PCBancard AI Coaching App - RAG Implementation Guide

Overview

This guide provides working code to transform your existing Replit coaching app into a scalable, continuously-learning AI system that can:

- Get smarter with every content upload
- Handle 20+ simultaneous users
- Provide context-aware role-play scenarios

Architecture

```
┌─────────────────────────────────────────────────────────────────┐
│                    Replit Autoscale Deployment                  │
├─────────────────────────────────────────────────────────────────┤
│  ┌─────────────────────────────────────────────────────────┐   │
│  │  Frontend (React)                                        │   │
│  │  - Role-play interface                                   │   │
│  │  - Admin content upload panel                            │   │
│  │  - User session management                               │   │
│  └─────────────────────────────────────────────────────────┘   │
│                              │                                  │
│  ┌─────────────────────────────────────────────────────────┐   │
│  │  Backend (Node.js/Express)                               │   │
│  │  - /api/chat - Role-play conversations                   │   │
│  │  - /api/upload - Content ingestion                       │   │
│  │  - /api/search - RAG retrieval                           │   │
│  └─────────────────────────────────────────────────────────┘   │
│          │              │                    │                  │
│          ▼              ▼                    ▼                  │
│  ┌──────────┐  ┌─────────────┐    ┌─────────────────┐          │
│  │PostgreSQL│  │Supabase     │    │ Anthropic/OpenAI│          │
│  │Sessions  │  │pgvector     │    │ API             │          │
│  │Users     │  │Embeddings   │    │                 │          │
│  └──────────┘  └─────────────┘    └─────────────────┘          │
```

Part 1: Database Schema

File: `database/schema.sql`

```
-- Run this in Replit's PostgreSQL or Supabase

-- Enable vector extension (Supabase has this; for Replit PostgreSQL, use text se
CREATE EXTENSION IF NOT EXISTS vector;

-- Users table
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  email VARCHAR(255) UNIQUE NOT NULL,
  name VARCHAR(255),
  role VARCHAR(50) DEFAULT 'agent', -- 'agent', 'admin'
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Chat sessions for role-play
CREATE TABLE chat_sessions (
  id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES users(id),
  scenario_type VARCHAR(100), -- 'cold_call', 'objection_handling', 'closing', 'p
  started_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  ended_at TIMESTAMP,
  score INTEGER,
  feedback TEXT
);

-- Individual messages within sessions
CREATE TABLE chat_messages (
  id SERIAL PRIMARY KEY,
  session_id INTEGER REFERENCES chat_sessions(id),
  role VARCHAR(20) NOT NULL, -- 'user', 'assistant', 'system'
  content TEXT NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Knowledge base content (the brain)
CREATE TABLE knowledge_content (
  id SERIAL PRIMARY KEY,
  title VARCHAR(500),
  content TEXT NOT NULL,
  content_type VARCHAR(100), -- 'objection', 'script', 'product_info', 'success_s
  category VARCHAR(100), -- 'PayLo', 'dual_pricing', 'POS', 'cold_calling', etc.
  tags TEXT[], -- Array of searchable tags
  source VARCHAR(255), -- Where it came from
  embedding vector(1536), -- OpenAI ada-002 embeddings (or 384 for smaller models
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Create index for vector similarity search
CREATE INDEX ON knowledge_content USING ivfflat (embedding vector_cosine_ops) WIT

-- Full-text search index as fallback
CREATE INDEX idx_knowledge_content_search ON knowledge_content
USING gin(to_tsvector('english', title || ' ' || content));

-- Scenarios/personas for role-play
CREATE TABLE roleplay_scenarios (
  id SERIAL PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  description TEXT,
  persona_prompt TEXT NOT NULL, -- The AI persona instructions
  difficulty VARCHAR(50), -- 'beginner', 'intermediate', 'advanced'
  category VARCHAR(100),
  success_criteria TEXT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Insert default scenarios
INSERT INTO roleplay_scenarios (name, description, persona_prompt, difficulty, ca
('Skeptical Business Owner',
 'A restaurant owner who has been burned by payment processors before',
 'You are Mike, owner of "Mikes Diner" for 15 years. You currently pay 3.2% effec
 'intermediate',
 'cold_call'),

('Price-Focused Retailer',
 'A retail shop owner who only cares about the bottom line',
 'You are Sandra, owner of a boutique clothing store. You process about $40k/mont
 'advanced',
 'objection_handling'),

('Friendly but Non-Committal',
 'A business owner who seems interested but won''t commit',
 'You are Dave, owner of an auto repair shop. You''re friendly and seem intereste
```

```
  'intermediate',
  'closing'),

 ('Compliance-Worried Professional',
  'A medical office manager worried about PCI compliance',
  'You are Jennifer, office manager at a dental practice. You process cards for co
  'advanced',
  'compliance');
```

Part 2: Backend API

File: `server/index.js`

```
const express = require('express');
const cors = require('cors');
const { Pool } = require('pg');
const multer = require('multer');
const pdf = require('pdf-parse');
const Anthropic = require('@anthropic-ai/sdk');
const OpenAI = require('openai');

const app = express();
app.use(cors());
app.use(express.json({ limit: '10mb' }));

// Database connection
const pool = new Pool({
  connectionString: process.env.DATABASE_URL,
  ssl: process.env.NODE_ENV === 'production' ? { rejectUnauthorized: false } : fa
});

// AI clients
const anthropic = new Anthropic({ apiKey: process.env.ANTHROPIC_API_KEY });
const openai = new OpenAI({ apiKey: process.env.OPENAI_API_KEY });

// File upload config
const upload = multer({ storage: multer.memoryStorage() });

// ============================================
// EMBEDDING & RAG FUNCTIONS
// ============================================

async function generateEmbedding(text) {
  const response = await openai.embeddings.create({
    model: 'text-embedding-ada-002',
    input: text.slice(0, 8000) // Token limit safety
  });
  return response.data[0].embedding;
}

async function searchKnowledge(query, limit = 5, category = null) {
  const queryEmbedding = await generateEmbedding(query);
  const embeddingStr = `[${queryEmbedding.join(',')}]`;

  let sql = `
    SELECT id, title, content, content_type, category, tags,
           1 - (embedding <=> $1::vector) as similarity
    FROM knowledge_content
    WHERE embedding IS NOT NULL
  `;
  const params = [embeddingStr];

  if (category) {
    sql += ` AND category = $2`;
    params.push(category);
  }

  sql += ` ORDER BY embedding <=> $1::vector LIMIT $${params.length + 1}`;
  params.push(limit);

  const result = await pool.query(sql, params);
  return result.rows;
}

// Fallback text search if vector DB not available
async function textSearchKnowledge(query, limit = 5, category = null) {
  let sql = `
    SELECT id, title, content, content_type, category, tags,
           ts_rank(to_tsvector('english', title || ' ' || content),
                   plainto_tsquery('english', $1)) as rank
    FROM knowledge_content
    WHERE to_tsvector('english', title || ' ' || content) @@ plainto_tsquery('eng
  `;
  const params = [query];

  if (category) {
    sql += ` AND category = $2`;
    params.push(category);
  }
  sql += ` ORDER BY rank DESC LIMIT $${params.length + 1}`;
  params.push(limit);

  const result = await pool.query(sql, params);
  return result.rows;
}

// ============================================
// CONTENT CHUNKING
// ============================================

function chunkText(text, maxChunkSize = 1000, overlap = 200) {
  const chunks = [];
  const sentences = text.split(/(?<=[.!?])\s+/);
  let currentChunk = '';

  for (const sentence of sentences) {
    if ((currentChunk + sentence).length > maxChunkSize && currentChunk) {
      chunks.push(currentChunk.trim());
      // Keep overlap from previous chunk
      const words = currentChunk.split(' ');
      currentChunk = words.slice(-Math.floor(overlap / 5)).join(' ') + ' ' + sent
    } else {
      currentChunk += (currentChunk ? ' ' : '') + sentence;
    }
  }

  if (currentChunk.trim()) {
    chunks.push(currentChunk.trim());
  }

  return chunks;
}

// ============================================
// API ENDPOINTS
// ============================================

// Health check
app.get('/api/health', (req, res) => {
  res.json({ status: 'ok', timestamp: new Date().toISOString() });
});

// Upload and process content
app.post('/api/content/upload', upload.single('file'), async (req, res) => {
  try {
    const { title, content_type, category, tags } = req.body;
```

```
let textContent = '';

if (req.file) {
  // Handle file upload
  if (req.file.mimetype === 'application/pdf') {
    const pdfData = await pdf(req.file.buffer);
    textContent = pdfData.text;
  } else if (req.file.mimetype.startsWith('text/')) {
    textContent = req.file.buffer.toString('utf-8');
  } else {
    return res.status(400).json({ error: 'Unsupported file type' });
  }
} else if (req.body.content) {
  textContent = req.body.content;
} else {
  return res.status(400).json({ error: 'No content provided' });
}

// Chunk the content
const chunks = chunkText(textContent);
const insertedIds = [];

for (let i = 0; i < chunks.length; i++) {
  const chunk = chunks[i];
  const chunkTitle = chunks.length > 1 ? `${title} (Part ${i + 1})` : title;

  // Generate embedding
  const embedding = await generateEmbedding(chunk);
  const embeddingStr = `[${embedding.join(',')}]`;

  const result = await pool.query(`
    INSERT INTO knowledge_content (title, content, content_type, category, ta
    VALUES ($1, $2, $3, $4, $5, $6::vector, $7)
    RETURNING id
  `, [
    chunkTitle,
    chunk,
    content_type,
    category,
    tags ? tags.split(',').map(t => t.trim()) : [],
    embeddingStr,
    req.file?.originalname || 'manual_entry'
  ]);

  insertedIds.push(result.rows[0].id);
}
```

```
    res.json({
      success: true,
      message: `Processed ${chunks.length} chunks`,
      ids: insertedIds
    });

  } catch (error) {
    console.error('Upload error:', error);
    res.status(500).json({ error: error.message });
  }
});

// Bulk upload from JSON
app.post('/api/content/bulk', async (req, res) => {
  try {
    const { items } = req.body; // Array of {title, content, content_type, catego
    const insertedIds = [];

    for (const item of items) {
      const embedding = await generateEmbedding(item.content);
      const embeddingStr = `[${embedding.join(',')}]`;

      const result = await pool.query(`
        INSERT INTO knowledge_content (title, content, content_type, category, ta
        VALUES ($1, $2, $3, $4, $5, $6::vector)
        RETURNING id
      `, [
        item.title,
        item.content,
        item.content_type,
        item.category,
        item.tags || [],
        embeddingStr
      ]);

      insertedIds.push(result.rows[0].id);
    }

    res.json({ success: true, count: insertedIds.length, ids: insertedIds });

  } catch (error) {
    console.error('Bulk upload error:', error);
    res.status(500).json({ error: error.message });
  }
});

// Search knowledge base
```

```
app.post('/api/search', async (req, res) => {
  try {
    const { query, limit = 5, category } = req.body;
    const results = await searchKnowledge(query, limit, category);
    res.json({ results });
  } catch (error) {
    console.error('Search error:', error);
    // Fallback to text search
    try {
      const results = await textSearchKnowledge(req.body.query, req.body.limit, r
      res.json({ results, fallback: true });
    } catch (fallbackError) {
      res.status(500).json({ error: error.message });
    }
  }
});

// Get all scenarios
app.get('/api/scenarios', async (req, res) => {
  try {
    const result = await pool.query(`
      SELECT id, name, description, difficulty, category
      FROM roleplay_scenarios
      ORDER BY difficulty, name
    `);
    res.json({ scenarios: result.rows });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

// Start a new role-play session
app.post('/api/session/start', async (req, res) => {
  try {
    const { user_id, scenario_id } = req.body;

    // Get scenario
    const scenarioResult = await pool.query(
      'SELECT * FROM roleplay_scenarios WHERE id = $1',
      [scenario_id]
    );

    if (!scenarioResult.rows.length) {
      return res.status(404).json({ error: 'Scenario not found' });
    }

    const scenario = scenarioResult.rows[0];
    // Create session
    const sessionResult = await pool.query(`
      INSERT INTO chat_sessions (user_id, scenario_type)
      VALUES ($1, $2)
      RETURNING id
    `, [user_id, scenario.category]);

    const session_id = sessionResult.rows[0].id;

    // Store system message
    await pool.query(`
      INSERT INTO chat_messages (session_id, role, content)
      VALUES ($1, 'system', $2)
    `, [session_id, scenario.persona_prompt]);

    res.json({
      session_id,
      scenario: {
        name: scenario.name,
        description: scenario.description,
        difficulty: scenario.difficulty
      }
    });

  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

// Send message in role-play (with RAG)
app.post('/api/session/:sessionId/message', async (req, res) => {
  try {
    const { sessionId } = req.params;
    const { content } = req.body;

    // Get session info and messages
    const sessionResult = await pool.query(
      'SELECT * FROM chat_sessions WHERE id = $1',
      [sessionId]
    );

    if (!sessionResult.rows.length) {
      return res.status(404).json({ error: 'Session not found' });
    }

    const session = sessionResult.rows[0];
    // Get conversation history
    const messagesResult = await pool.query(`
      SELECT role, content FROM chat_messages
      WHERE session_id = $1
      ORDER BY created_at
    `, [sessionId]);

    // Search for relevant knowledge to inject
    const relevantKnowledge = await searchKnowledge(content, 3, session.scenario_

    // Build context injection
    let knowledgeContext = '';
    if (relevantKnowledge.length > 0) {
      knowledgeContext = `\n\n[COACHING CONTEXT - Use this to evaluate the rep's
${relevantKnowledge.map(k => `- ${k.title}: ${k.content.slice(0, 500)}`).join('\n
    }

    // Build messages array for Claude
    const systemMessage = messagesResult.rows.find(m => m.role === 'system');
    const conversationHistory = messagesResult.rows
      .filter(m => m.role !== 'system')
      .map(m => ({ role: m.role, content: m.content }));

    // Store user message
    await pool.query(`
      INSERT INTO chat_messages (session_id, role, content)
      VALUES ($1, 'user', $2)
    `, [sessionId, content]);

    // Call Claude
    const response = await anthropic.messages.create({
      model: 'claude-sonnet-4-20250514',
      max_tokens: 1024,
      system: `${systemMessage.content}

You are playing a character in a sales training role-play. Stay in character comp
After your in-character response, on a new line starting with "---COACH---", prov
${knowledgeContext}`,
      messages: [
        ...conversationHistory,
        { role: 'user', content }
      ]
    });

    const aiResponse = response.content[0].text;
    // Parse response and coaching
    const [characterResponse, coachingFeedback] = aiResponse.split('---COACH---')

    // Store AI response
    await pool.query(`
      INSERT INTO chat_messages (session_id, role, content)
      VALUES ($1, 'assistant', $2)
    `, [sessionId, characterResponse.trim()]);

    res.json({
      response: characterResponse.trim(),
      coaching: coachingFeedback?.trim() || null,
      knowledge_used: relevantKnowledge.map(k => k.title)
    });

  } catch (error) {
    console.error('Message error:', error);
    res.status(500).json({ error: error.message });
  }
});

// End session and get final score
app.post('/api/session/:sessionId/end', async (req, res) => {
  try {
    const { sessionId } = req.params;

    // Get all messages
    const messagesResult = await pool.query(`
      SELECT role, content FROM chat_messages
      WHERE session_id = $1 AND role != 'system'
      ORDER BY created_at
    `, [sessionId]);

    const transcript = messagesResult.rows
      .map(m => `${m.role.toUpperCase()}: ${m.content}`)
      .join('\n\n');

    // Get comprehensive feedback from Claude
    const evaluation = await anthropic.messages.create({
      model: 'claude-sonnet-4-20250514',
      max_tokens: 1500,
      system: `You are an expert sales coach evaluating a role-play session for a
      messages: [{
        role: 'user',
        content: `Evaluate this sales role-play transcript:

${transcript}
Provide:
1. Overall score (1-100)
2. Top 3 strengths
3. Top 3 areas for improvement
4. Specific recommendations for next practice session
5. Would this have resulted in a sale? Why/why not?

Format as JSON with keys: score, strengths, improvements, recommendations, sale_l
      }]
    });

    let feedback;
    try {
      // Try to parse JSON from response
      const jsonMatch = evaluation.content[0].text.match(/\{[\s\S]*\}/);
      feedback = jsonMatch ? JSON.parse(jsonMatch[0]) : { raw: evaluation.content
    } catch {
      feedback = { raw: evaluation.content[0].text };
    }

    // Update session
    await pool.query(`
      UPDATE chat_sessions
      SET ended_at = CURRENT_TIMESTAMP,
          score = $1,
          feedback = $2
      WHERE id = $3
    `, [feedback.score || 0, JSON.stringify(feedback), sessionId]);

    res.json({ feedback });

  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

// Get user's session history
app.get('/api/user/:userId/sessions', async (req, res) => {
  try {
    const { userId } = req.params;
    const result = await pool.query(`
      SELECT id, scenario_type, started_at, ended_at, score
      FROM chat_sessions
      WHERE user_id = $1
      ORDER BY started_at DESC
      LIMIT 50
    `, [userId]);
    res.json({ sessions: result.rows });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

// List all content in knowledge base
app.get('/api/content', async (req, res) => {
  try {
    const { category, content_type } = req.query;
    let sql = 'SELECT id, title, content_type, category, tags, created_at FROM kn
    const params = [];

    if (category) {
      params.push(category);
      sql += ` AND category = $${params.length}`;
    }
    if (content_type) {
      params.push(content_type);
      sql += ` AND content_type = $${params.length}`;
    }

    sql += ' ORDER BY created_at DESC LIMIT 100';

    const result = await pool.query(sql, params);
    res.json({ content: result.rows });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

// Delete content
app.delete('/api/content/:id', async (req, res) => {
  try {
    await pool.query('DELETE FROM knowledge_content WHERE id = $1', [req.params.i
    res.json({ success: true });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

const PORT = process.env.PORT || 3001;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

Part 3: React Frontend Components

File: `client/src/components/AdminUpload.jsx`

```
import React, { useState, useEffect } from 'react';

const CATEGORIES = [
  'cold_calling',
  'objection_handling',
  'closing',
  'pricing',
  'compliance',
  'product_knowledge',
  'success_stories'
];

const CONTENT_TYPES = [
  'script',
  'objection',
  'product_info',
  'compliance',
  'success_story',
  'best_practice'
];

export default function AdminUpload() {
  const [title, setTitle] = useState('');
  const [content, setContent] = useState('');
  const [category, setCategory] = useState('');
  const [contentType, setContentType] = useState('');
  const [tags, setTags] = useState('');
  const [file, setFile] = useState(null);
  const [uploading, setUploading] = useState(false);
  const [message, setMessage] = useState('');
  const [existingContent, setExistingContent] = useState([]);

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

  const fetchExistingContent = async () => {
    try {
      const res = await fetch('/api/content');
      const data = await res.json();
      setExistingContent(data.content || []);
    } catch (err) {
      console.error('Error fetching content:', err);
    }
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setUploading(true);
    setMessage('');

    try {
      const formData = new FormData();
      formData.append('title', title);
      formData.append('category', category);
      formData.append('content_type', contentType);
      formData.append('tags', tags);

      if (file) {
        formData.append('file', file);
      } else {
        formData.append('content', content);
      }

      const res = await fetch('/api/content/upload', {
        method: 'POST',
        body: formData
      });

      const data = await res.json();

      if (data.success) {
        setMessage(`✅ Successfully uploaded! Created ${data.ids.length} knowledg
        setTitle('');
        setContent('');
        setFile(null);
        setTags('');
        fetchExistingContent();
      } else {
        setMessage(`❌ Error: ${data.error}`);
      }
    } catch (err) {
      setMessage(`❌ Error: ${err.message}`);
    }

    setUploading(false);
  };
```

```
const handleDelete = async (id) => {
  if (!confirm('Delete this content?')) return;

  try {
    await fetch(`/api/content/${id}`, { method: 'DELETE' });
    fetchExistingContent();
  } catch (err) {
    alert('Error deleting: ' + err.message);
  }
};

return (
  <div className="max-w-4xl mx-auto p-6">
    <h1 className="text-2xl font-bold mb-6">📚 Knowledge Base Admin</h1>

    <form onSubmit={handleSubmit} className="bg-white shadow rounded-lg p-6 mb-
      <h2 className="text-lg font-semibold mb-4">Add New Content</h2>

      <div className="grid grid-cols-2 gap-4 mb-4">
        <div>
          <label className="block text-sm font-medium mb-1">Title *</label>
          <input
            type="text"
            value={title}
            onChange={(e) => setTitle(e.target.value)}
            className="w-full border rounded px-3 py-2"
            placeholder="e.g., Handling 'I'm happy with my current processor'"
            required
          />
        </div>

        <div>
          <label className="block text-sm font-medium mb-1">Category *</label>
          <select
            value={category}
            onChange={(e) => setCategory(e.target.value)}
            className="w-full border rounded px-3 py-2"
            required
          >
            <option value="">Select category...</option>
            {CATEGORIES.map(c => (
              <option key={c} value={c}>{c.replace('_', ' ')}</option>
            ))}
          </select>
        </div>
```

```
          <div>
            <label className="block text-sm font-medium mb-1">Content Type *</lab
            <select
              value={contentType}
              onChange={(e) => setContentType(e.target.value)}
              className="w-full border rounded px-3 py-2"
              required
            >
              <option value="">Select type...</option>
              {CONTENT_TYPES.map(t => (
                <option key={t} value={t}>{t.replace('_', ' ')}</option>
              ))}
            </select>
          </div>

          <div>
            <label className="block text-sm font-medium mb-1">Tags (comma-separat
            <input
              type="text"
              value={tags}
              onChange={(e) => setTags(e.target.value)}
              className="w-full border rounded px-3 py-2"
              placeholder="e.g., PayLo, dual pricing, restaurant"
            />
          </div>
        </div>

        <div className="mb-4">
          <label className="block text-sm font-medium mb-1">Upload File (PDF or T
          <input
            type="file"
            accept=".pdf,.txt,.md"
            onChange={(e) => setFile(e.target.files[0])}
            className="w-full border rounded px-3 py-2"
          />
        </div>

        <div className="mb-4">
          <label className="block text-sm font-medium mb-1">Or Enter Content Dire
          <textarea
            value={content}
            onChange={(e) => setContent(e.target.value)}
            className="w-full border rounded px-3 py-2 h-40"
            placeholder="Paste scripts, objection handlers, product information..
            disabled={!!file}
          />
        </div>
        {message && (
          <div className={`mb-4 p-3 rounded ${message.includes('✅') ? 'bg-green-
            {message}
          </div>
        )}

        <button
          type="submit"
          disabled={uploading}
          className="bg-blue-600 text-white px-6 py-2 rounded hover:bg-blue-700 d
        >
          {uploading ? 'Processing...' : 'Upload Content'}
        </button>
      </form>

      <div className="bg-white shadow rounded-lg p-6">
        <h2 className="text-lg font-semibold mb-4">
          Existing Knowledge ({existingContent.length} items)
        </h2>

        <div className="space-y-2">
          {existingContent.map(item => (
            <div key={item.id} className="flex justify-between items-center p-3 b
              <div>
                <span className="font-medium">{item.title}</span>
                <span className="ml-2 text-xs bg-blue-100 text-blue-800 px-2 py-1
                  {item.category}
                </span>
                <span className="ml-2 text-xs bg-gray-200 px-2 py-1 rounded">
                  {item.content_type}
                </span>
              </div>
              <button
                onClick={() => handleDelete(item.id)}
                className="text-red-600 hover:text-red-800 text-sm"
              >
                Delete
              </button>
            </div>
          ))}
        </div>
      </div>
    </div>
  );
}
```

File: **`client/src/components/RolePlayChat.jsx`**

```
import React, { useState, useEffect, useRef } from 'react';

export default function RolePlayChat({ userId }) {
  const [scenarios, setScenarios] = useState([]);
  const [selectedScenario, setSelectedScenario] = useState(null);
  const [sessionId, setSessionId] = useState(null);
  const [messages, setMessages] = useState([]);
  const [input, setInput] = useState('');
  const [loading, setLoading] = useState(false);
  const [coaching, setCoaching] = useState(null);
  const [sessionEnded, setSessionEnded] = useState(false);
  const [feedback, setFeedback] = useState(null);
  const messagesEndRef = useRef(null);

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

  useEffect(() => {
    messagesEndRef.current?.scrollIntoView({ behavior: 'smooth' });
  }, [messages]);

  const fetchScenarios = async () => {
    const res = await fetch('/api/scenarios');
    const data = await res.json();
    setScenarios(data.scenarios);
  };

  const startSession = async (scenarioId) => {
    setLoading(true);
    const res = await fetch('/api/session/start', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ user_id: userId, scenario_id: scenarioId })
    });
    const data = await res.json();
    setSessionId(data.session_id);
    setSelectedScenario(data.scenario);
    setMessages([]);
    setCoaching(null);
    setSessionEnded(false);
    setFeedback(null);
    setLoading(false);
  };
```

```
const sendMessage = async (e) => {
  e.preventDefault();
  if (!input.trim() || loading) return;

  const userMessage = input.trim();
  setInput('');
  setMessages(prev => [...prev, { role: 'user', content: userMessage }]);
  setLoading(true);
  setCoaching(null);

  try {
    const res = await fetch(`/api/session/${sessionId}/message`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ content: userMessage })
    });
    const data = await res.json();

    setMessages(prev => [...prev, { role: 'assistant', content: data.response }
    if (data.coaching) {
      setCoaching(data.coaching);
    }
  } catch (err) {
    console.error('Error:', err);
  }

  setLoading(false);
};

const endSession = async () => {
  setLoading(true);
  const res = await fetch(`/api/session/${sessionId}/end`, {
    method: 'POST'
  });
  const data = await res.json();
  setFeedback(data.feedback);
  setSessionEnded(true);
  setLoading(false);
};

const resetSession = () => {
  setSessionId(null);
  setSelectedScenario(null);
  setMessages([]);
  setCoaching(null);
  setSessionEnded(false);
```

```
    setFeedback(null);
  };

  // Scenario selection screen
  if (!sessionId) {
    return (
      <div className="max-w-4xl mx-auto p-6">
        <h1 className="text-2xl font-bold mb-6">🎭 Sales Role-Play Training</h1>
        <p className="text-gray-600 mb-6">
          Select a scenario to practice. The AI will play the customer while coac
        </p>

        <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
          {scenarios.map(scenario => (
            <div
              key={scenario.id}
              className="border rounded-lg p-4 hover:shadow-md cursor-pointer tra
              onClick={() => startSession(scenario.id)}
            >
              <div className="flex justify-between items-start mb-2">
                <h3 className="font-semibold">{scenario.name}</h3>
                <span className={`text-xs px-2 py-1 rounded ${
                  scenario.difficulty === 'beginner' ? 'bg-green-100 text-green-8
                  scenario.difficulty === 'intermediate' ? 'bg-yellow-100 text-ye
                  'bg-red-100 text-red-800'
                }`}>
                  {scenario.difficulty}
                </span>
              </div>
              <p className="text-sm text-gray-600">{scenario.description}</p>
              <span className="text-xs text-blue-600 mt-2 inline-block">
                Category: {scenario.category}
              </span>
            </div>
          ))}
        </div>
      </div>
    );
  }

  // Chat interface
  return (
    <div className="max-w-4xl mx-auto p-6">
      <div className="flex justify-between items-center mb-4">
        <div>
          <h1 className="text-xl font-bold">{selectedScenario.name}</h1>
          <p className="text-sm text-gray-500">{selectedScenario.description}</p>
        </div>
        <div className="space-x-2">
          {!sessionEnded && (
            <button
              onClick={endSession}
              className="bg-orange-500 text-white px-4 py-2 rounded hover:bg-oran
            >
              End & Get Feedback
            </button>
          )}
          <button
            onClick={resetSession}
            className="bg-gray-500 text-white px-4 py-2 rounded hover:bg-gray-600
          >
            New Scenario
          </button>
        </div>
      </div>

      {/* Messages */}
      <div className="bg-white border rounded-lg h-96 overflow-y-auto p-4 mb-4">
        {messages.length === 0 && (
          <p className="text-gray-400 text-center mt-20">
            Start the conversation! You're the sales rep making the call.
          </p>
        )}

        {messages.map((msg, idx) => (
          <div
            key={idx}
            className={`mb-4 ${msg.role === 'user' ? 'text-right' : ''}`}
          >
            <span className={`inline-block px-4 py-2 rounded-lg max-w-[80%] ${
              msg.role === 'user'
                ? 'bg-blue-600 text-white'
                : 'bg-gray-100 text-gray-800'
            }`}>
              {msg.content}
            </span>
          </div>
        ))}

        {loading && (
          <div className="text-gray-400 animate-pulse">Typing...</div>
        )}

        <div ref={messagesEndRef} />
```

```
</div>

{/* Coaching panel */}
{coaching && (
  <div className="bg-yellow-50 border border-yellow-200 rounded-lg p-4 mb-4
    <h4 className="font-semibold text-yellow-800 mb-1">💡 Coach's Note:</h4
    <p className="text-yellow-900 text-sm">{coaching}</p>
  </div>
)}

{/* Final feedback */}
{feedback && (
  <div className="bg-green-50 border border-green-200 rounded-lg p-4 mb-4">
    <h4 className="font-semibold text-green-800 mb-2">📊 Session Results</h
    {feedback.score && (
      <div className="text-3xl font-bold text-green-600 mb-2">
        Score: {feedback.score}/100
      </div>
    )}
    {feedback.strengths && (
      <div className="mb-2">
        <strong>Strengths:</strong>
        <ul className="list-disc ml-5 text-sm">
          {feedback.strengths.map((s, i) => <li key={i}>{s}</li>)}
        </ul>
      </div>
    )}
    {feedback.improvements && (
      <div className="mb-2">
        <strong>Areas to Improve:</strong>
        <ul className="list-disc ml-5 text-sm">
          {feedback.improvements.map((s, i) => <li key={i}>{s}</li>)}
        </ul>
      </div>
    )}
    {feedback.raw && (
      <p className="text-sm whitespace-pre-wrap">{feedback.raw}</p>
    )}
  </div>
)}

{/* Input */}
{!sessionEnded && (
  <form onSubmit={sendMessage} className="flex gap-2">
    <input
      type="text"
      value={input}
```

```
            onChange={(e) => setInput(e.target.value)}
            placeholder="Type your sales pitch or response..."
            className="flex-1 border rounded-lg px-4 py-2"
            disabled={loading}
          />
          <button
            type="submit"
            disabled={loading || !input.trim()}
            className="bg-blue-600 text-white px-6 py-2 rounded-lg hover:bg-blue-
          >
            Send
          </button>
        </form>
      )}
    </div>
  );
}
```

Part 4: package.json Dependencies

File: `package.json`

```
{
  "name": "pcbancard-coaching-app",
  "version": "1.0.0",
  "scripts": {
    "start": "node server/index.js",
    "dev": "nodemon server/index.js",
    "client": "cd client && npm start",
    "build": "cd client && npm run build"
  },
  "dependencies": {
    "@anthropic-ai/sdk": "^0.24.0",
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "multer": "^1.4.5-lts.1",
    "openai": "^4.47.0",
    "pdf-parse": "^1.1.1",
    "pg": "^8.11.5"
  },
  "devDependencies": {
    "nodemon": "^3.1.0"
  }
}
```

Part 5: Environment Variables

File: `.env` (in Replit Secrets)

```
DATABASE_URL=postgresql://user:password@host:5432/database
ANTHROPIC_API_KEY=sk-ant-...
OPENAI_API_KEY=sk-...
```

Part 6: Starter Knowledge Content

File: `seed-knowledge.json`

```
{
  "items": [
    {
      "title": "Objection: I'm happy with my current processor",
      "content": "When a merchant says they're happy with their current processor
      "content_type": "objection",
      "category": "objection_handling",
      "tags": ["objection", "happy customer", "statement analysis"]
    },
    {
      "title": "Objection: I'm locked into a contract",
      "content": "Contracts are rarely as binding as merchants think. Response: '
      "content_type": "objection",
      "category": "objection_handling",
      "tags": ["objection", "contract", "buyout"]
    },
    {
      "title": "PayLo Dual Pricing Pitch",
      "content": "PayLo is our dual pricing program that eliminates your credit c
      "content_type": "product_info",
      "category": "pricing",
      "tags": ["PayLo", "dual pricing", "cash discount", "zero cost"]
    },
    {
      "title": "Cold Call Opening Script",
      "content": "Hi, this is [Name] with [Company]. I help local business owners
      "content_type": "script",
      "category": "cold_calling",
      "tags": ["cold call", "opening", "script", "door-to-door"]
    },
    {
      "title": "PCI Compliance Talking Points",
      "content": "PCI compliance is a set of security standards that all business
      "content_type": "compliance",
      "category": "compliance",
      "tags": ["PCI", "security", "compliance", "HIPAA", "encryption"]
    },
    {
      "title": "Closing: The Assumptive Close",
      "content": "Once you've addressed objections and the merchant seems interes
      "content_type": "best_practice",
      "category": "closing",
      "tags": ["closing", "assumptive close", "sales technique"]
    }
  ]
}
```

To seed this data, run:

```
curl -X POST http://localhost:3001/api/content/bulk \
  -H "Content-Type: application/json" \
  -d @seed-knowledge.json
```

Part 7: Deployment Instructions

Replit Setup

1. **Create new Repl** → Node.js template
2. **Enable PostgreSQL**:
   - Go to Tools → Database → PostgreSQL
   - Copy the connection string to Secrets as `DATABASE_URL`
3. **Add Secrets**:
   - `DATABASE_URL` - from step 2
   - `ANTHROPIC_API_KEY` - your API key

    - `OPENAI_API_KEY` - for embeddings

4. **Run schema**:
    - Open Shell tab
    - `psql $DATABASE_URL -f database/schema.sql`
5. **Deploy**:
    - Click Deploy → Autoscale
    - Set minimum instances to 1
    - Configure domain

For 20+ Concurrent Users

Autoscale Settings:

- Machine: 0.5 vCPU / 1GB RAM (sufficient for API relay)
- Min instances: 1
- Max instances: 5
- Target requests/instance: 100

Cost Estimate:

- ~$10-25/month for hosting
- ~$50-100/month API costs (Claude + OpenAI embeddings)
- Scales with usage

Quick Start Checklist

- Create Replit project
- Add PostgreSQL database
- Run schema.sql
- Add API keys to Secrets
- Install dependencies (`npm install`)
- Seed initial knowledge content

- Test locally (`npm run dev`)
- Deploy to Autoscale
- Upload your objection handling scripts
- Create custom scenarios for your team

Adding Content Over Time

The app gets smarter as you add:

1. **Call recordings transcripts** → Upload as PDFs or paste text
2. **Successful pitch scripts** → Tag as “best_practice”
3. **Product updates** → Add to “product_info” category
4. **New objections you encounter** → Document the objection AND the best response
5. **Compliance updates** → Keep the team current on regulations

Each piece of content is automatically chunked, embedded, and made searchable. The AI will pull relevant context during role-plays to provide better coaching.