AI Test Case Generation Platform - Complete PRD & Implementation Guide

Project Overview

An enterprise AI-powered test case generation platform that automatically creates comprehensive test suites from business documents (contracts, handbooks, tax filings, etc.) using RAG (Retrieval-Augmented Generation) technology. The platform integrates with internal tools to import customer configurations and generates categorized test cases with export capabilities.

Timeline: 2 Days MVP
Tech Stack: React, Node.js, Supabase, OpenAI API, RAG with Vector Search

---

User Journey & Workflow

Step-by-Step Process Flow

1. **Authentication** → Admin login only
2. **Internal Tools Configuration** → Configure dummy integrations (Salesforce, SAP, HR Portal)
3. **Customer Configuration** → Import customers using solution_id from internal tools
4. **Document Ingestion** → Upload/paste documents (PDF/TXT/MD) for configured customers
5. **AI Processing** → RAG-enhanced test case generation with context
6. **Test Case Management** → Review, categorize, and add additional test cases
7. **Review & Export** → Dashboard with stats and export options (ZIP, email)
8. **Execution Tracking** → Mark tests as ready/complete with simple reporting

---

Technical Architecture

Database Schema (Supabase PostgreSQL + pgvector)

```sql
```

```sql
-- Enable vector extension for RAG
CREATE EXTENSION IF NOT EXISTS vector;

-- Internal tools configuration
CREATE TABLE internal_tools (
  id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  name text NOT NULL,
  tool_type text NOT NULL, -- 'crm', 'erp', 'custom'
  api_endpoint text,
  auth_type text, -- 'api_key', 'oauth', 'basic_auth'
  config_fields jsonb, -- Dynamic configuration fields
  is_active boolean DEFAULT true,
  created_at timestamp DEFAULT now()
);

-- Customer management with solution_id
CREATE TABLE customers (
  id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  name text NOT NULL,
  solution_id text UNIQUE NOT NULL, -- Unique across all internal tools
  industry text,
  internal_tool_id uuid REFERENCES internal_tools(id),
  is_configured boolean DEFAULT false,
  tool_config jsonb, -- Store tool-specific config data
  last_sync timestamp,
  status text DEFAULT 'active',
  created_at timestamp DEFAULT now()
);

-- Document storage
CREATE TABLE documents (
  id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  customer_id uuid REFERENCES customers(id),
  filename text NOT NULL,
  content text,
  doc_type text, -- 'Contract', 'Handbook', 'Tax Filing', etc.
  status text DEFAULT 'uploaded',
  file_size integer,
  created_at timestamp DEFAULT now()
);

-- RAG: Document chunks with embeddings
CREATE TABLE document_chunks (
  id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  document_id uuid REFERENCES documents(id),
  chunk_text text NOT NULL,
  chunk_index integer,
  embedding vector(1536), -- OpenAI embeddings dimension
  metadata jsonb,
  created_at timestamp DEFAULT now()
);

-- Generated test cases
CREATE TABLE test_cases (
  id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  document_id uuid REFERENCES documents(id),
  content text NOT NULL,
  category text, -- 'Functional', 'Compliance', 'Edge Cases', 'Integration'
  source text DEFAULT 'generated', -- 'generated', 'uploaded', 'manual'
  confidence_score float,
  context_used text, -- RAG context that was used
  created_at timestamp DEFAULT now()
);
```

```sql
-- Create indexes for performance
CREATE INDEX ON document_chunks USING ivfflat (embedding vector_cosine_ops);
CREATE INDEX ON customers (solution_id);
CREATE INDEX ON documents (customer_id);
CREATE INDEX ON test_cases (document_id);
```

RAG Search Function

```
sql
```

```sql
-- Vector similarity search for RAG
CREATE OR REPLACE FUNCTION search_document_chunks(
  query_embedding vector(1536),
  doc_id uuid,
  similarity_threshold float = 0.7,
  match_count int = 5
)
RETURNS TABLE (
  id uuid,
  chunk_text text,
  similarity float,
  metadata jsonb
)
LANGUAGE sql
AS $$
  SELECT
    document_chunks.id,
    chunk_text,
    1 - (embedding <=> query_embedding) AS similarity,
    metadata
  FROM document_chunks
  WHERE
    document_chunks.document_id = doc_id
    AND 1 - (embedding <=> query_embedding) > similarity_threshold
  ORDER BY embedding <=> query_embedding
  LIMIT match_count;
$$;
```

Tech Stack & Dependencies

Frontend (React)

```bash
npm install react react-dom react-router-dom
npm install @supabase/supabase-js
npm install axios react-query zustand
npm install tailwindcss lucide-react
npm install react-hot-toast react-hook-form
npm install recharts # For dashboard charts
npm install pdf-parse mammoth # Document processing
```

Backend Integration

- **Database**: Supabase (PostgreSQL + Vector extension)
- **AI**: OpenAI API (GPT-4 + Embeddings)
- **File Storage**: Supabase Storage
- **Authentication**: Supabase Auth

Environment Variables

```bash
SUPABASE_URL=https://your-project.supabase.co
SUPABASE_ANON_KEY=your-anon-key
OPENAI_API_KEY=your-openai-key
```

Mock Data & Dummy Integrations

Internal Tools Mock Data

javascript

```
export const mockInternalTools = [
 {
  id: 'salesforce-crm',
  name: 'Salesforce CRM',
  tool_type: 'crm',
  api_endpoint: 'https://mock-sf-api.com/v1',
  auth_type: 'oauth',
  config_fields: {
   client_id: 'mock_sf_client_id',
   client_secret: 'mock_sf_secret',
   instance_url: 'https://company.salesforce.com'
  },
  status: 'connected',
  customers: [
   {
    solution_id: 'SF_001_TECH',
    name: 'TechCorp Inc',
    industry: 'Technology',
    contract_count: 15,
    last_sync: '2024-09-10T10:00:00Z'
   },
   {
    solution_id: 'SF_002_FIN',
    name: 'FinanceGroup LLC',
    industry: 'Financial Services',
    contract_count: 8,
    last_sync: '2024-09-10T09:30:00Z'
   }
  ]
 },
 {
  id: 'sap-erp',
    name: 'SAP Enterprise',
    tool_type: 'erp',
    api_endpoint: 'https://mock-sap-api.com/v2',
    auth_type: 'api_key',
    config_fields: {
      api_key: 'mock_sap_key_12345',
      environment: 'production',
      region: 'us-east-1'
    },
    status: 'connected',
    customers: [
      {
        solution_id: 'SAP_HC_001',
        name: 'HealthCare Partners',
        industry: 'Healthcare',
        contract_count: 12,
        last_sync: '2024-09-10T11:15:00Z'
      }
    ]
  },
  {
    id: 'custom-portal',
    name: 'Internal HR Portal',
    tool_type: 'custom',
    api_endpoint: 'https://mock-hr-portal.com/api',
    auth_type: 'basic_auth',
    config_fields: {
      username: 'admin_user',
      password: '****',
      department_code: 'HR_001'
    },
    status: 'connected',
    customers: [
      {
        solution_id: 'HR_EDU_001',
        name: 'Education Institute',
        industry: 'Education',
        contract_count: 6,
        last_sync: '2024-09-10T08:45:00Z'
      }
    ]
  }
];
```

Sample Document Templates

```
javascript
```

```
export const mockDocuments = {
  'SF_001_TECH': [
    {
      name: 'Software License Agreement 2024.pdf',
      type: 'Contract',
      size: '2.3 MB',
      content: 'Software licensing terms and conditions for enterprise deployment. Maximum 500 concurrent users
    },
    {
      name: 'Employee Handbook Q3.md',
      type: 'Handbook',
      size: '1.1 MB',
      content: 'Company policies, procedures, and guidelines for Q3 2024. Remote work policy allows up to 3 days
    }
  ],
  'SAP_HC_001': [
    {
      name: 'HIPAA Compliance Manual.pdf',
      type: 'Handbook',
      size: '4.2 MB',
      content: 'Healthcare data protection guidelines. Only authorized healthcare providers can access patient reco
    }
  ]
};
```

RAG Implementation

Document Processing Pipeline

```
javascript
```

```
// utils/ragUtils.js
import OpenAI from 'openai';

const openai = new OpenAI({
  apiKey: process.env.OPENAI_API_KEY
});

// 1. Document chunking
const chunkDocument = (content, maxWords = 500) => {
  const sentences = content.split(/[.!?]+/);
  const chunks = [];
  let currentChunk = '';
  let wordCount = 0;

  for (const sentence of sentences) {
    const sentenceWords = sentence.trim().split(' ').length;

    if (wordCount + sentenceWords > maxWords && currentChunk) {
      chunks.push({
        text: currentChunk.trim(),
        wordCount: wordCount
      });
      currentChunk = sentence.trim();
      wordCount = sentenceWords;
    } else {
      currentChunk += (currentChunk ? '. ' : '') + sentence.trim();
      wordCount += sentenceWords;
    }
  }

  if (currentChunk) {
    chunks.push({
      text: currentChunk.trim(),
      wordCount: wordCount
    });
  }

  return chunks;
};

// 2. Generate embeddings
const generateEmbedding = async (text) => {
  const response = await openai.embeddings.create({
    model: "text-embedding-ada-002",
    input: text,
  });
  return response.data[0].embedding;
};

// 3. Process document for RAG
export const processDocumentForRAG = async (documentId, content) => {
  const chunks = chunkDocument(content, 500);

  const chunksWithEmbeddings = await Promise.all(
    chunks.map(async (chunk, index) => {
      const embedding = await generateEmbedding(chunk.text);
      return {
        document_id: documentId,
        chunk_text: chunk.text,
        chunk_index: index,
        embedding: embedding,
        metadata: {
          word_count: chunk.text.split(' ').length,
          section: chunk.section || 'general'
        }
      };
    })
  );

  const { data, error } = await supabase
    .from('document_chunks')
    .insert(chunksWithEmbeddings);

  return data;
};

// 4. RAG search
export const searchSimilarContext = async (query, documentId, limit = 5) => {
  const queryEmbedding = await generateEmbedding(query);

  const { data: similarChunks, error } = await supabase
    .rpc('search_document_chunks', {
      query_embedding: queryEmbedding,
      doc_id: documentId,
      similarity_threshold: 0.7,
      match_count: limit
    });

  return similarChunks;
};
```

AI Test Case Generation with RAG

```
javascript
```

```
// Generate test cases using RAG context
export const generateTestCasesWithRAG = async (documentId) => {
 const { data: document } = await supabase
  .from('documents')
  .select('*')
  .eq('id', documentId)
  .single();

 const testCategories = [
  {
   category: 'Functional Tests',
   query: 'functional requirements, user actions, system behavior, workflow processes'
  },
  {
   category: 'Compliance Tests',
   query: 'compliance requirements, regulatory standards, legal obligations, audit requirements'
  },
  {
   category: 'Edge Cases',
   query: 'edge cases, error conditions, boundary conditions, exceptional scenarios'
  },
  {
   category: 'Integration Tests',
   query: 'system integration, data flow, external dependencies, API interactions'
  }
 ];

 const allTestCases = [];

 for (const category of testCategories) {
  // Get relevant context using RAG
  const context = await searchSimilarContext(category.query, documentId, 3);
  const contextText = context.map(c => c.chunk_text).join('\n\n');

  const prompt = `
Based on the following document context, generate 3-5 comprehensive test cases for ${category.category}.

Document Type: ${document.doc_type}
Context:
${contextText}

For each test case, provide:
1. Test Name
2. Description
3. Test Steps (numbered)
4. Expected Result
5. Priority (High/Medium/Low)

Format as JSON array with these fields: name, description, steps, expected_result, priority, category.
`;

  const response = await openai.chat.completions.create({
    model: "gpt-4",
    messages: [{ role: "user", content: prompt }],
    temperature: 0.7,
    max_tokens: 2000
  });

  const generatedTests = JSON.parse(response.choices[0].message.content);

  const testsWithContext = generatedTests.map(test => ({
    ...test,
    document_id: documentId,
    category: category.category,
      source: 'rag_generated',
      context_used: contextText,
      confidence_score: 0.85
    }));

    allTestCases.push(...testsWithContext);
  }

  // Save to database
  const { data, error } = await supabase
    .from('test_cases')
    .insert(allTestCases);

  return allTestCases;
};
```

Project Structure

```
src/
├── components/
│   ├── Auth/
│   │   ├── Login.jsx
│   │   └── Register.jsx
│   ├── Tools/
│   │   ├── ToolsList.jsx
│   │   ├── ToolConfigForm.jsx
│   │   ├── MockIntegration.jsx
│   │   └── ConnectionStatus.jsx
│   ├── Customers/
│   │   ├── CustomerCard.jsx
│   │   ├── ImportFromTool.jsx
│   │   ├── SolutionIdValidator.jsx
│   │   └── ConfigurationStatus.jsx
│   ├── Documents/
│   │   ├── CustomerSelector.jsx
│   │   ├── FileUpload.jsx
│   │   ├── DocumentList.jsx
│   │   ├── RAGProcessing.jsx
│   │   └── ChunkViewer.jsx
│   ├── TestCases/
│   │   ├── TestCaseCard.jsx
│   │   ├── RAGTestCase.jsx
│   │   ├── CategoryFilter.jsx
│   │   ├── ContextPreview.jsx
│   │   ├── ConfidenceScore.jsx
│   │   └── FileImport.jsx
│   ├── Dashboard/
│   │   ├── StatsCard.jsx
│   │   ├── ProgressBar.jsx
│   │   ├── RAGStats.jsx
│   │   └── ProcessingStatus.jsx
│   └── Layout/
│      ├── Navbar.jsx
│      └── Sidebar.jsx
├── pages/
│   ├── Auth.jsx
│   ├── InternalTools.jsx
│   ├── Customers.jsx
│   ├── Documents.jsx
│   ├── Generation.jsx
│   ├── TestCases.jsx
│   ├── Review.jsx
│   └── Execution.jsx
├── hooks/
│   ├── useAuth.js
│   ├── useTools.js
│   ├── useCustomers.js
│   ├── useRAG.js
│   ├── useMockTools.js
│   └── useApi.js
├── utils/
│   ├── supabase.js
│   ├── ragUtils.js
│   ├── mockToolsData.js
│   ├── toolIntegrations.js
│   ├── testGeneration.js
│   ├── exportUtils.js
│   └── validators.js
└── App.jsx
```

API Endpoints (Backend Routes)

```javascript
```

```
// Authentication
POST /auth/login
POST /auth/register

// Internal Tools Management
GET  /internal-tools
POST /internal-tools
PUT  /internal-tools/:id
POST /internal-tools/:id/test-connection
GET  /internal-tools/:id/customers

// Customer Management
GET  /customers
POST /customers
POST /customers/import-from-tool
GET  /customers/by-solution-id/:solutionId
PUT  /customers/:id/configure

// Document Processing
GET  /documents/:customerId
POST /documents/upload
POST /documents/text
POST /documents/:id/process-rag

// AI & RAG
POST /generate/:documentId
POST /rag/search
GET  /rag/chunks/:documentId

// Test Cases
GET  /test-cases/:documentId
POST /test-cases/upload
```

```
PUT  /test-cases/:id

// Export & Reporting
GET  /export/:documentId
POST /export/email
GET  /stats/dashboard

// Execution
POST /execute/:testCaseId
GET  /execution/status
```

2-Day Implementation Timeline

DAY 1: Foundation & Core Setup (8 hours)

Morning (4 hours)

- **Hour 1**: Project setup (Replit + dependencies)
- **Hour 2**: Supabase database schema creation
- **Hour 3**: Authentication & basic navigation
- **Hour 4**: Internal tools configuration UI

Afternoon (4 hours)

- **Hour 5**: Mock tool integration logic
- **Hour 6**: Customer import from tools
- **Hour 7**: Document upload functionality
- **Hour 8**: Basic RAG setup (chunking + embeddings)

DAY 2: AI Generation & Polish (8 hours)

Morning (4 hours)

- **Hour 1**: RAG search implementation
- **Hour 2**: AI test case generation with context
- **Hour 3**: Test case management UI
- **Hour 4**: Category filtering & display

Afternoon (4 hours)

- **Hour 5**: Review dashboard with stats
- **Hour 6**: Export functionality (ZIP + email)
- **Hour 7**: Execution tracking interface
- **Hour 8**: Final polish, bug fixes, demo preparation

Demo Scenarios

Scenario 1: Software License Agreement

- **Customer**: TechCorp Inc (SF_001_TECH)
- **Document**: Software License Agreement 2024.pdf
- **Generated Tests**: License usage limits, renewal conditions, compliance checks
- **RAG Context**: "Maximum 500 concurrent users as specified in section 3.2..."

Scenario 2: Healthcare Compliance

- **Customer**: HealthCare Partners (SAP_HC_001)

- **Document**: HIPAA Compliance Manual.pdf
- **Generated Tests**: Patient data access, retention policies, audit trails
- **RAG Context**: "Only authorized healthcare providers can access patient records..."

Key Features Checklist

Core Functionality

- [ ] Admin authentication (Supabase Auth)
- [ ] Internal tools configuration (3 mock tools)
- [ ] Customer import with solution_id
- [ ] Document upload (PDF/TXT/MD support)
- [ ] RAG processing (chunking + embeddings)
- [ ] AI test case generation with context
- [ ] Test case categorization (4 categories)
- [ ] Additional test case file upload
- [ ] Export functionality (ZIP format)
- [ ] Execution status tracking

Advanced Features

- [ ] Real-time progress indicators
- [ ] Vector similarity search
- [ ] Confidence scoring
- [ ] Context preview in UI
- [ ] Dashboard statistics
- [ ] Email notifications
- [ ] Responsive design

- [ ] Error handling & validation

Deployment Instructions

Supabase Setup

1. Create new Supabase project
2. Run SQL schema in SQL Editor
3. Enable Row Level Security (if needed)
4. Get project URL and anon key

Replit Deployment

1. Create React project from template
2. Install dependencies via Shell
3. Add environment variables as Secrets
4. Configure Supabase client
5. Test database connection
6. Deploy using Replit hosting

Environment Configuration

```bash
# Add these as Replit Secrets
SUPABASE_URL=https://your-project.supabase.co
SUPABASE_ANON_KEY=your-anon-key
OPENAI_API_KEY=your-openai-key
```

Success Metrics

MVP Success Criteria

- [ ] End-to-end workflow completion (8 steps)
- [ ] RAG-enhanced test case generation working
- [ ] 3+ document types supported
- [ ] Export functionality operational
- [ ] Demo-ready with realistic data

Technical Metrics

- [ ] <3 second document processing
- [ ] >0.7 RAG similarity threshold
- [ ] 15+ test cases per document
- [ ] 4 categorization types working
- [ ] Error rate <5%

Post-MVP Enhancements

Phase 2 Features

- Real internal tool integrations (APIs)
- Advanced test case validation
- Automated test execution
- Team collaboration features
- Advanced analytics dashboard

Phase 3 Features

- Multi-tenant architecture
- Custom AI model fine-tuning
- Workflow automation
- Integration marketplace
- Enterprise SSO

Team Responsibilities

Frontend Developer

- React components & UI/UX
- Tailwind CSS styling
- State management (Zustand)
- File upload & document display

Backend/Integration Developer

- Supabase database setup
- RAG implementation
- OpenAI API integration
- Mock tool integrations

Full-Stack Developer

- End-to-end feature implementation
- API endpoint creation

- Authentication setup
- Testing & debugging

Important Notes

Development Constraints

- **2-day timeline** - Focus on core MVP features
- **Dummy integrations** - No real API connections needed
- **Single OpenAI model** - No model selection UI
- **Basic UI** - Functional over fancy design

Technical Considerations

- Supabase vector extension required for RAG
- OpenAI API rate limits consideration
- File size limits for document upload
- Replit storage limitations

This document contains all specifications needed to build the AI Test Case Generation Platform MVP within 2 days. Share with your development team for implementation.