**PRD: Smart Note Organizer**

**Executive Summary**

Smart Note Organizer is a comprehensive web application designed for students and researchers to efficiently organize, search, and summarize academic notes. It leverages AI technologies to transform unstructured note content into a searchable, connected knowledge base with automatic tagging, summarization, and flashcard generation capabilities.

**Product Vision**

To create the most intuitive and powerful academic note management system that transforms scattered study materials into an organized, interconnected knowledge base that enhances learning efficiency.

**User Personas**

**Primary: College Student (Alex)**

* **Background**: Undergraduate biology major
* **Pain Points**: Overwhelmed by volume of notes, struggles to find specific information before exams
* **Goals**: Organize lecture notes efficiently, quickly review key concepts, prepare effectively for exams

**Secondary: Academic Researcher (Dr. Maya)**

* **Background**: Physics researcher working on multiple projects
* **Pain Points**: Needs to cross-reference information across many papers, loses track of connections between concepts
* **Goals**: Create a searchable database of research notes, identify connections between different studies

**Problem Statement**

Students and researchers struggle with:

1. Inefficiently managing large volumes of notes across different subjects/projects
2. Finding specific information when needed
3. Connecting related concepts across different sets of notes
4. Creating effective study materials from extensive notes
5. Converting handwritten or scanned notes into searchable digital formats

**Solution**

**MVP Features**

**1. Rich-Text & PDF Import**

* **Capability**: Import text files and PDFs (including scanned documents)
* **Technology**: Gemini 2.0 for OCR processing of handwritten/scanned notes
* **User Flow**:
  + User uploads files via drag-and-drop or file selector
  + System processes text and integrates into the database
  + OCR automatically converts images/scans to searchable text
  + User receives confirmation when processing is complete

**2. Auto-Tagging & Linking**

* **Capability**: Automatic identification of key topics and relationship mapping
* **Technology**: LLM Gemini 2.0 Api for content analysis
* **User Flow**:
  + System analyzes note content upon import
  + Key concepts are extracted and suggested as tags
  + System identifies connections between notes with similar topics
  + User can accept suggested tags/links or modify them

**3. AI-Powered Summaries**

* **Capability**: Generate concise 2-3 sentence summaries of notes
* **Technology**: LLM APIs (Grok, OpenAI)
* **User Flow**:
  + User selects notes to summarize
  + System generates and displays summary
  + User can edit, save, or regenerate summaries

**4. Global Search**

* **Capability**: Unified search across all notes, tags, and summaries
* **Technology**: Full-text search implementation with Supabase
* **User Flow**:
  + User enters search terms in a central search bar
  + System displays results categorized by notes, tags, and summaries
  + User can filter results by date, course, or other metadata

**5. Flashcard Export**

* **Capability**: Convert highlighted content into study flashcards
* **Technology**: Custom JSON export compatible with Anki
* **User Flow**:
  + User highlights key information in notes
  + System generates question-answer pairs based on content
  + User exports flashcards in Anki-compatible format

**UI/UX Design Principles**

* **Glassmorphic Interface**: Semi-transparent elements with subtle blur effects
* **Color Scheme**: Adaptive palette that works in both light and dark modes
* **Navigation**: Floating bottom navbar with curved edges instead of traditional top navigation
* **Layout**: Clean, minimalist design with ample white space
* **Accessibility**: High-contrast text, keyboard shortcuts, screen reader compatibility

**Technical Requirements**

**Frontend**

* **Framework**: React.js with functional components and hooks
* **State Management**: React Context API or Redux
* **Styling**: Tailwind CSS with custom glassmorphism utilities
* **Responsive Design**: Mobile-first approach with adaptive layouts

**Backend**

* **Database & Authentication**: Supabase
* **Storage**: Supabase Storage for PDFs and documents
* **API Integration**:
  + Gemini API for AI features and OCR

**Architecture**

* **Client-Server Model**: React SPA frontend with Supabase backend
* **Authentication Flow**: JWT-based auth through Supabase
* **Data Flow**: RESTful communication between frontend and backend

**User Flow**

1. **Onboarding**:
   * User creates account/signs in
   * Brief tutorial on core features
   * Option to import initial set of notes
2. **Note Import & Organization**:
   * User uploads documents
   * System processes, tags, and summarizes content
   * Notes organized into user-defined or auto-suggested categories
3. **Note Exploration & Search**:
   * User navigates through notes via tags or search
   * System displays related notes and suggested connections
   * User can view full notes or AI-generated summaries
4. **Study Material Creation**:
   * User selects content for flashcards
   * System generates Q&A pairs
   * User exports to Anki format

**Implementation Roadmap**

**Phase 1**

* Set up React project structure
* Implement Supabase integration
* Design and implement UI components
* Create basic note management functionality

**Phase 2**

* Implement PDF import with OCR
* Set up LLM API integration
* Build auto-tagging functionality
* Develop global search feature

**Phase 3**

* Implement AI summarization
* Create note linking system
* Develop flashcard export functionality
* Polish UI/UX and implement dark mode

**Phase 4**

* Extensive testing and bug fixing
* Performance optimization
* User feedback integration
* Documentation and launch preparation

**Success Metrics**

* **User Engagement**: Active use time per session (target: >10 minutes)
* **Feature Adoption**: % of users utilizing AI features (target: >60%)
* **Content Management**: Average number of notes imported per user (target: >20)
* **Utility**: User-reported time saved when searching for information (target: reduction by 50%)
* **Retention**: Weekly active users / Monthly active users ratio (target: >0.5)

**Future Enhancements**

* Voice note recording and transcription
* Collaborative note sharing and editing
* Integration with popular LMS platforms (Canvas, Blackboard)
* Custom AI training for domain-specific terminology
* Mobile app versions (iOS/Android)

**UI Mockups**

**Landing Page**

* Glassmorphic hero section with feature showcase
* Quick login/signup buttons
* Sample screenshots of the application

**Main Dashboard**

* Recent notes with AI summaries
* Tag cloud for quick navigation
* Search bar prominently displayed
* Bottom floating navbar with icons for Home, Search, Import, and Profile

**Note Viewer**

* Split view with original note and AI summary
* Side panel showing related notes and tags
* Highlighting tools for flashcard creation
* Floating action button for common actions

**Search Interface**

* Real-time search results
* Categorized results (by tags, content, summaries)
* Visual representation of note connections
* Filter options for refining search

**Settings & Profile**

* Theme toggle (light/dark mode)
* API integration settings
* Export/import preferences
* User profile and storage usage

**Technical Considerations**

**Security**

* End-to-end encryption for sensitive academic content
* Secure API key management for LLM services
* Regular security audits and compliance checks

**Performance**

* Lazy loading for large PDF documents
* Caching strategy for frequently accessed notes
* Background processing for OCR and AI tasks

**Accessibility**

* WCAG 2.1 AA compliance
* Support for screen readers
* Keyboard navigation throughout the application

**Conclusion**

Smart Note Organizer addresses a critical need for efficient academic note management by leveraging modern AI and OCR technologies. The glassmorphic UI design with a floating navigation bar creates a distinctive, user-friendly experience that stands apart from generic note-taking applications. By automatically organizing, tagging, and summarizing content, this tool will significantly enhance study efficiency and knowledge retention for students and researchers alike.

**Smart Note Organizer: Page Structure and User Flow**

**Core Pages and Flow Structure**

**1. Authentication Pages**

* **Login Page**
  + Email/password login form
  + Social login options
  + "Forgot password" link
  + Link to signup page
  + Glassmorphic background with subtle animations
* **Signup Page**
  + Registration form
  + Email verification step
  + Brief explanation of benefits
  + Link to login page
* **Password Reset Page**
  + Email input
  + Reset instructions
  + Confirmation screen

**2. Onboarding Flow**

* **Welcome Screen**
  + Personalized greeting
  + Overview of key features
  + Progress indicator
* **Feature Tour**
  + Interactive walkthrough of core functionalities
  + Skip option
  + "Next" and "Previous" navigation
* **Initial Import Screen**
  + Option to import first set of notes
  + Sample notes for demonstration
  + Skip for now option

**3. Main Application Pages**

* **Dashboard (Home)**
  + Recent notes grid/list
  + AI-generated summaries preview
  + Popular tags visualization
  + Quick stats (total notes, recently created)
  + Quick actions (new note, import, search)
  + Activity feed showing recent changes
* **Notes Explorer**
  + Filterable notes gallery
  + Sort options (date, title, course)
  + View toggle (list/grid/board)
  + Batch operations (tag, delete, export)
  + Note previews with AI-generated summaries
* **Single Note View**
  + Original note content
  + AI summary panel
  + Related notes sidebar
  + Tag management
  + Highlighting tools
  + Action buttons (edit, share, export to flashcards)
  + Version history access
* **Note Editor**
  + Rich text editing interface
  + Formatting toolbar
  + Auto-save functionality
  + Split view option (edit/preview)
  + Tag suggestions as you type
* **Import Center**
  + Drag-and-drop upload area
  + File browser button
  + Import history
  + Processing status indicators
  + Batch import options
  + OCR settings for scanned documents
* **Search Interface**
  + Global search bar
  + Real-time results
  + Advanced filters panel
  + Categorized results tabs (Notes, Tags, Summaries)
  + Search history
  + Saved searches
* **Flashcard Workshop**
  + Notes selection panel
  + Generated Q&A pairs
  + Edit/customize flashcards
  + Preview flashcards
  + Export options (Anki format, PDF)
  + Study mode preview
* **Tag Manager**
  + All tags list/cloud
  + Tag relationships visualization
  + Merge/split/rename tag options
  + Tag color customization
  + Notes count per tag
* **Settings**
  + Account settings
  + Appearance preferences (light/dark mode)
  + API integrations configuration
  + Export/backup options
  + Notification preferences
  + Advanced settings (OCR quality, AI models)
* **Profile Page**
  + User information
  + Usage statistics
  + Subscription details
  + Storage quota
  + Connected services

**4. Utility Pages**

* **Help Center**
  + Searchable FAQ
  + Video tutorials
  + Contextual help guides
  + Contact support option
* **404/Error Pages**
  + User-friendly error messages
  + Suggestions to return to working pages
  + Report issue option

**User Flow Diagrams**

**1. New User Flow**

Login/Signup → Welcome Screen → Feature Tour → Initial Import → Dashboard

**2. Note Creation Flow**

Dashboard → Note Editor → Save → AI Processing (tagging/summarizing) → Single Note View

**3. Import Flow**

Dashboard/Import Center → Upload Files → Processing Status → Notes Explorer (filtered to show new imports)

**4. Search and Discovery Flow**

Any Page → Search Bar → Search Results → Filter/Refine → Single Note View

**5. Study Material Creation Flow**

Notes Explorer → Select Notes → Flashcard Workshop → Edit Cards → Export to Anki

**Detailed Page Components**

**Dashboard Components**

1. **Header Section**
   * User profile thumbnail
   * Quick search bar
   * Notifications icon
2. **Quick Stats Widget**
   * Notes count
   * Tags count
   * Recent activity summary
3. **Recent Notes Grid**
   * Thumbnails with title and summary
   * Last modified date
   * Primary tags
4. **Tag Cloud Widget**
   * Interactive tag visualization
   * Size indicates frequency
   * Click to filter notes
5. **Quick Actions Panel**
   * New note button
   * Import button
   * Generate flashcards button

**Note Explorer Components**

1. **Filter Sidebar**
   * Date filters
   * Tag filters
   * Source filters (imported, created)
   * Full-text search field
2. **Note List/Grid View**
   * Sortable columns
   * Preview snippets
   * AI-generated summaries
   * Primary tags
3. **Batch Actions Toolbar**
   * Select all
   * Tag selected
   * Delete selected
   * Export selected

**Single Note View Components**

1. **Content Area**
   * Full note content
   * Inline highlighting tools
   * Image/attachment display
2. **AI Insights Panel**
   * Generated summary
   * Key concepts extracted
   * Suggested related reading
3. **Related Notes Sidebar**
   * Similar content suggestions
   * Notes with shared tags
   * Recently viewed notes
4. **Actions Footer**
   * Edit button
   * Share options
   * Export formats
   * Delete option

**Bottom Floating Navbar**

* **Home Icon** - Returns to dashboard
* **Search Icon** - Opens global search
* **Add Icon** - Creates new note or imports files
* **Tags Icon** - Opens tag manager
* **Profile Icon** - Opens user profile and settings

**Technical Page Structure**

**Component Hierarchy**

1. **App Shell**
   * Authentication wrapper
   * Theme provider
   * Navigation controller
   * Notification system
2. **Page Layout**
   * Floating navbar
   * Content area
   * Modal controller
   * Contextual help
3. **Content Components**
   * Note card
   * Tag chip
   * Summary block
   * Search result item
   * Flashcard component

**Responsive Behavior**

1. **Desktop Layout**
   * Multi-column layout
   * Side panels for related content
   * Expanded previews
   * Keyboard shortcuts visible
2. **Tablet Layout**
   * Two-column layout
   * Collapsible sidebars
   * Touch-optimized controls
   * Simplified previews
3. **Mobile Layout**
   * Single column layout
   * Bottom sheet for additional options
   * Swipe gestures
   * Minimized previews
   * Extended floating navbar functionality

**State Transitions**

**Note States**

1. **Draft** - Currently being edited, not yet processed
2. **Processing** - Undergoing AI analysis (tagging/summarizing)
3. **Complete** - Fully processed and searchable
4. **Archived** - Stored but not in active rotation

**User Session States**

1. **Anonymous** - Not logged in
2. **Authenticated** - Logged in, normal usage
3. **Onboarding** - First-time user experience
4. **Premium** - Authenticated with additional features

**Accessibility Considerations**

* Each page includes proper heading hierarchy
* Form elements have associated labels
* Interactive elements have appropriate focus states
* Color contrasts meet WCAG standards
* Screen reader friendly navigation
* Keyboard navigation paths for all core functions

This comprehensive structure ensures a smooth, intuitive user experience while maintaining the glassmorphic aesthetic across the application's floating navigation system.

**Smart Note Organizer: Supabase Database Schema**

**Complete Database Schema**

**Core Tables**

**1. profiles**

CREATE TABLE profiles (

id UUID REFERENCES auth.users(id) PRIMARY KEY,

email TEXT UNIQUE NOT NULL,

full\_name TEXT,

avatar\_url TEXT,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

preferences JSONB DEFAULT '{"theme": "light", "notification\_enabled": true}'::jsonb,

onboarding\_completed BOOLEAN DEFAULT FALSE,

storage\_used BIGINT DEFAULT 0,

last\_active TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

**2. notes**

CREATE TABLE notes (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

user\_id UUID REFERENCES profiles(id) NOT NULL,

title TEXT NOT NULL,

content TEXT,

original\_content TEXT,

is\_imported BOOLEAN DEFAULT FALSE,

source\_file\_path TEXT,

source\_file\_type TEXT,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

last\_viewed\_at TIMESTAMP WITH TIME ZONE,

ai\_processed BOOLEAN DEFAULT FALSE,

ai\_summary TEXT,

ai\_key\_points JSONB DEFAULT '[]'::jsonb,

status TEXT DEFAULT 'draft',

is\_favorite BOOLEAN DEFAULT FALSE,

is\_archived BOOLEAN DEFAULT FALSE,

metadata JSONB DEFAULT '{}'::jsonb

);

**3. tags**

CREATE TABLE tags (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

user\_id UUID REFERENCES profiles(id) NOT NULL,

name TEXT NOT NULL,

color TEXT DEFAULT '#3B82F6',

description TEXT,

is\_auto\_generated BOOLEAN DEFAULT FALSE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

UNIQUE(user\_id, name)

);

**4. note\_tags**

CREATE TABLE note\_tags (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

note\_id UUID REFERENCES notes(id) ON DELETE CASCADE NOT NULL,

tag\_id UUID REFERENCES tags(id) ON DELETE CASCADE NOT NULL,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

confidence FLOAT DEFAULT 1.0,

is\_auto\_assigned BOOLEAN DEFAULT FALSE,

UNIQUE(note\_id, tag\_id)

);

**5. note\_relations**

CREATE TABLE note\_relations (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

source\_note\_id UUID REFERENCES notes(id) ON DELETE CASCADE NOT NULL,

target\_note\_id UUID REFERENCES notes(id) ON DELETE CASCADE NOT NULL,

relation\_type TEXT DEFAULT 'related',

confidence FLOAT DEFAULT 0.5,

is\_auto\_generated BOOLEAN DEFAULT TRUE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

UNIQUE(source\_note\_id, target\_note\_id),

CHECK (source\_note\_id != target\_note\_id)

);

**6. highlights**

CREATE TABLE highlights (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

note\_id UUID REFERENCES notes(id) ON DELETE CASCADE NOT NULL,

user\_id UUID REFERENCES profiles(id) NOT NULL,

content TEXT NOT NULL,

context TEXT,

position\_start INTEGER,

position\_end INTEGER,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

color TEXT DEFAULT 'yellow',

comment TEXT

);

**7. flashcards**

CREATE TABLE flashcards (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

user\_id UUID REFERENCES profiles(id) NOT NULL,

note\_id UUID REFERENCES notes(id) ON DELETE SET NULL,

highlight\_id UUID REFERENCES highlights(id) ON DELETE SET NULL,

question TEXT NOT NULL,

answer TEXT NOT NULL,

is\_auto\_generated BOOLEAN DEFAULT FALSE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

last\_reviewed TIMESTAMP WITH TIME ZONE,

review\_count INTEGER DEFAULT 0,

difficulty\_rating INTEGER DEFAULT 3,

next\_review\_date TIMESTAMP WITH TIME ZONE

);

**8. flashcard\_decks**

CREATE TABLE flashcard\_decks (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

user\_id UUID REFERENCES profiles(id) NOT NULL,

name TEXT NOT NULL,

description TEXT,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

UNIQUE(user\_id, name)

);

**9. flashcard\_deck\_items**

CREATE TABLE flashcard\_deck\_items (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

deck\_id UUID REFERENCES flashcard\_decks(id) ON DELETE CASCADE NOT NULL,

flashcard\_id UUID REFERENCES flashcards(id) ON DELETE CASCADE NOT NULL,

position INTEGER NOT NULL,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

UNIQUE(deck\_id, flashcard\_id)

);

**10. search\_history**

CREATE TABLE search\_history (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

user\_id UUID REFERENCES profiles(id) NOT NULL,

query TEXT NOT NULL,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

result\_count INTEGER

);

**11. import\_jobs**

CREATE TABLE import\_jobs (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

user\_id UUID REFERENCES profiles(id) NOT NULL,

file\_path TEXT NOT NULL,

file\_name TEXT NOT NULL,

file\_type TEXT NOT NULL,

file\_size BIGINT NOT NULL,

status TEXT DEFAULT 'pending',

error\_message TEXT,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

completed\_at TIMESTAMP WITH TIME ZONE,

metadata JSONB DEFAULT '{}'::jsonb

);

**12. ai\_usage**

CREATE TABLE ai\_usage (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

user\_id UUID REFERENCES profiles(id) NOT NULL,

operation\_type TEXT NOT NULL,

tokens\_used INTEGER NOT NULL,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

note\_id UUID REFERENCES notes(id) ON DELETE SET NULL,

model\_used TEXT,

request\_data JSONB,

response\_data JSONB

);

**Indexes for Performance**

-- Notes by user index

CREATE INDEX idx\_notes\_user\_id ON notes(user\_id);

-- Notes search index

CREATE INDEX idx\_notes\_content\_search ON notes USING gin(to\_tsvector('english', content));

CREATE INDEX idx\_notes\_title\_search ON notes USING gin(to\_tsvector('english', title));

-- Tag related indexes

CREATE INDEX idx\_tags\_user\_id ON tags(user\_id);

CREATE INDEX idx\_note\_tags\_note\_id ON note\_tags(note\_id);

CREATE INDEX idx\_note\_tags\_tag\_id ON note\_tags(tag\_id);

-- Relations index

CREATE INDEX idx\_note\_relations\_source ON note\_relations(source\_note\_id);

CREATE INDEX idx\_note\_relations\_target ON note\_relations(target\_note\_id);

-- Flashcard indexes

CREATE INDEX idx\_flashcards\_user\_id ON flashcards(user\_id);

CREATE INDEX idx\_flashcards\_note\_id ON flashcards(note\_id);

-- Highlight indexes

CREATE INDEX idx\_highlights\_note\_id ON highlights(note\_id);

CREATE INDEX idx\_highlights\_user\_id ON highlights(user\_id);

-- Search history index

CREATE INDEX idx\_search\_history\_user\_id ON search\_history(user\_id);

-- Import jobs index

CREATE INDEX idx\_import\_jobs\_user\_id ON import\_jobs(user\_id);

CREATE INDEX idx\_import\_jobs\_status ON import\_jobs(status);

**Views for Common Queries**

-- User's notes with tags view

CREATE VIEW user\_notes\_with\_tags AS

SELECT

n.id AS note\_id,

n.user\_id,

n.title,

n.ai\_summary,

n.created\_at,

n.updated\_at,

n.is\_favorite,

n.is\_archived,

n.status,

ARRAY\_AGG(DISTINCT t.name) AS tags

FROM notes n

LEFT JOIN note\_tags nt ON n.id = nt.note\_id

LEFT JOIN tags t ON nt.tag\_id = t.id

GROUP BY n.id;

-- Active flashcards for review

CREATE VIEW flashcards\_due\_for\_review AS

SELECT

f.id,

f.user\_id,

f.question,

f.answer,

f.next\_review\_date,

f.difficulty\_rating,

n.title AS note\_title

FROM flashcards f

LEFT JOIN notes n ON f.note\_id = n.id

WHERE f.next\_review\_date <= NOW();

**Functions and Triggers**

-- Function to update timestamps

CREATE OR REPLACE FUNCTION update\_updated\_at()

RETURNS TRIGGER AS $$

BEGIN

NEW.updated\_at = NOW();

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- Trigger to update timestamps

CREATE TRIGGER update\_notes\_timestamp

BEFORE UPDATE ON notes

FOR EACH ROW

EXECUTE PROCEDURE update\_updated\_at();

CREATE TRIGGER update\_tags\_timestamp

BEFORE UPDATE ON tags

FOR EACH ROW

EXECUTE PROCEDURE update\_updated\_at();

CREATE TRIGGER update\_profiles\_timestamp

BEFORE UPDATE ON profiles

FOR EACH ROW

EXECUTE PROCEDURE update\_updated\_at();

-- Function to update user's storage usage

CREATE OR REPLACE FUNCTION update\_storage\_usage()

RETURNS TRIGGER AS $$

BEGIN

-- Calculate the storage difference

IF TG\_OP = 'INSERT' THEN

UPDATE profiles

SET storage\_used = storage\_used + NEW.file\_size

WHERE id = NEW.user\_id;

ELSIF TG\_OP = 'DELETE' THEN

UPDATE profiles

SET storage\_used = storage\_used - OLD.file\_size

WHERE id = OLD.user\_id;

END IF;

RETURN NULL;

END;

$$ LANGUAGE plpgsql;

-- Trigger to track storage usage

CREATE TRIGGER track\_import\_storage

AFTER INSERT OR DELETE ON import\_jobs

FOR EACH ROW

EXECUTE PROCEDURE update\_storage\_usage();

**Row Level Security (RLS) Policies**

**Enable RLS on all tables**

ALTER TABLE profiles ENABLE ROW LEVEL SECURITY;

ALTER TABLE notes ENABLE ROW LEVEL SECURITY;

ALTER TABLE tags ENABLE ROW LEVEL SECURITY;

ALTER TABLE note\_tags ENABLE ROW LEVEL SECURITY;

ALTER TABLE note\_relations ENABLE ROW LEVEL SECURITY;

ALTER TABLE highlights ENABLE ROW LEVEL SECURITY;

ALTER TABLE flashcards ENABLE ROW LEVEL SECURITY;

ALTER TABLE flashcard\_decks ENABLE ROW LEVEL SECURITY;

ALTER TABLE flashcard\_deck\_items ENABLE ROW LEVEL SECURITY;

ALTER TABLE search\_history ENABLE ROW LEVEL SECURITY;

ALTER TABLE import\_jobs ENABLE ROW LEVEL SECURITY;

ALTER TABLE ai\_usage ENABLE ROW LEVEL SECURITY;

**Profiles Table Policies**

-- Users can read their own profile

CREATE POLICY profiles\_select\_own ON profiles

FOR SELECT USING (auth.uid() = id);

-- Users can update their own profile

CREATE POLICY profiles\_update\_own ON profiles

FOR UPDATE USING (auth.uid() = id);

-- Profiles are created automatically by trigger from auth.users

CREATE POLICY profiles\_insert\_own ON profiles

FOR INSERT WITH CHECK (auth.uid() = id);

**Notes Table Policies**

-- Users can read their own notes

CREATE POLICY notes\_select\_own ON notes

FOR SELECT USING (auth.uid() = user\_id);

-- Users can insert their own notes

CREATE POLICY notes\_insert\_own ON notes

FOR INSERT WITH CHECK (auth.uid() = user\_id);

-- Users can update their own notes

CREATE POLICY notes\_update\_own ON notes

FOR UPDATE USING (auth.uid() = user\_id);

-- Users can delete their own notes

CREATE POLICY notes\_delete\_own ON notes

FOR DELETE USING (auth.uid() = user\_id);

**Tags Table Policies**

-- Users can read their own tags

CREATE POLICY tags\_select\_own ON tags

FOR SELECT USING (auth.uid() = user\_id);

-- Users can insert their own tags

CREATE POLICY tags\_insert\_own ON tags

FOR INSERT WITH CHECK (auth.uid() = user\_id);

-- Users can update their own tags

CREATE POLICY tags\_update\_own ON tags

FOR UPDATE USING (auth.uid() = user\_id);

-- Users can delete their own tags

CREATE POLICY tags\_delete\_own ON tags

FOR DELETE USING (auth.uid() = user\_id);

**Note Tags Table Policies**

-- Users can read their own note\_tags

CREATE POLICY note\_tags\_select\_own ON note\_tags

FOR SELECT USING (

EXISTS (

SELECT 1 FROM notes

WHERE notes.id = note\_tags.note\_id

AND notes.user\_id = auth.uid()

)

);

-- Users can insert note\_tags for their own notes

CREATE POLICY note\_tags\_insert\_own ON note\_tags

FOR INSERT WITH CHECK (

EXISTS (

SELECT 1 FROM notes

WHERE notes.id = note\_tags.note\_id

AND notes.user\_id = auth.uid()

)

);

-- Users can delete note\_tags for their own notes

CREATE POLICY note\_tags\_delete\_own ON note\_tags

FOR DELETE USING (

EXISTS (

SELECT 1 FROM notes

WHERE notes.id = note\_tags.note\_id

AND notes.user\_id = auth.uid()

)

);

**Note Relations Table Policies**

-- Users can read relations for their own notes

CREATE POLICY note\_relations\_select\_own ON note\_relations

FOR SELECT USING (

EXISTS (

SELECT 1 FROM notes

WHERE notes.id = note\_relations.source\_note\_id

AND notes.user\_id = auth.uid()

)

);

-- Users can insert relations for their own notes

CREATE POLICY note\_relations\_insert\_own ON note\_relations

FOR INSERT WITH CHECK (

EXISTS (

SELECT 1 FROM notes n1

WHERE n1.id = note\_relations.source\_note\_id

AND n1.user\_id = auth.uid()

) AND EXISTS (

SELECT 1 FROM notes n2

WHERE n2.id = note\_relations.target\_note\_id

AND n2.user\_id = auth.uid()

)

);

-- Users can delete relations for their own notes

CREATE POLICY note\_relations\_delete\_own ON note\_relations

FOR DELETE USING (

EXISTS (

SELECT 1 FROM notes

WHERE notes.id = note\_relations.source\_note\_id

AND notes.user\_id = auth.uid()

)

);

**Highlights Table Policies**

-- Users can read their own highlights

CREATE POLICY highlights\_select\_own ON highlights

FOR SELECT USING (user\_id = auth.uid());

-- Users can insert highlights on their own notes

CREATE POLICY highlights\_insert\_own ON highlights

FOR INSERT WITH CHECK (

user\_id = auth.uid() AND

EXISTS (

SELECT 1 FROM notes

WHERE notes.id = highlights.note\_id

AND notes.user\_id = auth.uid()

)

);

-- Users can update their own highlights

CREATE POLICY highlights\_update\_own ON highlights

FOR UPDATE USING (user\_id = auth.uid());

-- Users can delete their own highlights

CREATE POLICY highlights\_delete\_own ON highlights

FOR DELETE USING (user\_id = auth.uid());

**Flashcards Table Policies**

-- Users can read their own flashcards

CREATE POLICY flashcards\_select\_own ON flashcards

FOR SELECT USING (user\_id = auth.uid());

-- Users can insert their own flashcards

CREATE POLICY flashcards\_insert\_own ON flashcards

FOR INSERT WITH CHECK (user\_id = auth.uid());

-- Users can update their own flashcards

CREATE POLICY flashcards\_update\_own ON flashcards

FOR UPDATE USING (user\_id = auth.uid());

-- Users can delete their own flashcards

CREATE POLICY flashcards\_delete\_own ON flashcards

FOR DELETE USING (user\_id = auth.uid());

**Flashcard Decks Table Policies**

-- Users can read their own flashcard decks

CREATE POLICY flashcard\_decks\_select\_own ON flashcard\_decks

FOR SELECT USING (user\_id = auth.uid());

-- Users can insert their own flashcard decks

CREATE POLICY flashcard\_decks\_insert\_own ON flashcard\_decks

FOR INSERT WITH CHECK (user\_id = auth.uid());

-- Users can update their own flashcard decks

CREATE POLICY flashcard\_decks\_update\_own ON flashcard\_decks

FOR UPDATE USING (user\_id = auth.uid());

-- Users can delete their own flashcard decks

CREATE POLICY flashcard\_decks\_delete\_own ON flashcard\_decks

FOR DELETE USING (user\_id = auth.uid());

**Flashcard Deck Items Table Policies**

-- Users can read their own deck items

CREATE POLICY flashcard\_deck\_items\_select\_own ON flashcard\_deck\_items

FOR SELECT USING (

EXISTS (

SELECT 1 FROM flashcard\_decks

WHERE flashcard\_decks.id = flashcard\_deck\_items.deck\_id

AND flashcard\_decks.user\_id = auth.uid()

)

);

-- Users can insert items into their own decks

CREATE POLICY flashcard\_deck\_items\_insert\_own ON flashcard\_deck\_items

FOR INSERT WITH CHECK (

EXISTS (

SELECT 1 FROM flashcard\_decks

WHERE flashcard\_decks.id = flashcard\_deck\_items.deck\_id

AND flashcard\_decks.user\_id = auth.uid()

)

);

-- Users can delete items from their own decks

CREATE POLICY flashcard\_deck\_items\_delete\_own ON flashcard\_deck\_items

FOR DELETE USING (

EXISTS (

SELECT 1 FROM flashcard\_decks

WHERE flashcard\_decks.id = flashcard\_deck\_items.deck\_id

AND flashcard\_decks.user\_id = auth.uid()

)

);

**Search History Table Policies**

-- Users can read their own search history

CREATE POLICY search\_history\_select\_own ON search\_history

FOR SELECT USING (user\_id = auth.uid());

-- Users can insert their own search history

CREATE POLICY search\_history\_insert\_own ON search\_history

FOR INSERT WITH CHECK (user\_id = auth.uid());

-- Users can delete their own search history

CREATE POLICY search\_history\_delete\_own ON search\_history

FOR DELETE USING (user\_id = auth.uid());

**Import Jobs Table Policies**

-- Users can read their own import jobs

CREATE POLICY import\_jobs\_select\_own ON import\_jobs

FOR SELECT USING (user\_id = auth.uid());

-- Users can insert their own import jobs

CREATE POLICY import\_jobs\_insert\_own ON import\_jobs

FOR INSERT WITH CHECK (user\_id = auth.uid());

-- Users can update their own import jobs

CREATE POLICY import\_jobs\_update\_own ON import\_jobs

FOR UPDATE USING (user\_id = auth.uid());

-- Users can delete their own import jobs

CREATE POLICY import\_jobs\_delete\_own ON import\_jobs

FOR DELETE USING (user\_id = auth.uid());

**AI Usage Table Policies**

-- Users can read their own AI usage

CREATE POLICY ai\_usage\_select\_own ON ai\_usage

FOR SELECT USING (user\_id = auth.uid());

-- Users can insert their own AI usage records

CREATE POLICY ai\_usage\_insert\_own ON ai\_usage

FOR INSERT WITH CHECK (user\_id = auth.uid());

**Initial Setup Procedure**

-- Create user\_management schema for auth hooks

CREATE SCHEMA IF NOT EXISTS user\_management;

-- Create function to handle new user registration

CREATE OR REPLACE FUNCTION user\_management.handle\_new\_user()

RETURNS TRIGGER AS $$

BEGIN

INSERT INTO public.profiles (id, email, full\_name)

VALUES (NEW.id, NEW.email, NEW.raw\_user\_meta\_data->>'full\_name');

-- Create default tags for the new user

INSERT INTO public.tags (user\_id, name, color, description, is\_auto\_generated)

VALUES

(NEW.id, 'Important', '#EF4444', 'High priority content', FALSE),

(NEW.id, 'Review', '#F59E0B', 'Content to review before exams', FALSE),

(NEW.id, 'Question', '#10B981', 'Topics I have questions about', FALSE);

RETURN NEW;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

-- Create trigger to handle new users

CREATE TRIGGER on\_auth\_user\_created

AFTER INSERT ON auth.users

FOR EACH ROW EXECUTE PROCEDURE user\_management.handle\_new\_user();

-- Grant necessary permissions

GRANT USAGE ON SCHEMA public TO authenticated;

GRANT USAGE ON SCHEMA public TO anon;

-- Standard grants for authenticated users

GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO authenticated;

GRANT USAGE ON ALL SEQUENCES IN SCHEMA public TO authenticated;

-- Limited grants for anonymous users

GRANT SELECT ON TABLE public.profiles TO anon;

This complete database schema for Supabase includes all tables, indexes, views, functions, triggers and RLS policies required for the Smart Note Organizer application. The schema enforces proper data isolation between users while enabling all the required functionality.