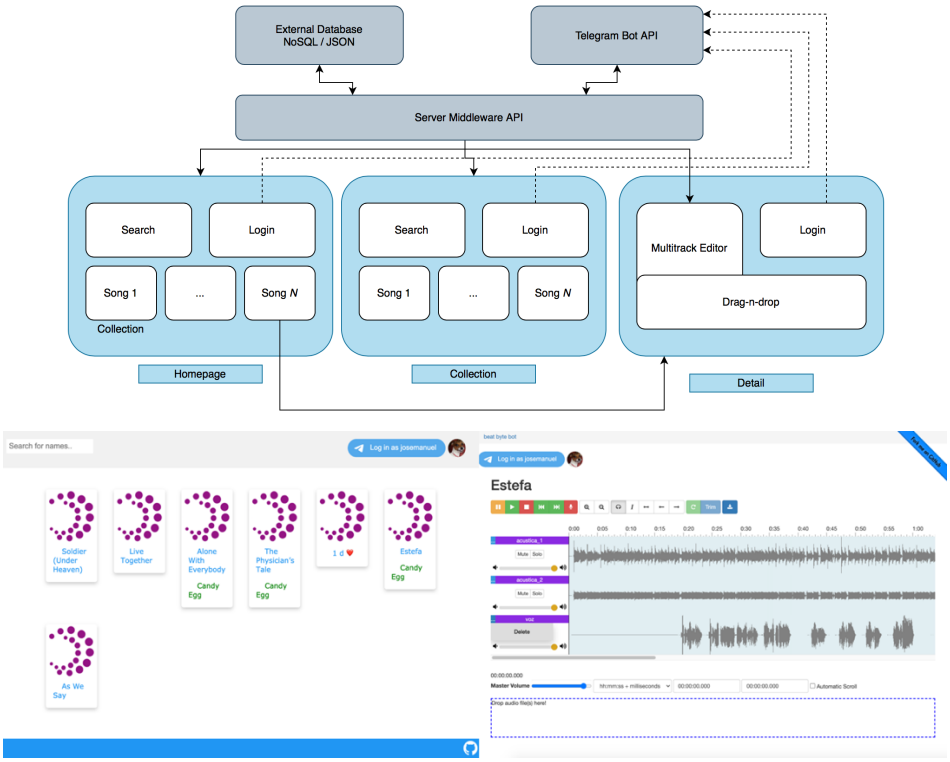

Figure 3. Diagram of the contents of the web application with links to the middleware API and two screenshots, the landing page and the multitrack editor.

4.1 Tests and Compatibility

In order to provide data regarding the use and support of the proposed software on common platforms, tests were developed. Focusing on the type of audio files and potential errors between the various modules, we used a *Telegram* application version 7.6.1 on an *iPhone* 7 running iOS 11.4.1 and the desktop version 2.7.1 on a macOS *Mojave* version 10.14.6. The multitrack tests were developed on a staging environment, version 0.0.1[11]. Results showed that the *.oga* file type is the default format used in *Telegram* applications for the voice recording option with the internal microphone, and for the browser is *wav* given the *MediaRecorder* API. The major browsers – *Firefox* 87.0, *Chrome* 89.0.4 and *Edge* 89.0.7 (on a Windows 10 Pro v.1909) – show the capability to present the multitrack editor apart from *Safari* 13.0.4, detailed in the next table. Regarding audio file types, the files: *mp3*, *ogg*, *m4a*, *acc* and *flac* show full compatibility with the upload both from the desktop application as well as through the multitrack editor. Issues show us that the *wav* type file when uploaded from the *Telegram* desktop application is recognized but doesn't play in the multitrack as well as *aiff* and *wma* that do not play and are not recognized – the main issue is that the library *telegraf.js* (version 3.38.0) recognizes them as documents instead of audio so they need

[11] https://bunchofsongs.web.app, https://github.com/gilpanal/beatbytebot_webapp.

a proper store in *Firebase*. As for the multitrack upload of *wav* files no problem is detected, *aiff* files are unable to decode using the *waveform-playlist* and *wma* is an unsupported file type.

Table 2. Compatibilities and issues regarding the platform.

Browser-based Issues

Platform	Multitrack Audio Controls and Visualization	Multitrack Audio Recording	Telegram Login Widget
Safari Build Version 13.0.4	1) .ogg and .oga audio format not supported when received from Telegram voice recording 2) Download .wav mix not working: error AudioContext()	Not working: error in *MediaRecorder* API	No user's profile preview

Audio Upload Issues from Telegram Desktop Application

Platform	Format	Recognized as Audio in *Telegram*?	Played at Multitrack	Error
Telegram MacOS 2.7.1	.wav	Yes	No	Telegraf
	.aiff	No	No	Telegraf
	.wma	No	No	Telegraf
MacOS *Chrome* v. 89.0.4389.90	Audio Upload Issues from Web Application Multitrack Editor			
	Format	Uploaded and Played in Multitrack?	Played in *Telegram* App?	Error
	.wav	Yes	Yes	-
	.aiff	No	No	Decode
	.wma	No	No	Type

Features found in other web-based multitrack editors [Buffa et al. 2015] can be implemented given the open nature of the libraries used, as well as trying to solve in our system, some of the problems found in decoding files [Jillings et al. 2016]. However, given the functionalities already presented and its cross-platform support, the web application proves to be a functional solution for the proposed chatbot, allowing several people to contribute, interact and make music together [Xambó et al. 2019]. We also underline that for a positive scalability, it will be necessary to develop adaptive processes of rules [see Arandas et al. 2019].

5. Discussion and future work

From the potentialities of the web, we can promote collaboration and interaction not only on a large scale but also ubiquitously [Stolfi et al. 2017]. As an augmented space for creation – given its network effect [Aggarwal and Philip 2012] – as well as the current technological advancement of its infrastructure, we developed research directed to a field that unites chatbots,

automatic data management and collaborative music applications [Roberts and Wakefield 2013]. An architecture designed to allow multiple users to interact together is proposed, taking into account previously proposed theories of ubiquitous music systems [Flores et al. 2009]. Given the lack of chatbots applied to the management of audio files, the proposed architecture links a virtual chat room – one of the most crowded places these days [Shih 2020] – with a platform for consumption and manipulation of audio material. This is done from a chatbot, an agent that exists in these same chat rooms and promotes all the structural manipulation of the data that users upload to participate. Taking advantage of the existing quality of the microphones in mobile devices, the quality that chat services put in audio recording, as well as the open APIs in the platforms used, we propose a collaborative and distributed system that can be used around the world, as a collaborative music platform. Users can use it in an open and inclusive way, democratising the system as a music platform through chat rooms.

The various services used for the prototyping and implementation of *Beat Byte Bot* serve as proof of concept, offering a starting point for future research. However, these should not be taken as the only option, as some of their development comes from the specificities of each platform and API. The architecture should be perceived as modular, extensible and changeable to other cloud platforms that minimally match the desired effect. In order to propose future work, we divide the proposals into two major paths: 1) compare with other platforms – both chats and server providers – that share similar characteristics, where we can raise e.g. quantitative tests in speed; and 2) the development of novel features in the project already developed, such as 2.2.) automatic signal processing in *Heroku* servers or in the front-end e.g. applying filters on the multitrack; 2.3) applying machine learning for better data clustering in *Firebase*; 2.4) the possibility of applying labels from the users to the various audio files in the collection they have access to.

6. Conclusion

This paper presents and discusses research into web audio music systems, chatbot architectures and data management. A review of relevant literature and web services matching the objectives of the explored solutions is made, a web system promoting collaboration and music assisted by chatbots and the cloud is proposed, the process is described and results are presented. It is on the pervasiveness of chatrooms and the possible musical applications to develop from them that the presented research focuses. A new extensible architecture is created and proposed – *Beat Byte Bot* – that helps to push the boundaries of what is normally thought of as musical collaboration by bridging different platforms in the cloud. Focusing specifically on the *Telegram* services, the various modules presented were developed with extensibility in mind, as well as the possible integrations with other web services. The proposed system that can be used from anywhere in the world promoting a valid ecosystem for music creation in an area with both scientific and creative potential.

Acknowledgements

The research leading to these results was financially supported by the Portuguese Foundation for Science and Technology (FCT), through the individual research grant 2020.07619.BD and by the project "Experimentation in music in Portuguese culture: History, contexts and practices in the 20th and 21st centuries", co-financed by the European Union, through the Operational Programme Competitiveness and Internationalization, in its ERDF component, and by national funds, through the Portuguese Foundation for Science and Technology.

References

Adamopoulou, E., & Moussiades, L. (2020). Chatbots: History, technology, and applications. Machine Learning with Applications, 2, 100006.

Aggarwal, Charu C., and S. Yu Philip. "On the network effect in Web 2.0 applications." *Electronic Commerce Research and Applications* 11.2 (2012): 142-151.

Andersson, Niklas, and Aleksandr Chernov. "Increasing the Throughput of a Node.js Application: Running on the Heroku Cloud App Platform." (2016).

Arandas, L., Gomes, J., and Rui Penha. "Towards large-scale artistic practice with web technologies." *Web Audio Conference 2019 - Diversity in Web Audio*. Norwegian University of Science and Technology, 2019.

Bii, P. K., J. K. Too, and C. W. Mukwa. "Teacher Attitude towards Use of Chatbots in Routine Teaching." *Universal Journal of Educational Research* 6.7 (2018): 1586-1597.

Brandtzaeg, Petter Bae, and Asbjørn Følstad. "Chatbots: changing user needs and motivations." *Interactions* 25.5 (2018): 38-43.

Buffa, Michel, Amine Hallili, and Philippe Renevier. "MT5: a HTML5 multitrack player for musicians." *WAC 1st Web Audio Conference January 26-28, 2015-IRCAM & Mozilla Paris, France*. 2015.

Daniel, Gwendal, et al. "Multi-platform chatbot modeling and deployment with the Jarvis framework." *International Conference on Advanced Information Systems Engineering*. Springer, Cham, 2019.

Følstad, Asbjørn, and Petter Bae Brandtzæg. "Chatbots and the new world of HCI." *interactions* 24.4 (2017): 38-42.

Flores, L.V., Pimenta, M., Capasso, A., Tinajero, P., & Keller, D. (2009). Ubiquitous music : concepts and metaphors.

Freed, Ned, and Nathaniel Borenstein. "Multipurpose internet mail extensions (MIME) part two: Media types." (1996): 10.

Godse, N. A., Deodhar, S., Raut, S., & Jagdale, P. (2018, August). Implementation of chatbot for ITSM application Using IBM watson. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)* (pp. 1-5). IEEE.

Harchol-Balter, Mor, et al. "Size-based scheduling to improve web performance." *ACM Transactions on Computer Systems (TOCS)* 21.2 (2003): 207-233.

Hasyim, M. W., and S. Pramono. "Web-Based Telegram Chatbot Management System: Create Chatbot Without Programming Language Requirements." *IOP Conference Series: Materials Science and Engineering*. Vol. 1096. No. 1. IOP Publishing, 2021.

Jannat, Rahatul, et al. "Ubiquitous emotion recognition using audio and video data." *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*. 2018.

Jillings, Nicholas, et al. "Web Audio Evaluation Tool: A framework for subjective assessment of audio." (2016): 1-4.

Jin, Yucheng, et al. "MusicBot: Evaluating critiquing-based music recommenders with conversational interaction." *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2019.

Khawas, Chunnu, and Pritam Shah. "Application of firebase in android app development-a study." *International Journal of Computer Applications* 179.46 (2018): 49-53.

Kim, Geun-Hyung. "How Will Blockchain Technology Affect the Future of the Internet?." *Advances in Computer Science and Ubiquitous Computing*. Springer, Singapore, 2021. 289-294.

Kozhevnikov, V. A., & Pankratova, E. S. (2018). Development of an intelligent recommender assistant using the telegram platform. Theoretical & Applied Science, (5), 77-83.

Kozhevnikov, Vadim Andreevich, Oleg Yurievich Sabinin, and Julia Efimovna Shats. "Library development for creating bots on Slack, Telegram and Facebook Messengers." *Theoretical & Applied Science* 6 (2017): 59-62.

Krause, Jörg. *Introducing Bootstrap 4*. Apress, 2016.

Lazzarini, Victor, Damian Keller, and Marcelo Soares Pimenta. "Prototyping of ubiquitous music ecosystems." *Journal of Cases on Information Technology (JCIT)* 17.4 (2015): 73-85.

Liang, L., Zhu, L., Shang, W., Feng, D., & Xiao, Z. (2017, May). Express supervision system based on NodeJS and MongoDB. In *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)* (pp. 607-612). IEEE.

Lind, Fredrik, and Andrew MacPherson. "Soundtrap: A collaborative music studio with Web Audio." (2017).

Mahadevan, Anand, et al. "EarSketch: Teaching computational music remixing in an online Web Audio based learning environment." *Web Audio Conference*. 2015.

Mardan, Azat. "Deploying Node.js Apps." *Practical Node.js*. Apress, Berkeley, CA, 2018. 365-388.

Medina, José, et al. "Audio fingerprint parameterization for multimedia advertising identification." *2017 IEEE Second Ecuador Technical Chapters Meeting (ETCM)*. IEEE, 2017.

Moroney, Laurence. "The firebase realtime database." *The Definitive Guide to Firebase*. Apress, Berkeley, CA, 2017. 51-71.

Pauwels, Johan, and M. Sandler. "pywebaudioplayer: Bridging the gap between audio processing code and attractive visualisations based on web technology." (2018).

Porcello, E., & Banks, A. (2018). *Learning GraphQL: declarative data fetching for modern web apps*. " O'Reilly Media, Inc.".

Roberts, Charles, Graham Wakefield, and Matthew Wright. "The Web Browser As Synthesizer And Interface." *NIME*. 2013.

Samizadeh Nikoui, Tina, et al. "Internet of Things architecture challenges: A systematic review." *International Journal of Communication Systems* 34.4 (2021): e4678.

Sari, A. C., Virnilia, N., Susanto, J. T., Phiedono, K. A., & Hartono, T. K. Chatbot Developments in The Business World. *Advances in Science, Technology and Engineering Systems Journal* Vol.

5, *No. 6, 627-635* (2020).

Saribekyan, Hayk, and Akaki Margvelashvili. "Security analysis of Telegram." *Diakses tanggal* 15 (2017).

Shakhovska, Nataliya, Oleh Basystiuk, and Khrystyna Shakhovska. "Development of the Speech-to-Text Chatbot Interface Based on Google API." *MoMLeT*. 2019.

Shih, Yen-An, and Ben Chang. "Empirical study on the effects of social network–supported group concept mapping." *Research and Practice in Technology Enhanced Learning* 15.1 (2020): 1-22.

Smith, Ben. *Beginning JSON*. Apress, 2015.

Stallman, Richard. Free software, free society: Selected essays of Richard M. Stallman Lulu. Com, 2002.

Stolfi, Ariane, et al. "Open band: Audience creative participation using web audio synthesis." (2017).

Tajane, K., Dave, S., Jahagirdar, P., Ghadge, A., & Musale, A. (2018, August). AI Based Chat-Bot Using Azure Cognitive Services. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)* (pp. 1-4). IEEE.

Tanna, Mayur, and Harmeet Singh. *Serverless Web Applications with React and Firebase: Develop real-time applications for web and mobile platforms*. Packt Publishing Ltd, 2018.

Werner, Nils, et al. "trackswitch.js: A versatile web-based audio player for presenting scientific results." (2017).

Wittern, Erik, Philippe Suter, and Shriram Rajagopalan. "A look at the dynamics of the JavaScript package ecosystem." *Proceedings of the 13th International Conference on Mining Software Repositories*. 2016.

Xambo Sedo, A., Støckert, R., Jensenius, A. R., & Saue, S. (2019). Facilitating team-based programming learning with web audio. In *Proceedings of the International Web Audio Conference (WAC)* (pp. 2-7). NTNU.

Yan, Mengting, et al. "Building a chatbot with serverless computing." *Proceedings of the 1st International Workshop on Mashups of Things and APIs*. 2016.

Yucheng Jin, Wanling Cai, Li Chen, Nyi Nyi Htun, and Katrien Verbert. 2019. MusicBot: Evaluating Critiquing-Based Music Recommenders with Conversational Interaction. In Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM '19). Association for Computing Machinery, New York, NY, USA, 951–960. DOI:https://doi.org/10.1145/3357384.3357923.

Zimmermann, Roger, et al. "Audiopeer: A collaborative distributed audio chat system." *Distributed Multimedia Systems, San Jose, CA* (2004).