Ai Deadlock Resolver

1. Project Overview

This project implements an AI-powered Deadlock Resolver system that leverages Google's Gemini Flash 2.0 model to detect and resolve deadlock situations in computing environments. Deadlocks are a significant challenge in operating systems and distributed computing, where processes become permanently blocked while waiting for resources held by other processes.

Key Capabilities

- Real-time monitoring of system states
- Analysis of resource allocation patterns
- Identification of circular wait conditions indicating potential or existing deadlocks
- Generation of optimal resolution strategies based on process priorities and resource requirements

Technical Implementation

- Modern web interface built with Next.js and React
- Professional color scheme of Deep Electric Blue and Slate Gray

2. Module-Wise Breakdown

Module 1: User Interface & Frontend

Purpose:

- Provide intuitive interface for deadlock visualization and team information

Key Components:

- Dashboard layout
- Team information cards
- Navigation system

Implementation:

- Next.js framework with React components
- TailwindCSS styling with brand colors (#0066CC, #2F4F4F)
- Framer Motion animations

Module 2: Deadlock Detection System

Purpose:

- Monitor and identify potential deadlock situations in computing resources

Key Components:

- System state analyzer
- Circular dependency detector
- Resource allocation tracker

Implementation:

- API integration for system monitoring
- Deadlock detection algorithms
- Real-time data processing

Module 3: Visualization & Reporting

Purpose:

- Present deadlock data and system status in visual format

Key Components:

- Resource allocation graphs
- Status indicators
- Team information display

Implementation:

- Interactive data visualizations
- Responsive design adaptation
- Exportable report generation

3. Functionalities

1. Resource Visualization

Shows resource allocation graphs to identify potential deadlock situations. Implements color-coding to highlight circular dependencies and resource conflicts. Includes wait-for graphs that illustrate process dependencies and potential deadlock cycles.

2. AI-Powered Analysis

Processes system state data to detect and predict deadlock scenarios. Provides automated resolution recommendations based on system priorities.

3. Interactive Dashboard

Displays real-time system metrics including CPU and memory usage. Features responsive design that adapts to different device screen sizes.

4. User Authentication

Controls access to sensitive system management functions. Implements role-based permissions for different levels of system interaction.

5. Notification System

Alerts administrators when deadlock conditions are detected. Provides immediate notification through in-app messages and email options.

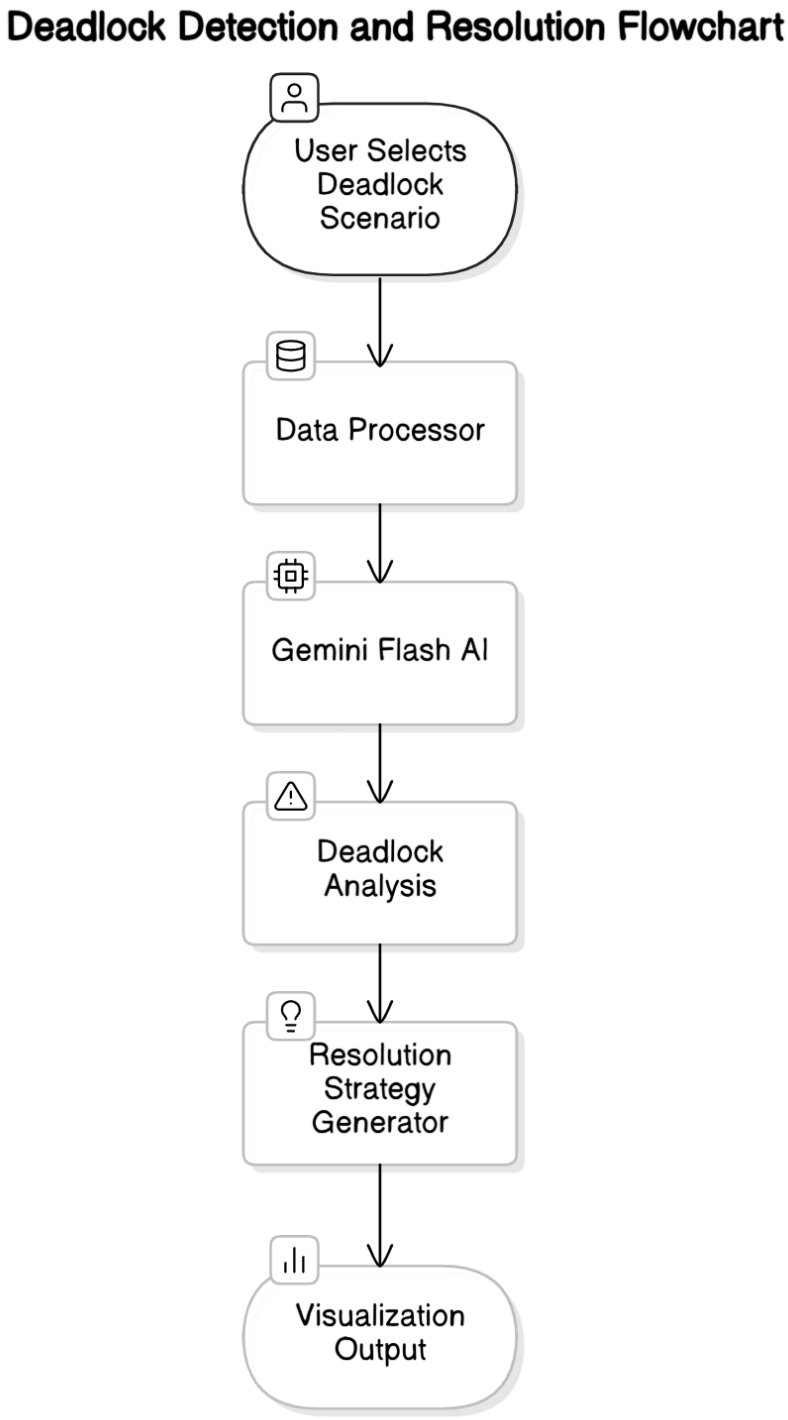
6. Team Member Display

Presents team information in responsive card layout with hover effects. Includes member photos, roles, and social media links styled with project color scheme.

4. Technology Used

- **JavaScript/TypeScript**: Core programming languages for frontend and backend development
- **Next.js**: React framework for building the web application structure
- **React**: Library for creating interactive UI components
- **TailwindCSS**: Utility-first CSS framework for styling with our blue and gray color scheme
- **Framer Motion**: Animation library for smooth transitions and interactive elements
- **Custom Components**: Self-developed visualization tools for resource graphs and CPU charts
- **Gemini AI**: Integrated through JavaScript API for deadlock detection and resolution
- **Next/Image**: Component for optimized loading of team member photos
- **Lucide React**: Icon library for social media and navigation elements
- **GitHub**: Version control and collaborative development platform
- **npm/yarn**: Package management for JavaScript dependencies
- **VS Code**: Primary integrated development environment

5. Flowchart



6. Revision Tracking on GitHub

- Repository Name: Ai-Dealock-Resolver
- GitHub Link: https://github.com/Gurbaksh363/Ai-Deadlock-Resolver

7. Conclusion and Future Scope

Conclusion

The AI Deadlock Resolver project successfully bridges the gap between theoretical deadlock concepts and practical system management through an intuitive interface. By implementing real-time visualization of resource allocation graphs and wait-for dependencies, we've created a valuable educational and diagnostic tool. The integration of AI analysis provides insights that would be difficult to achieve through conventional methods, enabling faster detection and more nuanced resolution recommendations.

Throughout development, we overcame challenges in accurately representing complex system states and optimizing the AI analysis pipeline for responsive feedback. The modular architecture ensures the system can be extended without significant refactoring.

Future Scope

- **Real-time Monitoring**: Implementing kernel-level integration for live system analysis
- **Machine Learning Integration**: Developing a component that learns from past deadlock patterns
- **Simulation Environment**: Creating dedicated testbeds for various resolution strategies
- **Specialized Environment Support**: Adding compatibility with database transaction systems
- **Advanced Visualization**: Expanding options with 3D resource dependency mapping
- **API Development**: Building interfaces for existing system monitoring tools
- **Performance Benchmarking**: Developing a comprehensive suite to evaluate resolution efficiency

8. References

1. Google Generative AI Documentation: https://ai.google.dev/docs
2. Next.js Documentation: https://nextjs.org/docs
3. "Modern Operating Systems" by Andrew S. Tanenbaum
4. React Documentation: https://reactjs.org/docs
5. TailwindCSS Documentation: https://tailwindcss.com/docs
6. Framer Motion API: https://www.framer.com/motion/

APPENDIX

A. AI-Generated Project Elaboration/Breakdown Report

C. Solution/Code:

1. Project Overview

This project aims to develop an intelligent system capable of predicting, detecting, and resolving deadlocks in computing environments. Deadlocks occur when processes are unable to proceed because each is waiting for resources held by others. The AI-based approach will analyze system behavior patterns to identify potential deadlocks before they occur and implement resolution strategies. A real-time dashboard will visualize process states and system resource utilization, enabling administrators to monitor system health effectively. The scope includes developing prediction algorithms, detection mechanisms, resolution strategies, and an intuitive visual interface that presents complex system data in an accessible format.

2. Module-Wise Breakdown

Module 1: System Monitoring and Data Collection

This module will gather real-time data about process states, resource allocations, and system metrics. It will serve as the foundation for analysis by capturing relevant information about CPU usage, memory consumption, process interdependencies, and resource locks.

Module 2: AI Analysis and Resolution Engine

The core intelligence of the system resides in this module. It will implement machine learning algorithms to analyze data patterns, identify potential deadlock conditions, and generate resolution strategies. This module will process the data collected by Module 1 and send resolution commands back to the system.

Module 3: Visualization Dashboard

This module will present system state information through an intuitive graphical interface. It will display real-time data about process states, resource utilization, and potential deadlock situations. The dashboard will include interactive elements allowing administrators to explore system details and manually trigger resolution actions if desired.

3. Functionalities

System Monitoring and Data Collection

- Continuous monitoring of process states and resource allocations
- Real-time tracking of CPU and memory usage per process
- Detection of wait conditions and resource dependencies
- Collection of historical data for pattern recognition
- Standardized data formatting for AI processing

AI Analysis and Resolution Engine

- Pattern recognition to identify potential deadlock scenarios
- Analysis of resource allocation graphs to detect circular wait conditions
- Prediction of impending deadlocks based on current system trajectory
- Generation of optimal resolution strategies with minimal disruption
- Automated implementation of resolution actions with rollback capability

Visualization Dashboard

- Real-time graphical representation of process states and dependencies
- Interactive resource allocation graphs with zoom and filter capabilities
- Color-coded status indicators showing system health and deadlock risk
- Historical trend visualization for system performance metrics
- Alert notifications for high-risk situations requiring attention

4. Technology Recommendations

Programming Languages:

- **JavaScript/TypeScript**: For frontend development and visualization
- **Python**: For backend development and AI implementation

Libraries and Tools:

- **Frontend Framework**: React or Next.js for building responsive UI components
- **State Management**: Redux or Context API for managing application state
- **Visualization**: D3.js or Chart.js for creating dynamic data visualizations
- **CSS Framework**: TailwindCSS or Material UI for styling
- **AI/ML**: TensorFlow, PyTorch, or scikit-learn for predictive modeling
- **Backend**: FastAPI or Flask for creating REST endpoints

Other Tools:

- **Version Control**: Git/GitHub for collaborative development
- **Testing**: Jest for frontend, pytest for backend
- **Deployment**: Docker for containerization
- **CI/CD**: GitHub Actions for automated testing and deployment

5. Execution Plan

Phase 1: Setup and Initial Development (Weeks 1-2)

1. Set up development environment and project structure
2. Implement basic data collection mechanisms for system monitoring
3. Design database schema for storing system state information
4. Create skeleton UI components for the dashboard

Phase 2: Core Functionality Development (Weeks 3-5)

1. Develop algorithms for analyzing resource allocation patterns
2. Implement deadlock detection logic using graph-based approaches
3. Create visualization components for representing system states
4. Build API endpoints for communication between modules

Phase 3: AI Integration and Enhancement (Weeks 6-8)

1. Implement machine learning models for pattern recognition
2. Develop predictive algorithms for deadlock forecasting
3. Create resolution strategy generator based on system priorities
4. Integrate AI recommendations into the system controller

Phase 4: Testing and Refinement (Weeks 9-10)

1. Conduct thorough testing with simulated deadlock scenarios
2. Optimize AI models based on performance metrics
3. Refine visualization for clarity and user experience
4. Implement feedback from test users

Phase 5: Finalization and Documentation (Weeks 11-12)

1. Polish UI elements and ensure responsive design
2. Create comprehensive documentation for the system
3. Prepare demonstration scenarios showing system capabilities
4. Deploy the final version and verify production readiness

B. Problem Statement

Title: AI-based Deadlock Resolver

Description: Design an AI-driven system to predict, detect, and resolve deadlocks in real time. Create a graphical dashboard that displays real-time information about process states, CPU usage, and memory consumption.

C. Solutioin/Code

Page.tsx:

```
"use client";

import { Button } from "@/components/ui/button";
import { ArrowRight, Cpu, HardDrive, Dam as Ram } from
"lucide-react";
import Link from "next/link";
import { motion } from "framer-motion";

export default function Home() {
 return (
   <div className="min-h-screen bg-gradient-to-b
from-[#2F4F4F]/15 to-[#0066CC]/10">
     <main className="container mx-auto px-4 py-16">
     {/* Hero Section */}
     <motion.div
     initial={{ opacity: 0, y: 20 }}
     animate={{ opacity: 1, y: 0 }}
     transition={{ duration: 0.6 }}
     className="text-center mb-20"
     >
     <div className="inline-block mb-3">
     <motion.div
       className="px-5 py-2 rounded-full bg-gradient-to-r
from-[#0066CC]/20 to-[#2F4F4F]/20 border border-[#0066CC]/30
text-[#0066CC] font-semibold text-sm shadow-sm"
       initial={{ opacity: 0, scale: 0.9 }}
       animate={{ opacity: 1, scale: 1 }}
       transition={{ delay: 0.2, type: "spring", stiffness: 200
}}
```

```
        whileHover={{ scale: 1.05 }}
      >
        Intelligent System Management
      </motion.div>
      </div>

      <h1 className="text-5xl md:text-7xl font-bold mb-6
bg-clip-text text-transparent bg-gradient-to-r from-[#0066CC]
to-[#2F4F4F] drop-shadow-sm">
      AI Deadlock Resolver
      </h1>
      <p className="text-lg md:text-xl text-[#2F4F4F]/80
max-w-2xl mx-auto mb-10">
      Harness the power of AI to detect and resolve operating
system deadlocks.
      Visualize resource allocation and optimize system
performance in real-time.
      </p>
      <Link href="/dashboard">
      <Button size="lg" className="bg-gradient-to-r
from-[#0066CC] to-[#0066CC]/90 hover:from-[#0066CC]/90
hover:to-[#0066CC] text-white group px-8 py-6 rounded-xl
shadow-lg shadow-[#0066CC]/30 transition-all duration-300 border
border-[#0066CC]/10">
        Get Started
        <ArrowRight className="ml-2 group-hover:translate-x-2
transition-transform" />
      </Button>
      </Link>
      </motion.div>

      {/* Features Grid */}
      <div className="grid md:grid-cols-3 gap-8 mb-24">
      {[
```

```
    {
      icon: <Cpu className="h-12 w-12" />,
      title: "CPU Monitoring",
      description: "Real-time CPU usage tracking and analysis"
    },
    {
      icon: <Ram className="h-12 w-12" />,
      title: "RAM Analysis",
      description: "Monitor memory allocation and consumption"
    },
    {
      icon: <HardDrive className="h-12 w-12" />,
      title: "Resource Tracking",
      description: "Visualize resource allocation graphs"
    }
    ].map((feature, index) => (
    <motion.div
      key={index}
      initial={{ opacity: 0, y: 20 }}
      animate={{ opacity: 1, y: 0 }}
      transition={{ duration: 0.6, delay: index * 0.2 }}
      whileHover={{ y: -10, boxShadow: "0 25px 50px -12px
rgba(0, 102, 204, 0.25)" }}
      className="p-8 rounded-2xl bg-white/95 shadow-xl
border-t-4 border-[#0066CC] hover:shadow-2xl transition-all
duration-300 backdrop-blur-sm"
    >
      <div className="mb-5 p-4 bg-gradient-to-br
from-[#0066CC]/15 to-[#2F4F4F]/10 rounded-xl inline-block
text-[#0066CC]">
      {feature.icon}
      </div>
      <h3 className="text-2xl font-bold mb-3
text-[#2F4F4F]">{feature.title}</h3>
```

```
        <p
className="text-[#2F4F4F]/80">{feature.description}</p>
      </motion.div>
      ))}
      </div>

      {/* Process Illustration */}
      <motion.div
      initial={{ opacity: 0 }}
      animate={{ opacity: 1 }}
      transition={{ duration: 0.8, delay: 0.4 }}
      className="relative h-96 md:h-[32rem] rounded-3xl
overflow-hidden bg-gradient-to-br from-[#2F4F4F]/10
to-[#0066CC]/15 shadow-2xl border border-[#0066CC]/20 mb-12"
      >
      {/* Background pattern */}
      <div className="absolute inset-0 opacity-10">
        <div className="absolute inset-0 bg-grid-pattern"></div>
      </div>

      {/* Process Visualization */}
      <div className="absolute inset-0 flex items-center
justify-center">
      <div className="w-[90%] h-[85%] relative">
        {/* Animated elements */}
        <motion.div
        className="absolute top-1/2 left-1/2 transform
-translate-x-1/2 -translate-y-1/2 w-32 h-32 md:w-40 md:h-40
rounded-full bg-gradient-to-r from-[#0066CC]/30 to-[#0066CC]/10
backdrop-blur-md border border-[#0066CC]/40 z-10 flex
items-center justify-center"
        animate={{
```

```
          boxShadow: ['0 0 20px rgba(0, 102, 204, 0.3)', '0 0
40px rgba(0, 102, 204, 0.5)', '0 0 20px rgba(0, 102, 204,
0.3)'],
        }}
        transition={{ duration: 3, repeat: Infinity }}
        >
        <div className="text-[#0066CC] font-bold text-lg">AI
Core</div>
        </motion.div>

        {/* Orbiting elements */}
        {[...Array(5)].map((_, i) => (
        <motion.div
          key={i}
          className="absolute w-16 h-16 md:w-20 md:h-20
rounded-full bg-gradient-to-r from-[#2F4F4F]/30 to-[#0066CC]/20
backdrop-blur-sm border border-white/30 flex items-center
justify-center"
          initial={{
          x: 0,
          y: 0,
          scale: 0.8 + (i * 0.05),
          }}
          animate={{
          x: Math.cos(2 * Math.PI * (i / 5)) * (150 + i * 20),
          y: Math.sin(2 * Math.PI * (i / 5)) * (150 + i * 10),
          scale: [0.8 + (i * 0.05), 0.9 + (i * 0.05), 0.8 + (i *
0.05)],
          boxShadow: ['0 0 10px rgba(0, 102, 204, 0.2)', '0 0
20px rgba(0, 102, 204, 0.4)', '0 0 10px rgba(0, 102, 204,
0.2)'],
          }}
          transition={{
          duration: 8 - i,
```

```
          repeat: Infinity,
          delay: i * 0.5,
          ease: "easeInOut"
          }}
        >
          <div className="text-[#2F4F4F] font-medium
text-xs">Process {i+1}</div>
        </motion.div>
        ))}

        {/* Connecting lines */}
        <svg className="absolute inset-0 w-full h-full z-0">
        {[...Array(5)].map((_, i) => (
          <motion.path
          key={i}
          d={`M0,0 L${Math.cos(2 * Math.PI * (i / 5)) *
150},${Math.sin(2 * Math.PI * (i / 5)) * 150}`}
          stroke={i % 2 === 0 ? "#0066CC" : "#2F4F4F"}
          strokeWidth="2"
          strokeDasharray="5,5"
          fill="none"
          initial={{ pathLength: 0, opacity: 0 }}
          animate={{ pathLength: 1, opacity: 0.7 }}
          transition={{ duration: 2, delay: i * 0.3 }}
          style={{ transformOrigin: 'center', translate: '50%
50%' }}
          />
        ))}
        </svg>

        {/* Pulse effects */}
        <motion.div
        className="absolute top-1/2 left-1/2 transform
-translate-x-1/2 -translate-y-1/2 rounded-full bg-[#0066CC]/10"
```

```
        animate={{
          scale: [1, 2.5, 1],
          opacity: [0.1, 0, 0.1],
        }}
        transition={{
          duration: 5,
          repeat: Infinity,
        }}
        style={{ width: '100px', height: '100px' }}
        />
      </div>
      </div>

      <div className="absolute bottom-8 left-1/2 -translate-x-1/2
w-[90%] md:w-auto">
      <motion.div
        initial={{ y: 20, opacity: 0 }}
        animate={{ y: 0, opacity: 1 }}
        transition={{ delay: 1 }}
        whileHover={{ scale: 1.05 }}
        className="px-8 py-5 bg-white/90 backdrop-blur-md
rounded-xl border border-[#0066CC]/30 shadow-xl"
      >
        <h3 className="text-2xl font-bold text-[#2F4F4F]
mb-2">Interactive Process Visualization</h3>
        <p className="text-[#0066CC] font-medium">Watch how AI
resolves complex deadlocks in real-time</p>
        <div className="mt-3 flex space-x-2">
        {[...Array(4)].map((_, i) => (
          <motion.div
          key={i}
          className="h-2 rounded-full bg-[#0066CC]"
          style={{ width: `${15 + i * 5}px` }}
          animate={{ opacity: [0.3, 1, 0.3] }}
```

```
          transition={{ duration: 1.5, delay: i * 0.2, repeat:
Infinity }}
          />
        ))}
        </div>
      </motion.div>
      </div>
      </motion.div>
      </main>
    </div>
  );
}
```

layout.tsx

```
import './globals.css';
import type { Metadata } from 'next';
import { Inter } from 'next/font/google';
import { ThemeProvider } from "@/components/theme-provider";
import { MainNav } from "@/components/main-nav";

const inter = Inter({ subsets: ['latin'] });

export const metadata: Metadata = {
 title: 'AI Deadlock Resolver',
 description: 'AI-powered operating system deadlock detection
and resolution',
};

export default function RootLayout({
 children,
}: {
 children: React.ReactNode;
}) {
```

```
 return (
   <html lang="en" suppressHydrationWarning>
     <body className={inter.className}>
       <ThemeProvider
         attribute="class"
         defaultTheme="system"
         enableSystem
         disableTransitionOnChange
       >
         <MainNav />
         {children}
       </ThemeProvider>
     </body>
   </html>
 );
}
```

global.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;

:root {
 --foreground-rgb: 0, 0, 0;
 --background-start-rgb: 214, 219, 220;
 --background-end-rgb: 255, 255, 255;
}

@media (prefers-color-scheme: dark) {
 :root {
   --foreground-rgb: 255, 255, 255;
   --background-start-rgb: 0, 0, 0;
   --background-end-rgb: 0, 0, 0;
 }
}
```

```
@layer base {
 :root {
   --background: 0 0% 100%;
   --foreground: 0 0% 3.9%;
   --card: 0 0% 100%;
   --card-foreground: 0 0% 3.9%;
   --popover: 0 0% 100%;
   --popover-foreground: 0 0% 3.9%;
   --primary: 0 0% 9%;
   --primary-foreground: 0 0% 98%;
   --secondary: 0 0% 96.1%;
   --secondary-foreground: 0 0% 9%;
   --muted: 0 0% 96.1%;
   --muted-foreground: 0 0% 45.1%;
   --accent: 0 0% 96.1%;
   --accent-foreground: 0 0% 9%;
   --destructive: 0 84.2% 60.2%;
   --destructive-foreground: 0 0% 98%;
   --border: 0 0% 89.8%;
   --input: 0 0% 89.8%;
   --ring: 0 0% 3.9%;
   --chart-1: 12 76% 61%;
   --chart-2: 173 58% 39%;
   --chart-3: 197 37% 24%;
   --chart-4: 43 74% 66%;
   --chart-5: 27 87% 67%;
   --radius: 0.5rem;
 }
 .dark {
   --background: 0 0% 3.9%;
   --foreground: 0 0% 98%;
   --card: 0 0% 3.9%;
   --card-foreground: 0 0% 98%;
```

```
    --popover: 0 0% 3.9%;
    --popover-foreground: 0 0% 98%;
    --primary: 0 0% 98%;
    --primary-foreground: 0 0% 9%;
    --secondary: 0 0% 14.9%;
    --secondary-foreground: 0 0% 98%;
    --muted: 0 0% 14.9%;
    --muted-foreground: 0 0% 63.9%;
    --accent: 0 0% 14.9%;
    --accent-foreground: 0 0% 98%;
    --destructive: 0 62.8% 30.6%;
    --destructive-foreground: 0 0% 98%;
    --border: 0 0% 14.9%;
    --input: 0 0% 14.9%;
    --ring: 0 0% 83.1%;
    --chart-1: 220 70% 50%;
    --chart-2: 160 60% 45%;
    --chart-3: 30 80% 55%;
    --chart-4: 280 65% 60%;
    --chart-5: 340 75% 55%;
  }
}

@layer base {
  * {
    @apply border-border;
  }
  body {
    @apply bg-background text-foreground;
  }
}
```

about/page.tsx:

```
"use client";
```

```
import { Card } from "@/components/ui/card";
import { motion } from "framer-motion";
import { AlertCircle, Cpu, HardDrive, Dam as Ram } from
"lucide-react";

export default function About() {
 return (
   <div className="min-h-screen bg-gradient-to-br from-white
via-[#0066CC]/5 to-[#2F4F4F]/10 p-8 relative overflow-hidden">
     {/* Enhanced decorative elements */}
     <div className="absolute top-20 right-10 w-96 h-96
bg-[#0066CC]/20 rounded-full blur-3xl animate-pulse"></div>
     <div className="absolute bottom-20 left-10 w-80 h-80
bg-[#2F4F4F]/15 rounded-full blur-3xl animate-pulse"
style={{animationDuration: '8s'}}></div>
     <div className="absolute top-1/3 left-1/4 w-64 h-64
bg-[#0066CC]/10 rounded-full blur-3xl"
style={{animationDuration: '15s'}}></div>

     {/* Decorative tech pattern */}
     <div className="absolute inset-0 opacity-5
pointer-events-none">
       <div className="absolute h-full w-full
bg-[url('
y53My5vcmcvMjAwMC9zdmciIHdpZHRoPSI2MCIgaGVpZ2h0PSI2MCIgdmlld0Jve
D0iMCAwIDYwIDYwIj48cGF0aCBkPSJNNTkuOTYgMEw2MCA2MGgtLjA0TDAgNjBWM
HoiIGZpbGw9Im5vbmUiIHN0cm9rZT0iIzAwNjZDQyIgc3Ryb2tlLXdpZHRoPSIuN
SIvPjwvc3ZnPg==')]"></div>
     </div>

     <motion.div
       initial={{ opacity: 0, y: 20 }}
       animate={{ opacity: 1, y: 0 }}
```

```
        transition={{ duration: 0.8 }}
        className="max-w-4xl mx-auto relative z-10"
      >
        <motion.div
          initial={{ opacity: 0, scale: 0.95 }}
          animate={{ opacity: 1, scale: 1 }}
          transition={{ duration: 0.8 }}
          className="mb-12 text-center"
        >
          <h1 className="text-6xl font-bold mb-3 text-[#2F4F4F]
drop-shadow-sm bg-clip-text text-transparent bg-gradient-to-r
from-[#2F4F4F] to-[#2F4F4F]/80">About</h1>
          <h2 className="text-3xl font-medium mb-2 text-[#0066CC]
bg-clip-text text-transparent bg-gradient-to-r from-[#0066CC]
to-[#0066CC]/80">AI Deadlock Resolver</h2>
          <div className="w-24 h-1 mx-auto bg-gradient-to-r
from-[#0066CC] to-[#2F4F4F]"></div>
        </motion.div>

        <motion.div
          whileHover={{ y: -5, boxShadow: "0 20px 25px -5px
rgba(0, 0, 0, 0.1), 0 10px 10px -5px rgba(0, 0, 0, 0.04)" }}
          transition={{ type: "spring", stiffness: 300 }}
        >
          <Card className="p-8 mb-12 bg-white/95 backdrop-blur-md
border-0 shadow-[0_10px_50px_rgba(0,102,204,0.2)] relative
overflow-hidden group">
            <div className="absolute inset-0 bg-gradient-to-br
from-[#0066CC]/10 via-transparent to-[#2F4F4F]/5 opacity-0
group-hover:opacity-100 transition-opacity duration-700"></div>
            <div className="absolute left-0 top-0 bottom-0 w-1.5
bg-gradient-to-b from-[#0066CC] to-[#2F4F4F]"></div>
            <div className="flex items-start gap-6">
```

```
            <div className="p-3 bg-[#0066CC]/10 rounded-xl
shadow-inner flex items-center justify-center">
              <AlertCircle className="h-8 w-8 text-[#0066CC]"
/>
            </div>
            <div>
              <h2 className="text-3xl font-bold mb-4
text-[#2F4F4F]">Understanding Deadlocks</h2>
              <p className="text-[#2F4F4F]/80 mb-4
leading-relaxed text-lg">
                A deadlock occurs when two or more processes
are unable to proceed because each is waiting for resources held
by another process. Our AI-powered system helps detect and
resolve these situations before they impact system performance.
              </p>
            </div>
          </div>
        </Card>
      </motion.div>

      <div className="flex items-center gap-4 mb-8">
        <h2 className="text-3xl font-bold text-[#0066CC]">Key
Features</h2>
        <div className="h-0.5 flex-grow bg-gradient-to-r
from-[#0066CC] to-transparent"></div>
      </div>

      <div className="grid md:grid-cols-2 gap-8">
        {[
          {
            icon: <Cpu className="h-8 w-8" />,
            title: "Resource Allocation Graph",
```

```
            description: "Visualize the relationships between
processes and resources to identify potential deadlock
situations."
          },
          {
            icon: <Ram className="h-8 w-8" />,
            title: "RAM Usage Monitoring",
            description: "Track memory allocation and usage
patterns in real-time to optimize system performance."
          },
          {
            icon: <HardDrive className="h-8 w-8" />,
            title: "Storage Analysis",
            description: "Monitor disk usage and I/O operations
to prevent resource conflicts."
          },
          {
            icon: <Cpu className="h-8 w-8" />,
            title: "CPU Monitoring",
            description: "Real-time tracking of processor
utilization and process scheduling."
          }
        ].map((feature, index) => (
          <motion.div
            key={index}
            initial={{ opacity: 0, y: 30 }}
            animate={{ opacity: 1, y: 0 }}
            transition={{ duration: 0.7, delay: 0.2 + index *
0.15 }}
            whileHover={{ scale: 1.03, y: -5 }}
          >
            <Card className="p-6 h-full bg-gradient-to-br
from-white to-[#0066CC]/5 backdrop-blur-md border-0 shadow-lg
```

```
hover:shadow-xl transition-all duration-500 overflow-hidden
relative">
              <div className="absolute right-0 top-0 h-full
w-1.5 bg-gradient-to-b from-[#0066CC]/50 to-[#2F4F4F]/50
opacity-0 group-hover:opacity-100 transition-opacity"></div>
              <div className="flex items-start gap-5">
                <div className="p-3 bg-gradient-to-br
from-[#0066CC]/20 to-[#2F4F4F]/10 rounded-lg shadow-inner">
                  <div
className="text-[#0066CC]">{feature.icon}</div>
                </div>
                <div>
                  <h3 className="text-xl font-bold mb-3
text-[#2F4F4F]">{feature.title}</h3>
                  <p className="text-[#2F4F4F]/80
leading-relaxed">{feature.description}</p>
                </div>
              </div>
              <div className="absolute bottom-0 left-0 right-0
h-1 bg-gradient-to-r from-transparent via-[#0066CC]/70
to-transparent transform scale-x-0 group-hover:scale-x-100
transition-transform duration-500"></div>
            </Card>
          </motion.div>
        ))}
      </div>
    </motion.div>
  </div>
 );
}
```

team/page.tsx:

```
"use client";

import { Card } from "@/components/ui/card";
```

```
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import { Label } from "@/components/ui/label";
import { motion, AnimatePresence } from "framer-motion";
import ReactFlow, {
 Background,
 Controls,
 Edge,
 Node,
 Position,
 MarkerType,
} from "react-flow-renderer";
import {
 AreaChart,
 Area,
 XAxis,
 YAxis,
 CartesianGrid,
 Tooltip,
 ResponsiveContainer,
 LineChart,
 Line,
 PieChart,
 Pie,
 Cell,
} from "recharts";
import { useState, useEffect, useMemo } from "react";
import {
 Table,
 TableBody,
 TableCell,
 TableHead,
 TableHeader,
 TableRow,
```

```
} from "@/components/ui/table";
import {
 AlertCircle,
 Plus,
 X,
 Info,
 Activity,
 Cpu,
 HardDrive,
 Sparkles,
} from "lucide-react";
import { Alert, AlertDescription, AlertTitle } from
"@/components/ui/alert";
import {
 Select,
 SelectContent,
 SelectItem,
 SelectTrigger,
 SelectValue,
} from "@/components/ui/select";
import {
 Dialog,
 DialogContent,
 DialogDescription,
 DialogHeader,
 DialogTitle,
 DialogTrigger,
} from "@/components/ui/dialog";
import { Tabs, TabsContent, TabsList, TabsTrigger } from
"@/components/ui/tabs";
import { Badge } from "@/components/ui/badge";
import { Separator } from "@/components/ui/separator";
import { GoogleGenerativeAI } from "@google/generative-ai";
import { marked } from "marked";
```

```
// Simulated data
const generateRandomData = (count: number) => {
 return Array.from({ length: count }, (_, i) => ({
   time: i,
   cpu: Math.floor(Math.random() * 100),
   ram: Math.floor(Math.random() * 100),
   storage: Math.floor(Math.random() * 100),
 }));
};

interface Process {
 id: string;
 resources: string[];
 status: "Running" | "Waiting" | "Blocked";
}

const processNodeStyle = {
 padding: 10,
 borderRadius: 8,
 border: "1px solid #555",
 width: 150,
 fontSize: "12px",
 boxShadow: "0 4px 6px rgba(0, 0, 0, 0.1)",
};

const resourceNodeStyle = {
 padding: 10,
 borderRadius: 8,
 border: "1px solid #555",
 width: 100,
 fontSize: "12px",
 boxShadow: "0 4px 6px rgba(0, 0, 0, 0.1)",
};
```

```
const COLORS = [
 "#0088FE",
 "#00C49F",
 "#FFBB28",
 "#FF8042",
 "#8884d8",
 "#FF6B6B",
];

export default function Dashboard() {
 const [data, setData] = useState(generateRandomData(10));
 const [hasDeadlock, setHasDeadlock] = useState(false);
 const [deadlockCycle, setDeadlockCycle] =
useState<string[]>([]);
 const [processes, setProcesses] = useState<Process[]>([
   { id: "P1", resources: ["R1", "R2"], status: "Running" },
   { id: "P2", resources: ["R2", "R3"], status: "Waiting" },
   { id: "P3", resources: ["R3", "R4"], status: "Running" },
   { id: "P4", resources: ["R1", "R4"], status: "Blocked" },
 ]);
 const [newProcess, setNewProcess] = useState<Process>({
   id: "",
   resources: [],
   status: "Running",
 });
 const [newResource, setNewResource] = useState("");
 const [simulationSpeed, setSimulationSpeed] = useState(2000);
 const [isDialogOpen, setIsDialogOpen] = useState(false);
 const [activeTab, setActiveTab] = useState("allocation");

 // Gemini API integration states
 const [geminiResponse, setGeminiResponse] =
useState<string>("");
```

```
 const [isGeminiLoading, setIsGeminiLoading] = useState(false);
 const [geminiSuggestions, setGeminiSuggestions] =
useState<string[]>([]);

 // Generate resource mapping (which process holds which
resource)
 const resourceAllocation = useMemo(() => {
   const allocation: Record<string, string[]> = {};

   processes.forEach((process) => {
     process.resources.forEach((resource) => {
       if (!allocation[resource]) {
         allocation[resource] = [];
       }
       allocation[resource].push(process.id);
     });
   });

   return allocation;
 }, [processes]);

 // Generate nodes and edges for the resource allocation graph
 const { nodes, edges } = useMemo(() => {
   const nodes: Node[] = [];
   const edges: Edge[] = [];
   const resourceSet = new Set<string>();

   // Collect all unique resources
   processes.forEach((process) => {
     process.resources.forEach((resource) =>
resourceSet.add(resource));
   });

   // Add process nodes
```

```
  processes.forEach((process, index) => {
    nodes.push({
      id: process.id,
      type: "default",
      data: {
        label: (
          <div>
            <div className="font-semibold">{process.id}</div>
            <div
              className={`text-xs ${
                process.status === "Running"
                  ? "text-green-500"
                  : process.status === "Waiting"
                  ? "text-yellow-500"
                  : "text-red-500"
              }`}
            >
              {process.status}
            </div>
          </div>
        ),
      },
      position: {
        x: 50 + (index % 2) * 300,
        y: 50 + Math.floor(index / 2) * 200,
      },
      style: {
        ...processNodeStyle,
        background:
          process.status === "Blocked"
            ? "linear-gradient(135deg, rgba(239, 68, 68, 0.1),
rgba(239, 68, 68, 0.2))"
            : "linear-gradient(135deg, rgba(59, 130, 246, 0.1),
rgba(59, 130, 246, 0.2))",
```

```
      },
    });
  });

  // Add resource nodes
  Array.from(resourceSet).forEach((resource, index) => {
    nodes.push({
      id: resource,
      type: "default",
      data: { label: resource },
      position: {
        x: 200 + (index % 2) * 200,
        y: 150 + Math.floor(index / 2) * 200,
      },
      style: {
        ...resourceNodeStyle,
        background:
          "linear-gradient(135deg, rgba(99, 102, 241, 0.1),
rgba(99, 102, 241, 0.2))",
      },
    });
  });

  // Add edges
  processes.forEach((process) => {
    process.resources.forEach((resource) => {
      edges.push({
        id: `${process.id}-${resource}`,
        source: process.id,
        target: resource,
        type: "smoothstep",
        animated: process.status === "Blocked",
        style: {
```

```
          stroke: process.status === "Blocked" ? "#ef4444" :
"#3b82f6",
          strokeWidth: 2,
        },
        markerEnd: {
          type: MarkerType.ArrowClosed,
          color: process.status === "Blocked" ? "#ef4444" :
"#3b82f6",
        },
      });
    });
  });

  return { nodes, edges };
}, [processes]);

// Generate Wait-For Graph
const waitForGraph = useMemo(() => {
  const nodes: Node[] = [];
  const edges: Edge[] = [];

  // Create nodes in a circle with enough spacing to avoid
overlaps
  const radius = 200; // Increased radius for more space
  const centerX = 225;
  const centerY = 225;

  // Create process nodes in a circle
  processes.forEach((process, index) => {
    const angle = (index / processes.length) * 2 * Math.PI;
    const x = centerX + radius * Math.cos(angle);
    const y = centerY + radius * Math.sin(angle);

    nodes.push({
```

```
      id: process.id,
      type: "default",
      data: {
        label: process.id,
      },
      position: { x, y },
      style: {
        ...processNodeStyle,
        background: deadlockCycle.includes(process.id)
          ? "rgba(239, 68, 68, 0.2)"
          : "rgba(14, 165, 233, 0.2)",
        borderColor: deadlockCycle.includes(process.id) ?
"#ef4444" : "#555",
      },
    });
  });

  // Create wait-for relationships
  processes.forEach((waitingProcess) => {
    // Only blocked or waiting processes can wait for others
    if (
      waitingProcess.status !== "Blocked" &&
      waitingProcess.status !== "Waiting"
    ) {
      return;
    }

    // Track which processes this process is waiting for
    const waitingFor = new Set<string>();

    // Find resources this process needs
    waitingProcess.resources.forEach((resource) => {
      // Find processes holding these resources
```

```
      const holdingProcesses = resourceAllocation[resource] ||
[];

      // Add holders (except self) to waiting set
      holdingProcesses
        .filter((holderId) => holderId !== waitingProcess.id)
        .forEach((holderId) => waitingFor.add(holderId));
    });

    // Create edges for wait relationships
    waitingFor.forEach((targetId) => {
      // Check if this edge is part of the deadlock cycle
      const isDeadlockEdge =
        deadlockCycle.length > 0 &&
        deadlockCycle.includes(waitingProcess.id) &&
        deadlockCycle.includes(targetId) &&
        deadlockCycle.indexOf(waitingProcess.id) ===
          (deadlockCycle.indexOf(targetId) - 1 +
deadlockCycle.length) %
            deadlockCycle.length;

      // Calculate curvature to avoid edges crossing through
nodes
      // Edges will curve differently based on their position
in the graph
      const sourceIdx = processes.findIndex(
        (p) => p.id === waitingProcess.id
      );
      const targetIdx = processes.findIndex((p) => p.id ===
targetId);
      const nodeDistance = Math.abs(sourceIdx - targetIdx);
      // Higher curvature for edges spanning across the graph
      const curvature = nodeDistance > processes.length / 2 ?
0.7 : 0.3;
```

```
      edges.push({
        id: `wait-${waitingProcess.id}-${targetId}`,
        source: waitingProcess.id,
        target: targetId,
        type: "bezier", // Changed from smoothstep to bezier
for better routing
        animated: isDeadlockEdge,
        style: {
          stroke: isDeadlockEdge ? "#ef4444" : "#0ea5e9",
          strokeWidth: isDeadlockEdge ? 3 : 1.5,
        },
        markerEnd: {
          type: MarkerType.ArrowClosed,
          color: isDeadlockEdge ? "#ef4444" : "#0ea5e9",
        },
        curvature: curvature, // Add curvature to bend the
edges around nodes
      });
    });
  });

  return { nodes, edges };
}, [processes, resourceAllocation, deadlockCycle]);

// NEW FUNCTION: Function to automatically update process
statuses based on resource availability
const updateProcessStatuses = () => {
  // Track which resources are already allocated to running
processes
  const allocatedResources = new Map();

  // First pass: build map of allocated resources
  setProcesses(prevProcesses => {
```

```
    // First pass: mark resources that are allocated to running
processes
    prevProcesses.forEach(process => {
      if (process.status === "Running") {
        process.resources.forEach(resource => {
          if (!allocatedResources.has(resource)) {
            allocatedResources.set(resource, process.id);
          } else {
            // If resource is already allocated, append this
process
            const current = allocatedResources.get(resource);
            if (Array.isArray(current)) {
              allocatedResources.set(resource, [...current,
process.id]);
            } else {
              allocatedResources.set(resource, [current,
process.id]);
            }
          }
        });
      }
    });

    // Second pass: update process statuses based on resource
contention
    const updatedProcesses = prevProcesses.map(process => {
      // Skip running processes
      if (process.status === "Running") return process;

      // Check if any of the resources this process needs are
already allocated
      const conflictingResources =
process.resources.filter(resource => {
        if (!allocatedResources.has(resource)) return false;
```

```
        const holders = allocatedResources.get(resource);
        if (Array.isArray(holders)) {
          return !holders.includes(process.id);
        }
        return holders !== process.id;
      });

      // If waiting for resources, mark as blocked
      if (conflictingResources.length > 0) {
        return { ...process, status: "Blocked" };
      }

      return process;
    });

    // Return updated processes array
    return updatedProcesses;
  });

  // After updating statuses, check for deadlocks
  setTimeout(checkForDeadlock, 100);
};

// NEW FUNCTION: Create a deadlock scenario for testing
const createDeadlockScenario = () => {
  // Create a simple deadlock scenario with 2 processes and 2
resources
  const deadlockProcesses = [
    { id: "D1", resources: ["R1"], status: "Running" },
    { id: "D2", resources: ["R2"], status: "Running" },
  ];

  // Set initial processes with allocated resources
```

```
    setProcesses(deadlockProcesses);

    // Reset deadlock state and Gemini analysis
    setHasDeadlock(false);
    setDeadlockCycle([]);
    setGeminiResponse("");
    setGeminiSuggestions([]);

    // After a brief delay, add blocked resource requests to
create circular wait
    setTimeout(() => {
      setProcesses((prev) => [
        { ...prev[0], resources: ["R1", "R2"], status: "Blocked"
},
        { ...prev[1], resources: ["R2", "R1"], status: "Blocked"
},
      ]);

      // Run deadlock detection
      setTimeout(checkForDeadlock, 100);
    }, 1000);
  };

  // Simulate real-time updates
  useEffect(() => {
    const interval = setInterval(() => {
      setData((prev) => [
        ...prev.slice(1),
        {
          time: prev[prev.length - 1].time + 1,
          cpu: Math.floor(Math.random() * 100),
          ram: Math.floor(Math.random() * 100),
          storage: Math.floor(Math.random() * 100),
        },
```

```
    ]);
  }, simulationSpeed);

  return () => clearInterval(interval);
}, [simulationSpeed]);

// Function to detect circular wait (find cycles in the
wait-for graph)
const findCycles = () => {
  // Create adjacency list for the wait-for graph
  const adjacencyList: Record<string, string[]> = {};
  processes.forEach((p) => {
    adjacencyList[p.id] = [];
  });

  // Build adjacency list of which process is waiting for which
other process
  processes.forEach((process) => {
    if (process.status === "Blocked" || process.status ===
"Waiting") {
      process.resources.forEach((resource) => {
        const holdingProcesses = resourceAllocation[resource]
|| [];

        holdingProcesses
          .filter((holderId) => holderId !== process.id) //
Avoid self-loops
          .forEach((holderId) => {
            adjacencyList[process.id].push(holderId);
          });
      });
    }
  });
```

```
  // DFS to find cycles
  const visited = new Set<string>();
  const recursionStack = new Set<string>();
  let cycleFound = false;
  let detectedCycle: string[] = [];

  const dfs = (node: string, path: string[] = []) => {
    if (cycleFound) return;

    visited.add(node);
    recursionStack.add(node);

    const currentPath = [...path, node];

    for (const neighbor of adjacencyList[node]) {
      if (!visited.has(neighbor)) {
        dfs(neighbor, currentPath);
      } else if (recursionStack.has(neighbor)) {
        // Found a cycle
        cycleFound = true;
        const cycleStartIndex = currentPath.indexOf(neighbor);
        detectedCycle = [...currentPath.slice(cycleStartIndex),
neighbor];
        return;
      }
    }

    recursionStack.delete(node);
  };

  // Run DFS from each node to find cycles
  for (const processId of Object.keys(adjacencyList)) {
    if (!visited.has(processId) && !cycleFound) {
      dfs(processId);
```

```
    }
  }

  return { hasCycle: cycleFound, cycle: detectedCycle };
};

const checkForDeadlock = () => {
  const { hasCycle, cycle } = findCycles();
  setHasDeadlock(hasCycle);
  setDeadlockCycle(cycle);

  if (hasCycle) {
    // Auto-switch to Wait-For graph tab to visualize the
deadlock
    setActiveTab("waitfor");
  }
};

// UPDATED: Add process function now updates statuses & checks
for deadlocks
const addProcess = () => {
  if (newProcess.id && newProcess.resources.length > 0) {
    // Use functional update pattern to ensure we get the
latest state
    setProcesses((prevProcesses) => [
      ...prevProcesses,
      {
        id: newProcess.id,
        resources: [...newProcess.resources], // Create a new
array copy
        status: newProcess.status,
      },
    ]);
```

```
    // Reset form fields
    setNewProcess({ id: "", resources: [], status: "Running"
});
    setIsDialogOpen(false);

    // Reset Gemini analysis when system state changes
    setGeminiResponse("");
    setGeminiSuggestions([]);

    // After adding a process, update statuses and check for
deadlocks
    setTimeout(() => {
      updateProcessStatuses();
    }, 100);
  }
 };

 // UPDATED: Add resource function now checks for deadlocks
after adding
 const addResourceToProcess = () => {
  if (newResource) {
    setNewProcess({
      ...newProcess,
      resources: [...newProcess.resources, newResource],
    });
    setNewResource("");
  }
 };

 const removeResource = (index: number) => {
  setNewProcess({
    ...newProcess,
    resources: newProcess.resources.filter((_, i) => i !==
index),
```

```
    });
  };

  const removeProcess = (processId: string) => {
    setProcesses(processes.filter((p) => p.id !== processId));
    // Reset deadlock state when a process is removed
    if (deadlockCycle.includes(processId)) {
      setHasDeadlock(false);
      setDeadlockCycle([]);
    }
    // Reset Gemini analysis when system state changes
    setGeminiResponse("");
    setGeminiSuggestions([]);

    // After removing a process, update statuses and check for
deadlocks
    setTimeout(updateProcessStatuses, 100);
  };

  // Gemini API integration for AI-powered deadlock analysis
  const analyzeDeadlockWithGemini = async () => {
    setIsGeminiLoading(true);
    try {
      // Initialize Gemini API (you'll need to get an API key
from Google AI Studio)
      const genAI = new
GoogleGenerativeAI(process.env.GEMINI_API_KEY ||
"AIzaSyDj-xN8OuC0w9S0382GkRT92xE-AaYPgC4");
      const model = genAI.getGenerativeModel({ model:
"gemini-1.5-pro" });

      // Prepare data for Gemini
      const processData = JSON.stringify(processes, null, 2);
      const resourceAllocationData = JSON.stringify(
```

```
      resourceAllocation,
      null,
      2
    );

    // Create prompt
    const prompt = `
      Analyze this system for deadlocks:

      Processes:
      ${processData}

      Resource Allocation:
      ${resourceAllocationData}

      Please provide:
      1. Whether a deadlock exists and which processes are
involved
      2. The exact circular wait chain if a deadlock exists
      3. Three specific suggestions to resolve the deadlock
      4. What would happen if we don't resolve this deadlock
      5. Potential future deadlock scenarios based on the
current allocation pattern

      Format your response in markdown.
    `;

    // Call Gemini API
    const result = await model.generateContent(prompt);
    const response = result.response.text();

    // Process response - extract suggestions
    const suggestions = extractSuggestions(response);
```

```
      setGeminiResponse(response);
      setGeminiSuggestions(suggestions);

      // If Gemini found a deadlock (inferred from suggestions)
      if (suggestions.length > 0 && !hasDeadlock) {
        // Run our local detection to visualize it
        checkForDeadlock();
      }
    } catch (error) {
      console.error("Error calling Gemini API:", error);
      setGeminiResponse(
        "Failed to analyze with Gemini API. Please check your API
key and try again."
      );
    } finally {
      setIsGeminiLoading(false);
    }
  };

  // Helper function to extract suggestions from Gemini response
  const extractSuggestions = (response: string): string[] => {
    // Simple regex to extract numbered suggestions
    const suggestionPattern = /(\d+\.\s+[^\n]+)/g;
    const matches = response.match(suggestionPattern) || [];
    return matches.slice(0, 3); // Return top 3 suggestions
  };

  // Apply a suggestion from Gemini (example implementation -
terminate a process)
  const applySuggestion = (suggestion: string) => {
    // Simple heuristic to extract process IDs from suggestions
    const processIdMatch = suggestion.match(
      /(?:terminate|kill|remove)\s+(\w+)/i
    );
```

```
    if (processIdMatch && processIdMatch[1]) {
      const processId = processIdMatch[1];
      removeProcess(processId);
      return;
    }

    // Add more suggestion parsing/application logic here
    alert(`Suggestion applied: ${suggestion}`);
  };

  // Calculate utilization for pie chart
  const utilizationData = useMemo(() => {
    const latest = data[data.length - 1];
    return [
      { name: "CPU", value: latest.cpu, color: "#0088FE" },
      { name: "RAM", value: latest.ram, color: "#00C49F" },
      { name: "Storage", value: latest.storage, color: "#FFBB28"
},
    ];
  }, [data]);

  return (
    <div className="min-h-screen bg-gradient-to-b from-[#1c2e4a]
to-[#0c1829] p-8 text-white">
      <motion.div
        initial={{ opacity: 0, y: 20 }}
        animate={{ opacity: 1, y: 0 }}
        transition={{ duration: 0.6 }}
        className="max-w-7xl mx-auto space-y-8"
      >
        <div className="flex flex-wrap gap-4 items-center
justify-between">
          <div className="flex items-center space-x-3">
            <motion.div
```

```
            initial={{ rotate: 0 }}
            animate={{ rotate: 360 }}
            transition={{ duration: 2, repeat: Infinity, ease:
"linear" }}
            className="text-[#0088FE] bg-[#0088FE]/10 p-2
rounded-full"
          >
            <Activity className="h-8 w-8" />
          </motion.div>
          <h1 className="text-4xl font-bold bg-clip-text
text-transparent bg-gradient-to-r from-[#0088FE] to-[#00C49F]">
            System Dashboard
          </h1>
        </div>
        <div className="flex gap-4">
          <Dialog open={isDialogOpen}
onOpenChange={setIsDialogOpen}>
            <DialogTrigger asChild>
              <Button className="bg-gradient-to-r
from-[#0088FE] to-[#00C49F] hover:from-[#0088FE]/90
hover:to-[#00C49F]/90 text-white border-none">
                <Plus className="mr-2 h-4 w-4" />
                Add Process
              </Button>
            </DialogTrigger>
            <DialogContent className="border-[#0088FE]/20
bg-[#1c2e4a] text-white">
              <DialogHeader>
                <DialogTitle className="text-white">
                  Add New Process
                </DialogTitle>
                <DialogDescription className="text-white/70">
                  Create a new process with resources and
status.
```

```
                </DialogDescription>
              </DialogHeader>
              <div className="space-y-4 pt-4">
                <div className="space-y-2">
                  <Label htmlFor="processId"
className="text-white">
                    Process ID
                  </Label>
                  <Input
                    id="processId"
                    value={newProcess.id}
                    onChange={(e) =>
                      setNewProcess({ ...newProcess, id:
e.target.value })
                    }
                    placeholder="Enter process ID (e.g., P5)"
                    className="bg-[#0c1829] border-[#304060]
text-white"
                  />
                </div>
                <div className="space-y-2">
                  <Label
className="text-white">Resources</Label>
                  <div className="flex gap-2">
                    <Input
                      value={newResource}
                      onChange={(e) =>
setNewResource(e.target.value)}
                      placeholder="Enter resource (e.g., R1)"
                      className="bg-[#0c1829] border-[#304060]
text-white"
                    />
                    <Button
                      onClick={addResourceToProcess}
```

```
                      type="button"
                      className="bg-[#0088FE]
hover:bg-[#0088FE]/90"
                    >
                      Add
                    </Button>
                  </div>
                  <div className="flex flex-wrap gap-2 mt-2">
                    {newProcess.resources.map((resource, index)
=> (
                      <Badge
                        key={index}
                        className="bg-[#304060] text-white flex
items-center gap-1"
                      >
                        {resource}
                        <button
                          onClick={() => removeResource(index)}
                          className="hover:text-[#FF6B6B]"
                        >
                          <X className="h-3 w-3" />
                        </button>
                      </Badge>
                    ))}
                  </div>
                </div>
                <div className="space-y-2">
                  <Label htmlFor="status"
className="text-white">
                    Status
                  </Label>
                  <Select
                    value={newProcess.status}
                    onValueChange={(
```

```
                    value: "Running" | "Waiting" | "Blocked"
                  ) => setNewProcess({ ...newProcess, status:
value })}
                >
                  <SelectTrigger className="bg-[#0c1829]
border-[#304060] text-white">
                    <SelectValue placeholder="Select status"
/>
                  </SelectTrigger>
                  <SelectContent className="bg-[#1c2e4a]
border-[#304060] text-white">
                    <SelectItem value="Running"
className="text-green-400">
                      Running
                    </SelectItem>
                    <SelectItem value="Waiting"
className="text-yellow-400">
                      Waiting
                    </SelectItem>
                    <SelectItem value="Blocked"
className="text-red-400">
                      Blocked
                    </SelectItem>
                  </SelectContent>
                </Select>
              </div>
              <Button
                onClick={addProcess}
                className="w-full bg-gradient-to-r
from-[#0088FE] to-[#00C49F]"
              >
                Create Process
              </Button>
            </div>
```

```
            </DialogContent>
          </Dialog>

          {/* NEW: Added Create Deadlock Scenario button */}
          <Button
            onClick={createDeadlockScenario}
            className="bg-gradient-to-r from-[#FF6B6B]
to-[#FFBB28] hover:opacity-90 text-white"
          >
            <AlertCircle className="mr-2 h-4 w-4" />
            Create Deadlock Example
          </Button>

          <div className="flex gap-2">
            <Button
              onClick={checkForDeadlock}
              size="default"
              className="bg-gradient-to-r from-[#FF6B6B]
to-[#FFBB28] hover:opacity-90 text-white"
            >
              <AlertCircle className="mr-2 h-4 w-4" />
              Quick Deadlock Check
            </Button>

            <Button
              onClick={analyzeDeadlockWithGemini}
              size="default"
              className="bg-gradient-to-r from-[#8884d8]
to-[#0088FE] hover:opacity-90 text-white"
              disabled={isGeminiLoading}
            >
              {isGeminiLoading ? (
                <>
                  <span className="animate-spin mr-2">⟳</span>
```

```
                  Analyzing...
                </>
              ) : (
                <>
                  <Sparkles className="mr-2 h-4 w-4" />
                  AI Deadlock Analysis
                </>
              )}
            </Button>
          </div>
        </div>
      </div>

      <AnimatePresence>
        {hasDeadlock && (
          <motion.div
            initial={{ opacity: 0, y: -20 }}
            animate={{ opacity: 1, y: 0 }}
            exit={{ opacity: 0, y: -20 }}
            transition={{ duration: 0.3 }}
          >
            <Alert
              variant="destructive"
              className="bg-red-500/20 border-red-500
text-white"
            >
              <AlertCircle className="h-4 w-4" />
              <AlertTitle>Deadlock Detected!</AlertTitle>
              <AlertDescription className="text-white/90">
                A circular wait condition has been detected
between processes:
                <span className="font-bold ml-1">
                  {deadlockCycle.join(" → ")} →
{deadlockCycle[0]}
```

```
                </span>
                . Consider terminating one of the blocked
processes to resolve
                the deadlock.
              </AlertDescription>
            </Alert>
          </motion.div>
        )}
      </AnimatePresence>

      {/* Gemini Analysis Results */}
      <AnimatePresence>
        {geminiResponse && (
          <motion.div
            initial={{ opacity: 0, y: -20 }}
            animate={{ opacity: 1, y: 0 }}
            exit={{ opacity: 0, y: -20 }}
            transition={{ duration: 0.3 }}
          >
            <Card className="p-6 bg-[#1c2e4a]/70
backdrop-blur-sm border-[#304060]">
              <div className="flex items-center justify-between
mb-4">
                <h2 className="text-xl font-semibold text-white
flex items-center">
                  <Sparkles className="mr-2 h-5 w-5
text-[#8884d8]" />
                  AI Deadlock Analysis
                </h2>
                <Button
                  variant="ghost"
                  size="sm"
                  onClick={() => {
                    setGeminiResponse("");
```

```
                    setGeminiSuggestions([]);
                  }}
                  className="text-white/70 hover:text-white"
                >
                  <X className="h-4 w-4" />
                </Button>
              </div>

              <div className="prose prose-invert max-w-none">
                <div
                  className="text-white/90"
                  dangerouslySetInnerHTML={{
                    __html: marked.parse(geminiResponse),
                  }}
                />
              </div>

              {geminiSuggestions.length > 0 && (
                <div className="mt-6">
                  <h3 className="text-lg font-semibold mb-2
text-white">
                    Quick Resolution Options
                  </h3>
                  <div className="flex flex-wrap gap-2">
                    {geminiSuggestions.map((suggestion, idx) =>
(
                      <Badge
                        key={idx}
                        className="bg-[#8884d8]/20
text-[#8884d8] p-2 text-sm cursor-pointer hover:bg-[#8884d8]/30"
                        onClick={() =>
applySuggestion(suggestion)}
                      >
                        {suggestion}
```

```
                    </Badge>
                  ))}
                </div>
              </div>
            )}
          </Card>
        </motion.div>
      )}
    </AnimatePresence>
    <div className="grid grid-cols-1 md:grid-cols-2
xl:grid-cols-3 gap-6">
      {/* Resource Allocation Graph Panel */}
      <Card className="xl:col-span-2 bg-[#1c2e4a]/70
backdrop-blur-sm border-[#304060]">
        <div className="p-6">
          <div className="flex justify-between items-center
mb-4">
            <h2 className="text-xl font-semibold">
              Resource Allocation Graph
            </h2>
            <Tabs
              value={activeTab}
              onValueChange={setActiveTab}
              className="w-auto"
            >
              <TabsList className="bg-[#0c1829]">
                <TabsTrigger value="allocation">
                  Allocation Graph
                </TabsTrigger>
                <TabsTrigger value="waitfor">Wait-For
Graph</TabsTrigger>
              </TabsList>
            </Tabs>
          </div>
```

```
            <div className="h-[400px] bg-[#0c1829] rounded-md
border border-[#304060]">
              {activeTab === "allocation" ? (
                <ReactFlow nodes={nodes} edges={edges} fitView>
                  <Controls className="bg-[#1c2e4a]
border-[#304060] rounded-md text-white" />
                  <Background color="#304060" gap={16} />
                </ReactFlow>
              ) : (
                <ReactFlow
                  nodes={waitForGraph.nodes}
                  edges={waitForGraph.edges}
                  fitView
                >
                  <Controls className="bg-[#1c2e4a]
border-[#304060] rounded-md text-white" />
                  <Background color="#304060" gap={16} />
                </ReactFlow>
              )}
            </div>

            <div className="mt-4 flex items-center
justify-between">
              <div className="flex items-center gap-2">
                <div className="flex items-center">
                  <div className="w-3 h-3 rounded-full
bg-green-500 mr-1"></div>
                  <span className="text-xs
text-white/70">Running</span>
                </div>
                <div className="flex items-center">
                  <div className="w-3 h-3 rounded-full
bg-yellow-500 mr-1"></div>
```

```
                  <span className="text-xs
text-white/70">Waiting</span>
                </div>
                <div className="flex items-center">
                  <div className="w-3 h-3 rounded-full
bg-red-500 mr-1"></div>
                  <span className="text-xs
text-white/70">Blocked</span>
                </div>
              </div>
              <div className="flex items-center gap-2">
                <span className="text-xs text-white/70">
                  Simulation Speed:
                </span>
                <Select
                  value={simulationSpeed.toString()}
                  onValueChange={(value) =>
                    setSimulationSpeed(parseInt(value))
                  }
                >
                  <SelectTrigger className="w-[120px]
bg-[#0c1829] border-[#304060] text-white h-8">
                    <SelectValue placeholder="Speed" />
                  </SelectTrigger>
                  <SelectContent className="bg-[#1c2e4a]
border-[#304060] text-white">
                    <SelectItem value="500">Very
Fast</SelectItem>
                    <SelectItem value="1000">Fast</SelectItem>
                    <SelectItem
value="2000">Normal</SelectItem>
                    <SelectItem value="3000">Slow</SelectItem>
                  </SelectContent>
                </Select>
```

```
              </div>
            </div>
          </div>
        </Card>

        {/* System Metrics */}
        <Card className="bg-[#1c2e4a]/70 backdrop-blur-sm
border-[#304060]">
          <div className="p-6">
            <h2 className="text-xl font-semibold mb-4 flex
items-center">
              <Activity className="mr-2 h-5 w-5 text-[#0088FE]"
/>
              System Metrics
            </h2>

            <div className="space-y-6">
              <div>
                <h3 className="text-sm text-white/70 mb-2">
                  Resource Utilization
                </h3>
                <div className="h-[200px]">
                  <ResponsiveContainer width="100%"
height="100%">
                    <PieChart>
                      <Pie
                        data={utilizationData}
                        cx="50%"
                        cy="50%"
                        labelLine={false}
                        outerRadius={80}
                        fill="#8884d8"
                        dataKey="value"
                        label={({ name, percent }) =>
```

```
                          `${name}: ${(percent *
100).toFixed(0)}%`
                        }
                      >
                        {utilizationData.map((entry, index) =>
(
                          <Cell key={`cell-${index}`}
fill={entry.color} />
                        ))}
                      </Pie>
                      <Tooltip
                        formatter={(value) => [`${value}%`,
"Utilization"]}
                        contentStyle={{
                          backgroundColor: "#1c2e4a",
                          borderColor: "#304060",
                        }}
                        itemStyle={{ color: "#fff" }}
                      />
                    </PieChart>
                  </ResponsiveContainer>
                </div>
              </div>

              <div>
                <h3 className="text-sm text-white/70 mb-2">
                  CPU Usage Trend
                </h3>
                <div className="h-[150px]">
                  <ResponsiveContainer width="100%"
height="100%">
                    <LineChart data={data}>
                      <CartesianGrid strokeDasharray="3 3"
stroke="#304060" />
```

```
                    <XAxis dataKey="time" stroke="#fff" />
                    <YAxis stroke="#fff" />
                    <Tooltip
                      contentStyle={{
                        backgroundColor: "#1c2e4a",
                        borderColor: "#304060",
                      }}
                      itemStyle={{ color: "#fff" }}
                    />
                    <Line
                      type="monotone"
                      dataKey="cpu"
                      stroke="#0088FE"
                      activeDot={{ r: 8 }}
                      strokeWidth={2}
                    />
                  </LineChart>
                </ResponsiveContainer>
              </div>
            </div>
          </div>
        </Card>

        {/* Process Management */}
        <Card className="xl:col-span-2 bg-[#1c2e4a]/70
backdrop-blur-sm border-[#304060]">
          <div className="p-6">
            <h2 className="text-xl font-semibold mb-4 flex
items-center">
              <Cpu className="mr-2 h-5 w-5 text-[#00C49F]" />
              Process Management
            </h2>
```

```
            <div className="rounded-md border border-[#304060]
overflow-hidden">
              <Table>
                <TableHeader className="bg-[#0c1829]">
                  <TableRow>
                    <TableHead className="text-white
font-medium">
                      Process ID
                    </TableHead>
                    <TableHead className="text-white
font-medium">
                      Status
                    </TableHead>
                    <TableHead className="text-white
font-medium">
                      Resources
                    </TableHead>
                    <TableHead className="text-white
font-medium w-[100px]">
                      Actions
                    </TableHead>
                  </TableRow>
                </TableHeader>
                <TableBody>
                  {processes.map((process) => (
                    <TableRow key={process.id}
className="border-[#304060]">
                      <TableCell className="font-medium">
                        {process.id}
                      </TableCell>
                      <TableCell>
                        <Badge
                          className={`${
                            process.status === "Running"
```

```
                                ? "bg-green-500/20
text-green-500"
                                : process.status === "Waiting"
                                ? "bg-yellow-500/20
text-yellow-500"
                                : "bg-red-500/20 text-red-500"
                            }`}
                          >
                            {process.status}
                          </Badge>
                        </TableCell>
                        <TableCell>
                          <div className="flex flex-wrap gap-1">
                            {process.resources.map((resource) =>
(
                              <Badge
                                key={resource}
                                variant="outline"
                                className="border-[#304060]"
                              >
                                {resource}
                              </Badge>
                            ))}
                          </div>
                        </TableCell>
                        <TableCell>
                          <Button
                            variant="ghost"
                            size="sm"
                            onClick={() =>
removeProcess(process.id)}
                            className="text-white/70
hover:text-white hover:bg-red-500/20"
                          >
```

```
                        <X className="h-4 w-4" />
                      </Button>
                    </TableCell>
                  </TableRow>
                ))}
              </TableBody>
            </Table>
          </div>
        </div>
      </Card>

      {/* Resource Allocation Stats */}
      <Card className="bg-[#1c2e4a]/70 backdrop-blur-sm
border-[#304060]">
        <div className="p-6">
          <h2 className="text-xl font-semibold mb-4 flex
items-center">
            <HardDrive className="mr-2 h-5 w-5
text-[#FFBB28]" />
            Resource Allocation
          </h2>

          <div className="space-y-4">
            {Object.entries(resourceAllocation).map(
              ([resource, processes]) => (
                <div
                  key={resource}
                  className="bg-[#0c1829] p-3 rounded-md
border border-[#304060]"
                >
                  <div className="flex justify-between
items-center mb-2">
                    <h3 className="font-medium
text-white">{resource}</h3>
```

```
                      <Badge className="bg-[#304060]
text-white">
                        {processes.length}{" "}
                        {processes.length === 1 ? "process" :
"processes"}
                      </Badge>
                    </div>
                    <div className="flex flex-wrap gap-1">
                      {processes.map((processId) => (
                        <Badge
                          key={processId}
                          className={`${
                            deadlockCycle.includes(processId)
                              ? "bg-red-500/20 text-red-500"
                              : "bg-blue-500/20 text-blue-500"
                          }`}
                        >
                          {processId}
                        </Badge>
                      ))}
                    </div>
                  </div>
                )
              )}

              {Object.keys(resourceAllocation).length === 0 &&
(
                <div className="bg-[#0c1829] p-4 rounded-md
border border-[#304060] text-center text-white/70">
                  <Info className="mx-auto h-8 w-8 mb-2
text-white/50" />
                  <p>No resources allocated yet.</p>
                </div>
              )}
```

```
            </div>
          </div>
        </Card>
      </div>

      {/* Footer */}
      <footer className="mt-8 text-center text-white/50
text-sm">
        <p>
          AI Deadlock Resolver Dashboard &copy; {new
Date().getFullYear()}
        </p>

      </footer>
    </motion.div>
  </div>
 );
}
```