# Sticky Scrolling Effect — Cross-Team Pattern Document

> **Pattern Category:** Landing Page / Front-End
> **Author:** Abhiram
> **Date:** 2026-02-23
> **Project Reference:** Ottobon Academy Landing Page
> **Tech Stack:** React 19 · Framer Motion v12 · Tailwind CSS v4 · Vite

## 1. What Is the Sticky Scrolling Effect?

A **"sticky scroll stack"** is a UI pattern where multiple full-screen cards are stacked in the same viewport. As the user scrolls, each card stays **pinned (sticky)** to the top of the screen, and the **next card scrolls in on top**, creating a layered "deck of cards" illusion.

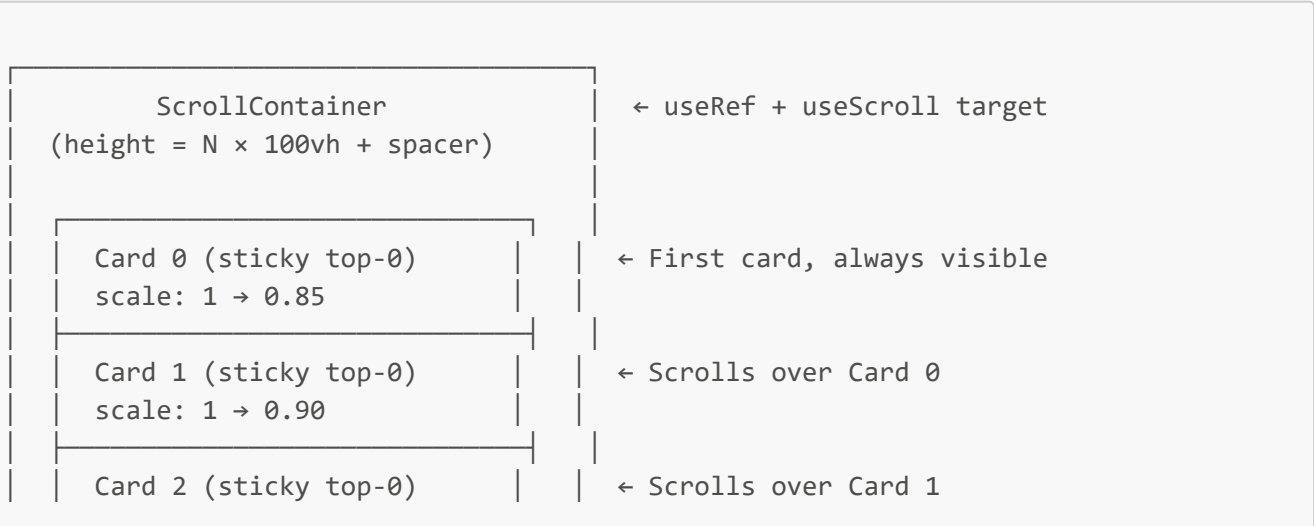
This is achieved by combining:

- **CSS** `position: sticky` — to pin each card in the viewport
- **Framer Motion** `useScroll` + `useTransform` — to drive scale/opacity/translate animations based on scroll progress 1

### Why This Pattern?

| Benefit | Explanation |
|---|---|
| **Engagement** | Forces users to scroll through all content; reduces bounce rate |
| **Narrative flow** | Each card tells a story in sequence — ideal for feature showcases |
| **Premium feel** | The stacking + scale effect gives a polished, app-like experience |
| **No extra libraries** | Only needs Framer Motion (commonly already available in React projects) |

## 2. Architecture Overview



```
  │  │  scale: 1 → 0.95            │   │
  │  └─────────────────────────────┘   │
  │                                    │
  │  [spacer div: h-[20vh]]            │  ← Ensures last card fully scrolls in
  └────────────────────────────────────┘
```

---

## 3. Implementation Steps

### Step 1: Install Dependencies

```
npm install framer-motion
```

### Step 2: Create the Container Component

The **container** wraps all cards and provides the scroll-tracking context.

```
// StickyScrollStack.tsx
import { useRef } from 'react';
import { useScroll } from 'framer-motion';
import { StickyCard } from './StickyCard';

// Your data items — adapt this interface to your needs
interface CardItem {
  id: string;
  title: string;
  description: string;
  color: string;
}

export function StickyScrollStack({ items }: { items: CardItem[] }) {
  const containerRef = useRef<HTMLDivElement>(null);

  // CORE HOOK: Track scroll progress of the container (0 → 1)
  // 'start start' = tracking begins when container top hits viewport top
  // 'end end'     = tracking ends when container bottom hits viewport bottom
  const { scrollYProgress } = useScroll({
    target: containerRef,
    offset: ['start start', 'end end'],
  });

  return (
    <div ref={containerRef} className="relative w-full">
      {items.map((item, index) => {
        // Each card shrinks slightly as the next card overlaps it
        // Last card stays at scale 1.0, earlier cards scale down more
        const targetScale = Math.max(
          0.85,
          1 - (items.length - index - 1) * 0.05
```

```
        );

        return (
          <StickyCard
            key={item.id}
            item={item}
            index={index}
            scrollProgress={scrollYProgress}
            // Evenly divide the scroll range among all cards
            range={[index * (1 / items.length), 1]}
            targetScale={targetScale}
          />
        );
      })}

      {/* IMPORTANT: Spacer ensures the last card has room to fully appear */}
      <div className="h-[20vh]" />
    </div>
  );
}
```

**Key decisions explained:**

- `offset: ['start start', 'end end']` — this means the scroll progress goes from 0→1 as the container passes through the viewport
- `targetScale` formula ensures earlier cards shrink more, creating visual depth
- The spacer `div` at the bottom prevents the last card from being cut off

### Step 3: Create the Card Component

Each card uses **CSS** `sticky` to stay pinned and **Framer Motion** `useTransform` to animate based on scroll.

```
// StickyCard.tsx
import { useRef } from 'react';
import { motion, useTransform, MotionValue } from 'framer-motion';

interface StickyCardProps {
  item: { id: string; title: string; description: string; color: string };
  index: number;
  scrollProgress: MotionValue<number>;
  range: [number, number];
  targetScale: number;
}

export function StickyCard({
  item,
  index,
  scrollProgress,
  range,
  targetScale,
}: StickyCardProps) {
```

```
  const cardRef = useRef<HTMLDivElement>(null);

  // CORE TRANSFORM: Map the container's scroll progress to a scale value
  // When scroll is at range[0], scale = 1 (full size)
  // When scroll is at range[1], scale = targetScale (shrunk)
  const scale = useTransform(scrollProgress, range, [1, targetScale]);

  return (
    // CRITICAL: `sticky top-0` + `h-screen` makes each card
    // fill the viewport and stay pinned as you scroll
    <div
      ref={cardRef}
      className="sticky top-0 flex items-center justify-center h-screen"
    >
      <motion.div
        style={{
          scale,
          // Offset each card slightly downward so stacking is visible
          top: `calc(5% + ${index * 35}px)`,
        }}
        className="relative w-[95%] max-w-6xl h-[650px] rounded-[32px]
                   overflow-hidden origin-top transition-all duration-700"
      >
        {/* YOUR CARD CONTENT GOES HERE */}
        <div className="p-12">
          <h3 className="text-5xl font-black text-white">{item.title}</h3>
          <p className="text-slate-400 mt-4">{item.description}</p>
        </div>
      </motion.div>
    </div>
  );
}
```

**Key decisions explained:**

- `sticky top-0` — the CSS property that pins the card to the viewport top
- `h-screen` — each card wrapper takes full viewport height, creating scroll space
- `origin-top` — the scale animation shrinks from the top edge (looks natural with stacking)
- `top: calc(5% + ${index * 35}px)` — offsets each card so the previous card's top edge peeks through

## Step 4: Use in Your Page

```
// Page.tsx
import { StickyScrollStack } from './StickyScrollStack';

const features = [
  { id: 'learn',  title: 'Learn',  description: 'Interactive courses', color:
'#5F9B8C' },
  { id: 'build',  title: 'Build',  description: 'Hands-on projects',   color:
'#FF7D2D' },
```

```
  { id: 'grow',   title: 'Grow',   description: 'Career support',      color:
'#A0C382' },
];

export default function Page() {
  return (
    <main>
      <header>{/* Hero Section */}</header>
      <StickyScrollStack items={features} />
      <footer>{/* Footer */}</footer>
    </main>
  );
}
```

---

## 4. Advanced Enhancements (Optional)

### 4a. Per-Card Entrance Animations

Add opacity and Y-translate so each card fades in as it enters:

```
// Inside StickyCard — additional transforms
const stepSize = 1 / totalCards;
const start = index * stepSize;

const opacity = useTransform(scrollProgress, [start, start + 0.15], [0, 1]);
const y = useTransform(scrollProgress, [start, start + 0.2], [100, 0]);

// Apply to the motion.div:
<motion.div style={{ scale, opacity: index === 0 ? 1 : opacity, y }}>
```

### 4b. Colored Glows & Borders

Give each card a themed glow:

```
<motion.div
  style={{
    scale,
    borderTop: `1px solid ${item.color}60`,
    boxShadow: `0 -10px 40px -10px ${item.color}10`,
  }}
>
```

### 4c. Lenis Smooth Scroll Integration

If you use Lenis for smooth scrolling, ensure it doesn't conflict with Framer Motion's scroll tracking:

```
// main.tsx or App.tsx
import Lenis from 'lenis';

const lenis = new Lenis({ lerp: 0.1 });
function raf(time: number) {
  lenis.raf(time);
  requestAnimationFrame(raf);
}
requestAnimationFrame(raf);
```

> ⚠ **Known Issue:** Lenis can sometimes intercept scroll events before Framer Motion reads them. If animations feel laggy, try increasing `lerp` or temporarily disabling Lenis on the sticky section.

---

## 5. Common Pitfalls & Troubleshooting

| Problem | Cause | Fix |
|---|---|---|
| Cards don't stack | Missing `sticky top-0` on card wrapper | Ensure each card's outer `div` has `className="sticky top-0 h-screen"` |
| Last card gets cut off | No spacer at the bottom | Add `<div className="h-[20vh]" />` after the last card |
| Animations feel off | Wrong `offset` in `useScroll` | Must be `['start start', 'end end']` for the container |
| Horizontal scrollbar appears | Card width exceeds viewport | Use `w-[95%]` or `max-w-6xl` with `overflow-hidden` on the card |
| Scale jumps instead of animating | `range` values overlap or are incorrect | Ensure `range` is `[index / total, 1]` with no gaps |

---

## 6. Cross-Validation Checklist

Use this checklist when reviewing this pattern against other team patterns:

- ☐ **Does the binding-side pattern handle scroll events?** — If yes, verify no conflict with `useScroll`
- ☐ **Does the data-side pattern load content dynamically?** — Ensure card data is available before the component mounts
- ☐ **Mobile responsiveness** — Test on viewport < 768px; cards should still stack correctly
- ☐ **Performance** — Scroll animations should run at 60fps; check with Chrome DevTools → Performance tab
- ☐ **Accessibility** — Ensure content within sticky cards is keyboard-navigable
- ☐ **Integration** — This pattern only needs a parent `<div>` with sufficient scroll height; it's self-contained

---

## 7. How to Share This Pattern

### For Cross-Team Members

1. **Share this document** — Send `docs/sticky-scroll-pattern.md` directly (it's self-contained)
2. **Reference code** — The abstracted `StickyScrollStack` and `StickyCard` components above are framework-agnostic React patterns; teammates can adapt them to their project
3. **Live demo** — Run the project locally: `npm run dev` → Scroll to "Our Offerings" section to see the pattern in action

### For Your Presentation

1. Show the **scroll effect** live on the landing page
2. Walk through the **architecture diagram** (Section 2)
3. Explain the **three key CSS/JS primitives** that make it work:
   - `position: sticky` → pins the card
   - `useScroll` → tracks scroll position as 0→1
   - `useTransform` → maps scroll to visual properties (scale, opacity)
4. Highlight the **pitfalls table** — shows you understand edge cases

---

## 8. File References (Project-Specific)

| File | Role |
|---|---|
| `src/components/carousel.tsx` | Active implementation using `OfferingSlide` + `OfferingsCarousel` |
| `src/components/OfferingsStack.tsx` | Alternative implementation with per-card entrance animations |
| `src/LandingPage.tsx` | Integration point — imports and renders `OfferingsCarousel` |
| `package.json` | Dependencies: `framer-motion@^12.23.26`, `lenis@^1.3.17` |