===== src/lib/buildsStore.ts =====

```
export type BuildStatus = "draft" | "submitted" | "reviewing" |
"quote_sent" | "approved" | "in_build" | "complete";

export type BuildCustomer = {
  name: string;
  phone: string;
  email: string;
  address?: string;
};

export type BuildDims = {
  lengthIn: number;
  widthIn: number;
  heightIn: number;
  topThicknessIn?: number;
};

export type BuildOptions = {
  woodSpecies: "Pine" | "Oak" | "Walnut" | "Maple" | "Poplar" |
"Plywood";
  finish: "Natural" | "Stain" | "Paint" | "Poly";
  joinery: "Screws" | "Pocket Holes" | "Mortise & Tenon" |
"Dowels";
};

export type RenderStatus = "queued" | "rendering" | "complete" |
"failed";

export type RenderJob = {
  renderId: string;
  view: "iso" | "front" | "top" | "detail";
  status: RenderStatus;
  imageDataUrl?: string;
  startedAt?: string;
  finishedAt?: string;
  estimatePublic?: {
    total: number;
    rangeLow?: number;
    rangeHigh?: number;
    label?: string;
  };
};

export type NoteAuthor = "customer" | "admin";
export type NoteKind = "initial" | "change" | "refinement" |
"note";

export type NoteItem = {
  noteId: string;
  createdAt: string;
  author: NoteAuthor;
  kind: NoteKind;
  text: string;
};

export type BuildVersion = {
  versionId: string;
  createdAt: string;
  customerChangeRequest?: string;
  inputsSnapshot: {
    type: string;
    dims: BuildDims;
    options: BuildOptions;

    // Legacy field kept for backwards compatibility ONLY.
    // Going forward: we keep this EMPTY to prevent duplicate/
phantom notes.
    notes?: string;

    // Canonical notes source of truth
    notesLog?: NoteItem[];
  };
  renders: RenderJob[];

  estimatePublic?: {
    total: number;
    rangeLow?: number;
    rangeHigh?: number;
    materials: number;
    labor: number;
    overhead: number;
    finish: number;
  };

  estimateInternal?: any;
  generatedPackage?: any;
};

export type BuildSubmission = {
  id: string;
  createdAt: string;
  updatedAt: string;

  status: BuildStatus;
  accessCode?: string;

  customer: BuildCustomer;

  project: {
    type: string;
    dims: BuildDims;
    options: BuildOptions;

    // Legacy compiled string (we keep it EMPTY going forward)
    notes?: string;

    // Canonical structured notes
    notesLog?: NoteItem[];

    refPhotos?: { name: string; type: string; dataUrl: string }[];
  };

  versions: BuildVersion[];
};

const KEY = "rv_build_submissions";

function safeParse<T>(raw: string | null, fallback: T): T {
  try {
    if (!raw) return fallback;
    return JSON.parse(raw) as T;
  } catch {
    return fallback;
  }
}

function uid() {
  return (crypto as any).randomUUID?.() ??
(Math.random().toString(16).slice(2) + Date.now().toString(16));
}

/**
 * If an old build exists with only legacy `notes` and no
`notesLog`,
 * migrate it into a single structured "initial" note.
 */
function ensureNotesLog(legacyNotes?: string, existing?:
NoteItem[], createdAtHint?: string): NoteItem[] {
  const log = Array.isArray(existing) ? existing.filter(Boolean) :
[];
  if (log.length) return log;

  const text = String(legacyNotes || "").trim();
  if (!text) return [];

  return [
    {
      noteId: uid(),
      createdAt: createdAtHint || new Date().toISOString(),
      author: "customer",
      kind: "initial",
      text,
    },
  ];
}

function normalizeBuild(b: any): BuildSubmission | null {
  if (!b || typeof b !== "object") return null;

  const createdAt = String(b.createdAt || new
Date().toISOString());
```

```
  const project = b.project || {};

  const legacyProjectNotes = String(project.notes || "").trim();
  const projectNotesLog = ensureNotesLog(legacyProjectNotes, project.notesLog, createdAt);

  const versions = Array.isArray(b.versions) ? b.versions : [];
  const nextVersions: BuildVersion[] = versions.map((v: any) => {
    const inputs = v?.inputsSnapshot || {};
    const vCreatedAt = String(v?.createdAt || createdAt);

    // Old builds may have inputsSnapshot.notes (compiled blob) but no notesLog.
    const legacyVNotes = String(inputs.notes ?? legacyProjectNotes ?? "").trim();
    const vNotesLog = ensureNotesLog(legacyVNotes, inputs.notesLog ?? projectNotesLog, vCreatedAt);

    return {
      ...v,
      createdAt: vCreatedAt,
      inputsSnapshot: {
        type: String(inputs.type || project.type || ""),
        dims: inputs.dims || project.dims,
        options: inputs.options || project.options,

        // IMPORTANT: going forward we keep legacy notes empty to avoid duplication.
        // We only keep it for old builds that still have it.
        notes: String(inputs.notes || "").includes("\n---\n") ? "" : "",

        notesLog: vNotesLog,
      },
      renders: Array.isArray(v?.renders) ? v.renders : [],
    } as BuildVersion;
  });

  const out: BuildSubmission = {
    ...b,
    createdAt,
    updatedAt: String(b.updatedAt || createdAt),
    status: (b.status || "draft") as BuildStatus,
    customer: b.customer || { name: "", phone: "", email: "" },
    project: {
      ...project,
      type: String(project.type || ""),
      dims: project.dims,
      options: project.options,

      // IMPORTANT: going forward we keep legacy notes empty (canonical is notesLog)
      notes: "",

      notesLog: projectNotesLog,
      refPhotos: Array.isArray(project.refPhotos) ? project.refPhotos : [],
    },
    versions: nextVersions,
  };

  return out;
}

/**
 * Compile notes for display + renderer.
 * We do NOT prepend legacy notes anymore (we keep legacy notes empty going forward).
 * This prevents "remove last note" from appearing to do nothing.
 */
export function compileNotes(notesLog?: NoteItem[], _legacyNotes?: string) {
  const items = Array.isArray(notesLog) ? notesLog : [];
  return items
    .map((n) => String(n?.text || "").trim())
    .filter(Boolean)
    .join("\n\n---\n\n");
}

export function normalizePhone(p: string) {
  return String(p || "").replace(/\D+/g, "");
}

export function makeAccessCode() {
  return String(Math.floor(100000 + Math.random() * 900000));
}

export function readBuilds(): BuildSubmission[] {
  const arr = safeParse<any[]>(localStorage.getItem(KEY), []);
  const raw = (Array.isArray(arr) ? arr : []).filter(Boolean);
  return raw.map(normalizeBuild).filter(Boolean) as BuildSubmission[];
}

export function writeBuilds(items: BuildSubmission[]) {
  localStorage.setItem(KEY, JSON.stringify(items));
}

export function getBuild(id: string): BuildSubmission | null {
  const all = readBuilds();
  const found = all.find((b) => String(b.id) === String(id));
  return found || null;
}

export function upsertBuild(next: BuildSubmission) {
  const all = readBuilds();
  const idx = all.findIndex((b) => String(b.id) === String(next.id));
  if (idx >= 0) all[idx] = next;
  else all.unshift(next);
  writeBuilds(all);
}

export function deleteBuild(id: string) {
  const all = readBuilds().filter((b) => String(b.id) !== String(id));
  writeBuilds(all);
}

export function createDraftBuild(args: {
  customer: BuildCustomer;
  type: string;
  dims: BuildDims;
  options: BuildOptions;
  notes?: string;
}): BuildSubmission {
  const now = new Date().toISOString();
  const id = uid();

  const baseNotes = String(args.notes || "").trim();

  // canonical notesLog (and legacy notes stays empty)
  const notesLog: NoteItem[] = baseNotes
    ? [
        {
          noteId: uid(),
          createdAt: now,
          author: "customer",
          kind: "initial",
          text: baseNotes,
        },
      ]
    : [];

  const version: BuildVersion = {
    versionId: uid(),
    createdAt: now,
    inputsSnapshot: {
      type: args.type,
      dims: args.dims,
      options: args.options,
      notes: "",
      notesLog,
    },
    renders: [
      { renderId: uid(), view: "iso", status: "queued" },
      { renderId: uid(), view: "front", status: "queued" },
      { renderId: uid(), view: "top", status: "queued" },
    ],
  };

  const build: BuildSubmission = {
    id,
    createdAt: now,
    updatedAt: now,
      status: "draft",
      customer: args.customer,
      project: {
        type: args.type,
        dims: args.dims,
        options: args.options,
        notes: "",
        notesLog,
        refPhotos: [],
      },
      versions: [version],
    };

    upsertBuild(build);
    return build;
  }

  export function addRevision(
    id: string,
    customerChangeRequest: string,
    patch?: Partial<BuildSubmission["project"]>
  ) {
    const b0 = getBuild(id);
    if (!b0) return null;

    const b = normalizeBuild(b0);
    if (!b) return null;

    const now = new Date().toISOString();
    const mergedProject: BuildSubmission["project"] =
{ ...b.project, ...(patch || {}) };

    const mergedNotesLog = ensureNotesLog("",
mergedProject.notesLog, b.createdAt);

    const version: BuildVersion = {
      versionId: uid(),
      createdAt: now,
      customerChangeRequest,
      inputsSnapshot: {
        type: mergedProject.type,
        dims: mergedProject.dims,
        options: mergedProject.options,
        notes: "",
        notesLog: mergedNotesLog,
      },
      renders: [
        { renderId: uid(), view: "iso", status: "queued" },
        { renderId: uid(), view: "front", status: "queued" },
        { renderId: uid(), view: "top", status: "queued" },
        { renderId: uid(), view: "detail", status: "queued" },
      ],
    };

    const next: BuildSubmission = {
```

```
    ...b,
    updatedAt: now,
    project: { ...mergedProject, notes: "", notesLog: mergedNotesLog },
    versions: [version, ...b.versions],
  };

  upsertBuild(next);
  return next;
}

export function addCustomerNote(id: string, changeRequest?: string, extraNotes?: string) {
  const b0 = getBuild(id);
  if (!b0) return null;

  const b = normalizeBuild(b0);
  if (!b) return null;

  const now = new Date().toISOString();
  const prevLog = ensureNotesLog("", b.project.notesLog, b.createdAt);

  const nextLog: NoteItem[] = [...prevLog];

  const req = String(changeRequest || "").trim();
  const add = String(extraNotes || "").trim();

  if (req) {
    nextLog.push({
      noteId: uid(),
      createdAt: now,
      author: "customer",
      kind: "change",
      text: req,
    });
  }

  if (add) {
    nextLog.push({
      noteId: uid(),
      createdAt: now,
      author: "customer",
      kind: "refinement",
      text: add,
    });
  }

  return addRevision(id, "Customer provided additional details", {
    notesLog: nextLog,
    notes: "",
  });
}

export function removeLastCustomerNote(id: string) {
  const b0 = getBuild(id);
  if (!b0) return null;

  const b = normalizeBuild(b0);
  if (!b) return null;

  const prevLog = ensureNotesLog("", b.project.notesLog, b.createdAt);
  if (!prevLog.length) return b;

  let idx = -1;
  for (let i = prevLog.length - 1; i >= 0; i--) {
    if (prevLog[i]?.author === "customer") {
      idx = i;
      break;
    }
  }
  if (idx < 0) return b;

  const nextLog = prevLog.slice(0, idx).concat(prevLog.slice(idx + 1));

  return addRevision(id, "Customer removed last note", {
    notesLog: nextLog,
    notes: "",
  });
}

export function markSubmitted(id: string) {
  const b = getBuild(id);
  if (!b) return null;

  const now = new Date().toISOString();
  const accessCode = b.accessCode && String(b.accessCode).trim().length >= 6 ? b.accessCode : makeAccessCode();

  const next: BuildSubmission = {
    ...b,
    updatedAt: now,
    status: b.status === "draft" ? "submitted" : b.status,
    accessCode,
  };

  upsertBuild(next);
  return next;
}

export function findBuildsByPhoneAndCode(phone: string, code: string) {
  const p = normalizePhone(phone);
  const c = String(code || "").replace(/\D+/g, "");
  if (!p || c.length < 6) return [];
  return readBuilds().filter((b) => normalizePhone(b.customer?.phone || "") === p && String(b.accessCode || "") === c);
```

```
}

export function findBuildsByNameAndPhone(name: string, phone: string) {
  const n = String(name || "").trim().toLowerCase();
  const p = normalizePhone(phone);
  if (!n || p.length < 7) return [];
  return readBuilds().filter((b) => {
    const bn = String(b.customer?.name || "").trim().toLowerCase();
    const bp = normalizePhone(b.customer?.phone || "");
    return bn.includes(n) && bp.endsWith(p.slice(-7));
  });
}
```

===== src/pages/BuildDesigner.tsx =====

```
import { useMemo, useState } from "react";
import { createDraftBuild, type BuildOptions, type BuildDims } from "../lib/buildsStore";

function num(v: any, fallback: number) {
  const n = Number(v);
  return Number.isFinite(n) ? n : fallback;
}

export default function BuildDesigner() {
  const [customerName, setCustomerName] = useState("");
  const [customerPhone, setCustomerPhone] = useState("");
  const [customerEmail, setCustomerEmail] = useState("");
  const [customerAddress, setCustomerAddress] = useState("");

  const [type, setType] = useState("Table");
  const [lengthIn, setLengthIn] = useState(60);
  const [widthIn, setWidthIn] = useState(30);
  const [heightIn, setHeightIn] = useState(30);
  const [topThicknessIn, setTopThicknessIn] = useState(1.5);

  const [woodSpecies, setWoodSpecies] = useState<BuildOptions["woodSpecies"]>("Pine");
  const [finish, setFinish] = useState<BuildOptions["finish"]>("Natural");
  const [joinery, setJoinery] = useState<BuildOptions["joinery"]>("Pocket Holes");

  const [notes, setNotes] = useState("");

  const dims: BuildDims = useMemo(
    () => ({
      lengthIn: Math.max(12, num(lengthIn, 60)),
      widthIn: Math.max(10, num(widthIn, 30)),
      heightIn: Math.max(10, num(heightIn, 30)),
      topThicknessIn: Math.max(0.5, num(topThicknessIn, 1.5)),
    }),
    [lengthIn, widthIn, heightIn, topThicknessIn]
  );

  const options: BuildOptions = useMemo(
    () => ({ woodSpecies, finish, joinery }),
    [woodSpecies, finish, joinery]
  );

  function onStart() {
    if (!customerName.trim() || !customerPhone.trim() || !customerEmail.trim()) {
      alert("Please enter name, phone, and email.");
      return;
    }

    const draft = createDraftBuild({
      customer: {
        name: customerName.trim(),
        phone: customerPhone.trim(),
        email: customerEmail.trim(),
        address: customerAddress.trim() || "",
      },
      type,
      dims,
      options,
      notes,
    });

    window.location.href = `/builds/${draft.id}`;
  }

  return (
    <div className="stack page" style={{ gap: 16 }}>
      <section className="panel card card-center" style={{ maxWidth: 980, margin: "0 auto", padding: 18 }}>
        <h1 className="h2" style={{ margin: 0, fontWeight: 950 }}>Start a Custom Build</h1>
        <p className="lead" style={{ maxWidth: 820 }}>
          Fill in your project details. On the next screen you’ll see 3D render previews + estimate boxes update as each view completes.
        </p>

        <div className="panel" style={{ padding: 14, borderRadius: 14, width: "100%", maxWidth: 860 }}>
          <div style={{ fontWeight: 950, color: "#0f172a" }}>Contact Info (required)</div>
          <div style={{ display: "grid", gap: 10, marginTop: 10 }}>
            <input className="field" placeholder="Full Name" value={customerName} onChange={(e) => setCustomerName(e.target.value)} />
            <input className="field" placeholder="Phone Number" value={customerPhone} onChange={(e) => setCustomerPhone(e.target.value)} />
            <input className="field" placeholder="Email" value={customerEmail} onChange={(e) => setCustomerEmail(e.target.value)} /
```

```
>
            <input className="field" placeholder="Address (optional)" value={customerAddress} onChange={(e) => setCustomerAddress(e.target.value)} />
          </div>
        </div>

        <div className="panel" style={{ padding: 14, borderRadius: 14, width: "100%", maxWidth: 860 }}>
          <div style={{ fontWeight: 950, color: "#0f172a" }}>Project Basics</div>

          <div style={{ display: "grid", gap: 10, marginTop: 10 }}>
            <label style={{ display: "grid", gap: 6 }}>
              <span className="label">Project Type</span>
              <select className="field" value={type} onChange={(e) => setType(e.target.value)}>
                <option>Table</option>
                <option>Bench</option>
                <option>Shelf</option>
                <option>Cabinet</option>
                <option>Planter Box</option>
                <option>Workbench</option>
              </select>
            </label>

            <div className="row" style={{ gap: 10, flexWrap: "wrap" }}>
              <label style={{ display: "grid", gap: 6, flex: 1, minWidth: 160 }}>
                <span className="label">Length (in)</span>
                <input className="field" inputMode="numeric" value={lengthIn} onChange={(e) => setLengthIn(Number(e.target.value))} />
              </label>
              <label style={{ display: "grid", gap: 6, flex: 1, minWidth: 160 }}>
                <span className="label">Width (in)</span>
                <input className="field" inputMode="numeric" value={widthIn} onChange={(e) => setWidthIn(Number(e.target.value))} />
              </label>
              <label style={{ display: "grid", gap: 6, flex: 1, minWidth: 160 }}>
                <span className="label">Height (in)</span>
                <input className="field" inputMode="numeric" value={heightIn} onChange={(e) => setHeightIn(Number(e.target.value))} />
              </label>
              <label style={{ display: "grid", gap: 6, flex: 1, minWidth: 160 }}>
                <span className="label">Top thickness (in)</span>
                <input className="field" inputMode="decimal" value={topThicknessIn} onChange={(e) => setTopThicknessIn(Number(e.target.value))} />
              </label>
            </div>

            <div className="row" style={{ gap: 10, flexWrap: "wrap" }}>
              <label style={{ display: "grid", gap: 6, flex: 1, minWidth: 220 }}>
                <span className="label">Wood Species</span>
                <select className="field" value={woodSpecies} onChange={(e) => setWoodSpecies(e.target.value as any)}>
                  <option value="Pine">Pine</option>
                  <option value="Poplar">Poplar</option>
                  <option value="Plywood">Plywood</option>
                  <option value="Oak">Oak</option>
                  <option value="Maple">Maple</option>
                  <option value="Walnut">Walnut</option>
                </select>
              </label>

              <label style={{ display: "grid", gap: 6, flex: 1, minWidth: 220 }}>
                <span className="label">Finish</span>
                <select className="field" value={finish} onChange={(e) => setFinish(e.target.value as any)}>
                  <option value="Natural">Natural</option>
                  <option value="Stain">Stain</option>
                  <option value="Paint">Paint</option>
                  <option value="Poly">Poly</option>
                </select>
              </label>

              <label style={{ display: "grid", gap: 6, flex: 1, minWidth: 220 }}>
                <span className="label">Joinery Preference</span>
                <select className="field" value={joinery} onChange={(e) => setJoinery(e.target.value as any)}>
                  <option value="Screws">Screws</option>
                  <option value="Pocket Holes">Pocket Holes</option>
                  <option value="Dowels">Dowels</option>
                  <option value="Mortise & Tenon">Mortise & Tenon</option>
                </select>
              </label>
            </div>

            <label style={{ display: "grid", gap: 6 }}>
              <span className="label">Notes / Special Requests</span>
              <textarea className="field" rows={4} value={notes} onChange={(e) => setNotes(e.target.value)} placeholder="Example: tapered legs, lower shelf, rounded corners, hidden fasteners, etc." />
            </label>

            <button className="btn btn-primary" onClick={onStart} style={{ fontWeight: 950 }}>
              Generate Render Previews →
            </button>

            <div className="muted" style={{ fontWeight: 850 }}>
              You’ll receive an Access Code after submission to view your build in the Build Portal.
            </div>
          </div>
        </div>
      </section>
    </div>
  );
}
```

===== src/pages/BuildPreview.tsx =====

```
import { useEffect, useMemo, useState } from "react";
import { useParams, Link } from "react-router-dom";
import { estimateBuild } from "../lib/buildPricing";
import { renderBuildPreviewPng } from "../lib/render3d";
import {
  addCustomerNote,
  compileNotes,
  getBuild,
  markSubmitted,
  removeLastCustomerNote,
  type BuildSubmission,
  type RenderJob,
  upsertBuild,
} from "../lib/buildsStore";

function fmt(iso: string) {
  const d = new Date(iso);
  if (Number.isNaN(d.getTime())) return iso;
  return d.toLocaleString();
}

function money(n: number) {
  return new Intl.NumberFormat("en-US", { style: "currency", currency: "USD" }).format(n);
}

export default function BuildPreview() {
  const { id } = useParams();
  const [build, setBuild] = useState<BuildSubmission | null>(null);

  // Customer refinement fields
  const [changeRequest, setChangeRequest] = useState("");
  const [extraNotes, setExtraNotes] = useState("");

  useEffect(() => {
    if (!id) return;
    setBuild(getBuild(id));
  }, [id]);

  const version = useMemo(() => build?.versions?.[0] ?? null, [build]);

  // Render queue: ONE image at a time
  useEffect(() => {
    if (!build || !version) return;

    const latest0 = getBuild(build.id) ?? build;
    const lv0 = latest0.versions?.[0];
    if (!lv0) return;

    const renders0 = lv0.renders || [];

    const currentlyRendering = renders0.find((r) => r.status === "rendering") ?? null;
    let target: RenderJob | null = currentlyRendering;

    if (!target) {
      const firstQueued = renders0.find((r) => r.status === "queued") ?? null;

      if (firstQueued) {
        const startedAt = new Date().toISOString();
        const updatedRenders = renders0.map((r) =>
          r.renderId === firstQueued.renderId ? ({ ...r, status: "rendering" as const, startedAt } as RenderJob) : r
        );

        const nextV = { ...lv0, renders: updatedRenders };
        const nextBuild: BuildSubmission = {
          ...latest0,
          updatedAt: new Date().toISOString(),
          versions: [nextV, ...latest0.versions.slice(1)],
        };

        upsertBuild(nextBuild);
        setBuild(nextBuild);

        target = updatedRenders.find((r) => r.renderId === firstQueued.renderId) as RenderJob;
      }
    }

    if (!target) return;

    let cancelled = false;

    const run = async () => {
      try {
        await new Promise((res) => setTimeout(res, 450));
        if (cancelled) return;

        const latest = getBuild(build.id);
        if (!latest) return;

        const lv = latest.versions[0];
        const est = estimateBuild(lv.inputsSnapshot.dims, lv.inputsSnapshot.options);

        const title = `${lv.inputsSnapshot.type} • ${lv.inputsSnapshot.dims.lengthIn}"×${lv.inputsSnapshot.dims.widthIn}"×${lv.inputsSnapshot.dims.heightIn}"`;

        // Notes used for rendering (structured log + base notes)
        const notesCompiled = compileNotes((lv.inputsSnapshot as any).notesLog, lv.inputsSnapshot.notes);

        const png = await renderBuildPreviewPng({
          projectType: lv.inputsSnapshot.type,
          view: target!.view,
          title,
          notes: notesCompiled,
          dims: lv.inputsSnapshot.dims,
          options: lv.inputsSnapshot.options,
          width: 1200,
          height: 800,
        });

        if (cancelled) return;

        const updatedRenders = (lv.renders || []).map((x) => {
          if (x.renderId !== target!.renderId) return x;
          return {
            ...x,
            status: "complete" as const,
            finishedAt: new Date().toISOString(),
            imageDataUrl: png,
            estimatePublic: {
              total: est.total,
              rangeLow: est.rangeLow,
              rangeHigh: est.rangeHigh,
              label: "Est. total (updates per view)",
            },
          };
        });

        const nextV = {
          ...lv,
          renders: updatedRenders,
          estimatePublic: {
            total: est.total,
            rangeLow: est.rangeLow,
            rangeHigh: est.rangeHigh,
            materials: est.materials,
            labor: est.labor,
            overhead: est.overhead,
            finish: est.finish,
          },
        };

        const nextBuild: BuildSubmission = {
          ...latest,
          updatedAt: new Date().toISOString(),
          versions: [nextV, ...latest.versions.slice(1)],
        };

        upsertBuild(nextBuild);
        setBuild(nextBuild);
      } catch (e) {
        console.error(e);
        if (cancelled) return;

        const latest = getBuild(build.id);
        if (!latest) return;

        const lv = latest.versions[0];
        const updatedRenders = (lv.renders || []).map((x) => {
          if (x.renderId !== target!.renderId) return x;
          return { ...x, status: "failed" as const, finishedAt: new Date().toISOString() };
        });

        const nextV = { ...lv, renders: updatedRenders };
        const nextBuild: BuildSubmission = {
          ...latest,
          updatedAt: new Date().toISOString(),
          versions: [nextV, ...latest.versions.slice(1)],
        };

        upsertBuild(nextBuild);
        setBuild(nextBuild);
      }
    };

    run();

    return () => {
      cancelled = true;
    };
  }, [build?.id, build?.updatedAt, version?.versionId]);

  if (!build || !version) {
    return (
      <div className="panel card card-center" style={{ maxWidth: 900, margin: "0 auto" }}>
        <h3 className="h3">Build not found</h3>
        <Link className="btn btn-primary" to="/builds/new">
          Start a Build
        </Link>
      </div>
    );
  }

  const v = version;
  const b = build;

  const notesLog = ((v.inputsSnapshot as any).notesLog || []) as any[];
  const compiledNotes = compileNotes(notesLog, v.inputsSnapshot.notes);

  const canRemoveCustomerNote =
    Array.isArray(notesLog) &&
    notesLog.some((n) => String(n?.author || "").toLowerCase() === "customer");

  function submit() {
    const next = markSubmitted(b.id);
    if (!next) return alert("Could not submit. Try again.");
    setBuild(next);
    alert(`Submitted! Your Build Access Code: ${String(next.accessCode || "—")}`);
  }

  function submitRefinement() {
    const req = changeRequest.trim();
    const add = extraNotes.trim();

    if (!req && !add) {
      alert("Please add a change request and/or extra notes.");
      return;
    }

    const next = addCustomerNote(b.id, req, add);
    if (!next) {
      alert("Could not save changes. Please refresh and try again.");
      return;
    }

    setChangeRequest("");
    setExtraNotes("");
    setBuild(next);
    alert("Saved! We’re generating updated previews now.");
  }

  function removeLastCustomerNoteClick() {
    if (!confirm("Remove your most recent note and regenerate previews?")) return;

    const next = removeLastCustomerNote(b.id);
    if (!next) {
      alert("Could not remove the last note. Please refresh and try again.");
      return;
    }

    setBuild(next);
    alert("Removed! We’re generating updated previews now.");
  }

  const est = v.estimatePublic;

  return (
    <div className="stack page" style={{ gap: 16 }}>
      <section className="panel card card-center"
style={{ maxWidth: 1100, margin: "0 auto", padding: 18 }}>
        <div style={{ display: "grid", gap: 8 }}>
          <h1 className="h2" style={{ margin: 0, fontWeight:
950 }}>
            Build Preview
          </h1>
          <div className="muted" style={{ fontWeight: 850 }}>
            Created: {fmt(b.createdAt)} • Status: <span
className="badge">{String(b.status).toUpperCase()}</span>
          </div>
          <div className="muted" style={{ fontWeight: 850 }}>
            Customer: <strong>{b.customer?.name}</strong> •
{b.customer?.phone} • {b.customer?.email}
          </div>
        </div>

        <div className="panel" style={{ padding: 14, borderRadius:
14, marginTop: 12, width: "100%", maxWidth: 1000 }}>
          <div style={{ fontWeight: 950, color: "#0f172a" }}>
            {v.inputsSnapshot.type} —
{v.inputsSnapshot.dims.lengthIn}" × {v.inputsSnapshot.dims.widthIn}" ×{"
"}
            {v.inputsSnapshot.dims.heightIn}"
          </div>
          <div className="muted" style={{ fontWeight: 850,
marginTop: 6 }}>
            Wood: {v.inputsSnapshot.options.woodSpecies} • Finish:
{v.inputsSnapshot.options.finish} • Joinery:{" "}
            {v.inputsSnapshot.options.joinery}
          </div>

          {compiledNotes ? (
            <div className="panel" style={{ padding: 12,
borderRadius: 12, marginTop: 10 }}>
              <div className="label">Notes on file (used to
improve the model)</div>
              <div className="muted" style={{ fontWeight: 850,
whiteSpace: "pre-wrap", marginTop: 6 }}>
                {compiledNotes}
              </div>
            </div>
          ) : null}

          <div className="row" style={{ gap: 10, flexWrap: "wrap",
marginTop: 12 }}>
            <Link className="btn btn-ghost" to="/builds/new">
              Start Another
            </Link>
            <Link className="btn btn-ghost" to="/builds/portal">
              Build Portal
            </Link>
            <button className="btn btn-primary" onClick={submit} style={{ fontWeight: 950 }}>
              Submit for Review
            </button>
          </div>

          <div className="muted" style={{ fontWeight: 850, marginTop: 10 }}>
            After submitting you’ll get a 6-digit Access Code to view your build in the Build Portal.
          </div>
        </div>

        <div className="panel" style={{ padding: 14, borderRadius: 14, marginTop: 12, width: "100%", maxWidth: 1000 }}>
          <div style={{ fontWeight: 950, color: "#0f172a" }}>Refine this build (add details)</div>
          <div className="muted" style={{ fontWeight: 850, marginTop: 6 }}>
            Add specific details and we’ll regenerate previews. Examples: “tapered legs”, “lower shelf”, “drawer”, “apron”, “feet”, “2 shelves”.
          </div>

          <div style={{ display: "grid", gap: 10, marginTop: 10 }}>
            <label style={{ display: "grid", gap: 6 }}>
              <span className="label">What would you like changed?</span>
              <input
                className="field"
                value={changeRequest}
                onChange={(e) => setChangeRequest(e.target.value)}
                placeholder="Example: Add a lower shelf and tapered legs"
              />
            </label>

            <label style={{ display: "grid", gap: 6 }}>
              <span className="label">Extra notes (used by the renderer)</span>
              <textarea
                className="field"
                rows={4}
                value={extraNotes}
                onChange={(e) => setExtraNotes(e.target.value)}
                placeholder="Example: drawer centered on front, apron on all sides, rounded corners, etc."
              />
            </label>

            <div className="row" style={{ gap: 10, flexWrap:
```

```
"wrap", justifyContent: "center" }}>
                <button type="button" className="btn btn-primary" onClick={(e) => { e.preventDefault(); e.stopPropagation(); submitRefinement(); }} style={{ fontWeight: 950 }}>
                  Save refinement + regenerate previews →
                </button>

                {canRemoveCustomerNote ? (
                  <button type="button" className="btn btn-ghost" onClick={(e) => { e.preventDefault(); e.stopPropagation(); removeLastCustomerNoteClick(); }} style={{ fontWeight: 950 }}>
                    Remove my last note
                  </button>
                ) : null}
              </div>
            </div>

            {Array.isArray(notesLog) && notesLog.length ? (
              <div className="panel" style={{ padding: 12, borderRadius: 12, marginTop: 12 }}>
                <div className="label">Notes timeline</div>
                <div className="muted" style={{ fontWeight: 850, marginTop: 6 }}>
                  Notes are saved in separate chunks so Admin can remove any later if needed.
                </div>

                <div style={{ display: "grid", gap: 10, marginTop: 10 }}>
                  {notesLog.map((n) => (
                    <div key={String(n.noteId)} className="panel" style={{ padding: 10, borderRadius: 12 }}>
                      <div className="row" style={{ justifyContent: "space-between", gap: 10, flexWrap: "wrap" }}>
                        <div style={{ fontWeight: 950, color: "#0f172a" }}>
                          {String(n.kind || "NOTE").toUpperCase()} • {String(n.author || "UNKNOWN").toUpperCase()}
                          <span className="badge" style={{ marginLeft: 8 }}>
                            {fmt(String(n.createdAt))}
                          </span>
                        </div>
                        <div className="muted" style={{ fontWeight: 850 }}>
                          Note ID: <span className="badge">{String(n.noteId).slice(-8).toUpperCase()}</span>
                        </div>
                      </div>
                      <div className="muted" style={{ fontWeight: 850, whiteSpace: "pre-wrap", marginTop: 8 }}>
                        {String(n.text || "")}
                      </div>
                    </div>
                  ))}
                </div>
              </div>
            ) : null}
          </div>

          <div className="panel" style={{ padding: 14, borderRadius: 14, marginTop: 12, width: "100%", maxWidth: 1000 }}>
            <div style={{ fontWeight: 950, color: "#0f172a" }}>Estimate (public)</div>
            {!est ? (
              <div className="muted" style={{ fontWeight: 850, marginTop: 6 }}>Estimating… (will populate as render previews complete)</div>
            ) : (
              <div style={{ display: "grid", gap: 10, marginTop: 10 }}>
                <div className="row" style={{ gap: 10, flexWrap: "wrap" }}>
                  <span className="badge rate-bright">Estimated Total: {money(est.total)}</span>
                  {typeof est.rangeLow === "number" && typeof est.rangeHigh === "number" ? (
                    <span className="badge">Range: {money(est.rangeLow)} – {money(est.rangeHigh)}</span>
                  ) : null}
                </div>

                <div className="muted" style={{ fontWeight: 850 }}>
                  Breakdown (customer-safe): Materials {money(est.materials)} • Labor {money(est.labor)} • Finish {money(est.finish)} • Overhead {money(est.overhead)}
                </div>
              </div>
            )}
          </div>
        </section>

        <section className="stack" style={{ maxWidth: 1100, margin: "0 auto", width: "100%" }}>
          <div className="h3" style={{ margin: 0 }}>Render Previews</div>
          <div className="muted" style={{ fontWeight: 850 }}>
            One render runs at a time (queued → rendering → complete). Each render has its own estimate box.
          </div>

          <div style={{ display: "grid", gridTemplateColumns: "repeat(auto-fit, minmax(280px, 1fr))", gap: 12, marginTop: 10 }}>
            {(v.renders || []).map((r) => (
              <article key={r.renderId} className="panel card" style={{ padding: 12, display: "grid", gap: 10 }}>
                <div style={{ fontWeight: 950, color: "#0f172a" }}>
                  View: {String(r.view).toUpperCase()}
                  <span className="badge" style={{ marginLeft: 8 }}>{String(r.status).toUpperCase()}</span>
                </div>

                <div
                  style={{
                    width: "100%",
                    height: 180,
                    borderRadius: 14,
                    overflow: "hidden",
                    border: "1px solid rgba(15,23,42,0.18)",
                    background: "rgba(2,6,23,0.25)",
                  }}
                >
                  {r.imageDataUrl ? (
                    <img
                      src={r.imageDataUrl}
                      alt={`${r.view} render`}
                      style={{ width: "100%", height: "100%", objectFit: "cover", display: "block" }}
                    />
                  ) : (
                    <div className="card-center" style={{ width: "100%", height: "100%", padding: 12 }}>
                      <div className="muted" style={{ fontWeight: 900 }}>
                        {r.status === "queued"
                          ? "Queued…"
                          : r.status === "rendering"
                          ? "Rendering…"
                          : r.status === "failed"
                          ? "Render failed (will re-run on refresh later)"
                          : "No image"}
                      </div>
                    </div>
                  )}
                </div>

                <div className="panel" style={{ padding: 10, borderRadius: 12 }}>
                  <div className="label">Estimate for this render</div>
                  {!r.estimatePublic ? (
                    <div className="muted" style={{ fontWeight: 850, marginTop: 6 }}>
                      {r.status === "complete" ? "Finalizing estimate…" : "Waiting for render to complete…"}
                    </div>
                  ) : (
                    <div style={{ display: "grid", gap: 8, marginTop: 8 }}>
                      <div className="badge rate-bright" style={{ justifyContent: "center" }}>
                        {r.estimatePublic.label || "Estimated Total"}: {money(r.estimatePublic.total)}
                      </div>
                      {typeof r.estimatePublic.rangeLow === "number" && typeof r.estimatePublic.rangeHigh === "number" ? (
                        <div className="muted" style={{ fontWeight: 850 }}>
                          Range: {money(r.estimatePublic.rangeLow)} – {money(r.estimatePublic.rangeHigh)}
                        </div>
                      ) : null}
                    </div>
                  )}
                </div>

                <div className="muted" style={{ fontWeight: 850 }}>
                  {r.startedAt ? `Started: ${fmt(r.startedAt)}` : "Not started yet"}
                  {r.finishedAt ? ` • Finished: ${fmt(r.finishedAt)}` : ""}
                </div>
              </article>
            ))}
          </div>
        </section>
      </div>
    );
  }
```

===== src/lib/render3d.ts =====

```
import * as THREE from "three";
import type { BuildDims, BuildOptions } from "./buildsStore";

type View = "iso" | "front" | "top" | "detail";

function clamp(n: number, min: number, max: number) {
  return Math.max(min, Math.min(max, n));
}

function safeIn(n: number, fallback: number) {
  const v = Number(n);
  return Number.isFinite(v) && v > 0 ? v : fallback;
}

function woodColor(species: BuildOptions["woodSpecies"]) {
  // Simple readable tones (non-photoreal). Upgrade later with textures.
  switch (species) {
    case "Pine": return 0xE6D2A6;
    case "Poplar": return 0xD8E0A8;
    case "Plywood": return 0xD9C7A3;
    case "Oak": return 0xC8A06C;
    case "Maple": return 0xEAD9B6;
    case "Walnut": return 0x6B4A2E;
    default: return 0xC8A06C;
  }
```

```
}

function finishSheen(finish: BuildOptions["finish"]) {
  if (finish === "Natural") return { roughness: 0.65, metalness: 0.02 };
  if (finish === "Stain") return { roughness: 0.55, metalness: 0.03 };
  if (finish === "Paint") return { roughness: 0.35, metalness: 0.01 };
  if (finish === "Poly") return { roughness: 0.25, metalness: 0.04 };
  return { roughness: 0.55, metalness: 0.03 };
}

function fitToView(length: number, depth: number, height: number) {
  const maxDim = Math.max(length, depth, height);
  return clamp(maxDim * 0.95, 40, 320);
}

function makeCamera(view: View, frustumSize: number) {
  const cam = new THREE.OrthographicCamera(
    -frustumSize, frustumSize, frustumSize, -frustumSize,
    0.1, 4000
  );

  if (view === "top") {
    cam.position.set(0, 600, 0.001);
  } else if (view === "front") {
    cam.position.set(0, 220, 600);
  } else if (view === "detail") {
    cam.position.set(420, 280, 420);
  } else {
    cam.position.set(520, 360, 520); // iso
  }

  cam.lookAt(0, 0, 0);
  cam.updateProjectionMatrix();
  return cam;
}

function normType(t: string) {
  return String(t || "").trim().toLowerCase();
}

function normNotes(n: string | undefined) {
  return String(n || "").trim().toLowerCase();
}

function has(notes: string, ...phrases: string[]) {
  return phrases.some((p) => notes.includes(p));
}

function addBox(
  group: THREE.Group,
  w: number,
  h: number,
  d: number,
  x: number,
  y: number,
  z: number,
  mat: THREE.Material
) {
  const geom = new THREE.BoxGeometry(w, h, d);
  const mesh = new THREE.Mesh(geom, mat);
  mesh.position.set(x, y, z);
  group.add(mesh);
  return geom;
}

/**
 * Notes-driven features (simple heuristics, safe defaults):
 * - "lower shelf", "bottom shelf", "shelf" -> adds a shelf panel for table/bench/workbench
 * - "apron" -> adds apron rails (unless notes include "no apron")
 * - "drawer" -> adds a simple drawer box under top (front-facing)
 * - "taper" / "tapered legs" -> visually tapers legs using mesh scaling
 * - "feet" -> adds small feet blocks for planter/cabinet
 *
 * This is NOT photoreal; it's an improving proxy model based on customer intent.
 */
function buildModel(args: {
  projectType: string;
  dims: BuildDims;
  options: BuildOptions;
  notes?: string;
}) {
  const group = new THREE.Group();
  const notes = normNotes(args.notes);

  // Interpret dims consistently:
  // lengthIn => X (long)
  // widthIn  => Z (depth)
  // heightIn => Y (overall height)
  const length = clamp(safeIn(args.dims.lengthIn, 60), 12, 240);
  const depth  = clamp(safeIn(args.dims.widthIn, 30), 10, 240);
  const height = clamp(safeIn(args.dims.heightIn, 30), 10, 240);

  const topThickness = clamp(safeIn(args.dims.topThicknessIn ?? 1.5, 1.5), 0.5, 6);

  const sheen = finishSheen(args.options.finish);
  const woodMat = new THREE.MeshStandardMaterial({
    color: woodColor(args.options.woodSpecies),
    roughness: sheen.roughness,
    metalness: sheen.metalness,
  });

  const darkMat = new THREE.MeshStandardMaterial({
    color: 0x0f172a,
    roughness: 0.85,
    metalness: 0.05,
  });

  const t = normType(args.projectType);

  // Shared thickness defaults (inches-as-units)
  const boardT = clamp(Math.min(depth, length) * 0.035, 0.6, 1.25); // panel/board thickness
  const legSize = clamp(Math.min(depth, length) * 0.06, 1.5, 4.0);

  // Helpers
  const xMin = -length / 2;
  const xMax =  length / 2;
  const zMin = -depth  / 2;
  const zMax =  depth  / 2;

  const geoms: THREE.BufferGeometry[] = [];

  const isTable = t.includes("table");
  const isBench = t.includes("bench");
  const isWorkbench = t.includes("workbench");

  // --- TABLE / BENCH / WORKBENCH (top + 4 legs) ---
  if (isTable || isBench || isWorkbench) {
    const topY = height - topThickness / 2;

    // Top
    geoms.push(addBox(group, length, topThickness, depth, 0, topY, 0, woodMat));

    // Legs
    const legH = Math.max(2, height - topThickness);
    const inset = clamp(legSize * 0.65, 1.25, 4.5);

    const lx1 = xMin + inset + legSize / 2;
    const lx2 = xMax - inset - legSize / 2;
    const lz1 = zMin + inset + legSize / 2;
    const lz2 = zMax - inset - legSize / 2;

    const legY = legH / 2;

    const legGeom = new THREE.BoxGeometry(legSize, legH, legSize);

    function addLeg(x: number, z: number) {
      const mesh = new THREE.Mesh(legGeom, darkMat);
      mesh.position.set(x, legY, z);

      // Notes: tapered legs (visual taper using non-uniform scale)
      if (has(notes, "taper", "tapered leg", "tapered legs")) {
        // slightly narrower at the top by scaling X/Z a bit (simple proxy)
        // Using scale affects the whole mesh uniformly; we fake taper by scaling and adding a small "cap"
        mesh.scale.set(0.88, 1, 0.88);
        const cap = new THREE.Mesh(new THREE.BoxGeometry(legSize * 0.92, legSize * 0.18, legSize * 0.92), darkMat);
        cap.position.set(x, legH - (legSize * 0.09), z);
        group.add(cap);
        geoms.push(cap.geometry as THREE.BufferGeometry);
      }

      group.add(mesh);
    }

    addLeg(lx1, lz1);
    addLeg(lx2, lz1);
    addLeg(lx1, lz2);
    addLeg(lx2, lz2);

    geoms.push(legGeom);

    // Notes: apron rails (default on workbench, optional on table/bench)
    const wantsApron = isWorkbench || (has(notes, "apron") && !has(notes, "no apron", "noapron"));
    if (wantsApron) {
      const apronH = clamp(height * 0.12, 2, 6);
      const apronY = height - topThickness - apronH / 2;

      // Front/back rails (along X)
      geoms.push(addBox(group, length - inset * 2, apronH, boardT, 0, apronY, zMax - inset - boardT / 2, darkMat));
      geoms.push(addBox(group, length - inset * 2, apronH, boardT, 0, apronY, zMin + inset + boardT / 2, darkMat));

      // Left/right rails (along Z)
      geoms.push(addBox(group, boardT, apronH, depth - inset * 2, xMin + inset + boardT / 2, apronY, 0, darkMat));
      geoms.push(addBox(group, boardT, apronH, depth - inset * 2, xMax - inset - boardT / 2, apronY, 0, darkMat));
    }

    // Notes: lower shelf / bottom shelf
    const wantsShelf = isWorkbench || has(notes, "lower shelf", "bottom shelf") || (has(notes, "shelf") && !has(notes, "no shelf"));
    if (wantsShelf) {
      const shelfY = clamp(legH * 0.28, 6, legH - 8);
      const shelfT = clamp(boardT, 0.6, 1.5);
      geoms.push(addBox(group, length - inset * 2 - legSize * 0.2, shelfT, depth - inset * 2 - legSize * 0.2, 0, shelfY, 0, woodMat));
    }

    // Notes: drawer (simple centered drawer under top, front-facing)
    if (has(notes, "drawer", "drawers")) {
      const drawerH = clamp(height * 0.18, 3, 8);
```

```
        const drawerW = clamp(length * 0.35, 10, length - 10);
        const drawerD = clamp(depth * 0.45, 8, depth - 6);
        const drawerY = height - topThickness - drawerH / 2 - 1.2;
        const drawerZ = zMax - drawerD / 2 - 1.2;

        geoms.push(addBox(group, drawerW, drawerH, drawerD, 0, drawerY, drawerZ, darkMat));

        // drawer face
        const faceT = clamp(boardT * 0.8, 0.4, 1.1);
        geoms.push(addBox(group, drawerW * 0.96, drawerH * 0.92, faceT, 0, drawerY, zMax - 0.6, woodMat));
      }

      // Simple stretcher for workbench feel
      if (isWorkbench) {
        const stretcherH = clamp(legSize * 0.45, 0.8, 2.0);
        const stretcherY = clamp(legH * 0.35, 6, legH - 4);
        geoms.push(addBox(group, length - inset * 2 - legSize, stretcherH, legSize * 0.6, 0, stretcherY, 0, darkMat));
      }
    }

    // --- SHELF (two uprights + shelves) ---
    else if (t.includes("shelf")) {
      const sideT = boardT;
      const shelfT = boardT;

      // Uprights (left/right)
      const upH = height;
      const upY = upH / 2;
      const upX = length / 2 - sideT / 2;

      geoms.push(addBox(group, sideT, upH, depth, -upX, upY, 0, woodMat));
      geoms.push(addBox(group, sideT, upH, depth,  upX, upY, 0, woodMat));

      // Shelves: bottom, mid, top
      const insideL = Math.max(8, length - sideT * 2);
      const shelfCount = has(notes, "5 shelf", "five shelf") ? 5 : has(notes, "4 shelf", "four shelf") ? 4 : 3;

      for (let i = 0; i < shelfCount; i++) {
        const frac = (i + 1) / (shelfCount + 1);
        const y = clamp(frac * (height - shelfT) + shelfT / 2, shelfT / 2, height - shelfT / 2);
        geoms.push(addBox(group, insideL, shelfT, depth, 0, y, 0, woodMat));
      }
    }

    // --- CABINET (box with shelf + back) ---
    else if (t.includes("cabinet")) {
      const wallT = clamp(boardT, 0.6, 1.25);
      const shelfT = wallT;
      const insideL = Math.max(10, length - wallT * 2);
      const insideD = Math.max(8, depth - wallT * 2);

      geoms.push(addBox(group, length, wallT, depth, 0, wallT / 2, 0, woodMat));                         // bottom
      geoms.push(addBox(group, length, wallT, depth, 0, height - wallT / 2, 0, woodMat));              // top
      geoms.push(addBox(group, wallT, height, depth, -length / 2 + wallT / 2, height / 2, 0, woodMat)); // left
      geoms.push(addBox(group, wallT, height, depth,  length / 2 - wallT / 2, height / 2, 0, woodMat)); // right
      geoms.push(addBox(group, length, height, wallT, 0, height / 2, -depth / 2 + wallT / 2, woodMat)); // back

      // Shelf count from notes
      const shelves = has(notes, "2 shelf", "two shelf") ? 2 : has(notes, "3 shelf", "three shelf") ? 3 : 1;
      for (let i = 0; i < shelves; i++) {
        const frac = (i + 1) / (shelves + 1);
        const y = clamp(frac * (height - shelfT) + shelfT / 2, shelfT / 2, height - shelfT / 2);
        geoms.push(addBox(group, insideL, shelfT, insideD, 0, y, wallT / 2, woodMat));
      }

      // Notes: feet
      if (has(notes, "feet", "legs")) {
        const foot = clamp(wallT * 1.2, 0.8, 2.2);
        const fy = foot / 2;
        geoms.push(addBox(group, foot, foot, foot, xMin + foot, fy, zMin + foot, darkMat));
        geoms.push(addBox(group, foot, foot, foot, xMax - foot, fy, zMin + foot, darkMat));
        geoms.push(addBox(group, foot, foot, foot, xMin + foot, fy, zMax - foot, darkMat));
        geoms.push(addBox(group, foot, foot, foot, xMax - foot, fy, zMax - foot, darkMat));
      }
    }

    // --- PLANTER BOX (open top, 4 walls + bottom) ---
    else if (t.includes("planter")) {
      const wallT = clamp(boardT, 0.6, 1.5);
      const bottomT = wallT;

      geoms.push(addBox(group, length, bottomT, depth, 0, bottomT / 2, 0, woodMat)); // bottom

      const wallH = Math.max(8, height - bottomT);
      const wallY = bottomT + wallH / 2;

      // front/back
      geoms.push(addBox(group, length, wallH, wallT, 0, wallY, depth / 2 - wallT / 2, woodMat));
      geoms.push(addBox(group, length, wallH, wallT, 0, wallY, -depth / 2 + wallT / 2, woodMat));

      // left/right
      geoms.push(addBox(group, wallT, wallH, depth - wallT * 2, -length / 2 + wallT / 2, wallY, 0, woodMat));
      geoms.push(addBox(group, wallT, wallH, depth - wallT * 2, length / 2 - wallT / 2, wallY, 0, woodMat));

      // Notes: feet
      if (has(notes, "feet", "legs", "stand")) {
        const foot = clamp(wallT * 1.25, 0.8, 2.4);
        const fy = foot / 2;
        geoms.push(addBox(group, foot, foot, foot, xMin + foot, fy, zMin + foot, darkMat));
        geoms.push(addBox(group, foot, foot, foot, xMax - foot, fy, zMin + foot, darkMat));
        geoms.push(addBox(group, foot, foot, foot, xMin + foot, fy, zMax - foot, darkMat));
        geoms.push(addBox(group, foot, foot, foot, xMax - foot, fy, zMax - foot, darkMat));
      }
    }

    // --- DEFAULT (fallback block) ---
    else {
      geoms.push(addBox(group, length, height, depth, 0, height / 2, 0, woodMat));
    }

    return { group, length, depth, height, geoms };
  }

  export async function renderBuildPreviewPng(args: {
    view: View;
    projectType: string;
    title?: string;
    notes?: string;
    dims: BuildDims;
    options: BuildOptions;
    width?: number;
    height?: number;
  }) {
    const W = args.width ?? 1200;
    const H = args.height ?? 800;

    // Scene
    const scene = new THREE.Scene();
    scene.background = new THREE.Color(0x0b1220);

    // Lights
    const hemi = new THREE.HemisphereLight(0xBBD7FF, 0x101827, 0.95);
    scene.add(hemi);

    const key = new THREE.DirectionalLight(0xFFFFFF, 1.05);
    key.position.set(240, 280, 200);
    scene.add(key);

    const rim = new THREE.DirectionalLight(0xA78BFA, 0.55);
    rim.position.set(-260, 160, -200);
    scene.add(rim);

    // Model (now notes-aware)
    const model = buildModel({
      projectType: args.projectType,
      dims: args.dims,
      options: args.options,
      notes: args.notes,
    });
    scene.add(model.group);

    // Ground
    const ground = new THREE.Mesh(
      new THREE.PlaneGeometry(1200, 1200),
      new THREE.MeshStandardMaterial({ color: 0x0f1a2f, roughness: 1, metalness: 0 })
    );
    ground.rotation.x = -Math.PI / 2;
    ground.position.y = 0;
    scene.add(ground);

    const grid = new THREE.GridHelper(1000, 32, 0x334155, 0x1f2a44);
    (grid.material as THREE.Material).transparent = true;
    (grid.material as THREE.Material).opacity = 0.25;
    scene.add(grid);

    // Camera
    const frustumSize = fitToView(model.length, model.depth, model.height);
    const camera = makeCamera(args.view, frustumSize);

    // Renderer (offscreen)
    const canvas = document.createElement("canvas");
    const renderer = new THREE.WebGLRenderer({
      canvas,
      antialias: true,
      alpha: false,
      preserveDrawingBuffer: true,
    });

    renderer.setPixelRatio(Math.min(window.devicePixelRatio || 1, 2));
    renderer.setSize(W, H, false);

    // Ortho aspect correction
    const aspect = W / H;
    const ortho = camera as THREE.OrthographicCamera;
    const s = frustumSize;
    ortho.left = -s * aspect;
    ortho.right = s * aspect;
    ortho.top = s;
    ortho.bottom = -s;
    ortho.updateProjectionMatrix();

    renderer.render(scene, camera);

    // Cleanup
    model.geoms.forEach((g) => g.dispose());
    (ground.geometry as THREE.BufferGeometry).dispose();
    (ground.material as THREE.Material).dispose();
    (grid.geometry as THREE.BufferGeometry).dispose();
    (grid.material as THREE.Material).dispose();
    renderer.dispose();

    return canvas.toDataURL("image/png");
  }
```