8D Problem Solving Platform - Technical Documentation

Project: 8D Problem Solving Platform (MVP)

Developer: Ahmet Boy

Date: December 2024

Live Demo: https://8dx.gt.tc/

Github: https://github.com/AhmetBoy/8dx_task

Tech Stack: React + Siemens iX + PHP + MySQL

1. Project Overview

Introduction

This project is a Full-Stack MVP application that digitalizes the **8D Problem Solving Methodology** used in manufacturing environments. It focuses on problem tracking and root cause analysis using a tree-based hierarchical approach.

Project Scope

The application implements the following 8D methodology stages:

- D1-D2: Problem definition and team assignment
- D4: Root cause analysis using 5 Why method with unlimited depth tree structure
- D5: Permanent corrective action planning

Key Features

✅ **Dashboard**: AG-Grid based problem list with filtering and sorting

✅ **Problem Management**: Full CRUD operations via modal forms

✅ **Dynamic Root Cause Tree**: Unlimited depth hierarchical analysis

✅ **5 Why Method**: Visual tree structure with parent-child relationships

✅ **Root Cause Marking**: Ability to mark any node as root cause

✅ **Action Planning**: Permanent solution input for root causes

✅ **Responsive Design**: Mobile and desktop support

✅ **Theme Support**: Dark/Light mode with localStorage persistence

✅ **Production Ready**: Deployed on InfinityFree hosting

Technology Stack

Frontend:

- React 18.2.0 - UI framework
- Siemens iX 2.0.0 - Design system (Required)
- AG-Grid React 31.0.0 - Data grid
- React Router 6.20.0 - SPA routing
- Axios 1.6.2 - HTTP client
- Vite 5.0.8 - Build tool

Backend:

- PHP (Native) - Backend language
- PDO - Database abstraction layer
- MySQL - Relational database
- RESTful API - JSON responses

Deployment:

- Local: XAMPP (Apache + MySQL)
- Production: InfinityFree hosting

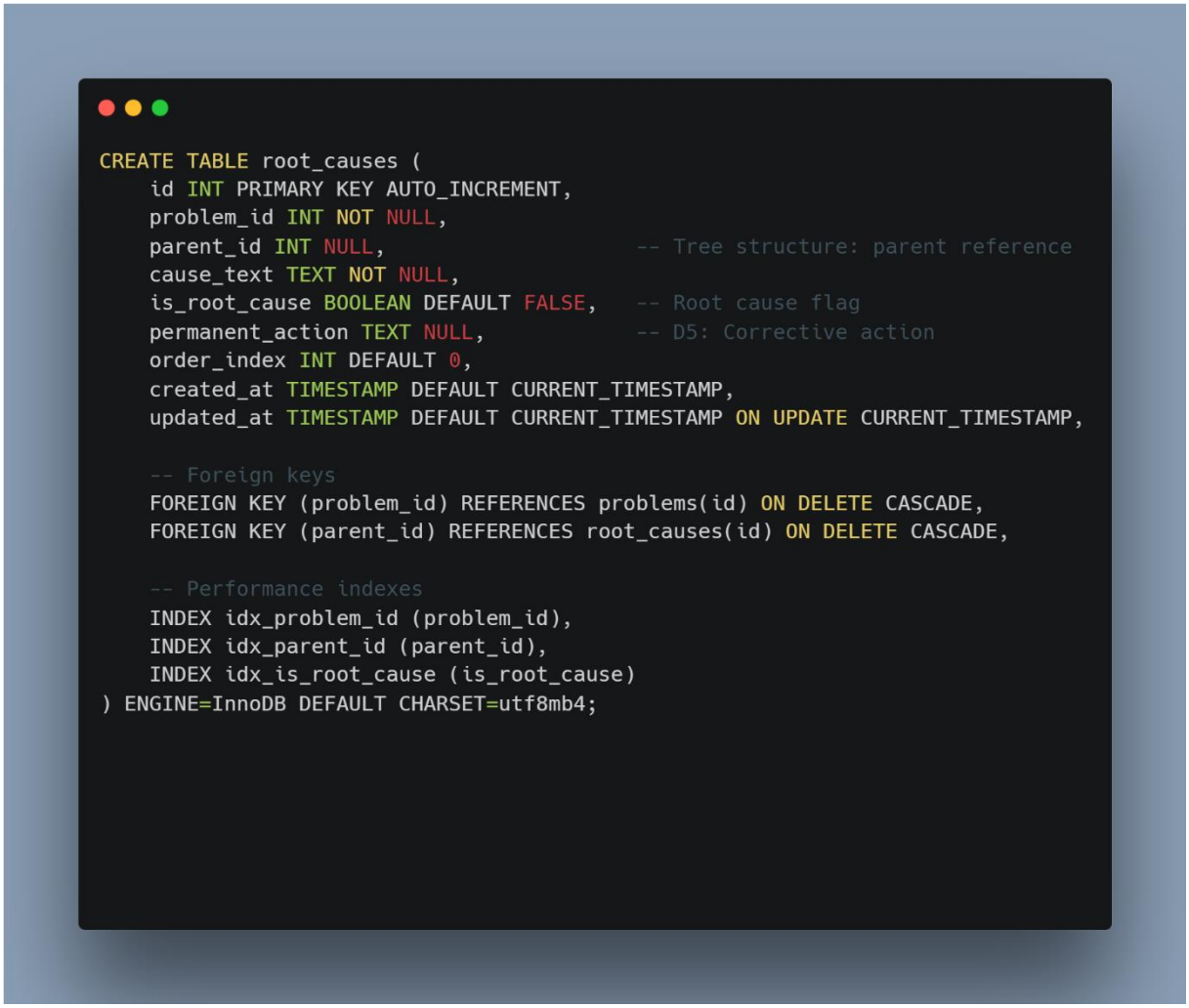
Şekil 1 Project File Schema

2. Data Model- Tree/Recursive Structure

The core challenge of this project is modeling the **5 Why analysis** as an unlimited depth tree structure in a relational database. We use the **Adjacency List pattern** with a `parent_id` foreign key to create recursive relationships.

Database Schema

`root_causes` Table:



```
CREATE TABLE root_causes (
    id INT PRIMARY KEY AUTO_INCREMENT,
    problem_id INT NOT NULL,
    parent_id INT NULL,                    -- Tree structure: parent reference
    cause_text TEXT NOT NULL,
    is_root_cause BOOLEAN DEFAULT FALSE,   -- Root cause flag
    permanent_action TEXT NULL,            -- D5: Corrective action
    order_index INT DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,

    -- Foreign keys
    FOREIGN KEY (problem_id) REFERENCES problems(id) ON DELETE CASCADE,
    FOREIGN KEY (parent_id) REFERENCES root_causes(id) ON DELETE CASCADE,

    -- Performance indexes
    INDEX idx_problem_id (problem_id),
    INDEX idx_parent_id (parent_id),
    INDEX idx_is_root_cause (is_root_cause)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Şekil 2 Database Root Causes Schema

Tree Structure Logic

Key Concepts:

- `parent_id = NULL` → **Root node** (top-level cause)

- `parent_id = X` → **Child node** (sub-cause of node X)

- Unlimited depth: Any node can be a parent of another node

- Cascade delete: Deleting a parent automatically deletes all children

Example Tree:

```
Problem: Machine Stopped
  ├── [id:1, parent_id:NULL] Fuse Blown
  │   ├── [id:3, parent_id:1] Overload
  │   │   └── [id:5, parent_id:3] No Maintenance (6 months) ⭐ ROOT CAUSE
  │   │       └── [id:7, parent_id:5] Budget Cuts
  │   └── [id:4, parent_id:1] Faulty Cable
  └── [id:2, parent_id:NULL] Operator Error
      └── [id:6, parent_id:2] No Training
```

Database Representation:

id	problem_id	parent_id	cause_text	is_root_cause
1	1	NULL	Fuse Blown	false
2	1	NULL	Operator Error	false
3	1	1	Overload	false
4	1	1	Faulty Cable	false
5	1	3	No Maintenance (6 months)	true
6	1	2	No Training	false
7	1	5	Budget Cuts	false

PHP Implementation - Recursive Tree Building

```php
<?php
class RootCause {
    private $conn;

    public function __construct($db) {
        $this->conn = $db;
    }
    public function getByProblemIdHierarchical($problemId) {
        // Step 1: Fetch all causes (flat list)
        $query = "SELECT * FROM root_causes
                  WHERE problem_id = :problem_id
                  ORDER BY order_index ASC, id ASC";
        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(':problem_id', $problemId);
        $stmt->execute();
        $allCauses = $stmt->fetchAll(PDO::FETCH_ASSOC);
        return $this->buildTree($allCauses, null);
    }
    private function buildTree($causes, $parentId = null) {
        $tree = [];
        foreach ($causes as $cause) {
            // Find causes with matching parent_id
            if ($cause['parent_id'] == $parentId) {
                // Recursively find children of this cause
                $cause['children'] = $this->buildTree($causes, $cause['id']);
                $tree[] = $cause;
            }
        }
        return $tree;
    }
    public function create($data) {
        $query = "INSERT INTO root_causes
                  (problem_id, parent_id, cause_text, order_index)
                  VALUES (:problem_id, :parent_id, :cause_text, :order_index)";
        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(':problem_id', $data['problem_id']);
        $stmt->bindParam(':parent_id', $data['parent_id']);
        $stmt->bindParam(':cause_text', $data['cause_text']);
        $stmt->bindParam(':order_index', $data['order_index']);
        return $stmt->execute();
    }
    public function update($id, $data) {
        $query = "UPDATE root_causes
                  SET is_root_cause = :is_root_cause,
                      permanent_action = :permanent_action
                  WHERE id = :id";
        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(':id', $id);
        $stmt->bindParam(':is_root_cause', $data['is_root_cause']);
        $stmt->bindParam(':permanent_action', $data['permanent_action']);

        return $stmt->execute();
    }
}
```

Şekil 3 Recursive Tree Building

Algorithm Complexity

- Time: O(n) - Single pass through all causes
- Space: O(n) - Stores all causes in memory
- Depth: Unlimited - Recursive algorithm handles any depth

Advantages

✅ **Flexibility**: Unlimited tree depth (5 Why, 10 Why, ∞)

✅ **Data Integrity**: Foreign key constraints + cascade delete

✅ **Performance**: Indexed queries for fast retrieval

✅ **Standard Pattern**: Adjacency List is widely used and understood

✅ **Simple Queries**: Easy to add/update/delete nodes

3. API Architecture

The API provides RESTful endpoints that return hierarchical JSON structures for the frontend to consume. The tree structure is built on the backend and sent as nested JSON with `children` arrays.

RESTful Endpoints

| GET | `/api/problems` | List all problems |

| POST | `/api/problems` | Create new problem |

| GET | `/api/problems/{id}` | Get problem details |

| PUT | `/api/problems/{id}` | Update problem |

| DELETE | `/api/problems/{id}` | Delete problem |

| **GET** | `/api/problems/{id}/causes` | **Get hierarchical tree** |

| POST | `/api/causes` | Create new cause |

| GET | `/api/causes/{id}` | Get cause details |

| PUT | `/api/causes/{id}` | Update cause (mark root) |

| DELETE | `/api/causes/{id}` | Delete cause |

Hierarchical JSON Response

Request: `GET /api/problems/1/causes`


```
{
  "success": true,
  "data": [
    {
      "id": 1,
      "problem_id": 1,
      "parent_id": null,
      "cause_text": "Fuse Blown",
      "is_root_cause": false,
      "permanent_action": null,
      "order_index": 1,
      "created_at": "2024-12-11 10:30:00",
      "children": [
        {
          "id": 3,
          "problem_id": 1,
          "parent_id": 1,
          "cause_text": "Overload",
          "is_root_cause": false,
          "permanent_action": null,
          "order_index": 1,
          "children": [
            {
              "id": 5,
              "problem_id": 1,
              "parent_id": 3,
              "cause_text": "No maintenance for 6 months",
              "is_root_cause": true,
              "permanent_action": "Implement monthly preventive maintenance
plan",
              "order_index": 1,
              "children": []
            }
          ]
        },
      ]
    },
  ]
}
```


Şekil 4 JSON Response

JSON Structure Features

✅ **Nested Children**: Each node contains a `children` array

✅ **Self-Contained**: Every node has complete information

✅ **Empty Arrays**: Leaf nodes have `children: []` for easier frontend handling

✅ **Metadata**: Includes timestamps, order, root cause status

✅ **Consistent Format**: Same structure at every level

4. Siemens iX Implementation

The project uses **Siemens iX Design System** (v2.0.0) as the UI library. All components follow Siemens iX standards and documentation (https://ix.siemens.io).

Components Used

| `**IxApplication**` | App container | App.jsx |

| `**IxApplicationHeader**` | Top header bar | App.jsx |

| `**IxMenu**` / `IxMenuItem` | Left sidebar navigation | App.jsx |

| `**IxModal**` ⭐ | Problem form (Required) | AddProblemModal.jsx |

| `**IxContentHeader**` | Page headers | All pages |

| `**IxCard**` / `IxCardContent` | Content containers | All pages |

| `**IxButton**` | Action buttons | All components |

| `**IxInputGroup**` | Form inputs | Modal forms |

| `**IxSpinner**` | Loading states | Dashboard |

| `**IxIcon**` | Icons and badges | CauseNode |

```
  return (
    <IxModal ref={modalRef} size="720">
      <IxModalHeader>Add New Problem</IxModalHeader>

      <IxModalContent>
        <form id="addProblemForm" onSubmit={handleSubmit}>
          {/* Title input */}
          <IxInputGroup label="Problem Title *">
            <input
              type="text"
              value={formData.title}
              onChange={(e) => setFormData({...formData, title:
e.target.value})}
              required
              placeholder="e.g., Machine Stopped - Line 2"
            />
          </IxInputGroup>

          {/* Description input (D2) */}
          <IxInputGroup label="Detailed Description (D2) *">
            <textarea
              value={formData.description}
              onChange={(e) => setFormData({...formData, description:
e.target.value})}
              required
              rows="4"
              placeholder="Detailed problem description..."
            />
          </IxInputGroup>

          {/* Responsible team input (D1) */}
          <IxInputGroup label="Responsible Team (D1) *">
            <input
              type="text"
              value={formData.responsible_team}
              onChange={(e) => setFormData({...formData, responsible_team:
e.target.value})}
              required
              placeholder="e.g., Maintenance Team"
            />
          </IxInputGroup>
        </form>
      </IxModalContent>

      <IxModalFooter>
        <IxButton outline onClick={() => modalRef.current?.close()}>
          Cancel
        </IxButton>
        <IxButton type="submit" form="addProblemForm">
          Save
        </IxButton>
      </IxModalFooter>
    </IxModal>
  );
}

export default AddProblemModal;
```

Şekil 5 Example Model Create With iX

Documentation References

- Layout: https://ix.siemens.io/docs/installation/react
- Modal: https://ix.siemens.io/docs/components/modal
- Buttons: https://ix.siemens.io/docs/components/button
- Cards: https://ix.siemens.io/docs/components/card
- Input Group: https://ix.siemens.io/docs/components/input-group
- Theme: https://ix.siemens.io/docs/theming

Conclusion

Technical Achievements

✅ **Tree Data Model**: Implemented unlimited depth tree structure using Adjacency List pattern

✅ **Recursive Algorithm**: PHP recursive function to build hierarchical JSON from flat data

✅ **RESTful API**: Clean API architecture with nested JSON responses

✅ **Siemens iX**: Proper implementation of 10+ Siemens iX components following official documentation

✅ **Recursive UI**: React recursive rendering for tree visualization

✅ **Full-Stack MVP**: Complete end-to-end implementation from database to UI

Project Metrics

- Backend: 9 RESTful endpoints, 2 controllers, 2 models
- Frontend: 8 React components, recursive rendering
- Database: 2 tables with tree structure, 6 indexes
- Siemens iX: 10+ components used according to standards

Live Demo

URL: https://8dx.gt.tc/

Project Version: 1.0.0 (MVP)

Date: December 2024

Documentation Version: 1.0