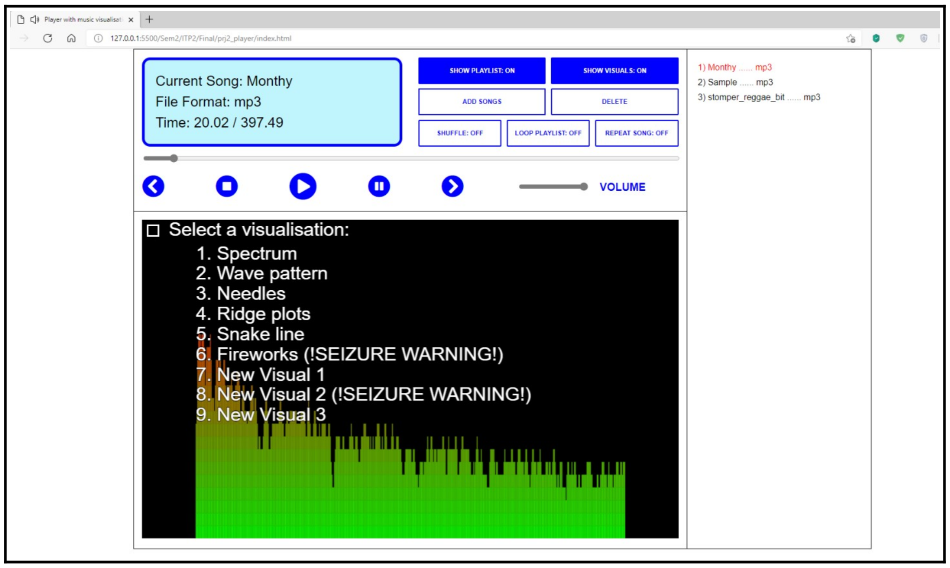
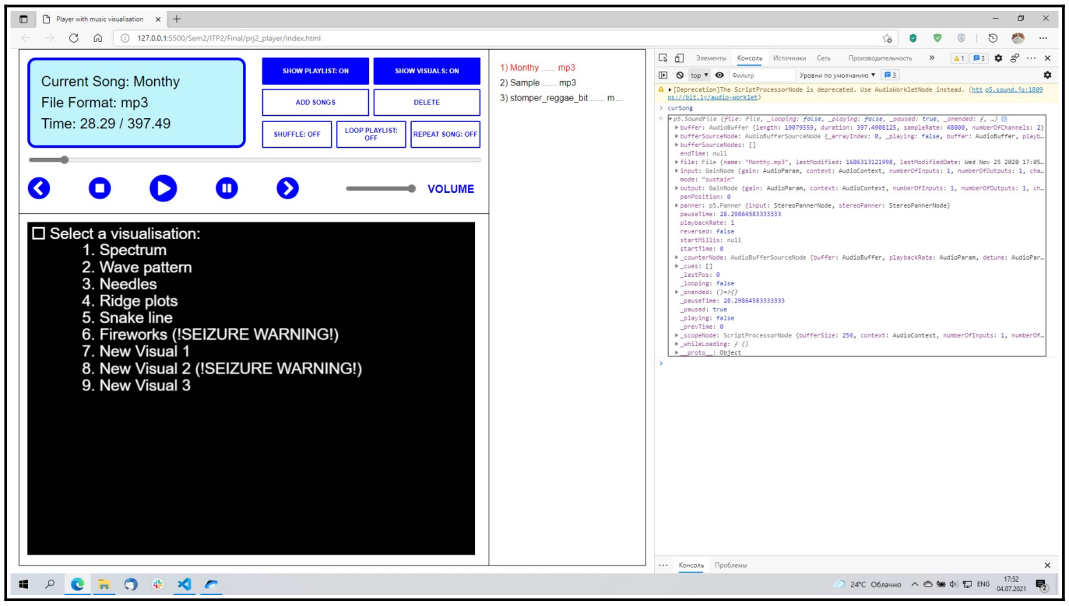
Part 3: Secure programming

For this part, I will try to analyze potential security problems and suggest potential solutions for the music visualizer app. I created a simple browser-based online audio player with an integrated visualization module.



There are five main categories for security concerns: Input validation, Buffer bounds, and memory, Design that abides by the security principles in software design, Careful use of side resources, and Program output in form of logs and other information.

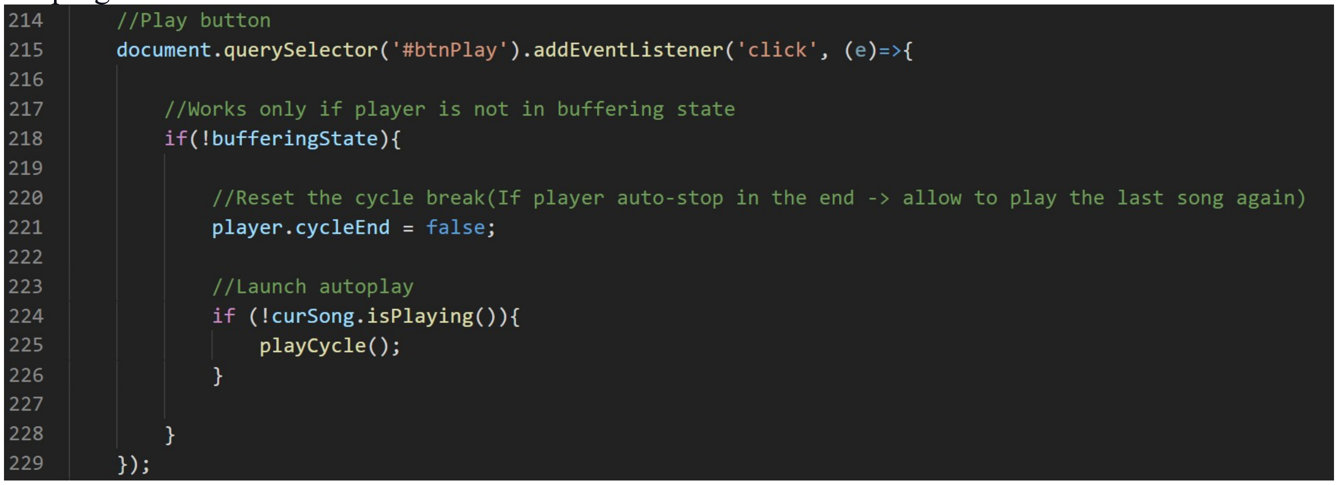
I start from the secure design perspective: there are many principles, that could be implemented, but I focus on two that I find most vulnerable - "Secure interface" and "Separate data and control". Citing D.A.Wheeler "Secure programming Howto" - "Interface should be minimal (simple as possible), narrow (provide only the functions needed), and non-bypassable". In the current state, the interface is simple and intuitive enough, but because the application is browser-based, it is bypassable.



```
[Deprecation]The ScriptProcessorNode is deprecated. Use AudioWorkletNode instead. (https://bit.ly/audio-worklet)   p5.sound.js:1809
> curSong
< p5.SoundFile {file: File, _looping: false, _playing: false, _paused: true, _onended: ƒ, …}
    buffer: AudioBuffer {length: 19079559, duration: 397.4908125, sampleRate: 48000, numberOfChannels: 2}
    bufferSourceNode: AudioBufferSourceNode {_arrayIndex: 0, _playing: false, buffer: AudioBuffer, playb…
    bufferSourceNodes: []
    endTime: null
    file: File {name: "Monthy.mp3", lastModified: 1606313121998, lastModifiedDate: Wed Nov 25 2020 17:05…
    input: GainNode {gain: AudioParam, context: AudioContext, numberOfInputs: 1, numberOfOutputs: 1, cha…
    mode: "sustain"
    output: GainNode {gain: AudioParam, context: AudioContext, numberOfInputs: 1, numberOfOutputs: 1, ch…
    panPosition: 0
    panner: p5.Panner {input: StereoPannerNode, stereoPanner: StereoPannerNode}
    pauseTime: 28.29864583333333
    playbackRate: 1
    reversed: false
    startMillis: null
    startTime: 0
    _counterNode: AudioBufferSourceNode {buffer: AudioBuffer, playbackRate: AudioParam, detune: AudioPar…
    _cues: []
    _lastPos: 0
    _looping: false
    _onended: ()=>{}
    _pauseTime: 28.29864583333333
    _paused: true
    _playing: false
    _prevTime: 0
    _scopeNode: ScriptProcessorNode {bufferSize: 256, context: AudioContext, numberOfInputs: 1, numberOf…
    _whileLoading: ƒ ()
    __proto__: Object
>
```

Users can use web-browser functional to access the inner workings of the program.
To prevent this, we could think about switching from the web-based platform to a full-fledged executable file that doesn't depend on another application to work. For example, we can rebuild this application using the Node.js framework to make it a standalone desktop application.

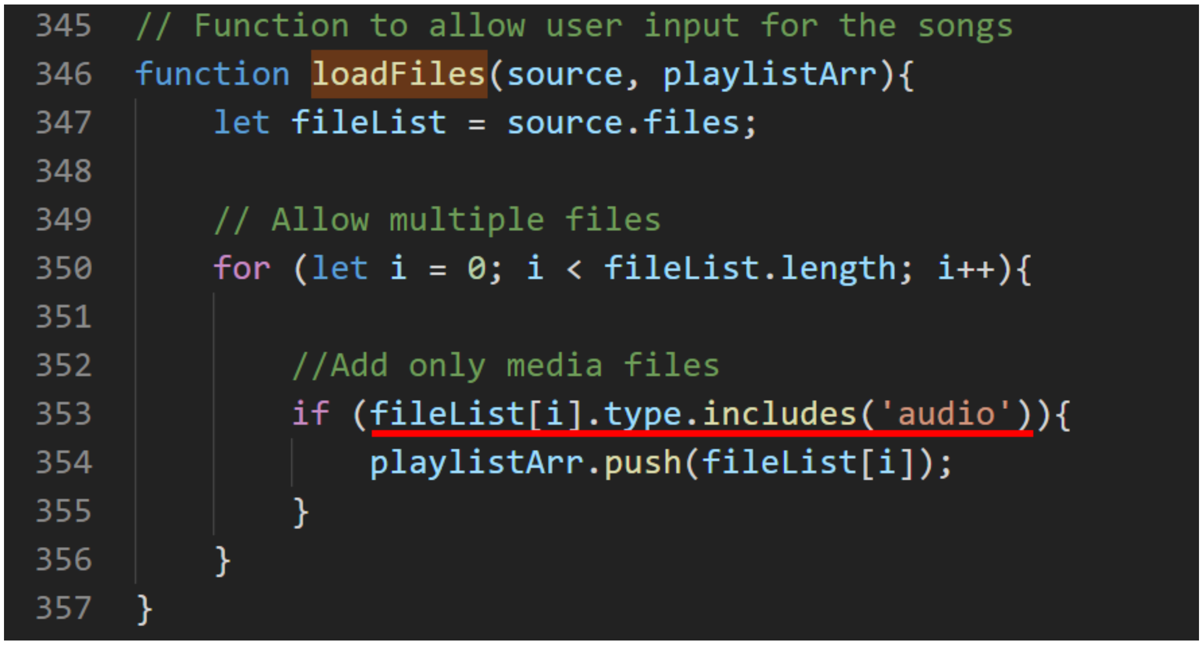
Another serious problem is the lack of OOP principles and the absence of functional separation. In my version, each interface element is tied to the function that directly affects the data that the program uses.



```
//Play button
document.querySelector('#btnPlay').addEventListener('click', (e)=>{

    //Works only if player is not in buffering state
    if(!bufferingState){

        //Reset the cycle break(If player auto-stop in the end -> allow to play the last song again)
        player.cycleEnd = false;

        //Launch autoplay
        if (!curSong.isPlaying()){
            playCycle();
        }

    }
});
```

To fix this problem, I propose to refactor the code to follow the Model-View-Controller (MVC) pattern. MVC is based on the separation of the program logic into three major blocks: model - to work with data, view - to create the user interface and register inputs, and controller - that interconnects the other two. We divide the code into three large blocks, encapsulate them and connect them only through the program interface.

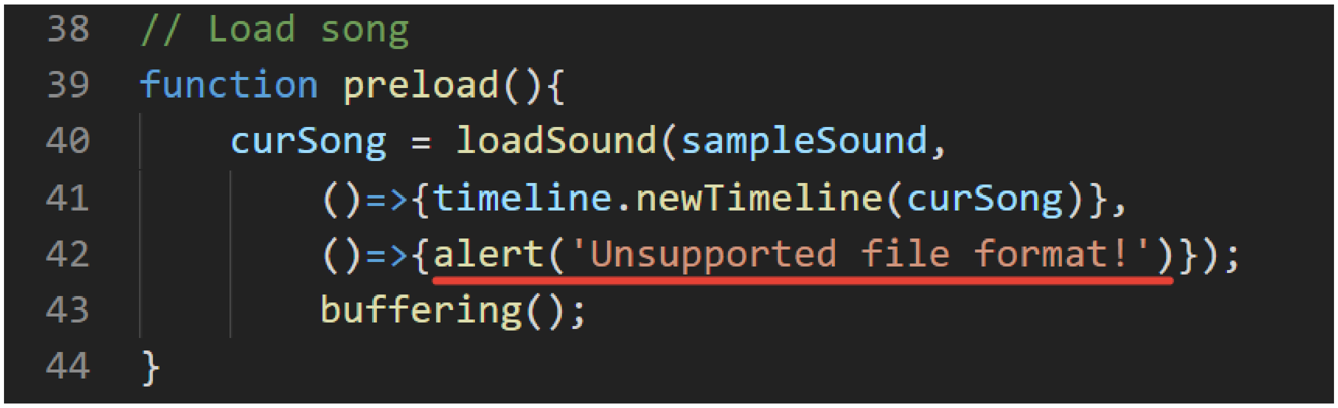
The next problem I wish to address is the input data and its validation. One of the features of this application is the data input functionality - the user can add his own songs to play and visualize. But there is no mechanism to validate this input besides the built-in format verification.



```
// Function to allow user input for the songs
function loadFiles(source, playlistArr){
    let fileList = source.files;

    // Allow multiple files
    for (let i = 0; i < fileList.length; i++){

        //Add only media files
        if (fileList[i].type.includes('audio')){
            playlistArr.push(fileList[i]);
        }
    }
}
```

These drawbacks create many possibilities for the potential breacher to cause security problems. One way to validate the input is to implement the name check using regular expressions. Using the RegEx we can filter input to prevent the entry of special symbols, and whitelist only letters and numbers.

The last problem that I tackle is the program output: the player uses the display to show some information and uses the built-in function to show error messages.



```javascript
// Load song
function preload(){
    curSong = loadSound(sampleSound,
        ()=>{timeline.newTimeline(curSong)},
        ()=>{alert('Unsupported file format!')});
        buffering();
}
```

To better adapt recommendations for secure output, I propose to format user messages for the user and incorporate a mechanism to generate more extensive error logs for the developer. The user messages should be short, clear, informative, and has the same string formatting. Logs should store all application settings to help identify the root of the problem and keep his privacy intact.