**Introduction**

Virtual Piano is web based piano instrument can be play using mouse click or keyboard keys on any Web Browser.

Piano can be implemented on web pages by using front end development techniques. Unlike real world piano virtual piano can be modified according to our requirement. Virtual is something not real but exist in digital environment.

CSS (Cascading Style Sheet) is used with HTML to create view of piano. JavaScript is used to apply methods to implement piano behaviour. This virtual piano is available for everyone publicly.

HTML CSS and JS are used to make a virtual piano. This Virtual piano can produce digital and high-quality audio. Because audio file is not used in this project, instead frequency is used to make various tones. So the quality of audio is crystal clear and fully customizable.

Web Audio API is used in this project to generate audio. The Web Audio API involves handling audio operations inside an **audio context**, and has been designed to allow **modular routing**. Basic audio operations are performed with **audio nodes**, which are linked together to form an **audio routing graph**. Several sources — with different types of channel layout — are supported even within a single context. This modular design provides the flexibility to create complex audio functions with dynamic effects.

Anyone can access this virtual piano using bottheem.github.io/virtualpiano.com

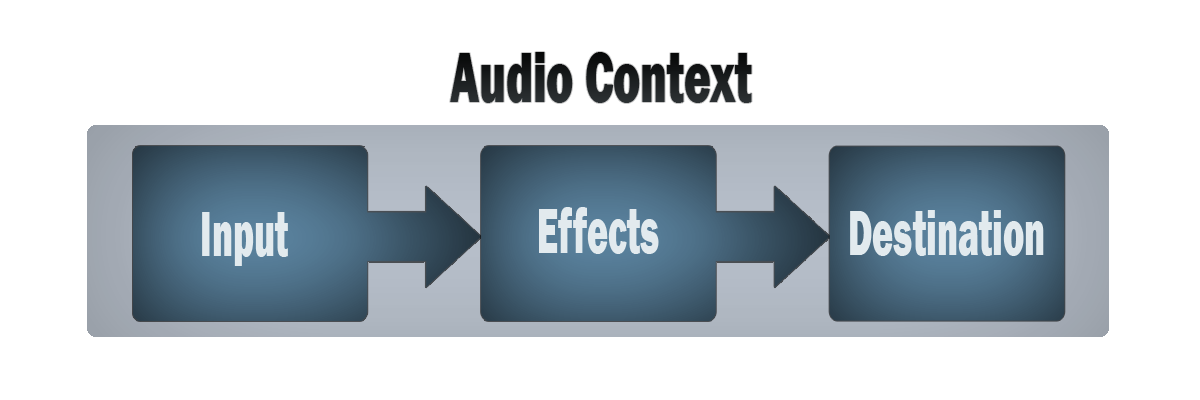
**Audio Web API**

The Web Audio API provides a powerful and versatile system for controlling audio on the Web, allowing developers to choose audio sources, add effects to audio, create audio visualizations, apply spatial effects (such as panning) and much more

The Web Audio API involves handling audio operations inside an audio context, and has been designed to allow modular routing. Basic audio operations are performed with audio nodes, which are linked together to form an audio routing graph. Several sources — with different types of channel layout — are supported even within a single context. This modular design provides the flexibility to create complex audio functions with dynamic effects.

Audio nodes are linked into chains and simple webs by their inputs and outputs. They typically start with one or more sources. Sources provide arrays of sound intensities (samples) at very small timeslices, often tens of thousands of them per second. These could be either computed mathematically (such as OscillatorNode), or they can be recordings from sound/video files (like AudioBufferSourceNode and MediaElementAudioSourceNode) and audio streams (MediaStreamAudioSourceNode). In fact, sound files are just recordings of sound intensities themselves, which come in from microphones or electric instruments, and get mixed down into a single, complicated wave.

Outputs of these nodes could be linked to inputs of others, which mix or modify these streams of sound samples into different streams. A common modification is multiplying the samples by a value to make them louder or quieter (as is the case with GainNode). Once the sound has been sufficiently processed for the intended effect, it can be linked to the input of a destination (AudioContext.destination), which sends the sound to the speakers or headphones. This last connection is only necessary if the user is supposed to hear the audio.



**PIANO – HTML**

There is one div tag as container for piano keys. Unordered list UL tag is used as piano keys. Each li tag helps to make keys both black and white.There are total 15 keys

<div class="container" id="main">

<ul class="keyList" id="zero">

<li class="keys white c" id="C5" onmousedown="playIt(event)"> c </li>

<li class="keys white a" id="A6" onmousedown="playIt(event)"> A </li>

<li class="keys black as" id="A#6" onmousedown="playIt(event)"> A# </li>

</ul>

</div>

**PIANO – CSS**

CSS (Cascading Style Sheet) is used to apply layout and colour on html elements to create the realistic view of real-world piano. CSS is also helps to create shadows when keys is pressed.

.white {

height: 16em;

width: 4em;

border-left: 1px solid #bbb;

border-bottom: 1px solid #bbb;

border-radius: 0 0 5px 5px;

background-color: white;

box-shadow: 14px 0px 18px 2px grey;

}

**PIANO-JS**

Javscript used to handle the events on the page. When key is pressed javscript executes the code, which can generate sound on speaker. Althougyh Tone.js, a javascript library is used for ease.

<script>

synth = new Tone.Synth().toMaster();;

function playIt() {

synth.triggerAttackRelease(event.target.id, "8n");

}

</script>

**Tone.js**

Tone.js is a framework for creating interactive music in the browser. It provides advanced scheduling capabilities, synths and effects, and intuitive musical abstractions built on top of the Web Audio API.

//create a synth and connect it to the master output (your speakers)

var synth = new Tone.Synth().toMaster()

//play a middle 'C' for the duration of an 8th note

synth.triggerAttackRelease('C4', '8n')

Tone.js makes working with the Web Audio API much easier because it takes care of a lot of tedious tasks for you. If you’ve ever worked with the Web Audio API you may have developed some strong opinions about it, but one thing you are likely to agree with is that it’s a bit tricky to do anything meaningful at first, apart from having a couple of oscillators generate a tone.

Tone.js has an extensive API—rather than explaining them all in detail here, I encourage you to check out the docs as well as some of the demos on the Tone.js site.

**Installation of Tone.js**

* Download tone.js file OR
* npm - npm install tone OR
* dev - npm install tone@next OR
* CDN - <https://cdnjs.cloudflare.com/ajax/libs/tone/13.0.1/Tone.min.js>

Tone.js file is used in this project. Tone.js file is placed in same folder where web page file exists. But this is not enough, web page file must call tone.js inside the page.

<script src=”/tone.js”>

</script>