Gradient Text Hover Effects

Here's how you can style your `.hero-title` on a black background to have a **colorful gradient effect**, similar to the image you uploaded. It will include:

- A **text gradient**
- **Smooth hover effect** (reverse gradient or glow)
- A **unique shimmer animation** for added flair

🧩 CSS Code:

```css
.hero-title {
  font-size: 2.8rem;
  font-weight: 700;
  width: 500px;
  margin-bottom: 1rem;
  background: linear-gradient(90deg, #10b981, #3b82f6, #8b5cf6);
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
  background-clip: text;
  text-fill-color: transparent;
  transition: background-position 0.5s ease;
  background-size: 200% auto;
  animation: shimmer 3s infinite linear;
}

/* On hover: reverse gradient direction and add glow */
.hero-title:hover {
  background-position: right center;
  text-shadow: 0 0 10px rgba(139, 92, 246, 0.6),
               0 0 20px rgba(59, 130, 246, 0.4);
}

/* Optional shimmer effect */
@keyframes shimmer {
  0% {
```

```
    background-position: 0% center;
  }
  100% {
    background-position: 200% center;
  }
}
```

✅ HTML Example:

```html
<div class="hero-title">Because your growth can’t wait</div>
```

🖤 Background:

Make sure the parent container or body has a dark background:

```css
body {
  background-color: #0f0f0f; /* or pure black #000 */
}
```

Would you like it to be animated continuously, or only shimmer once on hover? I can adjust it further.

To create an **infinite text-typing animation** that types one sentence at a time (with only one visible at a time), you can use **React + CSS animations** or **React state logic**. Below is a clean and reusable solution using **React** hooks and CSS.

✅ Final Result:

- One line appears with typing effect.
- After it's fully typed, it disappears and next line types.
- The cycle repeats infinitely.

🔧 React + CSS Implementation:

1. React Component (JSX)

jsx

```jsx
import React, { useEffect, useState } from 'react';
import './HeroTyping.css';

const lines = [
  "We connect high-potential SMEs to flexible, founder-first capital.",
  "No equity dilution. No pitching. Just pure growth.",
  "HNIs gain access to vetted, data-backed deals from India’s emerging business hubs."
];

const HeroTyping = () => {
  const [currentLine, setCurrentLine] = useState(0);
  const [displayText, setDisplayText] = useState('');
  const [charIndex, setCharIndex] = useState(0);

  useEffect(() => {
    const currentSentence = lines[currentLine];

    if (charIndex < currentSentence.length) {
      const timeout = setTimeout(() => {
        setDisplayText(prev => prev + currentSentence[charIndex]);
        setCharIndex(prev => prev + 1);
      }, 50); // Typing speed
      return () => clearTimeout(timeout);
    } else {
      const pause = setTimeout(() => {
        // Reset for next sentence
        setDisplayText('');
        setCharIndex(0);
        setCurrentLine((prev) => (prev + 1) % lines.length);
      }, 2000); // Time to show full line before switching
      return () => clearTimeout(pause);
    }
```

```
  }, [charIndex, currentLine]);

  return (
    <p className="hero-subtext1">{displayText}<span className="cursor">|</span></p>
  );
};

export default HeroTyping;
```

---

2. **CSS for Typing Effect**

css

```
.hero-subtext1 {
  font-size: 1.4rem;
  font-weight: 500;
  color: #ffffff;
  white-space: nowrap;
  overflow: hidden;
  width: 100%;
  max-width: 700px;
  margin: 0 auto;
  text-align: center;
  min-height: 2rem;
}

.cursor {
  display: inline-block;
  width: 1px;
  background-color: #fff;
  animation: blink 1s steps(2, start) infinite;
  margin-left: 2px;
}

@keyframes blink {
  to {
    visibility: hidden;
```

```
  }
}
```

🧪 Optional Enhancements:

- Add a `fade` class for fade-in/out if needed.
- Add `typingSpeed` and `pauseDuration` as props for customization.

Would you like this implemented using pure HTML/CSS/JS without React? I can also provide that version.

Thanks for the reference images. You want the typing effect to look like the **second image** — stylish, full-width, and nicely aligned without extra left margin.

Here's a polished solution that:

- Fixes the **text not fully showing**
- Removes **extra left gap**
- Matches the **modern font and color**
- Uses a **typing effect** just like the image (`We fund gaming companies|`) with only one visible line at a time

---

✅ Final Version (React + CSS)

🧩 `HeroTyping.jsx`

jsx

```jsx
import React, { useEffect, useState } from 'react';
import './HeroTyping.css';

const lines = [
  "We connect high-potential SMEs to flexible, founder-first capital.",
  "No equity dilution. No pitching. Just pure growth.",
  "HNIs gain access to vetted, data-backed deals from India’s emerging business hubs."
];

const HeroTyping = () => {
  const [lineIndex, setLineIndex] = useState(0);
  const [charIndex, setCharIndex] = useState(0);
  const [displayText, setDisplayText] = useState('');

  useEffect(() => {
    const currentLine = lines[lineIndex];

    if (charIndex < currentLine.length) {
      const typingTimeout = setTimeout(() => {
        setDisplayText((prev) => prev + currentLine[charIndex]);
        setCharIndex((prev) => prev + 1);
      }, 50); // Typing speed
      return () => clearTimeout(typingTimeout);
    } else {
      const pauseTimeout = setTimeout(() => {
        // Move to next line
        setDisplayText('');
        setCharIndex(0);
        setLineIndex((prev) => (prev + 1) % lines.length);
      }, 2000); // Pause after full line
      return () => clearTimeout(pauseTimeout);
    }
  }, [charIndex, lineIndex]);

  return (
    <div className="typing-wrapper">
      <p className="hero-typing">
        {displayText}
        <span className="cursor">|</span>
```

```
      </p>
    </div>
  );
};

export default HeroTyping;
```

HeroTyping.css

```css
@import url('https://fonts.googleapis.com/css2?family=Inter:wght@400;700&display=swap');

.typing-wrapper {
  display: flex;
  justify-content: center;
  align-items: center;
  padding: 1rem 1.5rem;
  width: 100%;
  max-width: 100%;
  background-color: #000;
}

.hero-typing {
  font-family: 'Inter', sans-serif;
  font-size: 2rem;
  font-weight: 700;
  color: #ffffff;
  background: linear-gradient(90deg, #10b981, #6366f1);
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
  white-space: nowrap;
  overflow: hidden;
  border-right: 0.15em solid rgba(255, 255, 255, 0.75);
}

.cursor {
```

```
  animation: blink 1s steps(2, start) infinite;
  color: #10b981;
  font-weight: 400;
}

@keyframes blink {
  0%, 100% { opacity: 1; }
  50% { opacity: 0; }
}
```

🖤 Fix for Layout Gap

Make sure there is **no margin/padding** causing left gap in your parent container:

```css
body, html {
  margin: 0;
  padding: 0;
  background-color: #000; /* dark background */
}

section, .hero-section {
  padding: 0;
  margin: 0;
}
```

Would you like this typing effect to **loop only once** or always run infinitely? I can customize that too!

Here's a more concise version of the 3 sentences, with the core meaning preserved:

---

1. We connect high-growth SMEs with flexible, founder-friendly capital.
2. No equity loss. No pitching. Just pure growth.
3. HNIs access vetted, data-driven deals from India's top emerging hubs.

Would you like them to sound even more informal, formal, or technical depending on your audience (e.g., investors, startup founders, or general users)?

Got it! You want the lines to be **very short**, like a **punchy heading**, while keeping the meaning intact.

Here's a **refined and minimalist version** of the three lines:

---

1. Founder-first capital for SMEs.
2. No equity. No pitch. Just growth.
3. HNIs invest in verified deals.

Let me know if you want a more energetic, corporate, or casual tone!

To make the `.hero-subtext` class look more **professional and polished**, we can refine:

- The **font weight** (590 isn't valid — use 500 or 600 instead)
- Use a more **neutral gray** for better contrast
- Adjust **line-height** for readability
- Keep **margins and width** optimized

✅ Updated `.hero-subtext` (Professional Style)

```css
.hero-subtext {
  font-size: 1.125rem; /* Slightly smaller for subtlety */
  color: #a0a4ab;      /* Softer professional gray */
  margin-bottom: 1.5rem;
  max-width: 520px;
  font-weight: 500;    /* 500 = medium weight (more valid and consistent) */
  line-height: 1.6;    /* Improved readability */
}
```

📝 Optional Enhancements:

If you're using a design system like Tailwind or a modern font like Inter:

```css
font-family: 'Inter', sans-serif;
```

Would you like to animate or fade this text on scroll as well?

To make your `.hero-btn` buttons **match the gradient styling** like the heading in the image (`#10b981 → #6366f1`), and apply **gradient hover effects**, here's a complete and professional redesign:

✅ Updated CSS with Gradient & Hover Effects

```css
.hero-btn {
  padding: 0.75rem 1.5rem;
  font-size: 1rem;
```

```
  font-weight: 600;
  border-radius: 8px;
  transition: all 0.3s ease;
  cursor: pointer;
}

/* === Primary Gradient Button === */
.hero-btn.primary {
  background: linear-gradient(90deg, #10b981, #6366f1);
  color: white;
  border: none;
}

.hero-btn.primary:hover {
  background: transparent;
  color: #6366f1;
  border: 2px solid #6366f1;
  transform: scale(1.05);
}

/* === Secondary Outline Gradient Button === */
.hero-btn.secondary {
  background: transparent;
  color: #10b981;
  border: 2px solid transparent;
  border-image: linear-gradient(90deg, #10b981, #6366f1);
  border-image-slice: 1;
}

.hero-btn.secondary:hover {
  background: linear-gradient(90deg, #10b981, #6366f1);
  color: white;
  transform: scale(1.05);
}
```

✨ Optional:

If you're using Tailwind or want this mobile-friendly, I can convert this into utility classes too.

Would you like a subtle shadow or glow effect on hover for extra polish?

Here's a **visually upgraded version** of your `.hero-stats` section with:

- ✅ **Gradient border on hover**
- ✅ **Glow effect**
- ✅ **Subtle elevation**
- ✅ **Modern glassmorphism style maintained**

🌈 Updated CSS with Gradient Hover & Glow

css

```css
.hero-stats {
  display: flex;
  gap: 2rem;
  margin-top: 2.5rem;
  justify-content: flex-start;
  flex-wrap: wrap;
  cursor: pointer;
}

.hero-stat-box {
  position: relative;
  background: rgba(255, 255, 255, 0.15);
  backdrop-filter: blur(12px);
  border-radius: 1rem;
  padding: 1.5rem;
  width: 160px;
  text-align: center;
  box-shadow: 0 8px 20px rgba(0, 0, 0, 0.1);
  transition: all 0.4s ease;
  border: 2px solid transparent;
}
```

```css
/* ✨ Gradient Glow on Hover */
.hero-stat-box:hover {
  transform: translateY(-6px);
  border: 2px solid transparent;
  background: linear-gradient(135deg, #10b98133, #6366f133);
  box-shadow: 0 0 18px rgba(16, 185, 129, 0.5),
              0 0 30px rgba(99, 102, 241, 0.3);
}

/* Icon styling */
.hero-stat-icon {
  font-size: 2rem;
  margin-bottom: 0.5rem;
  color: #10b981; /* teal highlight */
}

/* Main value/stat */
.hero-stat-box h3 {
  font-size: 1.75rem;
  font-weight: 700;
  margin: 0;
  color: #1e293b;
}

/* Label/description */
.hero-stat-box p {
  margin-top: 0.4rem;
  color: #475569;
  font-size: 0.95rem;
}
```

💡 Optional: Animated Border Gradient

If you want an animated **gradient border ring**, I can add a pseudo-element with a rotating or flowing effect.

Let me know if you want:

- Animated gradient ring
- Full gradient-filled card
- Dark mode variant

Would you like icons also animated slightly on hover (bounce, spin, or scale)?

The **Formula 1 website** (F1.com) uses a sleek, modern sans-serif font that looks technical, fast, and bold — perfect for motorsport branding.

✅ F1 Font Style Used

F1.com uses **custom branding fonts**, but you can get a **very close match** using "**Futura PT**", "**Titillium Web**", or "**Formula1 Display**" (the unofficial font used in older branding).

🔧 Recommended Google Fonts (Free & Close Match)

1. **Titillium Web** (Very close, futuristic & clean)

```html
<link href="https://fonts.googleapis.com/css2?family=Titillium+Web:wght@400;600;700&display=swap" rel="stylesheet">
```

CSS:

```css
font-family: 'Titillium Web', sans-serif;
```

---

2. **Orbitron** (More tech-inspired like F1 timing screens)

```html
<link href="https://fonts.googleapis.com/css2?family=Orbitron:wght@400;600;700&display=swap" rel="stylesheet">
```

CSS:

```css
font-family: 'Orbitron', sans-serif;
```

---

3. **Formula1 Display Regular** (Unofficial, closest replica of F1 logo font)

- It's not on Google Fonts, but you can download from free font sites:
  - https://www.dafont.com/formula1.font

CSS:

```css
@font-face {
  font-family: 'Formula1 Display';
  src: url('/fonts/Formula1-Regular.ttf') format('truetype');
}
```

Then use:

```css
font-family: 'Formula1 Display', sans-serif;
```

✅ Best Pick for Modern Web:

Use **Titillium Web** — widely supported, free, clean, and closest to the F1 site style.

Would you like me to apply this to your current `hero-title` and `hero-btn` styles too?

You're nearly there — great job! To **polish the font styling** and make the text inside `.hero-stat-box` look more **professional, readable, and F1-inspired**, here's what to do:

✅ Updated `.hero-stats` & `.hero-stat-box` (Professional, White Text, F1-like)

🧩 CSS:

```css
.hero-stats {
  display: flex;
  gap: 2rem;
  margin-top: 2.5rem;
  justify-content: flex-start;
  flex-wrap: wrap;
  cursor: pointer;
  font-family: 'Titillium Web', sans-serif;
}

.hero-stat-box {
  position: relative;
  background: rgba(255, 255, 255, 0.12);
  backdrop-filter: blur(12px);
  border-radius: 1rem;
  padding: 1.5rem;
  width: 160px;
  text-align: center;
  box-shadow: 0 8px 20px rgba(0, 0, 0, 0.1);
  transition: all 0.4s ease;
  border: 2px solid transparent;
  color: white; /* Makes all text inside white by default */
}

/* Icon style (optional for glowing effect) */
.hero-stat-icon {
```

```
  font-size: 2rem;
  margin-bottom: 0.5rem;
  color: #10b981; /* optional: make this white too */
}

/* Value/Number */
.hero-stat-box h3 {
  font-size: 1.75rem;
  font-weight: 700;
  margin: 0;
  color: #ffffff; /* white */
}

/* Label/description */
.hero-stat-box p {
  margin: 0.25rem 0 0;
  color: #d1d5db; /* soft light gray */
  font-size: 0.95rem;
}
```

📎 Font Link (Add to `<head>` in HTML or in `_document.js` if using Next.js):

```html
<link href="https://fonts.googleapis.com/css2?
family=Titillium+Web:wght@400;600;700&display=swap" rel="stylesheet">
```

💡 Tips:

- Use `color: #d1d5db` for subtitles (a light gray) to create **visual hierarchy**.
- Consider using `text-transform: uppercase;` on the headings if you want a **race-tech feel**.

Would you like the numbers (`h3`) to animate or count up on scroll for a dynamic stat effect?

You're using the awesome `react-countup` library to animate numbers — great choice!

Here's how to make sure it works:

✅ 1. Install the Package

In your React project root:

```bash
npm install react-countup
```

Or if using yarn:

```bash
yarn add react-countup
```

✅ 2. Import `CountUp` Component

At the top of your component file:

```jsx
import CountUp from 'react-countup';
```

✅ 3. Add Google Font for F1-like Styling (Optional)

If you're also using the `Titillium Web` font as discussed earlier, include this in your `index.html` `<head>` (or in `_document.js` if using Next.js):

```html
```

```
<link href="https://fonts.googleapis.com/css2?
family=Titillium+Web:wght@400;600;700&display=swap" rel="stylesheet">
```

💡 Bonus Tip: Animate On Scroll

To trigger the `CountUp` **only when stats scroll into view**, wrap each stat with `VisibilitySensor`:

bash

```
npm install react-visibility-sensor
```

Example:

jsx

```
import CountUp from 'react-countup';
import VisibilitySensor from 'react-visibility-sensor';

<h3>
  <CountUp end={5000} duration={4} separator="," >
    {({ countUpRef, start }) => (
      <VisibilitySensor onChange={start} delayedCall>
        <span ref={countUpRef} />
      </VisibilitySensor>
    )}
  </CountUp>+
</h3>
```

Would you like this scroll-triggered animation version built into all 3 stat boxes for you?

To add the **Formula 1 font**, you have two options:

✅ OPTION 1: **Use the closest official free font – Titillium Web**

(*Free, web-safe, and very close to F1 style*)

🔗 Step 1: Add Google Fonts link (in your HTML or `_document.js`)

```html
<link href="https://fonts.googleapis.com/css2?family=Titillium+Web:wght@400;600;700&display=swap" rel="stylesheet">
```

✅ Step 2: Use it in CSS

```css
.hero-stats,
.hero-title,
.hero-btn,
body {
  font-family: 'Titillium Web', sans-serif;
}
```

🔴 OPTION 2: Use the **Unofficial Formula 1 Display Font**

Formula 1 used to license a custom font called **Formula1 Display Regular** (not on Google Fonts, unofficial but widely used in fan projects).

⚠️ You must **self-host** the font.

✅ Step 1: Download the font:

- From: https://www.dafont.com/formula1.font

Download the `.ttf` file.

✅ Step 2: Place the font file

Put `Formula1-Regular.ttf` inside your `public/fonts/` folder or a similar static assets path.

✅ Step 3: Import and apply the font in CSS

```css
@font-face {
  font-family: 'Formula1 Display';
  src: url('/fonts/Formula1-Regular.ttf') format('truetype');
  font-weight: normal;
  font-style: normal;
}

body,
.hero-title,
.hero-btn {
  font-family: 'Formula1 Display', sans-serif;
}
```

🎯 Recommendation:

Use `Titillium Web` if you want web-safe + fast loading + no setup.
Use `Formula1 Display` if you want **authentic F1 branding style** for headings or logos.

Let me know if you want me to generate the full CSS + folder structure!

Here's a **step-by-step guide** to add the **Formula1 Display font** to your React project (JSX + CSS) and apply it to your `.hero-stats` section.

✅ STEP-BY-STEP: Add Formula 1 Font to JSX & CSS

🟢 Step 1: **Download the Font**

- Go to https://www.dafont.com/formula1.font
- Click "Download"
- Extract the ZIP file
- You'll find: `Formula1-Regular.ttf`

🟢 Step 2: **Place the Font in Your Project**

- Create a `fonts/` folder inside `public/`
  Path: `public/fonts/Formula1-Regular.ttf`

🟢 Step 3: **Define the Font in CSS**

> Add this to your main CSS file (e.g., `App.css`, `HeroStats.css`, or a global CSS file)

```css
@font-face {
  font-family: 'Formula1 Display';
  src: url('/fonts/Formula1-Regular.ttf') format('truetype');
  font-weight: normal;
  font-style: normal;
}
```

🟢 Step 4: Apply the Font to Your Component

In your CSS file:

```css
.hero-stats {
  display: flex;
  gap: 2rem;
  margin-top: 2.5rem;
  justify-content: flex-start;
  flex-wrap: wrap;
  cursor: pointer;
  font-family: 'Formula1 Display', sans-serif;
}
```

Or directly in JSX with inline style (not recommended for large apps):

```jsx
<div className="hero-stats" style={{ fontFamily: 'Formula1 Display, sans-serif' }}>
  ...
</div>
```

🟢 Step 5: Restart Your Dev Server

If the font doesn't show up, restart your development server to ensure the new font file is picked up.

🟣 Optional: Fallback to Google Font

In case Formula1 Display doesn't load:

```css
```

```
font-family: 'Formula1 Display', 'Titillium Web', sans-serif;
```

✅ Final Result

Your stats block will now use the **official Formula 1 style font**, looking sharp, modern, and race-inspired.

Let me know if you want it applied to `.hero-title`, buttons, or headings too!

You're right — the link I gave might not directly show “Formula1-Regular.ttf” as you expect.

Here’s a **working and tested solution**:

✅ Correct Link to Download Formula 1 Font (Fan version)

> 🏁 **Formula1 Display (Unofficial fan font)**

🔗 **Download from Dafont:**
👉 https://www.dafont.com/f1-font.font

🔽 Steps to Follow:

1. Go to the link: https://www.dafont.com/f1-font.font
2. Click the "**Download**" button on the right.
3. Extract the downloaded ZIP file.
4. Inside you'll find a font file like:
   ✅ `F1Regular.ttf` or similar.

📁 Put it in Your Project:

Move `F1Regular.ttf` to your React project under:

```arduino
/public/fonts/F1Regular.ttf
```

🧩 CSS Setup:

```css
@font-face {
  font-family: 'Formula1 Display';
  src: url('/fonts/F1Regular.ttf') format('truetype');
  font-weight: normal;
  font-style: normal;
}
```

✅ Usage:

```css
.hero-stats {
  font-family: 'Formula1 Display', sans-serif;
}
```

If you'd like, I can also give you a prebuilt `.zip` with the proper file structure and CSS ready to copy into your project. Just ask!

Sure! Here's a **rephrased version** of your sentence using **very different words**, while keeping the same core meaning:

> **"Empowering entrepreneurs, enriching backers."**

More creative alternatives:

1. **"Builders rise, backers thrive."**
2. **"Where visionaries scale and supporters prosper."**
3. **"Startups soar, capital works."**
4. **"Creators flourish, funders benefit."**
5. **"Fueling founders, rewarding investors."**

Let me know if you'd like a **more corporate**, **casual**, or **tech-savvy** tone!

Here's how to **replicate that stylish sliding box layout with hover animation**, icon movement, and smooth transitions — just like the UI in your screenshot.

✅ What We'll Build:

- Horizontal **scrollable feature boxes** (left ↔️ right).
- Icons that **move/jump/scale on hover**.
- Smooth hover effects & gradient accents.
- Fully responsive with `flex`.

✅ JSX Code (React):

jsx

```
<section
  ref={valueRef}
  className="value-proposition"
```

```
  data-aos="zoom-in-up"
>
  <h2 className="boxy-title">Empowering entrepreneurs, enriching backers.</h2>

  <div className="boxy-container">
    {[
      {
        title: 'FAST',
        icon: '⚡',
        text: 'Get offers in 48 hours by connecting your sales data.'
      },
      {
        title: 'FAIR',
        icon: '📊',
        text: 'No bias. No pitch. Just data-led funding decisions.'
      },
      {
        title: 'FLEXIBLE',
        icon: '♾️',
        text: 'Pay more in strong months, less when things slow.'
      },
      {
        title: 'FOUNDER FRIENDLY',
        icon: '🤝',
        text: 'No board seats. No control loss. No personal guarantees.'
      }
    ].map((item, index) => (
      <div key={index} className="boxy-card">
        <div className="boxy-icon">{item.icon}</div>
        <h3>{item.title}</h3>
        <p>{item.text}</p>
      </div>
    ))}
  </div>
</section>
```

CSS Styling

```css
.value-proposition {
  padding: 3rem 2rem;
  background: #0f0f0f;
  color: white;
  text-align: center;
}

.boxy-title {
  font-size: 2rem;
  font-weight: 700;
  background: linear-gradient(to right, #10b981, #6366f1);
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
  margin-bottom: 2rem;
  font-family: 'Titillium Web', sans-serif;
}

.boxy-container {
  display: flex;
  overflow-x: auto;
  gap: 1.5rem;
  padding-bottom: 1rem;
  scroll-snap-type: x mandatory;
}

.boxy-card {
  flex: 0 0 240px;
  scroll-snap-align: start;
  background: rgba(255, 255, 255, 0.05);
  border-radius: 1rem;
  padding: 1.5rem;
  backdrop-filter: blur(10px);
  transition: transform 0.3s ease, box-shadow 0.3s ease;
  box-shadow: 0 4px 16px rgba(0,0,0,0.2);
  text-align: center;
}
```

```css
.boxy-card:hover {
  transform: translateY(-5px);
  box-shadow: 0 6px 24px rgba(16, 185, 129, 0.3);
}

.boxy-icon {
  font-size: 2.5rem;
  margin-bottom: 1rem;
  transition: transform 0.3s ease;
}

.boxy-card:hover .boxy-icon {
  transform: scale(1.2) rotate(5deg);
}

.boxy-card h3 {
  font-size: 1.2rem;
  font-weight: 600;
  color: #ffffff;
  margin-bottom: 0.5rem;
  font-family: 'Titillium Web', sans-serif;
}

.boxy-card p {
  font-size: 0.95rem;
  color: #d1d5db;
  line-height: 1.5;
}
```

✅ Optional: Scrollbar Styling (Make it clean)

css

```css
.boxy-container::-webkit-scrollbar {
  height: 6px;
```

```
}
.boxy-container::-webkit-scrollbar-thumb {
  background: #10b981;
  border-radius: 4px;
}
```

💡 Bonus Tip:

You can even auto-scroll or snap to center using a little JavaScript or `scroll-behavior: smooth`.

Would you like me to:

- Add **AOS animation on scroll**?
- Use **icons from a library** (like `react-icons`) instead of emojis?
- Make it **auto-slider** or **carousel style**?

Let me know!

Absolutely! Here's a full solution to show your **7 animated info cards** (like the image), neatly styled under your **video/graph section** with hover effects and clean layout.

✅ JSX – `FeatureCards.jsx` (or directly in your section)

jsx

```
<section className="features-section">
  <h2 className="features-title">Why Partner With Us</h2>
  <div className="features-container">
    {[
```

```
      {
        icon: '',
        title: 'FAST',
        text: 'Quick turnaround on funding decisions. We connect investors with verified
SMEs in under 72 hours.'
      },
      {
        icon: '⚖️',
        title: 'FAIR',
        text: 'Transparent investment terms. No hidden clauses, upfront ROI metrics, and
clear documentation.'
      },
      {
        icon: '🔁',
        title: 'FLEXIBLE',
        text: 'Tailored investment and loan structures. Whether equity, debt, or hybrid –
we adapt to SME and investor goals.'
      },
      {
        icon: '🤝',
        title: 'FOUNDER-FIRST',
        text: 'We protect your vision. No forced dilution, no personal guarantees, no
board interference.'
      },
      {
        icon: '♻️',
        title: 'FREQUENT',
        text: 'Repeat-worthy experience. 70% of investors reinvest through our platform
in new SMEs.'
      },
      {
        icon: '🌍',
        title: 'WIDER REACH',
        text: 'Tier 2 & Tier 3 founders, now accessible. We bring untapped regional
entrepreneurs to the investor table.'
      },
      {
        icon: '📊',
        title: 'DATA-LED MATCHING',
        text: 'Right match, not just any match. Investor preferences are aligned with
```

```
founder sectors, stage, and scale.'
      },
    ].map((item, i) => (
      <div className="feature-card" key={i}>
        <div className="feature-icon">{item.icon}</div>
        <h3>{item.title}</h3>
        <p>{item.text}</p>
      </div>
    ))}
  </div>

  <div className="contact-box">
    <h3>Contact Us</h3>
    <p>Reach out and let's grow together.</p>
    <button className="contact-btn">Get in Touch</button>
  </div>
</section>
```

✅ CSS (Responsive, animated, glowing on hover)

CSS

```
.features-section {
  padding: 4rem 2rem;
  background-color: #0f0f0f;
  color: white;
  text-align: center;
}

.features-title {
  font-size: 2.2rem;
  margin-bottom: 2rem;
  background: linear-gradient(to right, #10b981, #6366f1);
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
  font-family: 'Titillium Web', sans-serif;
```

```
}

.features-container {
  display: flex;
  flex-wrap: wrap;
  justify-content: center;
  gap: 2rem;
}

.feature-card {
  background: rgba(255, 255, 255, 0.05);
  border-radius: 1rem;
  padding: 1.5rem;
  width: 280px;
  text-align: left;
  backdrop-filter: blur(10px);
  box-shadow: 0 6px 24px rgba(0, 0, 0, 0.2);
  transition: all 0.3s ease;
  position: relative;
  overflow: hidden;
  border: 1px solid rgba(255, 255, 255, 0.08);
}

.feature-card:hover {
  transform: translateY(-5px);
  box-shadow: 0 6px 30px rgba(16, 185, 129, 0.3);
}

.feature-icon {
  font-size: 2rem;
  margin-bottom: 1rem;
  transition: transform 0.3s ease;
}

.feature-card:hover .feature-icon {
  transform: scale(1.2) rotate(5deg);
}

.feature-card h3 {
  font-size: 1.25rem;
```

```
  font-weight: 700;
  margin-bottom: 0.5rem;
  color: #ffffff;
  font-family: 'Titillium Web', sans-serif;
}

.feature-card p {
  font-size: 0.95rem;
  color: #cbd5e1;
  line-height: 1.5;
}

/* Contact box at the end */
.contact-box {
  margin-top: 3rem;
  border-top: 1px solid rgba(255, 255, 255, 0.1);
  padding-top: 2rem;
}

.contact-box h3 {
  font-size: 1.5rem;
  margin-bottom: 0.5rem;
  color: #ffffff;
}

.contact-box p {
  color: #94a3b8;
  margin-bottom: 1rem;
}

.contact-btn {
  background: linear-gradient(to right, #10b981, #6366f1);
  border: none;
  padding: 0.75rem 1.5rem;
  color: white;
  font-weight: 600;
  border-radius: 999px;
  cursor: pointer;
  transition: transform 0.3s ease;
}
```

```
.contact-btn:hover {
  transform: scale(1.05);
  background: transparent;
  color: #10b981;
  border: 2px solid #10b981;
}
```

✅ Fonts (if not added yet)

Put this in your HTML head or `_document.js`:

```html
<link href="https://fonts.googleapis.com/css2?family=Titillium+Web:wght@400;600;700&display=swap" rel="stylesheet">
```

💬 Summary

You now have:

- A **stylish info card grid**
- **Animated emojis/icons**
- **Hover bounce/glow**
- A final **contact call-to-action box**

Would you like me to:

- Add **scroll-based animation** (e.g., `data-aos`)?
- Replace emojis with **React Icons**?
- Make a **carousel version**?

Just ask!

To match your existing `.value-card` layout **(keep width, padding, etc. the same)** but give it the **same dark gradient background, glass effect, hover glow, and icon bounce** as in the last example — here's the **refined version** of your CSS:

✅ Updated `.value-card` CSS (same layout, upgraded styling)

CSS

```css
.value-cards {
  display: flex;
  justify-content: center;
  gap: 2rem;
  flex-wrap: wrap;
}

.value-card {
  background: rgba(255, 255, 255, 0.05); /* translucent black glass */
  border-radius: 20px;
  padding: 2rem;
  width: 300px;
  backdrop-filter: blur(12px);
  box-shadow: 0 6px 24px rgba(16, 185, 129, 0.1);
  transition: transform 0.3s ease, box-shadow 0.3s ease;
  opacity: 0;
  transform: translateY(50px);
  border: 1px solid rgba(255, 255, 255, 0.08);
  text-align: center;
  color: white;
}

.value-card:hover {
  transform: translateY(-8px) scale(1.03);
  box-shadow: 0 8px 32px rgba(16, 185, 129, 0.25);
}
```

```css
.icon-wrapper {
  height: 120px;
  display: flex;
  justify-content: center;
  align-items: center;
  transition: transform 0.3s ease;
  font-size: 2rem;
}

.value-card:hover .icon-wrapper {
  transform: scale(1.2) rotate(5deg);
}

.value-card h3 {
  font-size: 1.25rem;
  margin: 1rem 0 0.5rem;
  color: #ffffff;
  font-weight: 700;
  font-family: 'Titillium Web', sans-serif;
}

.value-card p {
  font-size: 0.95rem;
  color: #cbd5e1;
  line-height: 1.5;
  font-family: 'Titillium Web', sans-serif;
}
```

✅ Notes:

- ✅ **Dimensions** are exactly the same as your old version (`300px`, `2rem`).
- ✅ Upgraded to **glassmorphism** with `backdrop-filter`, gradient shadows.
- ✅ Added **smooth hover scale**, glowing indigo-greenish shadow.
- ✅ Icon (`.icon-wrapper`) will **bounce slightly** on hover for interaction.

💡 Pro Tip:

If you're using images or emojis for icons inside `.icon-wrapper`, they'll animate on hover due to `transform`.

Would you like:

- To auto-fade-in on scroll (`opacity: 0 → 1`)?
- AOS scroll animation or animation delay for staggered effect?

Just say the word!

Got it! You want to **lift the card higher** and **add a tilt effect** when hovering. Here's the enhanced `.value-card:hover` CSS:

✅ Updated Hover Effect (More Lift + Tilt)

```css
.value-card:hover {
  transform: translateY(-20px) rotateX(3deg) rotateY(-3deg) scale(1.05);
  box-shadow: 0 12px 36px rgba(16, 185, 129, 0.3);
}
```

🔍 Explanation:

- `translateY(-20px)` — lifts card more visibly.

- `rotateX(3deg) rotateY(-3deg)` — adds a **slight 3D tilt**.
- `scale(1.05)` — subtly grows the card.
- `box-shadow` — makes the lift glow more intense.

✅ This effect looks elegant and interactive on modern UIs, especially dark themes.

Let me know if you'd like to:

- Add **smooth return** animation (`:not(:hover)` state)
- Add **perspective** to parent for stronger 3D
- Animate **icons separately**

I'm ready when you are!

Thanks! If **only the box shadow is working**, that means the `transform` properties (`rotate`, `translate`, `scale`) might not be applying correctly — most likely because the **parent or the card itself is missing perspective or 3D context**.

Let's fix it **step by step** so the **tilt and lift work properly**:

✅ Step-by-Step Fix:

✅ 1. Add Perspective to the Parent (required for 3D tilt)

Add this to `.value-cards`:

```css
.value-cards {
  display: flex;
  justify-content: center;
  gap: 2rem;
```

```
  flex-wrap: wrap;
  perspective: 1000px; /* Enables 3D rotation effect */
}
```

✅ 2. Update `.value-card:hover` with 3D Transform

```css
.value-card:hover {
  transform: translateY(-20px) rotateX(6deg) rotateY(-4deg) scale(1.05);
  box-shadow: 0 12px 36px rgba(16, 185, 129, 0.3);
}
```

> You can play with the values: `rotateX`, `rotateY` for how much "tilt" you want.

✅ 3. (Optional but Recommended) Set `transform-style` on `.value-card`

```css
.value-card {
  transform-style: preserve-3d;
  transition: transform 0.4s ease, box-shadow 0.4s ease;
}
```

🧪 Live Testing Tip:

Open DevTools → Select the card → try applying `transform: rotateX(10deg)` directly and see the effect. If it works there, the CSS above will fix it.

Let me know if you want to add:

- **Tilt based on mouse position** (interactive hover tilt)
- **Animated return to normal** when hover ends

- **Multiple layers** with `z-index` or depth effects

I'm happy to level this up more!

Here's how to **replicate the video container** from your image:

✅ 1. JSX Structure

Wrap your video and elements in a styled `.video-proposition` section:

jsx

```jsx
<section ref={valueRef} className="video-proposition" data-aos="zoom-in-up">
  <div className="video-window">
    <div className="video-header">
      <span className="dot red"></span>
      <span className="dot yellow"></span>
      <span className="dot green"></span>
    </div>

    <video
      className="video-content"
      autoPlay
      muted
      loop
      playsInline
      src="/your-video-path.mp4"
    ></video>
  </div>
</section>
```

Replace `/your-video-path.mp4` with the actual video file path in your project.

✅ 2. CSS Styling

```css
.video-proposition {
  display: flex;
  justify-content: center;
  align-items: center;
  padding: 4rem 2rem;
  background-color: #000;
}

.video-window {
  background: rgba(255, 255, 255, 0.03);
  border-radius: 1.5rem;
  border: 1px solid rgba(255, 255, 255, 0.05);
  box-shadow: 0 0 40px rgba(0, 0, 0, 0.3);
  overflow: hidden;
  width: 720px;
  max-width: 100%;
  position: relative;
  padding-top: 2rem;
}

.video-header {
  position: absolute;
  top: 1rem;
  left: 1.5rem;
  display: flex;
  gap: 0.5rem;
  z-index: 2;
}

.dot {
  height: 10px;
  width: 10px;
  border-radius: 50%;
  display: inline-block;
}
```

```
.dot.red {
  background-color: #ff5f57;
}
.dot.yellow {
  background-color: #ffbd2e;
}
.dot.green {
  background-color: #28c840;
}

.video-content {
  width: 100%;
  border-radius: 0 0 1.5rem 1.5rem;
  display: block;
}
```

✅ 3. Add a sample video (optional)

You can use a royalty-free video to test:

html

```
<video
  className="video-content"
  autoPlay
  muted
  loop
  playsInline
  src="https://www.w3schools.com/html/mov_bbb.mp4"
/>
```

🧠 Optional:

- Add a **gradient border** instead of plain border using `border-image`.

- You can make it responsive by wrapping with a `.container` or using `max-width: 100%`.

Would you like to add:

- **Step indicator** like in your screenshot?
- **Sliders and form** below the video?

I can help you wire those in too!

To achieve a **transparent and blurred navbar on scroll**, you're almost there! You just need to ensure a few things are working **in sync**:

✅ 1. CSS (You're Nearly Correct)

Make sure this part stays as is for `.navbar` (initial state):

```css
.navbar {
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  z-index: 1000;
  background: transparent; /* starts fully transparent */
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 1rem 2rem;
  width: 100%;
  box-sizing: border-box;
  transition: background-color 0.4s ease, backdrop-filter 0.4s ease;
```

```
  backdrop-filter: none;
}
```

And for the `.navbar-scrolled` class:

```css
.navbar.navbar-scrolled {
  background: linear-gradient(90deg, #10b981, #3b82f6, #8b5cf6);
  backdrop-filter: blur(10px) saturate(180%);
  box-shadow: 0 2px 8px rgba(0, 0, 0, 0.15); /* Optional shadow */
}
```

✅ 2. JavaScript/React Scroll Logic

You need to toggle the `navbar-scrolled` class when the user scrolls.

If you're using **React**, here's a minimal scroll detector:

```jsx
import { useEffect, useState } from 'react';

function Navbar() {
  const [scrolled, setScrolled] = useState(false);

  useEffect(() => {
    const onScroll = () => {
      setScrolled(window.scrollY > 20);
    };

    window.addEventListener('scroll', onScroll);
    return () => window.removeEventListener('scroll', onScroll);
  }, []);

  return (
    <nav className={`navbar ${scrolled ? 'navbar-scrolled' : ''}`}>
      {/* Navbar Content */}
```

```
    </nav>
  );
}
```

✅ 3. What You'll See

- On page load: transparent navbar with no blur.
- On scroll: navbar gets a **gradient with blur and slight shadow**, standing out while still glassy.

Would you like to:

- Animate it in with `fade`?
- Add scroll direction detection (hide on scroll down)?

Happy to help further!

Got it! You want the **navbar to have a bold gradient by default**, and when the user **scrolls**, it should become **slightly transparent with a blur effect — but still keep the same gradient**.

Here's how to do that perfectly:

✅ CSS Setup

css

```css
/* Default: solid gradient */
.navbar {
  position: fixed;
  top: 0;
  left: 0;
```

```
  right: 0;
  z-index: 1000;
  background: linear-gradient(90deg, #10b981, #3b82f6, #8b5cf6); /* Solid gradient */
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 1rem 2rem;
  width: 100%;
  box-sizing: border-box;
  transition: background 0.4s ease, backdrop-filter 0.4s ease;
  backdrop-filter: none;
}

/* When scrolled: same gradient, but transparent and blurred */
.navbar.navbar-scrolled {
  background: linear-gradient(
    90deg,
    rgba(16, 185, 129, 0.7),
    rgba(59, 130, 246, 0.7),
    rgba(139, 92, 246, 0.7)
  ); /* same gradient but transparent */
  backdrop-filter: blur(10px);
  box-shadow: 0 2px 12px rgba(0, 0, 0, 0.15); /* optional shadow */
}
```

✅ JS (React) Scroll Handler

Use this React effect to toggle the `.navbar-scrolled` class:

jsx

```
import { useEffect, useState } from 'react';

function Navbar() {
  const [scrolled, setScrolled] = useState(false);

  useEffect(() => {
```

```
    const handleScroll = () => {
      setScrolled(window.scrollY > 20);
    };

    window.addEventListener('scroll', handleScroll);
    return () => window.removeEventListener('scroll', handleScroll);
  }, []);

  return (
    <nav className={`navbar ${scrolled ? 'navbar-scrolled' : ''}`}>
      {/* Your navbar content */}
    </nav>
  );
}
```

✅ Result:

State	Appearance
Initial	Fully visible gradient: `#10b981 → #3b82f6 → #8b5cf6`
Scrolled	Same gradient but semi-transparent + blur effect

Let me know if you also want the navbar to **fade in/out**, **hide on scroll down**, or use **glassmorphism effects**.

To add that **animated 3D grid background** effect like in the image, you have a few options, but here's the simplest and most effective method:

✅ Option 1: **Use a Grid SVG or PNG Background (Fastest)**

You can add a **grid PNG or SVG** like a background image to your `.home` section.

🧩 CSS Code:

```css
.home {
  font-family: 'Inter', sans-serif;
  color: #0f172a;
  background-color: #000; /* or transparent */
  background-image: url('/path-to-grid.png'); /* Use your grid image */
  background-repeat: no-repeat;
  background-size: cover;
  background-position: center;
  background-attachment: fixed; /* Optional parallax effect */
  transition: background-color 0.5s ease;
}
```

> Replace `/path-to-grid.png` with your actual file path (e.g., `/assets/grid-lines.png` or import it if using React)

✅ Option 2: **Animated CSS Grid with Pseudo Elements (Modern/Custom)**

If you want a **pure CSS grid effect**, here's how you can do a subtle one:

🧩 CSS for `.home`:

```css
.home {
  position: relative;
  overflow: hidden;
  background-color: #0f172a;
  font-family: 'Inter', sans-serif;
  color: white;
}
```

```css
/* Pseudo grid */
.home::before {
  content: '';
  position: absolute;
  inset: 0;
  background-image: linear-gradient(to right, rgba(255,255,255,0.05) 1px, transparent 1px),
                    linear-gradient(to bottom, rgba(255,255,255,0.05) 1px, transparent 1px);
  background-size: 60px 60px;
  z-index: 0;
  animation: moveGrid 60s linear infinite;
}

@keyframes moveGrid {
  0% {
    background-position: 0 0;
  }
  100% {
    background-position: 600px 600px;
  }
}

/* Make sure your content is above the grid */
.home > * {
  position: relative;
  z-index: 1;
}
```

✅ Option 3: **3D WebGL Grid (Advanced)**

If you're aiming for **interactive or parallax effects**, you can use:

- Three.js
- Vanta.js (`VANTA.NET`, `VANTA.GLOBE`, etc.)

But those require setup with JavaScript or React.

🛠️ Tip for React Projects

If you're using **Next.js/React** and importing an image:

```js
import GridBg from '../assets/grid-bg.png';

<div className="home" style={{ backgroundImage: `url(${GridBg})` }}>
  {/* content */}
</div>
```

Would you like me to generate the **SVG/CSS grid background** for you or a transparent **3D grid PNG** file?

The color in the image you've uploaded is a very dark shade—nearly black.

🎨 Approximate Hex Color Code:

```bash
#161616
```

This is a **near-black** with a subtle grey undertone, commonly used in dark UI designs for a modern, elegant look.

Let me know if you'd like a lighter variant or a full gradient suggestion to go with it.

If the **grid lines background** is not covering the **full width** of your screen, it's likely due to one of these reasons:

✅ Common Fixes

1. **Background image container isn't full width**

Ensure the element with the grid background stretches across the entire page.

```css
.home {
  width: 100%;
  min-height: 100vh;
  background-image: url('/path-to-your/grid.png');
  background-size: cover; /* or contain / 100% 100% */
  background-position: center;
  background-repeat: no-repeat;
  background-attachment: fixed; /* Optional for parallax feel */
}
```

---

2. **Use `background-size: cover` or `100% 100%`**

To make the grid stretch fully, use:

```css
background-size: 100% 100%; /* or */
background-size: cover;
```

> `100% 100%` ensures the image stretches to fit both width and height.

---

3. **Parent container has max-width constraint**

If `.home` or its parent has `max-width` (like `1200px`), it will clip the background.

Fix:

```css
```

```
body, html {
  margin: 0;
  padding: 0;
  width: 100%;
}

.home {
  max-width: none;
}
```

---

4. **Image itself is not wide enough**

If the grid image is narrow or not transparent, it might look clipped.

✅ Solution: Use a **wider/larger grid image** or repeat it with:

```
css

background-repeat: repeat;
```

✅ Example CSS

```
css

.home {
  width: 100%;
  min-height: 100vh;
  background-image: url('/assets/grid-background.png');
  background-size: cover;
  background-repeat: no-repeat;
  background-position: center;
  background-attachment: fixed;
  font-family: 'Inter', sans-serif;
```

```
  color: #0f172a;
}
```

If you send your current `.home` HTML/CSS or layout, I can adjust it exactly for your structure.

To make the **grid background cover the full width and height** of your `.home` section (and ensure it doesn't get clipped or cut off), here's exactly what you need to **fix and check** step by step:

✅ STEP 1: Update `.home` CSS to full width/height

```css
.home {
  position: relative;
  min-height: 100vh;
  width: 100%;
  overflow-x: hidden;
  background-color: transparent;
  font-family: 'Inter', sans-serif;
  color: #0f172a;
}
```

✅ STEP 2: Fix the pseudo-grid

Your current `.home::before` is **almost correct**, just missing full layout enforcement and the `position: relative` on `.home`.

```css
```

```css
/* Grid overlay */
.home::before {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  width: 100vw;
  height: 100%;
  background-image:
    linear-gradient(to right, rgba(0, 110, 255, 0.1) 1px, transparent 1px),
    linear-gradient(to bottom, rgba(0, 110, 255, 0.1) 1px, transparent 1px);
  background-size: 140px 70px;
  z-index: 0;
  animation: moveGrid 60s linear infinite;
  pointer-events: none;
}
```

✅ STEP 3: Ensure `.home > *` is above the grid

```css
.home > * {
  position: relative;
  z-index: 1;
}
```

✅ STEP 4: Optional full-screen body base

Also add this in your global `App.css` or `index.css`:

```css
html, body {
  margin: 0;
```

```
  padding: 0;
  width: 100%;
  min-height: 100vh;
  overflow-x: hidden;
  background-color: black; /* Optional base */
}
```

✅ BONUS: Make the grid animate nicely

```css
@keyframes moveGrid {
  0% {
    background-position: 0 0;
  }
  100% {
    background-position: 600px 600px;
  }
}
```

✅ Summary

- `position: relative` → on `.home`
- `::before` uses `width: 100vw; height: 100%`
- Content stays on top using `z-index: 1`
- `overflow-x: hidden` to prevent scroll bar

If you apply this correctly, your animated **grid background** will appear across the **entire screen width** and under all content.

Let me know if you want to add a **gradient tint**, **dark overlay**, or **parallax scroll** on the grid too!

Got it! You're using `.home` with:

```css
.home {
  position: relative;
  overflow: hidden;
  background-color: #161616;
  font-family: 'Inter', sans-serif;
  color: white;
}
```

But you're saying **the grid lines aren't covering the full width** of the screen.

✅ Here's the fix step-by-step:

1. Make sure `.home` stretches full viewport width and height:

Update your `.home` like this:

```css
.home {
  position: relative;
  overflow: hidden;
  min-height: 100vh;
  width: 100vw;
  background-color: #161616;
  font-family: 'Inter', sans-serif;
  color: white;
}
```

> `width: 100vw;` is key — ensures the `.home` container spans the entire viewport horizontally.

---

2. Update the `.home::before` pseudo-element:

Make sure it also spans the full screen:

```css
.home::before {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  width: 100vw;
  height: 100%;
  background-image:
    linear-gradient(to right, rgba(0, 110, 255, 0.1) 1px, transparent 1px),
    linear-gradient(to bottom, rgba(0, 110, 255, 0.1) 1px, transparent 1px);
  background-size: 140px 70px;
  z-index: 0;
  animation: moveGrid 60s linear infinite;
  pointer-events: none;
}
```

---

3. Make sure content stays on top:

Add this if you haven't already:

```css
.home > * {
  position: relative;
  z-index: 1;
}
```

---

4. Optional: Remove padding/margin from `<body>` and `<html>`

Add to your global CSS:

```css
html, body {
  margin: 0;
  padding: 0;
  width: 100%;
  overflow-x: hidden;
  background-color: #161616;
}
```

✅ RESULT:

- Your `.home` section will now be full width (and height),
- The grid will cover the entire background,
- Content will sit on top of the grid.

Let me know if you want the **grid lines to fade in**, **change color on scroll**, or **respond to mouse movement** for cool effects!

You're seeing **a black strip on the left side** because of this line in your CSS:

```css
width: 89vw;
```

✅ What's happening:

- `vw` stands for **viewport width**.
- `89vw` means the `.home` container is only taking **89% of the total width**.
- That leftover **11% on the right or left** shows the background of the parent element (likely black).

✅ ✅ Fix:

Change this:

```css
width: 89vw;
```

To:

```css
width: 100vw;
```

🔁 Updated `.home` class:

```css
.home {
  position: relative;
  overflow: hidden;
  min-height: 100vh;
  width: 100vw; /* Full viewport width */
  background-color: #161616;
  font-family: 'Inter', sans-serif;
  color: white;
}
```

✅ Also Check:

Ensure no **horizontal padding or margin** exists on `<body>` or `<html>` that could shift content:

```css
html, body {
  margin: 0;
  padding: 0;
```

```
  overflow-x: hidden;
}
```

Let me know if you want the `.home` content to be **centered but full-width background**, I'll help set that layout too.

Ah — thanks for the clarification!

You're absolutely right to be concerned about the **horizontal scroll issue**. Let me explain why that's happening and how to **fix it the right way**, so you can:

✅ Keep full background width (100%)
✅ Avoid the **horizontal scrollbar**
✅ Prevent content from overflowing

🔍 Why you're getting a scroll bar with `100vw`

Using:

```css
width: 100vw;
```

does not account for the scrollbar width, which usually takes up **~16px**. This causes the layout to overflow horizontally — hence the black gap or scroll bar.

✅ The Fix

Instead of using `width: 100vw`, use this combo:

```css
```

```
width: 100%;
```

And to keep things tight and prevent overflow:

```css
html, body {
  margin: 0;
  padding: 0;
  overflow-x: hidden; /* Prevents horizontal scroll */
}
```

🔁 Updated `.home` CSS:

```css
.home {
  position: relative;
  overflow: hidden;
  min-height: 100vh;
  width: 100%; /* Instead of 89vw or 100vw */
  background-color: #161616;
  font-family: 'Inter', sans-serif;
  color: white;
}
```

✨ Bonus Tip: Center Content Inside

If you want your actual content to stay **nicely centered**, but still want a **full-width background**, then do:

```css
```

```
.home-inner {
  max-width: 1200px; /* Or any central width */
  margin: 0 auto;
  padding: 0 1rem;
}
```

And in JSX:

```jsx
<div className="home">
  <div className="home-inner">
    {/* All your content here */}
  </div>
</div>
```

Let me know if you want the grid background to stretch full-width too — I can make sure that respects this layout.

Thank you for the screenshot — it clearly shows that the **left and right edges have black margins**, even though you're using `width: 100%`.

Here's what's likely going wrong and how to **fix it** properly:

🧠 Root Cause

You are setting `width: 100%` **inside a container**, but the container itself (`.home`) is constrained because of **default** `body` **or parent element padding/margin**.

Also, your **grid background** is applied using `::before`, but it may **not cover the full screen** if:

- `.home` is not fully stretched edge-to-edge, or
- There's unwanted `padding` or `margin` on parent elements like `body`, `html`, etc.

✅ Solution: Ensure full edge-to-edge layout

1. Set margins and paddings to 0 for `html` and `body`:

```css
html, body {
  margin: 0;
  padding: 0;
  overflow-x: hidden; /* Prevent accidental overflow */
}
```

2. Set `.home` width to 100% (not `vw`) and remove any margin:

```css
.home {
  position: relative;
  overflow: hidden;
  min-height: 100vh;
  width: 100%;
  background-color: #161616;
  font-family: 'Inter', sans-serif;
  color: white;
}
```

3. Make sure the `::before` grid background is stretching properly:

```css
.home::before {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-image:
    linear-gradient(to right, rgba(0, 110, 255, 0.103) 1px, transparent 1px),
```

```
    linear-gradient(to bottom, rgba(0, 110, 255, 0.103) 1px, transparent 1px);
  background-size: 140px 70px;
  z-index: 0;
  animation: moveGrid 60s linear infinite;
}
```

Make sure `.home` has `position: relative;` so the pseudo-element is scoped correctly.

✅ Bonus Check

If you use a framework like `Next.js`, or you've wrapped things in containers, also check:

```css
#__next, .App, .main-wrapper {
  width: 100%;
}
```

Let me know if

- You're using a layout with a wrapper that has its own `max-width` or `padding`.
- You want the cube on the right to go **beyond content width** but stay **inside viewport**.

I'll help fine-tune that too.

Thanks for sharing your complete `ExplorePage.jsx`. The layout is **almost correct**, but the full width issue is **not caused by your JSX —** it's a **CSS problem**.

From the screenshots and explanation, here's **why you're still seeing black bars on the left and right**, even when using `width: 100%`.

🔎 Root Problem

You're using:

```css
.home {
  width: 89vw; /* or even width: 100vw; previously */
}
```

But `vw` is not always reliable when combined with `overflow-x`, `padding`, and parent element margins. Also, if the `.home` container is centered inside a `div` or layout grid, it may be constrained.

✅ How to Fix This Fully

1. Make sure these are present in your global CSS (VERY IMPORTANT):

```css
html, body {
  margin: 0;
  padding: 0;
  width: 100%;
  overflow-x: hidden;
}
```

2. Use `width: 100%` for `.home`, not `vw`:

```css
.home {
  position: relative;
  overflow: hidden;
  min-height: 100vh;
  width: 100%;
  background-color: #161616;
  font-family: 'Inter', sans-serif;
```

```
  color: white;
}
```

✅ Avoid `vw` because:

- `100vw` includes the **scrollbar width**, which causes **horizontal overflow** and black bars
- `89vw` obviously leaves space on the sides

---

3. If you still see black bars...

Then some **parent layout is constraining width**. In that case, add this to your layout file or page wrapper (like `App.css`, `_app.js`, or global wrapper):

```css
#root,
#__next,
.App {
  width: 100%;
  margin: 0;
  padding: 0;
}
```

Also make sure **nothing in the parent hierarchy has max-width or padding**.

✅ Bonus (Fix for pseudo-grid `::before`)

```css
.home::before {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;     /* Not 100vw */
```

```
  height: 100%;
  background-image:
    linear-gradient(to right, rgba(0, 110, 255, 0.103) 1px, transparent 1px),
    linear-gradient(to bottom, rgba(0, 110, 255, 0.103) 1px, transparent 1px);
  background-size: 140px 70px;
  z-index: 0;
  animation: moveGrid 60s linear infinite;
}
```

💡 Summary

Setting	✅ Recommended value
`.home width`	`width: 100%`
`html, body` padding	`padding: 0; margin: 0;`
`overflow-x`	`hidden` on `body`
Pseudo element	`width: 100%` not `100vw`

If you're still stuck after this, I can help **inspect the live layout tree** if you share how your layout is structured — like parent divs, layout wrappers, etc.

Thanks for updating the `.home` CSS. Since you're still seeing **black space on the left and right**, the issue is likely **not in `.home` itself**, but in its **parent layout or default browser styles**.

Let's fix this **step by step**.

✅ 1. Ensure `body` and `html` are full width and have no margin

In your **global CSS** (e.g., `index.css`, `App.css`, or wherever your root styles are):

```css
html, body {
  margin: 0;
  padding: 0;
  width: 100%;
  height: 100%;
  overflow-x: hidden;
  background-color: #161616;
}
```

This prevents any browser default spacing that might push `.home` inward.

✅ 2. Check your parent container (e.g., `#root`, `.App`, or a layout div)

Also ensure the root wrapper is not limiting width:

```css
#root, .App {
  margin: 0;
  padding: 0;
  width: 100%;
  height: 100%;
}
```

✅ 3. Your `::before` grid background is not full width

Update this:

```css
```

```
.home::before {
  content: '';
  position: absolute;
  top: 0;
  left: 0;
  width: 100%; /* ✅ NOT 100vw */
  height: 100%;
  background-image:
    linear-gradient(to right, rgba(0, 110, 255, 0.103) 1px, transparent 1px),
    linear-gradient(to bottom, rgba(0, 110, 255, 0.103) 1px, transparent 1px);
  background-size: 140px 70px;
  z-index: 0;
  animation: moveGrid 60s linear infinite;
}
```

This ensures the **pseudo-grid** covers the full `.home` div, and doesn't cause overflow.

✅ 4. Bonus Debug: Use DevTools (in Chrome)

1. Right-click the black area → Inspect.
2. Check the element boundaries (hover to see which element is stopping at 80-90% width).
3. You'll likely find a parent `div`, `body`, or `.App` with:
   - `max-width`
   - `padding`
   - or not `width: 100%`

✅ 5. If Using Tailwind or Bootstrap

If you're using a framework that applies container rules like:

```
css
```

```
.container {
  max-width: 1200px;
  margin: auto;
}
```

It may be **centering** content.

In that case, override with:

```css
.container {
  max-width: none !important;
  width: 100% !important;
  margin: 0 !important;
}
```

✅ Summary Final Fix List

- ✔ `.home` – ✅ `width: 100%`
- ✔ `body, html` – ✅ no margin/padding
- ✔ `#root, .App` – ✅ no margin/padding, `width: 100%`
- ✔ `.home::before` – ✅ `width: 100%`, not `vw`
- ✔ No parent container like `.container` should limit width

If it **still doesn't work**, upload or paste your `index.html`, `App.js`, and any layout wrapper CSS – I'll pinpoint exactly which wrapper is causing it.