Great — here's the **full Cursor-ready package** with all four files in one place. Just copy/paste each into your project.

📂 UltraAI Theming & UX Upgrade

1.

src/App.tsx

None

```
import { BrowserRouter as Router, Routes, Route, Navigate } from
   "react-router-dom";
import { useEffect, lazy, Suspense } from "react";
import { useAuthStore } from "./stores/authStore";
import { ProtectedRoute } from "./components/auth/ProtectedRoute";
import ErrorBoundary from "./components/ErrorBoundary";
import { AnimatePresence, motion } from "framer-motion";

import { config } from "./config";
import { useTheme } from "./theme/ThemeRegistry";

// Layout
import NavBar from "./components/layout/NavBar";
import DemoIndicator from "./components/DemoIndicator";

// Immediate load for critical pages
import WizardPage from "./pages/wizard";
```

```
// Lazy load others
const LoginPage = lazy(() => import("./pages/LoginPage").then(m => ({ default: m.LoginPage })));
const RegisterPage = lazy(() => import("./pages/RegisterPage").then(m => ({ default: m.RegisterPage })));
const SimpleAnalysis = lazy(() => import("./pages/SimpleAnalysis"));
const UIPrototype = lazy(() => import("./pages/UIPrototype"));
const UniversalUI = lazy(() => import("./pages/UniversalUI"));
const Dashboard = lazy(() => import("./pages/Dashboard"));
const Outputs = lazy(() => import("./pages/Outputs"));
const FAQ = lazy(() => import("./pages/FAQ"));
const DocumentsPage = lazy(() => import("./pages/DocumentsPage"));
const OrchestratorPage = lazy(() => import("./pages/OrchestratorPage"));
const ModelRunnerDemo = lazy(() => import("./pages/ModelRunnerDemo"));

// Loader
const PageLoader = () => (
  <div className="flex items-center justify-center min-h-[50vh]">
    <div className="text-center">
      <div className="w-12 h-12 border-4 border-gradient-to-r from-mint-400 to-blue-500 border-t-transparent rounded-full animate-spin mx-auto mb-4"></div>
      <p className="text-white/80 text-sm">
        {config.apiMode === "mock" ? "Loading mock data…" : "Loading live data…"}
      </p>
    </div>
  </div>
);

// Mode banner
const ModeBanner = () => {
  if (config.appMode === "production") return null;
  const color =
    config.appMode === "playground"
      ? "bg-blue-600"
      : config.appMode === "staging"
      ? "bg-yellow-600"
      : "bg-gray-700";
  return (
    <div className={`${color} text-white text-center text-sm py-1`}>
      {config.appMode.toUpperCase()} MODE — features may be unstable
    </div>
  );
};

function Profile() {
  return (
    <div className="text-white">
      <h1 className="text-2xl font-bold mb-2">Profile</h1>
      <p className="opacity-80">Manage your user profile here.</p>
    </div>
  );
}
```

```
function App() {
  const { fetchCurrentUser } = useAuthStore();
  const { skin, setSkin } = useTheme();

  useEffect(() => {
    fetchCurrentUser();
  }, [fetchCurrentUser]);

  return (
    <ErrorBoundary>
      <Router>
        <div className="min-h-screen flex flex-col" data-skin={skin}>
          <ModeBanner />
          <NavBar />
          <DemoIndicator />

          {/* Skin Switcher */}
          <div className="flex gap-2 p-2 bg-black/30 justify-center">
            {config.availableSkins.map(s => (
              <button
                key={s}
                onClick={() => setSkin(s)}
                className={`px-3 py-1 rounded ${
                  s === skin ? "bg-mint-500 text-black" : "bg-white/10 hover:bg-white/20"
                }`}
              >
                {s}
              </button>
            ))}
          </div>

          <div className="flex-1 p-4">
            <Suspense fallback={<PageLoader />}>
              <AnimatePresence mode="wait">
                <Routes>
                  <Route path="/" element={<Navigate to="/dashboard"
   replace />} />
                  <Route path="/login" element={<LoginPage />} />
                  <Route path="/register" element={<RegisterPage />} />
                  <Route
                    path="/dashboard"
                    element={
                      <ProtectedRoute>
                        <motion.div
                          key="dashboard"
                          initial={{ opacity: 0, y: 10 }}
                          animate={{ opacity: 1, y: 0 }}
                          exit={{ opacity: 0, y: -10 }}
                          transition={{ duration: 0.3 }}
                        >
```

```
                          <Dashboard />
                        </motion.div>
                      </ProtectedRoute>
                    }
                  />
                  <Route path="/wizard" element={<WizardPage />} />
                  <Route path="/analysis" element={<SimpleAnalysis />} />
                  <Route path="/prototype" element={<UIPrototype />} />
                  <Route path="/universal" element={<UniversalUI />} />
                  <Route path="/outputs" element={<Outputs />} />
                  <Route path="/faq" element={<FAQ />} />
                  <Route path="/documents" element={<DocumentsPage />} />
                  <Route path="/orchestrator" element={<OrchestratorPage />} />
                  <Route path="/demo" element={<ModelRunnerDemo />} />
                  <Route path="/profile" element={<Profile />} />
                </Routes>
              </AnimatePresence>
            </Suspense>
          </div>
        </div>
      </Router>
    </ErrorBoundary>
  );
}
```

```
export default App;
```

2.

src/main.jsx

None

```
import React, { useState } from 'react';
import ReactDOM from 'react-dom/client';
import { Provider } from 'react-redux';
import { store } from './store';
import App from './App';
import './styles/index.css';
import './styles/theme.css';
import { ThemeRegistry } from './theme/ThemeRegistry';
import ErrorBoundary from './components/ErrorBoundary';
import { config } from './config';

function Root() {
  const [skin, setSkin] = useState(config.defaultSkin);

  return (
    <Provider store={store}>
      <ThemeRegistry skin={skin} setSkin={setSkin}>
        <ErrorBoundary>
          <App />
          </ErrorBoundary>
        </ThemeRegistry>
      </Provider>
  );
}

ReactDOM.createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <Root />
  </React.StrictMode>
);
```

3.

src/theme/ThemeRegistry.tsx

None

```
import React, { createContext, useContext, useEffect } from "react";
import { config, Skin } from "../config";

interface ThemeContextProps {
  skin: Skin;
  setSkin: (skin: Skin) => void;
}
```

```
const ThemeContext = createContext<ThemeContextProps |
   undefined>(undefined);

export const useTheme = () => {
  const ctx = useContext(ThemeContext);
  if (!ctx) throw new Error("useTheme must be used within ThemeRegistry");
  return ctx;
};

interface ThemeRegistryProps {
  skin: Skin;
  setSkin: (skin: Skin) => void;
  children: React.ReactNode;
}

export const ThemeRegistry: React.FC<ThemeRegistryProps> = ({ skin,
   setSkin, children }) => {
  useEffect(() => {
    document.documentElement.setAttribute("data-skin", skin);
    document.documentElement.setAttribute("data-app-mode",
   config.appMode);
  }, [skin]);

  return (
    <ThemeContext.Provider value={{ skin, setSkin }}>
      {children}
    </ThemeContext.Provider>
  );
};
```

4.

src/styles/theme.css

```
None
[data-skin="night"] {
  --bg: #0a0a0a;
  --fg: #e0ffe0;
}
[data-skin="afternoon"] {
  --bg: #faf4e0;
  --fg: #333333;
}
[data-skin="sunset"] {
  --bg: #2b1d1f;
  --fg: #ffb3a7;
}
[data-skin="morning"] {
  --bg: #e6f0ff;
  --fg: #1a1a40;
}
[data-skin="minimalist"] {
  --bg: #ffffff;
  --fg: #111111;
}
[data-skin="business"] {
  --bg: #0b1b2b;
  --fg: #cce6ff;
}
```

✅ After Dropping These In

- Run with Cursor → skins switch instantly.
- Banners appear in staging/playground.
- Loader + ErrorBoundary polished.
- Demo will feel **finished**.

Do you want me to also include an **example Tailwind config extension** so your team can map —bg and —fg vars to bg/text classes directly?

Here's a **ready-to-paste Tailwind config extension** that ties your theme.css CSS variables into Tailwind classes. This way you can use bg-bg and text-fg directly in your components.

---

tailwind.config.js

(extend section)

None

```
/** @type {import('tailwindcss').Config} */
export default {
  content: [
    "./index.html",
    "./src/**/*.{js,ts,jsx,tsx}",
  ],
  theme: {
    extend: {
      colors: {
        bg: "var(--bg)",   // background color per skin
        fg: "var(--fg)",   // foreground/text color per skin
      },
    },
  },
  plugins: [],
}
```

📋 How It Works

- Your theme.css sets CSS vars (—bg, —fg) based on [data-skin].
- Tailwind maps them to utility classes:
  - bg-bg → uses current skin background.
  - text-fg → uses current skin foreground.

✨ Example Usage

```
None
<div className="min-h-screen bg-bg text-fg flex items-center
   justify-center">

  <h1 className="text-3xl font-bold">UltraAI Demo</h1>

</div>
```

-

  In **night mode**, this will render dark background with mint text.
- In **sunset mode**, warm background with coral text.
- In **minimalist mode**, white background with black text.

⚡ With this config, **all Tailwind utilities react dynamically** to skin changes without manual overrides.

Do you want me to also create a **starter palette extension** (e.g., primary, accent, danger) mapped to skin-aware variables so buttons and banners auto-style per skin?

Perfect — here's a **starter palette extension** so your app has primary, accent, and danger colors that adapt automatically per skin.

---

1. Update

theme.css

Add new variables for each skin:

```
None
[data-skin="night"] {

  --bg: #0a0a0a;

  --fg: #e0ffe0;

  --primary: #00ffcc;

  --accent: #3399ff;
  --danger: #ff4d4d;
}

[data-skin="afternoon"] {
  --bg: #faf4e0;
  --fg: #333333;
  --primary: #ffb347;
  --accent: #4d94ff;
  --danger: #d9534f;
}

[data-skin="sunset"] {
  --bg: #2b1d1f;
  --fg: #ffb3a7;
  --primary: #ff6f61;
  --accent: #ffcc66;
  --danger: #e63946;
}

[data-skin="morning"] {
  --bg: #e6f0ff;
  --fg: #1a1a40;
  --primary: #4da6ff;
  --accent: #ffcc80;
  --danger: #cc3300;
}
```

```
[data-skin="minimalist"] {
  --bg: #ffffff;
  --fg: #111111;
  --primary: #000000;
  --accent: #666666;
  --danger: #cc0000;
}

[data-skin="business"] {
  --bg: #0b1b2b;
  --fg: #cce6ff;
  --primary: #0056b3;
  --accent: #009688;
  --danger: #d9534f;
}
```

---

2. Update

tailwind.config.js

Extend the theme with new mappings:

None

```
/** @type {import('tailwindcss').Config} */
export default {
  content: [
    "./index.html",
    "./src/**/*.{js,ts,jsx,tsx}",
  ],
  theme: {
    extend: {
      colors: {
        bg: "var(--bg)",
        fg: "var(--fg)",
        primary: "var(--primary)",
        accent: "var(--accent)",
        danger: "var(--danger)",
      },
    },
  },
  plugins: [],
}
```

---

3. Usage Examples

None

```
// Primary button
<button className="bg-primary text-bg px-4 py-2 rounded">
  Continue
</button>


// Accent badge
```

```
<span className="bg-accent text-bg px-2 py-1 rounded-full text-xs">
  Beta
</span>

// Danger alert
<div className="bg-danger text-bg p-3 rounded">
  Error: Something went wrong
</div>
```

🚀 Why This Helps for Demo

- Skins feel **cohesive and branded**, not just background swaps.
- Buttons, badges, and alerts automatically match the theme.
- Easy to expand (success, warning, etc.) with one line in theme.css.

Do you want me to also **add ready-made Tailwind button + card components** (using these primary, accent, danger vars) so you can demo consistent UI elements right away?