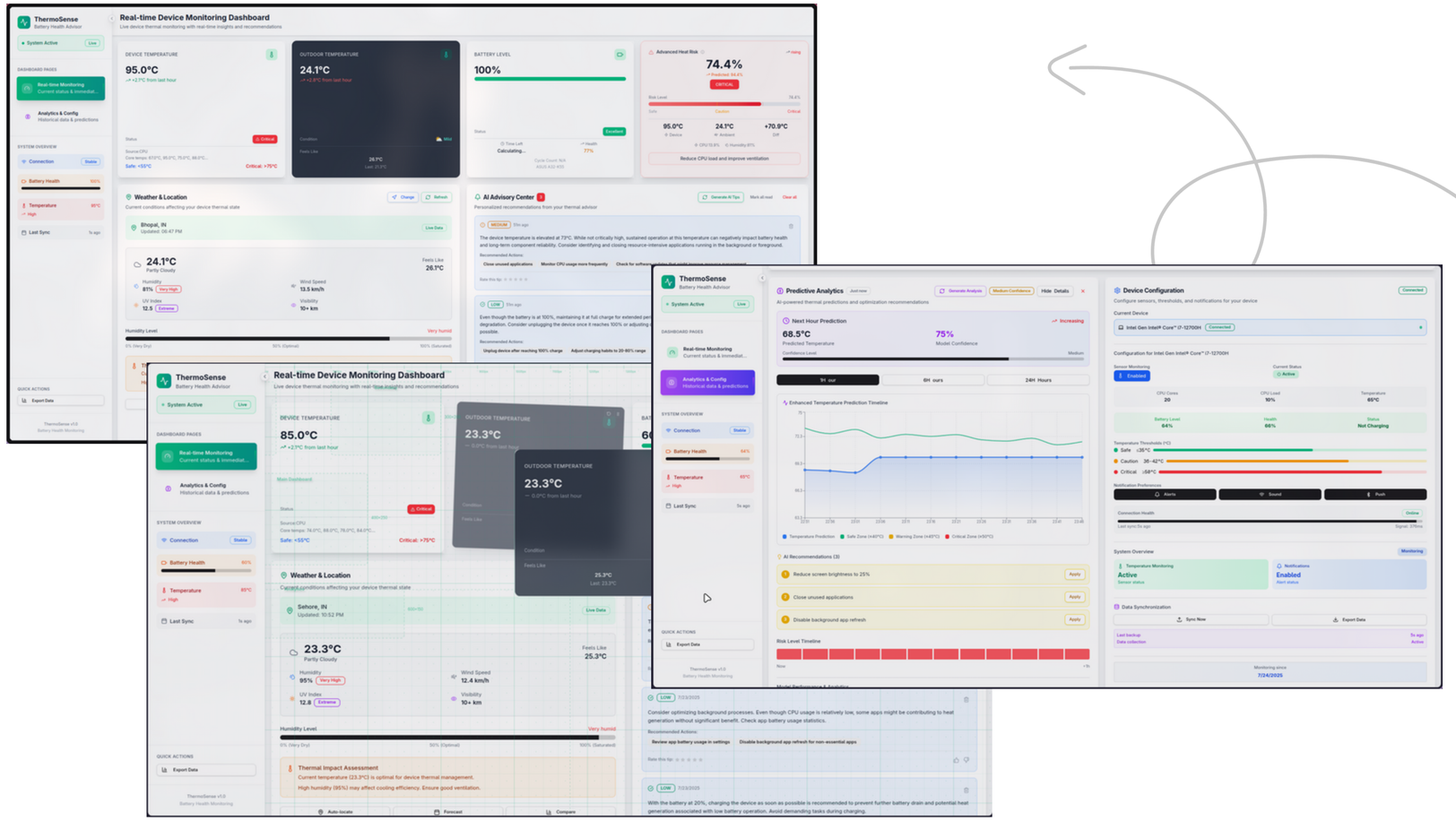
PROBLEM STATEMENT #6

THERMOSENSE

An intelligent thermal management system that helps users understand how ambient weather affects their device's thermal state

NEXT →



github.com/parazeeknova/ThermoSense

```
// Risk factors with weights
const weights = {
  tempDiff: 0.3,       // Device vs ambient temperature
  absoluteTemp: 0.25,  // Absolute device temperature
  cpuLoad: 0.2,        // Current CPU utilization
  trend: 0.15,         // Temperature trend direction
  humidity: 0.05,      // Environmental humidity
  timeOfDay: 0.05      // Peak usage hours
}
```

RISK FACTOR

The heat risk calculation in ThermoSense uses a weighted scoring system to assess the likelihood of device overheating. Each risk factor such as the difference between device and ambient temperature, the absolute device temperature, CPU load, temperature trend, humidity, and time of day is assigned a specific weight based on its impact on thermal risk. For example, the temperature difference and absolute temperature are the most influential, while environmental humidity and peak usage hours have smaller but still significant effects. By multiplying each factor by its weight and summing the results, the system produces a comprehensive risk score that reflects both immediate and environmental influences on device temperature.

github.com/parazeeknova/ThermoSense#risk-assessment-algorithm

```ts
// src/hooks/use-device-info.ts
export function useDeviceInfo() {
  const queryClient = useQueryClient()
  const query = useQuery({
    queryKey: deviceInfoKeys.current(),
    queryFn: fetchDeviceInfo,
    refetchInterval: 2000,
    staleTime: 1000,
    gcTime: 5000,
    retry: 3,
    retryDelay: (attemptIndex) => Math.min(1000 * 2 ** attemptIndex, 30000),
  })
  useEffect(() => {
    if (query.data?.timestamp) {
      queryClient.invalidateQueries({
        queryKey: ['historical-data'],
        exact: false,
      })
    }
  }, [query.data?.timestamp, queryClient])
  return query
}
```

REAL-TIME DEVICE MONITORING HOOK

This custom React hook is the backbone of ThermoSense's real-time monitoring. It fetches device information (like CPU temperature, battery, and load) every 2 seconds, ensuring the dashboard always displays up-to-date data. It uses caching and retry logic for resilience and performance, and coordinates with other data sources by invalidating related caches when new data arrives.

github.com/parazeeknova/ThermoSense#1-real-time-device-monitoring-hook

```
// src/components/dashboard/cards/heat-risk-meter.tsx
const calculateAdvancedRisk = (
  deviceTemp: number,
  ambientTemp: number,
  cpuLoad: number = 50,
  history: TemperatureHistory[] = [],
  humidity: number = 50,
): RiskCalculation => {
  const tempDiff = deviceTemp - ambientTemp

  // Calculate trend from historical data
  let trend = 0
  if (history.length >= 3) {
    const recent = history.slice(-3)
    const tempTrend = (recent[2].temperature - recent[0].temperature) / 2
    trend = Math.max(-10, Math.min(10, tempTrend))
  }

  // Risk factors calculation
  const factors: RiskFactors = {
    tempDifference: Math.min(100, Math.max(0, (tempDiff / 35) * 100)),
    absoluteTemp: Math.min(100, Math.max(0, (deviceTemp - 20) / 60 * 100)),
    cpuLoad,
    trend: Math.min(100, Math.max(0, (trend + 10) / 20 * 100)),
    humidity,
    timeOfDay: currentHour >= 12 && currentHour <= 18 ? 20 : 0,
  }

  // Weighted scoring system
  const weights = {
    tempDiff: 0.3,
    absoluteTemp: 0.25,
    cpuLoad: 0.2,
    trend: 0.15,
    humidity: 0.05,
    timeOfDay: 0.05,
  }

  const totalRisk = Object.entries(factors).reduce((sum, [key, value]) => {
    return sum + (value * weights[key as keyof typeof weights])
  }, 0)

  return {
    risk: totalRisk < 25 ? 'low' : totalRisk < 50 ? 'medium' : totalRisk < 75 ?
'high' : 'critical',
    value: Math.round(totalRisk),
    breakdown: factors,
    prediction: deviceTemp + (trend * 0.5)
  }
}
```

ADVANCED HEAT RISK CALCULATION

This function calculates the device's heat risk using a weighted scoring system. It considers the difference between device and ambient temperature, absolute temperature, CPU load, temperature trends, humidity, and time of day. The result is a risk level (low, medium, high, critical) and a prediction for future temperature, providing both immediate and forward-looking insights.

github.com/parazeeknova/ThermoSense#3-advanced-heat-risk-calculation-core

```
// src/lib/gemini-service.ts
async generatePredictiveAnalytics(context: DeviceContext): Promise<PredictiveData> {
  const prompt = `
You are a predictive analytics AI for thermal management. Based on the current
device and environmental conditions, predict future temperature trends and provide
optimization recommendations.
Current Conditions:
- Device Temperature: ${context.deviceTemp || 'unknown'}°C
- Battery Level: ${context.batteryLevel || 'unknown'}%
- Ambient Temperature: ${context.weatherTemp || 'unknown'}°C
- CPU Usage: ${context.cpuUsage || 'unknown'}%
- Screen Brightness: ${context.screenBrightness || 'unknown'}%
- Active Applications: ${context.activeApps || 'unknown'}
Please respond in the following JSON format:
{
  "nextHourTemp": 45.2,
  "riskTrend": "increasing|decreasing|stable",
  "recommendedActions": ["action1", "action2", "action3"],
  "confidence": 85
}
`
  const response = await this.callGemini(prompt)
  const parsed = JSON.parse(response.replace(/```json\n?|\n?```/g, ''))
  return {
    nextHourTemp: parsed.nextHourTemp,
    riskTrend: parsed.riskTrend as 'increasing' | 'decreasing' | 'stable',
    recommendedActions: parsed.recommendedActions,
    confidence: Math.min(Math.max(parsed.confidence, 0), 100),
  }
}
```

PREDICTIVE ANALYTICS WITH AI (SIMPLE)

This function uses AI to predict the device's temperature trend for the next hour, assess the risk trend, and suggest optimization actions. It provides a confidence score, helping users understand the reliability of the prediction and take proactive steps to prevent overheating.

github.com/parazeeknova/ThermoSense#2-ai-powered-recommendations-service

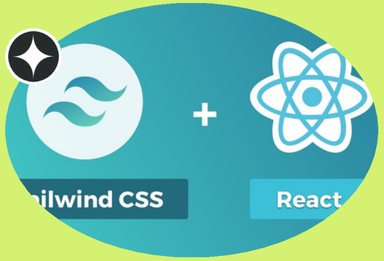
```tsx
// src/components/dashboard/dashboard.tsx (core drag-and-drop logic)
const [draggedCard, setDraggedCard] = useState<string | null>(null)
const [dragOverTarget, setDragOverTarget] = useState<string | null>(null)

const handleDragStart = (id: string) => setDraggedCard(id)
const handleDragEnd = () => {
  setDraggedCard(null)
  setDragOverTarget(null)
}
const handleDragOver = (e: React.DragEvent) => {
  e.preventDefault()
  const target = e.currentTarget as HTMLElement
  const cardId = target.closest('[data-card-id]')?.getAttribute('data-card-id')
  if (cardId && cardId !== draggedCard) setDragOverTarget(cardId)
}
const handleDrop = (e: React.DragEvent, targetId: string) => {
  e.preventDefault()
  setDragOverTarget(null)
  if (draggedCard && draggedCard !== targetId) {
    const newOrder = [...currentCardOrder]
    const draggedIndex = newOrder.findIndex(card => card.id === draggedCard)
    const targetIndex = newOrder.findIndex(card => card.id === targetId)
    if (draggedIndex !== -1 && targetIndex !== -1) {
      const [draggedItem] = newOrder.splice(draggedIndex, 1)
      newOrder.splice(targetIndex, 0, draggedItem)
      setCardOrder(newOrder)
    }
  }
}
```

DRAG & DROP (WIDGETS)

ThermoSense's dashboard supports intuitive drag-and-drop reordering of cards. When a user starts dragging a card, its ID is stored in state. As the card is dragged over other cards, the target is tracked. On drop, the system updates the card order by removing the dragged card from its original position and inserting it at the new location. This logic, combined with the DraggableCard component, enables a smooth, interactive, and customizable dashboard experience for users.

THERMOSENSE TECH STACK

NEXTJS →



FRONTEND

React: UI library
Next.js: React framework for server-side rendering and API routes
Tailwind CSS & Typescript

BACKEND / APIS

Next.js API Routes: For device info, AI, and openweather endpoints
systeminformation (Node.js library): For real-time device hardware data collection

STATE & DATA MANAGEMENT

TanStack Query: For efficient data fetching, caching, and synchronization

OTHER

Docker: Containerization for easy deployment and development
Bun or npm: Package management and scripts