The Dugout Training Facility

Cage Management & Scheduling Dashboard

Copyright Registration Deposit

Deposit contents:
- index.html
- admin.html
- app.js
- admin.js
- style.css
- service-worker.js
- migrate-reservations.html
- manifest.json

File: index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>The Dugout – Live Cage Dashboard</title>

  <!-- Icons + Manifest -->
  <link rel="icon" type="image/png" href="/icons/dugout-logo.png" />
  <link rel="apple-touch-icon" href="/icons/dugout-app-512-transparent.png" />
  <link rel="manifest" href="/manifest.json" />

  <!-- Styles -->
  <link rel="stylesheet" href="style.css" />
</head>

<body>
  <!-- HERO HEADER -->
  <header class="hero">
    <div class="hero-left">
      <img src="/icons/dugout-logo.png" class="hero-logo" alt="The Dugout Training Facility logo">
      <div>
        <h1>The Dugout Training Facility</h1>
        <p>Live Cage &amp;amp; Lane Dashboard</p>
      </div>
    </div>

    <div class="hero-right">
      <div class="open-status">
        <span id="openStatusDot" class="status-dot"></span>
        <span id="openStatusLabel">Checking hours…</span>
      </div>
      <div id="lastUpdated" class="last-updated">Last updated: --:--</div>
    </div>
  </header>

  <!-- INFO PILL STRIP -->
  <section class="info-strip">
    <div class="info-pill">
      Staffed hours: Mon–Fri 5–8PM • Sat 10–2
    </div>
    <div class="info-pill">
      Members: 6AM–10PM daily
    </div>
    <div class="info-pill">
      Live cage updates occur during staffed hours only. Trainer sessions update automatically.
    </div>
  </section>

  <!-- MAIN BOARD -->
  <main class="board">
    <section id="cards-container" class="cards-grid"></section>

    <!-- WAITLIST -->
    <section class="waitlist-section">
      <div class="waitlist-card">
        <div class="waitlist-header">
          <h2>Waitlist</h2>
          <span class="waitlist-tagline">Managed by staff in the admin panel.</span>
        </div>
        <div id="waitlist-body" class="waitlist-body">
          <p class="waitlist-empty">No one on the waitlist yet.</p>
        </div>
      </div>
    </section>
  </main>

  <footer class="footer-note">
```

```
    For real-time changes, see staff at the front desk. Data updates instantly from the Dugout Admin Panel.
  </footer>

  <!-- Firebase + Dashboard Scripts -->
  <script type="module" src="app.js"></script>

  <!-- Service Worker -->
  <script>
    if ("serviceWorker" in navigator) {
      window.addEventListener("load", function () {
        navigator.serviceWorker.register("/service-worker.js");
      });
    }
  </script>
</body>
</html>
```

File: admin.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Dugout Admin Panel</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="style.css" />
  <style>
    body {
      background: #111827;
      color: #f9fafb;
      font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI", sans-serif;
      margin: 0;
      padding: 0;
    }

    header.admin-header {
      display: flex;
      align-items: center;
      gap: 16px;
      padding: 12px 20px;
      background: #0b1220;
      border-bottom: 1px solid #1f2937;
    }

    header.admin-header h1 {
      margin: 0;
      font-size: 1.5rem;
      letter-spacing: 0.05em;
      text-transform: uppercase;
    }

    main {
      max-width: 1000px;
      margin: 16px auto 40px;
      padding: 0 16px;
    }

    .admin-section {
      background: #020617;
      border-radius: 16px;
      padding: 16px 18px 20px;
      margin-bottom: 16px;
      box-shadow: 0 10px 30px rgba(0,0,0,0.4);
      border: 1px solid #1f2937;
    }

    .admin-section h2 {
      margin: 0 0 10px;
      font-size: 1.1rem;
      text-transform: uppercase;
      letter-spacing: 0.08em;
      color: #e5e7eb;
    }

    .admin-section p.sub {
      margin: 0 0 12px;
      font-size: 0.85rem;
      color: #9ca3af;
    }

    .row {
      display: flex;
      flex-wrap: wrap;
      gap: 12px;
      margin-bottom: 12px;
    }
```

```
.field {
  display: flex;
  flex-direction: column;
  gap: 4px;
  min-width: 140px;
  flex: 1;
}

.field label {
  font-size: 0.8rem;
  text-transform: uppercase;
  letter-spacing: 0.06em;
  color: #9ca3af;
}

.field select,
.field input,
.field textarea {
  padding: 6px 8px;
  border-radius: 8px;
  border: 1px solid #374151;
  background: #020617;
  color: #f9fafb;
  font-size: 0.9rem;
}

.field input::placeholder,
.field textarea::placeholder {
  color: #6b7280;
}

textarea {
  resize: vertical;
  min-height: 60px;
}

.btn-row {
  display: flex;
  gap: 10px;
  margin-top: 4px;
  flex-wrap: wrap;
}

.btn-primary,
.btn-secondary,
.btn-delete {
  border-radius: 999px;
  border: none;
  padding: 7px 14px;
  font-size: 0.85rem;
  font-weight: 600;
  cursor: pointer;
  text-transform: uppercase;
  letter-spacing: 0.08em;
}

.btn-primary {
  background: #22c55e;
  color: #022c22;
}

.btn-primary:hover {
  background: #16a34a;
}

.btn-secondary {
  background: #111827;
  color: #e5e7eb;
  border: 1px solid #4b5563;
}

.btn-secondary:hover {
```

```
  background: #1f2937;
}

.btn-delete {
  padding: 3px 10px;
  font-size: 0.75rem;
  border-radius: 999px;
  background: #111827;
  color: #fca5a5;
  border: 1px solid #4b5563;
}

.btn-delete:hover {
  background: #7f1d1d;
  color: #fee2e2;
  border-color: #b91c1c;
}

.message {
  margin-top: 8px;
  font-size: 0.8rem;
}

/* PIN gate */
.pin-gate {
  position: fixed;
  inset: 0;
  background: radial-gradient(circle at top, #1f2937, #020617 55%);
  display: flex;
  align-items: center;
  justify-content: center;
  z-index: 50;
}

.pin-card {
  background: #020617;
  padding: 24px 20px 20px;
  border-radius: 16px;
  border: 1px solid #1f2937;
  box-shadow: 0 25px 50px rgba(0,0,0,0.8);
  width: 280px;
  text-align: center;
}

.pin-card h2 {
  margin: 0 0 8px;
  font-size: 1.1rem;
  text-transform: uppercase;
  letter-spacing: 0.08em;
  color: #e5e7eb;
}

.pin-card p {
  margin: 0 0 12px;
  font-size: 0.85rem;
  color: #9ca3af;
}

#pinInput {
  width: 100%;
  padding: 8px 10px;
  border-radius: 999px;
  border: 1px solid #4b5563;
  background: #020617;
  color: #f9fafb;
  text-align: center;
  letter-spacing: 0.35em;
  font-size: 1.1rem;
  box-sizing: border-box;
}

#pinBtn {
```

```
      margin-top: 10px;
      width: 100%;
    }

    #pinError {
      margin-top: 8px;
      font-size: 0.8rem;
      color: #f97373;
    }

    /* Schedule table */
    .schedule-table {
      width: 100%;
      border-collapse: collapse;
      margin-top: 8px;
      font-size: 0.85rem;
    }

    .schedule-table th,
    .schedule-table td {
      padding: 6px 8px;
      border-bottom: 1px solid #1f2937;
      text-align: left;
      white-space: nowrap;
    }

    .schedule-table th {
      text-transform: uppercase;
      letter-spacing: 0.06em;
      font-size: 0.75rem;
      color: #9ca3af;
    }

    .pill {
      display: inline-block;
      padding: 2px 8px;
      border-radius: 999px;
      font-size: 0.75rem;
      text-transform: uppercase;
      letter-spacing: 0.08em;
    }

    .pill--trainer {
      background: rgba(59, 130, 246, 0.15);
      color: #93c5fd;
    }

    .pill--team {
      background: rgba(236, 72, 153, 0.15);
      color: #f9a8d4;
    }

    .pill--athlete {
      background: rgba(16, 185, 129, 0.15);
      color: #6ee7b7;
    }

    @media (max-width: 640px) {
      main {
        padding: 0 10px;
      }
    }
  </style>
</head>
<body>
  <!-- PIN Gate -->
  <div class="pin-gate" id="pinGate">
    <div class="pin-card">
      <h2>Dugout Staff Login</h2>
      <p>Enter staff PIN to manage cages & reservations.</p>
      <input id="pinInput" type="password" maxlength="6" autocomplete="off" />
      <button id="pinBtn" class="btn-primary">Unlock</button>
```

```
    <div id="pinError"></div>
  </div>
</div>

<header class="admin-header">
  <h1>The Dugout • Admin Panel</h1>
</header>

<main id="adminMain" style="display:none;">
  <!-- SECTION 1: Quick Live Cage Update -->
  <section class="admin-section">
    <h2>Live Cage Update (Now)</h2>
    <p class="sub">
      Use this when athletes walk in. Updates hit the TV + public board instantly.
    </p>

    <div class="row">
      <div class="field">
        <label for="liveCage">Cage / Lane</label>
        <select id="liveCage">
          <option value="1">Cage 1</option>
          <option value="2">Cage 2</option>
          <option value="3">Cage 3</option>
          <option value="4">Cage 4</option>
          <option value="5">Cage 5</option>
          <option value="6">Cage 6</option>
          <option value="7">Cage 7</option>
          <option value="8">Cage 8</option>
          <option value="9">Lane 1</option>
          <option value="10">Lane 2</option>
          <option value="11">Lane 3</option>
          <option value="12">Lane 4</option>
          <option value="13">Lane 5</option>
        </select>
      </div>

      <div class="field">
        <label for="liveType">Type</label>
        <select id="liveType">
          <option value="athlete">Athlete</option>
          <option value="team">Team</option>
          <option value="trainer">Trainer</option>
        </select>
      </div>

      <div class="field">
        <label for="liveName">Name</label>
        <input
          id="liveName"
          type="text"
          placeholder="Athlete / team / trainer name"
        />
      </div>

      <div class="field">
        <label for="liveMinutes">Duration (minutes)</label>
        <input
          id="liveMinutes"
          type="number"
          min="5"
          max="180"
          step="5"
          value="60"
        />
      </div>
    </div>

    <div class="btn-row">
      <button id="btnLiveBusy" class="btn-primary">Mark BUSY Now</button>
      <button id="btnLiveOpen" class="btn-secondary">Mark OPEN / Clear</button>
    </div>
    <div id="liveMessage" class="message"></div>
```

```
</section>

<!-- SECTION 2: Future Reservations (Date-based) -->
<section class="admin-section">
  <h2>Schedule Future Reservation</h2>
  <p class="sub">
    Build tomorrow or next week's schedule. When the date + time hits,
    the board automatically shows it as RESERVED with countdown.
  </p>

  <div class="row">
    <div class="field">
      <label for="resCage">Cage / Lane</label>
      <select id="resCage">
        <option value="1">Cage 1</option>
        <option value="2">Cage 2</option>
        <option value="3">Cage 3</option>
        <option value="4">Cage 4</option>
        <option value="5">Cage 5</option>
        <option value="6">Cage 6</option>
        <option value="7">Cage 7</option>
        <option value="8">Cage 8</option>
        <option value="9">Lane 1</option>
        <option value="10">Lane 2</option>
        <option value="11">Lane 3</option>
        <option value="12">Lane 4</option>
        <option value="13">Lane 5</option>
      </select>
    </div>

    <div class="field">
      <label for="resType">Type</label>
      <select id="resType">
        <option value="athlete">Athlete</option>
        <option value="team">Team</option>
        <option value="trainer">Trainer</option>
      </select>
    </div>

    <div class="field">
      <label for="resName">Name</label>
      <input
        id="resName"
        type="text"
        placeholder="Cameron Peters, TBS 12U, Coach Joey..."
      />
    </div>
  </div>

  <div class="row">
    <div class="field">
      <label for="resDate">Date</label>
      <input id="resDate" type="date" />
    </div>

    <div class="field">
      <label for="resStart">Start time</label>
      <input id="resStart" type="time" />
    </div>

    <div class="field">
      <label for="resEnd">End time</label>
      <input id="resEnd" type="time" />
    </div>
  </div>

  <div class="btn-row">
    <button id="resAddBtn" class="btn-primary">Add Reservation</button>
  </div>
  <div id="resMessage" class="message"></div>
</section>
```

```
<!-- SECTION 3: Schedule Overview -->
<section class="admin-section">
  <h2>Schedule Overview</h2>
  <p class="sub">
    See everything already booked for a given day so you don't double-enter a cage or lane.
    Tap the ■■ icon to remove a reservation.
  </p>

  <div class="row">
    <div class="field">
      <label for="scheduleDate">Date</label>
      <input id="scheduleDate" type="date" />
    </div>

    <div class="field">
      <label for="scheduleCageFilter">Filter</label>
      <select id="scheduleCageFilter">
        <option value="all">All cages & lanes</option>
        <option value="1">Cage 1</option>
        <option value="2">Cage 2</option>
        <option value="3">Cage 3</option>
        <option value="4">Cage 4</option>
        <option value="5">Cage 5</option>
        <option value="6">Cage 6</option>
        <option value="7">Cage 7</option>
        <option value="8">Cage 8</option>
        <option value="9">Lane 1</option>
        <option value="10">Lane 2</option>
        <option value="11">Lane 3</option>
        <option value="12">Lane 4</option>
        <option value="13">Lane 5</option>
      </select>
    </div>
  </div>

  <div id="scheduleList" class="message"></div>
</section>

<!-- SECTION 4: Waitlist Management -->
<section class="admin-section">
  <h2>Waitlist</h2>
  <p class="sub">
    Add athletes to the waitlist and manage the live list shown on the public board.
  </p>

  <div class="row">
    <div class="field">
      <label for="waitName">Name</label>
      <input
        id="waitName"
        type="text"
        placeholder="Athlete / group name"
      />
    </div>

    <div class="field">
      <label for="waitCage">Preferred cage / lane (optional)</label>
      <input
        id="waitCage"
        type="text"
        placeholder="e.g. Cage 3, Any lane"
      />
    </div>
  </div>

  <div class="row">
    <div class="field">
      <label for="waitNotes">Notes (optional)</label>
      <textarea
        id="waitNotes"
        placeholder="Parent waiting, hitting only, etc."
      ></textarea>
```

```
        </div>
      </div>

      <div class="btn-row">
        <button id="waitAddBtn" class="btn-primary">Add to Waitlist</button>
      </div>

      <div id="waitMessage" class="message"></div>
      <div id="waitAdminList" class="message" style="margin-top:10px;"></div>
    </section>
  </main>

  <script type="module" src="admin.js"></script>
</body>
</html>
```

File: app.js

```
// app.js
import { initializeApp } from "https://www.gstatic.com/firebasejs/11.0.1/firebase-app.js";
import {
  getDatabase,
  ref,
  onValue
} from "https://www.gstatic.com/firebasejs/11.0.1/firebase-database.js";

// Your Firebase config
const firebaseConfig = {
  apiKey: "AIzaSyDhuSPwW2M2SyRpOyWMq36ze9G5fzU8zH8",
  authDomain: "dugout-cage-dashboard.firebaseapp.com",
  databaseURL: "https://dugout-cage-dashboard-default-rtdb.firebaseio.com",
  projectId: "dugout-cage-dashboard",
  storageBucket: "dugout-cage-dashboard.firebasestorage.app",
  messagingSenderId: "373269622046",
  appId: "1:373269622046:web:064221adde4ea632636d0f"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
const db = getDatabase(app);

// DOM references
const cardsContainer = document.getElementById("cards-container");
const lastUpdated = document.getElementById("lastUpdated");
const openStatusDot = document.getElementById("openStatusDot");
const openStatusLabel = document.getElementById("openStatusLabel");
const waitlistBody = document.getElementById("waitlist-body");

// In-memory snapshots
let cagesData = {};
let reservationsData = {};

// Listen for cages
onValue(
  ref(db, "cages"),
  (snapshot) => {
    cagesData = snapshot.val() || {};
    renderAll();
  },
  (error) => {
    console.error("Error listening to cages:", error);
  }
);

// Listen for reservations
onValue(
  ref(db, "reservations"),
  (snapshot) => {
    reservationsData = snapshot.val() || {};
    renderAll();
  },
  (error) => {
    console.error("Error listening to reservations:", error);
  }
);

// Listen for waitlist (public view)
onValue(
  ref(db, "waitlist"),
  (snapshot) => {
    const wt = snapshot.val() || {};
    renderWaitlist(wt);
  },
  (error) => {
    console.error("Error listening to waitlist:", error);
    if (waitlistBody) {
```

```
      waitlistBody.innerHTML =
        `<p class="waitlist-empty">Error loading waitlist.</p>`;
    }
  }
);

// Render everything dependent on cages + reservations
function renderAll() {
  renderCageCards(cagesData, reservationsData);
  updateLastUpdated();
  updateOpenStatusBanner();
}

// Last updated
function updateLastUpdated() {
  if (!lastUpdated) return;
  const now = new Date();
  lastUpdated.textContent =
    "Last updated: " +
    now.toLocaleTimeString([], { hour: "numeric", minute: "2-digit" });
}

// Staffed vs members-only (openStatus banner)
function updateOpenStatusBanner() {
  if (!openStatusDot || !openStatusLabel) return;

  const now = new Date();
  const day = now.getDay(); // 0=Sun..6=Sat
  const hour = now.getHours();
  const minute = now.getMinutes();
  const totalMinutes = hour * 60 + minute;

  let openToPublic = false;

  // Mon–Fri 5–8pm
  if (day >= 1 && day <= 5) {
    if (totalMinutes >= 17 * 60 && totalMinutes < 20 * 60) {
      openToPublic = true;
    }
  }
  // Sat 10am–2pm
  if (day === 6) {
    if (totalMinutes >= 10 * 60 && totalMinutes < 14 * 60) {
      openToPublic = true;
    }
  }

  openStatusDot.classList.remove("status-dot--open", "status-dot--closed");

  if (openToPublic) {
    openStatusDot.classList.add("status-dot--open");
    openStatusLabel.textContent = "Open to public (walk-ins welcome)";
  } else {
    openStatusDot.classList.add("status-dot--closed");
    openStatusLabel.textContent =
      "Members only – live updates during staffed hours";
  }
}

// Render cage cards
function renderCageCards(cagesData, reservationsData) {
  if (!cardsContainer) return;

  // We have 13 “slots” (Cages 1–8, Lanes 1–5)
  const ids = [];
  for (let i = 1; i <= 13; i++) {
    ids.push(String(i));
  }

  cardsContainer.innerHTML = "";

  const nowMs = Date.now();
```

```
const today = new Date(nowMs);
const todayYear = today.getFullYear();
const todayMonth = today.getMonth();
const todayDate = today.getDate();

ids.forEach((id) => {
  const cage = cagesData[id] || {};
  const baseLabel = cageLabelFromId(id);

  let statusRaw = (cage.status || "OPEN").toUpperCase();
  let currentName = cage.currentName || "";
  const type = (cage.type || "").toLowerCase();

  const startTimeMs =
    typeof cage.startTime === "number" ? cage.startTime : null;
  const endTimeMs =
    typeof cage.endTime === "number" ? cage.endTime : null;

  // Derive reservations for this cage
  const cageReservations = [];
  const bucket = (reservationsData && reservationsData[id]) || {};
  Object.keys(bucket).forEach((resId) => {
    const res = bucket[resId];
    if (!res || typeof res.start !== "number" || typeof res.end !== "number")
      return;

    cageReservations.push({
      id: resId,
      ...res
    });
  });

  // Separate into current vs upcoming
  let activeReservation = null;
  let nextReservation = null;
  let thenReservation = null;

  cageReservations.forEach((res) => {
    const s = res.start;
    const e = res.end;

    if (s <= nowMs && nowMs < e) {
      // Currently in this reservation window
      if (!activeReservation || s < activeReservation.start) {
        activeReservation = res;
      }
    } else if (s > nowMs) {
      // Upcoming
      if (!nextReservation || s < nextReservation.start) {
        // shift the old next into then
        if (nextReservation) {
          thenReservation = nextReservation;
        }
        nextReservation = res;
      } else if (!thenReservation || s < thenReservation.start) {
        thenReservation = res;
      }
    }
  });

  // If not BUSY but there is an active reservation, treat as RESERVED
  let activeStart = startTimeMs;
  let activeEnd = endTimeMs;

  if (statusRaw !== "BUSY" && activeReservation) {
    statusRaw = "RESERVED";
    currentName = activeReservation.name || "";
    activeStart = activeReservation.start;
    activeEnd = activeReservation.end;
  }

  // Determine status class & label
```

```
let statusClass = "card--open";
let statusLabel = "OPEN";

if (statusRaw === "BUSY") {
  statusClass = "card--busy";
  statusLabel = "BUSY";
} else if (statusRaw === "RESERVED") {
  statusClass = "card--reserved";
  statusLabel = "RESERVED";
} else {
  // statusRaw = OPEN but there might be an upcoming reservation
  if (nextReservation && typeof nextReservation.start === "number") {
    const nsDate = new Date(nextReservation.start);
    statusLabel = `OPEN until ${formatTime(nsDate)}`;
  } else {
    statusLabel = "OPEN";
  }
}

// Timer and progress
let timerText = "";
let progressPct = 0;

if (
  typeof activeStart === "number" &&
  typeof activeEnd === "number" &&
  activeEnd > activeStart
) {
  const total = activeEnd - activeStart;
  const remaining = Math.max(0, activeEnd - nowMs);

  if (remaining > 0 && nowMs >= activeStart) {
    const minsRemaining = Math.ceil(remaining / 60000);
    const hours = Math.floor(minsRemaining / 60);
    const mins = minsRemaining % 60;
    timerText = `Time left: ${hours}:${String(mins).padStart(2, "0")}`;
    progressPct = (remaining / total) * 100;
  } else if (nowMs >= activeEnd) {
    timerText = "Time is up";
    progressPct = 0;
  }
}

// Next / Then text
let nextText = "";
const pieces = [];

if (
  nextReservation &&
  typeof nextReservation.start === "number" &&
  typeof nextReservation.end === "number"
) {
  const s = new Date(nextReservation.start);
  const e = new Date(nextReservation.end);

  const isFutureDay =
    s.getFullYear() !== todayYear ||
    s.getMonth() !== todayMonth ||
    s.getDate() !== todayDate;

  const datePart = isFutureDay
    ? s.toLocaleDateString([], {
        month: "numeric",
        day: "numeric"
      }) + " "
    : "";

  pieces.push(
    `Next: ${nextReservation.name || ""} ${datePart}${formatTime(
      s
    )}–${formatTime(e)}`
  );
```

```
}

if (
  thenReservation &&
  typeof thenReservation.start === "number" &&
  typeof thenReservation.end === "number"
) {
  const s2 = new Date(thenReservation.start);
  const e2 = new Date(thenReservation.end);

  const isFutureDay2 =
    s2.getFullYear() !== todayYear ||
    s2.getMonth() !== todayMonth ||
    s2.getDate() !== todayDate;

  const datePart2 = isFutureDay2
    ? s2.toLocaleDateString([], {
        month: "numeric",
        day: "numeric"
      }) + " "
    : "";

  pieces.push(
    `Then: ${thenReservation.name || ""} ${datePart2}${formatTime(
      s2
    )}–${formatTime(e2)}`
  );
}

if (pieces.length) {
  nextText = pieces.join(" | ");
}

// Build card element
const card = document.createElement("article");
card.className = `card ${statusClass}`;

const typeClass =
  type === "trainer"
    ? "type-pill--trainer"
    : type === "team"
    ? "type-pill--team"
    : "type-pill--athlete";

const typeLabel =
  type === "trainer" ? "Trainer" : type === "team" ? "Team" : "Athlete";

card.innerHTML = `
  <div class="card-title">${baseLabel}</div>
  <div class="card-status">${escapeHtml(statusLabel)}</div>

  <div class="card-meta-row">
    <div class="card-name">
      ${currentName ? escapeHtml(currentName) : " "}
    </div>
    ${
      currentName
        ? `<span class="type-pill ${typeClass}">${typeLabel}</span>`
        : ""
    }
  </div>

  <div class="card-timer-row">
    <div class="timer-label">
      ${timerText ? escapeHtml(timerText) : " "}
    </div>
    <div class="timer-bar-wrap">
      <div class="timer-bar-fill" style="width:${progressPct}%;"></div>
    </div>
  </div>

  <div class="card-next">
```

```
        ${nextText ? escapeHtml(nextText) : " "}
      </div>
    `;

    cardsContainer.appendChild(card);
  });
}

// Public waitlist (on dashboard)
function renderWaitlist(waitlistData) {
  if (!waitlistBody) return;

  const entries = Object.values(waitlistData).map((entry) => entry || {});

  // Sort oldest first by createdAt
  entries.sort((a, b) => {
    const aTs = typeof a.createdAt === "number" ? a.createdAt : 0;
    const bTs = typeof b.createdAt === "number" ? b.createdAt : 0;
    return aTs - bTs;
  });

  if (!entries.length) {
    waitlistBody.innerHTML =
      `<p class="waitlist-empty">No one on the waitlist yet.</p>`;
    return;
  }

  const lines = entries.slice(0, 10).map((entry) => {
    const name = entry.name || "";
    const cage = entry.cage || entry.preferredCage || "";
    const notes = entry.notes || "";

    return `
      <div class="waitlist-row">
        <div class="waitlist-name">${escapeHtml(name)}</div>
        <div class="waitlist-meta">
          ${
            cage
              ? `<span class="waitlist-chip">${escapeHtml(cage)}</span>`
              : ""
          }
          ${
            notes
              ? `<span class="waitlist-notes">${escapeHtml(notes)}</span>`
              : ""
          }
        </div>
      </div>
    `;
  });

  waitlistBody.innerHTML = lines.join("");
}

// Helpers
function cageLabelFromId(id) {
  const n = parseInt(id, 10);
  if (!isNaN(n)) {
    if (n <= 8) return `Cage ${n}`;
    return `Lane ${n - 8}`;
  }
  return `Cage ${id}`;
}

function formatTime(date) {
  return date.toLocaleTimeString([], {
    hour: "numeric",
    minute: "2-digit"
  });
}

function escapeHtml(str) {
```

```
  return String(str)
    .replace(/&/g, "&amp;")
    .replace(/</g, "&lt;")
    .replace(/>/g, "&gt;")
    .replace(/"/g, "&quot;");
}

// Smooth 20-second refresh for dashboard timers (no page reload)
setInterval(() => {
  try {
    renderAll();
  } catch (e) {
    console.error("Error during smooth refresh:", e);
  }
}, 20000);
```

File: admin.js

```
// admin.js
import { initializeApp } from "https://www.gstatic.com/firebasejs/11.0.1/firebase-app.js";
import {
  getDatabase,
  ref,
  onValue,
  push,
  update,
  remove
} from "https://www.gstatic.com/firebasejs/11.0.1/firebase-database.js";

// ========================
// CONFIG & HELPERS
// ========================

// ■ Admin PIN – do NOT change unless you explicitly request it
const STAFF_PIN = "2580";

const firebaseConfig = {
  apiKey: "AIzaSyDhuSPwW2M2SyRpOyWMq36ze9G5fzU8zH8",
  authDomain: "dugout-cage-dashboard.firebaseapp.com",
  databaseURL: "https://dugout-cage-dashboard-default-rtdb.firebaseio.com",
  projectId: "dugout-cage-dashboard",
  storageBucket: "dugout-cage-dashboard.firebasestorage.app",
  messagingSenderId: "373269622046",
  appId: "1:373269622046:web:064221adde4ea632636d0f"
};

const app = initializeApp(firebaseConfig);
const db = getDatabase(app);

function setMessage(el, text, color) {
  if (!el) return;
  el.textContent = text || "";
  el.style.color = color || "";
}

function cageLabelFromId(id) {
  const n = parseInt(id, 10);
  if (!isNaN(n)) {
    if (n <= 8) return `Cage ${n}`;
    return `Lane ${n - 8}`;
  }
  return `Cage ${id}`;
}

function escapeHtml(str) {
  return String(str)
    .replace(/&/g, "&amp;")
    .replace(/</g, "&lt;")
    .replace(/>/g, "&gt;")
    .replace(/"/g, "&quot;");
}

function formatTime(date) {
  return date.toLocaleTimeString([], {
    hour: "numeric",
    minute: "2-digit"
  });
}

function sameDay(a, b) {
  return (
    a.getFullYear() === b.getFullYear() &&
    a.getMonth() === b.getMonth() &&
    a.getDate() === b.getDate()
  );
}
```

```
// ========================
// PIN GATE
// ========================
const pinGate = document.getElementById("pinGate");
const pinInput = document.getElementById("pinInput");
const pinBtn = document.getElementById("pinBtn");
const pinError = document.getElementById("pinError");
const adminMain = document.getElementById("adminMain");

function handlePin() {
  const value = (pinInput.value || "").trim();
  if (value === STAFF_PIN) {
    if (pinGate) pinGate.style.display = "none";
    if (adminMain) adminMain.style.display = "block";
    if (pinError) pinError.textContent = "";
  } else {
    if (pinError) pinError.textContent = "Incorrect PIN. Please try again.";
  }
}

if (pinBtn) {
  pinBtn.addEventListener("click", handlePin);
}
if (pinInput) {
  pinInput.addEventListener("keyup", (e) => {
    if (e.key === "Enter") handlePin();
  });
}

// ========================
// SECTION 1 – LIVE CAGE UPDATE
// ========================
const liveCageEl = document.getElementById("liveCage");
const liveTypeEl = document.getElementById("liveType");
const liveNameEl = document.getElementById("liveName");
const liveMinutesEl = document.getElementById("liveMinutes");
const btnLiveBusy = document.getElementById("btnLiveBusy");
const btnLiveOpen = document.getElementById("btnLiveOpen");
const liveMessageEl = document.getElementById("liveMessage");

if (btnLiveBusy) {
  btnLiveBusy.addEventListener("click", async () => {
    const cageId = liveCageEl.value;
    const type = liveTypeEl.value;
    const name = (liveNameEl.value || "").trim();
    let minutes = parseInt(liveMinutesEl.value, 10);

    if (!cageId || !type || !name) {
      setMessage(
        liveMessageEl,
        "Please choose a cage, type, and enter a name.",
        "orange"
      );
      return;
    }

    if (isNaN(minutes) || minutes <= 0) {
      minutes = 30;
    }

    const now = Date.now();
    const startTime = now;
    const endTime = now + minutes * 60 * 1000;

    try {
      await update(ref(db, "cages/" + cageId), {
        status: "BUSY",
        type,
        currentName: name,
        startTime,
        endTime
```

```
      });

      setMessage(
        liveMessageEl,
        `${cageLabelFromId(cageId)} set to BUSY for ${name}`,
        "limegreen"
      );

      liveNameEl.value = "";
      liveMinutesEl.value = "";
    } catch (err) {
      console.error("Error updating cage BUSY:", err);
      setMessage(liveMessageEl, "Error updating cage.", "red");
    }
  });
}

if (btnLiveOpen) {
  btnLiveOpen.addEventListener("click", async () => {
    const cageId = liveCageEl.value;
    if (!cageId) {
      setMessage(liveMessageEl, "Choose a cage first.", "orange");
      return;
    }

    try {
      await update(ref(db, "cages/" + cageId), {
        status: "OPEN",
        type: "",
        currentName: "",
        startTime: null,
        endTime: null
      });

      setMessage(
        liveMessageEl,
        `${cageLabelFromId(cageId)} set to OPEN.`,
        "limegreen"
      );

      liveNameEl.value = "";
      liveMinutesEl.value = "";
    } catch (err) {
      console.error("Error updating cage OPEN:", err);
      setMessage(liveMessageEl, "Error updating cage.", "red");
    }
  });
}

// ========================
// SECTION 2 – FUTURE RESERVATIONS
// ========================
const resCageEl = document.getElementById("resCage");
const resTypeEl = document.getElementById("resType");
const resNameEl = document.getElementById("resName");
const resDateEl = document.getElementById("resDate");
const resStartEl = document.getElementById("resStart");
const resEndEl = document.getElementById("resEnd");
const resAddBtn = document.getElementById("resAddBtn");
const resMessageEl = document.getElementById("resMessage");

// default reservation date to today
if (resDateEl) {
  const todayIso = new Date().toISOString().slice(0, 10);
  resDateEl.value = todayIso;
}

// global snapshot of reservations
let reservationsSnapshot = {}; // structure: { cageId: {resId: {...}} }

onValue(ref(db, "reservations"), (snapshot) => {
  reservationsSnapshot = snapshot.val() || {};
```

```
  renderScheduleOverview();
});

if (resAddBtn) {
  resAddBtn.addEventListener("click", async () => {
    const cageId = resCageEl.value;
    const type = resTypeEl.value;
    const name = (resNameEl.value || "").trim();
    const date = resDateEl.value;
    const startTimeStr = resStartEl.value;
    const endTimeStr = resEndEl.value;

    if (!cageId || !type || !name || !date || !startTimeStr || !endTimeStr) {
      setMessage(resMessageEl, "Please fill all fields.", "orange");
      return;
    }

    const start = new Date(date + "T" + startTimeStr);
    const end = new Date(date + "T" + endTimeStr);

    if (isNaN(start.getTime()) || isNaN(end.getTime()) || end <= start) {
      setMessage(
        resMessageEl,
        "Please check your date and time range.",
        "orange"
      );
      return;
    }

    // overlap check within the same cage & same day
    const cageRes = reservationsSnapshot[cageId] || {};
    const newStartMs = start.getTime();
    const newEndMs = end.getTime();
    let conflict = null;

    Object.keys(cageRes).forEach((resId) => {
      const res = cageRes[resId];
      if (!res || typeof res.start !== "number" || typeof res.end !== "number")
        return;

      const existingStart = new Date(res.start);
      const existingEnd = new Date(res.end);

      if (!sameDay(existingStart, start)) return;

      // overlap if newStart < existingEnd && existingStart < newEnd
      if (newStartMs < res.end && res.start < newEndMs) {
        conflict = {
          name: res.name || "",
          start: existingStart,
          end: existingEnd
        };
      }
    });

    if (conflict) {
      setMessage(
        resMessageEl,
        `Conflict: ${cageLabelFromId(
          cageId
        )} already booked for ${conflict.name} (${formatTime(
          conflict.start
        )}–${formatTime(conflict.end)}).`,
        "red"
      );
      return;
    }

    // save reservation
    try {
      const resRef = ref(db, "reservations/" + cageId);
      await push(resRef, {
```

```
        name,
        type,
        start: start.getTime(),
        end: end.getTime(),
        createdAt: Date.now()
      });

      setMessage(resMessageEl, "Reservation added.", "limegreen");
      resNameEl.value = "";
      resStartEl.value = "";
      resEndEl.value = "";
    } catch (err) {
      console.error("Error adding reservation:", err);
      setMessage(resMessageEl, "Error adding reservation.", "red");
    }
  });
}

// ========================
// SECTION 3 – SCHEDULE OVERVIEW (by date)
// ========================
const scheduleDateEl = document.getElementById("scheduleDate");
const scheduleCageFilterEl = document.getElementById("scheduleCageFilter");
const scheduleListEl = document.getElementById("scheduleList");

// default schedule date to today
if (scheduleDateEl) {
  const todayIso = new Date().toISOString().slice(0, 10);
  scheduleDateEl.value = todayIso;
}

if (scheduleDateEl) {
  scheduleDateEl.addEventListener("change", renderScheduleOverview);
}
if (scheduleCageFilterEl) {
  scheduleCageFilterEl.addEventListener("change", renderScheduleOverview);
}

// delete from schedule overview
if (scheduleListEl) {
  scheduleListEl.addEventListener("click", async (e) => {
    const btn = e.target.closest("button[data-res-id]");
    if (!btn) return;

    const cageId = btn.getAttribute("data-cage-id");
    const resId = btn.getAttribute("data-res-id");
    if (!cageId || !resId) return;

    const confirmDelete = window.confirm(
      `Remove this reservation from ${cageLabelFromId(cageId)}?`
    );
    if (!confirmDelete) return;

    try {
      await remove(ref(db, "reservations/" + cageId + "/" + resId));
    } catch (err) {
      console.error("Error deleting reservation:", err);
      alert("Error deleting reservation. See console for details.");
    }
  });
}

function renderScheduleOverview() {
  if (!scheduleListEl) return;

  if (!scheduleDateEl || !scheduleDateEl.value) {
    scheduleListEl.innerHTML =
      "<p class='message'>Select a date to see the schedule.</p>";
    return;
  }

  const targetDate = new Date(scheduleDateEl.value + "T00:00:00");
```

```
if (isNaN(targetDate.getTime())) {
  scheduleListEl.innerHTML =
    "<p class='message'>Invalid date selection.</p>";
  return;
}

const filterValue = scheduleCageFilterEl
  ? scheduleCageFilterEl.value || "all"
  : "all";

const rows = [];

Object.keys(reservationsSnapshot).forEach((cageId) => {
  const bucket = reservationsSnapshot[cageId] || {};
  Object.keys(bucket).forEach((resId) => {
    const res = bucket[resId];
    if (!res || typeof res.start !== "number" || typeof res.end !== "number")
      return;

    const s = new Date(res.start);
    const e = new Date(res.end);

    if (!sameDay(s, targetDate)) return;
    if (filterValue !== "all" && String(cageId) !== String(filterValue))
      return;

    rows.push({
      cageId,
      resId,
      name: res.name || "",
      type: res.type || "",
      start: s,
      end: e
    });
  });
});

if (!rows.length) {
  scheduleListEl.innerHTML =
    "<p class='message'>No reservations for this date.</p>";
  return;
}

rows.sort((a, b) => {
  const aId = parseInt(a.cageId, 10);
  const bId = parseInt(b.cageId, 10);
  if (!isNaN(aId) && !isNaN(bId) && aId !== bId) {
    return aId - bId;
  }
  return a.start - b.start;
});

let html = `
  <table class="schedule-table">
    <thead>
      <tr>
        <th>Cage / Lane</th>
        <th>Time</th>
        <th>Type</th>
        <th>Name</th>
        <th>Actions</th>
      </tr>
    </thead>
    <tbody>
`;

rows.forEach((row) => {
  html += `
    <tr>
      <td>${cageLabelFromId(row.cageId)}</td>
      <td>${formatTime(row.start)} – ${formatTime(row.end)}</td>
      <td>${escapeHtml(row.type)}</td>
```

```
        <td>${escapeHtml(row.name)}</td>
        <td>
          <button
            class="btn-delete"
            data-cage-id="${row.cageId}"
            data-res-id="${row.resId}"
          >
            ■■
          </button>
        </td>
      </tr>
    `;
  });

  html += "</tbody></table>";
  scheduleListEl.innerHTML = html;
}

// ========================
// SECTION 4 – WAITLIST
// ========================
const waitNameEl = document.getElementById("waitName");
const waitCageEl = document.getElementById("waitCage");
const waitNotesEl = document.getElementById("waitNotes");
const waitAddBtn = document.getElementById("waitAddBtn");
const waitMessageEl = document.getElementById("waitMessage");
const waitAdminListEl = document.getElementById("waitAdminList");

let waitlistSnapshot = {};

if (waitAddBtn) {
  waitAddBtn.addEventListener("click", async () => {
    const name = (waitNameEl.value || "").trim();
    const cage = (waitCageEl.value || "").trim();
    const notes = (waitNotesEl.value || "").trim();

    if (!name) {
      setMessage(waitMessageEl, "Please enter a name.", "orange");
      return;
    }

    try {
      const wlRef = ref(db, "waitlist");
      await push(wlRef, {
        name,
        cage,
        notes,
        createdAt: Date.now()
      });

      setMessage(waitMessageEl, `Added ${name} to waitlist.`, "limegreen");
      waitNameEl.value = "";
      waitCageEl.value = "";
      waitNotesEl.value = "";
    } catch (err) {
      console.error("Error adding waitlist entry:", err);
      setMessage(waitMessageEl, "Error adding to waitlist.", "red");
    }
  });
}

onValue(ref(db, "waitlist"), (snapshot) => {
  waitlistSnapshot = snapshot.val() || {};
  renderWaitlistAdmin();
});

if (waitAdminListEl) {
  waitAdminListEl.addEventListener("click", async (e) => {
    const useBtn = e.target.closest("button[data-wl-use]");
    const delBtn = e.target.closest("button[data-wl-id]");

    // Use → prefill live cage form
```

```
    if (useBtn) {
      const id = useBtn.getAttribute("data-wl-use");
      const entry = waitlistSnapshot[id];
      if (!entry) return;

      if (liveNameEl) liveNameEl.value = entry.name || "";
      if (liveCageEl && entry.cage) liveCageEl.value = String(entry.cage);
      if (liveTypeEl) liveTypeEl.value = "athlete";

      const removeNow = window.confirm(
        `Use ${entry.name || "this athlete"} and remove from waitlist?`
      );
      if (removeNow) {
        try {
          await remove(ref(db, "waitlist/" + id));
        } catch (err) {
          console.error("Error removing waitlist entry:", err);
          alert("Used entry, but error removing from waitlist.");
        }
      }
      return;
    }

    // Delete
    if (delBtn) {
      const id = delBtn.getAttribute("data-wl-id");
      const entry = waitlistSnapshot[id] || {};
      const ok = window.confirm(
        `Remove ${entry.name || "this entry"} from the waitlist?`
      );
      if (!ok) return;

      try {
        await remove(ref(db, "waitlist/" + id));
      } catch (err) {
        console.error("Error deleting waitlist entry:", err);
        alert("Error deleting waitlist entry. See console.");
      }
    }
  });
}

function renderWaitlistAdmin() {
  if (!waitAdminListEl) return;

  const entries = Object.keys(waitlistSnapshot).map((id) => {
    const e = waitlistSnapshot[id] || {};
    return {
      id,
      name: e.name || "",
      cage: e.cage || "",
      notes: e.notes || "",
      createdAt: e.createdAt || 0
    };
  });

  entries.sort((a, b) => a.createdAt - b.createdAt);

  if (!entries.length) {
    waitAdminListEl.innerHTML = "<p>No one on the waitlist yet.</p>";
    return;
  }

  let html = `
    <table class="schedule-table">
      <thead>
        <tr>
          <th>#</th>
          <th>Name</th>
          <th>Preferred Cage/Lane</th>
          <th>Notes</th>
          <th>Actions</th>
```

```
        </tr>
      </thead>
      <tbody>
  `;

  entries.forEach((entry, index) => {
    const cageLabel = entry.cage ? cageLabelFromId(entry.cage) : "";
    html += `
      <tr>
        <td>${index + 1}</td>
        <td>${escapeHtml(entry.name)}</td>
        <td>${escapeHtml(cageLabel)}</td>
        <td>${escapeHtml(entry.notes)}</td>
        <td>
          <button class="btn-secondary" data-wl-use="${entry.id}">Use</button>
          <button class="btn-delete" data-wl-id="${entry.id}">■■</button>
        </td>
      </tr>
    `;
  });

  html += "</tbody></table>";
  waitAdminListEl.innerHTML = html;
}

// Initial draw
renderScheduleOverview();
renderWaitlistAdmin();
```

File: style.css

```
/* Base */
*,
*::before,
*::after {
  box-sizing: border-box;
}

body {
  margin: 0;
  font-family: system-ui, -apple-system, BlinkMacSystemFont, "Segoe UI",
    sans-serif;
  background: radial-gradient(circle at top, #111827 0, #020617 55%);
  color: #f9fafb;
}

/* HERO HEADER */
.hero {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 12px 20px;
  border-bottom: 1px solid #1f2937;
  background: linear-gradient(90deg, #7f1d1d, #991b1b);
}

.hero-left {
  display: flex;
  align-items: center;
  gap: 10px;
}

/* ■ NEW LOGO SETTINGS — NO SQUISHING */
.hero-logo {
  max-height: 72px;    /* adjust to 80–90px if you want it larger */
  height: auto;
  width: auto;
  object-fit: contain;
  flex-shrink: 0;
  filter: drop-shadow(0 4px 10px rgba(0,0,0,0.6));
}

.hero-left h1 {
  margin: 0;
  font-size: 1.4rem;
  letter-spacing: 0.12em;
  text-transform: uppercase;
}

.hero-left p {
  margin: 2px 0 0;
  font-size: 0.9rem;
  color: #fde68a;
}

.hero-right {
  text-align: right;
  display: flex;
  flex-direction: column;
  gap: 4px;
  align-items: flex-end;
}

/* OPEN/CLOSED BADGE */
.open-status {
  display: inline-flex;
  align-items: center;
  gap: 6px;
  padding: 4px 10px;
```

```
  border-radius: 999px;
  background: rgba(15, 23, 42, 0.6);
  font-size: 0.8rem;
}

.status-dot {
  width: 10px;
  height: 10px;
  border-radius: 999px;
  background: #facc15;
  box-shadow: 0 0 0 4px rgba(250, 204, 21, 0.2);
}

.status-dot--open {
  background: #22c55e;
  box-shadow: 0 0 0 4px rgba(34, 197, 94, 0.25);
}

.status-dot--closed {
  background: #dc2626;
  box-shadow: 0 0 0 4px rgba(220, 38, 38, 0.25);
}

.last-updated {
  font-size: 0.75rem;
  color: #e5e7eb;
  opacity: 0.8;
}

/* INFO STRIP */
.info-strip {
  display: flex;
  flex-wrap: wrap;
  gap: 8px;
  padding: 8px 20px 12px;
  border-bottom: 1px solid #111827;
  background: #0b1120;
}

.info-pill {
  font-size: 0.8rem;
  padding: 4px 10px;
  border-radius: 999px;
  background: rgba(15, 23, 42, 0.85);
  border: 1px solid #1f2937;
  color: #e5e7eb;
}

/* BOARD LAYOUT */
.board {
  max-width: 1200px;
  margin: 16px auto 24px;
  padding: 0 16px 24px;
}

/* GRID OF CARDS */
.cards-grid {
  display: grid;
  grid-template-columns: repeat(4, minmax(0, 1fr));
  gap: 14px;
  margin-bottom: 18px;
}

@media (max-width: 1024px) {
  .cards-grid {
    grid-template-columns: repeat(3, minmax(0, 1fr));
  }
}

@media (max-width: 768px) {
  .cards-grid {
    grid-template-columns: repeat(2, minmax(0, 1fr));
```

```
  }
}

@media (max-width: 520px) {
  .cards-grid {
    grid-template-columns: 1fr;
  }
}

/* INDIVIDUAL CARD BASE */
.card {
  background: #0f172a;
  border-radius: 18px;
  padding: 10px 10px 12px;
  border: 2px solid #22c55e;
  box-shadow: 0 18px 40px rgba(0, 0, 0, 0.65);
  display: flex;
  flex-direction: column;
  gap: 6px;
  min-height: 120px;
}

/* CARD VARIANTS */
.card--busy {
  background: radial-gradient(circle at top left, #b91c1c, #7f1d1d 55%, #0f172a);
}

.card--reserved {
  background: radial-gradient(circle at top left, #1d4ed8, #1e293b 55%, #020617);
}

.card--open {
  background: radial-gradient(circle at top left, #0f172a, #020617 60%);
}

/* TITLES */
.card-title {
  font-size: 1rem;
  font-weight: 700;
  letter-spacing: 0.08em;
  text-transform: uppercase;
}

.card-status {
  font-size: 0.75rem;
  letter-spacing: 0.24em;
  text-transform: uppercase;
  opacity: 0.85;
}

/* NAME/ATHLETE ROW */
.card-meta-row {
  display: flex;
  justify-content: space-between;
  align-items: center;
  gap: 6px;
}

.card-name {
  font-weight: 600;
  font-size: 0.9rem;
}

.card-next {
  font-size: 0.8rem;
  color: #e5e7eb;
  opacity: 0.9;
}

/* PILLS */
.type-pill {
  border-radius: 999px;
```

```
  padding: 2px 8px;
  font-size: 0.7rem;
  text-transform: uppercase;
  letter-spacing: 0.08em;
}

.type-pill--athlete {
  background: rgba(16, 185, 129, 0.18);
  color: #6ee7b7;
}

.type-pill--team {
  background: rgba(236, 72, 153, 0.18);
  color: #f9a8d4;
}

.type-pill--trainer {
  background: rgba(59, 130, 246, 0.18);
  color: #93c5fd;
}

/* TIMER BAR */
.card-timer-row {
  display: flex;
  justify-content: space-between;
  align-items: center;
  gap: 6px;
  font-size: 0.8rem;
}

.timer-label {
  opacity: 0.85;
}

.timer-bar-wrap {
  flex: 1;
  height: 8px;
  border-radius: 999px;
  background: rgba(15, 23, 42, 0.9);
  overflow: hidden;
  border: 1px solid rgba(55, 65, 81, 0.9);
}

.timer-bar-fill {
  height: 100%;
  width: 0%;
  background: linear-gradient(
    90deg,
    #22c55e,
    #facc15,
    #f97316,
    #dc2626
  );
  transition: width 0.25s linear;
}

/* WAITLIST */
.waitlist-section {
  margin-top: 8px;
}

.waitlist-card {
  background: #020617;
  border-radius: 18px;
  border: 2px solid #22c55e;
  padding: 10px 12px 12px;
  box-shadow: 0 18px 40px rgba(0, 0, 0, 0.65);
}

.waitlist-header {
  display: flex;
  justify-content: space-between;
```

```
  align-items: baseline;
  gap: 8px;
}

.waitlist-header h2 {
  margin: 0;
  font-size: 0.95rem;
  letter-spacing: 0.18em;
  text-transform: uppercase;
}

.waitlist-tagline {
  font-size: 0.75rem;
  color: #9ca3af;
}

.waitlist-body {
  border-top: 1px solid #1f2937;
  margin-top: 6px;
  padding-top: 6px;
  font-size: 0.8rem;
}

.waitlist-row {
  display: flex;
  flex-direction: column;
  padding: 4px 0;
  border-bottom: 1px dashed #1f2937;
}

.waitlist-row:last-child {
  border-bottom: none;
}

.waitlist-name {
  font-weight: 600;
}

.waitlist-meta {
  font-size: 0.75rem;
  color: #9ca3af;
  display: flex;
  flex-wrap: wrap;
  gap: 6px;
  margin-top: 2px;
}

.waitlist-chip {
  border-radius: 999px;
  padding: 2px 8px;
  border: 1px solid #4b5563;
}

.waitlist-notes {
  opacity: 0.9;
}

.waitlist-empty {
  font-size: 0.8rem;
  color: #9ca3af;
  padding-top: 4px;
}

/* FOOTER */
.footer-note {
  text-align: center;
  font-size: 0.8rem;
  color: #9ca3af;
  padding: 8px 12px 14px;
  border-top: 1px solid #0b1120;
  background: #020617;
}
```

```
/* RESPONSIVE */
@media (max-width: 640px) {
  .hero {
    flex-direction: column;
    align-items: flex-start;
    gap: 8px;
  }

  .hero-right {
    align-items: flex-start;
  }
}
/* ============================
   Admin – Waitlist buttons
   ============================ */

/* "Use" button in waitlist Actions column */
.btn-secondary {
  display: inline-flex;
  align-items: center;
  justify-content: center;
  padding: 4px 12px;
  border-radius: 999px;
  border: none;
  font-size: 0.75rem;
  font-weight: 600;
  cursor: pointer;
  background: #22c55e;      /* bright green */
  color: #020617;           /* near-black text */
  transition: background 0.15s ease, transform 0.1s ease;
  white-space: nowrap;
}

.btn-secondary:hover {
  background: #16a34a;
  transform: translateY(-1px);
}

.btn-secondary:active {
  transform: translateY(0);
}

/* Trash button in waitlist (keep consistent) */
.btn-delete {
  display: inline-flex;
  align-items: center;
  justify-content: center;
  padding: 4px 10px;
  border-radius: 999px;
  border: 1px solid #475569;
  background: transparent;
  color: #e2e8f0;
  font-size: 0.75rem;
  cursor: pointer;
  margin-left: 6px;
  transition: background 0.15s ease, color 0.15s ease, border-color 0.15s ease;
}

.btn-delete:hover {
  background: #ef4444;
  border-color: #ef4444;
  color: #0b1120;
}

/* Make actions column nice and centered */
.waitlist-section table td:last-child,
.waitlist-section table th:last-child {
  text-align: center;
  white-space: nowrap;
}
```

File: service-worker.js

```
// service-worker.js
const CACHE_NAME = "dugout-dashboard-v1";

const ASSETS_TO_CACHE = [
  "/",              // root
  "/index.html",
  "/style.css",
  "/app.js",
  "/manifest.json",
  "/icons/icon-192.png",
  "/icons/icon-512.png",
  "/icons/apple-touch-icon.png",
  "/favicon.png"
];

// INSTALL: cache core assets
self.addEventListener("install", (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME).then((cache) => {
      return cache.addAll(ASSETS_TO_CACHE);
    })
  );
});

// ACTIVATE: clean up old caches
self.addEventListener("activate", (event) => {
  event.waitUntil(
    caches.keys().then((keys) =>
      Promise.all(
        keys
          .filter((key) => key !== CACHE_NAME)
          .map((key) => caches.delete(key))
      )
    )
  );
});

// FETCH: cache-first for static files, network fallback
self.addEventListener("fetch", (event) => {
  const req = event.request;

  // Only handle GET requests
  if (req.method !== "GET") return;

  event.respondWith(
    caches.match(req).then((cachedRes) => {
      if (cachedRes) {
        return cachedRes;
      }
      return fetch(req);
    })
  );
});
```

File: migrate-reservations.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Dugout Reservation Migration</title>
</head>
<body style="background:#020617;color:#e5e7eb;font-family:system-ui;padding:20px;">
  <h1>Dugout Reservation Migration</h1>
  <p>This will COPY old <code>/reservations</code> data into <code>/reservationsByCage</code>.</p>
  <p><strong>It does NOT delete anything.</strong></p>
  <button id="runBtn" style="padding:10px 18px;border-
radius:999px;border:none;background:#22c55e;color:#020617;font-weight:600;cursor:pointer;">Run
Migration</button>
  <pre id="log" style="margin-top:20px;white-space:pre-wrap;"></pre>

  <script type="module">
    import { initializeApp } from "https://www.gstatic.com/firebasejs/11.0.1/firebase-app.js";
    import {
      getDatabase,
      ref,
      get,
      update
    } from "https://www.gstatic.com/firebasejs/11.0.1/firebase-database.js";

    const firebaseConfig = {
      apiKey: "AIzaSyDhuSPwW2M2SyRpOyWMq36ze9G5fzU8zH8",
      authDomain: "dugout-cage-dashboard.firebaseapp.com",
      databaseURL: "https://dugout-cage-dashboard-default-rtdb.firebaseio.com",
      projectId: "dugout-cage-dashboard",
      storageBucket: "dugout-cage-dashboard.firebasestorage.app",
      messagingSenderId: "373269622046",
      appId: "1:373269622046:web:064221adde4ea632636d0f"
    };

    const app = initializeApp(firebaseConfig);
    const db = getDatabase(app);

    const logEl = document.getElementById("log");
    const btn = document.getElementById("runBtn");

    function log(msg) {
      logEl.textContent += msg + "\\n";
    }

    btn.addEventListener("click", async () => {
      btn.disabled = true;
      log("Starting migration...");

      try {
        const snap = await get(ref(db, "reservations"));
        if (!snap.exists()) {
          log("No /reservations node found. Nothing to migrate.");
          return;
        }

        const data = snap.val() || {};
        const updates = {};
        let count = 0;

        Object.keys(data).forEach((cageId) => {
          const bucket = data[cageId] || {};
          Object.keys(bucket).forEach((resId) => {
            const resObj = bucket[resId];
            updates[`reservationsByCage/${cageId}/${resId}`] = resObj;
            count++;
          });
        });
```

```
        if (count === 0) {
          log("No reservation entries found under /reservations.");
          return;
        }

        log(`Prepared ${count} entries. Writing to /reservationsByCage...`);
        await update(ref(db), updates);
        log("Migration complete. Old data is now also under /reservationsByCage.");
      } catch (err) {
        console.error(err);
        log("Error: " + err.message);
      }
    });
  </script>
</body>
</html>
```

File: manifest.json

```
{
  "name": "The Dugout Training Facility",
  "short_name": "Dugout",
  "start_url": "/index.html",
  "scope": "/",
  "display": "standalone",
  "orientation": "portrait",
  "background_color": "#000000",
  "theme_color": "#000000",

  "icons": [
    {
      "src": "/icons/dugout-app-192-transparent.png",
      "sizes": "192x192",
      "type": "image/png",
      "purpose": "any maskable"
    },
    {
      "src": "/icons/dugout-app-512-transparent.png",
      "sizes": "512x512",
      "type": "image/png",
      "purpose": "any maskable"
    }
  ]
}
```