Scribal MVP - Detailed Product Description

Product Overview

Scribal is an AI writing assistant that learns your unique writing voice to create authentic essays that sound like YOU wrote them. Upload 2-3 of your previous essays, and Scribal creates a personalized writing profile that captures your vocabulary, style, and voice. Then generate new essays that maintain your authentic writing style while helping you complete assignments faster.

Target Customers

Primary: College Students (Ages 18-22)

- **Busy STEM students** who struggle with required writing courses
- **International students** who want to improve their English writing while maintaining authenticity
- **Overwhelmed students** juggling multiple assignments and part-time jobs
- **Average writers** who have good ideas but struggle to express them clearly

Market Pain Points:

- Students spend 8+ hours per week on writing assignments
- 67% of students report anxiety about writing quality
- Generic AI tools produce content that doesn't match their voice
- Fear of academic integrity violations with existing AI tools

Database Schema

Core Tables

```
sql
-- Users table for authentication and profile
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    university VARCHAR(200),
    major VARCHAR(200),
    profile_picture_url TEXT,
    email_verified BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Writing styles table - stores user's unique writing DNA
CREATE TABLE writing_styles (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    style_name VARCHAR(100) NOT NULL DEFAULT 'My Writing Style',
    vocabulary_level INTEGER CHECK (vocabulary_level >= 1 AND vocabulary_level <= 10),
    avg_sentence_length DECIMAL(5,2),
    complexity_score DECIMAL(3,2),
    tone_analysis JSONB, -- stores personality traits, formality level, etc.
    writing_patterns JSONB, -- stores sentence structures, transitions, etc.
    sample_phrases TEXT[], -- array of commonly used phrases
    authenticity_baseline DECIMAL(3,2) DEFAULT 0.00,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Sample essays uploaded by users for style analysis
CREATE TABLE sample_essays (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    writing_style_id UUID REFERENCES writing_styles(id) ON DELETE CASCADE,
    title VARCHAR(300) NOT NULL,
    content TEXT NOT NULL,
    word_count INTEGER NOT NULL,
    file_name VARCHAR(255),
    file_size INTEGER,
    upload_date TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    analysis_status VARCHAR(50) DEFAULT 'pending', -- pending, analyzing, completed, err
    analysis_results JSONB, -- detailed style analysis data
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Generated essays and their metadata
CREATE TABLE generated_essays (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    writing_style_id UUID REFERENCES writing_styles(id) ON DELETE CASCADE,
    title VARCHAR(300) NOT NULL,
    prompt TEXT NOT NULL,
    requirements TEXT, -- assignment requirements, word count, etc.
    generated_content TEXT NOT NULL,
    word_count INTEGER NOT NULL,
    authenticity_score DECIMAL(3,2) DEFAULT 0.00,
    generation_time_ms INTEGER, -- time taken to generate
    status VARCHAR(50) DEFAULT 'completed', -- generating, completed, error
    is_favorite BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Essay sections for granular editing and authenticity tracking
CREATE TABLE essay_sections (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    generated_essay_id UUID REFERENCES generated_essays(id) ON DELETE CASCADE,
    section_type VARCHAR(50) NOT NULL, -- introduction, body_paragraph, conclusion
    section_order INTEGER NOT NULL,
    content TEXT NOT NULL,
    authenticity_score DECIMAL(3,2) DEFAULT 0.00,
    user_edited BOOLEAN DEFAULT FALSE,
    edit_count INTEGER DEFAULT 0,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
    updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- User sessions for tracking active usage
CREATE TABLE user_sessions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    session_token VARCHAR(500) UNIQUE NOT NULL,
    expires_at TIMESTAMP WITH TIME ZONE NOT NULL,
    ip_address INET,
    user_agent TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Usage tracking for analytics (simplified for MVP)
CREATE TABLE usage_events (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    event_type VARCHAR(100) NOT NULL, -- essay_generated, style_analyzed, etc.
    event_data JSONB,
    ip_address INET,
    user_agent TEXT,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

Indexes for Performance

```
sql
-- Essential indexes for MVP
CREATE INDEX idx_users_email ON users(email);
CREATE INDEX idx_writing_styles_user_id ON writing_styles(user_id);
CREATE INDEX idx_sample_essays_user_id ON sample_essays(user_id);
CREATE INDEX idx_sample_essays_writing_style_id ON sample_essays(writing_style_id);
CREATE INDEX idx_generated_essays_user_id ON generated_essays(user_id);
CREATE INDEX idx_generated_essays_created_at ON generated_essays(created_at DESC);
CREATE INDEX idx_essay_sections_generated_essay_id ON essay_sections(generated_essay_id)
CREATE INDEX idx_user_sessions_user_id ON user_sessions(user_id);
CREATE INDEX idx_user_sessions_token ON user_sessions(session_token);
CREATE INDEX idx_usage_events_user_id ON usage_events(user_id);
CREATE INDEX idx_usage_events_created_at ON usage_events(created_at DESC);
```

Next.js Routes Structure

File Structure

```
src/
├── app/
│   ├── (auth)/
│   │   ├── login/
│   │   │   └── page.tsx
│   │   ├── register/
│   │   │   └── page.tsx
│   │   ├── verify-email/
│   │   │   └── page.tsx
│   │   └── reset-password/
│   │       └── page.tsx
│   ├── (dashboard)/
│   │   ├── dashboard/
│   │   │   └── page.tsx
│   │   ├── profile/
│   │   │   └── page.tsx
│   │   ├── style-analysis/
│   │   │   ├── page.tsx
│   │   │   └── [styleId]/
│   │   │       └── page.tsx
│   │   ├── essay-generator/
│   │   │   ├── page.tsx
│   │   │   └── [essayId]/
│   │   │       ├── page.tsx
│   │   │       └── edit/
│   │   │           └── page.tsx
│   │   └── essays/
│   │       ├── page.tsx
│   │       └── [essayId]/
│   │           └── page.tsx
│   ├── api/
│   │   ├── auth/
│   │   │   ├── login/
│   │   │   │   └── route.ts
│   │   │   ├── register/
│   │   │   │   └── route.ts
│   │   │   ├── verify-email/
│   │   │   │   └── route.ts
│   │   │   ├── reset-password/
│   │   │   │   └── route.ts
│   │   │   └── session/
│   │   │       └── route.ts
│   │   ├── user/
│   │   │   ├── profile/
│   │   │   │   └── route.ts
│   │   │   └── usage/
│   │   │       └── route.ts
│   │   ├── writing-style/
│   │   │   ├── route.ts
│   │   │   ├── [styleId]/
│   │   │   │   └── route.ts
│   │   │   └── analyze/
│   │   │       └── route.ts
│   │   ├── essays/
│   │   │   ├── samples/
│   │   │   │   ├── route.ts
│   │   │   │   ├── upload/
│   │   │   │   │   └── route.ts
│   │   │   │   └── [essayId]/
│   │   │   │       └── route.ts
│   │   │   └── generated/
│   │   │       ├── route.ts
│   │   │       ├── generate/
│   │   │       │   └── route.ts
│   │   │       └── [essayId]/
│   │   │           ├── route.ts
│   │   │           └── sections/
│   │   │               └── route.ts
│   │   └── analytics/
│   │       └── track/
│   │           └── route.ts
│   ├── globals.css
│   ├── layout.tsx
│   └── page.tsx
├── components/
│   ├── ui/
│   │   ├── button.tsx
│   │   ├── input.tsx
│   │   ├── card.tsx
│   │   ├── modal.tsx
│   │   ├── progress.tsx
│   │   ├── badge.tsx
│   │   └── textarea.tsx
│   ├── auth/
│   │   ├── login-form.tsx
│   │   ├── register-form.tsx
│   │   └── auth-guard.tsx
│   ├── dashboard/
│   │   ├── dashboard-header.tsx
│   │   ├── sidebar.tsx
│   │   ├── stats-cards.tsx
│   │   └── recent-essays.tsx
│   ├── essay/
│   │   ├── essay-upload.tsx
│   │   ├── essay-generator-form.tsx
│   │   ├── essay-editor.tsx
│   │   ├── authenticity-meter.tsx
│   │   ├── essay-preview.tsx
│   │   └── essay-export.tsx
│   ├── style/
│   │   ├── style-analyzer.tsx
│   │   ├── style-profile.tsx
│   │   ├── style-report.tsx
│   │   └── style-comparison.tsx
│   └── common/
│       ├── loading-spinner.tsx
│       ├── error-boundary.tsx
│       ├── file-upload.tsx
│       └── progress-bar.tsx
├── lib/
│   ├── auth.ts
│   ├── database.ts
│   ├── openai.ts
│   ├── style-analyzer.ts
│   ├── essay-generator.ts
│   ├── file-processor.ts
│   ├── validations.ts
│   └── utils.ts
├── hooks/
│   ├── use-auth.ts
│   ├── use-style-analysis.ts
│   ├── use-essay-generation.ts
│   └── use-file-upload.ts
├── types/
│   ├── database.ts
│   ├── auth.ts
│   ├── essay.ts
│   └── style.ts
└── middleware.ts
```

API Routes Detailed

Authentication Routes

- `POST /api/auth/register` - User registration

- `POST /api/auth/login` - User login
- `POST /api/auth/verify-email` - Email verification
- `POST /api/auth/reset-password` - Password reset
- `GET /api/auth/session` - Get current session
- `DELETE /api/auth/session` - Logout

User Management Routes

- `GET /api/user/profile` - Get user profile
- `PUT /api/user/profile` - Update user profile
- `GET /api/user/usage` - Get usage statistics

Writing Style Routes

- `GET /api/writing-style` - Get user's writing styles
- `POST /api/writing-style` - Create new writing style
- `GET /api/writing-style/[styleId]` - Get specific writing style
- `PUT /api/writing-style/[styleId]` - Update writing style
- `DELETE /api/writing-style/[styleId]` - Delete writing style
- `POST /api/writing-style/analyze` - Analyze uploaded essays for style

Essay Management Routes

- `GET /api/essays/samples` - Get user's sample essays
- `POST /api/essays/samples/upload` - Upload sample essay
- `GET /api/essays/samples/[essayId]` - Get specific sample essay
- `DELETE /api/essays/samples/[essayId]` - Delete sample essay
- `GET /api/essays/generated` - Get user's generated essays
- `POST /api/essays/generated/generate` - Generate new essay
- `GET /api/essays/generated/[essayId]` - Get specific generated essay
- `PUT /api/essays/generated/[essayId]` - Update generated essay
- `DELETE /api/essays/generated/[essayId]` - Delete generated essay
- `GET /api/essays/generated/[essayId]/sections` - Get essay sections
- `PUT /api/essays/generated/[essayId]/sections` - Update essay sections

Analytics Routes

- `POST /api/analytics/track` - Track user events

Core Features (MVP)

1. Personal Writing Style Analyzer

What it does: Analyzes your previous essays to create a unique "writing DNA" profile

How it works: - Upload 2-3 previous essays (PDF, DOCX, or text) - AI analyzes vocabulary patterns, sentence structure, and writing quirks - Creates a personalized style profile in 60 seconds - Shows you a "Style Report" with your writing characteristics

Technical Implementation: - File upload with drag-and-drop interface - PDF/DOCX text extraction using libraries - OpenAI API for style analysis with custom prompts - Vector embeddings for style pattern matching - Real-time progress updates via Server-Sent Events

Database Flow: 1. User uploads essay → `sample_essays` table 2. Background job analyzes style → updates `writing_styles` table 3. Analysis results stored in `analysis_results` JSONB field 4. User can view style report from processed data

Value: Ensures all generated content matches YOUR authentic voice, not generic AI writing

2. Style-Matched Essay Generator ✍

What it does: Generates complete essays that sound like you wrote them

How it works: - Input your assignment prompt and basic requirements - AI generates essays using your personal writing style - Content matches your vocabulary level and sentence patterns - Maintains your authentic voice throughout

Technical Implementation: - Form-based prompt input with requirement fields - OpenAI API with custom system prompts using style data - Section-by-section generation (intro, body paragraphs, conclusion) - Real-time generation progress with streaming responses - Authenticity scoring using style comparison algorithms

Database Flow: 1. User submits prompt → creates `generated_essays` record 2. Essay generated in sections → `essay_sections` table 3. Authenticity score calculated and stored 4. User can edit sections with tracking

Value: Complete essays in 10 minutes instead of 5+ hours, with authentic personal style

3. Authenticity Score Dashboard

What it does: Measures how well generated content matches your natural writing style

How it works: - Real-time scoring of generated content (0-100% authenticity) - Highlights sections that don't match your style - Suggests improvements to make content more "you" - Tracks your writing consistency over time

Technical Implementation: - Style comparison algorithms using vector similarity - Section-level authenticity scoring - Visual feedback with color-coded indicators - Suggestion engine for style improvements - Historical tracking and analytics

Database Flow: 1. Generated content analyzed against style profile 2. Scores stored in `authenticity_score` fields 3. Usage events tracked for analytics 4. User editing tracked for improvement

Value: Confidence that your essays sound authentic and won't raise red flags

Technical Implementation

Architecture

- **Frontend:** Next.js 14 with TypeScript, Tailwind CSS, React Hook Form
- **Backend:** Supabase (PostgreSQL + Auth + Storage)
- **AI:** OpenAI GPT-4 API for style analysis and generation
- **File Processing:** pdf-parse, mammoth for document parsing
- **Hosting:** Vercel with edge functions
- **Analytics:** PostHog for user behavior tracking

Key Technical Features

- **File Upload:** Drag-and-drop with progress tracking
- **Real-time Generation:** Server-sent events for live updates
- **Style Matching:** Vector embeddings with cosine similarity
- **Secure Storage:** Row-level security with encrypted data
- **Responsive Design:** Mobile-first approach with PWA capabilities

Security & Privacy

- **Data Encryption:** All essay content encrypted at rest
- **Row-Level Security:** Users can only access their own data
- **Session Management:** Secure JWT tokens with refresh rotation

- **File Scanning:** Virus scanning for uploaded documents
- **Rate Limiting:** API rate limiting to prevent abuse

User Journey

First-Time User (Complete Flow)

1. **Landing Page** → Register with university email
2. **Email Verification** → Click verification link
3. **Onboarding** → Complete profile setup
4. **Upload Essays** → Drag-and-drop 2-3 sample essays
5. **Style Analysis** → Wait 60 seconds for processing
6. **Style Report** → Review writing characteristics
7. **Generate First Essay** → Use sample prompt
8. **Review & Edit** → Check authenticity score
9. **Export Essay** → Download in preferred format

Returning User (Streamlined Flow)

1. **Dashboard** → View recent essays and stats
2. **New Essay** → Click "Generate New Essay"
3. **Input Prompt** → Paste assignment requirements
4. **Generate** → AI creates essay in user's style
5. **Review** → Check authenticity and make edits
6. **Export** → Download final essay

Mobile Experience

- **Responsive Design** → Works on all devices
- **Touch-Friendly** → Large buttons and easy navigation
- **Offline Capability** → PWA with offline essay editing
- **Quick Actions** → Swipe gestures for common tasks

User Interface Design

Design System

- **Color Palette:** Modern blues and purples for trust and creativity
- **Typography:** Clean, readable fonts (Inter/Roboto)
- **Components:** Consistent UI library with shadcn/ui
- **Animations:** Smooth transitions and micro-interactions
- **Accessibility:** WCAG 2.1 AA compliance

Key Screens

1. **Dashboard:** Overview of essays, style score, recent activity
2. **Style Analyzer:** Upload interface with progress tracking
3. **Style Profile:** Visual representation of writing characteristics
4. **Essay Generator:** Clean form with real-time generation
5. **Essay Editor:** Rich text editor with authenticity feedback
6. **Essay Library:** Grid view of all generated essays

Performance Optimization

Frontend Performance

- **Code Splitting:** Route-based code splitting
- **Image Optimization:** Next.js Image component
- **Caching:** SWR for data fetching and caching
- **Lazy Loading:** Components loaded on demand
- **Bundle Size:** Tree shaking and dead code elimination

Backend Performance

- **Database Optimization:** Proper indexing and query optimization
- **Caching:** Redis cache for frequently accessed data
- **CDN:** Static assets served via CDN
- **API Optimization:** Efficient pagination and filtering
- **Background Jobs:** Async processing for heavy tasks

Success Metrics

Product KPIs

- **Monthly Active Users:** 1,000 by month 6
- **Essay Generation Rate:** 3 essays per user per month
- **Style Authenticity Score:** 90%+ average
- **User Satisfaction:** 4.5+ star rating
- **Time to First Essay:** Under 5 minutes

Technical KPIs

- **Page Load Time:** Under 2 seconds
- **API Response Time:** Under 500ms
- **Uptime:** 99.9% availability

- **Error Rate:** Under 1%
- **Generation Success Rate:** 95%+

Business KPIs

- **Monthly Recurring Revenue:** $15,000 by month 12
- **Customer Acquisition Cost:** Under $30
- **Monthly Churn Rate:** Under 10%
- **Conversion Rate:** 25% free to paid
- **Customer Lifetime Value:** $180

Competitive Advantage

Why Scribal Wins

1. **Personal Style Focus:** Only tool that truly learns YOUR voice
2. **Authenticity First:** Built for academic integrity, not cheating
3. **Student-Centric:** Designed specifically for student needs and budgets
4. **Simple & Fast:** Get authentic essays in minutes, not hours
5. **Privacy-Focused:** Your essays stay private and secure

What Makes It Different

- **Not a generic AI tool** - it's YOUR writing assistant
- **Focuses on authenticity** over raw content generation
- **Learns and improves** with each essay you write
- **Transparent about AI assistance** - maintains academic integrity
- **Built for students** - affordable and education-focused

Implementation Timeline

Phase 1: Foundation (Weeks 1-4)

- Database setup and migrations
- User authentication system
- Basic UI components and layouts
- File upload functionality
- OpenAI API integration

Phase 2: Core Features (Weeks 5-8)

- Style analysis engine

- Essay generation system
- Authenticity scoring
- Basic dashboard
- User profile management

Phase 3: Polish & Testing (Weeks 9-12)

- UI/UX improvements
- Performance optimization
- Security hardening
- Beta testing with students
- Bug fixes and refinements

Phase 4: Launch Preparation (Weeks 13-16)

- Final testing and QA
- Documentation and help guides
- Marketing website
- Analytics setup
- Production deployment

Risk Mitigation

Academic Integrity

- **Transparency:** Clear labeling of AI assistance
- **Educational Focus:** Positioned as learning tool, not cheating
- **Usage Guidelines:** Clear policies on appropriate use
- **Plagiarism Detection:** Built-in originality checking

Technical Risks

- **API Rate Limits:** Implement queuing and retry logic
- **Data Privacy:** Full encryption and secure storage
- **Cost Control:** Usage tracking and budget alerts
- **Scalability:** Auto-scaling infrastructure
- **Downtime:** Multiple deployment environments

Business Risks

- **Market Competition:** Focus on unique value proposition
- **User Acquisition:** Multiple marketing channels

- **Feature Creep:** Strict MVP scope management
- **Technical Debt:** Regular code reviews and refactoring

Development Resources

Team Structure

- **Frontend Developer:** React/Next.js specialist
- **Backend Developer:** Node.js/PostgreSQL expert
- **AI/ML Engineer:** OpenAI API and NLP specialist
- **UI/UX Designer:** Student-focused design experience
- **Product Manager:** Educational technology background

Technology Stack

- **Frontend:** Next.js 14, TypeScript, Tailwind CSS
- **Backend:** Supabase, PostgreSQL, Claude API
- **Hosting:** Vercel, Supabase for file storage
- **Monitoring:** PostHog, Vercel Analytics

Development Environment

- **Version Control:** Git with feature branching
- **Code Quality:** ESLint, Prettier, TypeScript

This detailed MVP provides a comprehensive foundation for building Scribal, with clear technical specifications, user flows, and implementation guidelines that can be followed by a development team to create a successful AI-powered writing assistant for students.