GSoC 2024 Project Proposal for

21.1 HarmonyHub : A Web-Based Platform for Learning Variable-Pitch Musical Instruments.

(Duration : 350 Hours)

Lead Mentor: Alberto Acquilino (alberto.acquilino@mail.mcgill.ca)
Backup mentor: Mirko D'Andrea (mirko.dandrea@gmail.com)

KEERTHI REDDY KAMBHAM

INDIAN INSTITUTE OF TECHNOLOGY (BHU), VARANASI, INDIA

📞 +91 9848024503
📧 reddykkeerthi@gmail.com
keerthi-reddy-b7a1b9269
reddykkeerthi

TABLE OF CONTENTS

1. CONTACT DETAILS

Full Name : Keerthi Reddy Kambham
Email : reddykkeerthi@gmail.com
Neurostars ID : Keerthi_Reddy_Kambha
University : Indian Institute of Technology (BHU), Varanasi
Location : Varanasi, India
Time zone : Indian Standard Time (UTC +5:30)

2. PROJECT DETAILS

2.1 Project Title

Development of a Web-Based Personalized Platform for Learning Variable-Pitch Musical Instruments.

2.2 Project Synopsis

This project aims to build HarmonyHub, a web platform for personalized and engaging learning experiences with variable-pitch instruments like wind (e.g., Trumpet, Flute, Clarinet) and bowed string instruments (e.g., Violin, Viola). It offers customized exercises tailored by teachers based on age, skill level, and progression speed, with real-time feedback for improvement. HarmonyHub seeks to bridge the gap between traditional music education and modern technological capabilities. Most wind and string instruments fall into the category of being among the most difficult musical instruments to master.

Consequently, they demand deliberate practice strategies and support for students undertaking the learning process. This website tries to bridge the gap and assists students in achieving mastery in these instruments. Despite promising Information and Communication Technologies (ICT) in other educational contexts, they are underutilized in musical education. Digitalising music education is crucial in today's tech-driven era, as ICT can effectively enhance skills, creativity, and motivation in student-centered teaching. ICT facilitates focus on self-expression, self-control, reflection, ability to process feedback, overall creativity and musical composition, and even performance on certain instruments. These digitalization benefits make the learning journey more intuitive and rewarding.

2.3 Project in detail

2.3.1 Choice of Technologies and the Reasons Behind it

The frontend (client-side), encompasses all aspects visible and interacted with by users. The “backend” (server-side) acts as the website’s brain and manages databases supporting its architecture. Backend makes a website dynamic and engaging as it is used for server-side communication, API creation, and database management. Frontend developers use HTML for content structuring, CSS for styling, & JavaScript for interactivity and use frameworks like Angular, React.js or Vue.js etc.. Backend developers utilize languages like Python, Ruby, PHP, and frameworks like Express, Django, and Laravel.

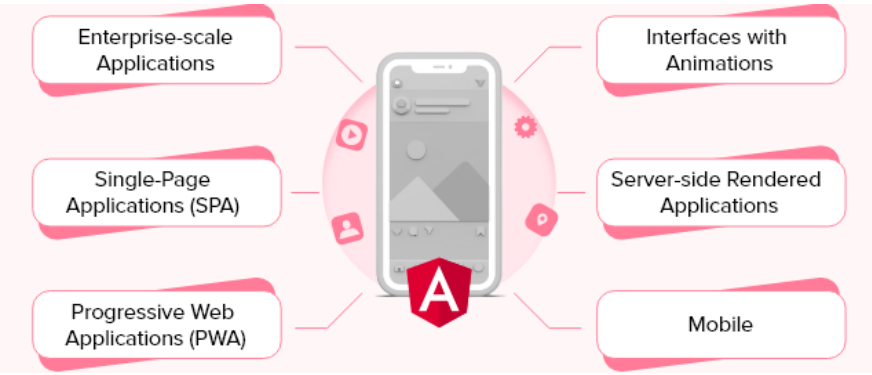
In essence, frontend development focuses on enhancing user experience with interactive interfaces, while backend development ensures reliable performance and seamless data management. Both are essential components in creating robust web applications or websites.

For the creation of this website, I propose the technologies : HTML, CSS and Angular framework for the frontend, Django framework for the backend and PostgreSQL database.

Angular framework is ideal as -

1. It utilizes HTML, CSS, and TypeScript. **TypeScript**, a syntactical superset of JavaScript enhances **type safety** and minimizes the likelihood of bugs.
2. It has flexibility and **cross-platform applications** (web, mobile, desktop or mobile first approach). As we intend to develop software that should run both on mobile and web, Angular is the perfect choice.
3. We offer features with only slight modifications across a wide range of instruments. The design and experience we provide for, let's say, the flute will closely resemble what we offer for, say, the trumpet. Angular's architecture, which is **Component-based**, enables us to create reusable components which simplifies unit testing, improves readability, and enhances ease of maintenance.
4. As this project is a **Single-Page Application**, routing procedures that Angular enables and its easy data management makes it the best choice.
5. It is a **full-fledged framework** and has modules to create attractive UI/UX, complemented by **Material Design's** sleek visual elements to create captivating animations around the instruments, making the website aesthetically pleasing.
6. **Angular Progressive Web Applications** (PWAs) offer a solution for students who may be offline or experience interruptions. They allow caching, proxies to intercept networks and conserve bandwidth.
7. It has a **MVC (Model-View-Controller)** software architectural setup. The components can be developed in parallel and separately tested.
8. **Two-way data binding technique** ensures that changes in the Model reflect in the View and vice versa reducing the development time.
9. Has Google's backing, providing reliable **support and documentation**.

10. Despite being considered a drawback, it can be optimized for **SEO**, especially for this website, which operates as SPA.



Django framework is ideal as -

1. It has **high-quality documentation** and is simple to use & understand.
2. Follows the **"batteries included"** approach with over 4,000 packages of additional features that cover debugging, profiling, and testing.
3. Packs in multiple features and is **time effective** for developing MVPs and prototypes. It has a library of reusable apps features and templates.
4. Suitable for projects of any size, **cross-platform** compatibility.
5. Based on the **DRY** (Don't Repeat Yourself) principle and **KISS** (Keep It Short and Simple) principle.
6. Always maintained the highest standards and is **regularly updated** with security patches. Also ensures compatibility with previous versions.

PostgreSQL is ideal as -

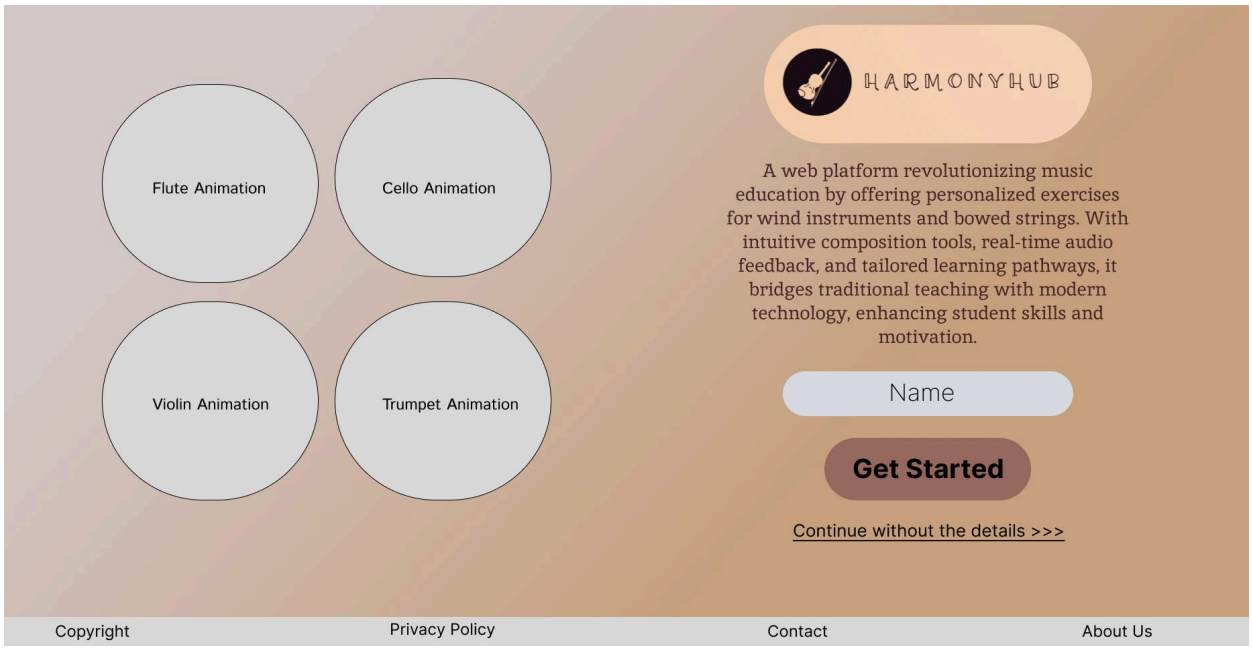
1. It is used where there is **structured data** and is vertically scalable which is exactly the case of this website which stores in a vertical manner for the new users incoming with fixed horizontal scale.
2. It is **compatible** with Angular and Django with high **Data Consistency and Integrity** along with being **highly scalable.**

The initial step post technology selection is to think of a design. Below are rough wireframes/concept sketches outlining the website's features and flow. These are not final and are subject to change based on existing progress.

2.3.2 Concept sketches/ Rough Wireframes/ Brief description of the website and its features (UI & UX)

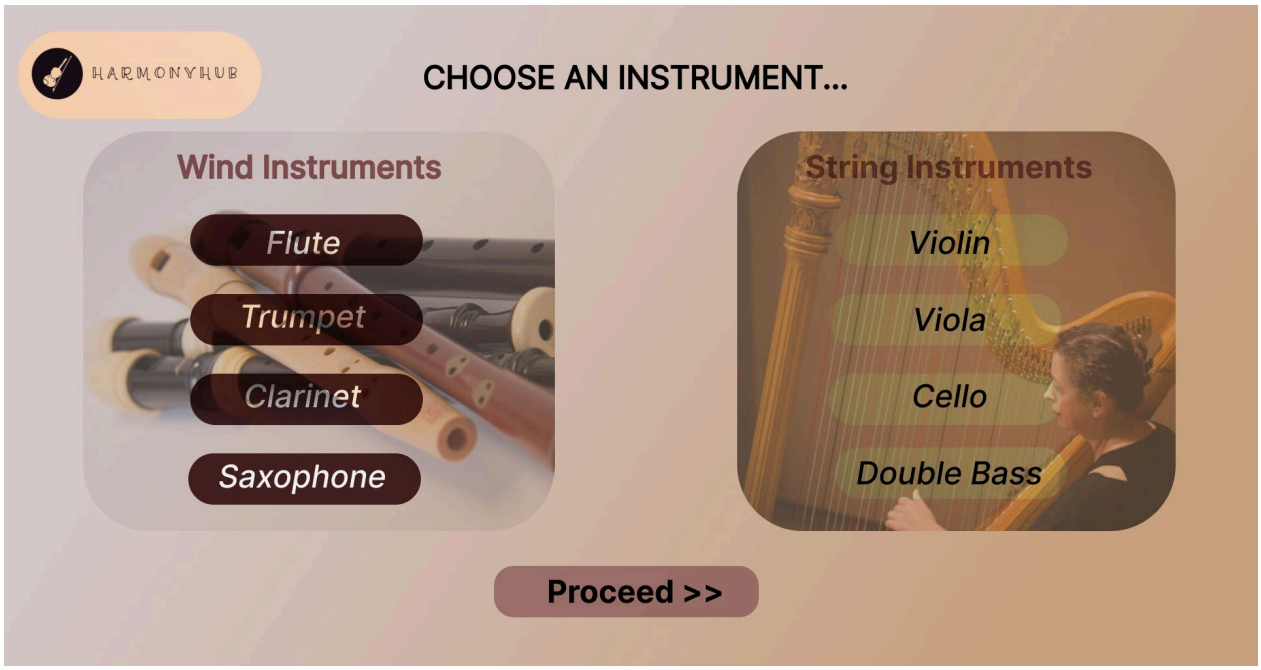
1. **THE LANDING PAGE -**

It will feature animated representations of the instruments addressed by the website (like Flute, Cello, Violin, Trumpet, etc.) along with the logo and a brief description highlighting the website's features and purpose. Name of the user is collected to display personalized messages like 'Hello, `{Name}`' or 'Welcome back, `{Name}`' or even while giving the real-time feedback displaying 'Good job, `{Name}`'. Sign up option has been deprecated to avoid the resistance and to streamline the user experience. Users can also proceed without entering their name.



2. **PAGE TO CHOOSE THE INSTRUMENT FROM -**

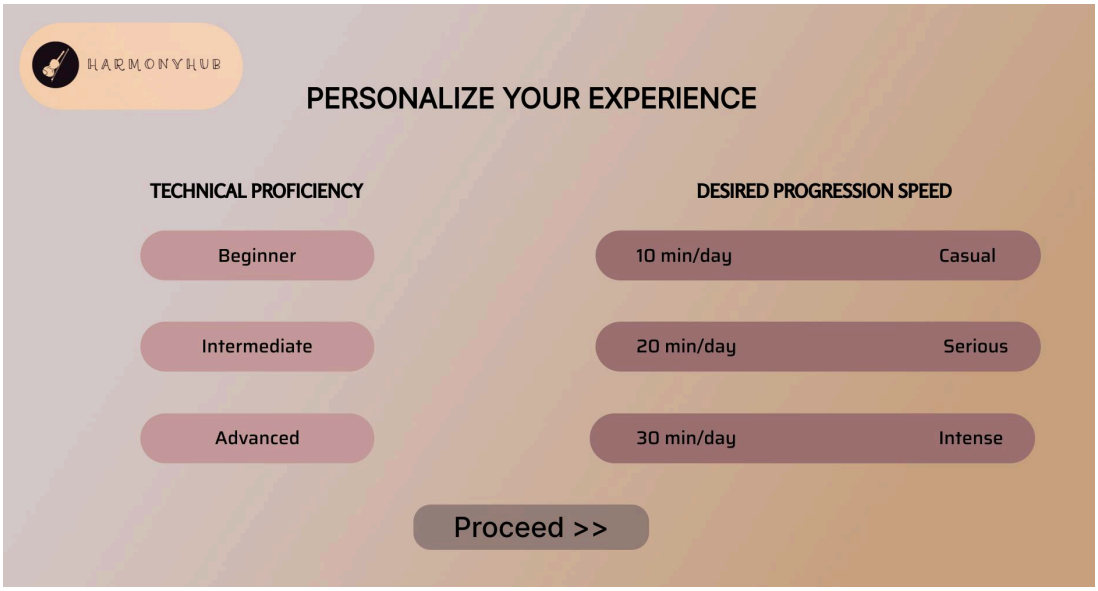
Users will then be directed to a page to choose an instrument they want to learn. Wind instruments such as the flute, trumpet and clarinet, as well as bowed string instruments like the violin, viola and cello, will be available for learning at this stage. After choosing the appropriate instrument, 'Proceed>>' button will activate. On clicking on this, a dialog box will be generated asking the user whether they'd like to personalize their learning. If the user chooses to opt for a YES!, then they will be prompted to the personalisation page. Else, they will directly be redirected to the 'playing instrument' page.



3. **PERSONALISATION PAGE -**

Depending on the technical proficiency chosen, I would provide different functionalities to master different rhythms with right control intonation, loudness and implement different quizzes for different technical levels. Users should also input their desired progression speed. To monitor this, the website will calculate the start time and end time using `performance.now()` function

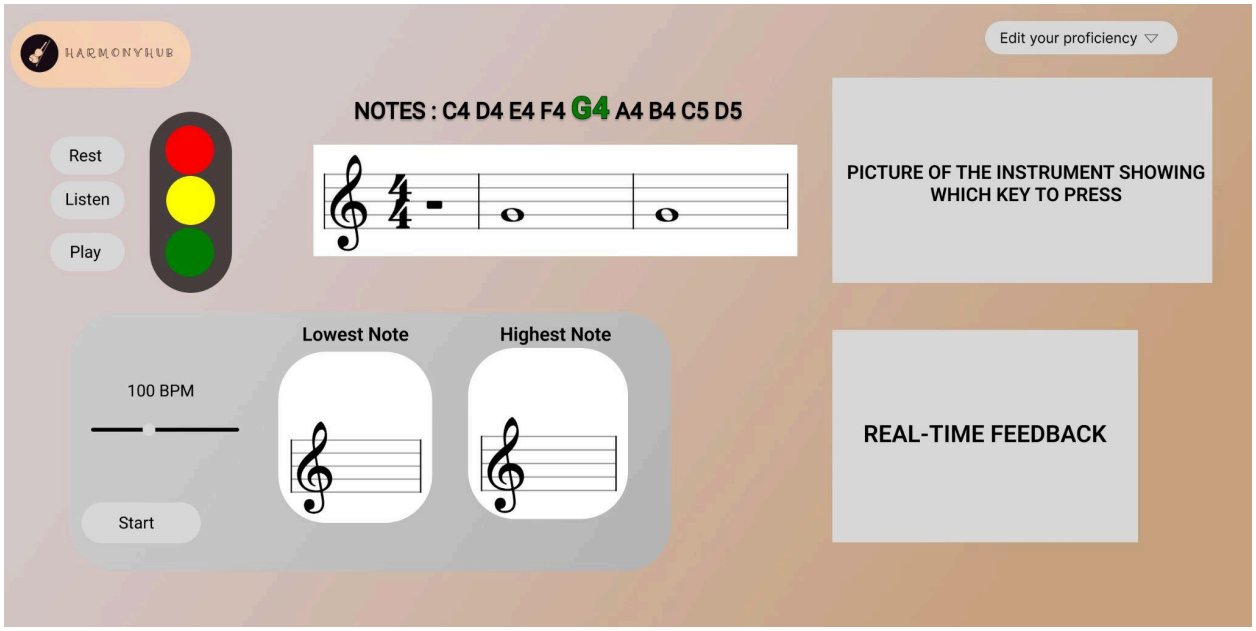
of JavaScript whenever `essentiaStart()` and `essentiaStop()` (defined in later sections) are called and subtract them to get the time spent. We do this for the entire time a student plays the instrument, add the values and store it in a variable locally (as users may not have a stable internet connection) using the `localStorage.setItem()` function. We can retrieve this data when a user gets connected to the internet (and doesn't clear cache) using the JS function, `localStorage.getItem()` and update it to the database. We also send a message when the daily goal is reached. The `time_spent` attribute in the database resets every 24 hours from the time it was last updated. After selection, 'Proceed>>' button will activate, prompting the user to the next page.



4. PLAYING INSTRUMENT PAGE -

The final landing page of the website features an input section to input appropriate parameters. The sound is then played along with displaying it on the musical stave. Upon starting, the first set of four beats signifies 'REST' (or a red light in a traffic symbol), prompting the student to inhale, relax, and prepare to play. The next four beats indicate 'LISTEN' (or a yellow light in a traffic

signal), during which the student should attentively listen to the sound being played. The final four beats signal 'PLAY' (a green light in a traffic signal), indicating the student should play the instrument. On the right-hand side, a picture of the musical instrument shows the keys to press on the instrument. Real-time feedback, facilitated by Essentia, is integrated, providing the audio analysis as the student plays. It can for eg., display the desired pitch, current pitch, and the difference, allowing for real-time correction.



(For a beginner level user)

2.3.3 Backend, Database, Django REST Framework (API) Setup

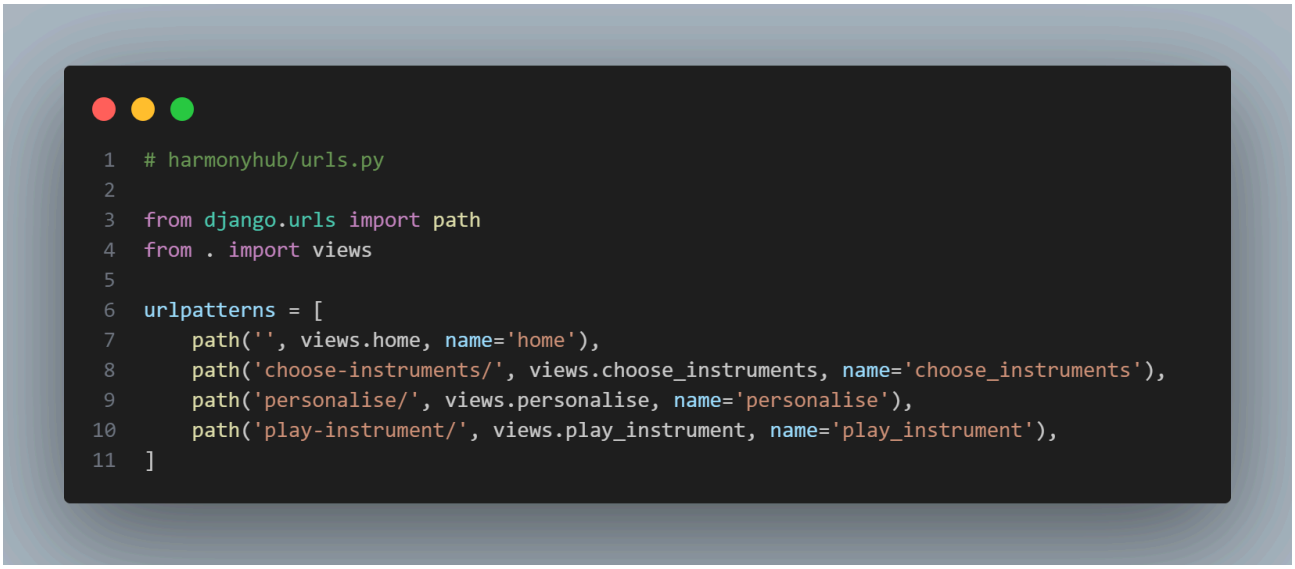
To setup the backend, we use **`django-admin startproject harmonyhub`** and the command **`python manage.py startapp harmonyhub`** (on windows). Next, we register the application and specify the database. As already mentioned we will be using the PostgreSQL database for this project.

```
DATABASES = {
```

```
  "default": {
      "ENGINE": "django.db.backends.postgresql",
      "NAME": "mydatabase",
      "USER": "mydatabaseuser",
      "PASSWORD": "mypassword",
      "HOST": "127.0.0.1",
      "PORT": "5432",
  }
}
```

URL MAPPING -

Now we need to map URLs in urls.py file using the `urlpatterns` list of path() functions. Each path() function either associates a URL pattern to a specific view. For this website, the urlpatterns variable is as follows -



```python
# harmonyhub/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('choose-instruments/', views.choose_instruments, name='choose_instruments'),
    path('personalise/', views.personalise, name='personalise'),
    path('play-instrument/', views.play_instrument, name='play_instrument'),
]
```

CREATING AND DESIGNING MODELS -

Models define the *structure* of stored data, including the field *types* and also their maximum size, default values etc.. But we should decide the data first. Upon a user's initial visit, a unique identifier is assigned and a cookie containing this is set in their browser. Depending on the user's preference on cookies, it may either persist or be deleted at the end of the session. Identifiers can be used to recognize the user on their revisit. The primary attribute in the database

is the `uniqueID`. The user's name is stored if provided; otherwise, a random value is assigned. The instrument they choose to play is recorded as `'instrument_name'`. If users opt to personalize their learning experience, their `'technical_proficiency'` data is stored; otherwise, it defaults to 'Beginner'. Similarly, the attribute `'desired_speed'` is recorded if provided by the user, otherwise set to NULL. Editable attributes are `name, instrument_name, technical_proficiency, and desired_speed`

```python
from django.db import models
import uuid

class UserSession(models.Model):
  unique_id = models.UUIDField(primary_key=True, default=uuid.uuid4, editable=False)
  name = models.CharField(max_length=255, blank=True, null=True)
  instrument_name = models.CharField(max_length=20)
  technical_proficiency = models.CharField(max_length=20, default="Beginner")
  desired_speed = models.IntegerField(blank=True, null=True)


  def __str__(self):
    return f"Session ID: {self.unique_id} - Name: {self.name}"
```

After creating, we can render data. A view function processes an HTTP request, fetches the required data, renders it in a HTML page, and returns this in a HTTP response. Below is the code to do this simple save and fetch -

```python
from django.shortcuts import render, redirect
from .models import UserInfo

def home_page(request):
    if request.method == 'POST':
        name = request.POST.get('name')
        #unique_id = Code to generate a random unique ID to each session.

        info = UserInfo.objects.create(name=name, unique_id=unique_id)
        return redirect('choose-instruments', unique_id=info.unique_id)
    else:
        return render(request, 'home.html')

def choose_instruments_page(request, unique_id):
    info = UserInfo.objects.get(unique_id=unique_id)
    context = {'name': info.name}
    return render(request, 'choose-instruments.html', context)
```

Django REST Framework - We install and update in INSTALLED APPS to include ‘rest_framework’ in it. We make /api as our endpoint (routing address) with the extension of <id> as a primary key. The object would be at api/<id>.

```python
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('harmonyhub.urls')),
]
```

Serializer translates the data into a JSON format that can be passed around the internet and used by other parties, such as frontend in this case.

```python
from rest_framework import serializers
from .models import HarmonyHub

class HarmonyHubSerializer(serializers.ModelSerializer):
    class Meta:
        fields = (
            'unique_id',
            'name',
            'instrument_name',
            'technical_proficiency',
            'desired_speed'
        )
        model = HarmonyHub
```

Finally, The API is ready to be passed around to the front end.

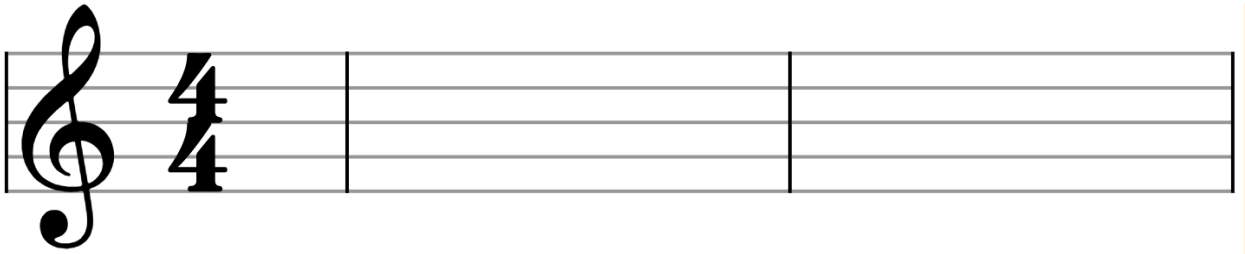
2.3.4 Detailed Explanation and Working Demo Link for the Critical ‘Playing Instrument’ Section.

The page where students learn to play the instrument is the most crucial and challenging aspect. This section showcases its logic and implementation.

2.3.4.1 Using VexFlow API to Render Musical Score and Notes.

VexFlow is an open-source online music notation rendering API. It is written completely in JavaScript, and runs right in the browser without the use of any server. VexFlow supports HTML5 Canvas and SVG. To add Vexflow API, we run `npm install vexflow` or use a script tag, `<script src="https://cdn.jsdelivr.net/npm/vexflow@4.2.2/build/cjs/vexflow.js"></script>`

a. **Rendering Stave -**



First, we create three empty staves. To achieve this, we render the SVG of the stave and get its context from VexFlow. We then assign this context to the variables we define and draw the stave. The first stave is positioned at x position : 3 units and has a width of 100 units. The second stave should start at x position : 103 units (3 units + 100 units) with a width of 130 units. Hence, the third stave should start at x position : 233 units (3 units + 100 units + 130 units) with a width of 130 units.

Code -

```
function drawEmptyStave() {
    VF = Vex.Flow;
    //get this atve at div with id = emptyStave in HTML
    var div = document.getElementById("emptyStave");
    //rendering the stave using SVG
    var renderer = new VF.Renderer(div, VF.Renderer.Backends.SVG);
    renderer.resize(550, 180); //setting size of entire context
    var context = renderer.getContext();

    var emptyStave1 = new VF.Stave(3, -25, 100);
    var emptyStave2 = new VF.Stave(103, -25, 130);
    var emptyStave3 = new VF.Stave(233, -25, 130);
    //adding treble sign and time signature to the first stave and
    //leaving the rest two empty
    emptyStave1.addClef("treble").addTimeSignature("4/4");
    emptyStave1.setContext(context).draw(); //setting the context and drawing
    emptyStave2.setContext(context).draw();
    emptyStave3.setContext(context).draw();
    return { emptyStave1, emptyStave2, emptyStave3, context };
}
const { emptyStave1, emptyStave2, emptyStave3, context } = drawEmptyStave();
context.scale(1.5, 1.5);
```

b. Rendering notes after inputting lowest and highest notes -

Users should be able to input the notes they want to play. For this, an empty stave should be rendered. We aim for the website to display corresponding notes (in gray with reduced opacity) when the user hovers/ drags over the stave. Upon clicking on the stave, we fix the selected note (displayed in black with full opacity) and return the chosen note.

To implement this, we convert browser coordinates to SVG coordinates and handle this situation in terms of x and y positions within this coordinate system. Initially create a class, `SVGInteraction` to add event listeners for the following `events =['touchStart', 'touchEnd', 'hover', 'drag', and 'mouseOut']`. This is designed to enable click & mouse interactions on SVGs.

```
const isDragSymbol = Symbol('dragging');
const isOutSymbol = Symbol('isOut');
const windowListeners = Symbol('windowListeners');
const listeners = Symbol('listener');

class SVGInteraction {
  constructor(svg, svgPt) {
    if (svgPt) this.svgPt = svgPt;
    this.svg = svg;
    this.makeInteractive();
  }

  addEventListener(type, callback) {
    this[listeners].push([type, callback]);
  }

  removeEventListener(type, callback) {
    const index = this[listeners].findIndex(([listenerType, listenerCallback]) => {
      return type === listenerType && callback === listenerCallback;
    });
    if (index !== -1) this[listeners].splice(index, 1);
    return index !== -1;
  }

  callListeners(type, e, coords) {
    console.log('calling listeners ' + type);
    this[listeners].forEach(([listenerType, callback]) => {
      console.log('checking listener ', listenerType);
      if (type === listenerType) {
        console.log('calling!');
        callback(e, coords);
      }
    });
  }

  /* eslint-disable no-unused-vars */
  // These are here as holders -- override whichever you need when you inherit this class.
  touchStart(e, coords) { this.callListeners('touchStart', e, coords); }
  touchEnd(e, coords) { this.callListeners('touchEnd', e, coords); }
  drag(e, coords) { this.callListeners('drag', e, coords); }
  hover(e, coords) { this.callListeners('hover', e, coords); }
  mouseOut(e, coords) { this.callListeners('mouseOut', e, coords); }
  /* eslint-enable no-unused-vars */
```

A bounding box should be defined within which these events will be made to be listened to.

```
makeInteractive(svg = this.svg) {
    console.log('making interactive');
    this[listeners] = [];
    // We will add listeners to the SVG bounding box itself:
    svg.style.pointerEvents = 'bounding-box';
    // An SVG point is used to translate div space to SVG space.

    this.svgPt = this.svgPt || svg.createSVGPoint();
    // A  not-combinator
    const not = condition => () => !condition();
    // Get whether we're dragging
    const isDragging = () => this[isDragSymbol];
    // Set whether the mouse is down or up.
    const down = () => { this[isDragSymbol] = true; return true; };
    const up = () => { this[isDragSymbol] = false; return true; };

    // Get whether we're outside of the SVG bounds
    const isOutside = () => this[isOutSymbol];
    // Set whether we're in our out; if we move out while dragging we need window listeners briefly.
    const inside = () => {
      this[isOutSymbol] = false;
      this[windowListeners].forEach(([eventType, listener]) => {
        window.removeEventListener(eventType, listener, false);
      });
      return true;
    };
    const outside = () => {
      this[isOutSymbol] = true;
      this[windowListeners].forEach(([eventType, listener]) => {
        window.addEventListener(eventType, listener);
      });
      return true;
    };

    // We'll hold the window listeners here so we can add & remove them as needed.
    this[windowListeners] = [];

    // Utility function to check event conditions & fire class events if needed
    const addListener = ([eventType, callback, ifTrue, el]) => {
      el = el || svg;
      const listener = (evt) => {
        if (ifTrue()) {
          const coords = getCoords(evt, svg, this.svgPt, this);
          callback.call(this, evt, coords);
        }
      };

      if (el !== window) el.addEventListener(eventType, listener);
      else this[windowListeners].push([eventType, listener]);
    };
```

Later, I brainstormed all the possible combinations of events within this bounding box and defined them.

```
[
      /* events occuring within SVG */
      ['mousedown', this.touchStart, down], // Touch is started
      ['touchstart', this.touchStart, down], // Touch is started
      ['mouseup', this.touchEnd, up], // Touch ends or mouse up
      ['touchend', this.touchEnd, up], // Touch ends or mouse up
      ['touchmove', this.drag, isDragging], // Dragging
      ['mousemove', this.drag, isDragging], // Dragging
      ['mousemove', this.hover, not(isDragging)], // Hover
      ['mouseout', this.mouseOut, not(isDragging)], // Mouseout
      ['touchcancel', this.mouseOut, not(isDragging)], // Mouseout (touch interrupt)
      ['mouseout', () => { }, () => isDragging() && outside()], // goes out of bounds isDown & set
      ['touchcancel', () => { }, () => isDragging() && outside()], // goes out of bounds isDown & set
      ['touchmove', inside, isOutside], // comes back inside
      ['mousemove', inside, isOutside], // comes back inside
      ['mousedown', inside, isOutside], // comes back inside, this shouldn't happen, but just in case
      ['touchstart', inside, isOutside], // comes back inside, this shouldn't happen, but just in case
      /* out of bounds events */
      ['touchmove', this.drag, () => isOutside() && isDragging(), window], // if outside in window & dragging
      ['mousemove', this.drag, () => isOutside() && isDragging(), window], // if outside in window & dragging
      ['mouseup', this.touchEnd, () => isOutside() && isDragging() && up() && inside(), window], // if outside in window & dragging & released
      ['touchend', this.touchEnd, () => isOutside() && isDragging() && up() && inside(), window], // if outside in window & dragging & released
    ]
      .forEach(addListener);
  }
}

```

Now define a function `getCoords(e, svg, svgPt)` which returns an object containing `{x_coordinate, y_cordinate, [touches]}` after taking in e (mouse event), SVGPt (a SVG point) and svg (SVG's client bounding rectangle). The SVG of staves drawn above is passed to the `SVGInteraction` class defined above. This provides us with an option of interacting with the stave recording the events like `['touchStart', 'touchEnd', 'hover', 'drag', 'mouseOut']` events and style and size the viewBox.

```
function getCoords(e, svg, svgPt) {
  if ('changedTouches' in e) {
    const length = e.changedTouches.length;
    const touches = [];
    for (let i = 0; i < length; i++) {
      touches.push(getCoords(e.changedTouches.item(i), svg, svgPt));
    }
    return { x: touches[0].x, y: touches[0].y, touches };
  }

  svgPt.x = e.clientX;
  svgPt.y = e.clientY;
  const svgCoords = svgPt.matrixTransform(svg.getScreenCTM().inverse());
  return { x: svgCoords.x, y: svgCoords.y };
}
```

Code -

```
function drawMusic() {
  VF = Vex.Flow;
  var lowestNote = document.getElementById("lowestNote")
  var rendererL = new VF.Renderer(lowestNote, VF.Renderer.Backends.SVG);
  rendererL.resize(500, 500);
  var contextL = rendererL.getContext();
  var staveL = new VF.Stave(10, 40, 100);
  staveL.addClef("treble");
  staveL.setContext(contextL).draw();
  var highestNote = document.getElementById("highestNote")
  var rendererH = new VF.Renderer(highestNote, VF.Renderer.Backends.SVG);
  rendererH.resize(500, 500);
  var contextH = rendererH.getContext();
  var staveH = new VF.Stave(10, 40, 100);
  staveH.addClef("treble");
  staveH.setContext(contextH).draw();
  return { staveL, contextL, staveH, contextH }
}
const { staveL, contextL, staveH, contextH } = drawMusic();
contextL.resize(180, 240);
contextL.setViewBox(0, 0, 500, 200);
contextL.scale(1.5, 1.5);
const svgL = contextL.svg;
svgL.removeAttribute('width');
svgL.removeAttribute('height');
svgL.style.border = "black solid 1px";
const interactionL = new SVGInteraction(svgL);
contextH.resize(180, 240);
contextH.setViewBox(0, 0, 500, 200);
contextH.scale(1.5, 1.5);
const svgH = contextH.svg;
svgH.removeAttribute('width');
svgH.removeAttribute('height');
svgH.style.border = "black solid 1px";
const interactionH = new SVGInteraction(svgH);
```

L denotes the stave and context for the lowest note and H denotes the stave and the context for highest note.

Until this point, we have added interaction classes to the staves and can get the coordinates and the type of events happening inside the stave. The next job would be to assign and display a particular note for a particular set of coordinates and also style the notes differently for different events happening inside the view box. We can do this by defining simple if-else conditions and may look to optimize it further in future.

For the lowest note, we maintain an `eventBuffer` array containing the events and coordinates of interaction, along with a `continueUpdating` variable set to `true`, indicating that we should listen to events and update coordinates. We define the following boundaries: 40 <= x <= 106 (covering the area to the right of the treble to the end of the bounding box) and 44 <= y <= 148 (covering the area within vertical boundaries).

Next, we define the y coordinates for different notes starting from F3 note to E6 note. If the event type is `'touchStart', 'hover', or 'drag'`, we display the note in a different style without committing to it, continuously updating it. The previous note vanishes, and a new note is created at a new place in the SVG context by drawing a new rectangle over the previous rectangle view box.

However, if the event type is `'touchEnd'`, we set the `continueUpdating` variable to `'false'`, indicating that we should stop listening to events and commit to coordinates during `'touchEnd'`, returning the note clicked upon.

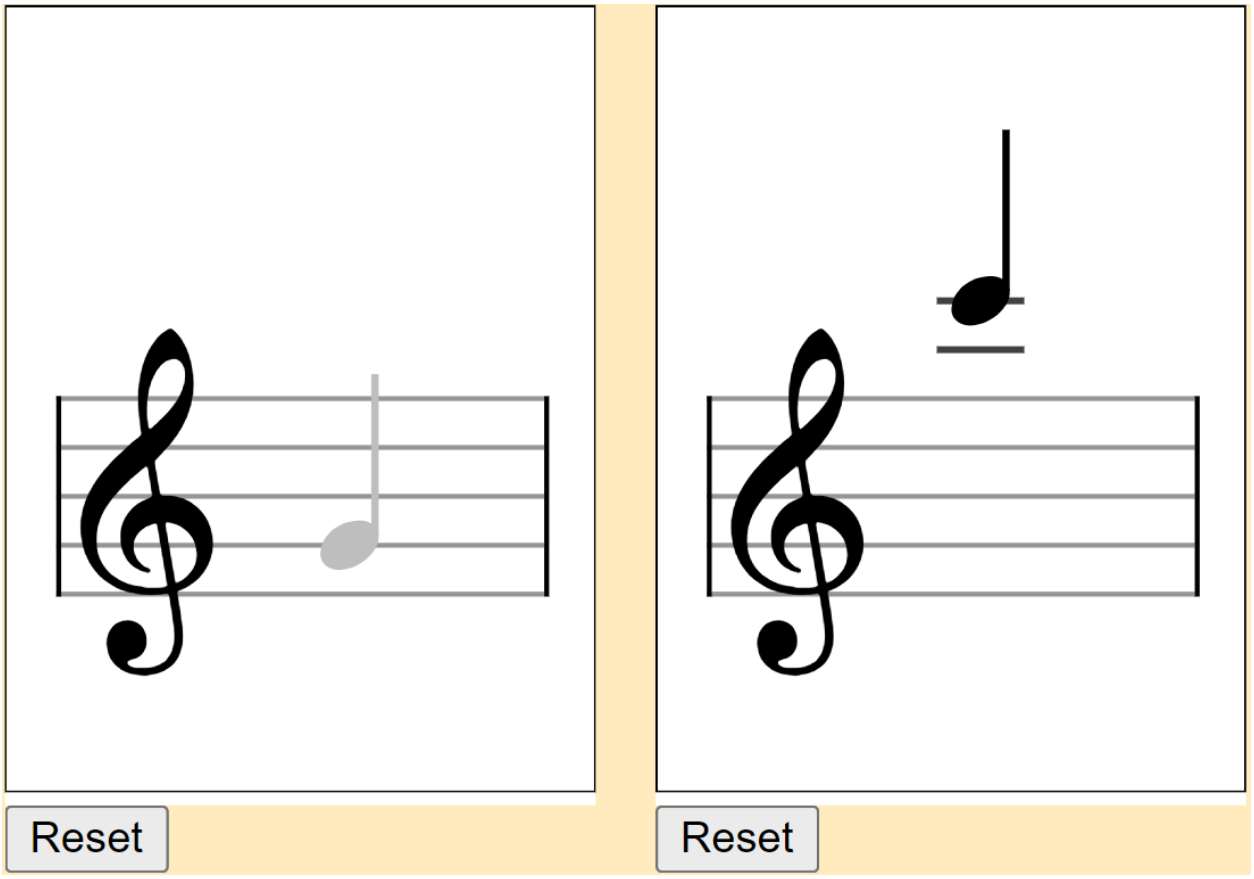
A reset button at the end will completely reset the view box area and set the `continueUpdating` variable to true again.

```
const eventsBuffer = [];
let continueUpdating = true;

function update(type, { x, y }) {
  if (!continueUpdating) return;
  x = Math.floor(x);
  y = Math.floor(y);
  var tempnotes = [];
  if (x >= 40 && x <= 106 && y >= 44 && y <= 148) {
    if (type == 'touchStart' || type == 'hover' || type == 'drag') {
      contextL.rect(0, 0, 500, 300, { stroke: 'none', fill: 'white' });
      staveL.setContext(contextL).draw();

      if (y <= 48 && y >= 44) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["e/6"], duration: "q" })];
      }

      else if (y >= 49 && y <= 53) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["d/6"], duration: "q" })];
      }
      else if (y <= 58 && y >= 54) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["c/6"], duration: "q" })];
      }
      else if (y >= 59 && y <= 63) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["b/5"], duration: "q" })];
      }
      else if (y <= 68 && y >= 64) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["a/5"], duration: "q" })];
      }
      else if (y >= 69 && y <= 73) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["g/5"], duration: "q" })];
      }
      else if (y <= 78 && y >= 74) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["f/5"], duration: "q" })];
      }
      else if (y >= 79 && y <= 83) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["e/5"], duration: "q" })];
      }
      else if (y <= 88 && y >= 84) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["d/5"], duration: "q" })];
      }
      else if (y >= 89 && y <= 93) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["c/5"], duration: "q" })];
      }
      else if (y <= 98 && y >= 94) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["b/4"], duration: "q" })];
      }
      else if (y >= 99 && y <= 103) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["a/4"], duration: "q" })];
      }
      else if (y <= 108 && y >= 104) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["g/4"], duration: "q" })];
      }
      else if (y >= 109 && y <= 113) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["f/4"], duration: "q" })];
      }
      else if (y <= 118 && y >= 114) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["e/4"], duration: "q" })];
      }
      else if (y >= 119 && y <= 123) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["d/4"], duration: "q" })];
      }
      else if (y <= 128 && y >= 124) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["c/4"], duration: "q" })];
      }
      else if (y >= 129 && y <= 133) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["b/3"], duration: "q" })];
      }
      else if (y <= 138 && y >= 134) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["a/3"], duration: "q" })];
      }
      else if (y >= 139 && y <= 143) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["g/3"], duration: "q" })];
      }
      else if (y <= 148 && y >= 144) {
        tempnotes = [new VF.StaveNote({ clef: "treble", keys: ["f/3"], duration: "q" })];
      }

      ar voices = [new VF.Voice({ num_beats: 1, beat_value: 4 }).addTickables(tempnotes)];
      ar formatter = new VF.Formatter().joinVoices(voices).format(voices, 400);
      empnotes[0].setStyle({ fillStyle: 'rgb(192,192,192)', strokeStyle: 'rgb(192,192,192)', opacity: 0.6 });
      empnotes[0].getTickContext().setX(x - 60);
      oices.forEach(function (v) { v.draw(contextL, staveL); })
```

```
else if (type == 'touchEnd') {
      var notes = [];
      continueUpdating = false;
      contextL.rect(0, 0, 500, 300, { stroke: 'none', fill: 'white' });
      staveL.setContext(contextL).draw();
      if (y <= 48 && y >= 44) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["e/6"], duration: "q" })];
      }

      else if (y >= 49 && y <= 53) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["d/6"], duration: "q" })];
      }
      else if (y <= 58 && y >= 54) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["c/6"], duration: "q" })];
      }
      else if (y >= 59 && y <= 63) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["b/5"], duration: "q" })];
      }
      else if (y <= 68 && y >= 64) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["a/5"], duration: "q" })];
      }
      else if (y >= 69 && y <= 73) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["g/5"], duration: "q" })];
      }
      else if (y <= 78 && y >= 74) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["f/5"], duration: "q" })];
      }
      else if (y >= 79 && y <= 83) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["e/5"], duration: "q" })];
      }
      else if (y <= 88 && y >= 84) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["d/5"], duration: "q" })];
      }
      else if (y >= 89 && y <= 93) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["c/5"], duration: "q" })];
      }
      else if (y <= 98 && y >= 94) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["b/4"], duration: "q" })];
      }
      else if (y >= 99 && y <= 103) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["a/4"], duration: "q" })];
      }
      else if (y <= 108 && y >= 104) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["g/4"], duration: "q" })];
      }
      else if (y >= 109 && y <= 113) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["f/4"], duration: "q" })];
      }
      else if (y <= 118 && y >= 114) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["e/4"], duration: "q" })];
      }
      else if (y >= 119 && y <= 123) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["d/4"], duration: "q" })];
      }
      else if (y <= 128 && y >= 124) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["c/4"], duration: "q" })];
      }
      else if (y >= 129 && y <= 133) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["b/3"], duration: "q" })];
      }
      else if (y <= 138 && y >= 134) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["a/3"], duration: "q" })];
      }
      else if (y >= 139 && y <= 143) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["g/3"], duration: "q" })];
      }
      else if (y <= 148 && y >= 144) {
        notes = [new VF.StaveNote({ clef: "treble", keys: ["f/3"], duration: "q" })];
      }
      var voices = [new VF.Voice({ num_beats: 1, beat_value: 4 }).addTickables(notes)];
      var formatter = new VF.Formatter().joinVoices(voices).format(voices, 400);
      notes[0].getTickContext().setX(x - 60);
      voices.forEach(function (v) { v.draw(contextL, staveL); })
      return notes[0].keys[0];
  }
```

```
const resetButton = document.getElementById('resetButton');
resetButton.addEventListener('click', reset);

// Function to reset the updating process
function reset() {
  contextL.rect(0, 0, 500, 300, { stroke: 'none', fill: 'white' });
  staveL.setContext(context).draw();
  continueUpdating = true; // Set continueUpdating flag to true to resume updating
}
```

The same code also applies for the highest note too. The output:



Left side has the lowest note when we ‘hover/drag’ the mouse over it and the right side shows the highest note when we click on it.

2.3.4.2 Generating Sound : From Inputted Lowest to Highest Note

After the user inputs the desired notes, the sound will be played sequentially from the lowest note to the highest note. For example, if the student selects to play sound from the lowest note, C5, to the highest note, A6, then the sequence C5, D5, E5, F5, G5, A6 will be played, with the corresponding notes highlighted on the stave. To implement this process, the inputted notes are stored in a string. This information, along with the BPM data (as discussed in a later section), is then passed to the MidiWriterJS API, which converts the data into a MIDI file (.mid). Finally, this MIDI data is passed to any API capable of reading and playing the notes, while also displaying the notes on the stave.
Detailed Implementation -

a. The `update()` function, as defined previously, returns a string upon triggering the `'touchEnd'` event. This returned value can be stored in `lowestNote` and `highestNote`. For other events, the return value is undefined. To ensure readability for the API, '/' character is removed. Both notes are then stored in an `inputNotes[]` array for later access. And the `playNotes()` function is called to play the `inputNotes[]`
b. To play the notes sequentially, we initialize an array called `noteSequence` with notes already sorted in ascending order of their pitch. Next, we find the indices of `lowestNote` and `highestNote` within this `noteSequence`. We then initialize another array to store all notes between these indices.
c. With the sequence of notes prepared, we proceed to pass it to MIDIWriterJs by specifying an appropriate instrument number to identify which instrument's notes to play, along with the BPM (from the metronome code) Instrument numbers can be referenced from here. MIDIWriterJs initializes a track, sets the tempo, adds events (notes to be played) into this track, and finally passes this track into a constant called `write`, which writes onto any sound generation API.

d. To generate sound from this MIDI data, I propose using the **MagentaJS** API, which offers both a MIDI player and a MIDI visualizer, perfectly meeting our requirements. The fast, efficient implementation of the Magenta API can be explored here. The `write` constant mentioned earlier operates on `<midi-player>` and `<midi-visualizer>`, enabling the playback of sound and visualization on a stave.

Code -

```
let inputNotes = [];
    const events = ["touchStart", "touchEnd", "drag", "hover", "mouseOut"];
    events.forEach((type) => {
      interactionL.addEventListener(type, (e, coords) => {
        var returnedNote = update(type, coords);
        if (returnedNote != undefined) {
          let lowestNote = returnedNote;
          lowestNote = lowestNote.replace("/", "");
          inputNotes[0] = lowestNote;
          if (inputNotes[1] != undefined) playNotes(inputNotes);
        }
      });
    });
    events.forEach((type) => {
      interactionH.addEventListener(type, (e, coords) => {
        var returnedNote1 = update1(type, coords);
        if (returnedNote1 != undefined) {
          let highestNote = returnedNote1;
          highestNote = highestNote.replace("/", "");
          inputNotes[1] = highestNote;
          if (inputNotes[0] != undefined) playNotes(inputNotes);
        }
      });
    });
```

L denotes the interaction for lowest note and H denotes the interaction for highest note

```
const noteSequence = ["f3", "g3", "a3", "b3", "c4", "d4",
      "e4", "f4", "g4", "a4", "b4", "c5",
      "d5", "e5", "f5", "g5", "a5", "b5",
      "c6", "d6", "e6"];
    function playNotes(inputNotes) {
      let temp = 0;
      const track = new midiWriterJs.Track();
      track.addEvent(new midiWriterJs.ProgramChangeEvent({ instrument: 73 }));
      track.setTempo(bpm, 0);
      let lowestNoteIndex = noteSequence.indexOf(inputNotes[0]);
      let highestNoteIndex = noteSequence.indexOf(inputNotes[1]);
      const notes = noteSequence.slice(lowestNoteIndex, highestNoteIndex + 1);
      for (let i = 0; i < notes.length; i++) {
        const n = notes[i];
        const e = new midiWriterJs.NoteEvent({ pitch: n, duration: "1" });
        track.addEvent(e);
      }
      let player = document.getElementById("player");
      let v_staff = document.getElementById("v_staff");
      let v_roll = document.getElementById("v_roll");
      const write = new midiWriterJs.Writer(track);
      player.src = write.dataUri();
      v_staff.src = write.dataUri();
      v_roll.src = write.dataUri();

      player.addVisualizer(v_staff);
      player.addVisualizer(v_roll);
    }
```

2.3.4.3 Generating Beats at a Specified Tempo

When playing a note, beats at the inputted BPM should accompany it. Following the traffic light mechanism for enhanced user experience, let's assume each note lasts for 4 beats. Consequently, beats are grouped in sets of 4. For a REST phase lasting 4 beats, a LISTEN phase for four beats, and a PLAY phase for 4 beats, they should also be grouped in sets of 12.
To implement this,

a. First, we need to require the beats audio and set the variables accordingly in the code. We define a `playClick()` function to keep track of the beat count, enabling the implementation of the traffic light mechanism and playing different audio at the start of each set of 4 beats. When the beat count is divisible by 4, we play a different audio. Additionally, when the count is less than or equal to 4, it's the 'REST' phase; when it's less than or equal to 8, it's the 'LISTEN' phase; and when it's less than or equal to 12, it's the 'PLAY' phase, represented respectively by the colors red, yellow, and green in traffic lights.
b. We add an event listener to the ‘start’ or ‘stop’ event, that is triggered when `<midi-player id = “player”> </midi-player>` activates or deactivates to call `playBeat()` function. This function begins playing the beats when the beats are not playing and stops when the beats are already playing. The scenario of changing the BPM value is also addressed in `handleBpmChange()`.

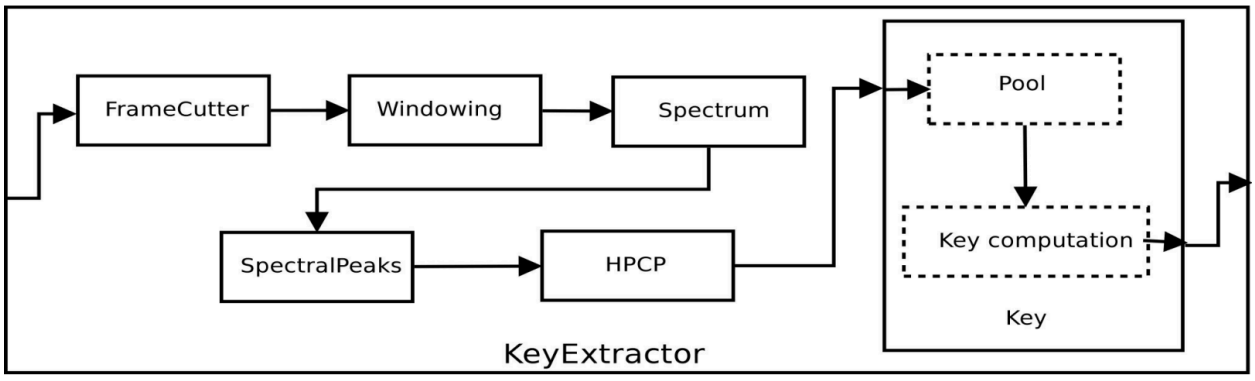
```
//Event Listeners
  document.getElementById('player').addEventListener('start', playBeat);
  document.getElementById('player').addEventListener('stop', playBeat);
  document.getElementById('bpmSlider').addEventListener('input', handleBpmChange);
```

```
const click1Url = 'click1.wav';
    const click2Url = 'click2.wav';
    // Create Audio objects with the URLs
    const click1 = new Audio(click1Url);
    const click2 = new Audio(click2Url);

    let playing = false;
    let count = 0;
    let bpm = 100;
    let beatsPerMeasure = 4;
    let timer;
    let countLight = 1;
    let beatsPerMeasureLight = 12;
    function playClick() {
      if (countLight <= 4) {
        rest();
      }
      else if (countLight <= 8) {
        listen();
      }
      else if (countLight <= 12) {
        playInstrument();
      }
      countLight = (countLight + 1) % beatsPerMeasureLight;
      if (count % beatsPerMeasure === 0) {
        click2.play();
      } else {
        click1.play();
      }
      count = (count + 1) % beatsPerMeasure;
    }
```

```
function playBeat() {
   if (playing) {
     clearInterval(timer);
     playing = false;
   } else {
     timer = setInterval(playClick, (60 / bpm) * 1000);
     playing = true;
     // Play a click "immediately"
     playClick();
   }
 }

 function handleBpmChange(event) {
   bpm = event.target.value;
   if (playing) {
     clearInterval(timer);
     timer = setInterval(playClick, (60 / bpm) * 1000);
     count = 0;
   }
   document.getElementById('bpmDisplay').innerText = bpm + ' BPM';
 }
 function rest() {
   clearLight('goLight');
   document.getElementById('stopLight').style.backgroundColor = "red";
 }
 function listen() {
   clearLight('stopLight');
   document.getElementById('slowLight').style.backgroundColor = "yellow";
 }
 function playInstrument() {
   clearLight('slowLight');
   document.getElementById('goLight').style.backgroundColor = "green";
 }
 function clearLight(lightId) {
   document.getElementById(lightId).style.backgroundColor = "black";
 }
```

2.3.4.4 Real-Time Feedback using EssentiaJs

During the 'PLAY' phase, students play the instrument while the EssentiaJs library API is activated. This library's core, powered by Essentia C++ library back-end using WebAssembly built via Emscripten along with a high-level JS and TypeScript API and utility modules, captures sound through the device's microphone and directly analyzes it on the client-side. It supports both real-time and offline audio analysis use cases, distinguishing itself from other APIs that predominantly analyze audio server-side. Additionally, it accommodates mobile devices, provides pedagogical feedback on student performance, and offers cross-platform support. Potential audio analysis features include loudness, timbre, tonality, pitch, melody, onsets, and rhythm. Furthermore, a combination of Essentia features can be employed to extract additional information, such as determining the key the student is playing in.



We use an algorithm that displays the frequency of sound in Hertz, PitchYin for the implementation below. We could also provide feedback on whether the user's pitch is consistently sharp, flat, or fluctuating.

Implementation -

Can use `npm install essentia.js` or a simple script tags `<script src = "https://cdn.jsdelivr.net/npm/essentia.js@0.1.1/dist/essentia-wasm.web.js"></script>`

`<script src = "https://cdn.jsdelivr.net/npm/essentia.js@0.1.1/dist/essentia.js-model.js"></script>`

Upon loading the window (`window.onload()`), Essentia is activated by initializing a new `AudioContext()`.

```
let essentia = null;
let micStream;
let AudioContext;
let audioCtx;
let mic;
let gain;
let essentiaProcessorNode;
let bufferSize = 8192;
window.onload = () => {
  AudioContext = window.AudioContext || window.webkitAudioContext;
  audioCtx = new AudioContext();
  EssentiaWASM().then(wasmModule => {
    essentia = new Essentia(wasmModule, false);
  })
};
```

During the 'PLAY' phase of the website, an eventListener is added, triggering the `essentiaStart()` function. Upon deactivation, the `essentiaStop()` function is called, halting the microphone stream and disconnecting all nodes. The `essentiaStart()` function requests microphone access and invokes `startAudioProcessing()`, which establishes a script processor and sets a buffer size, typically at the exponents of 2 (Here, I am using the thirteenth power of 2 (8192) to maintain stable pitch dynamics)

```
function essentiaStart() {
      requestMicAccess()
        .then(startAudioProcessing)
        .then(function onAudioStart() {
          document.getElementById("listening").innerHTML = "LISTENING";
        })
        .catch(err => {
          if (err.name == "NotAllowedError") {
            alert("Could not access microphone - Please allow microphone access for this site");
          } else {
            alert(err);
            console.log("Exception name: ", err.name);
            throw err;
          }
        });
    }
    async function requestMicAccess() {
      console.log("Initializing mic stream...");
      let micStream = await navigator.mediaDevices.getUserMedia({ audio: true, video: false });
      return micStream;
    }
    function essentiaStop() {
      micStream.getAudioTracks().forEach(function (track) {
        track.stop();
        micStream.removeTrack(track);
      });
      console.log("Stopped mic stream ...");
      audioCtx.close().then(function resetState() {
        document.getElementById("listening").innerHTML = "idle";
        mic.disconnect();
        essentiaProcessorNode.disconnect();
        gain.disconnect();
        mic = null;
        essentiaProcessorNode = null;
        gain = null;
      });
    }
```

Following audio processing, data is fed into the Essentia processor, offering various methods for tailored processing. For instance, `essentia.PitchYin(audioSignal).pitch` provides pitch information, while `essentia.PitchYin(audioSignal).pitchConfidence` offers the confidence level of the detected pitch, ensuring reliable data display with lesser error. Here, I am setting the confidence threshold at 50% (0.5).

```
function startAudioProcessing(stream) {
      micStream = stream;
      if (!micStream.active) {
        throw "Mic stream not active";
      }
      if (audioCtx.state == "closed") {
        audioCtx = new AudioContext();
      } else if (audioCtx.state == "suspended") {
        audioCtx.resume();
      }
      mic = audioCtx.createMediaStreamSource(micStream);
      essentiaProcessorNode = audioCtx.createScriptProcessor(bufferSize);
      essentiaProcessorNode.onaudioprocess = essentiaProcessor;
      gain = audioCtx.createGain();
      gain.gain.setValueAtTime(0, audioCtx.currentTime);
      mic.connect(essentiaProcessorNode);
      essentiaProcessorNode.connect(gain);
      gain.connect(audioCtx.destination);
    }

    function essentiaProcessor(audioProcessingEvent) {
      const inputBuffer = audioProcessingEvent.inputBuffer;
      const inputData = inputBuffer.getChannelData(0);
      const audioVector = essentia.arrayToVector(inputData);
      const pitchValue = essentia.PitchYin(audioVector).pitch;
      const pitchConfidence = essentia.PitchYin(audioVector).pitchConfidence;
      if (pitchConfidence >= 0.5) {
        displayResults({ pitch: pitchValue });
      }
    }
    function displayResults(features) {
      const display = document.querySelector("#activation-display");
      display.innerHTML = `<pre>${JSON.stringify(features, null, 4)}</pre>`;
    }
```

2.3.4.5 Rendering a 2-Dimensional Picture with Fingering Positions

To generate fingering patterns for different notes, the above used API from MagentaJs in the form of `<midi-player id = “player”>` has a property called note which gives the information about the current note playing, We get a `const player = document.getElementById(“player”)` and read the

property using `player.note`. This gives the information about the notes and we can match this information with the appropriate fingering patterns.

LINK TO A SIMPLE DEMO THAT INTEGRATES THE APIs & BRING OUT OUR DESIRED FUNCTIONALITY - HarmonyHub Demo For Flute

(Set the BPM, input the desired lowest and highest note and you have the functionality!)

2.4 Project Implementation and Timeline

2.4.1 Minimal set of Deliverables

During the GSoC'24 period, I will deliver these features and functionalities:

- I will design a user-friendly and attractive UI/UX with unique themes and functionalities to ensure a better user experience for all types of users.
- I will implement personalized features, such as adapting exercises based on their technical proficiency and will cater to their desired progression speed. Depending on their proficiency, I would provide different functionalities to master different rhythms with right control intonation, loudness and implement different quizzes for different technical levels.
- The website's flow, including the navigation from one page to another, as determined through discussions with mentors, will be implemented.
- To reduce user resistance, I will provide direct access to exercises without requiring extensive inputs. Additionally, I will implement cookies to save user history, streamlining future interactions.
- I will provide input mechanisms to allow students to input BPM/tempo and score, with the website producing sound from their input for a particular instrument. As I have already implemented the basic integrations and functionalities of APIs for score generation (VexFlow),

sound generation (MagentaJs) and real-time feedback mechanisms (EssentiaJs), I will optimize this implementation and explore for more sophisticated APIs and complex functionalities to implement.

- I will develop a Django backend and use PostgreSQL database to store user information, utilizing APIs to fetch necessary data from the backend.
- The website will cater to wind instruments such as the Flute, Clarinet, & Trumpet, and string instruments including the Violin, Viola, and Cello.
- Two-dimensional instrument images will be provided on the website to guide students on which keys to press at any given moment during play.
- Integration of Essentia to offer real-time feedback on their performance.
- For documentation and testing, I will write unit tests and E2E tests for features and functions, ensuring they work as intended in various scenarios. Compatibility and performance testing will also be conducted. Clear documentation will accompany the project, and best practices will be followed to write optimized code for smooth rendering.
- In the deployment phase, I will upload files to the production server, configure domain settings and DNS records, set up SSL certificates for secure connections (HTTPS), and perform final checks to ensure everything works correctly in the production environment.

2.4.2 Additional ‘if-time allows’ deliverables

- If time permits, I intend to expand the website to encompass additional wind and string instruments.
- If time allows, I aim to develop three-dimensional models of the instruments on the website when users play an instrument, enhancing students' understanding of fingering positions and orientations.
- Moreover, I will explore the possibility of integration of AI/ML models (tensorflow) from EssentiaJs to provide more detailed pedagogical audio feedback to the user.

- ❖ Furthermore, time permitting, I will also request for web-cam access as an addition to microphone access to judge the proper orientation and fingering positions while the user plays an instrument.

2.4.3 Detailed Timeline of Tasks Implementation

Timeline	Deliverables
	Community Bonding Period
May 1 - May 26	• Familiarize with all progress made on the project by this point. Discuss with mentors and other developers to determine the next steps and prioritize tasks. • Join communication channels to engage with team members, establish rapport, and review documentation and code for completed portions. Identify necessary resources for reference before commencing new feature development. • Discussions with mentors about their preferred communication medium for project updates.
	Coding Officially Begins!
May 27 - June 9	• Design the UI/UX of the website which includes logo, animations, coloring and other visible elements of the website, features and the flow of the website. • Setup the frontend and develop the homepage, page to choose instruments and personalisation page.
June 10 - June 23	• Develop the 'playing instrument' page for beginner, intermediate and advanced staged users of any two of the **string instruments** using the score generation API, sound generation API, metronome player providing and integrating essentia to provide real-time feedback.
June 24 -	• Develop the 'playing instrument' page for beginner, intermediate and advanced staged users of any two of the **wind instruments** using the

July 10	score generation API, sound generation API, metronome player providing and integrating essentia to provide real-time feedback. • Obtain feedback and improve accordingly.
Midterm Evaluation	
July 13 - July 28	• Setup Django backend and PostgreSQL database. • Code the logic of the website in urlpatterns, define and register the database schema. • Use Django REST Framework and create an API to integrate the backend with the frontend and get the website running.
July 29 - August 11	• Develop the 'playing instrument' page for beginner, intermediate and advanced staged users of all the remaining instruments using the score generation API, sound generation API, metronome player providing and integrating essentia to provide real-time feedback. • Do the testing and once all unit tests and E2E tests will pass, receive confirmation from mentors and do the documentation. • Deploy the website!
August 11 - September 3	BUFFER PERIOD
Initial Results will be announced	
September 4 - October 13	• If possible, include a 3D model showing the exact fingering position and orientation • If time permits, train or include AI/ML models from EssentiaJs to provide more sophisticated feedback.
Final Evaluation	

2.5 Plan for Communication with Mentors

Throughout the project, I will uphold regular communication with mentors and other expert developers working on the project to ensure alignment with project goals. I plan to schedule a fixed time every week to update progress from the previous week, discuss goals for the upcoming week, and seek assistance if necessary. Although I am flexible and can work during night time hours as well, typically, I will be available for calls or messages from 9 AM IST to 11 PM IST. I'll stay connected via emails and Zoom calls, and I suggest creating a Microsoft Team for smoother communication if feasible.

3. CANDIDATE DETAILS

3.1 A Short Summary of Past Experiences

I began with competitive programming, using C++. Later, I delved into software development, mastering HTML, CSS, and JavaScript, and built projects using the MERN stack. Currently, I'm exploring Angular and PostgreSQL to expand my skill set. I've also learned Python, explored the Django framework and delved into Machine Learning (Link), and am now diving into Deep Learning. I'm a member of the Club of Programmers (COPS) at IIT(BHU), a vibrant community where students come together to share knowledge and learn from each other. The club also organizes various hackathons and coding events to foster a culture of programming excellence. I joined in my first year and have been actively contributing ever since. Within COPS, I collaborated with a team of 20+ developers on the SDG Site project. We developed a high-performance website using Next.js and enhanced its performance further using Lighthouse. I played a key role in optimizing the site through caching, layout shift optimizations, promise optimizations, and async code optimizations. Additionally, I worked on the frontend of Hackalog, a web application

supporting hackathon and dev sprint hosting. This project provided valuable learning opportunities and practical experience, particularly in working with Django Rest Framework for the backend and React.js for the frontend. We also organized a successful dev sprint for new club members, launched in February. For the next release, we're planning to enhance the admin interface to make hosting such events more accessible to anyone.

3.2 Motivation to do This Project

I'm deeply passionate about music education. While researching this project, I delved into technical music details (sometimes, more than I needed to) out of sheer curiosity. Music transcends boundaries, uniting people across traditions and cultures. Every culture in the world has a dedicated section for music in its history. The idea of integrating something as pure and powerful as music with technology feels empowering. As a developer, I aspire to contribute to such a project, enhancing my reputation and motivation. Moreover, I couldn't continue learning the Harmonium when I moved to a new city during my childhood due to the lack of proper resources. Developing technology that prevents others from facing the same problem drives my motivation. This project can positively impact countless music students, uplifting their skills. Collaborating with expert developers who are already working on this, and sharing equal enthusiasm and motivation with mentors, is truly exciting as we develop the project further.

3.3 What makes me a good candidate for this project?

I have been working with JavaScript and Python ever since I was introduced to technology, for more than 3 years now, yet my most incredible skill is figuring things out along the way. This project caught my attention immediately when the project list was out and made me want to explore more as it combines both

of my favorite sectors: music and development. Since then, I have played around with integrating different instruments and building exercises based on them. Out of curiosity, I built a demo website featuring the flute. So, I have a strong idea about what it takes to contribute to this project, like the libraries, methods, integrations, etc. Although I am new to learning about musical instruments and used to not know anything, I quickly adapted from not knowing anything to having a good idea about instruments, musical notations, sound generation of notes, sound analysis, and used all these features myself when building the demo, getting hands-on working experience with all of those. I am an avid learner and can adapt to any new tech as fast as possible. My passion and never-give-up attitude when it comes to software development sets me apart from others. I also have great communication skills and can make anyone understand my ideas and approaches. Overall, I believe I can be a great resource to INCF and the open-source community in general. I am a responsible individual and a great team player and believe in the utmost importance of active communication.

3.4 Is this the only project I am applying for?

Yes, this is the only project I am applying for.

3.5 Working Time

As this is a large project, I am willing to dedicate 5-6 hours per day until its completion. Overall, I plan to dedicate 35-40 hours per week (or more, if necessary) to building the project. Having built the basic outline and fundamental implementation already, I intend to allocate my time as follows: 50% to optimizing approaches, exploring better APIs, and implementing advanced features and functionalities; 20% to backend development and database management; and 30% to frontend features.

3.5 Other Commitments

My semester examinations will conclude on May 2nd, 2024, followed by holidays extending until the end of July. During these three months, I don't have any commitments and can dedicate my entire time to GSoC (if I get selected, hopefully :)) When my university classes commence in August for the 7th semester, I'll have four-hour classes each day. However, managing these classes alongside my participation in GSoC will be easily manageable. In September, I'll have mid-semester examinations spanning four days. Nevertheless, my proficiency in academics assures me that this won't impede my dedication to my work schedule. Additionally, I've allocated a buffer period to cover any pending work efficiently.

THANK YOU