Southern Haulers Landing Page Architecture

Table of Contents

Overview

The Southern Haulers landing page is built on a modern, scalable architecture leveraging:

- **NextJS 14+ (App Router)**: Server-first architecture with optimized performance
- **TypeScript**: Full type safety across the application
- **Tailwind CSS v4 + Quartz Theme**: Modern utility-first CSS with custom design system
- **ShadCN UI + Tailark Components**: Pre-built, accessible UI components
- **Centralized Data Registry**: Single source of truth for all content
- **Modular Component System**: Reusable, composable UI components

Key Features

✅ **100vh Sections**: All core sections are full-viewport-height for modern UX
✅ **Alternating Split Layouts**: Left/right splits with content + visuals
✅ **Real-time Tracking**: Live container tracking demonstration
✅ **Dynamic Routing**: SEO-optimized routes for all locations, ports, and services
✅ **Light/Dark Theme**: Black-on-white and white-on-black theme system
✅ **Performance First**: Code splitting, lazy loading, optimized images
✅ **Accessibility**: WCAG AA compliant, semantic HTML, keyboard navigation

---

Architecture Principles

1. Single Source of Truth (SSOT)

All content lives in `/apps/web/src/data/` registries:
- No hardcoded content in components
- Update once, reflect everywhere

- Type-safe data access
- Easy content management

2. Component Modularity

- Small, focused components
- Single responsibility principle
- Composable building blocks
- Reusable across pages

3. Server-First Architecture

- Server Components by default
- Client Components only when needed
- Reduced JavaScript bundle size
- Improved initial load performance

4. Progressive Enhancement

- Works without JavaScript
- Enhanced with client-side interactions
- Graceful degradation for animations
- Mobile-first responsive design

5. Type Safety

- TypeScript throughout
- Strict type checking
- Exported types from registries
- Compile-time error detection

Folder Structure

```
apps/web/
├── app/                          # NextJS App Router
│   ├── (marketing)/              # Marketing layout group
│   │   ├── layout.tsx            # Marketing-specific layout
│   │   ├── page.tsx              # Homepage (/)
│   │   ├── about/
│   │   │   └── page.tsx
│   │   ├── locations/
│   │   │   ├── page.tsx          # All locations
│   │   │   └── [slug]/
│   │   │       └── page.tsx      # Dynamic location page
│   │   ├── ports/
│   │   │   ├── page.tsx          # All ports
│   │   │   └── [slug]/
│   │   │       └── page.tsx      # Dynamic port page
│   │   └── services/
│   │       ├── page.tsx          # All services
│   │       └── [slug]/
│   │           └── page.tsx      # Dynamic service page
│   ├── contact/
│   │   └── page.tsx
│   ├── quote/
│   │   └── page.tsx
│   ├── track/
│   │   └── page.tsx
│   ├── api/                      # API routes
│   │   ├── contact/
│   │   ├── quote/
│   │   └── tracking/
│   ├── layout.tsx                # Root layout
│   ├── globals.css               # Global styles
│   └── error.tsx                 # Error boundary
│
├── src/
│   ├── components/               # Reusable UI components
│   │   ├── ui/                   # Base UI primitives (ShadCN)
│   │   │   ├── button.tsx
│   │   │   ├── input.tsx
│   │   │   ├── card.tsx
│   │   │   └── ...
│   │   ├── sections/             # Landing page sections
│   │   │   ├── hero-section.tsx
│   │   │   ├── services-section.tsx
│   │   │   ├── locations-section.tsx
│   │   │   ├── features-section.tsx
│   │   │   ├── calculator-section.tsx
│   │   │   ├── stats-section.tsx
│   │   │   ├── testimonials-section.tsx
│   │   │   ├── faq-section.tsx
│   │   │   └── cta-section.tsx
│   │   ├── layout/               # Layout components
│   │   │   ├── header.tsx
│   │   │   ├── footer.tsx
│   │   │   ├── navigation.tsx
│   │   │   └── mobile-menu.tsx
│   │   ├── features/             # Feature-specific components
│   │   │   ├── quote-calculator.tsx
│   │   │   ├── live-tracking.tsx
│   │   │   ├── port-map.tsx
│   │   │   └── ...
│   │   └── shared/               # Shared utility components
│   │       ├── animated-counter.tsx
│   │       ├── theme-toggle.tsx
│   │       └── ...
│   │
│   ├── data/                       # Data registries (SSOT)
│   │   ├── registry.ts             # Master registry
│   │   ├── ports.ts
│   │   ├── locations.ts
│   │   ├── services.ts
│   │   ├── features.ts
│   │   ├── testimonials.ts
│   │   ├── stats.ts
│   │   └── faqs.ts
│   │
│   ├── styles/                     # Modular stylesheets
│   │   ├── base.css                # Base styles, CSS variables
│   │   ├── components.css          # Component-specific styles
│   │   ├── animations.css          # Animation definitions
│   │   └── theme.css               # Theme-specific overrides
│   │
│   ├── lib/                        # Utility functions
│   │   ├── utils.ts                # General utilities
│   │   ├── seo.ts                  # SEO helpers
│   │   ├── images.ts               # Image optimization helpers
│   │   └── animations.ts           # Animation utilities
│   │
│   └── types/                      # TypeScript type definitions
│       ├── components.ts
│       └── api.ts
│
├── public/                         # Static assets
│   ├── images/                     # Image assets
│   │   ├── hero/
│   │   ├── services/
│   │   ├── locations/
│   │   ├── ports/
│   │   └── ...
│   ├── icons/
│   └── fonts/
│
├── styles/                         # Global CSS
│   └── tailwind.config.ts          # Tailwind configuration
│
├── components.json                 # ShadCN configuration
├── package.json
└── tsconfig.json
```

Data Flow

Registry → Component → UI

```
┌─────────────────────────────────────────────────────────────┐
│                    Data Registries                          │
│  (Single Source of Truth - /apps/web/src/data/)            │
└──────────────────────────┬──────────────────────────────────┘
                           │
                           │ Import & Type-safe access
                           │
┌──────────────────────────▼──────────────────────────────────┐
│                   Page Components                           │
│     (Server Components - fetch/filter data)                 │
│                                                              │
│  • Homepage (page.tsx)                                       │
│  • Service Pages ([slug]/page.tsx)                           │
│  • Location Pages ([slug]/page.tsx)                          │
│  • Port Pages ([slug]/page.tsx)                              │
└──────────────────────────┬──────────────────────────────────┘
                           │
                           │ Pass data as props
                           │
┌──────────────────────────▼──────────────────────────────────┐
│                 Section Components                          │
│   (Presentation Components - display data)                  │
│                                                              │
│  • HeroSection                                               │
│  • ServicesSection                                           │
│  • LocationsSection                                          │
│  • etc.                                                       │
└──────────────────────────┬──────────────────────────────────┘
                           │
                           │ Compose with UI primitives
                           │
┌──────────────────────────▼──────────────────────────────────┐
│                  UI Components                              │
│        (Base primitives from ShadCN/Tailark)                │
│                                                              │
│  • Button, Card, Input                                       │
│  • Navigation Menu                                           │
│  • etc.                                                       │
└─────────────────────────────────────────────────────────────┘
```

Example: Services Section

```
// 1. Registry provides data
// apps/web/src/data/services.ts
export const SERVICES: Service[] = [
  {
    id: 'container-drayage',
    name: 'Container Drayage',
    description: '...',
    features: ['...'],
    // ...
  },
  // ...
];

// 2. Page imports and uses data
// apps/web/app/page.tsx (Server Component)
import { SERVICES } from '@/data/registry';
import { ServicesSection } from '@/components/sections/services-section';

export default function HomePage() {
  return (
    <>
      <ServicesSection services={SERVICES} />
    </>
  );
}

// 3. Section renders with data
// apps/web/src/components/sections/services-section.tsx
export function ServicesSection({ services }: { services: Service[] }) {
  return (
    <section className="min-h-screen">
      {services.map(service => (
        <ServiceCard key={service.id} service={service} />
      ))}
    </section>
  );
}
```

Component Architecture

Component Types

1. Server Components (Default)

Location: Page files, section components

Purpose: Data fetching, static rendering, SEO

Characteristics:

- Rendered on server
- No interactivity
- Smaller bundle size
- Better SEO

Example:

```
// apps/web/app/page.tsx
import { SERVICES } from '@/data/registry';

export default function HomePage() {
  return <ServicesSection services={SERVICES} />;
}
```

2. Client Components

Location: Interactive components
Purpose: User interactions, animations, state management
Characteristics:
- Rendered on client
- Interactive
- Uses hooks
- Event handlers

Example:

```
'use client';

import { useState } from 'react';

export function InteractiveMap() {
  const [selected, setSelected] = useState(null);
  return <div onClick={() => setSelected('...')}>...</div>;
}
```

3. Hybrid Components

Location: Sections with interactive parts
Purpose: Server rendering with client interactivity
Characteristics:
- Server component wrapper
- Client component children
- Best of both worlds

Example:

```
// Server Component
export function TestimonialsSection({ testimonials }) {
  return (
    <section>
      <TestimonialCarousel testimonials={testimonials} />
    </section>
  );
}

// Client Component
'use client';
export function TestimonialCarousel({ testimonials }) {
  // Interactive carousel logic
}
```

Section Component Pattern

All sections follow this pattern:

```
interface SectionProps {
  // Data from registry
  data: DataType[];

  // Optional configuration
  variant?: 'default' | 'alternate';
  className?: string;
}

export function Section({ data, variant = 'default', className }: SectionProps) {
  return (
    <section
      className={cn(
        'min-h-screen', // 100vh requirement
        'flex items-center', // Center content
        'py-16 md:py-24', // Vertical padding
        className
      )}
    >
      <div className="container">
        {/* Section content */}
      </div>
    </section>
  );
}
```

Alternating Split Layout Pattern

For alternating left/right layouts:

```
export function AlternatingSplitSection({ items }) {
  return (
    <section className="min-h-screen">
      {items.map((item, index) => (
        <div
          key={item.id}
          className={cn(
            'grid md:grid-cols-2 gap-12 min-h-screen',
            index % 2 === 0 ? 'md:flex-row' : 'md:flex-row-reverse'
          )}
        >
          {/* Content side */}
          <div className="flex flex-col justify-center">
            <h3>{item.title}</h3>
            <p>{item.description}</p>
          </div>

          {/* Visual side */}
          <div className="flex items-center justify-center">
            <Image src={item.image} alt={item.title} />
          </div>
        </div>
      ))}
    </section>
  );
}
```

Styling System

Modular CSS Architecture

1. base.css

Purpose: Foundation styles, CSS variables, resets

```
@import 'tailwindcss';

/* Theme variables */
@theme {
  --color-background: var(--background);
  --color-foreground: var(--foreground);
  /* ... more variables */
}

/* Base resets */
* {
  @apply outline-ring/50;
}

/* Typography */
h1, h2, h3, h4, h5, h6 {
  @apply font-semibold;
}
```

2. components.css

Purpose: Component-specific styles

```
/* Card styles */
.card-gradient {
  @apply bg-gradient-to-br from-card to-card/50;
}

/* Button styles */
.btn-primary {
  @apply shadow-md border-[0.5px] border-white/25;
}

/* Section styles */
.section-100vh {
  @apply min-h-screen flex items-center py-16 md:py-24;
}
```

3. animations.css

Purpose: Animation definitions and utilities

```css
/* Scroll-triggered animations */
@keyframes fade-in-up {
  from {
    opacity: 0;
    transform: translateY(30px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

.animate-fade-in-up {
  animation: fade-in-up 0.6s ease-out;
}

/* Stagger animations */
.stagger-animation > * {
  animation: fade-in-up 0.6s ease-out;
  animation-fill-mode: both;
}

.stagger-animation > *:nth-child(1) { animation-delay: 0.1s; }
.stagger-animation > *:nth-child(2) { animation-delay: 0.2s; }
.stagger-animation > *:nth-child(3) { animation-delay: 0.3s; }
```

4. theme.css

Purpose: Theme-specific overrides

```css
/* Quartz theme */
[data-theme="quartz"] {
  --radius: 0.625rem;
  --color-primary: var(--color-indigo-500);
  --color-border-illustration: color-mix(in oklab, var(--color-zinc-950) 7.5%, trans-
parent);
}

/* Dark mode */
.dark {
  --background: var(--color-zinc-950);
  --foreground: var(--color-white);
  /* ... dark mode colors */
}
```

Tailwind Configuration

```
// tailwind.config.ts
import type { Config } from 'tailwindcss';

const config: Config = {
  content: [
    './app/**/*.{js,ts,jsx,tsx,mdx}',
    './src/**/*.{js,ts,jsx,tsx,mdx}',
  ],
  darkMode: ['class'],
  theme: {
    extend: {
      colors: {
        // Quartz theme colors
        primary: 'var(--color-primary)',
        background: 'var(--color-background)',
        // ...
      },
      animation: {
        // Custom animations
        'fade-in': 'fade-in 0.6s ease-out',
        'slide-in': 'slide-in 0.5s ease-out',
      },
    },
  },
  plugins: [require('tailwindcss-animate')],
};

export default config;
```

Animation System

Scroll-Triggered Animations

Using Intersection Observer for performance:

```
// src/lib/animations.ts
export function useScrollAnimation() {
  useEffect(() => {
    const observer = new IntersectionObserver(
      (entries) => {
        entries.forEach((entry) => {
          if (entry.isIntersecting) {
            entry.target.classList.add('animate-fade-in-up');
          }
        });
      },
      { threshold: 0.1 }
    );

    // Observe elements with .animate-on-scroll class
    document.querySelectorAll('.animate-on-scroll').forEach((el) => {
      observer.observe(el);
    });

    return () => observer.disconnect();
  }, []);
}
```

Animation Types

1. **Entry Animations**: Elements fade/slide in on scroll
2. **Hover Animations**: Interactive feedback on hover
3. **Micro-interactions**: Button clicks, form interactions
4. **Loading States**: Skeleton screens, spinners
5. **Page Transitions**: Smooth navigation transitions

Performance Guidelines

- Use `transform` and `opacity` for 60fps animations
- Implement `will-change` sparingly
- Respect `prefers-reduced-motion`
- Use CSS animations over JavaScript when possible
- Lazy-load animation libraries

Routing Strategy

Static Generation (Default)

All pages are statically generated at build time:

```
// apps/web/app/services/[slug]/page.tsx
export async function generateStaticParams() {
  return SERVICES.map((service) => ({
    slug: service.slug,
  }));
}

export default function ServicePage({ params }: { params: { slug: string } }) {
  const service = getServiceBySlug(params.slug);
  return <ServiceDetail service={service} />;
}
```

Dynamic Routes

Services Routes

- `/services` - All services
- `/services/drayage` - Container drayage service
- `/services/agricultural` - Agricultural hauling
- etc.

Locations Routes

- `/locations` - All locations
- `/locations/savannah-ga` - Savannah, GA location
- `/locations/atlanta-ga` - Atlanta, GA location
- etc.

Ports Routes

- `/ports` - All ports
- `/ports/savannah` - Port of Savannah
- `/ports/charleston` - Charleston Harbor
- `/ports/jacksonville` - JAXPORT

Route Metadata

Each dynamic route generates proper metadata:

```
export async function generateMetadata({ params }: { params: { slug: string } }) {
  const service = getServiceBySlug(params.slug);

  return {
    title: `${service.name} | Southern Haulers`,
    description: service.description,
    openGraph: {
      title: service.name,
      description: service.description,
      images: [service.imageUrl],
    },
  };
}
```

SEO Strategy

On-Page SEO

1. Meta Tags

```
// app/layout.tsx
export const metadata: Metadata = {
  title: {
    default: 'Southern Haulers - Container Drayage & Agricultural Hauling',
    template: '%s | Southern Haulers',
  },
  description: 'Expert container drayage and agricultural hauling...',
  keywords: ['container drayage', 'port drayage', 'agricultural hauling'],
  openGraph: {
    type: 'website',
    locale: 'en_US',
    url: 'https://southern-haulers.com',
    siteName: 'Southern Haulers',
  },
};
```

2. Structured Data

```
// src/components/structured-data.tsx
export function OrganizationSchema() {
  const schema = {
    '@context': 'https://schema.org',
    '@type': 'Organization',
    name: 'Southern Haulers',
    url: 'https://southern-haulers.com',
    logo: 'https://upload.wikimedia.org/wikipedia/commons/thumb/3/30/South-
ern_Company_logo_new.svg/2560px-Southern_Company_logo_new.svg.png',
    // ...
  };

  return <script type="application/ld+json">{JSON.stringify(schema)}</script>;
}
```

3. Sitemap Generation

```
// app/sitemap.ts
export default function sitemap(): MetadataRoute.Sitemap {
  const services = SERVICES.map((service) => ({
    url: `https://southern-haulers.com/services/${service.slug}`,
    lastModified: new Date(),
    changeFrequency: 'monthly',
    priority: 0.8,
  }));

  const locations = LOCATIONS.map((location) => ({
    url: `https://southern-haulers.com/locations/${location.id}`,
    lastModified: new Date(),
    changeFrequency: 'monthly',
    priority: 0.7,
  }));

  return [
    {
      url: 'https://southern-haulers.com',
      lastModified: new Date(),
      changeFrequency: 'weekly',
      priority: 1,
    },
    ...services,
    ...locations,
  ];
}
```

4. Robots.txt

```
// app/robots.ts
export default function robots(): MetadataRoute.Robots {
  return {
    rules: {
      userAgent: '*',
      allow: '/',
      disallow: ['/admin/', '/api/'],
    },
    sitemap: 'https://southern-haulers.com/sitemap.xml',
  };
}
```

Local SEO

- Google Business Profile optimization
- Local schema markup
- Location-specific pages with local keywords
- NAP (Name, Address, Phone) consistency

Performance Optimization

Image Optimization

```
// Use Next.js Image component
import Image from 'next/image';

<Image
  src="/images/hero/truck.jpg"
  alt="Southern Haulers container drayage truck"
  width={1200}
  height={800}
  priority // For above-the-fold images
  placeholder="blur"
  blurDataURL="data:image/jpeg;base64,..."
/>
```

Code Splitting

```
// Dynamic imports for heavy components
import dynamic from 'next/dynamic';

const InteractiveMap = dynamic(() => import('@/components/features/port-map'), {
  loading: () => <MapSkeleton />,
  ssr: false, // Don't render on server
});
```

Font Optimization

```
// app/layout.tsx
import { Inter } from 'next/font/google';

const inter = Inter({
  subsets: ['latin'],
  display: 'swap',
  variable: '--font-inter',
});
```

Bundle Analysis

```
# Analyze bundle size
npm run build
npm run analyze
```

Adding New Sections

Step-by-Step Guide

1. Plan the Section

- Define purpose and content
- Sketch layout (100vh, split, grid, etc.)
- Identify data requirements

• Plan animations

2. Update Data Registry (if needed)

```
// apps/web/src/data/new-data.ts
export interface NewData {
  id: string;
  title: string;
  description: string;
}

export const NEW_DATA: NewData[] = [
  {
    id: 'item-1',
    title: 'Item 1',
    description: 'Description',
  },
];
```

3. Create Section Component

```
// src/components/sections/new-section.tsx
import { NewData } from '@/data/new-data';

interface NewSectionProps {
  data: NewData[];
  variant?: 'default' | 'alternate';
}

export function NewSection({ data, variant = 'default' }: NewSectionProps) {
  return (
    <section className="min-h-screen flex items-center py-16 md:py-24">
      <div className="container">
        <h2 className="text-4xl font-bold mb-8">Section Title</h2>
        <div className="grid md:grid-cols-3 gap-8">
          {data.map((item) => (
            <div key={item.id} className="p-6 rounded-lg border">
              <h3 className="text-xl font-semibold">{item.title}</h3>
              <p className="text-muted-foreground">{item.description}</p>
            </div>
          ))}
        </div>
      </div>
    </section>
  );
}
```

4. Add to Homepage

```tsx
// app/page.tsx
import { NewSection } from '@/components/sections/new-section';
import { NEW_DATA } from '@/data/new-data';

export default function HomePage() {
  return (
    <>
      {/* Existing sections */}
      <NewSection data={NEW_DATA} />
      {/* More sections */}
    </>
  );
}
```

5. Add Images (if needed)

```
# Add images to public/images/
public/images/new-section/
  ├── image-1.jpg
  ├── image-2.jpg
  └── ...
```

6. Test and Optimize

- Test on mobile, tablet, desktop
- Verify dark mode styling
- Check accessibility (keyboard nav, screen readers)
- Optimize images
- Test animations
- Verify SEO (meta tags, structured data)

Conclusion

This architecture provides a solid foundation for a scalable, maintainable, and performant landing page. By following these patterns and principles, you can easily extend the application with new features while maintaining consistency and quality.

Key Takeaways

1. **Data First**: Always start with data registries
2. **Component Modularity**: Build small, reusable components
3. **Type Safety**: Leverage TypeScript for reliability
4. **Performance**: Optimize images, code split, lazy load
5. **Accessibility**: Build for everyone
6. **SEO**: Think about search engines from the start
7. **Maintainability**: Write code others (and future you) can understand

---

Last Updated: October 28, 2025

Architecture Version: 1.0

Maintained By: Southern Haulers Development Team