Gemini Pro 2.0 Prompts - FWSR Critical Fixes

🚨 CRITICAL ISSUE #1: Remove "Official" Claims & Rebrand

I'm building a sustainability compliance platform for fashion brands applying to Berlin Fashion Week. Currently, I'm incorrectly positioning it as "Official Compliance Resource" without authorization.

Please help me rebrand the entire application to position as an independent expert consultancy instead:

1. Replace ALL instances of "official" language with professional but independent positioning
2. Change "Independently operated by the FWSR Board" to accurate description
3. Update the footer copyright and tagline
4. Rewrite the About page to clarify we're expert consultants, NOT affiliated with Berlin Fashion Week
5. Add clear disclaimers that we're independent and not endorsed by BFW

Current problematic text in App.tsx:
- "Official Compliance Resource" in footer
- "FWSR Board" references
- "The global compliance hub for Fashion Week"

Please provide:
- New brand positioning statement
- Updated copy for all pages (Landing, About, Footer, Nav)
- Professional disclaimer text to add to bottom of all pages
- New metadata description for metadata.json

Make it professional, authoritative, but clearly independent. Use language like "expert consultancy," "specialized compliance services," "industry-leading assessment tools."

🚨 CRITICAL ISSUE #2: Add Legal Documents (ToS, Privacy Policy)

I need to create legally compliant Terms of Service and Privacy Policy for my FWSR platform before launch. This is a B2B SaaS platform that:

- Serves fashion brands in the EU (Germany specifically)
- Collects payment via Stripe ($19-$595 tiers)
- Uses Supabase for authentication and data storage
- Uses Google Gemini AI for chatbot features
- Stores user assessment data and compliance results
- Must comply with GDPR

Please create:

1. **Terms of Service** that includes:
   - Service description and scope
   - User responsibilities
   - Payment terms and refund policy
   - Limitation of liability (critical: our assessments are guidance, not guarantees)
   - Disclaimer that we're not affiliated with Berlin Fashion Week
   - Intellectual property rights
   - Governing law (Germany)

2. **Privacy Policy (GDPR-compliant)** that covers:
   - What data we collect (email, payment info, assessment responses)
   - How we use it (service delivery, AI processing)
   - Third-party services (Stripe, Supabase, Google Gemini)
   - User rights (access, deletion, portability)
   - Cookie usage
   - Data retention periods
   - Contact information for data requests

3. **Cookie Consent Banner** component in React

4. **Professional Liability Disclaimer** to add to assessment results pages

Format as:
- React components for ToS and Privacy pages
- Add routes to App.tsx navigation
- Make footer links functional
- Include modal for cookie consent that appears on first visit

---

🚨 CRITICAL ISSUE #3: Build the RiskCalculator Component (Core Product)

I need to build the RiskCalculator component - this is the core product of my FWSR platform. It's a comprehensive assessment of the 19 Minimum Sustainability Standards for Berlin Fashion Week.

Requirements:

The 19 Pillars are:
1. Sustainability Strategy & Transparency
2. Supply Chain Mapping
3. Restricted Substances List (RSL)
4. Chemical Management
5. Water & Energy Management
6. Waste Management & Circularity
7. Carbon Footprint Measurement

8. Science-Based Targets
9. Social Code of Conduct
10. Living Wages
11. Worker Health & Safety
12. Freedom of Association
13. Gender Equality
14. Child Labor Prevention
15. Forced Labor Prevention
16. Animal Welfare Standards
17. Biodiversity Protection
18. Packaging Standards
19. Third-Party Certifications

Build a RiskCalculator component that:

1. **Multi-step form** - One section per pillar (19 total steps)
2. **Questions per pillar** - 3-5 questions each with options:
   - "Yes, fully implemented" (compliant)
   - "Partially implemented" (at risk)
   - "No, not implemented" (non-compliant)
   - "Unsure / Need guidance" (requires consultation)

3. **Progress tracking** - Visual progress bar showing 1/19, 2/19, etc.

4. **Scoring logic** - Calculate compliance percentage per pillar

5. **Data structure** - Store results in format:
```typescript
interface AssessmentResult {
  pillarId: string;
  pillarName: string;
  score: number; // 0-100
  responses: QuestionResponse[];
  status: 'compliant' | 'at-risk' | 'non-compliant';
}
```

6. **UI Requirements:**
   - Clean, professional design matching existing FWSR aesthetic
   - Mobile responsive
   - Save progress to Supabase (so users can resume)
   - Show estimated time remaining
   - Add helpful tooltips explaining technical terms

7. **On completion:**
   - Navigate to ResultPage with full assessment data
   - Store in Supabase linked to user account

- Generate PDF download option

Please provide the complete RiskCalculator.tsx component with all 19 pillars, sample questions for each, and the ResultPage component that displays a detailed compliance report.

🚨 CRITICAL ISSUE #4: Fix API Key Security (Move to Backend)

My current implementation exposes the Gemini API key in the frontend code via vite.config.ts. This is a critical security vulnerability.

I need to:

1. **Create a secure backend API** to handle Gemini AI calls
2. **Remove API key from frontend** completely
3. **Implement proper authentication** so only paid users can access the chatbot

Current setup:
- Using Vite + React (frontend only)
- Gemini API key defined in vite.config.ts
- Supabase for auth
- Need to verify user has purchased "The Auditor" ($595 plan) before allowing chat

Please provide:

1. A serverless function (Supabase Edge Function or similar) that:
   - Receives chat messages from frontend
   - Verifies user is authenticated via Supabase auth
   - Checks user has purchased "auditor" plan (from Supabase database)
   - Calls Gemini API server-side with the API key
   - Returns response to frontend

2. Updated AuditorChat.tsx component that:
   - Calls the secure backend endpoint instead of Gemini directly
   - Handles authentication errors
   - Shows upgrade prompt if user hasn't purchased auditor tier

3. Database schema for tracking purchases:
```sql
CREATE TABLE user_purchases (
  id UUID PRIMARY KEY,
  user_id UUID REFERENCES auth.users(id),
  plan_id TEXT, -- 'survey', 'chat', 'auditor'
  purchase_date TIMESTAMP,
  stripe_payment_id TEXT,
```

```
  active BOOLEAN
);
```

4. Row Level Security (RLS) policies for the purchases table

Format as complete, copy-paste ready code with clear file structure.

⚠️ HIGH PRIORITY ISSUE #5: Input Validation & Error Handling

My FWSR application currently lacks proper input validation and error handling. I need to add comprehensive validation across all forms and API calls.

Requirements:

1. **Install and configure Zod** for TypeScript-first schema validation

2. **Add validation to these components:**
   - AuthModal (email, password validation)
   - PaymentModal (before Stripe checkout)
   - AuditorChat (message length, content filtering)
   - RiskCalculator (ensure all questions answered per section)

3. **Create error boundaries:**
   - Top-level error boundary in App.tsx
   - Catch rendering errors gracefully
   - Show user-friendly error messages

4. **Add try-catch blocks to:**
   - All Supabase auth calls
   - All Stripe API interactions
   - All Gemini AI calls
   - Database queries

5. **Error UI components:**
   - Toast notifications for temporary errors
   - Error pages for fatal errors
   - Inline form validation messages

6. **Validation schemas needed:**
```typescript
// Email validation
const emailSchema = z.string().email();
```

```
// Password requirements
const passwordSchema = z.string().min(8).regex(/[A-Z]/).regex(/[0-9]/);

// Payment validation
const paymentSchema = z.object({
  planId: z.enum(['survey', 'chat', 'auditor']),
  // ... etc
});
```

Please provide:
- Complete Zod schemas for all forms
- ErrorBoundary component
- Toast notification system
- Updated components with validation
- Utility functions for error handling

⚠️ HIGH PRIORITY ISSUE #6: Database Schema & Data Persistence

I'm using Supabase but haven't defined a proper database schema. I need a complete schema to store all application data.

Required tables:

1. **user_profiles** - Extended user data beyond auth
2. **assessments** - Store compliance assessment results
3. **assessment_responses** - Individual question responses
4. **purchases** - Track plan purchases and entitlements
5. **chat_history** - Store auditor chat conversations
6. **documents** - Store uploaded compliance documents

Requirements:

1. Create complete SQL schema with:
   - Proper foreign key relationships
   - Row Level Security (RLS) policies
   - Indexes for performance
   - Timestamps (created_at, updated_at)

2. Schema should support:
   - Users can have multiple assessments (saved drafts + completed)
   - Each assessment has 19 pillar scores
   - Users can purchase multiple plans
   - Chat history is tied to user and assessment
   - Document uploads per pillar

3. Provide:
   - SQL migration files
   - TypeScript types matching database schema
   - Supabase client helper functions (CRUD operations)
   - Example queries for common operations

4. **Data persistence for RiskCalculator:**
   - Auto-save progress every 30 seconds
   - Allow users to resume incomplete assessments
   - Store timestamp of last update

Please provide complete SQL schema, RLS policies, and TypeScript integration code.

💡 MEDIUM PRIORITY ISSUE #7: Payment Verification & Access Control

I need to implement proper payment verification and access control for the three-tier system:

Tiers:
- Survey ($19) - Access to RiskCalculator only
- Chat ($89) - Access to RiskCalculator + Knowledge Hub
- Auditor ($595) - Full access: RiskCalculator + Chat + AI Auditor

Current issues:
- No verification that user actually paid before accessing features
- Routes aren't protected server-side
- Could manually navigate to /chat without paying

Requirements:

1. **Stripe webhook handler** to verify successful payments:
   - Listen for `checkout.session.completed` event
   - Write purchase record to Supabase
   - Send confirmation email

2. **Access control middleware:**
   - Check user's active plan before rendering protected routes
   - Show upgrade prompt if insufficient tier
   - Prevent API calls from unauthorized users

3. **Protected routes:**
   - /calculating - Requires 'survey', 'chat', or 'auditor' plan
   - /chat - Requires 'chat' or 'auditor' plan
   - /result - Requires completed paid assessment

4. **Supabase Edge Function** for webhook:
   - Verify Stripe signature
   - Extract customer and plan info
   - Update user_purchases table
   - Handle edge cases (duplicate payments, refunds)

5. **Frontend access control:**
   - `useUserAccess()` hook to check entitlements
   - `<ProtectedRoute>` component wrapper
   - Graceful upgrade prompts

Please provide:
- Complete webhook handler code
- Access control utilities
- Protected route components
- User entitlement checking logic

NICE-TO-HAVE ISSUE #8: Analytics & Monitoring

Add analytics and monitoring to track user behavior and conversion metrics.

Requirements:

1. **Install Posthog** for product analytics

2. **Track these events:**
   - Page views (landing, how-it-works, etc.)
   - Button clicks ("Verify Status", "Start Assessment")
   - Funnel stages:
     - Viewed pricing
     - Started checkout
     - Completed payment
     - Started assessment
     - Completed assessment
   - Feature usage (chat messages sent, documents uploaded)
   - Errors (auth failures, payment failures, API errors)

3. **User properties to track:**
   - Plan tier (survey/chat/auditor)
   - Assessment completion status
   - Days since signup
   - Number of assessments completed

4. **Dashboard KPIs:**
   - Conversion rate (visitor → paid user)
   - Average revenue per user (ARPU)
   - Assessment completion rate
   - Feature adoption by tier

Please provide:
- Posthog integration setup
- Event tracking utilities
- Example dashboard queries
- Privacy-compliant tracking (GDPR)

🎨 POLISH ISSUE #9: Loading States & UX Improvements

Add professional loading states and improve overall user experience.

Requirements:

1. **Loading states for:**
   - Initial app load (while checking auth)
   - Payment processing
   - Assessment submission
   - AI chat responses (typing indicator)
   - Data fetching (assessments list)

2. **Skeleton screens** for:
   - RiskCalculator questions (while loading next section)
   - Chat history
   - Assessment results

3. **Progress indicators:**
   - Assessment progress (X/19 pillars complete)
   - Upload progress for documents
   - Chat thinking indicator

4. **Micro-interactions:**
   - Success animations after completing assessment
   - Confetti on payment success
   - Smooth transitions between quiz sections
   - Tooltip animations

5. **Empty states:**
   - No assessments yet
   - No chat history

- No results to show

Please provide:
- Loading component library
- Skeleton screen components
- Animation utilities (Framer Motion or CSS)
- Updated components with loading states

🚀 DEPLOYMENT CHECKLIST PROMPT

I'm ready to deploy my FWSR platform. Please create a comprehensive pre-launch checklist and deployment guide.

Provide:

1. **Pre-launch checklist** covering:
   - All legal documents in place (ToS, Privacy, Disclaimers)
   - Security audit (API keys, authentication, RLS)
   - Payment testing (test mode → live mode)
   - Mobile responsiveness testing
   - Browser compatibility
   - SEO optimization (meta tags, sitemap, robots.txt)
   - Analytics installed and verified
   - Error monitoring setup
   - Backup procedures

2. **Deployment steps** for:
   - Vercel/Netlify deployment
   - Supabase production setup
   - Environment variables configuration
   - Custom domain setup
   - SSL certificate
   - CDN configuration

3. **Post-launch monitoring:**
   - Health check endpoints
   - Error tracking (Sentry setup)
   - Performance monitoring
   - User feedback collection

4. **Launch sequence:**
   - Soft launch with beta users
   - Feedback iteration
   - Public announcement plan
   - Marketing channels

5. **Rollback plan** if issues occur

Please provide step-by-step deployment guide with commands and configuration files.

📝 USAGE INSTRUCTIONS

How to use these prompts with Gemini Pro 2.0:

1. Copy one prompt at a time
2. Paste into Google AI Studio
3. Review the generated code carefully
4. Test in your development environment
5. Iterate if needed with follow-up questions

Recommended order:

1. Issue #1 (Rebrand) - Do this TODAY
2. Issue #2 (Legal docs) - Before collecting any payments
3. Issue #3 (RiskCalculator) - Your core product
4. Issue #4 (API security) - Before using AI features
5. Issue #5 (Validation) - Before any user testing
6. Issue #6 (Database) - Foundation for everything
7. Issue #7 (Payment verification) - Before launch
8. Issues #8-9 (Polish) - Nice-to-haves
9. Deployment checklist - Final step

Pro tips:

- Ask Gemini for clarification if any code is unclear
- Request tests/documentation for complex features
- Iterate on UI/UX until it feels right
- Test each component thoroughly before moving to next