LANDING PAGE BUILD GUIDE

How to Build a Premium Animated Landing Page

The floating-card aesthetic, the GPU-cheap motion system, and the transparent 3D hero loop — the exact recipe behind a real, shipped landing page.

A complete, copy-paste-ready playbook — reverse-engineered from a live page.

This is the real playbook. Not "use a template." The page you liked is built from a handful of small, deliberate techniques layered on top of each other — a quiet design system, a GPU-cheap motion layer, and one show-off element (a transparent 3D character loop) that does the heavy lifting.

You can rebuild all of it with plain HTML/CSS and ~60 lines of JavaScript. No animation library. No page builder. Every snippet here is production code, generalized so you can point it at **your** brand — not clone someone else's.

One rule before you start: design *from* your brand, not from this page. Pick your own color, your own type, your own hero. The techniques transfer; the taste should be yours.

The recipe at a glance

A page feels "premium animated" when five layers stack up:

1. **A tight design system** — a few tokens (color, type, easing, shadow) used *everywhere*, nothing hardcoded.
2. **A soft, living background** — a fixed gradient "aura" with a faint grid, so the page never reads as a flat white sheet.
3. **Floating cards** — content sits on white cards with one well-tuned shadow, lifted off the background.
4. **A motion system** — three cheap effects (pointer parallax, idle float, scroll-reveal) that make everything feel alive without being busy.
5. **One hero moment** — a single high-effort element (here: a transparent 3D loop) that earns the visit. Everything else is restraint; this is the splurge.

Get these five right and the page feels expensive. Let's build each one.

Layer 1 — The design system (do this first)

Premium pages are *consistent*, and consistency comes from never typing a raw value twice. Define everything once as CSS custom properties, then reference the variables. Five categories cover a landing page:

```css
:root {
  /* COLOR — pick ONE accent and use it as a signal, not a paint bucket */
  --color-paper:      #ffffff;   /* page canvas */
  --color-surface:    #f6f6f8;   /* elevated gray */
  --color-ink:        #15151a;   /* near-black text */
  --color-ink-muted:  #6e6e78;   /* secondary text */
  --color-accent:     #f2541b;   /* your ONE signal color */
  --color-border:     #e8e8ec;   /* hairline */

  /* TYPE — one display face, one mono (mono only for code/data moments) */
  --font-display: "General Sans", ui-sans-serif, system-ui, sans-serif;
  --font-data:    "JetBrains Mono", ui-monospace, "SF Mono", Menlo, monospace;

  /* EASING — the single most underrated premium signal */
  --ease-out:  cubic-bezier(0.16, 1, 0.3, 1);   /* enters: fast then settle */
  --ease-move: cubic-bezier(0.33, 0, 0.2, 1);   /* movement: smooth both ends */

  /* SHADOW — two-layer, tuned. This is what makes a card "float" */
  --shadow-float:    0 14px 34px -16px rgba(18,18,30,0.18),
                     0 6px 14px -8px  rgba(18,18,30,0.10);
  --shadow-float-lg: 0 40px 80px -30px rgba(18,18,30,0.26),
                     0 14px 28px -16px rgba(18,18,30,0.13);

  --radius-md: 9px;
}
```

The three taste rules baked into these tokens:

- **One accent, used as a signal.** The amber here appears on maybe 1% of the pixels — a logo dot, one word in the headline, a bullet marker. An accent everywhere is an accent nowhere. Primary buttons are *ink* (near-black), never the accent. The accent points; it doesn't shout.
- **Custom easing beats the defaults.** `ease`, `ease-in-out`, and especially `linear` are the tell of an amateur page. `cubic-bezier(0.16, 1, 0.3, 1)` — fast out of the gate, gentle landing — is most of why expensive sites *feel* expensive. Steal these two curves.
- **Two-layer shadows, pushed up with negative spread.** A single `box-shadow` looks like a sticker. A tight close shadow + a soft far shadow, both pulled in with negative spread (`-16px`), reads like a real object resting on a surface.

Free, excellent fonts to start from: **General Sans** and **Cabinet Grotesk** (fontshare.com), **Inter** and **JetBrains Mono** (Google Fonts).

Layer 2 — The living background ("aura")

A flat white background is the difference between "page" and "product." The fix is a *fixed* (non-scrolling) background made of three stacked pieces: soft color blooms, a faint blueprint grid that fades at the edges, and a couple of blurred color blobs.

```css
.lp-aura {
  position: fixed; inset: 0; z-index: -2; pointer-events: none;
  background:
    radial-gradient(58% 44% at 50% -6%,  rgba(242,84,27,0.10), transparent 62%),
    radial-gradient(40% 38% at 88% 66%,  rgba(242,84,27,0.055), transparent 64%),
    linear-gradient(180deg, #ffffff 0%, #fbfbfd 42%, #f6f6fa 100%);
}

/* faint grid, masked so it dissolves toward the edges */
.lp-aura::before {
  content: ""; position: absolute; inset: 0;
  background-image:
    linear-gradient(rgba(20,20,45,0.035) 1px, transparent 1px),
    linear-gradient(90deg, rgba(20,20,45,0.035) 1px, transparent 1px);
  background-size: 46px 46px;
  -webkit-mask-image: radial-gradient(86% 60% at 50% 26%, #000 0%, transparent 76%);
          mask-image: radial-gradient(86% 60% at 50% 26%, #000 0%, transparent 76%);
}

/* two blurred accent blobs for depth */
.lp-aura i { position: absolute; border-radius: 50%; filter: blur(64px); opacity: .55; }
.lp-aura i:first-child { top: -60px; left: 50%; width: 540px; height: 360px;
  transform: translateX(-50%);
  background: radial-gradient(circle, rgba(242,84,27,0.16), transparent 70%); }
.lp-aura i:last-child  { bottom: 8%; right: -60px; width: 420px; height: 420px;
  background: radial-gradient(circle, rgba(242,84,27,0.10), transparent 70%); }
```

```html
<div class="lp-aura" aria-hidden="true"><i></i><i></i></div>
```

Why it works: the grid gives subliminal structure (it reads as "engineered"), the mask makes it feel deliberate rather than wallpapered, and the blurred blobs add depth the eye can't quite name. Keep all of it at very low opacity — you should feel it, not see it. It's `position: fixed`, so it sits still while content scrolls over it. That parallax-by-default is half the effect for zero JavaScript.

Layer 3 — Floating cards

Every piece of content — a mockup, a code snippet, a pricing tier — sits on a white card lifted off the aura with `--shadow-float`. That's the whole "floating" look.

```css
.card {
  background: #fff;
  border: 1px solid var(--color-border);
  border-radius: 16px;
  box-shadow: var(--shadow-float);
}

/* lift on hover — note the custom easing and the deeper shadow swap */
.lift-card { transition: box-shadow .32s var(--ease-move),
                         transform .32s var(--ease-move); }
.lift-card:hover { transform: translateY(-4px); box-shadow: var(--shadow-float-lg); }
```

Generous radius (14–18px), a hairline border so the white card still has an edge against white, and the two-layer shadow. That's it. The restraint is the point: the cards are plain so the *motion* and the *hero* can be the show.

Layer 4 — The motion system (the part people ask about)

Three effects, all GPU-cheap, all driven by one tiny "motion island" script. The trick that keeps it fast: **JavaScript only writes two numbers** (the damped pointer position), and **CSS does all the actual moving** via custom properties. No per-frame style thrash, no layout, no library.

4a — Pointer parallax

As the mouse moves, floating elements drift opposite the cursor, each at its own depth. This is what gives the hero that subtle 3D "I could reach into it" feel.

The script publishes a damped pointer position as `--mx` / `--my` on `<html>`:

```js
// motion-island.js  — runs once, renders nothing
const root = document.documentElement;
const reduce = matchMedia("(prefers-reduced-motion: reduce)").matches;

let tx = 0, ty = 0, cx = 0, cy = 0, raf = 0;
const tick = () => {
  cx += (tx - cx) * 0.07;            // 0.07 = how "heavy"/damped the follow is
  cy += (ty - cy) * 0.07;
  root.style.setProperty("--mx", cx.toFixed(4));
  root.style.setProperty("--my", cy.toFixed(4));
  raf = (Math.abs(tx - cx) > 5e-4 || Math.abs(ty - cy) > 5e-4)
        ? requestAnimationFrame(tick) : 0;   // stops itself when settled
};
const onMove = (e) => {
  tx = (e.clientX / innerWidth  - 0.5) * 2;   // -1 .. 1
  ty = (e.clientY / innerHeight - 0.5) * 2;
  if (!raf) raf = requestAnimationFrame(tick);
};
if (!reduce) addEventListener("pointermove", onMove, { passive: true });
```

Then each floating element reads those numbers in pure CSS. `--d` is its **depth**: bigger number = moves more = feels closer.

```
.par {
  transform: translate3d(
    calc(var(--mx,0) * var(--d,6) * -1px),
    calc(var(--my,0) * var(--d,6) * -1px), 0);
  will-change: transform;
}
```

```
<div class="par" style="--d: 6">  …near-ish card…  </div>
<div class="par" style="--d: 11"> …closest card…   </div>
```

Two details that make it feel pro, not janky:

- **Damping** (`* 0.07` toward the target each frame) — the layers *chase* the cursor with weight instead of snapping to it.
- **Self-halting rAF** — the animation loop stops once movement settles, so the page isn't burning a frame loop while idle.

For an even stronger 3D read, wrap the hero in a tilt container that rotates slightly with the pointer:

```
.stage-tilt {
  transform-style: preserve-3d;
  transform: rotateY(calc(var(--mx,0) * 4deg))
             rotateX(calc(var(--my,0) * -4deg));
  transition: transform .2s var(--ease-move);
}
```

4b — Idle float (bob)

Floating cards gently bob up and down on their own, so the page breathes even when the mouse is still. Stagger the durations so cards don't bob in unison (unison reads as mechanical).

```
@media (prefers-reduced-motion: no-preference) {
  .bob { animation: lp-bob var(--bobdur, 7s) var(--ease-move) infinite;
         will-change: transform; }
}
@keyframes lp-bob {
  0%, 100% { transform: translateY(0); }
  50%      { transform: translateY(-8px); }
}
```

```
<div class="par" style="--d: 8"><div class="bob" style="--bobdur: 6.6s">…</div></div>
```

Nest it *inside* the `.par` element so float and parallax compose cleanly (`.par` owns the parallax transform, `.bob` owns the idle transform — separate elements, no conflict).

4c — Scroll-reveal

Sections fade and rise into place as they enter the viewport — once each, then they're left alone. Built on `IntersectionObserver`, ~15 lines:

```
const els = document.querySelectorAll("[data-reveal]");
if (reduce) {
  els.forEach(el => el.classList.add("in"));          // no-motion: just show
} else {
  const io = new IntersectionObserver((entries) => {
    for (const e of entries) if (e.isIntersecting) {
      e.target.classList.add("in");
      io.unobserve(e.target);                          // reveal once, then forget
    }
  }, { threshold: 0.12, rootMargin: "0px 0px -6% 0px" });
  els.forEach(el => io.observe(el));
}
root.classList.add("fx-ready");                        // gates the hidden state
```

```
/* hidden state ONLY applies once JS is alive — so no-JS users see everything */
.fx-ready [data-reveal] {
  opacity: 0; transform: translateY(22px);
  transition: opacity .7s var(--ease-out), transform .7s var(--ease-out);
}
.fx-ready [data-reveal].in { opacity: 1; transform: none; }

/* stagger the hero's lines so they cascade in */
.lp-hero [data-reveal]:nth-child(2) { transition-delay: .06s; }
.lp-hero [data-reveal]:nth-child(3) { transition-delay: .12s; }
.lp-hero [data-reveal]:nth-child(4) { transition-delay: .18s; }
```

```
<h1   data-reveal>Your headline</h1>
<p    data-reveal>Your subhead</p>
<div  data-reveal>Your CTA</div>
```

The `fx-ready` gate is the important bit: the "hidden before reveal" CSS only exists once the script has run. If JavaScript fails or is disabled, nothing is ever hidden — the page degrades to fully visible. **Never hide content behind an animation that might not run.**

The non-negotiable: `prefers-reduced-motion`

Notice every effect above checks it. Parallax and float **don't attach** when a visitor has reduced motion on; reveals show instantly; the hero swaps to a still image (next section). This isn't optional polish — it's an accessibility requirement, and it's three lines. Premium pages respect the setting.

```
@media (prefers-reduced-motion: reduce) {
  /* turn off anything that moves; show static fallbacks */
}
```

Layer 5 — The transparent 3D hero loop (the splurge)

This is the element that makes people DM you. It's a short, **seamless, transparent** 3D character animation that floats directly on the page — no video box, no background seam, just the character living in your layout.

There are two hard problems most people get wrong: making it loop seamlessly, and making the background actually transparent. Here's how each was solved.

Step 1 — Generate the character on a solid white background

Use any 3D / generative video tool (Higgsfield, Runway, Sora, Kling, or a real 3D render). Generate the character against a **pure, uniform white** background — not a scene, not a gradient. The uniform white is what makes clean cutout possible later. Render at the highest resolution you can; you'll scale down.

Step 2 — Make it loop seamlessly (the start = end trick)

A loop only feels seamless if the **first frame and last frame are identical**. The reliable way to guarantee that:

- Lock one frame as your **anchor** (a neutral resting pose).
- Make that anchor *both* the start frame and the end frame of the generation.
- Any "event" in the loop (a reaction, an object appearing) must fully resolve back to the anchor — and **enter from offscreen / behind the character**, so its appearance and disappearance don't pop at the loop seam.
- Keep the **camera locked** (no pan/zoom). Tell the model plainly *not* to be "cinematic" — that instruction is what makes generators add drifting camera moves that destroy a loop.

Review the whole loop as a contact sheet (all frames in one image) before you commit — it's the fastest way to catch a pose that won't seam.

Step 3 — Cut out the background to true transparency

Generators won't give you alpha. You make it. The naive approach — a chroma/ white key — punches holes through any *light* parts of your subject. The robust technique exploits the uniform white you generated on:

1. Explode the video to PNG frames.

2. For each frame, build an alpha matte with a **luma key off the white background**, gated to the subject's vicinity so light parts of the subject survive:

   ```
   alpha = clip( (248 - max(R,G,B)) / 26 )
   ```

   (Pixels near white → transparent; the subject → opaque. Tune the `248` / `26` to your footage.)

3. If your subject *is* white/light, an AI matting model (rembg `u2net` works on CPU; BiRefNet is higher quality but wants a GPU) is the fallback — but the luma key off uniform white gives crisper edges in practice.

Step 4 — Encode for the web with the alpha channel intact

No single video format with alpha plays everywhere. You ship **two**, plus a poster, and let the browser pick:

- **VP9-alpha WebM** for Chrome / Firefox / Edge
- **HEVC-alpha MOV** for Safari
- a **transparent PNG poster** for first paint + the reduced-motion fallback

```
# scale the FRAMES first — adding -vf scale on the VP9 encode strips the alpha
# 1) WebM (Chrome/FF/Edge):
ffmpeg -framerate 24 -i frame_%04d.png \
  -c:v libvpx-vp9 -pix_fmt yuva420p -b:v 0 -crf 30 hero-loop.webm

# 2) MOV (Safari, HEVC w/ alpha, hardware encoder on macOS):
ffmpeg -framerate 24 -i frame_%04d.png \
  -c:v hevc_videotoolbox -alpha_quality 0.9 -vtag hvc1 -pix_fmt bgra hero-loop.mov
```

> **Two traps that will waste an afternoon:**
>
> 1. An inline `-vf scale=...` on the **VP9-alpha** encode silently strips the alpha channel. Pre-scale the PNG frames, *then* encode.
> 2. `ffprobe` / `ffmpeg` will **lie** about whether the alpha survived (HEVC alpha often probes as `yuv420p`). Don't trust the probe — **verify by opening it in an actual browser.**

A full transparent ~10s loop lands around **2–3 MB** (WebM) / **7 MB** (MOV) at 1280px wide. That's hero-acceptable.

Step 5 — Drop it in the page

```html
<div class="lp-stage" aria-hidden="true">
  <video class="lp-hero-video" autoplay loop muted playsinline
         preload="metadata" poster="/hero-loop-poster.png">
    <source src="/hero-loop.webm" type="video/webm">
    <source src="/hero-loop.mov"  type="video/quicktime">
  </video>
  <!-- shown instead when the visitor prefers reduced motion -->
  <img class="lp-hero-video-still" src="/hero-loop-poster.png" alt="">
</div>
```

```css
.lp-stage { position: relative; height: 560px; perspective: 1500px; }
.lp-hero-video, .lp-hero-video-still {
  position: absolute; inset: 0; width: 100%; height: 100%;
  object-fit: contain; transform: scale(1.28); /* fill the stage nicely */
}
.lp-hero-video-still { display: none; }

@media (prefers-reduced-motion: reduce) {
  .lp-hero-video       { display: none; }
  .lp-hero-video-still { display: block; }   /* swap to the still */
}
```

`autoplay loop muted playsinline` is the exact combination browsers require to autoplay (muted + playsinline is mandatory on iOS). The poster paints instantly while the video streams in, and doubles as the reduced-motion still — one asset, two jobs.

> **No budget for 3D?** This same stage works with a looping product UI mockup, an animated SVG, a Lottie file, or even a tasteful CSS-only scene. The floating-stage + parallax treatment is what sells it — the content inside can start humble and level up later.

Putting it together — the page skeleton

```
<body>
  <div class="lp-aura" aria-hidden="true"><i></i><i></i></div>

  <nav class="lp-nav"> … sticky, frosted (backdrop-filter: blur) … </nav>

  <header class="lp-hero">
    <div class="lp-hero-copy">
      <h1  data-reveal>Headline with <span class="sig">one accent word</span></h1>
      <p   data-reveal>Subhead.</p>
      <div data-reveal><a class="lp-cta" href="#">Primary CTA</a></div>
    </div>
    <div class="lp-stage" data-reveal aria-hidden="true">
      <!— the transparent hero loop —>
    </div>
  </header>

  <section class="lp-section" data-reveal>
    <div class="par" style="--d: 6"><div class="bob"> <div class="card">…</div> </div></div>
  </section>
  <!— repeat: one idea per section, alternating sides —>

  <footer> … </footer>

  <script type="module" src="/motion-island.js"></script>
</body>
```

Structural choices worth copying:

- **Sticky frosted nav** — `position: sticky; backdrop-filter: saturate(180%) blur(14px); background: rgba(255,255,255,.72)`. The blur over the moving aura is a premium tell.
- **One idea per section, sides alternating.** Copy left / visual right, then flip. It gives rhythm and keeps each section a single, clear thought.
- **Headline with exactly one accent word** in your signal color (a gradient text-fill looks great here). One word — not the whole line.
- **Primary buttons in ink, not the accent.** Let the accent stay a pointer.

Performance & accessibility checklist

The animations are designed to be cheap. Keep them that way:

- [ ] Animate **only** `transform` **and** `opacity` — never `top` / `left` / `width` (those trigger layout on every frame). Everything above obeys this.
- [ ] JS writes **two CSS variables**; CSS does the moving. No per-frame inline style churn.

- [ ] The pointer-parallax rAF loop **stops itself** when motion settles.
- [ ] `will-change: transform` on the moving layers (but only those).
- [ ] Every motion effect checks `prefers-reduced-motion` and has a static fallback.
- [ ] Content is **never hidden behind an animation that might not run** (the `fx-ready` gate).
- [ ] Hero video: `muted playsinline` (required for autoplay), `poster` for instant first paint, `preload="metadata"` (not the whole file).
- [ ] Hero loop served as **WebM + MOV + poster**, verified in a real browser.
- [ ] On mobile, consider **hiding the heaviest effects** (the original hides the hero loop and the pointer-parallax accents on small screens — touch has no cursor, and you save the bytes).

The mindset, in one line

Restraint everywhere, splurge in one place. A quiet system — one accent, one shadow, two easing curves, three cheap motion effects — makes a single high-effort hero look even more expensive by contrast. Build the system first; spend your effort budget on the one thing people will remember.

Now go build it for *your* brand. Pick your own color, your own type, your own hero. The techniques are universal — the taste should be yours.

The page this guide reverse-engineers is speedydebug.com. Now go build something great.