# PITCHCRAFT - COMPLETE PROJECT DOCUMENTATION

**Built at SMIT Hackathon 2025**
**Developer:** Sumair Khan
**Duration:** 12 hours
**Date:** 19-Oct-2025

---

## TABLE OF CONTENTS

---

## 1. PROJECT OVERVIEW

### What is PitchCraft?

PitchCraft is an AI-powered startup pitch generator that transforms rough business ideas into professional, investor-ready pitches in minutes. The platform leverages Google Gemini AI for content generation and Reve AI for logo design, providing entrepreneurs with a complete branding package.

### Problem Statement

Aspiring entrepreneurs face three major challenges:

- Lack of professional copywriting skills for pitches
- Cannot afford expensive designers for logos and branding
- No technical knowledge to build landing pages
- Limited time to prepare for investor meetings

### Solution

PitchCraft democratizes startup pitch creation by:

- Generating comprehensive pitches using AI
- Creating professional logos automatically
- Producing landing page content with brand colors
- Exporting everything to PDF format
- Enabling easy sharing via public links

### Target Users

- First-time founders
- Hackathon participants
- Student entrepreneurs
- Startup accelerator members
- Anyone with a business idea

---

## 2. TECH STACK

### Frontend Technologies

- **React.js** - UI library for building interactive interfaces
- **Vite** - Next-generation frontend tooling for fast builds
- **TailwindCSS** - Utility-first CSS framework
- **Lucide React** - Beautiful icon library
- **React Router DOM v6** - Client-side routing

### Backend & Database

- **Supabase** - Backend-as-a-Service platform
  - PostgreSQL Database
  - Authentication & Authorization
  - Cloud Storage
  - Real-time subscriptions

### AI Integration

- **Google Gemini 2.0 Flash** - Natural language generation
- **Reve AI** - Image generation for logos

### Additional Libraries

- **jsPDF** - PDF generation
- **html2canvas** - HTML to canvas conversion
- **React Hot Toast** - Toast notifications

### Development Tools

- Git & GitHub - Version control
- VS Code - Code editor
- Postman - API testing
- Chrome DevTools - Debugging

---

## 3. PROJECT STRUCTURE

```
hackathon-project/
│
├── src/
│   ├── components/            # Reusable UI Components
│   │   ├── Sidebar.jsx         # Left navigation sidebar
│   │   ├── TopBar.jsx          # Top action bar
│   │   ├── ChatArea.jsx        # Main content display
│   │   ├── InputArea.jsx       # User input section
│   │   ├── PitchCard.jsx       # Pitch display cards
│   │   ├── LandingPageModal.jsx # Landing page preview
│   │   └── index.js            # Component exports
│   │
│   ├── Pages/                 # Main application pages
│   │   ├── Login.jsx           # User login
│   │   ├── Signup.jsx          # User registration
│   │   ├── ForgotPassword.jsx  # Password reset request
│   │   ├── UpdatePassword.jsx  # New password entry
│   │   ├── Dashboard.jsx       # Main application
│   │   └── SharedPitch.jsx     # Public pitch view
│   │
│   ├── hooks/                 # Custom React Hooks
│   │   ├── usePitchGenerator.js # Pitch generation logic
│   │   ├── useLogoGenerator.js  # Logo creation logic
│   │   └── usePitchManagement.js# CRUD operations
│   │
│   ├── utils/                 # Helper Functions
│   │   ├── authHelper.js       # Authentication utilities
│   │   ├── pitchParser.js      # Response parsing
│   │   └── pdfExporter.js      # PDF generation
│   │
│   ├── lib/
│   │   └── supabaseClient.js   # Supabase configuration
│   │
│   ├── App.jsx                # Root component
│   └── main.jsx               # Application entry point
│
├── .env                       # Environment variables
├── vercel.json             # Deployment configuration
├── package.json              # Dependencies
└── README.md                   # Project documentation
```

---

# 4. APPLICATION FLOW

## 4.1 Authentication Flow

**Step 1: User Registration**

1. User opens application
2. Clicks "Sign up here" on login page
3. Navigates to `/signup`
4. Enters: Full Name, Phone, Email, Password
5. Submits form
6. Supabase creates user account
7. User metadata stored: Display name, Phone
8. Success: Redirects to login page

**Step 2: User Login**

1. User enters email and password
2. Supabase validates credentials
3. JWT token generated
4. User object stored in localStorage
5. Navigates to `/dashboard`

**Step 3: Password Reset**

1. User clicks "Forgot password"
2. Enters email address
3. Supabase sends reset link to email
4. User clicks link in email
5. Redirects to `/update-password?token=...`
6. Enters new password twice
7. Supabase updates password
8. User signed out and redirected to login

**Code Example:**

```javascript
// Login Handler
const handleLogin = async (e) => {
  e.preventDefault();
  setLoading(true);
  const { data, error } = await supabase.auth.signInWithPassword({
    email,
    password,
  });

  if (error) {
    setError(error.message);
  } else {
    localStorage.setItem("user", JSON.stringify(data.user));
    navigate("/dashboard");
  }

  setLoading(false);
};
```

---

### 4.2 Dashboard Initialization Flow

**Step 1: Check Authentication**

javascript

```
useEffect(() => {
  const checkAuth = async () => {
    // Check Supabase session
    const { data: { session } } = await supabase.auth.getSession();

    if (session?.user) {
      setUser(session.user);
      localStorage.setItem("user", JSON.stringify(session.user));

      // Fetch user's saved pitches
      fetchSavedPitches(session.user.id);

      // Initialize or restore conversation
      const savedConvId = localStorage.getItem("currentConversationId");
      if (savedConvId) {
        setCurrentConversationId(savedConvId);
        fetchConversationHistory(savedConvId);
      } else {
        const newConvId = `conv_${Date.now()}_${Math.random().toString(36).substr(2, 9)}`;
        setCurrentConversationId(newConvId);
        localStorage.setItem("currentConversationId", newConvId);
      }
    } else {
      navigate("/");
    }
  };
```

````
  checkAuth();
}, []);
````

**Step 2: Fetch Saved Pitches**

javascript
~~~
const fetchSavedPitches = async (userId) => {
  const { data, error } = await supabase
    .from("pitches")
    .select("*")
    .eq("user_id", userId)
    .order("created_at", { ascending: false });

  if (!error) {
    setSavedPitches(data);
  }
};
```


---

### **4.3 Pitch Generation Flow**

**Complete Flow Diagram:**
```
User Input
    ↓
Validation Check
    ↓
Build AI Prompt
    ↓
Call Gemini API
    ↓
Receive Response
    ↓
Parse Response
    ↓
Save to Database
    ↓
Update UI
    ↓
Save to Conversation History
~~~

**Step-by-Step Implementation:**

**Step 1: User Input**

```javascript
// InputArea.jsx
<textarea
  placeholder="Describe your startup idea..."
  value={prompt}
  onChange={(e) => setPrompt(e.target.value)}
  onKeyDown={(e) => {
    if (e.key === "Enter" && !e.shiftKey) {
      e.preventDefault();
      generatePitch();
    }
  }}
/>
```

**Step 2: Validation**

```javascript
// usePitchGenerator.js
const isValidStartupPrompt = (text) => {
  const lowerText = text.toLowerCase().trim();

  if (!lowerText) return false;

  // If previous pitch exists, allow follow-up questions
  if (responseData?.name) return true;

  // Reject general knowledge questions
  const rejectedPatterns = [
    /^who is (donald trump|elon musk)/i,
    /^what is the capital of/i,
    /^tell me a joke/i,
  ];

  const isRejected = rejectedPatterns.some(pattern =>
    pattern.test(lowerText)
  );
  if (isRejected) return false;

  // Check for startup keywords
  const startupKeywords = [
    "startup", "app", "idea", "product", "business",
    "service", "platform", "build", "create", "solve"
  ];

  const hasKeyword = startupKeywords.some(keyword =>
    lowerText.includes(keyword)
  );
  // Allow long descriptions
  const isLongDescription = text.trim().length > 40;

  return hasKeyword || isLongDescription;
};
```

**Step 3: Build Prompt for Gemini**

javascript
```javascript
const fullPrompt = `
You are a startup pitch assistant.
Task: Generate a startup name, tagline, pitch, target audience,
landing page content, brand colors, and logo concept.

Format your response EXACTLY in this structure:

Startup Name: [name]
Tagline: [tagline]
Pitch: [2-3 sentence elevator pitch]
Target Audience: [describe ideal customers]
Landing Page Content:
Hero Section: [content]
Problem Statement: [content]
Solution: [content]
Key Features:
- [feature 1]
- [feature 2]
- [feature 3]
Call to Action: [content]
Brand Colors: [#hex1, #hex2, #hex3, #hex4, #hex5]
Logo Concept: [description]

Business Idea: ${prompt}
Tone: ${tone}
`;
```

**Step 4: Call Gemini API**

javascript
```javascript
const generatePitch = async () => {
  setLoading(true);

  const API_KEY = import.meta.env.VITE_GEMINI_API_KEY;
  const url =
`https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateConten
t?key=${API_KEY}`;
```

```
  const res = await fetch(url, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      contents: [{ parts: [{ text: fullPrompt }] }],
    }),
  });

  const data = await res.json();
  const text = data?.candidates?.[0]?.content?.parts?.[0]?.text;

  // Continue to parsing...
  setLoading(false);
};
```

**Step 5: Parse Response**

javascript

```
const parsePitchResponse = (text) => {
  const lines = text.split("\n").map((l) => l.trim());
  const sections = {
    name: "",
    tagline: "",
    pitch: "",
    audience: "",
    landing: "",
    colors: [],
    logoIdea: "",
  };

  let currentKey = null;
  let landingContent = "";

  lines.forEach((line) => {
    if (/^Startup Name:/i.test(line)) {
      currentKey = "name";
      sections.name = line.replace(/^(Startup Name:|Name:)\s*/i, "");
    } else if (/^Tagline:/i.test(line)) {
      currentKey = "tagline";
      sections.tagline = line.replace(/^Tagline:\s*/i, "");
    } else if (/^Pitch:/i.test(line)) {
      currentKey = "pitch";
      sections.pitch = line.replace(/^Pitch:\s*/i, "");
    }
    // ... continue for other sections
  });
```

~~~
  sections.landing = landingContent.trim();
  return sections;
};
~~~

**Step 6: Save to Database**

javascript
~~~~
const { data: insertedData, error } = await supabase
  .from("pitches")
  .insert([{
    user_id: user.id,
    conversation_id: currentConversationId,
    idea: prompt,
    tone,
    name: parsed.name,
    tagline: parsed.tagline,
    pitch: parsed.pitch,
    audience: parsed.audience,
    landing: parsed.landing,
    colors: parsed.colors.join(","),
    logo_idea: parsed.logoIdea,
    is_latest_pitch: true,
  }])
  .select();

if (!error) {
  setCurrentChatId(insertedData[0].id);
  fetchSavedPitches(user.id);
  toast.success("Pitch generated and saved!");
}
```


---

### **4.4 Logo Generation Flow**

**Complete Flow:**
```
User Clicks "Generate AI Logo"
    ↓
Extract Logo Concept from Pitch
    ↓
Build Prompt for Reve AI
    ↓
Call Reve API
    ↓
~~~~

```
Receive Base64 Image
    ↓
Convert Base64 to Blob
    ↓
Upload to Supabase Storage
    ↓
Get Public URL
    ↓
Save URL to Database
    ↓
Display Logo in UI
```

**Implementation:**

**Step 1: Initiate Generation**

javascript
```
const generateLogo = async () => {
  if (!responseData?.logoIdea) {
    toast.error("Logo concept not found. Generate a pitch first.");
    return;
  }

  setLogoGenerating(true);

  // Continue...
};
```

**Step 2: Call Reve AI**

javascript
```
const API_URL = "https://api.reve.com/v1/image/create";
const API_KEY = import.meta.env.VITE_REVE_API_KEY;

const res = await fetch(API_URL, {
  method: "POST",
  headers: {
    Authorization: `Bearer ${API_KEY}`,
    "Content-Type": "application/json",
    Accept: "application/json",
  },
  body: JSON.stringify({
    prompt: `Professional startup logo: ${responseData.logoIdea}. Modern, clean, minimalist 
design for ${responseData.name}`,
    aspect_ratio: "1:1",
    version: "latest",
  }),
});

const data = await res.json();
const base64Image = `data:image/png;base64,${data.image}`;
```

**Step 3: Convert to Blob**

javascript
```
const uploadLogoToSupabase = async (base64Image, logoName) => {
  // Extract base64 data
  const base64Data = base64Image.split(',')[1];
  const byteCharacters = atob(base64Data);
  const byteNumbers = new Array(byteCharacters.length);

  for (let i = 0; i < byteCharacters.length; i++) {
    byteNumbers[i] = byteCharacters.charCodeAt(i);
  }

  const byteArray = new Uint8Array(byteNumbers);
  const blob = new Blob([byteArray], { type: 'image/png' });

  // Continue to upload...
};
```

**Step 4: Upload to Storage**

javascript
```
const timestamp = Date.now();
const fileName = `${logoName.replace(/[^a-zA-Z0-9]/g, '-')}-${timestamp}.png`;
const filePath = `logos/${user.id}/${fileName}`;

const { data, error } = await supabase.storage
  .from('hackathon-images')
  .upload(filePath, blob, {
    contentType: 'image/png',
    upsert: false
  });

if (error) throw error;

// Get public URL
const { data: urlData } = supabase.storage
  .from('hackathon-images')
  .getPublicUrl(filePath);

return urlData.publicUrl;
```

**Step 5: Save to Database**

javascript
```
const publicUrl = await uploadLogoToSupabase(base64Image, responseData.name);

setGeneratedLogoUrl(publicUrl);
toast.success("Logo generated and saved successfully!");

if (currentChatId) {
  await supabase
    .from("pitches")
    .update({ generated_logo_url: publicUrl })
    .eq("id", currentChatId);

  fetchSavedPitches(user.id);
}
```

---

### 4.5 Landing Page Preview Flow

**Step 1: Parse Landing Content**

javascript
```
const formatLandingPage = (content) => {
  const sections = {
    hero: "",
    problem: "",
    solution: "",
    features: [],
    cta: "",
  };

  const lines = content.split("\n");
  let currentSection = null;

  lines.forEach((line) => {
    const trimmed = line.trim();
    if (!trimmed) return;

    if (/^Hero Section:/i.test(trimmed)) {
      currentSection = "hero";
      sections.hero = trimmed.replace(/^Hero Section:\s*/i, "");
    } else if (/^Problem Statement:/i.test(trimmed)) {
      currentSection = "problem";
      sections.problem = trimmed.replace(/^Problem Statement:\s*/i, "");
    }
    // ... continue for other sections
  });

  return sections;
};
```

**Step 2: Render Modal**

javascript
```
// LandingPageModal.jsx
const LandingPageModal = ({ showLandingPage, setShowLandingPage, responseData })
=> {
  const sections = formatLandingPage(responseData.landing);
  const colors = responseData.colors;

  return (
    <div className="fixed inset-0 bg-black/80 z-50 overflow-y-auto">
      {/* Hero Section */}
      <div style={{
        background: `linear-gradient(135deg, ${colors[0]}, ${colors[1]})`
      }}>
        <h1>{responseData.name}</h1>
        <p>{sections.hero}</p>
      </div>

      {/* Problem Section */}
      <div>
        <h2>The Problem</h2>
        <p>{sections.problem}</p>
      </div>

      {/* Solution Section */}
      <div>
        <h2>Our Solution</h2>
        <p>{sections.solution}</p>
      </div>

      {/* Features Section */}
      <div>
        {sections.features.map((feature, idx) => (
          <div key={idx}>{feature}</div>
        ))}
      </div>

      {/* CTA Section */}
      <div>
        <h2>Ready to Get Started?</h2>
        <p>{sections.cta}</p>
    </div>
   </div>
  );
};
```

---

## 4.6 PDF Export Flow

**Step 1: Initialize PDF**

javascript
```
const handleExportPDF = async (responseData, generatedLogoUrl) => {
  const doc = new jsPDF();
  const pageWidth = doc.internal.pageSize.getWidth();
  const pageHeight = doc.internal.pageSize.getHeight();
  const margin = 20;
  let yPos = 20;

  // Add header
  doc.setFillColor(6, 182, 212);
  doc.rect(0, 0, pageWidth, 30, "F");
  doc.setTextColor(255, 255, 255);
  doc.text("PitchCraft", margin, 18);

  yPos = 45;

  // Continue...
};
```

**Step 2: Add Text Content**

javascript
```
// Startup Name
doc.setTextColor(0, 0, 0);
doc.setFontSize(22);
doc.setFont("helvetica", "bold");
doc.text(responseData.name, margin, yPos);
yPos += 15;

// Tagline
doc.setFontSize(14);
doc.setFont("helvetica", "italic");
doc.text(responseData.tagline, margin, yPos);
yPos += 20;

// Elevator Pitch
```

```
doc.setFont("helvetica", "bold");
doc.setTextColor(6, 182, 212);
doc.text("Elevator Pitch", margin, yPos);
yPos += 8;
doc.setFont("helvetica", "normal");
doc.setTextColor(0, 0, 0);
const pitchLines = doc.splitTextToSize(responseData.pitch, maxWidth);
doc.text(pitchLines, margin, yPos);
yPos += pitchLines.length * 6 + 12;
```

**Step 3: Capture Landing Page**

javascript

```
// Create temporary container
const tempContainer = document.createElement("div");
tempContainer.style.position = "absolute";
tempContainer.style.left = "-9999px";
tempContainer.style.width = "1200px";
document.body.appendChild(tempContainer);

// Generate HTML
const sections = formatLandingPage(responseData.landing);
tempContainer.innerHTML = `
  <div style="width: 1200px; background: white;">
    <!-- Hero Section -->
    <div style="background: linear-gradient(135deg, ${colors[0]}, ${colors[1]});">
      <h1>${responseData.name}</h1>
      <p>${sections.hero}</p>
    </div>
    <!-- Other sections... -->
  </div>
`;

// Capture as image
const canvas = await html2canvas(tempContainer, {
  scale: 2,
  useCORS: true,
  backgroundColor: '#ffffff'
});

document.body.removeChild(tempContainer);

// Add to PDF
const imgData = canvas.toDataURL("image/png");
const imgWidth = pageWidth - 2 * margin;
const imgHeight = (canvas.height * imgWidth) / canvas.width;
```

```
doc.addImage(imgData, "PNG", margin, yPos, imgWidth, imgHeight);
```

**Step 4: Add Logo**

javascript
```
if (generatedLogoUrl) {
  const logoImg = await loadImageAsBase64(generatedLogoUrl);
  const logoSize = 60;
  doc.addImage(logoImg, "PNG", margin, yPos, logoSize, logoSize);
}

// Helper function
const loadImageAsBase64 = (url) => {
  return new Promise((resolve, reject) => {
    const img = new Image();
    img.crossOrigin = "anonymous";
    img.onload = () => {
      const canvas = document.createElement("canvas");
      canvas.width = img.width;
      canvas.height = img.height;
      const ctx = canvas.getContext("2d");
      ctx.drawImage(img, 0, 0);
      resolve(canvas.toDataURL("image/png"));
    };
    img.onerror = reject;
    img.src = url;
  });
};
```

**Step 5: Add Footer & Download**

javascript
```
// Add page numbers
const totalPages = doc.internal.pages.length - 1;
for (let i = 1; i <= totalPages; i++) {
  doc.setPage(i);
  doc.setFontSize(8);
  doc.setTextColor(150, 150, 150);
  doc.text(
    `Generated by PitchCraft • Page ${i} of ${totalPages}`,
    pageWidth / 2,
    pageHeight - 10,
    { align: "center" }
  );
}

// Download
```

```
doc.save(`PitchCraft-${responseData.name}.pdf`);
toast.success("PDF downloaded successfully!");
```

---

### 4.7 Share Pitch Flow

#### Step 1: Generate Share Link

javascript
```
const handleShare = async () => {
  if (!currentChatId) {
    toast.error("No pitch to share");
    return;
  }

  const shareUrl = `${window.location.origin}/pitch/${currentChatId}`;
  await navigator.clipboard.writeText(shareUrl);
  toast.success("Share link copied to clipboard!");
};
```

#### Step 2: Public View Page

javascript
```
// SharedPitch.jsx
const SharedPitch = () => {
  const { id } = useParams();
  const [pitch, setPitch] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchPitch = async () => {
      const { data, error } = await supabase
        .from("pitches")
        .select("*")
        .eq("id", id)
        .single();

      if (error) {
        toast.error("Pitch not found");
      } else {
        setPitch({
          name: data.name,
          tagline: data.tagline,
          pitch: data.pitch,
          audience: data.audience,
          colors: data.colors.split(","),
          logoIdea: data.logo_idea,
          generatedLogoUrl: data.generated_logo_url,
        });
      }
      setLoading(false);
    };

    fetchPitch();
  }, [id]);

  if (loading) return <div>Loading...</div>;
  if (!pitch) return <div>Pitch not found</div>;

  return (
    <div>
      {/* Display pitch content */}
      <h1>{pitch.name}</h1>
      <p>{pitch.tagline}</p>
      {/* ... other content */}
    </div>
  );
};
```

---

## 4.8 Edit & Save Flow

**Step 1: Enable Edit Mode**

javascript
```
const handleEdit = (field) => {
  if (!editedData) {
    setEditedData({ ...responseData });
  }
  setEditMode({ ...editMode, [field]: true });
};
```

**Step 2: Handle Input Changes**

javascript
```
const handleInputChange = (field, value) => {
  setEditedData({ ...editedData, [field]: value });
};
```

**Step 3: Save to Database**

javascript

```
const handleSave = async (field) => {
  setEditMode({ ...editMode, [field]: false });
  setResponseData({ ...editedData });

  if (currentChatId) {
    const { error } = await supabase
      .from("pitches")
      .update({ [field]: editedData[field] })
      .eq("id", currentChatId);

    if (error) {
      toast.error("Failed to save changes");
    } else {
      toast.success("Changes saved!");
      fetchSavedPitches(user.id);
    }
  }
};
```

**Step 4: UI Implementation**

javascript
```
// PitchCard.jsx
{editMode.name ? (
  <input
    type="text"
    value={editedData?.name || ""}
    onChange={(e) => handleInputChange("name", e.target.value)}
    onBlur={() => handleSave("name")}
    autoFocus
    className="text-3xl font-bold text-white bg-slate-700/50 border border-cyan-500
rounded-lg px-3 py-2 w-full"
  />
) : (
  <h3
    className="text-3xl font-bold text-white cursor-pointer hover:text-cyan-400"
    onClick={() => handleEdit("name")}
  >
    {responseData.name}
  </h3>
)}
```

---

## 5. DATABASE SCHEMA

## 5.1 Supabase Tables

**Table 1: pitches**

sql
```
CREATE TABLE pitches (
  id SERIAL PRIMARY KEY,
  user_id UUID NOT NULL REFERENCES auth.users(id) ON DELETE CASCADE,
  conversation_id TEXT,
  idea TEXT NOT NULL,
  tone TEXT DEFAULT 'Formal',
  name TEXT NOT NULL,
  tagline TEXT,
  pitch TEXT,
  audience TEXT,
  landing TEXT,
  colors TEXT,
  logo_idea TEXT,
  generated_logo_url TEXT,
  is_latest_pitch BOOLEAN DEFAULT true,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indexes for performance
CREATE INDEX idx_pitches_user_id ON pitches(user_id);
CREATE INDEX idx_pitches_created_at ON pitches(created_at DESC);

CREATE INDEX idx_pitches_conversation_id ON pitches(conversation_id);
```

**Field Descriptions:**

- `id` - Unique identifier (auto-increment)
- `user_id` - Foreign key to auth.users
- `conversation_id` - Groups related pitches
- `idea` - Original user input
- `tone` - Selected tone (Formal, Fun, Professional, Casual)
- `name` - Generated startup name
- `tagline` - Generated tagline
- `pitch` - Elevator pitch
- `audience` - Target audience description
- `landing` - Landing page content
- `colors` - Comma-separated hex colors
- `logo_idea` - AI-generated logo concept
- `generated_logo_url` - Public URL of uploaded logo
- `is_latest_pitch` - Flag for latest version

- `created_at` - Timestamp of creation
- `updated_at` - Timestamp of last update

---

**Table 2: pitch_conversations**

sql

```
CREATE TABLE pitch_conversations (
  id SERIAL PRIMARY KEY,
  conversation_id TEXT NOT NULL,
  user_id UUID NOT NULL REFERENCES auth.users(id) ON DELETE CASCADE,
  pitch_id INTEGER REFERENCES pitches(id) ON DELETE SET NULL,
  message_type TEXT CHECK (message_type IN ('user_prompt', 'user_question',
'ai_response')),
  user_message TEXT NOT NULL,
  response_data JSONB,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Indexes
CREATE INDEX idx_conversations_user_id ON pitch_conversations(user_id);
CREATE INDEX idx_conversations_conv_id ON pitch_conversations(conversation_id);

CREATE INDEX idx_conversations_created_at ON pitch_conversations(created_at DESC);
```

**Field Descriptions:**

- `id` - Unique identifier
- `conversation_id` - Groups messages in same conversation
- `user_id` - Foreign key to auth.users
- `pitch_id` - Optional reference to pitch
- `message_type` - Type of message (prompt, question, response)
- `user_message` - User's input text
- `response_data` - AI response stored as JSON
- `created_at` - Timestamp

---

## 5.2 Row Level Security (RLS) Policies

**Pitches Table Policies:**