# Project 6: Image-Generating Chatbot - Technical Implementation Guide

## Project Setup & Dependencies

### Key Dependencies

```json
// Backend additions
{
  "openai": "^4.20.1",
  "sharp": "^0.32.6",       // Image processing
  "axios": "^1.5.0"         // HTTP requests
}

// Frontend additions
{
  "@mui/icons-material": "^5.14.3",
  "react-query": "^3.39.3"
}
```

### Environment Variables

```shell
# Add to existing .env
OPENAI_API_KEY=your_dalle_api_key_here
MAX_IMAGE_GENERATIONS_PER_HOUR=20
IMAGE_QUALITY_DEFAULT=standard
```

## Database Schema Implementation

### New Tables

SQL

```sql
-- filepath: backend/migrations/004-create-generated-images.sql
CREATE TABLE generated_images (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) NOT NULL,
    thread_id INTEGER REFERENCES chat_threads(id),
    message_id INTEGER REFERENCES messages(id),
    original_prompt TEXT NOT NULL,
    revised_prompt TEXT,
    image_data TEXT NOT NULL,  -- Base64 encoded
    thumbnail_data TEXT,       -- Base64 encoded thumbnail
    metadata JSONB DEFAULT '{}',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE INDEX idx_images_user_id ON generated_images(user_id);
CREATE INDEX idx_images_thread_id ON generated_images(thread_id);
```

## Direct Image Generation Service

## Core Service (Simplified from Complex Version)

JavaScript

```javascript
// filepath: backend/src/services/imageGeneration.js
const OpenAI = require('openai');
const axios = require('axios');
const sharp = require('sharp');

class SimpleImageGenerator {
  constructor() {
```

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

async generateImage(prompt, userId, threadId = null) {
  try {
    // Direct DALL-E API call (no queuing)
    const response = await this.openai.images.generate({
      model: 'dall-e-3',
      prompt: this.enhancePrompt(prompt),
      size: '1024x1024',
      quality: 'standard',
      n: 1
    });

    // Download and process image immediately
    const imageUrl = response.data[0].url;
    const imageBuffer = await this.downloadImage(imageUrl);
    const thumbnailBuffer = await this.createThumbnail(imageBuffer);

    // Store in database immediately
    const imageRecord = await this.storeImage({
      userId,
      threadId,
      originalPrompt: prompt,
      revisedPrompt: response.data[0].revised_prompt,
      imageData: imageBuffer.toString('base64'),
      thumbnailData: thumbnailBuffer.toString('base64')
    });

    return {
      id: imageRecord.id,
      imageData: `data:image/png;base64,${imageBuffer.toString('base64')}`,
      originalPrompt: prompt,
      revisedPrompt: response.data[0].revised_prompt
    };
  } catch (error) {
    throw new Error(`Image generation failed: ${error.message}`);
  }
}

enhancePrompt(prompt) {
  // Simple prompt enhancement
  return prompt.length < 50 ? `${prompt}, high quality, detailed` : prompt;
}

async downloadImage(url) {
  const response = await axios.get(url, { responseType: 'arraybuffer' });
  return Buffer.from(response.data);
}

async createThumbnail(imageBuffer) {
  return await sharp(imageBuffer).resize(256, 256).jpeg({ quality: 80 }).toBuffer();
}

async storeImage(data) {
  // Database storage implementation
  return await db.query(
    `INSERT INTO generated_images (user_id, thread_id, original_prompt,
     revised_prompt, image_data, thumbnail_data)
     VALUES ($1, $2, $3, $4, $5, $6) RETURNING id`,
      [data.userId, data.threadId, data.originalPrompt, data.revisedPrompt,
       data.imageData, data.thumbnailData]
    );
  }
}

module.exports = SimpleImageGenerator;
```

## API Routes (Simple Pattern)

JavaScript

```javascript
// filepath: backend/src/routes/images.js
const express = require('express');
const rateLimit = require('express-rate-limit');
const SimpleImageGenerator = require('../services/imageGeneration');

const router = express.Router();
const imageGenerator = new SimpleImageGenerator();

// Rate limiting
const imageLimit = rateLimit({
  windowMs: 60 * 60 * 1000, // 1 hour
  max: 20, // 20 images per hour
  message: 'Too many image generations, try again later'
});

// Generate image endpoint
router.post('/generate', imageLimit, async (req, res) => {
  try {
    const { prompt, threadId } = req.body;
    const result = await imageGenerator.generateImage(prompt, req.user.id, threadId);
    res.json({ success: true, image: result });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});

// Serve image
router.get('/:imageId', async (req, res) => {
  try {
    const image = await db.query(
      'SELECT image_data FROM generated_images WHERE id = $1 AND user_id = $2',
      [req.params.imageId, req.user.id]
    );

    if (!image.rows[0]) return res.status(404).json({ error: 'Image not found' });

    const imageBuffer = Buffer.from(image.rows[0].image_data, 'base64');
    res.set('Content-Type', 'image/png');
    res.send(imageBuffer);
  } catch (error) {
    res.status(500).json({ error: 'Failed to serve image' });
  }
});

module.exports = router;
```

## Frontend Implementation

### Image Generator Component (Simplified)

```
None
    // filepath: frontend/src/components/Images/ImageGenerator.jsx
    import React, { useState } from 'react';
    import { Button, TextField, CircularProgress, Alert, Card } from '@mui/material';

    const ImageGenerator = ({ threadId, onImageGenerated }) => {
      const [prompt, setPrompt] = useState('');
      const [isGenerating, setIsGenerating] = useState(false);
      const [error, setError] = useState(null);

      const handleGenerate = async () => {
        if (!prompt.trim()) return;

        setIsGenerating(true);
        setError(null);

        try {
          const response = await fetch('/api/images/generate', {
            method: 'POST',
            headers: {
              'Content-Type': 'application/json',
              'Authorization': `Bearer ${localStorage.getItem('token')}`
            },
            body: JSON.stringify({ prompt: prompt.trim(), threadId })
          });

          if (!response.ok) throw new Error('Generation failed');
      const result = await response.json();
      onImageGenerated?.(result.image);
      setPrompt(''); // Clear prompt after success
    } catch (err) {
      setError(err.message);
    } finally {
      setIsGenerating(false);
    }
  };

  return (
    <Card sx={{ p: 2 }}>
      <TextField
        fullWidth
        multiline
        rows={3}
        label="Describe the image you want to create"
        value={prompt}
        onChange={(e) => setPrompt(e.target.value)}
        disabled={isGenerating}
        sx={{ mb: 2 }}
      />

      <Button
        variant="contained"
        onClick={handleGenerate}
        disabled={isGenerating || !prompt.trim()}
        startIcon={isGenerating ? <CircularProgress size={20} /> : null}
        fullWidth
        >
          {isGenerating ? 'Generating...' : 'Generate Image'}
        </Button>

        {error && <Alert severity="error" sx={{ mt: 2 }}>{error}</Alert>}
      </Card>
    );
  };

  export default ImageGenerator;
```

## Chat Integration

### Enhanced Chat Component (Key Changes Only)

None
```
// filepath: frontend/src/components/Chat/ChatInterface.jsx
import React, { useState } from 'react';
import ImageGenerator from '../Images/ImageGenerator';
// ...existing imports...

const ChatInterface = ({ threadId }) => {
  // ...existing code...
  const [showImageGenerator, setShowImageGenerator] = useState(false);

  const handleImageGenerated = (image) => {
    // Add image message to chat
    const imageMessage = {
      id: Date.now(),
    content: `Generated: ${image.originalPrompt}`,
    role: 'assistant',
    messageType: 'image',
    metadata: { imageData: image.imageData, imageId: image.id },
    createdAt: new Date().toISOString()
  };
  setMessages(prev => [...prev, imageMessage]);
  setShowImageGenerator(false);
};

const renderMessage = (message) => {
  // ...existing code...

  if (message.messageType === 'image') {
    return (
      <div className="image-message">
        <img
          src={message.metadata.imageData}
          alt={message.content}
          style={{ maxWidth: '100%', borderRadius: '8px' }}
        />
        <p>{message.content}</p>
      </div>
    );
  }

  // ...existing message rendering...
};

return (
    <div>
      {/* ...existing chat UI... */}

      {/* Image Generator Toggle */}
      <button onClick={() => setShowImageGenerator(!showImageGenerator)}>
        🎨 Generate Image
      </button>

      {showImageGenerator && (
        <ImageGenerator
          threadId={threadId}
          onImageGenerated={handleImageGenerated}
        />
      )}
    </div>
  );
};

export default ChatInterface;
```