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

import { Button } from '../spread_sheet_swarm/ui/button';

import { Loader2, Sparkles } from 'lucide-react';

import { CoreTool, CoreToolCallUnion, CoreToolResultUnion, tool } from 'ai';

import { z } from 'zod';

import { createOpenAI } from '@ai-sdk/openai';

import {

  experimental_createProviderRegistry as createProviderRegistry,

  generateText,

} from 'ai';

import { Dialog, DialogContent, DialogFooter, DialogHeader, DialogTitle } from
'../spread_sheet_swarm/ui/dialog';

import { Label } from '../spread_sheet_swarm/ui/label';

import { Textarea } from '../spread_sheet_swarm/ui/textarea';


// Create provider registry

const registry = createProviderRegistry({

  openai: createOpenAI({

    apiKey: process.env.NEXT_PUBLIC_OPENAI_API_KEY,

  }),

});




// Define schema for a single agent

const AgentConfigSchema = z.object({

   name: z.string().describe('The unique identifier for the agent, reflecting its specific role within the
swarm and its role and purpose. Be very specific '),
  systemPrompt: z.string()
    .min(50)
      .describe('A comprehensive and detailed prompt that outlines the agent\'s expected behavior, responsibilities, and objectives'),
  description: z.string()
     .describe('A concise summary of the agent\'s primary function, highlighting its area of expertise and capabilities'),
  model: z.enum(['gpt-4o', 'gpt-4o-mini'])
    .describe('AI model to use for this agent'),
});



// Define the swarm generation tool schema
const SwarmToolSchema = z.object({
  agents: z.array(AgentConfigSchema)
  .min(4)
  .max(7)
  .describe('Array of agent configurations forming a balanced team')
});


// Define types for the swarm generation tool
type SwarmToolSet = {
  createSwarm: any
  // Ensure that the third type argument is provided if required
};
```

```
  // Define tool call and result types
type SwarmToolCall = CoreToolCallUnion<SwarmToolSet>;
type SwarmToolResult = CoreToolResultUnion<SwarmToolSet>;

  // Define component props
interface AutoGenerateSwarmProps {
    addAgent: (agent: any) => void;
    reactFlowInstance: any;
    setPopup: (popup: { message: string; type: 'success' | 'error' } | null) => void;
}

const AutoGenerateSwarm: React.FC<AutoGenerateSwarmProps> = ({ addAgent, setPopup,
reactFlowInstance }) => {
    const [isGenerating, setIsGenerating] = useState(false);
    const [isDialogOpen, setIsDialogOpen] = useState(false);
    const [task, setTask] = useState('');

    // Create the tool set
    const swarmTools: SwarmToolSet = {
      createSwarm: tool({
        description: 'Generate a balanced team of AI agents with complementary roles',
        parameters: SwarmToolSchema,
        execute: async ({ agents }) => {
          agents.forEach((agent, index) => {
            const position = calculateOptimalPosition(index, agents.length);
            addAgentWithAnimation(agent, position, index);
          });
          return { success: true, count: agents.length };
        }
      })
    };

    const calculateOptimalPosition = (index: number, totalAgents: number) => {
      const viewport = reactFlowInstance.getViewport();
      const { width, height } = reactFlowInstance.getViewport();
      const centerX = (-viewport.x + width / 2) / viewport.zoom;
      const centerY = (-viewport.y + height / 2) / viewport.zoom;
      const radius = Math.min(width, height) / 4;
      const angle = (2 * Math.PI * index) / totalAgents;

      return {
        x: centerX + radius * Math.cos(angle),
        y: centerY + radius * Math.sin(angle)
      };
    };

    const addAgentWithAnimation = (
      agent: z.infer<typeof AgentConfigSchema>,
      position: { x: number; y: number },
      index: number
    ) => {
```

```
const newNode = {
  ...agent,
  id: `auto-${Date.now()}-${index}`,
  position,
  data: {
    ...agent,
    style: {
      opacity: 0,
      scale: 0,
      transition: {
        type: 'spring',
        stiffness: 260,
        damping: 20,
        delay: index * 0.2
      }
    }
  }
};

addAgent(newNode);

setTimeout(() => {
  const updatedNode = {
    ...newNode,
    data: {
      ...newNode.data,
        style: {
          opacity: 1,
          scale: 1,
          transition: {
            type: 'spring',
            stiffness: 260,
            damping: 20
          }
        }
      }
    };
    addAgent(updatedNode);
  }, index * 200);
};

const generateAgentConfig = async (taskDescription: string) => {
  if (!taskDescription.trim()) {
    setPopup({
      message: 'Please provide a task description',
      type: 'error'
    });
    return;
  }

  setIsGenerating(true);
  try {
```

```
const result = await generateText({
  model: registry.languageModel('openai:gpt-4-turbo'),
  tools: swarmTools,
    prompt: `Analyze this task and create a specialized team of AI agents to accomplish it effectively:

  Task: "${taskDescription}"

    Create a production-ready swarm of AI agents that will work together to accomplish this specific task. Design the team considering:

  1. The specific requirements and challenges of the given task
  2. Different aspects and stages of the task that need specialized attention
  3. Required coordination and oversight for task completion

  For each agent, provide:
  - A name that reflects their specific role in handling this task
  - A role-specific system prompt that includes:
    * Clear definition of their responsibilities for this task
    * How they should interact with other team members
    * Specific outputs they should generate
    * Error handling relevant to their part of the task
  - An appropriate AI model based on their role complexity

  The team structure should include:
  - Boss agent(s) for coordinating the specific aspects of this task
```

```
          - Worker agents with complementary specialties needed for the task
          - Clear collaboration protocols specific to this task's requirements


          Call the createSwarm tool with the generated team configuration in the required format.`
        });


        setPopup({
          message: `Successfully generated agent swarm with ${result.toolResults.length} agents`,
          type: 'success'
        });
      } catch (error) {
        console.error('Error generating swarm:', error);
        setPopup({
           message: 'Failed to generate swarm: ' + (error instanceof Error ? error.message : 'Unknown
error'),
          type: 'error'
        });
      } finally {
        setIsGenerating(false);
        setIsDialogOpen(false);
        setTask('');
      }
    };


    return (
      <>
```

```
<Button
  variant="outline"
  className="bg-card hover:bg-muted"
  onClick={() => setIsDialogOpen(true)}
  disabled={isGenerating}
>
  {isGenerating ? (
    <Loader2 className="w-4 h-4 mr-2 animate-spin" />
  ) : (
    <Sparkles className="w-4 h-4 mr-2" />
  )}
  Auto-Generate Swarm
</Button>

<Dialog open={isDialogOpen} onOpenChange={setIsDialogOpen}>
  <DialogContent>
    <DialogHeader>
      <DialogTitle>Generate Agent Swarm</DialogTitle>
    </DialogHeader>
    <div className="grid gap-4 py-4">
      <div className="grid gap-2">
        <Label htmlFor="task">Task Description</Label>
        <Textarea
          id="task"
          placeholder="Describe the task you want the swarm to accomplish..."
          value={task}
```

```
              onChange={(e) => setTask(e.target.value)}
              className="min-h-[100px]"
            />
          </div>
        </div>
        <DialogFooter>
          <Button
            type="submit"
            onClick={() => generateAgentConfig(task)}
            disabled={isGenerating || !task.trim()}
          >
            {isGenerating ? (
              <Loader2 className="w-4 h-4 mr-2 animate-spin" />
            ) : (
              'Generate Swarm'
            )}
          </Button>
        </DialogFooter>
      </DialogContent>
    </Dialog>
  </>
  );
};

export default AutoGenerateSwarm;
```