# Auditory Skills Web-Application in the Context of Software Engineering and Machine Learning

Stephen Owensey
Virginia Tech
McLean Virginia, USA
stepowes@gmail.com

## ABSTRACT

This paper delves into the profound significance of music in human life and details the implementation and ideas for the auditory skills application. The web application presented in this study serves as a front end, designed to elevate users' musical skills through an interactive game. Users can select difficulty levels, monitor their performance over time, and participate in friendly competitions via leaderboards. The game, implemented with React and TypeScript, hones users' relative pitch skills by challenging them to identify musical intervals. The Django-based backend manages user registration, game logic, and database operations, employing models defining schemas for user and game data. The paper highlights the project's potential for machine learning research, envisioning its application in auditory task analysis. Overall, this work offers a comprehensive exploration of music's role in human life, accompanied by a detailed description of a technology-integrated application with implications for user enhancement and broader scientific inquiry.

**ACM Reference Format:**
Stephen Owensey. 2018. Auditory Skills Web-Application in the Context of Software Engineering and Machine Learning. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 INTRODUCTION

Music plays a significant role in our daily lives. What defines music is widely debated, but it is a definitionally agreed upon fact that music consists of some intentionally created auditory experience that is to be listened to. Music involves three main components: rhythm, melody, and harmony. No matter what age you are, what profession you are in, what language you speak, or where you come from, music is the universal language that connects us all. The appreciation for musical structure is deeply embedded in human nature. If you consider the fact that the universe operates in rhythm: from the celestial rhythms of the earth's orbit around the sun that give us night and day, to the ecological rhythms that govern seasonal migration of animals and the blooming of withering of plants, to the biological rhythms of heartbeat, cellular respiration, DNA replication, and any of the other feedback loops that mediate the existence of life, you can start to see why music is so important. The organism is a model of it's environment where longer lasting phenomena like photons to be seen and vibrations to be heard are more fundamental attributes of reality and correspond to more permanent structures like an eye or ear while the nuances of society change year by year and are represented in the mind conceptually and linguistically, mediated far more transient structures. We are constantly observing the patterns in our environment in an attempt to leverage them to our advantage, patterns that belong to a interconnected mural of quickly changing things that are embedded within less quickly changing things and so on and so forth. That is exactly the structure of a song, where the bass changes less frequently than the harmonies, which change less frequently than the melodies, and so on. In this manner, music is a very unique art form that differs from the way visual arts represent reality with space and color as snapshot in time, or than how theater and film represent reality encoded in patterns of behavior and archetypes of the subjective experience. Music represents the nature of reality in the most abstract form: via mimicry of the dynamic patterns of structure and behavior that for which propagate it through time and space, which is why organized noises are so mystically yet undeniably meaningful to us.

In the western musical system, only 12 notes constitute all melodies and harmony: C, C sharp/D flat, D, D sharp/E flat, E, F, F sharp/G flat, G, G sharp/A flat, A, A sharp/B flat, and B in ascending order. These notes can all exist within different octave registers defined by the octave number. Notes correspond to physical frequencies measured in hertz and each note has a distinct ratio that defines its relationship mathematically to any other note. If you focus on one note in the scale such as C and permute it with every other note available in the western musical system, you will get every interval relationship. The most fundamental interval is called the octave, which is defined by the ratio 1:2. The term "octave," derived from Latin, represents a musical interval that spans the aforementioned seven letter named notes once through, such as from C0 to C1 or D4 to D3. This concept traces back to ancient Greece, where people realized that sounds with frequencies in rational proportions were harmonious concerning the lengths of vibrating strings, rather than frequencies. Pythagoras further substantiated this idea by subdividing vibrating strings into rational proportions, creating consonant sounds that allowed the audience in ancient Greece to easily distinguish each interval. Every mammal


*Conference acronym 'XX, June 03–05, 2018, Woodstock, NY*

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

can distinguish the octave interval, however it is common knowledge that any person can be trained to identify any of the intervals defined in the western musical system by ear. This skill is known as relative pitch and is the skill that our interval quiz is largely concerned with developing.

## 2 FRONT END

The front end of this project is a web application that serves as the medium for which the users use interact with the game. The framework we chose to accomplish this is React, a JavaScript library that provides a declarative and efficient way to build reusable UI components. The programming language we used for our implementation is TypeScript, an super-set of JavaScript that adds static typing to the language allowing for a more readable and well defined programmatic flow. Typescript is commonly used for building web applications providing text inline styling, animation, etc., all of which is used to design, visualize, and implement the interactive user interface for our game.

### 2.1 Router

The router at the front end can best be depicted as the technique at which the user can navigate to different websites they prefer to take a look at. For instance, a user currently staying at a Canvas Homepage at Virginia Tech canvas.vt.edu wants to go to CS4664 Data-Centric Computing Capstone course page. In order for him to successfully enter into this course page, the routers can be defined as follows.

```
<Route path= "/" element={<Canvas />}
<Route path="/canvas/CS4664_DCC_Capstone"
       element={<CS4664 />} />
```

where the names of both the path of the router and the webpage can be configured, depending upon the user's interest.

Given the scenarios mentioned above, the routers at the software project can be briefly outlines as follows.

```
<Route path="/"
       element={<Login />} />
<Route path="/register"
       element={<Register />} />
<Route path="/register/success"
       element={<RegisterSuccess />} />
<Route path="/menu"
       element={<Menu />} />
<Route path="/menu/userdata"
       element={<UserRecord />} />
<Route path="/menu/gamestart"
       element={<GameMode />} />
<Route path="/menu/gamestart/play"
       element={<Game />}/>
```

### 2.2 Login/Register

Once the routers are all set up, it is better to start with both the login and the register pages to guarantee the interconnection between two pages. To accomplish this particular task, the important information associated with the register page frequently seen by people are used, including but not limited to first name, last name, email address, password, and confirmed password. More specific details with regard to the fields at a register page can be found as below.

- first name: string
- last name: string
- username: string
- age: int
- musical year: int
- email: email
- password: password
- confirmed Password: password

Before clicking the register button at the register page, we need to ensure that the password and the confirmed Password must be congruent in order to avoid any ambiguities confused by the system. Additionally, the fields username and email are checked against the database of users to make sure they are both unique because these are defined as unique fields in our system. After an account is established, a user is created in the database and the username and password created are used to log into the game page successfully, which will be explained later.

### 2.3 User's Menu Page

Based on the previous operations mentioned above the user's menu page is loaded upon login where the user is welcomed by their username and there are several actions that the user can take. The four buttons available to be selected are as follows.

- Games
- User Data
- Leaderboard
- Log Out

### 2.4 Games

The most crucial part of the front end section is indeed the game that the user interacts with to help practice their musical skills and enhance their abilities. When the 'Games' button is selected there are currently 4 levels of difficulties to choose from: *easy*, *medium*, *hard*, and *insane* which will decide how the questions are generated in the backend. Once a game mode is chosen, the user is navigated to an intermediate page where they are given the option to start the game with the game mode given or cancel this process. Once the start button is clicked, the questions are fetched from the back end using the difficulty to guide the question generation. The back end generates the questions and stores them in a directory in the front end as mp3 files. After the data is done loading into the frontend, the game begins by automatically sounding the mp3 file for the current question. Provided is a sound icon that the user can click to resound the interval in case they weren't paying attention during the automatic sounding of the interval upon the initial loading of the question. There is a button for each musical interval that spans the octave abbreviated by its proper category and numeric association. If the answer selected corresponds to the interval that was used to generate the sound file, the answer selected will be highlighted in green indicating to the user that they were correct in their choice while all other buttons turn red. Otherwise, the answer selected turns red indicating to the user that this was the incorrect choice and all of the other answers besides the correct one turn red

and the correct answer turns green, showing the user what they should have chosen. All the while a current score, question number, and timer are displayed at the top of the screen. When the answer of the last question is clicked, the player can choose to either play the game again or view his current and previous records at User Record page, which will be presented next.

## 2.5 User Data

This section of the front end allows the user to access a view of their performance over time. This is an important feature for the user to have so that they can analyze how their performance is changing across time to see improvements and patterns in their abilities. Upon selection of the 'User Data' button, the user is navigated to a page containing a scatter-plot that allows the user to visualize all of the game attempts that user has made where the x-axis contains a window of time and the y-axis represents the user's score on the game. There are two drop down menus, one for each axis of the scatter-plot. For the x-axis, the drop down menu contains the options *All*, *Day*, *Month*, and *Year*. Selecting one of these actually resizes the window of time the data fits within, specifying the query accordingly and reshaping the x-axis and the data that is rendered. Similarly for the y-axis, the drop down menu contains the options *All*, *Easy*, *Medium*, *Hard*, and *Insane*. Selecting one of these options makes a detailed query that only returns the game data that adheres to the difficulty chosen. In both cases, the option *All* loosens the query to include all of the data for that dimension. Each data point that is rendered onto the scatter-plot represents a game that the user has played and hovering over the data point will provide a tool tip that contains information such as the specific date it was played, the score, as well as the difficulty of the game that was played. This empowers the user to visualize their own data easily giving them control over specific querying parameters.

## 2.6 Leaderboard

This section of the front end is dedicated to make the game a little more fun when taking into account other users that have been playing the game. This introduces a little bit of healthy competitive spirit so that the user is incentivized to practice their skills and perform well while practicing so they can continue to improve and try to beat others' scores on the leaderboard. When the 'Leaderboard' button is selected a new page is loaded that contains a drop down menu and a list of users. The drop down menu provides the options *Easy*, *Medium*, *Hard*, and *Insane*, which represent the game mode that the leader-board pertains to. When selecting a difficulty, the list of users is dynamically rendered based on the top 10 players of that particular game mode displaying their individual ranking from ranks 1 to 10, their username, as well as the performance score that they are ranked by. This performance score is computed by an equation in the back end that will later be explain in more depth.

# 3 BACK END

The backend server of our web application handles all of the logic for the application including user registration, user verification, logic for the interval quiz game, as well as audio generation for the quiz questions and database related tasks such as storing the user base and related data. The framework we chose to use is Django, a high-level open-source web framework for the Python programming language. It is designed to simplify and speed up the development of web applications by providing a robust set of tools and conventions. Django follows the model-view-controller (MVC) architectural pattern (internally referred to as the model-view-template (MVT) pattern). This architectural pattern handles a lot of the communication between the frontend and the backend of our application. In general, the views handle API requests from the front end, receiving and managing any data communicated to the back end in the request and preparing any data that is to be send back in the response. While the models are used to design the schema for the database, defining fields that generate lookup tables in the database and handle instantiations of the model, loading data into the database governed by the fields. Django was a strong choice for the purposes of this application because it handles a lot of the heavy lifting in regards to the databases, security features, and implementation of a userbase. For our application, the main driving features of the backend include the game engine that generates games implemented in 'interval-quiz.py', the helper files that aid the generation of games in the files 'audio.py' as well as 'musical-data.py', and the implementation of the model and view aspects of the MVT architectural pattern provided by Django implemented in 'models.py' and 'views.py' respectively.

## 3.1 Musical Data Utility Module

The musical data utility module implemented in 'musical-data.py' is used to define information relevant to the operations performed for the construction and execution of the interval quiz. It is actually very simple containing 2 main data structure definitions and 1 single function.

The first structure labeled 'INTERVALS' is a dictionary of dictionaries where the key of the dictionary is a string denoting the abbreviated musical interval. There is a key for each musical interval involved in the quiz and these keys map to a dictionary each containing two main elements. The first element defines a 'name' feature which simply maps to the full name of the interval. For example, for the interval abbreviated 'P8', it contains a name key that maps to the string 'Perfect Octave'. The second element defines a 'ratio' feature which maps to a float that represents the ratio.

The second structure labeled 'NOTES' is also a dictionary of dictionary which is structured very similarly to 'INTERVALS'. In this case, the structure contains all 12 musical tones recognized by the western musical system as keys abbreviated as strings. These note name keys map to dictionaries each containing a 'name' feature which maps to the name of the note including its octave and the second feature labeled 'frequency' maps to a float that represents the frequency value associated with the note in hertz. In this case, the 12 musical tones that are listed are all part of the 0 octave. These are the only frequencies that are stored because the following function to be described allows for any other frequency to be generated.

The only function in this utility module is called 'getNoteFrequency' and takes in 4 parameters. The first parameter is a string value 'noteName' that represents the note name, the second is an integer value 'octave' that represents the octave number, the third is a string value 'interval' that represents the interval, and the fourth parameter is an integer value 'decreasing' which is 1 when the

interval is to ascend and -1 when the interval is to descend. The purpose of this function is to not only allow for a single note's frequency to be calculated, but also to allow for a calculation of a second frequency given a first note and an interval. If you simply need to look up the frequency of a particular note given a note name and the octave register you desire, the interval can be set to 'P1' or perfect unison whose ratio is simply 1:1. However, if you have a initial frequency and you want to calculate the frequency that is a particular interval above or below that note you can provide the interval and the state of decreasing in order to calculate the relative frequency of a note.

## 3.2 Audio Utility Module

The audio utility module utilizes the Pydub library to create and manipulate audio tones for the interval quiz.

The 'generate-tone' function generates a tone based on a given frequency, duration, and an optional volume parameter. It utilizes NumPy to create audio samples, converts them to a Pydub AudioSegment object, and returns the generated audio.

The 'createQuestion' function takes in two frequencies (freq1 and freq2), a duration, a timbre parameter, and a filename. It generates two tones based on the provided frequencies and duration using the generate-tone function. The tones are then concatenated, and the resulting audio is exported as an MP3 file with the specified filename to a directory in the front end that aggregates the mp3 files to be played for each question of the game.

## 3.3 Interval Quiz Class

The module starts by importing essential functions from the "musicaldata" and "audio" modules, along with standard libraries like os and random. It establishes the current and MP3 directories, crucial for file operations. The main focus is on the IntervalQuiz class, which initializes with the chosen difficulty level, managing game parameters such as score and question data. It dynamically sets difficulty-specific parameters based on the provided difficulty level, ensuring diverse and challenging questions for the quiz.

The core functionality lies in the generateQuestion method of the IntervalQuiz class. This method is responsible for randomizing musical elements based on the specified difficulty. It ensures the generation of varied questions by selecting random octaves, notes, and intervals within the defined ranges. The method handles both ascending and descending intervals, providing a comprehensive set of questions. The resulting question data, including frequencies and correct answers, is stored in the questionsData dictionary. Additionally, the method utilizes the "audio" module's createQuestion function to generate audio questions for the quiz.

A complementary function, clearDirectory, assists in maintaining a clean MP3 directory by deleting its contents. This function is crucial before generating new quiz questions to prevent any interference with previous audio files and is invoked when a new interval quiz object is instantiated.

## 3.4 Models

As previously mentioned the models in this file provide an object oriented way to load data into the database that is created using Django's MVT pattern. When updates are made to the models.py file, it is crucial to generate migrations. Migrations in Django are a way of propagating changes made to the models into the database schema. These changes could include creating new tables, modifying existing ones, or defining relationships between tables. By generating migrations, the developer ensures that the database schema stays synchronized with the evolving structure of the application. In our application, we have 3 main schemas defined: one for the user, one for a general gamedata as to keep our application extensible for the insertion of future games, as well as one reified game: interval quiz data.

The 'CustomUser' model extends the built-in AbstractUser model provided by Django, encapsulating data related to users. It includes fields such as age, dateCreated, firstname, lastname, email, and musicalYear. These fields store user-specific information, such as the user's age, the date of account creation, first and last names, unique email address, and the musical year. The model also establishes relationships with other Django models by incorporating fields like groups and user-permissions, both defined as ManyToManyFields. These fields enable associations with Django's Group and Permission models, respectively.

The 'GameData' model represents data related to a game played within the application. It features fields such as id (UUIDField), gameName, user (ForeignKey to CustomUser), datePlayed, score, numOfQuestions, and totalTime. The id field serves as the primary key, ensuring a unique identifier for each game record. The gameName field stores the name of the game, while user establishes a foreign key relationship with the CustomUser model, linking each game to a specific user. Other fields, such as datePlayed, score, numOfQuestions, and totalTime, capture additional information about the game instance.

The IntervalQuizData model inherits from GameData and introduces fields specific to an interval quiz game. These additional fields include gameDifficulty, questionsData, and userAnswers. The gameDifficulty field stores the difficulty level of the quiz, while questionsData and userAnswers are JSONFields, accommodating the storage of complex data structures. These fields store information about the questions generated for the quiz and the corresponding user answers, respectively. The inheritance from GameData implies that IntervalQuizData inherits all the fields and behaviors of GameData and extends it with quiz-specific attributes.

## 3.5 Views

As previously mentioned, views serve as the central component responsible for handling HTTP requests and generating appropriate responses in Django's MVT pattern. A view is essentially a Python function or class that takes a web request as input and returns a web response as output. In our application, views are the highest level that assert control and modulation over all other responsibilities of the backend including generating games, authenticating users, instantiating instances in the models framework to store data in the database, and making queries to the database.

The 'CreateUserView' derived from Django's APIView, handles user registration. In the post method, it utilizes the CustomUserSerializer defined in 'serializers.py' to validate and save user registration data. In the Meta class within the serializer, the model attribute is set to CustomUser, and the fields attribute specifies the fields

that should be included in the serialized data. These fields include 'lastname', 'firstname', 'username', 'age', 'musicalYear', 'email', and 'password'. The create method within the serializer is overridden to customize the creation logic when creating a new user instance. The method is responsible for handling the validation and creation of a new user object. If the data is valid, it creates a new user and returns the user data with a success status; otherwise, it returns validation errors.

The 'LoginView' derived from Django's View, manages user authentication and login. The post method handles POST requests containing login credentials. It authenticates the user and, if successful, logs them in. Upon successful login, it returns a success message with the username; otherwise, it returns an error message for invalid credentials. Normally, this view would generate a user session key using cookies from the front end which would later be passed through requests handled by other views, however in the case of our application if we make a request relevant to a specific user's information we simply pass the username within that request to associate the request with the user due to issues in cross-site request forgery (csrf) tokenization handling. If this application were to ever actually be deployed, this method we have used temporarily would have to be replaced with the typical session tracking methods provided by Django for security purposes.

The 'StartGameView' derived from Django's APIView, initiates a new game for the user. The post method processes requests to start a game, extracting information about the game difficulty and the current user. It initializes a game using the IntervalQuiz class and returns a response with a message and questions data. If questions fail to generate, it returns an error. The information passed to the front end is used to verify the users answers to the questions in real time as the user plays the game as well as calculate the user's score.

The 'SaveGameDataView' derived from Django's APIView, saves game data after a user completes a game. The post method extracts data from the request, including the username, score, total time, user answers, questions data, and difficulty. It then saves this game data as an IntervalQuizData instance associated with the corresponding user. The post function also handles error articulation for specific cases where the user doesn't exist in the database or for if some how more than one user is found in the database, which should not be possible based on how the CustomUser model is defined in models.py.

The 'LeaderboardView' derived from Django's View, provides a leaderboard of user performances by making queries to the database. It does this by defining a metric called performance score which is essentially the percentage of the answers the user got correct divided by the amount of total time it took for them to complete the game. In this way, the user's performance is not only dependent on accuracy but also speed which indicates a level of certainty in the user's processing ability of the intervals. The get method handles performs this calculation by first partitioning up all of the data in the interval quiz data tables defined by the IntervalQuizData model by difficulty, then for each user in each difficulty partition, performance scores are extracted from each game and then averaged accross each user's history of games for that particular difficulty. Finally, each users average for each game mode are sorted in descending order taking the top 10 user performances for each difficulty and returning this data to the front end in a json format mapping the groups of users by difficulty.

The 'UserRecordView' derived from Django's View, retrieves user records based on certain criteria. In this case, unlike the data that the LeaderboardView returns which is limited to 10 users per gamemode, a comparatively small amount of information, the UserRecordView returns all of the games played by a given user which could theoretically be any amount of games associated with this user. For this reason, it is designed in the front end so that the time window and difficulty are passed to the backend so the query can be tailored to these parameters accordingly, reducing unnecessary computational load on the system. This process is handled by the POST method which retrieves user data from IntervalQuizData, filters it based on the specified criteria, and returns data in json format that is serialized to include only the features relevant to the user record interface; namely: date played, score, total time, number of questions, and game difficulty.

### 3.6 Other Features

Some additional features of the backend include some scripts located in the auditory-skills/management/commands. These were created for the superusers of our application and include scripts that perform some tasks useful for testing, visualizing database content, or provide control over the database such as querying games given a username as an argument, querying all users in the userbase, populating the database with a fake user with randomized user fields and games played, erasing game history of a user passed in as argument, or erasing the profile of a given user passed.

Some other files that are necessary to the functioning of the Django project include 'urls.py' which define url patterns that are able to communicate with the server, as well as 'settings.py' that define a lot of the setup and behavior of the Django project.

## 4 FUTURE FEATURES, GOALS, AND APPLICATIONS

A very interesting opportunity that opens up when you build engines that can generate theoretically endless amounts of data in the manner that our interval quiz can is elucidated by the question: how can one make use of this data in other ways than just allowing users to interact with the raw data? Our original idea was to build this application not only as an interactive and challenging experience for human users, but to also use this engine to generate data that can be used to conduct an comparative analysis on the efficacy of various machine learning architectures and model instantiations to complete the same task that our human users are training: identifying intervals just with the sound data from an mp3 file. Using genetic algorithms to refine each candidate model, we can perform a sort of speciation of neuroevolutionary training of various architectures and model configurations. The design of our software was done to the best of our ability to keep in mind extensibility and scalability. In this way, articulating the logic of many other auditory tasks such as chord identification, perfect pitch identification, chord progression identification, or other tasks can easily be plugged into our software and the training of machine learning models on those tasks alike. There is a lot of room for improvement of this application for the user experience both in the

aesthetic aspects of the user interface as well as the functionality of the games the user interacts with to better achieve the end goal of making the user a more capable processor of auditory data. The more information the user can access about the way they play the game, the more inferences both the user and the program can make about the specific profile of skills the user has. This is not only interesting for the user to be able to observe, but also powerful and useful for the user to be aware of and the program can self modify its own logic in order to optimize the progress the user makes in improving these cognitive abilities. One can imagine the power of this system when considering how it can be applied in various contexts, such as perhaps the context of central auditory processing disorder (CAPD). This condition affects 3 to 5 percent of school aged children, causing difficulties in understanding auditory information such as speech which severely affects learning in school. A system that generally follows the concept of our application but is specifically contextualized for (CAPD) could not only help individual users with the disorder to continue to improve their personal skills, but could even potentially use neural network architecture's that are specifically designed to resemble the neural networks that comprise the auditory cortex of human beings to research the nature of this disorder in relation to the structure of the brain.

## 5 CODEBASE

Our codebase is relatively intuitive and simple to setup, navigate, and start using. For more nuanced version control during the development of our application, we kept the frontend and the backend in separate repositories labeled 'curly-disco' and 'music-engine' respectively. If you include these two repositories in the same directory, everything should function as intended in terms of references. To start the client, navigate to curly-disco music-quiz and run the command "npm start". To start the server, navigate to music-engine music-engine-django and run the command "python manage.py runserver". There will be a list of dependencies to install for the environment you intend to run the application in. You will have to resolve these dependencies so that the import statements across the codebase are satisfied with the properly libraries.

## 6 CONCLUSION

The assertion that humans are inherently attuned to musical structures due to the rhythmic nature of the universe underscores the intrinsic relationship between music and the human psyche. The web application presented as part of this study represents a tangible effort to harness technology for the enhancement of users' musical skills, providing an engaging platform for skill development, performance tracking, and friendly competition. The integration of React, TypeScript, and Django showcases the synergy between frontend and backend technologies to create a seamless and interactive user experience. Moreover, the exploration of the project's potential for machine learning research introduces an innovative dimension, envisioning applications beyond user improvement to broader scientific inquiry. As we contemplate the future, this work prompts us to consider the ever-expanding horizons where technology, music, and scientific exploration converge, offering promising avenues for understanding both the intricacies of human cognition and the broader dynamics of auditory processing. The journey from musical exploration to technological implementation and potential scientific contributions demonstrates the versatility and forward-looking nature of this interdisciplinary endeavor.