Stage 1 — Critical Performance Fixes (Final)

Architecture: React 19 + TypeScript + Vite 7 + Tailwind v4 + `vite-plugin-singlefile`
URL: `analytics.whiteknight.academy`
Current PageSpeed Mobile: Performance 45 · FCP 10.0s · LCP 12.2s · TBI 410ms · CLS 0.064
Goal after Stage 1: Performance 75+ · FCP <3s · LCP <4s

Root Cause Analysis

`vite-plugin-singlefile` **is the single biggest performance killer.** Everything else is secondary.

Here is what this plugin does: it takes the entire Vite build output — all JavaScript, all CSS, all font files — and inlines them as base64 into one HTML file. The browser must download, parse, and execute this entire monolith before rendering a single pixel.

Concrete damage:

What gets inlined	Approximate size	Effect
10 `@fontsource` CSS files (Unbounded 6 weights + Manrope 4 weights)	~800 KB–1.2 MB (base64 WOFF2)	Blocks render until all fonts are parsed
Tailwind CSS output	~30–50 KB (purged, which is fine)	Normally cacheable, now stuck in HTML
React + React DOM + React Router	~150 KB	Cannot be cached between pages
Stripe JS (`@stripe/stripe-js`)	~50 KB + triggers ~400 KB download from Stripe CDN	Loads even on landing page where checkout is not needed
`lucide-react` icons	~20–40 KB (tree-shaken)	Cannot be code-split
`DashboardLayout` + `CheckoutPage` code	Unknown, but not needed on `/`	Dead code on landing page, still downloaded
Total single HTML file	**Likely 2–3 MB+**	**10s FCP on mobile because the entire file must download and parse before anything renders**

Fix 1.1 — Remove `vite-plugin-singlefile`

This one change will have more impact than all other fixes combined.

Step 1: Uninstall the plugin

```bash
npm uninstall vite-plugin-singlefile
```

Step 2: Update `vite.config.ts`

```ts
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import tailwindcss from '@tailwindcss/vite'

export default defineConfig({
  plugins: [react(), tailwindcss()],
  build: {
    rollupOptions: {
      output: {
        // Separate vendor chunks for better caching
        manualChunks: {
          'react-vendor': ['react', 'react-dom', 'react-router-dom'],
        },
      },
    },
  },
})
```

Step 3: Rebuild

```bash
npm run build
```

Step 4: Verify the output

```bash
ls -la dist/assets/
```

You should now see multiple files:

- `index-[hash].js` — your app code (~50–100 KB)
- `react-vendor-[hash].js` — React runtime (~150 KB, cacheable)
- `index-[hash].css` — Tailwind CSS (~30–50 KB)
- Several `.woff2` font files in `dist/assets/`

The total is similar, but now:

- The browser downloads files **in parallel** (not sequentially in one blob)
- CSS and JS are **separately cacheable** (return visits are instant)
- Gzip compresses each file **much more efficiently** than one giant file
- Code-splitting becomes possible (Fix 1.4)

Step 5: Deploy

Upload the contents of `dist/` to your subdomain document root via FTP (same as before, but now it is a folder of files instead of one HTML file).

Make sure `.htaccess` is in place for SPA routing (it should already exist from your implementation plan):

```apache
<IfModule mod_rewrite.c>
  RewriteEngine On
  RewriteBase /
  RewriteRule ^index\.html$ - [L]
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteRule . /index.html [L]
</IfModule>
```

Verification:

- PageSpeed → FCP should drop from 10s to ~3–5s just from this change
- DevTools → Network → you should see 5–10 separate file requests instead of one
- DevTools → Coverage → unused CSS should drop dramatically (Tailwind v4 purges correctly when not inlined)

Fix 1.2 — Reduce font weight imports

Problem: You import 10 font weight files:

- Unbounded: 400, 500, 600, 700, 800, 900 (6 files)
- Manrope: 400, 500, 600, 700 (4 files)

Each WOFF2 file is ~20–40 KB. That is 200–400 KB of fonts. On a landing page, you almost certainly do not use all 10 weights.

Action: Audit and trim

1. Open DevTools → Elements → select text elements across the page → check Computed styles → look at `font-weight` values actually used.
2. In `src/main.tsx`, keep only the weights you actually use. A typical landing page needs 3–4 weights:

```tsx
// Fonts — only import weights you actually use
import '@fontsource/unbounded/700.css';  // Bold — headings
import '@fontsource/unbounded/800.css';  // ExtraBold — hero title

import '@fontsource/manrope/400.css';    // Regular — body text
import '@fontsource/manrope/600.css';    // Semibold — buttons, labels
```

3. Delete all other `@fontsource` imports.
4. After removing weights, search your codebase for any `font-weight` values that reference removed weights. For example, if you remove Unbounded 400 but have `font-normal` on an Unbounded element, the browser will synthesize (faux bold/normal) which looks bad. Change those to a weight you kept.

Verification:

- Rebuild → check `dist/assets/` → font files should be 4 instead of 10
- Total font payload: ~80–120 KB instead of ~200–400 KB
- No visual differences on the page (all text still looks correct)

Fix 1.3 — Lazy-load Stripe (do not load on landing page)

Problem: `@stripe/stripe-js` (~50 KB) is imported at the top level in `CheckoutPage`. Since React Router imports all route components eagerly in `App.tsx`, Stripe code is bundled and downloaded even when the user visits `/` (the landing page) and never goes to `/checkout`.

Action:

1. In `App.tsx`, lazy-load the checkout route:

```tsx
```

```
import { lazy, Suspense } from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import LandingPage from './components/pages/LandingPage';

// Lazy-load pages that are not the landing page
const CheckoutPage = lazy(() => import('./components/pages/CheckoutPage'));
const DashboardLayout = lazy(() => import('./DashboardLayout'));

const App = () => {
 return (
  <Router>
   <Routes>
    <Route path="/" element={<LandingPage />} />
    <Route
     path="/checkout"
     element={
      <Suspense fallback={<div className="min-h-screen bg-[#080C14]" />}>
       <CheckoutPage />
      </Suspense>
     }
    />
    <Route
     path="/dashboard"
     element={
      <Suspense fallback={<div className="min-h-screen bg-[#080C14]" />}>
       <DashboardLayout />
      </Suspense>
     }
    />
   </Routes>
  </Router>
 );
};

export default App;
```

2. This ensures:
    - Landing page (`/`) downloads only LandingPage code + shared React runtime
    - Stripe JS only downloads when user navigates to `/checkout`
    - DashboardLayout only downloads when user navigates to `/dashboard`
3. If there is a Stripe Pricing Table (`<stripe-pricing-table>`) embedded directly on the landing page (visible in screenshots at the bottom), move it into a lazy-loaded component:

Create `src/components/StripePricingLazy.tsx`:

tsx

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

export default function StripePricingLazy({
  pricingTableId,
  publishableKey,
}: {
  pricingTableId: string;
  publishableKey: string;
}) {
  const ref = useRef<HTMLDivElement>(null);
  const [loaded, setLoaded] = useState(false);

  useEffect(() => {
    const el = ref.current;
    if (!el) return;

    const observer = new IntersectionObserver(
      ([entry]) => {
        if (entry.isIntersecting) {
          const script = document.createElement('script');
          script.src = 'https://js.stripe.com/v3/pricing-table.js';
          script.async = true;
          script.onload = () => setLoaded(true);
          document.head.appendChild(script);
          observer.disconnect();
        }
      },
      { rootMargin: '300px' }
    );

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

  return (
    <div ref={ref} className="min-h-[400px]">
      {loaded ? (
        // @ts-expect-error — Stripe custom element
        <stripe-pricing-table
          pricing-table-id={pricingTableId}
          publishable-key={publishableKey}
        />
      ) : (
        <div className="h-[400px] flex items-center justify-center text-slate-500 text-sm">
          Loading pricing...
        </div>
```

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

Replace the direct `<stripe-pricing-table>` on the landing page with this component.

Verification:

- Visit `/` → DevTools Network → filter "stripe" → **zero** Stripe requests
- Navigate to `/checkout` → Stripe JS downloads at that point
- Scroll to pricing section on landing page → Stripe pricing table loads
- Build output → you should see a separate `CheckoutPage-[hash].js` chunk

Fix 1.4 — Code-split landing page sections

Problem: The landing page itself contains multiple heavy sections (Hero, Analytics chart, Opponent Prep, Coaching, Beginners, Results, Pricing, Footer). All are in one bundle. Only the Hero and Nav need to render immediately.

Action:

In `LandingPage.tsx` (or wherever sections are composed):

```tsx
```

```
import { lazy, Suspense } from 'react';

// Above the fold — load immediately
import Navbar from '../Navbar';
import Hero from '../sections/Hero';

// Below the fold — lazy load
const Analytics = lazy(() => import('../sections/Analytics'));
const OpponentPrep = lazy(() => import('../sections/OpponentPrep'));
const Coaching = lazy(() => import('../sections/Coaching'));
const Beginners = lazy(() => import('../sections/Beginners'));
const Results = lazy(() => import('../sections/Results'));
const Pricing = lazy(() => import('../sections/Pricing'));
const Footer = lazy(() => import('../sections/Footer'));

export default function LandingPage() {
  return (
    <div className="bg-[#080C14] min-h-screen text-white relative font-sans overflow-x-hidden">
      <Navbar />
      <Hero />
      <Suspense fallback={null}>
        <Analytics />
        <OpponentPrep />
        <Coaching />
        <Beginners />
        <Results />
        <Pricing />
        <Footer />
      </Suspense>
    </div>
  );
}
```

Each section component must use `export default`. Adjust import paths to match your actual file structure.

Verification:

- `npm run build` → `dist/assets/` should have 8–12 JS chunks instead of 1–2
- Landing page initial load downloads only ~100–150 KB of JS (hero + nav + react)
- Other sections load as browser becomes idle
- PageSpeed → TBI should decrease significantly

Fix 1.5 — Enable Gzip + cache headers via `.htaccess`

Problem: Now that we have separate files (after removing singlefile plugin), we need proper compression and caching.

Action:

Update `.htaccess` at your subdomain document root:

apache

```
# SPA Routing
<IfModule mod_rewrite.c>
  RewriteEngine On
  RewriteBase /
  RewriteRule ^index\.html$ - [L]
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteRule . /index.html [L]
</IfModule>

# Gzip Compression
<IfModule mod_deflate.c>
  AddOutputFilterByType DEFLATE text/html
  AddOutputFilterByType DEFLATE text/css
  AddOutputFilterByType DEFLATE application/javascript
  AddOutputFilterByType DEFLATE application/json
  AddOutputFilterByType DEFLATE image/svg+xml
  AddOutputFilterByType DEFLATE text/plain
</IfModule>

# Cache Policy — Vite hashes filenames, so long cache is safe
<IfModule mod_expires.c>
  ExpiresActive On
  ExpiresByType text/html "access plus 10 minutes"
  ExpiresByType text/css "access plus 1 year"
  ExpiresByType application/javascript "access plus 1 year"
  ExpiresByType image/jpeg "access plus 1 year"
  ExpiresByType image/png "access plus 1 year"
  ExpiresByType image/webp "access plus 1 year"
  ExpiresByType image/svg+xml "access plus 1 year"
  ExpiresByType font/woff2 "access plus 1 year"
  ExpiresByType font/woff "access plus 1 year"
</IfModule>

# Security
<IfModule mod_headers.c>
  Header set X-Content-Type-Options "nosniff"
  Header set Referrer-Policy "strict-origin-when-cross-origin"
</IfModule>
```

Verification:

- DevTools → Network → Response Headers → confirm `Content-Encoding: gzip`
- Confirm `Cache-Control` or `Expires` present on `.js` / `.css` files
- Second page load should show `(disk cache)` for all static assets

Execution Order

Step	Fix	What Changes	Expected Impact
1	**Remove `vite-plugin-singlefile`**	`vite.config.ts`, deploy process	FCP: 10s → 4–5s. This is the biggest single win.
2	**Trim font weights**	`src/main.tsx`	Saves 100–300 KB of font data
3	**Lazy-load Stripe + routes**	`src/App.tsx`, new component	Saves ~450 KB JS on landing page
4	**Code-split landing sections**	`LandingPage.tsx`	Initial JS drops to ~100–150 KB
5	**Gzip + cache `.htaccess`**	`.htaccess` on server	All transfers ~70% smaller
	Run final PageSpeed		**Target: Mobile 75+, FCP <3s, LCP <4s**

Run PageSpeed after **each step** to measure isolated impact.

Summary of file changes

File	Action
`vite.config.ts`	Remove `viteSingleFile()`, add `manualChunks`
`package.json`	`npm uninstall vite-plugin-singlefile`
`src/main.tsx`	Remove unused `@fontsource` weight imports
`src/App.tsx`	`React.lazy()` for CheckoutPage + DashboardLayout
`src/components/pages/LandingPage.tsx`	`React.lazy()` for below-fold sections
New: `src/components/StripePricingLazy.tsx`	Lazy Stripe embed via IntersectionObserver
`.htaccess` (server)	Add gzip + cache rules
Deploy script (`deploy.js`)	Update to upload `dist/` folder contents (not single file)

What NOT to do

- **Do not keep** `vite-plugin-singlefile` and try to optimize around it. The plugin fundamentally prevents every standard web performance optimization (parallel loading, caching, code-splitting, compression efficiency). Remove it.
- **Do not add SSR or Next.js.** The Vite SPA is the right architecture for this project. The problem was never the framework — it was the singlefile plugin.
- **Do not self-host fonts manually.** `@fontsource` already self-hosts them through npm. Once the singlefile plugin is removed, Vite will output them as proper `.woff2` files that load correctly.
- **Do not add a loading spinner or skeleton.** Fix the root cause, not the symptom.