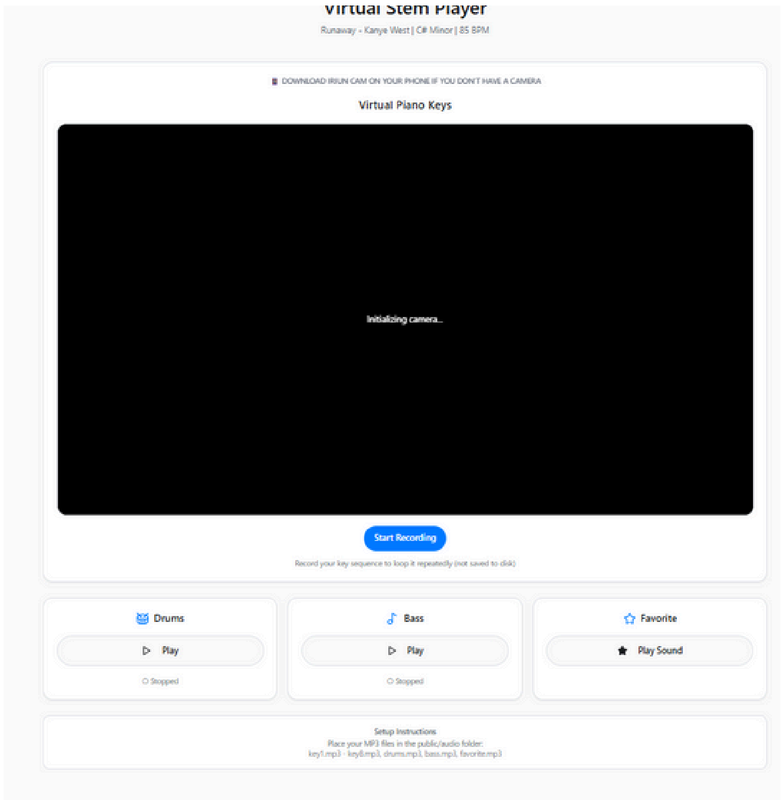
Virtual Stem Buddy

By: Abdul Haseeb
07/07/2025

For my friends Roy, Adrit and Harish.

Table of Content

Introduction



Virtual Stem Player

Runaway - Kanye West | C# Minor | 85 BPM

DOWNLOAD IRIUN CAM ON YOUR PHONE IF YOU DON'T HAVE A CAMERA

Virtual Piano Keys

Initializing camera...

Record your key sequence to loop it repeatedly (not saved to disk)

Setup Instructions

Place your MP3 files in the public/audio folder:

key1.mp3 - key8.mp3, drums.mp3, bass.mp3, favorite.mp3

Inspired by the Yeezy Stem Player by Ye and Mi.Mu Gloves by Imogen Heap, the Virtual Stem Player (also known as Virtual Stem Buddy) is a prototype offline tool that bridges virtual and physical approaches to music interaction. This software tool allows users to upload a song of their choice and automatically splits it into different stems: melody, bass, drums, and vocals.

The tool analyzes the song's BPM and key, then divides it into individual components, allowing users to play each part separately or together in a virtual environment. By deconstructing complex musical arrangements into smaller, understandable pieces, this project offers a more interactive and immersive way to experience music—moving beyond passive listening to active exploration.

For this prototype, I chose "Runaway" by Kanye West from the album My Beautiful Dark Twisted Fantasy (85 BPM, C♯ Minor). This song serves as an ideal test case due to its distinct layers: a simple piano melody, prominent drums, and clear bassline, making the stem separation process more effective and audible.

The Virtual Stem Buddy uses MediaPipe's HandLandmarker for real-time hand motion detection, enabling users to play the virtual piano keys using hand gestures captured through their device's camera. Built with React, TypeScript, and Web Audio APIs, the tool supports recording, looping, and live playback of the separated stems. The goal is to create an experience that allows people to truly interact with music, rather than simply listening through headphones.

Requirements

Physical requirements

- Webcam/Camera:

Built-in or external webcam with minimum 720p resolution (1080p recommended for better hand tracking)

- Processor:

Dual-core processor minimum (Quad-core i5/Ryzen 5 or better recommended)

- RAM:

Minimum 4GB (8GB recommended for smooth performance)

- Audio Output:

Speakers or headphones for audio playback

- Display:

Monitor with minimum 1280x720 resolution

- Internet Connection:

Required for initial setup and loading MediaPipe models

Software requirements

Operating System:

- Windows 10/11
- macOS 10.15 or later
- Linux (Ubuntu 20.04 or equivalent)

Web Browser (following with camera permissions):

- Google Chrome 90+ (recommended)
- Microsoft Edge 90+
- Firefox 88+
- Safari 14+ (macOS only)

Development Environment (if running locally):

- **Node.js v16.0.0 or higher**
- **npm v7.0.0 or higher**
- **Git (for cloning repository)**
- **Browser Permissions Required:**
- **Camera access (for hand gesture detection)**
- **Microphone access (optional, for future features)**
- **Local storage access**

External Dependencies:

- **MediaPipe Hand Landmarker model (automatically loaded from /public/models/hand_landmarker.task)**
- **Audio files in /public/audio/ directory for piano keys, drums, bass, and sound effects**

Network Requirements:

- **Minimum 5 Mbps download speed for initial model loading**
- **Stable connection for CDN resources**

Implementation & Setup

PhishShield Global follows a simple modular organized body structure, designed to make each component of the module responsible for a specific part of the phishing detection process. All components of these lightweight setup intregate to play a role in order to function tehy are corelated This allows the tool to stay lightweight, flexible, and can be expanded later to more detailed updated under developenent

Here is the detailed body structured of the the VirtualStemBuddy:

```
virtual-stem-buddy/
├── public/                    # Static assets served directly
├── src/                       # Source code
├── Configuration files        # Project setup files
└── Documentation             # Project information
```

Here's a breakdown of how the tool is built and how each part works, without showing the actual code:

Contains files served directly without processing:

📁/public/audio/ - Audio file storage
- Stores MP3/WAV files for piano keys (C, D, E, F, G, A, B)
- Drum loop audio file
- Bass loop audio file
- Special sound effects

📁/public/models/hand_landmarker.task - MediaPipe ML Model
- Pre-trained machine learning model for hand detection
- Used by MediaPipe's HandLandmarker
- Enables real-time finger tracking (21 landmarks per hand)

📁/public/robots.txt - SEO Configuration
- Controls search engine crawling behavior

📁 src/ - Source Code Directory

Main Application Files:

This folder contains all the logic for detecting phishing behavior. Each Python file inside it handles one specific job, like a mini detective.

- **src/main.tsx - Application Entry Point**
  - Initializes React application
  - Mounts root component to DOM
  - Sets up React Router and global providers
- **src/App.tsx - Root Component**
  - Defines application routing structure
  - Wraps app with theme and query providers
  - Manages global layout
- **src/App.css - Global Styles**
  - Application-wide CSS rules
- **src/index.css - Design System**
  - CSS custom properties (CSS variables)
  - Tailwind base styles and utilities
  - Color tokens for theming (light/dark mode)
  - Typography and spacing system

/src/pages/ - Page Components

- src/pages/Index.tsx - Landing/Home Page
  - Application entry interface
  - Navigation to main features

- **`src/pages/VirtualStemPlayer.tsx` - Main Application Page**
  - Orchestrates the virtual piano and audio controls
  - Manages state for drum, bass, and sound effect playback
  - Integrates VirtualPiano component with AudioControls
  - Handles audio file initialization for piano keys

- **`src/pages/NotFound.tsx` - 404 Error Page**
  - Displays when user navigates to invalid route

📁/src/components/ - Reusable Components

Custom Components:

- **`src/components/VirtualPiano.tsx` - Core Piano Component**

1. Hand Detection Integration: Uses useHandDetection hook
2. Canvas Rendering: Draws 7 piano keys on HTML canvas
3. Collision Detection: Checks if finger positions overlap with keys
4. Key Triggering: Plays audio when finger touches virtual key
5. Recording System: Records key presses with timestamps
6. Loop Playback: Plays recorded sequences repeatedly
7. Visual Feedback: Highlights active keys and shows finger positions

- **`src/components/AudioControls.tsx` - Audio Control UI**

1. Provides play/pause buttons for drum and bass loops
2. Special button for one-shot sound effects
3. Visual feedback with icons (Play, Pause, Star)
4. Responsive button states

/src/components/ui/ - UI Component Library (shadcn/ui) Pre-built, customizable components:

- button.tsx, card.tsx, dialog.tsx - Basic UI elements
- toast.tsx, toaster.tsx - Notification system
- accordion.tsx, tabs.tsx, slider.tsx - Interactive components
- form.tsx, input.tsx, select.tsx - Form elements
- 40+ production-ready components for consistent UI

📁 /src/hooks/ - Custom React Hooks

- **src/hooks/useHandDetection.tsx - Hand Tracking Logic**
    - Initializes MediaPipe HandLandmarker
    - Requests camera permissions
    - Captures video stream from webcam
    - Detects hand landmarks in real-time (60 FPS)
    - Returns array of detected hands with 21 landmarks each
    - Provides isReady status for UI feedback
    - Handles cleanup on component unmount

- **src/hooks/useAudioPlayer.tsx - Audio Loop Management**
    - Manages looping audio playback (drums, bass)
    - Provides play/pause controls
    - Handles audio element lifecycle
    - Returns isPlaying state and togglePlay function

- **src/hooks/use-toast.ts - Toast Notification Hook**
  - Manages toast notification state and actions

- **src/hooks/use-mobile.tsx - Responsive Design Hook**
  - Detects mobile vs desktop viewport

📁src/lib/ - Utility Libraries

- **src/lib/utils.ts - Helper Functions**
  - cn() function for conditional className merging
  - Combines Tailwind classes efficiently

Configuration Files (Root Directory)

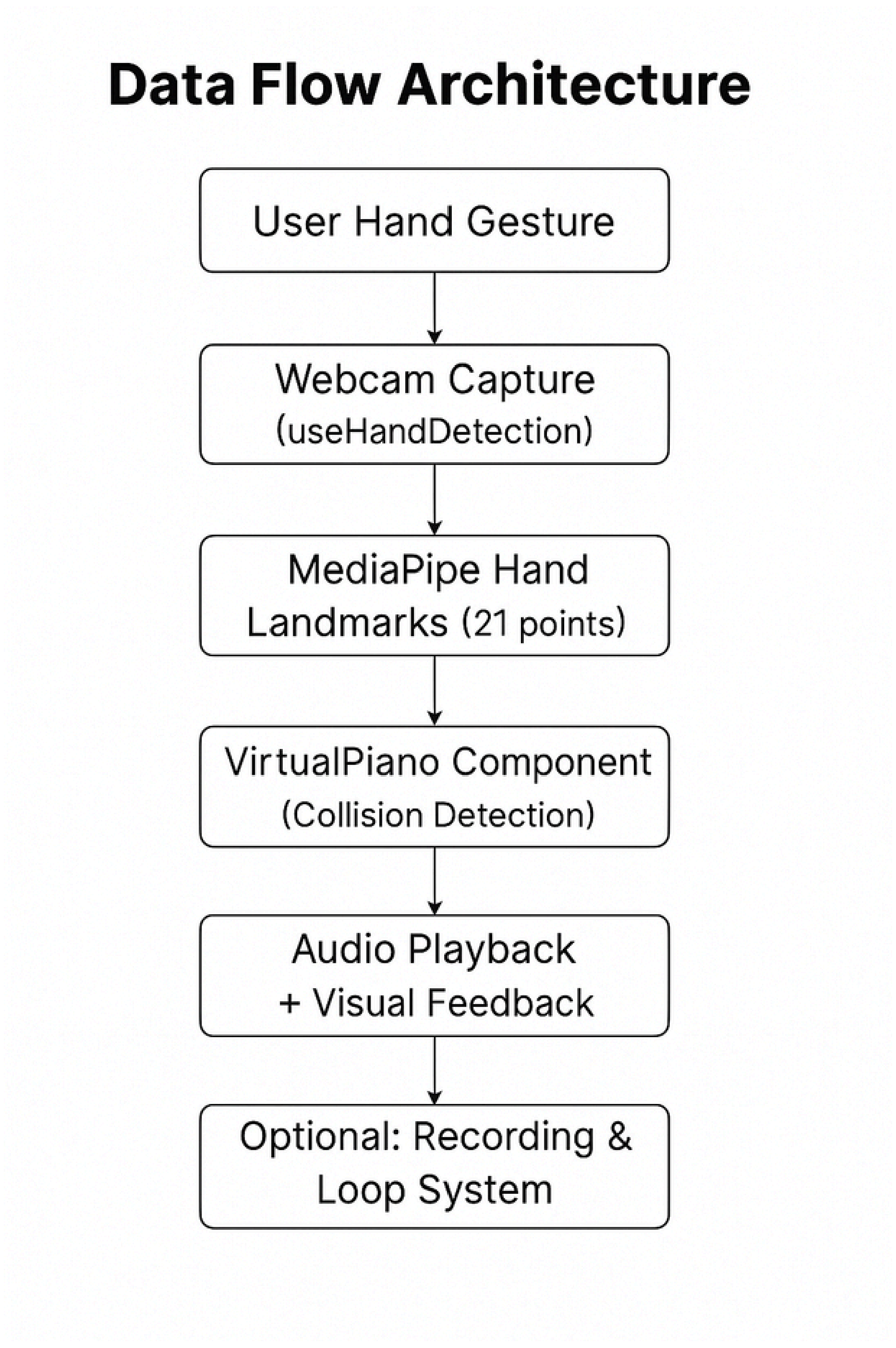
- **package.json - Project Dependencies**
  - Lists all npm packages required
  - Defines scripts (dev, build, preview)
  - Project metadata

- **vite.config.ts - Build Tool Configuration**
  - Configures Vite bundler
  - Sets up path aliases (@/ for src/)
  - Development server settings (port 8080)

- **tailwind.config.ts - Tailwind CSS Configuration**
  - Extends default Tailwind theme
  - Defines custom colors, animations
  - References CSS variables from index.css

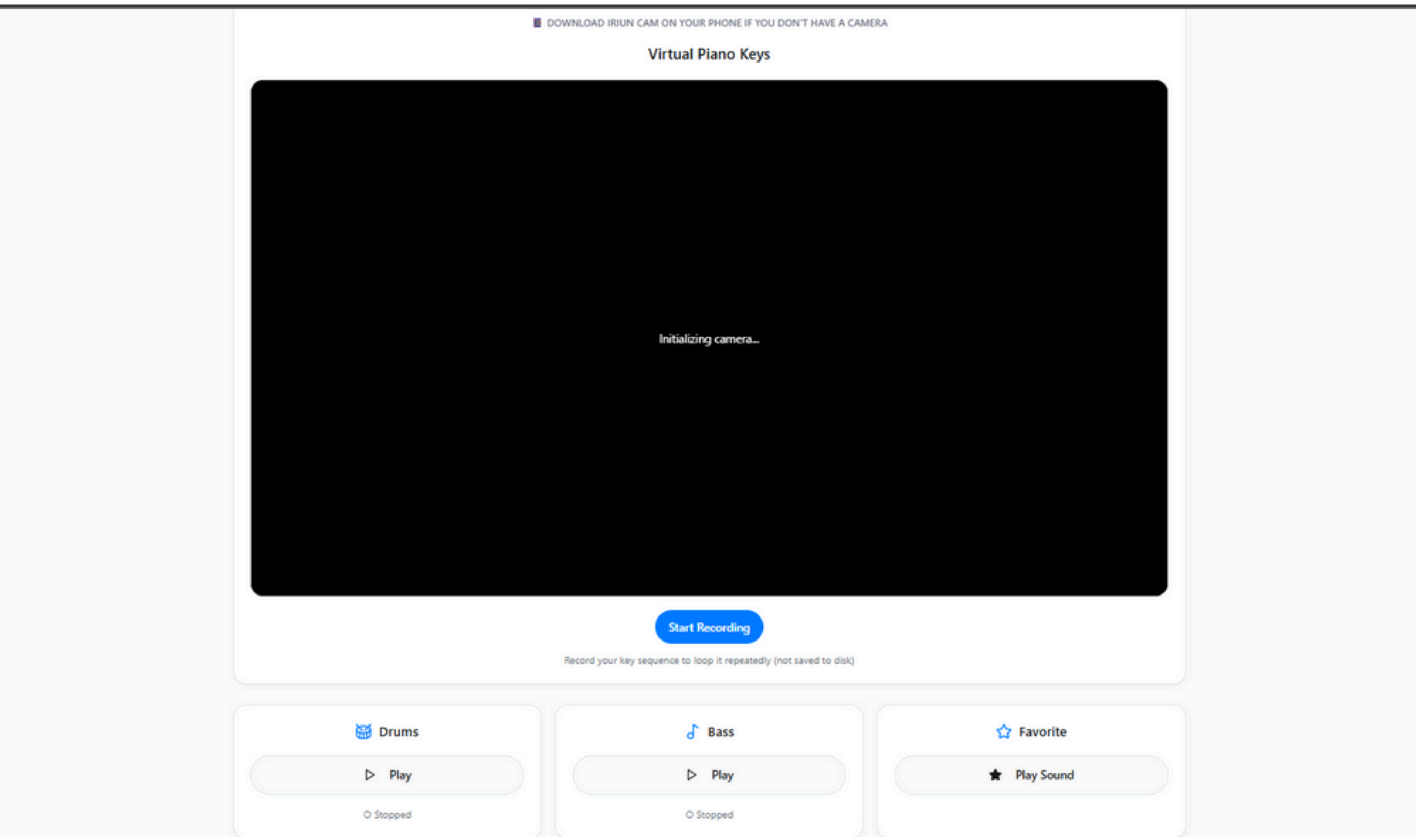
- **tsconfig.json - TypeScript Configuration**
  - TypeScript compiler options
  - Module resolution settings

- **`eslint.config.js` - Code Quality Rules**
  - Linting rules for consistent code style
- **`components.json` - shadcn/ui Configuration**
  - Configures UI component library settings


Data Flow Architecture
User Hand Gesture
Webcam Capture (useHandDetection)
MediaPipe Hand Landmarks (21 points)
VirtualPiano Component (Collision Detection)
Audio Playback + Visual Feedback
Optional: Recording & Loop System

<u>Key Technology Integration Points</u>

1. **MediaPipe → React:** Hand detection hook provides landmarks to piano component
2. **Canvas API → React**: Real-time rendering of piano keys and finger positions
3. **Web Audio API → React:** Audio playback managed through custom hooks
4. **Tailwind CSS → React:** Utility-first styling with design tokens
5. **TypeScript → React:** Type safety across all components


DOWNLOAD IRIUN CAM ON YOUR PHONE IF YOU DON'T HAVE A CAMERA

Virtual Piano Keys

Initializing camera...

Record your key sequence to loop it repeatedly (not saved to disk)

Working

Part 1: Installation & Launch

Step 1: Prerequisites Check

Ensure the following are installed:

- Node.js (v16.0.0 or higher)
- npm (v7.0.0 or higher)
- Git (if cloning from repository)

Step 2: Obtaining the Project

Download ZIP:

1. Download ZIP from GitHub repository
2. Extract to desired location
3. Navigate to extracted folder in terminal/command prompt

Step 3: Install Dependencies npm install

This command:

- Downloads all required npm packages listed in package.json
- Installs React, MediaPipe, audio libraries, and UI components
- Sets up the development environment

Step 4: Prepare Audio Files

`Before launching, ensure audio files are placed in` **`/public/audio/ directory:`**

- Piano keys: C.mp3, D.mp3, E.mp3, F.mp3, G.mp3, A.mp3, B.mp3
- Drum loop: drums.mp3
- Bass loop: bass.mp3
- Favorite sound: favorite.mp3

Step 5: Launch Development Server

```
npm run dev
```

This command:

- Starts Vite development server
- Compiles TypeScript and React code
- Opens application on http://localhost:8080
- Enables hot module replacement (instant updates on code changes)

Step 6: Open in Browser

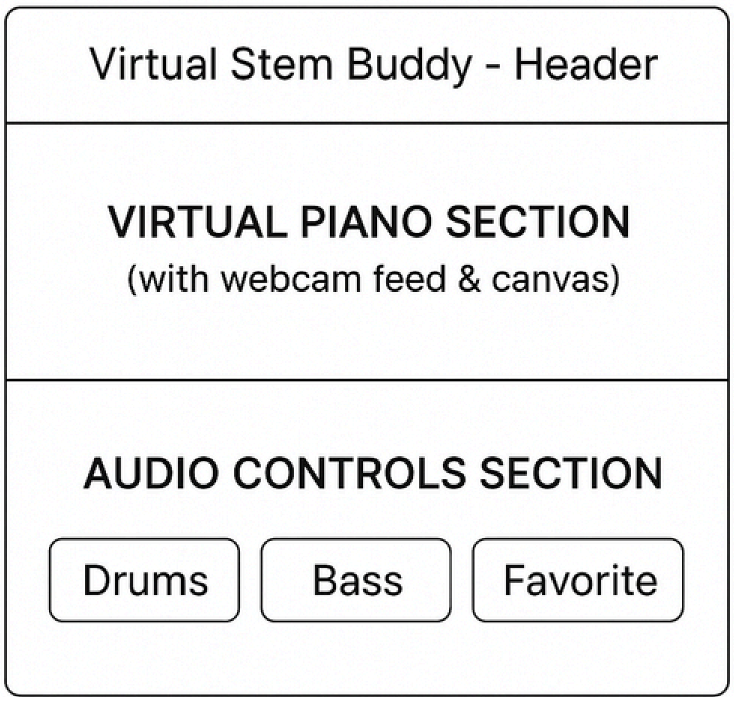
- Automatically opens in default browser, or
- Manually navigate to http://localhost:8080

Step 7: Grant Camera Permissions

- Browser will prompt for camera access
- Click "Allow" to enable hand detection
- Camera feed appears in the virtual piano section

Part 2: Tool Interface Layout

The application consists of two main sections:



Part 3: Virtual Piano Section – Detailed Explanation

3.1 Visual Components

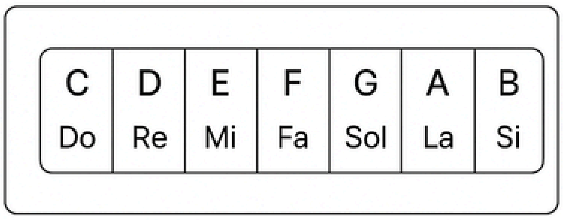
Webcam Feed Display:

- Live video stream from user's camera
- Mirrors user's movements (like looking in a mirror)
- Resolution: 640x480 pixels
- Positioned behind the interactive canvas

Interactive Canvas Overlay:

- Transparent overlay on top of webcam feed
- Dimensions: 640x480 pixels
- Displays 7 virtual piano keys
- Shows real-time finger position indicators

Piano Key Layout:



Each key:

- Width: ~91 pixels
- Height: 150 pixels
- Y-position: 250 pixels from top
- Labeled with note name and solfège syllable

3.2 Hand Detection System

How It Works:

1. Camera Capture: Webcam continuously captures video frames (30-60 FPS)
2. MediaPipe Processing: Each frame is analyzed by HandLandmarker model

- <u>Landmark Detection: Model identifies 21 points on each visible hand:</u>
    - Wrist (1 point)
    - Thumb (4 points)
    - Index finger (4 points)
    - Middle finger (4 points)
    - Ring finger (4 points)
    - Pinky finger (4 points)
- <u>Fingertip Tracking: Specifically tracks fingertips:</u>
    - Index finger tip (landmark #8)
    - Middle finger tip (landmark #12)
    - Ring finger tip (landmark #16)
    - Pinky finger tip (landmark #20)
    - Thumb tip (landmark #4)

3.3 <u>Key Triggering Mechanism Collision Detection Algorithm: For each detected fingertip:</u>

```
1. Convert normalized coordinates (0-1) to pixel coordinates

2. For each piano key:
     - Check if fingertip X is within key's X boundaries
     - Check if fingertip Y is within key's Y boundaries

3. If collision detected:
    - Mark key as "touched"
     - Play corresponding audio file
     - Change key visual to "active" state
```

Visual Feedback:

- Idle Key: Semi-transparent white with black border
- Active Key (being touched): Bright green (#4ade80)
- Finger Indicators: Purple circles (10px radius) at fingertip positions

Audio Playback:

- Loads audio file from /public/audio/[key].mp3
- Plays immediately when key is touched
- Can overlap multiple sounds simultaneously
- Resets playback if same key triggered again

3.4 Recording & Looping System Recording Feature:

- Start Recording Button: Begins capture mode
  - Records timestamp of each key press
  - Stores key index (0-6) and time offset
  - Visual indicator shows recording is active

- Stop Recording Button: Ends capture
  - Finalizes recorded sequence
  - Prepares for loop playback

Loop Playback:

- Play Loop Button: Starts automated playback
  - Uses setTimeout to trigger keys at recorded times
  - Maintains original timing between key presses
  - Repeats indefinitely until stopped

- Stop Loop Button: Halts playback
  - Clears all pending timeouts
  - Resets to idle state

Example: User plays: C → D → E (with 500ms gaps)
Recording captures:
[{ key: 0, time: 0 },{ key: 1, time: 500 },{ key: 2, time: 1000 }]
Loop replays this exact sequence repeatedly

Loop Playback:

- Play Loop Button: Starts automated playback
    - Uses setTimeout to trigger keys at recorded times
    - Maintains original timing between key presses
    - Repeats indefinitely until stopped

- Stop Loop Button: Halts playback
    - Clears all pending timeouts
    - Resets to idle state

Use Case Example:

User plays:
C → D → E (with 500ms gaps)
Recording captures:
[{ key: 0, time: 0 }, { key: 1, time: 500 }, { key: 2, time: 1000 }]

Loop replays this exact sequence repeatedly

Part 4: Audio Controls Section

Located below the virtual piano, contains three control buttons:

4.1 Drums Button Appearance:

- Icon: Play/Pause symbol
- Default state: Outlined button
- Active state: Filled primary color button
- Label: "Drums"

Functionality:

- Click to Play: Starts drum loop audio
    - Loads /public/audio/drums.mp3
    - Plays continuously on repeat (loop=true)
    - Button changes to "Pause" icon
    - Button fills with primary color

- Click to Pause: Stops drum playback
    - Pauses audio at current position
    - Button returns to "Play" icon
    - Button becomes outlined

Technical Details:

- Managed by useAudioPlayer hook
- Audio element created in memory
- Loop property set to true for continuous play
- Volume adjustable (default: 100%)

.<u>2 Bass Button Appearance:</u>

- Icon: Play/Pause symbol
- Default state: Outlined button
- Active state: Filled primary color button
- Label: "Bass"

Functionality:

- Click to Play: Starts bass loop audio
    - Loads /public/audio/bass.mp3
    - Plays continuously on repeat (loop=true)
    - Button changes to "Pause" icon
    - Button fills with primary color

- Click to Pause: Stops bass playback
    - Pauses audio at current position
    - Button returns to "Play" icon
    - Button becomes outlined

Technical Details:

- Independent from drums control
- Both can play simultaneously for layered sound
- Managed by separate useAudioPlayer instance

4.3 Favorite Button Appearance:

- Icon: Star symbol (filled)
- Default state: Outlined button with hover scale effect
- Playing state: Pulsing animation, filled button
- Label: "Play Sound" (changes to "Playing..." when active)

Functionality:

- Click to Play: Triggers one-shot sound effect
    - Loads /public/audio/favorite.mp3
    - Plays once (no loop)
    - Star icon spins during playback
    - Button text changes to "Playing..."
    - Button pulses with animation

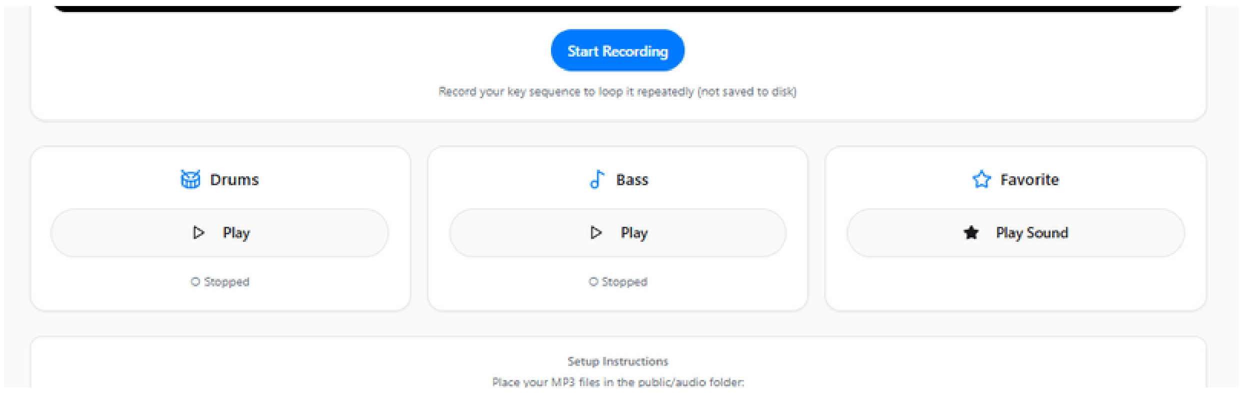
- Auto-Reset: After sound finishes
    - Returns to "Play Sound" text
    - Stops pulsing animation
    - Ready for next trigger

Special Features:

- Enhanced hover effect (scales to 110%)
- Active press effect (scales to 95%)
- Shadow effect on hover
- Different behavior from drum/bass (one-shot vs looping)

Technical Details:

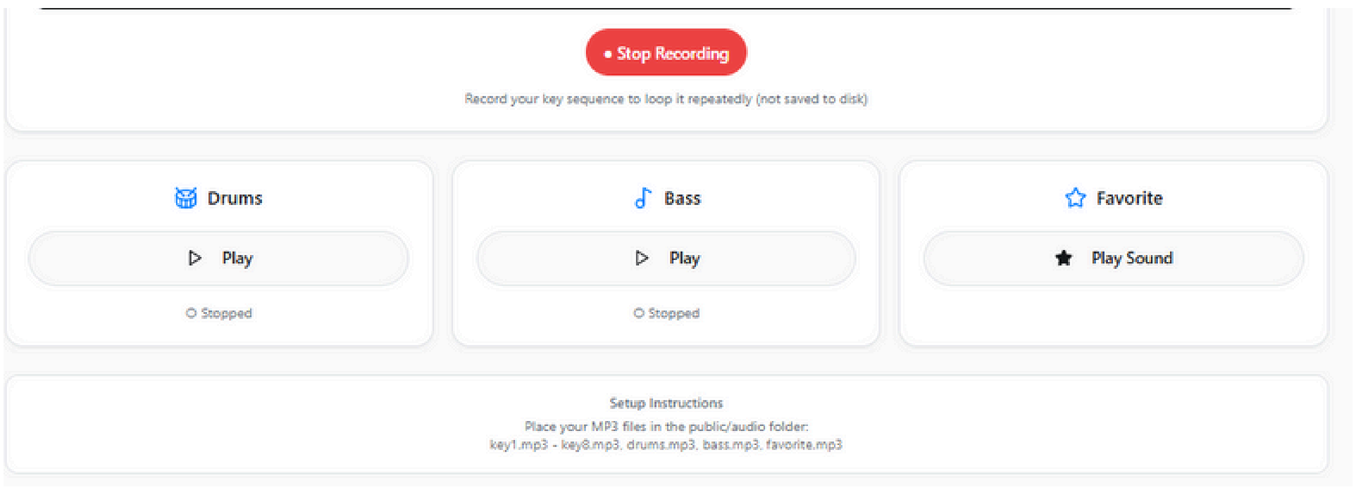
- Uses separate audio hook designed for single playback
- Automatic state reset when audio ends
- Independent from loop controls



Record your key sequence to loop it repeatedly (not saved to disk)

Setup Instructions

Place your MP3 files in the public/audio folder:



Record your key sequence to loop it repeatedly (not saved to disk)

Setup Instructions

Place your MP3 files in the public/audio folder:

key1.mp3 - key8.mp3, drums.mp3, bass.mp3, favorite.mp3

Part 5: Workflow Example - Complete Session

Scenario: User Creates a Musical Loop

1. Launch Application: User opens browser to localhost:8080

1. Grant Permissions: Allow camera access

1. Wait for Initialization: "Initializing camera..." message appears, then webcam feed loads

1. Start Background Beat: Click "Drums" button → drum loop begins

1. Add Bass Layer: Click "Bass" button → bass loop plays alongside drums

1. Play Piano Melody:
    - Hold index finger over keys
    - Touch virtual key C → plays C note
    - Move to E → plays E note
    - Create melody by moving hand across keys

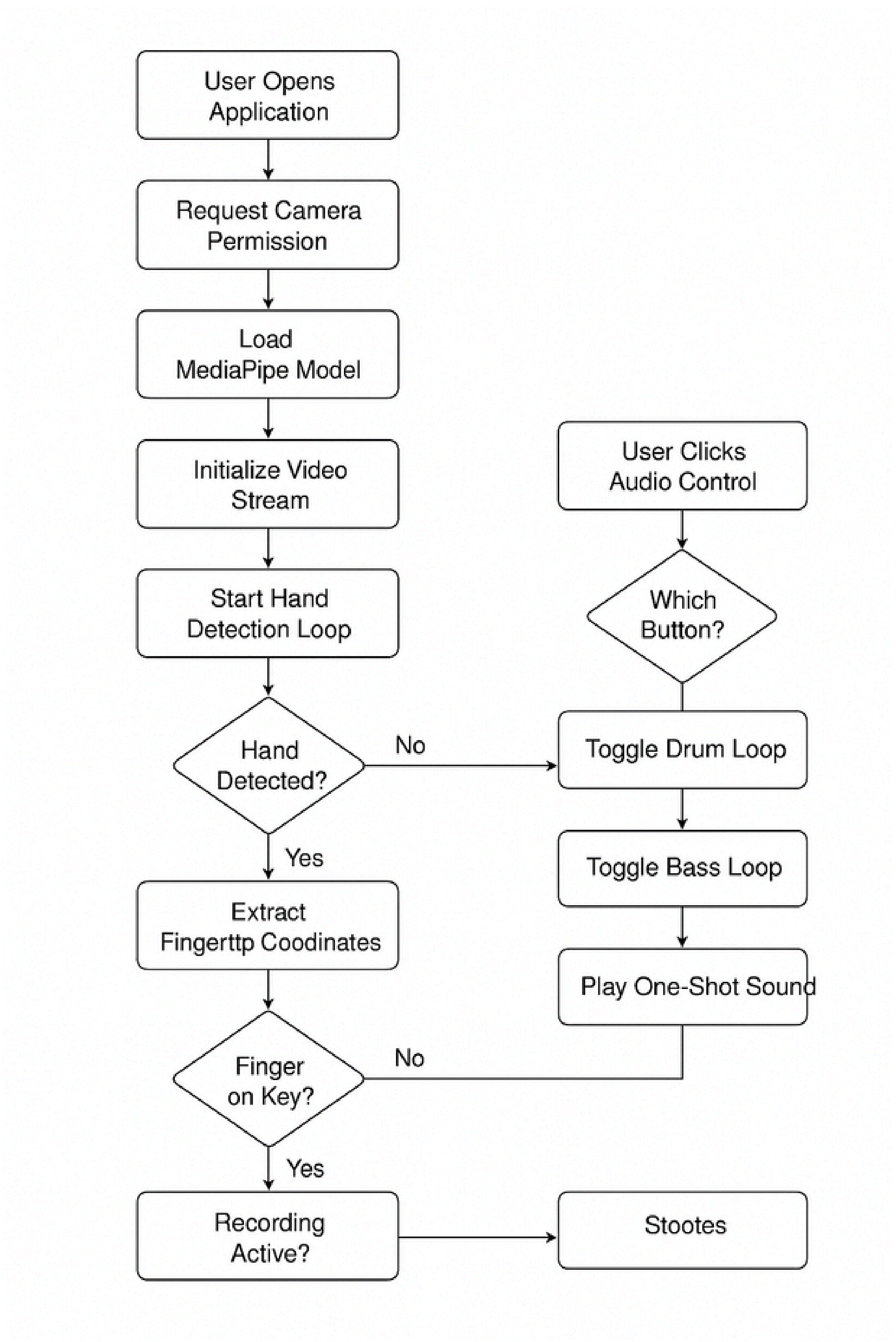
1. Record Melody:
    - Click "Start Recording"
    - Play desired melody with hand gestures
    - Click "Stop Recording"

1. Play Loop: Click "Play Loop" → melody repeats automatically

1. Add Accent: Periodically click "Favorite" button to add special sound effects

1. Performance Complete: Click "Stop Loop", pause drums and bass when finished

Part 6: Technical Process Flow



Conclusion & Futurescope

The Virtual Stem Buddy project demonstrates how interactive technology and creative expression can merge to redefine the way individuals experience music. Unlike traditional listening experiences, this tool encourages user engagement and experimentation—allowing anyone to remix, isolate, and manipulate stems of their favorite songs in real time. As technology continues to evolve, it becomes essential to rethink how we consume media. Music, being a universal form of cultural and emotional expression, deserves tools that foster creativity rather than passive consumption.

In the current technological landscape, especially within India, there remains a significant gap in research and development for creative technologies. Although the nation continues to progress as a technological powerhouse, innovation in fields combining AI, sound engineering, and digital art is still emerging. Projects like Virtual Stem Buddy highlight the potential of bridging this gap, offering a platform where technology can enhance—not replace—human creativity.

For future development, the system can be expanded with features such as advanced audio processing controls (including delay, reverb, EQ balancing, and pitch modulation) supported by real-time AI-driven stem separation. Furthermore, establishing partnerships with major record labels like Universal Music Group and integrating the tool with streaming platforms such as Spotify and Apple Music would ensure compliance with music licensing standards, enabling safe and legitimate remixing opportunities.

In conclusion, Virtual Stem Buddy is not merely a tool—it is a step toward reshaping how users interact with music in the digital age. By merging creativity with artificial intelligence, it lays the foundation for a new wave of interactive media experiences that empower users to engage with sound in deeper, more meaningful ways.

Thank You./...

@2abdulabdul on @youtube
my music xD