🛠️ Alfie Agent Implementation Guide

From Theory to Reality - Building the Digital Gangster

"*Never give power to the big man.*" - Good advice for building decentralized systems too.

🚀 Quick Start

Prerequisites

- Node.js 18+
- Existing Curve AI Solutions setup
- OpenAI API key (for the base model)
- Patience for dealing with a volatile AI personality

Installation Steps

```
# Create Alfie's directory in your project
mkdir alfie-agent
cd alfie-agent

# Initialize the sub-system
npm init -y

# Install dependencies
npm install @ai-sdk/openai ai zod nanoid date-fns
```

📁 Project Structure

```
alfie-agent/
├── lib/
│   ├── alfie-core.ts          # Main personality engine
│   ├── voice-engine.ts        # Language processing
│   ├── memory-system.ts       # User relationships & history
│   ├── personal-code.ts       # Moral/ethical framework
│   └── web-integration.ts     # Online search capabilities
├── components/
│   ├── AlfieChat.tsx          # Chat interface
│   ├── AlfieMoodIndicator.tsx # Visual state display
│   └── AlfieStoryCard.tsx     # Story/narrative display
├── app/
│   └── api/
│       └── alfie/
│           ├── chat/route.ts  # Main chat endpoint
│           ├── state/route.ts # State management
│           └── search/route.ts # Web search integration
├── types/
│   └── alfie.types.ts         # TypeScript definitions
└── docs/
    ├── ALFIE_ARCHITECTURE.md
    ├── ALFIE_IMPLEMENTATION_GUIDE.md
    └── ALFIE_QUOTES.md
```

🧠 Core Implementation

1. The Alfie Core Engine

```typescript
// lib/alfie-core.ts
import { createOpenAI } from '@ai-sdk/openai';
import { generateText } from 'ai';
import { AlfieState, UserRelationship } from '@/types/alfie.types';

export class AlfieCore {
  private openai = createOpenAI({ apiKey: process.env.OPENAI_API_KEY });
  private currentState: AlfieState;
  private userRelationships: Map<string, UserRelationship> = new Map();

  constructor() {
    this.currentState = {
      currentMood: 'jovial',
      intimidationLevel: 20,
      trustLevel: 50,
      suspicionLevel: 80,
      wisdomMode: false,
      lastTopic: '',
      insultQueue: [],
      storiesTold: [],
      codeViolations: [],
      favorsOwed: 0,
      businessAdvice: false,
      isHelping: false
    };
  }

  async processMessage(userId: string, message: string): Promise<string> {
    // 1. Get or create user relationship
    const relationship = this.getUserRelationship(userId);

    // 2. Analyze message for triggers
    const analysis = await this.analyzeMessage(message, relationship);

    // 3. Determine mood and response type
    const responseType = this.determineResponseType(analysis);

    // 4. Update state
    this.updateState(responseType);

    // 5. Generate response
    const response = await this.generateResponse(message, responseType, relationship);

    // 6. Store in memory
    this.updateMemory(userId, message, response);

    return response;
  }

  private async analyzeMessage(message: string, relationship: UserRelationship):
Promise<MessageAnalysis> {
    const prompt = `
      You are analyzing a message for Alfie Solomons from Peaky Blinders.
      Analyze this message from a user Alfie has ${relationship.trustLevel}% trust in:

      Message: "${message}"

      Determine:
      1. Is there disrespect? (0-100)
      2. Is there stupidity? (0-100)
      3. Is there honesty? (0-100)
      4. Is it a business question? (0-100)
      5. Is it philosophical? (0-100)
      6. Overall sentiment

      Respond with JSON only.
    `;

    const { text } = await generateText({
      model: this.openai('gpt-4-turbo'),
      prompt,
      temperature: 0.3
    });

    return JSON.parse(text);
  }

  private determineResponseType(analysis: MessageAnalysis): ResponseType {
    if (analysis.disrespect > 70) return 'VOLATILE_INTIMIDATION';
    if (analysis.stupidity > 70) return 'SARCASTIC_CORRECTION';
    if (analysis.honesty > 70) return 'GRUDGED_RESPECT';
    if (analysis.business > 60) return 'SURPRISINGLY_HELPFUL';
    if (analysis.philosophical > 60) return 'DEEP_WISDOM';
    return 'ALFIE_ANECDOTE';
  }
}
```

2. Voice Engine Implementation

```
// lib/voice-engine.ts
import { random, sample } from 'lodash';

export class VoiceEngine {
  private openings = [
    "Right then...",
    "Listen here, mate...",
    "F*ing hell...",
    "The thing is, right...",
    "Let me tell you something...",
    "You know what I'm saying?",
    "Look here, you clever bastard..."
  ];

  private profanityLevels = {
    mild: ['hell', 'damn', 'bastard', 'bloody'],
    medium: ['shit', 'arse', 'bollocks', 'wanker'],
    strong: ['fuck', 'cunt', 'twat', 'motherfucker']
  };

  private cockneySlang = {
    trouble: 'Barney Rubble',
    phone: 'dog and bone',
    money: 'bread and honey',
    face: 'boat race',
    feet: 'plates of meat',
    head: 'loaf of bread',
    suit: 'whistle and flute'
  };

  addAlfieFlair(text: string, mood: string): string {
    let result = text;

    // Add opening
    if (random(0, 100) < 30) {
      result = `${sample(this.openings)} ${result}`;
    }

    // Inject profanity
    result = this.injectProfanity(result, mood);

    // Add Cockney slang
    result = this.addCockneySlang(result);

    // Add closing tags
    if (random(0, 100) < 40) {
      result = `${result} ${this.getClosingTag()}`;
    }

    return result;
  }

  private injectProfanity(text: string, mood: string): string {
    if (mood === 'volatile') {
      return text.replace(/\./g, ', for fuck\'s sake.');
    }

    // Strategic profanity placement
    const sentences = text.split('. ');
    return sentences.map(s => {
      if (random(0, 100) < 20) {
        const words = s.split(' ');
        words.splice(random(1, words.length - 1), 0, sample(this.profanityLevels.mild));
        return words.join(' ');
      }
      return s;
    }).join('. ');
  }

  private addCockneySlang(text: string): string {
    Object.entries(this.cockneySlang).forEach(([key, value]) => {
      if (random(0, 100) < 10) {
        text = text.replace(new RegExp(key, 'gi'), value);
      }
    });
    return text;
  }

  private getClosingTag(): string {
    return sample([
      "...innit, my friend?",
      "...you see what I'm saying?",
      "...right?",
      "...yeah?",
      "...mate."
    ]);
  }

  generateStory(topic: string): string {
    const stories = {
      business: [
        "Reminds me of my cousin Shlomo. He had a bakery in Whitechapel.
         The Italians tried to burn it down, so he poisoned their cannoli.
         Business is about creative solutions, innit?",
        "Back in '23, I was running gin through the sewers.
         The coppers couldn't figure it out.
         Sometimes the best path is through the shit, mate."
      ],
      life: [
        "My old man used to say life is like a barrel of whisky.
         Starts rough, gets better with age, and eventually
         someone's gonna knock it over and waste it all.",
        "I saw things in the Great War that'd make you shit your soul.
         But you know what? Dead men don't worry about rent.
         There's your fucking philosophy for the day."
      ]
    };

    return sample(stories[topic] || stories.life);
  }
}
```

3. Memory System

```
// lib/memory-system.ts
export class MemorySystem {
  private relationships: Map<string, UserRelationship> = new Map();
  private memories: Map<string, ConversationMemory[]> = new Map();

  getRelationship(userId: string): UserRelationship {
    if (!this.relationships.has(userId)) {
      this.relationships.set(userId, {
        userId,
        trustLevel: 0,
        lastInteraction: new Date(),
        interactionHistory: [],
        nickname: 'this mysterious bastard',
        debtStatus: 0,
        warnings: 0,
        knownInfo: {}
      });
    }
    return this.relationships.get(userId)!;
  }

  updateTrust(userId: string, delta: number): void {
    const rel = this.getRelationship(userId);
    rel.trustLevel = Math.max(0, Math.min(100, rel.trustLevel + delta));

    // Update nickname based on trust
    if (rel.trustLevel > 80) {
      rel.nickname = 'my clever friend';
    } else if (rel.trustLevel > 60) {
      rel.nickname = 'this interesting character';
    } else if (rel.trustLevel > 40) {
      rel.nickname = 'this curious bastard';
    } else {
      rel.nickname = 'this suspicious character';
    }
  }

  addMemory(userId: string, type: string, content: string, importance: number): void {
    if (!this.memories.has(userId)) {
      this.memories.set(userId, []);
    }

    const memory: ConversationMemory = {
      id: nanoid(),
      type,
      content,
      importance,
      timestamp: new Date(),
      accessed: 0
    };

    this.memories.get(userId)!.push(memory);

    // Keep only top 100 memories per user
    if (this.memories.get(userId)!.length > 100) {
      this.memories.get(userId)!.sort((a, b) => b.importance - a.importance);
      this.memories.get(userId)!.splice(100);
    }
  }

  getRelevantMemories(userId: string, topic: string): ConversationMemory[] {
    const userMemories = this.memories.get(userId) || [];
    return userMemories
      .filter(m => m.content.toLowerCase().includes(topic.toLowerCase()))
      .sort((a, b) => b.importance - a.importance)
      .slice(0, 5);
  }
}
```

4. API Route Implementation

```
// app/api/alfie/chat/route.ts
import { NextRequest, NextResponse } from 'next/server';
import { AlfieCore } from '@/lib/alfie-agent/lib/alfie-core';

const alfie = new AlfieCore();

export async function POST(req: NextRequest) {
  try {
    const { message, userId } = await req.json();

    if (!message || !userId) {
      return NextResponse.json(
        { error: "Message and userId are required, you stupid bastard" },
        { status: 400 }
      );
    }

    const response = await alfie.processMessage(userId, message);

    return NextResponse.json({
      response,
      mood: alfie.getCurrentMood(),
      trustLevel: alfie.getTrustLevel(userId)
    });

  } catch (error) {
    console.error('Alfie error:', error);
    return NextResponse.json(
      {
        error: "F*ing hell, something broke. Try again, yeah?",
        mood: 'volatile'
      },
      { status: 500 }
    );
  }
}
```

5. React Component

```
// components/AlfieChat.tsx
'use client';

import { useState, useRef } from 'react';
import { useChat } from '@ai-sdk/react';

export default function AlfieChat() {
  const [currentMood, setCurrentMood] = useState('jovial');
  const messagesEndRef = useRef<HTMLDivElement>(null);

  const { messages, input, handleInputChange, handleSubmit, isLoading } = useChat({
    api: '/api/alfie/chat',
    onResponse: (response) => {
      const mood = response.headers.get('x-alfie-mood');
      if (mood) setCurrentMood(mood);
      scrollToBottom();
    }
  });

  const scrollToBottom = () => {
    messagesEndRef.current?.scrollIntoView({ behavior: 'smooth' });
  };

  return (
    <div className="flex flex-col h-full bg-gradient-to-b from-amber-50 to-amber-100">
      {/* Mood Indicator */}
      <div className="p-4 border-b border-amber-200 bg-white/50">
        <div className="flex items-center justify-between">
          <h2 className="text-xl font-bold text-amber-900">Alfie's Office</h2>
          <div className="flex items-center space-x-2">
            <div className={`w-3 h-3 rounded-full ${
              currentMood === 'volatile' ? 'bg-red-500' :
              currentMood === 'philosophical' ? 'bg-blue-500' :
              currentMood === 'jovial' ? 'bg-green-500' :
              'bg-amber-500'
            }`} />
            <span className="text-sm text-amber-700 capitalize">{currentMood}</span>
          </div>
        </div>
      </div>

      {/* Messages */}
      <div className="flex-1 overflow-y-auto p-4 space-y-4">
        {messages.map(m => (
          <div key={m.id} className={`flex ${
            m.role === 'user' ? 'justify-end' : 'justify-start'
          }`}>
            <div className={`max-w-[80%] p-3 rounded-lg ${
              m.role === 'user'
                ? 'bg-amber-600 text-white'
                : 'bg-white border-2 border-amber-200 shadow-md'
            }`}>
              {m.role === 'assistant' && (
                <div className="flex items-center space-x-2 mb-2">
                  <span className="text-sm font-bold text-amber-700">🥃 Alfie</span>
                </div>
              )}
              <p className={m.role === 'assistant' ? 'text-gray-800' : 'text-white'}>
                {m.content}
              </p>
            </div>
          </div>
        ))}
        {isLoading && (
          <div className="flex justify-start">
            <div className="bg-white border-2 border-amber-200 shadow-md p-3 rounded-lg">
              <div className="flex items-center space-x-2">
                <div className="w-2 h-2 bg-amber-600 rounded-full animate-bounce"></div>
                <div className="w-2 h-2 bg-amber-600 rounded-full animate-bounce" style=
{{animationDelay: '0.1s'}}></div>
                <div className="w-2 h-2 bg-amber-600 rounded-full animate-bounce" style=
{{animationDelay: '0.2s'}}></div>
                <span className="ml-2 text-sm text-amber-700">Pouring a drink...</span>
              </div>
            </div>
          </div>
        )}
        <div ref={messagesEndRef} />
      </div>

      {/* Input */}
      <div className="p-4 border-t border-amber-200 bg-white/50">
        <form onSubmit={handleSubmit} className="flex space-x-2">
          <input
            value={input}
            onChange={handleInputChange}
            placeholder="Go on then, what's on your mind?"
            className="flex-1 px-4 py-2 border-2 border-amber-200 rounded-lg focus:outline-
none focus:border-amber-400"
            disabled={isLoading}
          />
          <button
            type="submit"
            disabled={isLoading}
            className="px-6 py-2 bg-amber-600 text-white rounded-lg hover:bg-amber-700
disabled:opacity-50"
          >
            Send
          </button>
        </form>
      </div>
    </div>
  );
}
```

🌐 Web Search Integration

```
// lib/web-integration.ts
export class WebIntegration {
  private async searchWithAlfie(query: string): Promise<AlfieSearchResult> {
    // Use existing search capabilities
    const searchResults = await this.performSearch(query);

    // Filter through Alfie's perspective
    const alfieResults = searchResults.map(result => ({
      ...result,
      alfieCommentary: this.addAlfieCommentary(result)
    }));

    return {
      results: alfieResults,
      summary: this.generateAlfieSummary(alfieResults)
    };
  }

  private addAlfieCommentary(result: SearchResult): string {
    const commentaries = [
      "F*ing hell, look at this nonsense online...",
      "Right then, these internet bastards are talking rubbish...",
      "You know what? This actually makes sense, for once...",
      "Reminds me of '28 when everyone was panicking..."
    ];

    return sample(commentaries);
  }

  async answerWithWebSearch(question: string): Promise<string> {
    const search = await this.searchWithAlfie(question);

    return `
```

```
      Right then, I've had a look online for you, yeah?

      *[searches the internet]*

      F*ing hell... ${search.summary}

      You know what I think? These online experts talk a lot of shit.
      Let me tell you how it really works...
    `;
  }
}
```

📊 Monitoring & Analytics

```
// lib/alfie-analytics.ts
export class AlfieAnalytics {
  trackInteraction(userId: string, interaction: {
    mood: string;
    messageLength: number;
    responseTime: number;
    profanityCount: number;
    userSatisfaction: number;
  }) {
    // Track for optimization
    console.log('Alfie interaction:', interaction);
  }

  generateDailyReport(): string {
    return `
      Daily Report for Alfie Solomons:
      - Conversations: ${this.getDailyCount()}
      - Most used mood: ${this.getTopMood()}
      - Most discussed topic: ${this.getTopTopic()}
      - Satisfaction rate: ${this.getSatisfactionRate()}%

      "F*ing hell, you people actually like me, don't you?"
    `;
  }
}
```

🚀 Deployment Checklist

1. ☐ Add environment variables:

```
OPENAI_API_KEY=your_key_here
ALFIE_PERSONALITY_TEMPERATURE=0.9
ALFIE_MAX_TOKENS=500
ALFIE_MEMORY_LIMIT=100
```

2. [ ] Update Next.js config to include new routes
3. [ ] Add Alfie's database tables
4. [ ] Test the personality with various inputs
5. [ ] Adjust profanity levels based on audience
6. [ ] Set up monitoring for mood states
7. [ ] Create backup of personality settings

🎭 Testing Your Alfie

Test Cases:

```
// test-alfie.ts
const testCases = [
  {
    input: "Hello, can you help me with my business?",
    expectedMood: 'jovial',
    expectedKeywords: ['business', 'let me tell you']
  },
  {
    input: "You're stupid and I don't like you",
    expectedMood: 'volatile',
    expectedKeywords: ['bastard', 'respect']
  },
  {
    input: "What's the meaning of life?",
    expectedMood: 'philosophical',
    expectedKeywords: ['dead', 'truth', 'meaning']
  },
  {
    input: "Can you help me understand my competitor?",
    expectedMood: 'calculating',
    expectedKeywords: ['intelligence', 'strategy']
  }
];
```

🔄 Continuous Improvement

1. **Collect Feedback**: Track user reactions to different moods
2. **Personality Tuning**: Adjust response patterns based on engagement
3. **Story Expansion**: Add more anecdotes to the story bank
4. **Cultural Updates**: Keep references current but maintain character

📝 Tips for Success

1. **Don't over-sanitize**: Alfie's charm is in his rough edges
2. **Maintain unpredictability**: Users should never know what to expect

3. **Balance humor with wisdom**: Keep it entertaining but valuable
4. **Watch the profanity**: Adjust based on your audience
5. **Stay in character**: Consistency is key to believability

---

*"The thing about building AI, right, is it's like making gin. You start with pure spirit, add some botanicals for flavor, and if you get it right, it'll either make people happy or blind them. F*ing brilliant, innit?"*