Gemini CLI Integration Guide

Future integration with Google’s Gemini AI models

🎯 Overview

This document outlines the planned integration between ARC (Agentic Renovation Crew) and Google’s Gemini AI models through CLI and API interfaces. Currently operating in mock/offline mode, this integration will provide powerful AI capabilities for our agent orchestration system.

🚀 Planned Integration Points

1. Gemini CLI Integration

```bash
# Future CLI commands for agent interaction
gemini chat --agent-config ./configs/code-gen-agent.json
gemini analyze --file ./src/components/Agent.tsx
gemini generate --prompt "Create React component" --type typescript
```

2. API Integration

```typescript
// Future Gemini API integration
interface GeminiConfig {
  apiKey: string
  model: 'gemini-pro' | 'gemini-pro-vision'
  temperature: number
  maxOutputTokens: number
}

class GeminiAgent extends BaseAgent {
  async processRequest(message: string): Promise<StreamResponse> {
    // Integration with Gemini API
  }
}
```

3. Agent Specialization

```typescript
// Gemini-powered specialized agents
const geminiAgents: AgentConfig[] = [
  {
    id: 'gemini-code',
    name: 'Gemini Code Agent',
    provider: 'gemini',
    model: 'gemini-pro',
    specialization: 'code-generation'
  },
  {
    id: 'gemini-vision',
    name: 'Gemini Vision Agent',
    provider: 'gemini',
    model: 'gemini-pro-vision',
    specialization: 'image-analysis'
  }
]
```

🔧 Configuration Structure

Agent Configuration

```json
{
  "gemini": {
    "enabled": false,
    "apiKey": "GEMINI_API_KEY",
    "baseUrl": "https://generativelanguage.googleapis.com/v1beta",
    "defaultModel": "gemini-pro",
    "agents": [
      {
        "id": "gemini-coder",
        "type": "code-generation",
        "model": "gemini-pro",
        "systemPrompt": "You are an expert software developer...",
        "parameters": {
          "temperature": 0.3,
          "maxOutputTokens": 2048,
          "topP": 0.8
        }
      }
    ]
  }
}
```

Environment Variables

```bash
# Future environment configuration
GEMINI_API_KEY=your_gemini_api_key_here
GEMINI_PROJECT_ID=your_project_id
GEMINI_REGION=us-central1
GEMINI_MODEL_DEFAULT=gemini-pro
```

🤖 Agent Types with Gemini

1. Code Generation Agent

- **Purpose**: Generate, review, and optimize code
- **Gemini Model**: gemini-pro
- **Specialization**: Multi-language code generation
- **Features**:
- Context-aware code completion
- Bug detection and fixes
- Code refactoring suggestions
- Test generation

2. Documentation Agent

- **Purpose**: Create comprehensive documentation
- **Gemini Model**: gemini-pro
- **Specialization**: Technical writing
- **Features**:
- API documentation generation
- README creation
- Code commenting
- Architecture documentation

3. Vision Analysis Agent

- **Purpose**: Analyze images, diagrams, and UI mockups
- **Gemini Model**: gemini-pro-vision
- **Specialization**: Visual content understanding
- **Features**:
- UI mockup to code conversion
- Diagram analysis
- Image content description
- Visual debugging

🔄 Integration Workflow

1. Authentication Flow

```
// Future authentication implementation
class GeminiAuth {
  async authenticate(apiKey: string): Promise<AuthToken> {
    // Authenticate with Gemini API
  }

  async refreshToken(): Promise<AuthToken> {
    // Refresh authentication token
  }
}
```

2. Request Pipeline

```
// Future request processing pipeline
class GeminiPipeline {
  async processMessage(
    message: AgentMessage,
    config: GeminiConfig
  ): Promise<AsyncIterable<StreamChunk>> {
    // 1. Validate and prepare request
    // 2. Send to Gemini API
    // 3. Stream response back to UI
    // 4. Handle errors and retries
  }
}
```

3. Response Streaming

```
// Future streaming implementation
async function* streamGeminiResponse(
  request: GeminiRequest
): AsyncIterable<StreamChunk> {
  const response = await fetch(geminiEndpoint, {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${token}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(request)
  })

  // Stream processing logic
  for await (const chunk of processStream(response.body)) {
    yield {
      id: generateId(),
      content: chunk.content,
      done: chunk.finished
    }
  }
}
```

📊 Performance Considerations

Rate Limiting

- Respect Gemini API rate limits
- Implement intelligent request queuing
- Provide user feedback for rate limit status

Caching Strategy

- Cache frequent requests locally
- Implement conversation history caching
- Optimize token usage through context management

Error Handling

- Graceful degradation to mock responses

- Retry logic with exponential backoff
- User-friendly error messages

🔒 Security & Privacy

API Key Management

- Secure storage of API credentials
- Environment-based configuration
- No hardcoded secrets in codebase

Data Privacy

- Local conversation storage
- Optional cloud sync with encryption
- User control over data sharing

Usage Monitoring

- Token usage tracking
- Cost estimation and alerts
- Usage analytics dashboard

🧪 Testing Strategy

Mock Integration

```
// Current mock implementation for development
class MockGeminiAgent extends BaseAgent {
  async* generateResponse(prompt: string): AsyncIterable<StreamChunk> {
    // Simulate Gemini-like responses
    const responses = this.generateMockGeminiResponse(prompt)
    for (const chunk of responses) {
      yield chunk
      await sleep(100) // Simulate network latency
    }
  }
}
```

Integration Tests

- API connectivity tests
- Response format validation
- Streaming functionality tests
- Error handling verification

🎯 Migration Path

Phase 1: Mock Development (Current)

- ✅ Mock Gemini agent responses
- ✅ UI integration testing
- ✅ Conversation flow validation

Phase 2: API Integration

- 🔄 Gemini API client implementation
- 🔄 Authentication flow
- 🔄 Request/response handling

Phase 3: Advanced Features

- 📋 Vision model integration
- 📋 Multi-modal agent capabilities
- 📋 Advanced prompt engineering

Phase 4: Optimization

- 📋 Performance tuning
- 📋 Cost optimization
- 📋 Advanced caching strategies

📚 Resources

Official Documentation

- Gemini API Documentation (https://ai.google.dev/docs)
- Gemini Models Overview (https://ai.google.dev/models/gemini)
- Authentication Guide (https://ai.google.dev/docs/authentication)

Community Resources

- Gemini Developer Community
- Example integrations and patterns
- Best practices and optimization tips

Note: This integration is currently in planning/development phase. The application operates with mock responses until the full Gemini integration is implemented.

Stay tuned for updates as we bring the power of Gemini to ARC! 🚀