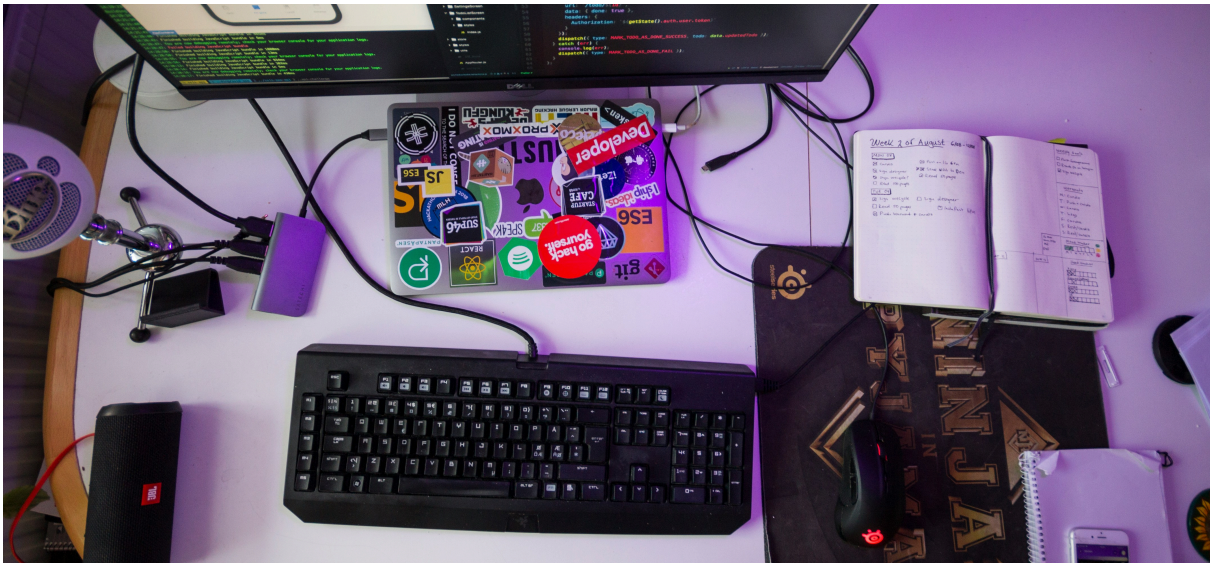

The Complete Guide to Facebook Pixel Integration in React with TypeScript

Introduction

Integrating Facebook Pixel (now Meta Pixel) into a React application with TypeScript can be tricky due to:

- **TypeScript type errors** (`Property 'fbq' does not exist on type 'Window'`)
- **React hydration issues** (Pixel not loading in Single Page Apps)
- **Event tracking challenges** (custom conversions, form submissions)

This guide provides a **step-by-step** implementation with **best practices** and **debugging tips**.

Step 1: Setting Up the Meta Pixel in Facebook Events Manager

1. Go to Meta Events Manager.
2. Click **Connect Data Sources** → **Web** → **Meta Pixel**.
3. Name your Pixel (e.g., `Krishna_Pixel`).
4. Copy the **Pixel ID** (e.g., `727444xxxxx`).
5. Copy manual code from pixel.

Step 2: Adding Global Type Declarations for TypeScript

If you don't do this then you will get a typescript error. To fix `Property 'fbq' does not exist on type 'Window'` , create a global type definition.

Option A: Using `global.d.ts` (Recommended)

1. Create `src/types/global.d.ts` :

```
interface FacebookPixel {
  (command: 'init', pixelId: string): void;
  (command: 'track', event: string, parameters?: Record<string, unknown>): void;
```

```
  (command: 'trackCustom', event: string, parameters?: Record<string, unknown>): void;
  push: (...args: unknown[]) => void;
}

declare global {
  interface Window {
    fbq?: FacebookPixel;
  }
}

export {};
```

2. Update `tsconfig.json` to include custom types:

```
{
  "compilerOptions": {
    "typeRoots": ["./node_modules/@types", "./src/types"],
    "types": []
  },
  "include": ["src/**/*", "src/types/**/*"]
}
```

Option B: Quick Fix (Inline Declaration)

If you don't want a global file, add this at the top of any component using `fbq`:

```
declare const fbq: {
  (command: 'init', pixelId: string): void;
  (command: 'track', event: string, parameters?: unknown): void;
};
```

Step 3: Implementing the Pixel in React

Option A: Using `react-helmet` (Best for SSR/SEO)

1. Install `react-helmet`:

```
npm install react-helmet
```

2. Create `FacebookPixel.tsx`:

```
import React, { useEffect } from 'react';
import { Helmet } from 'react-helmet';

const FacebookPixel = () => {
  useEffect(() => {
    const initPixel = () => {
      if (window.fbq) {
        window.fbq('init', '2371832499868359');
```

```
        window.fbq('track', 'PageView');
      } else {
        setTimeout(initPixel, 100); // Retry if not loaded
      }
    };
    initPixel();
  }, []);

  return (
    <>
      <Helmet>
        <script>
          {`
            Your manual script code from Facebook Pixel
          `}
        </script>
      </Helmet>
      <noscript>
        Your manual noscript code from Facebook Pixel
      </noscript>
    </>);
};

export default FacebookPixel;
```

3. Add it to `App.tsx`:

```
import FacebookPixel from './components/FacebookPixel';

function App() {
  return (
    <>
      <FacebookPixel />
      {/* Rest of your app */}
    </>);
}
```

Option B: Direct Script Injection (Simpler for SPAs)

If you don't need SSR, inject the script directly into `index.html`:

```
<head>
  <script>
    Manual code here
    </script>
</head>
<body>
  <noscript>
```

```
    manual code here
  </noscript>
</body>
```

Step 4: Tracking Custom Events (Form Submissions, Clicks)

1. Form Submission Tracking

```
const handleSubmit = async (e: React.FormEvent) => {
  e.preventDefault();

  // Track form start
  if (window.fbq) window.fbq('track', 'Lead');

  try {
    const response = await submitForm(data);

    // Track success
    if (window.fbq) {
      window.fbq('track', 'CompleteRegistration', {
        value: 100, // Estimated conversion value
        currency: 'INR',
      });
    }
  } catch (error) {
    // Track failure
    if (window.fbq) window.fbq('trackCustom', 'FormError');
  }
};
```

2. Button Click Tracking

```
<button
  onClick={() => {
    if (window.fbq) window.fbq('track', 'Contact');
  }}>
  Contact Us
</button>
```

3. Route Change Tracking (React Router)

```
import { useEffect } from 'react';
import { useLocation } from 'react-router-dom';

const RouteTracker = () => {
  const location = useLocation();
```

```
  useEffect(() => {
    if (window.fbq) {
      window.fbq('track', 'PageView');
    }
  }, [location]);

  return null;
};

// Add to App.tsx
<Router>
  <RouteTracker />
  {/* Routes */}
</Router>
```

Step 5: Debugging & Verification

1. **Meta Pixel Helper** (Chrome Extension)
   - Shows active Pixels
   - Lists fired events
   - Detects errors
2. **Test Events in Meta Events Manager**
   - Go to **Test Events** tab
   - Check if events appear in real-time
3. **Common Issues & Fixes**
   - **Pixel not loading?**

     → Check ad blockers, use `noscript` fallback.
   - **Events not firing?**

     → Ensure `fbq` is defined before tracking.
   - **TypeScript errors?**

     → Verify `global.d.ts` is included in `tsconfig.json`.

Conclusion

By following this guide, you've:

✅ Properly typed `fbq` for TypeScript

✅ Implemented Pixel loading in React (with SSR support)

✅ Added custom event tracking (forms, clicks, routes)

✅ Set up debugging with Meta Pixel Helper

Next Steps:

- Set up **Conversions API** for server-side tracking
- Use **Advanced Matching** for better attribution
- Define **Custom Audiences** based on user actions

"Struggled with Pixel integration? Share your war stories in the comments!"