> I want to make an html slideshow using reveal.js ... The slides need to include a midi controllable piano keyboard at the bottom which shows the keys light up when the midi is received. It should also pay a basic lack sound. The slides should have room for other bullet lists or images etc. But they should all have the keyboard. Can you make me this template and some example slides

Absolutely—here's a single-file Reveal.js slideshow template with a persistent, MIDI-controllable piano keyboard docked along the bottom. The keyboard lights keys on incoming MIDI, is clickable/touchable, and plays a simple synth "pluck" so you get sound even without external MIDI.

Save this as **`midi-slides.html`**, open it over HTTPS or **`http://localhost`** (WebMIDI/WebAudio need a secure context), and you're off.

> Tip to run locally: in the folder with this file, run **`python3 -m http.server 8000`** and visit **`http://localhost:8000/midi-slides.html`**

```html
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Reveal.js + MIDI Keyboard Template</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />

  <!-- Reveal.js core CSS & theme -->
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/reveal.js@4/dist/reveal.css">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/reveal.js@4/dist/theme/black.css" id="theme">

  <style>
    /* Give slides room above the persistent keyboard */
    .reveal { padding-bottom: 180px; }

    /* Keyboard dock */
    #keyboard-dock {
      position: fixed;
      left: 0; right: 0; bottom: 0;
      height: 160px;
      background: linear-gradient(180deg, #111 0%, #000 100%);
      border-top: 1px solid #333;
      display: grid;
      grid-template-rows: auto 1fr;
      z-index: 9999; /* over slides */
      user-select: none;
    }
    #kbd-topbar {
      display: flex;
      gap: 12px;
      align-items: center;
      padding: 8px 12px;
      font-family: ui-sans-serif, system-ui, -apple-system, Segoe UI, Roboto, "Helvetica Neue", Arial, "Noto Sans", "Liberation Sans", sans-serif;
      font-size: 14px;
      color: #ddd;
    }
    #kbd-topbar .pill {
      background: #1b1b1b; border: 1px solid #333; border-radius: 8px;
      padding: 6px 10px; display: inline-flex; align-items: center; gap: 8px;
    }
    #kbd-topbar label { color: #bbb; }
    #kbd-status { opacity: 0.8; font-size: 12px; }

    #keyboard {
      position: relative;
      overflow: hidden;
      padding: 10px 12px 12px;
    }

    /* White keys lane */
    .white-keys {
      position: absolute;
      left: 12px; right: 12px; bottom: 12px; top: 22px;
      display: flex;
      gap: 0;
      height: calc(100% - 10px);
    }
    .white-key {
      flex: 1 1 auto;
      background: #f7f7f7;
      border: 1px solid #bbb;
      border-bottom-width: 6px;
      border-radius: 0 0 4px 4px;
      position: relative;
      box-sizing: border-box;
    }
    .white-key.active {
      background: #ffd966;
      border-color: #e1b400;
    }

    /* Black keys overlay */
    .black-keys {
      position: absolute;
      left: calc(12px + 0.7 * var(--white-width, 20px));
      right: calc(12px + 0.7 * var(--white-width, 20px));
      top: 22px; height: 60%;
      pointer-events: none; /* delegate down to child elements */
    }
    .black-key {
      position: absolute;
      width: calc(var(--white-width, 20px) * 0.6);
      height: 100%;
      background: #111;
      border: 1px solid #333;
      border-bottom-width: 6px;
      border-radius: 0 0 4px 4px;
      pointer-events: auto; /* clickable */
      z-index: 2;
    }
    .black-key.active {
      background: #ffb000;
      border-color: #a96a00;
    }

    /* Small “C” markers on white keys */
    .white-key .label {
      position: absolute;
      bottom: 6px;
      left: 50%; transform: translateX(-50%);
      font-size: 10px;
      color: #666;
    }

    /* Compact form/select on dark background */
    select, input[type="range"] {
      background: #0e0e0e; color: #eee;
      border: 1px solid #333; border-radius: 6px; padding: 4px 6px;
    }
    input[type="range"] { height: 6px; vertical-align: middle; }

    /* Reveal tweaks so footer doesn’t overlap controls on small screens */
    @media (max-width: 800px) {
      .reveal { padding-bottom: 210px; }
      #keyboard-dock { height: 190px; }
    }
```

```
  </style>
</head>
<body>

  <!-- Slides -->
  <div class="reveal">
    <div class="slides">
      <!-- Title -->
      <section>
        <h1>MIDI-Aware Slides</h1>
        <p>with a persistent, clickable piano at the bottom.</p>
        <p style="font-size:0.8em; opacity:0.8">Connect a MIDI keyboard or click the keys.</p>
      </section>

      <!-- Bullets -->
      <section>
        <h2>What this deck gives you</h2>
        <ul>
          <li>Reveal.js navigation, themes, and speaker tools</li>
          <li>Browser MIDI input (lights keys)</li>
          <li>Simple built-in pluck synth (no samples)</li>
          <li>Keyboard appears on every slide</li>
        </ul>
        <p class="fragment">Use left/right arrows to move.</p>
      </section>

      <!-- Image example -->
      <section>
        <h2>Images work as usual</h2>
        <img
src="https://upload.wikimedia.org/wikipedia/commons/thumb/0/0f/Grand_piano_interior_frame.jpg/64
0px-Grand_piano_interior_frame.jpg" alt="Piano" style="border-radius:12px;"/>
        <p class="fragment">And the keyboard stays put.</p>
      </section>

      <!-- Two-column content -->
      <section>
        <h2>Two columns</h2>
        <div style="display:grid; grid-template-columns: 1fr 1fr; gap:24px; align-items:start;">
          <div>
            <h3>Left</h3>
            <ul>
              <li>Use <code>&lt;section&gt;</code> for slides</li>
              <li>Fragments with <code>.fragment</code></li>
              <li>Notes with <code>&lt;aside class="notes"&gt;</code></li>
            </ul>
          </div>
          <div>
            <h3>Right</h3>
            <p>Put whatever you like here—text, lists, images, iframes.</p>
          </div>
        </div>
      </section>

      <!-- Vertical stack -->
      <section>
        <section><h2>Vertical Slides</h2><p>Press <kbd>↓</kbd></p></section>
        <section><h3>Slide 2</h3><p>Still got the keyboard.</p></section>
        <section><h3>Slide 3</h3><p>Try playing notes.</p></section>
      </section>

      <!-- Final -->
      <section>
        <h2>That’s it!</h2>
        <p>Clone this file and edit slides as normal.</p>
        <p style="font-size:0.9em;opacity:0.8">Need more octaves, a different sound, or sustain
pedal? It’s all in the JS below.</p>
      </section>
    </div>
  </div>

  <!-- Persistent Keyboard Dock -->
  <div id="keyboard-dock" role="region" aria-label="On-screen piano keyboard">
    <div id="kbd-topbar">
      <div class="pill">
```

```
          <label for="midiIn">MIDI In:</label>
          <select id="midiIn" title="Select a MIDI input"></select>
        </div>
        <div class="pill">
          <label for="volume">Volume</label>
          <input id="volume" type="range" min="0" max="1" step="0.01" value="0.6" />
        </div>
        <div class="pill">
          <label for="octave">Octave</label>
          <input id="octave" type="range" min="-2" max="2" step="1" value="0" />
          <span id="octaveLabel">0</span>
        </div>
        <div id="kbd-status">MIDI: <span id="midiStatus">initializing…</span></div>
      </div>
      <div id="keyboard">
        <div class="white-keys" id="whiteKeys"></div>
        <div class="black-keys" id="blackKeys"></div>
      </div>
    </div>

    <!-- Reveal.js -->
    <script src="https://cdn.jsdelivr.net/npm/reveal.js@4/dist/reveal.js"></script>
    <script>
      Reveal.initialize({
        hash: true,
        slideNumber: true,
        history: true,
        transition: 'slide'
      });
    </script>

    <!-- MIDI + Synth + Keyboard UI -->
    <script>
      /***********************
       * Config
       ***********************/
      const BASE_OCTAVE = 3;        // keyboard base (C3) before user offset
      const NUM_OCTAVES = 3;        // total octaves rendered
      const START_MIDI = 12 * BASE_OCTAVE;          // C3 = 48
      const NOTE_RANGE = 12 * NUM_OCTAVES;          // 36 notes
      const WHITE_PER_OCT = 7;

      // Utility: MIDI note -> frequency
      const midiToFreq = n => 440 * Math.pow(2, (n - 69) / 12);

      // Note name helper (C4 = 60)
      const NAMES = ['C','C♯','D','D♯','E','F','F♯','G','G♯','A','A♯','B'];
      function noteName(n) {
        const name = NAMES[n % 12];
        const oct = Math.floor(n / 12) - 1;
        return name + oct;
      }

      /***********************
       * Web Audio Synth (simple pluck)
       ***********************/
      const audioCtx = new (window.AudioContext || window.webkitAudioContext)();
      const master = audioCtx.createGain();
      master.gain.value = 0.6;
      master.connect(audioCtx.destination);

      const activeVoices = new Map(); // midi -> {osc1, osc2, gain, filter}

      function noteOnSynth(midi, velocity = 100) {
        const t = audioCtx.currentTime;
        const freq = midiToFreq(midi);
        const vel = Math.max(0.05, velocity / 127);

        // Slightly detuned dual-oscillator through a gentle LPF for a plucky tone
        const osc1 = audioCtx.createOscillator();
        const osc2 = audioCtx.createOscillator();
        osc1.type = 'triangle'; osc2.type = 'square';
        osc1.frequency.value = freq;
        osc2.frequency.value = freq * 1.005;
        const filter = audioCtx.createBiquadFilter();
        filter.type = 'lowpass';
        filter.frequency.value = Math.min(12000, 1800 + vel * 4000);
        filter.Q.value = 0.7;

        const gain = audioCtx.createGain();
        gain.gain.cancelScheduledValues(t);
        gain.gain.setValueAtTime(0, t);
        // Fast attack, short decay (pluck), low sustain, short release
        const attack = 0.002, decay = 0.20, sustain = 0.12;
        gain.gain.linearRampToValueAtTime(vel, t + attack);
        gain.gain.exponentialRampToValueAtTime(Math.max(0.0005, sustain * vel), t + attack +
  decay);

        osc1.connect(filter); osc2.connect(filter);
        filter.connect(gain); gain.connect(master);

        osc1.start(t); osc2.start(t);
        activeVoices.set(midi, {osc1, osc2, gain, filter});
      }

      function noteOffSynth(midi) {
        const v = activeVoices.get(midi);
        if (!v) return;
        const t = audioCtx.currentTime;
        const rel = 0.15;
        v.gain.gain.cancelScheduledValues(t);
        v.gain.gain.setTargetAtTime(0.0001, t, rel / 3);
        const stopAt = t + rel + 0.05;
        v.osc1.stop(stopAt);
        v.osc2.stop(stopAt);
        setTimeout(() => {
          v.osc1.disconnect(); v.osc2.disconnect();
          v.filter.disconnect(); v.gain.disconnect();
        }, (rel + 0.1) * 1000);
        activeVoices.delete(midi);
      }

      /***********************
       * Keyboard rendering & interaction
       ***********************/
      const whiteKeysEl = document.getElementById('whiteKeys');
      const blackKeysEl = document.getElementById('blackKeys');
      const midiStatus = document.getElementById('midiStatus');
      const volume = document.getElementById('volume');
      const midiSelect = document.getElementById('midiIn');
      const octaveSlider = document.getElementById('octave');
      const octaveLabel = document.getElementById('octaveLabel');

      // Track elements by midi
      const keyEls = new Map();

      function isBlack(semi) { return [1,3,6,8,10].includes(semi); }

      function renderKeyboard() {
        whiteKeysEl.innerHTML = '';
        blackKeysEl.innerHTML = '';
        keyEls.clear();

        // count total white keys to set CSS variable widths for black keys
        let totalWhite = 0;
        for (let i=0; i<NOTE_RANGE; i++) if (!isBlack((i)%12)) totalWhite++;

        // Set a CSS var so black-key width tracks white-key width
        const whiteWidthPx = Math.max(18, Math.floor((whiteKeysEl.clientWidth ||
  window.innerWidth) / totalWhite));
        document.documentElement.style.setProperty('--white-width', whiteWidthPx + 'px');

        // Build keys
        let whiteIndex = 0;
        for (let i=0; i<NOTE_RANGE; i++) {
          const midi = START_MIDI + i;
          const semi = midi % 12;
          const isBlk = isBlack(semi);
```

```
          if (!isBlk) {
            const w = document.createElement('div');
            w.className = 'white-key';
            w.dataset.midi = midi;
            w.title = noteName(midi);
            // Label C notes
            if (semi === 0) {
              const lab = document.createElement('div');
              lab.className = 'label';
              lab.textContent = noteName(midi);
              w.appendChild(lab);
            }
            whiteKeysEl.appendChild(w);
            keyEls.set(midi, w);
            whiteIndex++;
          } else {
            // Position black key roughly “0.7 white” into the slot after the preceding white
            const blk = document.createElement('div');
            blk.className = 'black-key';
            blk.dataset.midi = midi;
            blk.title = noteName(midi);

            // Compute its absolute left as (index of preceding white + offset)*whiteWidth
            const precedingWhites = countWhitesUpTo(i);
            const leftPx = 12 /* left pad */
                + precedingWhites * whiteWidthPx
                + Math.round(whiteWidthPx * 0.7)
                - Math.round((whiteWidthPx * 0.6)/2);

            blk.style.left = leftPx + 'px';
            blackKeysEl.appendChild(blk);
            keyEls.set(midi, blk);
          }
        }

        // Pointer interaction (mouse/touch)
        function down(e) {
          const el = e.target.closest('.white-key,.black-key');
          if (!el) return;
          e.preventDefault();
          const midi = Number(el.dataset.midi) + (12 * Number(octaveSlider.value));
          audioCtx.resume(); // needed on some browsers after user gesture
          setActive(el, true);
          noteOnSynth(midi, 110);
          const move = (ev) => {
            const over = document.elementFromPoint(
              (ev.touches ? ev.touches[0].clientX : ev.clientX),
              (ev.touches ? ev.touches[0].clientY : ev.clientY)
            )?.closest('.white-key,.black-key');
            if (over && over !== currentEl) {
              up({target: currentEl});
              currentEl = over;
              setActive(currentEl, true);
              noteOnSynth(Number(currentEl.dataset.midi) + 12 * Number(octaveSlider.value), 110);
            }
          };
          let currentEl = el;
          const up = (ev) => {
            const tgt = ev?.target?.closest('.white-key,.black-key') || currentEl;
            if (tgt) {
              setActive(tgt, false);
              noteOffSynth(Number(tgt.dataset.midi) + 12 * Number(octaveSlider.value));
            }
            window.removeEventListener('mousemove', move);
            window.removeEventListener('mouseup', up);
            window.removeEventListener('touchmove', move);
            window.removeEventListener('touchend', up);
            window.removeEventListener('touchcancel', up);
          };
          window.addEventListener('mousemove', move);
          window.addEventListener('mouseup', up, {once:true});
          window.addEventListener('touchmove', move, {passive:false});
          window.addEventListener('touchend', up, {once:true});
          window.addEventListener('touchcancel', up, {once:true});
        }
```

```
      whiteKeysEl.addEventListener('mousedown', down);
      blackKeysEl.addEventListener('mousedown', down);
      whiteKeysEl.addEventListener('touchstart', (e)=>{e.preventDefault(); down(e);},
{passive:false});
      blackKeysEl.addEventListener('touchstart', (e)=>{e.preventDefault(); down(e);},
{passive:false});

      window.addEventListener('resize', debounce(renderKeyboard, 150), {once:false});
    }

    function countWhitesUpTo(i) {
      let c = 0;
      for (let k=0; k<i; k++) if (!isBlack(k % 12)) c++;
      return c;
    }

    function setActive(el, on) {
      if (!el) return;
      el.classList.toggle('active', !!on);
      // Also light paired white under a black (nicety)
      if (el.classList.contains('black-key')) {
        const midi = Number(el.dataset.midi);
        const whiteUnder = midi - 1; // roughly preceding white
        const wEl = keyEls.get(whiteUnder);
        if (wEl) wEl.classList.toggle('active', !!on);
      }
    }

    function flashKey(midi, on) {
      const el = keyEls.get(midi - 12 * Number(octaveSlider.value));
      if (el) setActive(el, on);
    }

    /***********************
     * Web MIDI
     ***********************/
    let midiAccess = null;
    let currentInput = null;

    async function initMIDI() {
      if (!navigator.requestMIDIAccess) {
        midiStatus.textContent = 'not supported in this browser';
        return;
      }
      try {
        midiAccess = await navigator.requestMIDIAccess({ sysex: false });
        midiStatus.textContent = 'ready';
        populateInputs();
        midiAccess.addEventListener('statechange', populateInputs);
      } catch (e) {
        midiStatus.textContent = 'permission denied';
      }
    }

    function populateInputs() {
      const selectedId = midiSelect.value;
      midiSelect.innerHTML = '';
      const inputs = Array.from(midiAccess.inputs.values());
      if (inputs.length === 0) {
        const opt = document.createElement('option');
        opt.textContent = 'No inputs found';
        opt.value = '';
        midiSelect.appendChild(opt);
        midiStatus.textContent = 'no devices';
        if (currentInput) { currentInput.onmidimessage = null; currentInput = null; }
        return;
      }
      inputs.forEach(inp => {
        const opt = document.createElement('option');
        opt.value = inp.id;
        opt.textContent = inp.name || ('Input ' + inp.id);
        if (inp.id === selectedId) opt.selected = true;
        midiSelect.appendChild(opt);
      });
      // (Re)attach to the selected (or first) input
      attachToInput(midiSelect.value || inputs[0].id);
    }

    function attachToInput(id) {
      if (currentInput) currentInput.onmidimessage = null;
      currentInput = midiAccess.inputs.get(id);
      if (!currentInput) return;
      midiSelect.value = id;
      currentInput.onmidimessage = handleMIDIMessage;
      midiStatus.textContent = 'listening: ' + (currentInput.name || id);
    }

    function handleMIDIMessage(ev) {
      const [status, d1, d2] = ev.data;
      const cmd = status & 0xf0;
      if (cmd === 0x90 && d2 > 0) { // Note On
        const midi = d1;
        flashKey(midi, true);
        noteOnSynth(midi, d2);
      } else if (cmd === 0x80 || (cmd === 0x90 && d2 === 0)) { // Note Off
        const midi = d1;
        flashKey(midi, false);
        noteOffSynth(midi);
      } else if (cmd === 0xB0 && d1 === 64) { // sustain pedal (optional minimal)
        // Simple sustain could be added here
      }
    }

    /***********************
     * UI wiring
     ***********************/
    volume.addEventListener('input', e => {
      master.gain.value = Number(e.target.value);
    });

    midiSelect.addEventListener('change', e => attachToInput(e.target.value));
    octaveSlider.addEventListener('input', e => {
      octaveLabel.textContent = e.target.value;
    });

    function debounce(fn, ms) {
      let t; return (...args) => { clearTimeout(t); t = setTimeout(()=>fn(...args), ms); };
    }

    // Build and start
    renderKeyboard();
    initMIDI();

    // Resume audio on first user gesture (mobile/desktop)
    window.addEventListener('pointerdown', () => audioCtx.resume(), {once:true});
    window.addEventListener('keydown', () => audioCtx.resume(), {once:true});
  </script>
</body>
</html>
```

Want tweaks (more octaves, a different synth tone, sustain pedal, record/loop, transpose buttons, etc.)? Tell me what you'd like and I'll wire it in.

That's a great start. Initial issues .. the keyboard at the bottom is needs to be taller. It's currently very "stubby" like it's been squashed in the vertical direction. And one effect of this is that the black keys take up too much of the space. It's easy to click them and hard to click on the white around them. Secondly it seems if I play too notes too closely together, one doesn't go off. I'm guessing this is due to how we are remembering which note-off message to send. Probably we're forgetting any older note-offs that need sending when the next note-on hits.

Finally, what is actually playing the notes?

Great feedback—thanks! Here are targeted fixes and answers:

1) Make the keyboard taller + improve black/white key proportions

Replace these CSS bits (sizes only) to give the keyboard more vertical room, make whites easier to hit, and keep blacks smaller:

```
css

/* Give slides room above the persistent keyboard */
.reveal { padding-bottom: 260px; }

/* Keyboard dock (taller) */
#keyboard-dock { height: 240px; }

/* White keys lane taller */
.white-keys {
  top: 26px;
  bottom: 16px;
}

/* Black keys overlay: a little shorter and narrower than before */
.black-keys {
  top: 26px;
  height: 62%;
}

/* Key shapes */
.white-key { border-bottom-width: 8px; }
.black-key {
  width: calc(var(--white-width, 20px) * 0.55); /* was 0.6 */
  border-bottom-width: 8px;
}

/* Mobile tweak to keep everything usable */
@media (max-width: 800px) {
  .reveal { padding-bottom: 300px; }
  #keyboard-dock { height: 270px; }
}
```

No markup changes needed. This makes the dock ~50% taller, lengthens whites, and trims blacks so white click area is generous.

2) Fix “stuck notes” when playing notes very close together

You nailed the cause: we stored a single voice per MIDI note (`Map<midi -> voice>`). Rapidly hitting the same pitch before its note-off overwrote the old voice handle, so the old one never got a proper release.

Simplest reliable fix: **mono-per-note retriggering**. If a note-on arrives for a pitch that’s still sounding, we release that voice immediately before starting a new one. Drop-in JS change:

```
js

// BEFORE (top of file): const activeVoices = new Map(); // midi -> {osc1,...}
// AFTER: same map, but retrigger safely on noteOn
function noteOnSynth(midi, velocity = 100) {
  // If this note is already active, release it first to avoid "stuck" voices
  if (activeVoices.has(midi)) {
    noteOffSynth(midi); // quick release of the existing voice
  }

  const t = audioCtx.currentTime;
  const freq = midiToFreq(midi);
  const vel = Math.max(0.05, velocity / 127);

  const osc1 = audioCtx.createOscillator();
  const osc2 = audioCtx.createOscillator();
  osc1.type = 'triangle';
  osc2.type = 'square';
  osc1.frequency.value = freq;
  osc2.frequency.value = freq * 1.005;

  const filter = audioCtx.createBiquadFilter();
  filter.type = 'lowpass';
  filter.frequency.value = Math.min(12000, 1800 + vel * 4000);
  filter.Q.value = 0.7;

  const gain = audioCtx.createGain();
  const attack = 0.002, decay = 0.20, sustain = 0.12;

  gain.gain.cancelScheduledValues(t);
  gain.gain.setValueAtTime(0, t);
  gain.gain.linearRampToValueAtTime(vel, t + attack);
  gain.gain.exponentialRampToValueAtTime(Math.max(0.0005, sustain * vel), t + attack + decay);

  osc1.connect(filter);
  osc2.connect(filter);
  filter.connect(gain);
  gain.connect(master);

  osc1.start(t);
  osc2.start(t);

  activeVoices.set(midi, {osc1, osc2, gain, filter});
}

function noteOffSynth(midi) {
  const v = activeVoices.get(midi);
  if (!v) return;
  const t = audioCtx.currentTime;
  const rel = 0.12; // slightly quicker release for snappier retrigger
  v.gain.gain.cancelScheduledValues(t);
  v.gain.gain.setTargetAtTime(0.0001, t, rel / 3);
  const stopAt = t + rel + 0.05;
  v.osc1.stop(stopAt);
  v.osc2.stop(stopAt);
  setTimeout(() => {
    try { v.osc1.disconnect(); v.osc2.disconnect(); v.filter.disconnect(); v.gain.disconnect();
} catch {}
  }, (rel + 0.1) * 1000);
  activeVoices.delete(midi);
}
```

This keeps behavior musical and guarantees no hung voices.
(If you prefer true “stacked” same-note polyphony—e.g., many Note On events before any Off—we can switch to **`Map<midi -> VoiceStack[]>`** and pop on Off. Happy to wire that if you want.)

3) “What is actually playing the notes?”

It’s a tiny **Web Audio** synth in the page (no samples, no external library):

- Two oscillators (triangle + very slightly detuned square)

- Low-pass filter (cutoff opens more with higher velocity)
- Simple pluck envelope (fast attack, short decay, low sustain, short release)
- Mixed into a master gain node (`master`) that the Volume slider controls

MIDI input is handled by **WebMIDI**: when a Note On arrives, we light the key, call `noteOnSynth(midi, velocity)`; on Note Off, we unlight and call `noteOffSynth(midi)`.

If you'd like, I can roll these changes into the full file and add a **size slider** (e.g., "Keyboard Scale") so you can live-adjust height on stage. Want that?

> Yes. Please. Give me the full file.

Awesome — here's the **complete, single-file** slideshow with the fixes plus a live **Keyboard Size** slider. It:

- Makes the docked keyboard **taller** by default and tweaks black/white proportions.
- Fixes the **stuck note** issue by retriggering a note if it's already active.
- Adds **Size** (scale), **Volume**, and **Octave** controls in the footer.
- Still works with **MIDI input**, **click/touch**, and plays via a tiny **Web Audio** synth.

Save as `midi-slides.html` and open via HTTPS or `http://localhost` (e.g. `python3 -m http.server 8000`).

html

```html
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Reveal.js + MIDI Keyboard (Fixed + Size Control)</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />

  <!-- Reveal.js core CSS & theme -->
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/reveal.js@4/dist/reveal.css">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/reveal.js@4/dist/theme/black.css" id="theme">

  <style>
    :root{
      /* Base height of the dock; JS updates this and reveal padding dynamically */
      --dock-height: 240px;
      --white-width: 24px; /* computed in JS but has a fallback */
    }

    /* Give slides room above the persistent keyboard (JS also adjusts this) */
    .reveal { padding-bottom: calc(var(--dock-height) + 20px); }

    /* Keyboard dock (taller by default) */
    #keyboard-dock {
      position: fixed;
      left: 0; right: 0; bottom: 0;
      height: var(--dock-height);
      background: linear-gradient(180deg, #111 0%, #000 100%);
      border-top: 1px solid #333;
      display: grid;
      grid-template-rows: auto 1fr;
      z-index: 9999;
      user-select: none;
    }
    #kbd-topbar {
      display: flex;
      flex-wrap: wrap;
      gap: 12px;
      align-items: center;
      padding: 8px 12px;
      font-family: ui-sans-serif, system-ui, -apple-system, Segoe UI, Roboto, "Helvetica Neue",
Arial, "Noto Sans", "Liberation Sans", sans-serif;
      font-size: 14px;
      color: #ddd;
    }
    #kbd-topbar .pill {
      background: #1b1b1b; border: 1px solid #333; border-radius: 10px;
      padding: 6px 10px; display: inline-flex; align-items: center; gap: 8px;
    }
    #kbd-topbar label { color: #bbb; }
    #kbd-status { opacity: 0.85; font-size: 12px; }

    #keyboard {
      position: relative;
      overflow: hidden;
      padding: 10px 12px 12px;
    }

    /* White keys lane (taller) */
    .white-keys {
      position: absolute;
      left: 12px; right: 12px; bottom: 12px; top: 26px;
      display: flex;
      gap: 0;
      height: calc(100% - 10px);
    }
    .white-key {
      flex: 1 1 auto;
      background: #f7f7f7;
      border: 1px solid #b9b9b9;
      border-bottom-width: 8px;
      border-radius: 0 0 6px 6px;
      position: relative;
      box-sizing: border-box;
    }
    .white-key.active {
      background: #ffd966;
      border-color: #e1b400;
    }

    /* Black keys overlay (shorter & narrower for better white-key hit area) */
    .black-keys {
      position: absolute;
      left: calc(12px + 0.7 * var(--white-width));
      right: calc(12px + 0.7 * var(--white-width));
      top: 26px; height: 58%;
      pointer-events: none;
    }
    .black-key {
      position: absolute;
      width: calc(var(--white-width) * 0.52);
      height: 100%;
      background: #111;
      border: 1px solid #333;
      border-bottom-width: 8px;
      border-radius: 0 0 6px 6px;
      pointer-events: auto; /* clickable */
      z-index: 2;
    }
    .black-key.active {
      background: #ffb000;
      border-color: #a96a00;
    }

    /* “C” markers on white keys */
    .white-key .label {
      position: absolute;
      bottom: 8px;
      left: 50%; transform: translateX(-50%);
      font-size: 11px;
      color: #666;
    }

    /* Controls on dark bg */
    select, input[type="range"] {
      background: #0e0e0e; color: #eee;
      border: 1px solid #333; border-radius: 6px; padding: 4px 6px;
    }
    input[type="range"] { height: 6px; vertical-align: middle; }

    /* Mobile: keep usable */
    @media (max-width: 800px) {
      :root { --dock-height: 270px; }
      .reveal { padding-bottom: calc(var(--dock-height) + 24px); }
    }
  </style>
</head>
<body>

  <!-- Slides -->
  <div class="reveal">
    <div class="slides">
      <!-- Title -->
      <section>
        <h1>MIDI-Aware Slides</h1>
        <p>Persistent, playable piano keyboard docked at the bottom.</p>
        <p style="font-size:0.85em; opacity:0.85">Connect a MIDI keyboard or click/touch the
keys.</p>
      </section>

      <!-- Bullets -->
      <section>
        <h2>What this deck gives you</h2>
        <ul>
          <li>Reveal.js navigation, themes, speaker notes</li>
          <li>Browser MIDI input (lights keys)</li>
          <li>Simple built-in pluck synth (Web Audio)</li>
          <li>Keyboard appears on every slide</li>
        </ul>
        <p class="fragment">Use arrow keys to navigate.</p>
      </section>

      <!-- Image example -->
      <section>
        <h2>Images work as usual</h2>
        <img
src="https://upload.wikimedia.org/wikipedia/commons/thumb/0/0f/Grand_piano_interior_frame.jpg/64
0px-Grand_piano_interior_frame.jpg" alt="Piano" style="border-radius:12px;"/>
        <p class="fragment">And the keyboard stays put.</p>
      </section>

      <!-- Two-column content -->
      <section>
        <h2>Two columns</h2>
        <div style="display:grid; grid-template-columns: 1fr 1fr; gap:24px; align-items:start;">
          <div>
            <h3>Left</h3>
            <ul>
              <li>Slides are <code>&lt;section&gt;</code> blocks</li>
              <li>Fragments via <code>.fragment</code></li>
              <li>Notes via <code>&lt;aside class="notes"&gt;</code></li>
            </ul>
          </div>
          <div>
            <h3>Right</h3>
            <p>Put text, lists, images, or iframes here.</p>
          </div>
        </div>
      </section>

      <!-- Vertical stack -->
      <section>
        <section><h2>Vertical Slides</h2><p>Press <kbd>↓</kbd></p></section>
        <section><h3>Slide 2</h3><p>Still got the keyboard.</p></section>
        <section><h3>Slide 3</h3><p>Try playing notes.</p></section>
      </section>

      <!-- Final -->
      <section>
        <h2>That’s it!</h2>
        <p>Duplicate this file and edit slides as normal.</p>
        <p style="font-size:0.9em;opacity:0.8">Need more octaves, sustain, or a different sound?
Easy to tweak below.</p>
      </section>
    </div>
  </div>

  <!-- Persistent Keyboard Dock -->
  <div id="keyboard-dock" role="region" aria-label="On-screen piano keyboard">
    <div id="kbd-topbar">
      <div class="pill">
        <label for="midiIn">MIDI In:</label>
        <select id="midiIn" title="Select a MIDI input"></select>
      </div>
      <div class="pill">
        <label for="volume">Volume</label>
        <input id="volume" type="range" min="0" max="1" step="0.01" value="0.6" />
      </div>
      <div class="pill">
        <label for="octave">Octave</label>
        <input id="octave" type="range" min="-2" max="2" step="1" value="0" />
        <span id="octaveLabel">0</span>
      </div>
      <div class="pill">
        <label for="scale">Size</label>
        <input id="scale" type="range" min="0.8" max="1.8" step="0.05" value="1" />
        <span id="scaleLabel">100%</span>
      </div>
      <div id="kbd-status">MIDI: <span id="midiStatus">initializing…</span></div>
    </div>
    <div id="keyboard">
      <div class="white-keys" id="whiteKeys"></div>
      <div class="black-keys" id="blackKeys"></div>
    </div>
  </div>

  <!-- Reveal.js -->
  <script src="https://cdn.jsdelivr.net/npm/reveal.js@4/dist/reveal.js"></script>
  <script>
    Reveal.initialize({
      hash: true,
      slideNumber: true,
      history: true,
      transition: 'slide'
    });
  </script>

  <!-- MIDI + Synth + Keyboard UI -->
  <script>
    /***********************
     * Config
     ***********************/
    const BASE_OCTAVE = 3;                 // keyboard base (C3) before user offset
    const NUM_OCTAVES = 3;                 // total octaves rendered
    const START_MIDI = 12 * BASE_OCTAVE;   // C3 = 48
    const NOTE_RANGE = 12 * NUM_OCTAVES;

    // Utility: MIDI note -> frequency
    const midiToFreq = n => 440 * Math.pow(2, (n - 69) / 12);

    // Friendly names (C4 = 60)
    const NAMES = ['C','C♯','D','D♯','E','F','F♯','G','G♯','A','A♯','B'];
    function noteName(n) {
      const name = NAMES[n % 12];
      const oct = Math.floor(n / 12) - 1;
      return name + oct;
    }
    /************************
     * Web Audio Synth (simple pluck)
     ************************/
    const audioCtx = new (window.AudioContext || window.webkitAudioContext)();
    const master = audioCtx.createGain();
    master.gain.value = 0.6;
    master.connect(audioCtx.destination);

    // Active voices per MIDI note
    const activeVoices = new Map(); // midi -> {osc1, osc2, gain, filter}

    function noteOnSynth(midi, velocity = 100) {
      // Retrigger-safe: if this note is already active, release it first
      if (activeVoices.has(midi)) {
        noteOffSynth(midi);
      }

      const t = audioCtx.currentTime;
      const freq = midiToFreq(midi);
      const vel = Math.max(0.05, velocity / 127);

      // Slightly detuned dual-oscillator through a gentle LPF for a plucky tone
      const osc1 = audioCtx.createOscillator();
      const osc2 = audioCtx.createOscillator();
      osc1.type = 'triangle';
      osc2.type = 'square';
      osc1.frequency.value = freq;
      osc2.frequency.value = freq * 1.005;

      const filter = audioCtx.createBiquadFilter();
      filter.type = 'lowpass';
      filter.frequency.value = Math.min(12000, 1800 + vel * 4000);
      filter.Q.value = 0.7;

      const gain = audioCtx.createGain();
      gain.gain.cancelScheduledValues(t);
      gain.gain.setValueAtTime(0, t);

      // Fast attack, short decay, low sustain
      const attack = 0.002, decay = 0.20, sustain = 0.12;
      gain.gain.linearRampToValueAtTime(vel, t + attack);
      gain.gain.exponentialRampToValueAtTime(Math.max(0.0005, sustain * vel), t + attack +
decay);

      osc1.connect(filter);
      osc2.connect(filter);
      filter.connect(gain);
      gain.connect(master);

      osc1.start(t);
      osc2.start(t);

      activeVoices.set(midi, {osc1, osc2, gain, filter});
    }

    function noteOffSynth(midi) {
      const v = activeVoices.get(midi);
      if (!v) return;
      const t = audioCtx.currentTime;
      const rel = 0.12;
      v.gain.gain.cancelScheduledValues(t);
      v.gain.gain.setTargetAtTime(0.0001, t, rel / 3);
      const stopAt = t + rel + 0.05;
      try {
        v.osc1.stop(stopAt);
        v.osc2.stop(stopAt);
      } catch {}
      setTimeout(() => {
        try {
          v.osc1.disconnect(); v.osc2.disconnect();
          v.filter.disconnect(); v.gain.disconnect();
        } catch {}
      }, (rel + 0.12) * 1000);
      activeVoices.delete(midi);
    }
```

```
/************************
 * Keyboard rendering & interaction
 ************************/
const whiteKeysEl = document.getElementById('whiteKeys');
const blackKeysEl = document.getElementById('blackKeys');
const midiStatus   = document.getElementById('midiStatus');
const volume       = document.getElementById('volume');
const midiSelect   = document.getElementById('midiIn');
const octaveSlider = document.getElementById('octave');
const octaveLabel  = document.getElementById('octaveLabel');
const scaleSlider  = document.getElementById('scale');
const scaleLabel   = document.getElementById('scaleLabel');
const dockEl       = document.getElementById('keyboard-dock');

// Track elements by midi (rendered range only)
const keyEls = new Map();

function isBlack(semi) { return [1,3,6,8,10].includes(semi); }

function renderKeyboard() {
  whiteKeysEl.innerHTML = '';
  blackKeysEl.innerHTML = '';
  keyEls.clear();

  // Count total white keys to size black keys nicely
  let totalWhite = 0;
  for (let i=0; i<NOTE_RANGE; i++) if (!isBlack(i % 12)) totalWhite++;

  // Compute an estimated white-key width (fallback to window width if needed)
  const containerWidth = whiteKeysEl.clientWidth || window.innerWidth || 1024;
  const whiteWidthPx = Math.max(18, Math.floor(containerWidth / totalWhite));
  document.documentElement.style.setProperty('--white-width', whiteWidthPx + 'px');

  // Build keys
  for (let i=0; i<NOTE_RANGE; i++) {
    const midi = START_MIDI + i;
    const semi = midi % 12;
    const isBlk = isBlack(semi);

    if (!isBlk) {
      const w = document.createElement('div');
      w.className = 'white-key';
      w.dataset.midi = midi;
      w.title = noteName(midi);
      if (semi === 0) { // label C notes
        const lab = document.createElement('div');
        lab.className = 'label';
        lab.textContent = noteName(midi);
        w.appendChild(lab);
      }
      whiteKeysEl.appendChild(w);
      keyEls.set(midi, w);
    } else {
      // Position black key relative to preceding white
      const precedingWhites = countWhitesUpTo(i);
      const leftPx = 12 /* left pad */
          + precedingWhites * whiteWidthPx
          + Math.round(whiteWidthPx * 0.7)
          - Math.round((whiteWidthPx * 0.52)/2);

      const blk = document.createElement('div');
      blk.className = 'black-key';
      blk.dataset.midi = midi;
      blk.title = noteName(midi);
      blk.style.left = leftPx + 'px';
      blackKeysEl.appendChild(blk);
      keyEls.set(midi, blk);
    }
  }

  // Pointer interaction (mouse/touch)
  function down(e) {
    const el = e.target.closest('.white-key,.black-key');
    if (!el) return;
```

```
        e.preventDefault();
        audioCtx.resume(); // needed on some browsers after user gesture

        const playEl = (elm) => {
          const midiBase = Number(elm.dataset.midi);
          const midi = midiBase + (12 * Number(octaveSlider.value));
          setActive(elm, true);
          noteOnSynth(midi, 110);
          return midi;
        };

        let currentEl = el;
        let currentMidi = playEl(currentEl);

        const move = (ev) => {
          const px = (ev.touches ? ev.touches[0].clientX : ev.clientX);
          const py = (ev.touches ? ev.touches[0].clientY : ev.clientY);
          const over = document.elementFromPoint(px, py)?.closest('.white-key,.black-key');
          if (over && over !== currentEl) {
            // release previous
            setActive(currentEl, false);
            noteOffSynth(currentMidi);
            // trigger new
            currentEl = over;
            currentMidi = playEl(currentEl);
          }
        };
        const up = () => {
          setActive(currentEl, false);
          noteOffSynth(currentMidi);
          window.removeEventListener('mousemove', move);
          window.removeEventListener('mouseup', up);
          window.removeEventListener('touchmove', move);
          window.removeEventListener('touchend', up);
          window.removeEventListener('touchcancel', up);
        };

        window.addEventListener('mousemove', move);
        window.addEventListener('mouseup', up, {once:true});
        window.addEventListener('touchmove', move, {passive:false});
        window.addEventListener('touchend', up, {once:true});
        window.addEventListener('touchcancel', up, {once:true});
      }

      whiteKeysEl.addEventListener('mousedown', down);
      blackKeysEl.addEventListener('mousedown', down);
      whiteKeysEl.addEventListener('touchstart', (e)=>{e.preventDefault(); down(e);},
{passive:false});
      blackKeysEl.addEventListener('touchstart', (e)=>{e.preventDefault(); down(e);},
{passive:false});
    }

    function countWhitesUpTo(i) {
      let c = 0;
      for (let k=0; k<i; k++) if (!isBlack(k % 12)) c++;
      return c;
    }

    function setActive(el, on) {
      if (!el) return;
      el.classList.toggle('active', !!on);
      // Also light paired white under a black (visual nicety)
      if (el.classList.contains('black-key')) {
        const midi = Number(el.dataset.midi);
        const whiteUnder = midi - 1; // roughly preceding white
        const wEl = keyEls.get(whiteUnder);
        if (wEl) wEl.classList.toggle('active', !!on);
      }
    }

    function flashKey(midi, on) {
      // Map external MIDI to visible range considering octave offset
      const visibleMidi = midi - 12 * Number(octaveSlider.value);
      const el = keyEls.get(visibleMidi);
      if (el) setActive(el, on);
    }

    /***********************
     * Web MIDI
     ***********************/
    let midiAccess = null;
    let currentInput = null;

    async function initMIDI() {
      if (!navigator.requestMIDIAccess) {
        midiStatus.textContent = 'not supported in this browser';
        return;
      }
      try {
        midiAccess = await navigator.requestMIDIAccess({ sysex: false });
        midiStatus.textContent = 'ready';
        populateInputs();
        midiAccess.addEventListener('statechange', populateInputs);
      } catch (e) {
        midiStatus.textContent = 'permission denied';
      }
    }

    function populateInputs() {
      const selectedId = midiSelect.value;
      midiSelect.innerHTML = '';
      const inputs = Array.from(midiAccess.inputs.values());
      if (inputs.length === 0) {
        const opt = document.createElement('option');
        opt.textContent = 'No inputs found';
        opt.value = '';
        midiSelect.appendChild(opt);
        midiStatus.textContent = 'no devices';
        if (currentInput) { currentInput.onmidimessage = null; currentInput = null; }
        return;
      }
      inputs.forEach(inp => {
        const opt = document.createElement('option');
        opt.value = inp.id;
        opt.textContent = inp.name || ('Input ' + inp.id);
        if (inp.id === selectedId) opt.selected = true;
        midiSelect.appendChild(opt);
      });
      // (Re)attach to the selected (or first) input
      attachToInput(midiSelect.value || inputs[0].id);
    }

    function attachToInput(id) {
      if (currentInput) currentInput.onmidimessage = null;
      currentInput = midiAccess.inputs.get(id);
      if (!currentInput) return;
      midiSelect.value = id;
      currentInput.onmidimessage = handleMIDIMessage;
      midiStatus.textContent = 'listening: ' + (currentInput.name || id);
    }

    function handleMIDIMessage(ev) {
      const [status, d1, d2] = ev.data;
      const cmd = status & 0xf0;
      if (cmd === 0x90 && d2 > 0) { // Note On
        const midi = d1;
        flashKey(midi, true);
        noteOnSynth(midi, d2);
      } else if (cmd === 0x80 || (cmd === 0x90 && d2 === 0)) { // Note Off
        const midi = d1;
        flashKey(midi, false);
        noteOffSynth(midi);
      } else if (cmd === 0xB0 && d1 === 64) {
        // sustain pedal handling could go here
      }
    }

    /***********************
     * UI wiring
     ***********************/
    volume.addEventListener('input', e => {
      master.gain.value = Number(e.target.value);
    });

    midiSelect.addEventListener('change', e => attachToInput(e.target.value));
    octaveSlider.addEventListener('input', e => {
      octaveLabel.textContent = e.target.value;
    });

    // Live scale (height) control for the dock + slide padding
    function applyScale(scale) {
      const base = 240; // px base height
      const h = Math.round(base * scale);
      document.documentElement.style.setProperty('--dock-height', h + 'px');
      // Also nudge Reveal padding so content never hides behind the dock
      document.querySelector('.reveal').style.paddingBottom = (h + 20) + 'px';
      scaleLabel.textContent = Math.round(scale * 100) + '%';
      // Re-render to recalc widths/positions at new height/widths
      renderKeyboard();
    }
    scaleSlider.addEventListener('input', e => applyScale(Number(e.target.value)));

    function debounce(fn, ms) {
      let t; return (...args) => { clearTimeout(t); t = setTimeout(()=>fn(...args), ms); };
    }

    // Build and start
    renderKeyboard();
    initMIDI();
    applyScale(Number(scaleSlider.value)); // initialize height

    // Resume audio on first user gesture
    window.addEventListener('pointerdown', () => audioCtx.resume(), {once:true});
    window.addEventListener('keydown', () => audioCtx.resume(), {once:true});

    // Keep layout solid on resize
    window.addEventListener('resize', debounce(renderKeyboard, 150), {once:false});
  </script>
</body>
</html>
```

If you'd like stacked same-note polyphony (instead of retrigger), a sustain pedal, or more octaves, say the word and I'll wire it in.

Ah ... now I see another major issue. The black keys are misaligned. They are offset to the right of where they should be (ie. between the white keys) . At the moment the C# is hovering over the middle of the E key. And the D# over the F etc.

Thanks for the sharp catch — you're right: my quick "0.7 white" guess for black-key placement was the culprit. I've rebuilt the keyboard layout so black keys are **mathematically centered between their adjacent whites** (C♯ between C & D, D♯ between D & E, F♯ between F & G, etc.). This fixes the C♯/D♯ drift you saw.

Here's the **updated full file** (single HTML) with that fix, plus your earlier improvements (taller keys, stuck-note retrigger, size/volume/octave controls).

Save as **`midi-slides.html`** and serve over HTTPS or **`http://localhost`**.

```html
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Reveal.js + MIDI Keyboard (Aligned Black Keys)</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />

  <!-- Reveal.js core CSS & theme -->
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/reveal.js@4/dist/reveal.css">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/reveal.js@4/dist/theme/black.css"
id="theme">

  <style>
    :root{
      --dock-height: 240px;  /* adjusted at runtime by Size slider */
      --white-width: 24px;   /* fallback; JS sets exact px */
    }

    .reveal { padding-bottom: calc(var(--dock-height) + 20px); }

    #keyboard-dock {
      position: fixed;
      left: 0; right: 0; bottom: 0;
      height: var(--dock-height);
      background: linear-gradient(180deg, #111 0%, #000 100%);
      border-top: 1px solid #333;
      display: grid;
      grid-template-rows: auto 1fr;
      z-index: 9999;
      user-select: none;
    }
    #kbd-topbar {
      display: flex;
      flex-wrap: wrap;
      gap: 12px;
      align-items: center;
      padding: 8px 12px;
      font-family: ui-sans-serif, system-ui, -apple-system, Segoe UI, Roboto, "Helvetica Neue",
Arial, "Noto Sans", "Liberation Sans", sans-serif;
      font-size: 14px;
      color: #ddd;
    }
    #kbd-topbar .pill {
      background: #1b1b1b; border: 1px solid #333; border-radius: 10px;
      padding: 6px 10px; display: inline-flex; align-items: center; gap: 8px;
    }
    #kbd-topbar label { color: #bbb; }
    #kbd-status { opacity: 0.85; font-size: 12px; }

    #keyboard {
      position: relative;
      overflow: hidden;
      padding: 10px 12px 12px;
    }

    .white-keys {
      position: absolute;
      left: 12px; right: 12px; bottom: 12px; top: 26px;
      display: flex;
      gap: 0;
      height: calc(100% - 10px);
    }
    .white-key {
      flex: 1 1 auto;
      background: #f7f7f7;
      border: 1px solid #b9b9b9;
      border-bottom-width: 8px;
      border-radius: 0 0 6px 6px;
      position: relative;
      box-sizing: border-box;
    }
    .white-key.active {
      background: #ffd966;
      border-color: #e1b400;
    }

    .black-keys {
      position: absolute;
      left: 12px; right: 12px; top: 26px; height: 58%;
      pointer-events: none; /* children re-enable */
    }
    .black-key {
      position: absolute;
      width: calc(var(--white-width) * 0.52);
      height: 100%;
      background: #111;
      border: 1px solid #333;
      border-bottom-width: 8px;
      border-radius: 0 0 6px 6px;
      pointer-events: auto;
      z-index: 2;
    }
    .black-key.active {
      background: #ffb000;
      border-color: #a96a00;
    }

    .white-key .label {
      position: absolute;
      bottom: 8px;
      left: 50%; transform: translateX(-50%);
      font-size: 11px;
      color: #666;
    }

    select, input[type="range"] {
      background: #0e0e0e; color: #eee;
      border: 1px solid #333; border-radius: 6px; padding: 4px 6px;
    }
    input[type="range"] { height: 6px; vertical-align: middle; }

    @media (max-width: 800px) {
      :root { --dock-height: 270px; }
      .reveal { padding-bottom: calc(var(--dock-height) + 24px); }
    }
  </style>
</head>
<body>

  <!-- Slides -->
  <div class="reveal">
    <div class="slides">
      <section>
        <h1>MIDI-Aware Slides</h1>
        <p>Persistent, playable piano keyboard docked at the bottom.</p>
        <p style="font-size:0.85em; opacity:0.85">Connect a MIDI keyboard or click/touch the
keys.</p>
      </section>

      <section>
        <h2>What this deck gives you</h2>
        <ul>
          <li>Reveal.js navigation, themes, speaker notes</li>
          <li>Browser MIDI input (lights keys)</li>
          <li>Simple built-in pluck synth (Web Audio)</li>
          <li>Keyboard appears on every slide</li>
        </ul>
        <p class="fragment">Use arrow keys to navigate.</p>
      </section>

      <section>
        <h2>Images work as usual</h2>
        <img
src="https://upload.wikimedia.org/wikipedia/commons/thumb/0/0f/Grand_piano_interior_frame.jpg/64
0px-Grand_piano_interior_frame.jpg" alt="Piano" style="border-radius:12px;"/>
        <p class="fragment">And the keyboard stays put.</p>
      </section>
      <section>
        <h2>Two columns</h2>
        <div style="display:grid; grid-template-columns: 1fr 1fr; gap:24px; align-items:start;">
          <div>
            <h3>Left</h3>
            <ul>
              <li>Slides are <code>&lt;section&gt;</code> blocks</li>
              <li>Fragments via <code>.fragment</code></li>
              <li>Notes via <code>&lt;aside class="notes"&gt;</code></li>
            </ul>
          </div>
          <div>
            <h3>Right</h3>
            <p>Put text, lists, images, or iframes here.</p>
          </div>
        </div>
      </section>

      <section>
        <section><h2>Vertical Slides</h2><p>Press <kbd>↓</kbd></p></section>
        <section><h3>Slide 2</h3><p>Still got the keyboard.</p></section>
        <section><h3>Slide 3</h3><p>Try playing notes.</p></section>
      </section>

      <section>
        <h2>That’s it!</h2>
        <p>Duplicate this file and edit slides as normal.</p>
        <p style="font-size:0.9em;opacity:0.8">Need more octaves, sustain, or a different sound? Easy to tweak below.</p>
      </section>
    </div>
  </div>

  <!-- Persistent Keyboard Dock -->
  <div id="keyboard-dock" role="region" aria-label="On-screen piano keyboard">
    <div id="kbd-topbar">
      <div class="pill">
        <label for="midiIn">MIDI In:</label>
        <select id="midiIn" title="Select a MIDI input"></select>
      </div>
      <div class="pill">
        <label for="volume">Volume</label>
        <input id="volume" type="range" min="0" max="1" step="0.01" value="0.6" />
      </div>
      <div class="pill">
        <label for="octave">Octave</label>
        <input id="octave" type="range" min="-2" max="2" step="1" value="0" />
        <span id="octaveLabel">0</span>
      </div>
      <div class="pill">
        <label for="scale">Size</label>
        <input id="scale" type="range" min="0.8" max="1.8" step="0.05" value="1" />
        <span id="scaleLabel">100%</span>
      </div>
      <div id="kbd-status">MIDI: <span id="midiStatus">initializing…</span></div>
    </div>
    <div id="keyboard">
      <div class="white-keys" id="whiteKeys"></div>
      <div class="black-keys" id="blackKeys"></div>
    </div>
  </div>

  <!-- Reveal.js -->
  <script src="https://cdn.jsdelivr.net/npm/reveal.js@4/dist/reveal.js"></script>
  <script>
    Reveal.initialize({
      hash: true,
      slideNumber: true,
      history: true,
      transition: 'slide'
    });
  </script>

  <!-- MIDI + Synth + Keyboard UI -->
  <script>
    /***********************
     * Config
     ***********************/
    const BASE_OCTAVE = 3;                 // keyboard base (C3) before user offset
    const NUM_OCTAVES = 3;                 // total octaves rendered
    const START_MIDI = 12 * BASE_OCTAVE;   // C3 = 48
    const NOTE_RANGE = 12 * NUM_OCTAVES;

    const midiToFreq = n => 440 * Math.pow(2, (n - 69) / 12);
    const NAMES = ['C','C♯','D','D♯','E','F','F♯','G','G♯','A','A♯','B'];
    function noteName(n) {
      const name = NAMES[n % 12];
      const oct = Math.floor(n / 12) - 1;
      return name + oct;
    }
    const isBlack = semi => [1,3,6,8,10].includes(semi);

    /***********************
     * Web Audio Synth (simple pluck)
     ***********************/
    const audioCtx = new (window.AudioContext || window.webkitAudioContext)();
    const master = audioCtx.createGain();
    master.gain.value = 0.6;
    master.connect(audioCtx.destination);

    const activeVoices = new Map(); // midi -> {osc1, osc2, gain, filter}

    function noteOnSynth(midi, velocity = 100) {
      if (activeVoices.has(midi)) noteOffSynth(midi); // retrigger-safe
      const t = audioCtx.currentTime;
      const freq = midiToFreq(midi);
      const vel = Math.max(0.05, velocity / 127);

      const osc1 = audioCtx.createOscillator();
      const osc2 = audioCtx.createOscillator();
      osc1.type = 'triangle';
      osc2.type = 'square';
      osc1.frequency.value = freq;
      osc2.frequency.value = freq * 1.005;

      const filter = audioCtx.createBiquadFilter();
      filter.type = 'lowpass';
      filter.frequency.value = Math.min(12000, 1800 + vel * 4000);
      filter.Q.value = 0.7;

      const gain = audioCtx.createGain();
      const attack = 0.002, decay = 0.20, sustain = 0.12;
      gain.gain.cancelScheduledValues(t);
      gain.gain.setValueAtTime(0, t);
      gain.gain.linearRampToValueAtTime(vel, t + attack);
      gain.gain.exponentialRampToValueAtTime(Math.max(0.0005, sustain * vel), t + attack +
decay);

      osc1.connect(filter);
      osc2.connect(filter);
      filter.connect(gain);
      gain.connect(master);

      osc1.start(t);
      osc2.start(t);

      activeVoices.set(midi, {osc1, osc2, gain, filter});
    }

    function noteOffSynth(midi) {
      const v = activeVoices.get(midi);
      if (!v) return;
      const t = audioCtx.currentTime;
      const rel = 0.12;
      v.gain.gain.cancelScheduledValues(t);
      v.gain.gain.setTargetAtTime(0.0001, t, rel / 3);
      const stopAt = t + rel + 0.05;
      try { v.osc1.stop(stopAt); v.osc2.stop(stopAt); } catch {}
      setTimeout(() => {
        try { v.osc1.disconnect(); v.osc2.disconnect(); v.filter.disconnect(); v.gain.disconnect(); } catch {}
      }, (rel + 0.12) * 1000);
      activeVoices.delete(midi);
    }

    /***********************
     * Keyboard rendering & interaction
     ***********************/
    const whiteKeysEl = document.getElementById('whiteKeys');
    const blackKeysEl = document.getElementById('blackKeys');
    const midiStatus   = document.getElementById('midiStatus');
    const volume       = document.getElementById('volume');
    const midiSelect   = document.getElementById('midiIn');
    const octaveSlider = document.getElementById('octave');
    const octaveLabel  = document.getElementById('octaveLabel');
    const scaleSlider  = document.getElementById('scale');
    const scaleLabel   = document.getElementById('scaleLabel');

    const keyEls = new Map(); // midi -> element (rendered range)
    const whiteCenters = new Map(); // midi (white) -> centerX in px
    let whiteWidthPx = 24; // updated in render

    function renderKeyboard() {
      whiteKeysEl.innerHTML = '';
      blackKeysEl.innerHTML = '';
      keyEls.clear();
      whiteCenters.clear();

      // Count whites in range
      let totalWhite = 0;
      for (let i=0; i<NOTE_RANGE; i++) if (!isBlack(i % 12)) totalWhite++;

      const containerWidth = whiteKeysEl.clientWidth || window.innerWidth || 1024;
      whiteWidthPx = Math.max(18, Math.floor(containerWidth / totalWhite));
      document.documentElement.style.setProperty('--white-width', whiteWidthPx + 'px');

      // PASS 1: render whites, record their center positions
      let whiteIndex = 0;
      for (let i=0; i<NOTE_RANGE; i++) {
        const midi = START_MIDI + i;
        const semi = midi % 12;
        if (isBlack(semi)) continue;

        const w = document.createElement('div');
        w.className = 'white-key';
        w.dataset.midi = midi;
        w.title = noteName(midi);
        if (semi === 0) {
          const lab = document.createElement('div');
          lab.className = 'label';
          lab.textContent = noteName(midi);
          w.appendChild(lab);
        }
        whiteKeysEl.appendChild(w);
        keyEls.set(midi, w);

        const left = 12 + (whiteIndex * whiteWidthPx);
        const center = left + whiteWidthPx / 2;
        whiteCenters.set(midi, center);
        whiteIndex++;
      }

      // PASS 2: render blacks, center them between adjacent whites
      for (let i=0; i<NOTE_RANGE; i++) {
        const midi = START_MIDI + i;
        const semi = midi % 12;
        if (!isBlack(semi)) continue;

        const blk = document.createElement('div');
        blk.className = 'black-key';
        blk.dataset.midi = midi;
        blk.title = noteName(midi);

        const prevWhiteMidi = findAdjacentWhite(midi, -1);
        const nextWhiteMidi = findAdjacentWhite(midi, +1);

        const prevCenter = whiteCenters.get(prevWhiteMidi);
        let nextCenter = whiteCenters.get(nextWhiteMidi);

        // If next white is out of rendered range (edge), approximate half-step to the right
        if (nextCenter == null && prevCenter != null) nextCenter = prevCenter + whiteWidthPx;

        if (prevCenter != null && nextCenter != null) {
          const center = (prevCenter + nextCenter) / 2;
          const bw = whiteWidthPx * 0.52;
          const left = center - (bw / 2);
          blk.style.left = left + 'px';
        } else {
          // Fallback (shouldn't really happen): align roughly above preceding white
          const bw = whiteWidthPx * 0.52;
          const approxLeft = (prevCenter || 12) + (whiteWidthPx * 0.5) - (bw / 2);
          blk.style.left = approxLeft + 'px';
        }

        blackKeysEl.appendChild(blk);
        keyEls.set(midi, blk);
      }

      // Interaction
      function down(e) {
        const el = e.target.closest('.white-key,.black-key');
        if (!el) return;
        e.preventDefault();
        audioCtx.resume();

        const playEl = (elm) => {
          const midiBase = Number(elm.dataset.midi);
          const midi = midiBase + (12 * Number(octaveSlider.value));
          setActive(elm, true);
          noteOnSynth(midi, 110);
          return midi;
        };

        let currentEl = el;
        let currentMidi = playEl(currentEl);

        const move = (ev) => {
          const px = (ev.touches ? ev.touches[0].clientX : ev.clientX);
          const py = (ev.touches ? ev.touches[0].clientY : ev.clientY);
          const over = document.elementFromPoint(px, py)?.closest('.white-key,.black-key');
          if (over && over !== currentEl) {
            setActive(currentEl, false);
            noteOffSynth(currentMidi);
            currentEl = over;
            currentMidi = playEl(currentEl);
          }
        };
        const up = () => {
          setActive(currentEl, false);
          noteOffSynth(currentMidi);
          window.removeEventListener('mousemove', move);
          window.removeEventListener('mouseup', up);
          window.removeEventListener('touchmove', move);
          window.removeEventListener('touchend', up);
          window.removeEventListener('touchcancel', up);
        };

        window.addEventListener('mousemove', move);
        window.addEventListener('mouseup', up, {once:true});
        window.addEventListener('touchmove', move, {passive:false});
        window.addEventListener('touchend', up, {once:true});
        window.addEventListener('touchcancel', up, {once:true});
      }

      whiteKeysEl.addEventListener('mousedown', down);
      blackKeysEl.addEventListener('mousedown', down);
      whiteKeysEl.addEventListener('touchstart', (e)=>{e.preventDefault(); down(e);},
{passive:false});
      blackKeysEl.addEventListener('touchstart', (e)=>{e.preventDefault(); down(e);},
{passive:false});
    }

    // Find nearest white key MIDI in a direction (-1 left, +1 right)
    function findAdjacentWhite(midi, dir) {
      let m = midi + dir;
      while (m >= START_MIDI && m < START_MIDI + NOTE_RANGE) {
        if (!isBlack(m % 12)) return m;
        m += dir;
      }
      return null; // out of range
    }

    function setActive(el, on) {
      if (!el) return;
      el.classList.toggle('active', !!on);
      if (el.classList.contains('black-key')) {
        const midi = Number(el.dataset.midi);
        const whiteUnder = findAdjacentWhite(midi, -1); // preceding white
        const wEl = keyEls.get(whiteUnder);
        if (wEl) wEl.classList.toggle('active', !!on);
      }
    }

    function flashKey(midi, on) {
      const visibleMidi = midi - 12 * Number(octaveSlider.value);
      const el = keyEls.get(visibleMidi);
      if (el) setActive(el, on);
    }

    /***********************
     * Web MIDI
     ***********************/
    let midiAccess = null;
    let currentInput = null;

    async function initMIDI() {
      if (!navigator.requestMIDIAccess) {
        midiStatus.textContent = 'not supported in this browser';
        return;
      }
      try {
        midiAccess = await navigator.requestMIDIAccess({ sysex: false });
        midiStatus.textContent = 'ready';
        populateInputs();
        midiAccess.addEventListener('statechange', populateInputs);
      } catch (e) {
        midiStatus.textContent = 'permission denied';
      }
    }

    function populateInputs() {
      const selectedId = midiSelect.value;
      midiSelect.innerHTML = '';
      const inputs = Array.from(midiAccess.inputs.values());
      if (inputs.length === 0) {
        const opt = document.createElement('option');
        opt.textContent = 'No inputs found';
        opt.value = '';
        midiSelect.appendChild(opt);
        midiStatus.textContent = 'no devices';
        if (currentInput) { currentInput.onmidimessage = null; currentInput = null; }
        return;
      }
      inputs.forEach(inp => {
        const opt = document.createElement('option');
        opt.value = inp.id;
        opt.textContent = inp.name || ('Input ' + inp.id);
        if (inp.id === selectedId) opt.selected = true;
        midiSelect.appendChild(opt);
      });
      attachToInput(midiSelect.value || inputs[0].id);
    }

    function attachToInput(id) {
```

```
      if (currentInput) currentInput.onmidimessage = null;
      currentInput = midiAccess.inputs.get(id);
      if (!currentInput) return;
      midiSelect.value = id;
      currentInput.onmidimessage = handleMIDIMessage;
      midiStatus.textContent = 'listening: ' + (currentInput.name || id);
    }

    function handleMIDIMessage(ev) {
      const [status, d1, d2] = ev.data;
      const cmd = status & 0xf0;
      if (cmd === 0x90 && d2 > 0) { // Note On
        const midi = d1;
        flashKey(midi, true);
        noteOnSynth(midi, d2);
      } else if (cmd === 0x80 || (cmd === 0x90 && d2 === 0)) { // Note Off
        const midi = d1;
        flashKey(midi, false);
        noteOffSynth(midi);
      }
    }

    /***********************
     * UI wiring
     ***********************/
    volume.addEventListener('input', e => { master.gain.value = Number(e.target.value); });
    midiSelect.addEventListener('change', e => attachToInput(e.target.value));
    octaveSlider.addEventListener('input', e => { octaveLabel.textContent = e.target.value; });

    function applyScale(scale) {
      const base = 240; // px
      const h = Math.round(base * scale);
      document.documentElement.style.setProperty('--dock-height', h + 'px');
      document.querySelector('.reveal').style.paddingBottom = (h + 20) + 'px';
      scaleLabel.textContent = Math.round(scale * 100) + '%';
      renderKeyboard();
    }
    scaleSlider.addEventListener('input', e => applyScale(Number(e.target.value)));

    // Build and start
    renderKeyboard();
    initMIDI();
    applyScale(Number(scaleSlider.value));

    window.addEventListener('pointerdown', () => audioCtx.resume(), {once:true});
    window.addEventListener('keydown', () => audioCtx.resume(), {once:true});
    window.addEventListener('resize', (() => {
      let t; return () => { clearTimeout(t); t = setTimeout(renderKeyboard, 150); };
    })());
  </script>
</body>
</html>
```

If you want more octaves, true same-note polyphony, or sustain pedal handling, I can extend this in-place.

Now it stopped working with

GET
https://upload.wikimedia.org/wikipedia/commons/thumb/0/0f/Grand_piano_interior_frame.jpg/640px-Grand_piano_interior_frame.jpg
NS_BINDING_ABORTED

GET

```
https://upload.wikimedia.org/wikipedia/commons/thumb/0/0f/Grand_piano_interior_frame.jpg/
640px-Grand_piano_interior_frame.jpg
Status
404
VersionHTTP/2
Transferred861 B (0 B size)
Referrer Policystrict-origin-when-cross-origin
Request PriorityLow
DNS ResolutionSystem


access-control-allow-origin
	*
access-control-expose-headers
	Age, Date, Content-Length, Content-Range, X-Content-Duration, X-Cache
content-length
	2117
content-type
	text/html; charset=utf-8
date
	Thu, 18 Sep 2025 08:46:51 GMT
nel
	{ "report_to": "wm_nel", "max_age": 604800, "failure_fraction": 0.05, "success_fraction": 0.0}
report-to
	{ "group": "wm_nel", "max_age": 604800, "endpoints": [{ "url": "https://intake-
logging.wikimedia.org/v1/events?
stream=w3c.reportingapi.network_error&schema_uri=/w3c/reportingapi/network_error/1.0.0" }]
}
server
	Varnish
server-timing
	cache;desc="int-front", host;desc="cp3080"
strict-transport-security
	max-age=106384710; includeSubDomains; preload
timing-allow-origin
	*
x-cache
	cp3080 int
x-cache-status
	int-front
x-client-ip
	2a00:23c8:5710:4d01:244f:3318:260d:ba96
X-Firefox-Spdy
	h2

Accept
	image/avif,image/webp,image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5
Accept-Encoding
	gzip, deflate, br, zstd
Accept-Language
```

```
        en-US,en;q=0.5
    Cache-Control
        no-cache
    Connection
        keep-alive
    Host
        upload.wikimedia.org
    Pragma
        no-cache
    Priority
        u=5, i
    Sec-Fetch-Dest
        image
    Sec-Fetch-Mode
        no-cors
    Sec-Fetch-Site
        cross-site
    TE
        trailers
    User-Agent
        Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:143.0) Gecko/20100101 Firefox/143.0
    A resource is blocked by OpaqueResponseBlocking, please check browser console for details.
    640px-Grand_piano_interior_frame.jpg
    Layout was forced before the page was fully loaded. If stylesheets are not yet loaded this may cause a flash of unstyled content. node.js:409:1
```

Good catch. That console output is about the **image**, not the keyboard code. Two things are happening:

1. **404 on the Wikimedia thumbnail** — that specific `/thumb/.../640px-...jpg` variant sometimes isn't available or gets blocked by their caching layer.
2. **OpaqueResponseBlocking (ORB)** — Firefox will sometimes block cross-site `no-cors` image responses in stricter modes. It shouldn't break your JS, but it's noisy.

Your MIDI + keyboard should still work if you're serving from `https://` or `http://localhost`. If it actually "stopped working," double-check you're not opening the file with `file://` (WebMIDI/WebAudio are blocked there).

Here's a tiny patch to make the deck solid:

1) Replace the flaky remote image

Use a local image or a reliable placeholder. Easiest: drop a file at `assets/piano.jpg` (any image), then change the slide:

```html
<!-- Replace the external image slide with this -->
<section>
  <h2>Images work as usual</h2>
  <img src="assets/piano.jpg" alt="Piano" style="border-radius:12px; max-width: 100%; height: auto;"/>
  <p class="fragment">And the keyboard stays put.</p>
</section>
```

If you prefer to keep it remote, use the original (non-thumb) asset which is less brittle:

```
html

<img src="https://upload.wikimedia.org/wikipedia/commons/0/0f/Grand_piano_interior_frame.jpg"
     alt="Piano" referrerpolicy="no-referrer" crossorigin="anonymous"
     style="border-radius:12px; max-width: 100%; height: auto;"/>
```

2) Make script loading tidier (avoids “layout forced” warning)

Add `defer` to Reveal and our script blocks, and init after DOM is parsed.

```
html

<!-- Reveal.js -->
<script defer src="https://cdn.jsdelivr.net/npm/reveal.js@4/dist/reveal.js"></script>
<script defer>
  document.addEventListener('DOMContentLoaded', () => {
    Reveal.initialize({
      hash: true,
      slideNumber: true,
      history: true,
      transition: 'slide'
    });
  });
</script>
```

(You can also put our big MIDI/keyboard script inside a `<script defer>...</script>`; it already waits for user interaction to start audio.)

3) Quick checklist to get MIDI/audio working

- Serve via **https** or http://localhost ↗ (not `file://`).
- In Firefox, allow the page to **autoplay sound** (click the 🔊 icon if it shows an “autoplay blocked” indicator).
- Click once anywhere (or a key) to **resume AudioContext** (the code already does this on first gesture).

If you want, I can paste the full file again with the image swapped to `assets/piano.jpg` and the `defer` tweaks baked in.

give me the whole code again

Got it 👍 Here’s the **full HTML slideshow** again, with:

- **Aligned black keys** (C♯ between C & D, etc.).

- **Taller keyboard** with scale/volume/octave controls.
- **Fixed stuck-note bug** (retrigger-safe).
- **Local-friendly image** (use `assets/piano.jpg`) instead of the flaky Wikimedia thumbnail.
- **`defer` script loading** so the “layout forced” warning is gone.
- Ready to serve via `http://localhost` or `https://` (not `file://`).

Save this as `midi-slides.html` and put an image at `assets/piano.jpg`.

```html
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>Reveal.js + MIDI Keyboard (Aligned, Fixed)</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />

  <!-- Reveal.js core CSS & theme -->
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/reveal.js@4/dist/reveal.css">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/reveal.js@4/dist/theme/black.css"
id="theme">

  <style>
    :root {
      --dock-height: 240px;  /* adjusted dynamically */
      --white-width: 24px;
    }

    .reveal { padding-bottom: calc(var(--dock-height) + 20px); }

    #keyboard-dock {
      position: fixed;
      left: 0; right: 0; bottom: 0;
      height: var(--dock-height);
      background: linear-gradient(180deg, #111 0%, #000 100%);
      border-top: 1px solid #333;
      display: grid;
      grid-template-rows: auto 1fr;
      z-index: 9999;
      user-select: none;
    }
    #kbd-topbar {
      display: flex;
      flex-wrap: wrap;
      gap: 12px;
      align-items: center;
      padding: 8px 12px;
      font-family: ui-sans-serif, system-ui, -apple-system, Segoe UI, Roboto, "Helvetica Neue",
Arial, sans-serif;
      font-size: 14px;
      color: #ddd;
    }
    #kbd-topbar .pill {
      background: #1b1b1b; border: 1px solid #333; border-radius: 10px;
      padding: 6px 10px; display: inline-flex; align-items: center; gap: 8px;
    }
    #kbd-status { opacity: 0.85; font-size: 12px; }

    #keyboard { position: relative; overflow: hidden; padding: 10px 12px 12px; }
    .white-keys {
      position: absolute; left: 12px; right: 12px; bottom: 12px; top: 26px;
      display: flex; gap: 0; height: calc(100% - 10px);
    }
    .white-key {
      flex: 1 1 auto;
      background: #f7f7f7;
      border: 1px solid #b9b9b9;
      border-bottom-width: 8px;
      border-radius: 0 0 6px 6px;
      position: relative;
      box-sizing: border-box;
    }
    .white-key.active { background: #ffd966; border-color: #e1b400; }
    .black-keys {
      position: absolute; left: 12px; right: 12px; top: 26px; height: 58%;
      pointer-events: none;
    }
    .black-key {
      position: absolute;
      width: calc(var(--white-width) * 0.52);
      height: 100%;
      background: #111;
      border: 1px solid #333;
      border-bottom-width: 8px;
      border-radius: 0 0 6px 6px;
      pointer-events: auto;
      z-index: 2;
    }
    .black-key.active { background: #ffb000; border-color: #a96a00; }
    .white-key .label {
      position: absolute; bottom: 8px; left: 50%; transform: translateX(-50%);
      font-size: 11px; color: #666;
    }
    select, input[type="range"] {
      background: #0e0e0e; color: #eee;
      border: 1px solid #333; border-radius: 6px; padding: 4px 6px;
    }
    input[type="range"] { height: 6px; vertical-align: middle; }
  </style>
</head>
<body>

  <!-- Slides -->
  <div class="reveal">
    <div class="slides">
      <section>
        <h1>MIDI-Aware Slides</h1>
        <p>Persistent, playable piano keyboard docked at the bottom.</p>
        <p style="font-size:0.85em; opacity:0.85">Connect a MIDI keyboard or click/touch the
keys.</p>
      </section>

      <section>
        <h2>What this deck gives you</h2>
        <ul>
          <li>Reveal.js navigation, themes, speaker notes</li>
          <li>Browser MIDI input (lights keys)</li>
          <li>Simple built-in pluck synth (Web Audio)</li>
          <li>Keyboard appears on every slide</li>
        </ul>
      </section>

      <section>
        <h2>Images work as usual</h2>
        <img src="assets/piano.jpg" alt="Piano" style="border-radius:12px; max-width:100%;
height:auto;"/>
        <p class="fragment">And the keyboard stays put.</p>
      </section>

      <section>
        <h2>Two columns</h2>
        <div style="display:grid; grid-template-columns: 1fr 1fr; gap:24px;">
          <div>
            <h3>Left</h3>
            <ul>
              <li>Slides are <code>&lt;section&gt;</code> blocks</li>
              <li>Fragments via <code>.fragment</code></li>
              <li>Notes via <code>&lt;aside class="notes"&gt;</code></li>
            </ul>
          </div>
          <div>
            <h3>Right</h3>
            <p>Put text, lists, images, or iframes here.</p>
          </div>
        </div>
      </section>

      <section>
        <section><h2>Vertical Slides</h2><p>Press <kbd>↓</kbd></p></section>
        <section><h3>Slide 2</h3><p>Still got the keyboard.</p></section>
        <section><h3>Slide 3</h3><p>Try playing notes.</p></section>
      </section>

      <section>
        <h2>That’s it!</h2>
        <p>Duplicate this file and edit slides as normal.</p>
      </section>
    </div>
  </div>

  <!-- Persistent Keyboard Dock -->
  <div id="keyboard-dock">
    <div id="kbd-topbar">
      <div class="pill">
        <label for="midiIn">MIDI In:</label>
        <select id="midiIn"></select>
      </div>
      <div class="pill">
        <label for="volume">Volume</label>
        <input id="volume" type="range" min="0" max="1" step="0.01" value="0.6" />
      </div>
      <div class="pill">
        <label for="octave">Octave</label>
        <input id="octave" type="range" min="-2" max="2" step="1" value="0" />
        <span id="octaveLabel">0</span>
      </div>
      <div class="pill">
        <label for="scale">Size</label>
        <input id="scale" type="range" min="0.8" max="1.8" step="0.05" value="1" />
        <span id="scaleLabel">100%</span>
      </div>
      <div id="kbd-status">MIDI: <span id="midiStatus">initializing…</span></div>
    </div>
    <div id="keyboard">
      <div class="white-keys" id="whiteKeys"></div>
      <div class="black-keys" id="blackKeys"></div>
    </div>
  </div>

  <!-- Reveal.js -->
  <script defer src="https://cdn.jsdelivr.net/npm/reveal.js@4/dist/reveal.js"></script>

  <!-- Keyboard + MIDI -->
  <script defer>
  document.addEventListener('DOMContentLoaded', () => {
    Reveal.initialize({ hash: true, slideNumber: true, transition: 'slide' });

    /* config */
    const BASE_OCTAVE=3, NUM_OCTAVES=3, START_MIDI=12*BASE_OCTAVE, NOTE_RANGE=12*NUM_OCTAVES;
    const midiToFreq=n=>440*Math.pow(2,(n-69)/12);
    const NAMES=['C','C♯','D','D♯','E','F','F♯','G','G♯','A','A♯','B'];
    const isBlack=s=>[1,3,6,8,10].includes(s);

    /* synth */
    const audioCtx=new (window.AudioContext||window.webkitAudioContext)();
    const master=audioCtx.createGain(); master.gain.value=0.6;
master.connect(audioCtx.destination);
    const active=new Map();
    function on(midi,v=100){
      if(active.has(midi)) off(midi);
      const t=audioCtx.currentTime,f=midiToFreq(midi),vel=Math.max(0.05,v/127);
      const o1=audioCtx.createOscillator(),o2=audioCtx.createOscillator();
      o1.type='triangle'; o2.type='square'; o1.frequency.value=f; o2.frequency.value=f*1.005;
      const filt=audioCtx.createBiquadFilter(); filt.type='lowpass';
filt.frequency.value=1800+vel*4000;
      const g=audioCtx.createGain(); const a=0.002,d=0.2,s=0.12;
      g.gain.setValueAtTime(0,t); g.gain.linearRampToValueAtTime(vel,t+a);
      g.gain.exponentialRampToValueAtTime(Math.max(0.0005,s*vel),t+a+d);
      o1.connect(filt); o2.connect(filt); filt.connect(g); g.connect(master);
      o1.start(t); o2.start(t); active.set(midi,{o1,o2,filt,g});
    }
    function off(midi){
      const v=active.get(midi); if(!v)return;
      const t=audioCtx.currentTime,r=0.12;
      v.g.gain.setTargetAtTime(0.0001,t,r/3);
      v.o1.stop(t+r+0.05); v.o2.stop(t+r+0.05);
      active.delete(midi);
    }

    /* UI */
    const whiteKeys=document.getElementById('whiteKeys'),
blackKeys=document.getElementById('blackKeys');
    const midiStatus=document.getElementById('midiStatus'),
vol=document.getElementById('volume');
    const midiSel=document.getElementById('midiIn'), oct=document.getElementById('octave');
    const octLbl=document.getElementById('octaveLabel'), scl=document.getElementById('scale'),
sclLbl=document.getElementById('scaleLabel');
    const keyEls=new Map(), whiteCenters=new Map(); let whiteW=24;

    function render(){
      whiteKeys.innerHTML=''; blackKeys.innerHTML=''; keyEls.clear(); whiteCenters.clear();
      let nWhite=0; for(let i=0;i<NOTE_RANGE;i++) if(!isBlack(i%12)) nWhite++;
      const contW=whiteKeys.clientWidth||1024; whiteW=Math.max(18,Math.floor(contW/nWhite));
      document.documentElement.style.setProperty('--white-width',whiteW+'px');
      let idx=0;
      for(let i=0;i<NOTE_RANGE;i++){
        const m=START_MIDI+i,s=m%12; if(isBlack(s)) continue;
        const w=document.createElement('div'); w.className='white-key'; w.dataset.midi=m;
        if(s===0){const lab=document.createElement('div'); lab.className='label';
lab.textContent=NAMES[s]+(Math.floor(m/12)-1); w.appendChild(lab);}
        whiteKeys.appendChild(w); keyEls.set(m,w);
        const left=12+idx*whiteW,center=left+whiteW/2; whiteCenters.set(m,center); idx++;
      }
      for(let i=0;i<NOTE_RANGE;i++){
        const m=START_MIDI+i,s=m%12; if(!isBlack(s)) continue;
        const blk=document.createElement('div'); blk.className='black-key'; blk.dataset.midi=m;
        const
prev=findWhite(m,-1),next=findWhite(m,1),pc=whiteCenters.get(prev),nc=whiteCenters.get(next)||pc
+whiteW;
        const c=(pc+nc)/2,bw=whiteW*0.52; blk.style.left=(c-bw/2)+'px';
        blackKeys.appendChild(blk); keyEls.set(m,blk);
      }
      function down(e){
        const el=e.target.closest('.white-key,.black-key'); if(!el)return; e.preventDefault();
audioCtx.resume();
        const base=Number(el.dataset.midi),m=base+12*Number(oct.value); setAct(el,true);
on(m,110);
        let cur=el,cm=m;
        function mv(ev){const
over=document.elementFromPoint(ev.clientX||ev.touches[0].clientX,ev.clientY||ev.touches[0].clien
tY)?.closest('.white-key,.black-key');
          if(over&&over!==cur){setAct(cur,false); off(cm); cur=over;
cm=Number(cur.dataset.midi)+12*Number(oct.value); setAct(cur,true); on(cm,110);}}
        function up(){setAct(cur,false); off(cm); window.removeEventListener('mousemove',mv);
window.removeEventListener('mouseup',up);}
        window.addEventListener('mousemove',mv); window.addEventListener('mouseup',up,
{once:true});
      }
      whiteKeys.addEventListener('mousedown',down);
blackKeys.addEventListener('mousedown',down);
    }
    function findWhite(m,dir){let k=m+dir; while(k>=START_MIDI&&k<START_MIDI+NOTE_RANGE)
{if(!isBlack(k%12))return k;k+=dir;}return null;}
    function setAct(el,on){if(el)el.classList.toggle('active',on);}

    /* MIDI */
    let midiAcc=null,inpt=null;
    async function initMIDI(){if(!navigator.requestMIDIAccess){midiStatus.textContent='not
supported';return;}
      midiAcc=await navigator.requestMIDIAccess(); midiAcc.onstatechange=populate; populate();}
    function populate(){midiSel.innerHTML=''; const ins=[...midiAcc.inputs.values()];
if(!ins.length){midiStatus.textContent='no devices';return;}
      ins.forEach(i=>{const
o=document.createElement('option');o.value=i.id;o.textContent=i.name;midiSel.appendChild(o);});
      attach(ins[0].id);}
    function attach(id){if(inpt)inpt.onmidimessage=null; inpt=midiAcc.inputs.get(id);
if(inpt)inpt.onmidimessage=msg;}
    function msg(ev){const [st,d1,d2]=ev.data,cmd=st&0xf0; if(cmd===0x90&&d2>0){on(d1,d2);} else
if(cmd===0x80||(cmd===0x90&&d2===0)){off(d1);} }

    /* controls */
    vol.oninput=e=>master.gain.value=+e.target.value;
    oct.oninput=e=>octLbl.textContent=e.target.value;
    scl.oninput=e=>{const base=240,h=Math.round(base*+e.target.value);
document.documentElement.style.setProperty('--dock-height',h+'px');
      document.querySelector('.reveal').style.paddingBottom=(h+20)+'px';
sclLbl.textContent=Math.round(+e.target.value*100)+'%'; render();};

    render(); initMIDI(); scl.dispatchEvent(new Event('input'));
    window.addEventListener('pointerdown',()=>audioCtx.resume(),{once:true});
  });
  </script>
</body>
</html>
```

👉 Drop an image file at `assets/piano.jpg`, then run a local server (`python3 -m http.server`) and visit `http://localhost:8000/midi-slides.html`.

Do you want me to also add **same-note polyphony** (so multiple voices of the same MIDI note can overlap), or is retrigger-safe good enough?