# AI Project Report

## 1. Project Overview

The Automatic Deadlock Detection System can find and resolve the deadlocks of concurrent systems as a result of the process-resource interaction. It uses the Wait-For Graph and Banker's algorithm for deadlock detection and resolution, illustrates process relations as graphs for interaction, and provides recovery actions such as process termination and resource preemption as necessary. This system serves as a guide and teaching material to equip system administrators dealing with real-time deadlocks while concurrently acting as an operating system concepts learning tool for students.

## 2. Module-Wise Breakdown

The project includes a couple of modules made for an essential component for deadlock detection, visualization, security, and user interaction.

### ****1.**** Core Deadlock Detection

Module For handling detection, it will also leverage graph algorithms.

#### ****Capabilities:****

* It constructs and analyzes the Wait For Graph (WFG).
* Uses cycle detection algorithms (DFS) to determine the occurrences of deadlock.
* Confirms real manifest deadlocks by detected cycles.

#### ****Appendix:****

* Deadlock Detection Algorithms
* Logical Flow Explanation.
* Code Implementation.

### ****2.**** Visualization Module

This feature offers an interactive graph-based visualization of your deadlock scenarios.

#### ****Capabilities:****

* Shows processes (nodes) and resource relationships (edges).
* Shows deadlocked processes in red for easy visibility.
* Enables zoom, pan, and hover interactions to help with analysis.

#### ****Appendix:****

* UI Design Breakdown
* Graph Rendering Approaches
* Interactive Features

### ****3.**** Simulation & User Interface Module

This module allows users to create and test their personalized deadlock scenarios.

#### ****Capabilities:****

* Allows users to define processes, resources, and their allocations.
* Provides a step-through method to observe the formation of deadlock.
* Show users detection results, statistics, and suggested resolutions.

#### ****Appendix:****

* UI Components and Widgets
* Event Handling
* Simulation Logic

### ****4.**** Security & File Management Module

This module ensures secure storage, user authentication, and access control.

#### ****Capabilities:****

* Implements bcrypt hashing for password encryption.
* Provides user registration and authentication along with role-based access control (RBAC).
* Admins manage users, encryption keys, and all stored data.
* Encrypts and stores process-resource allocation logs securely.

#### ****Appendix:****

* Authentication Mechanisms
* Role-Based Access Control Implementation
* Secure File Storage Methods

## 3. Functionalities

The "HydrationTracker" project includes multiple core functionalities:

* **DeadLock Detection:** Identifies deadlocks using Wait-For Graph and cycle detection algorithms (DFS, Banker’s Algorithm).
* **Graphical Visualization:** Displays process-resource dependencies with interactive graphs and deadlock highlighting.
* **Scenario Simulation:** Allows users to define, execute, and analyze custom deadlock scenarios step by step.
* **Recovery suggestions:** Recommends solutions like process termination or resource preemption.
* **User authentication & security:** Implements secure login, role-based access control (RBAC), and encrypted data storage.

## 4. Technology Used

### ****Programming Languages:****

* **Python** (for core logic and backend processing)

### ****Libraries and Tools:****

* **Tkinter:** Used for GUI development.
* **Requests:** Fetches weather data from APIs.
* **OS & JSON:** Manages configuration and data storage.

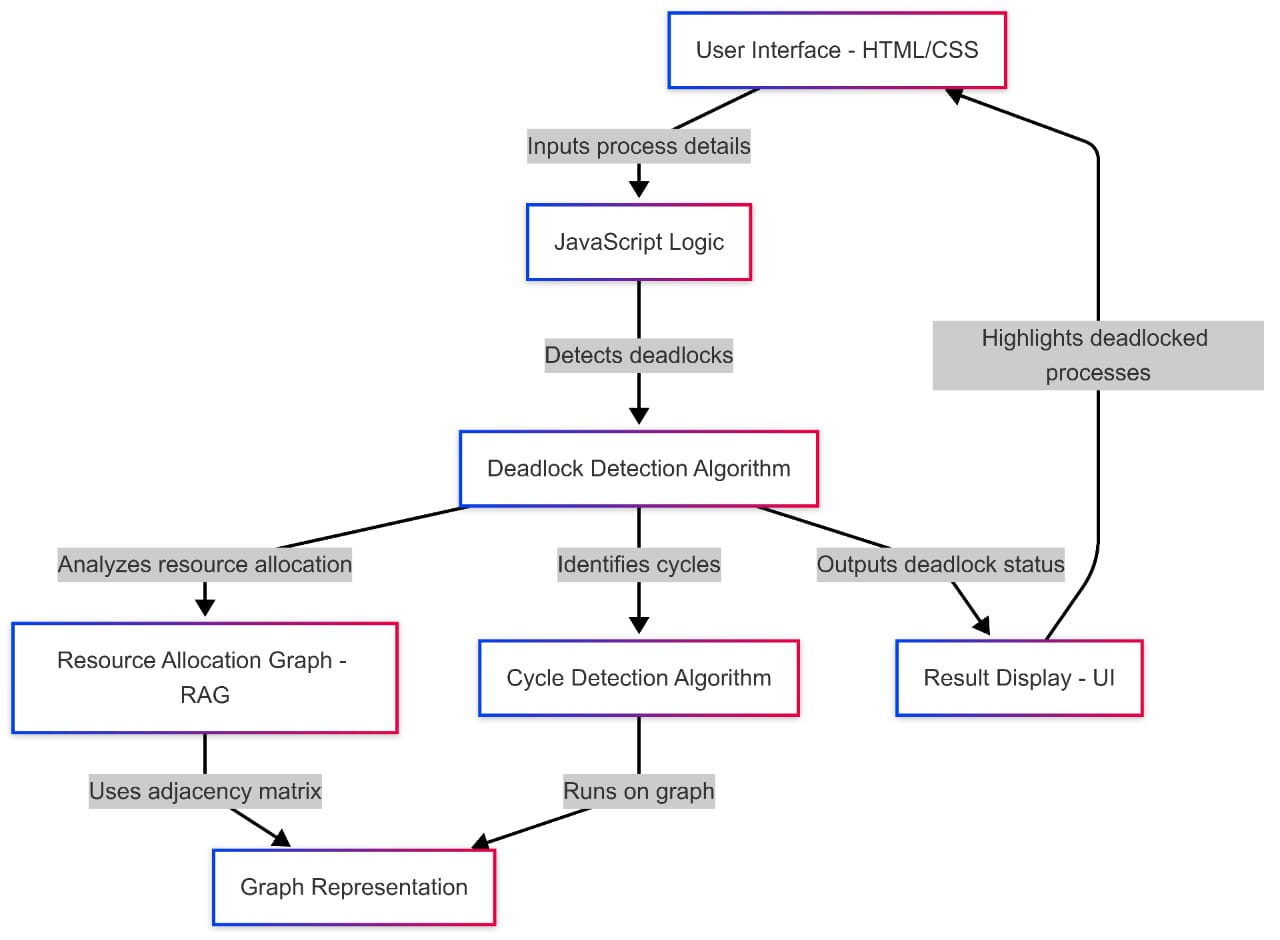
### ****Recommended Technologies:****

* **SQLite or Firebase** (for structured database storage)
* **Flask/Django** (for developing a web-based version)
* **React Native** (for mobile app development)

## 5. Flow Diagram

The project follows a structured flow:

1. User inputs process-resource allocations and requests..
2. The system constructs a Wait-For Graph (WFG).
3. Deadlock detection algorithms (DFS, Banker’s Algorithm) analyze the graph.
4. If a cycle is found, it verifies the deadlock.
5. The system highlights deadlocked processes in a graphical representation.
6. Recovery suggestions (process termination, resource preemption) are provided.
7. The user selects an action to resolve the deadlock.
8. The system updates the state and logs the results for future analysis.



## 6. Execution Plans (Step-wise Breakdown)

### ****Step 1:**** Research & Design

* Study deadlock detection algorithms (Wait-For Graph, Banker’s Algorithm).
* Design data structures for process-resource management.
* Create system architecture and UI wireframes.

### ****Step 2:**** Core Detection Engine

* Implement process and resource management logic.
* Develop the Wait-For Graph and integrate cycle detection (DFS).
* Validate detection with test cases (simple and complex deadlocks).

### ****Step 3:**** Visualization System

* Implement interactive graph visualization using Matplotlib/Plotly.
* Highlight deadlocked processes in real-time.
* Add zoom, pan, and hover functionality for better analysis.

### ****Step 4:**** Simulation & User Interface

* Develop a GUI for inputting process-resource scenarios.
* Integrate step-by-step execution for deadlock analysis.
* Provide real-time feedback and detailed explanations.

### ****Step 5:**** Testing & Optimization

* Test with edge cases (no deadlock, multiple deadlocks, high resource contention).
* Optimize performance for large-scale process-resource graphs.
* Gather user feedback for refinements.

### ****Step 6:**** Deployment & Documentation

* Package the system into an executable (PyInstaller).
* Write a user guide and technical documentation.
* Record demo videos and tutorials.

## 7. Conclusion and Future Scope

The Automatic Deadlock Detection System quickly detects and recovers from deadlocks with algorithmic detection, visualization and simulations. Its modular and scalable design makes the system easy to analyze and perform deadlock recovery. This makes it an excellent tool for developers, administrators, and students.

* **Real-time Detection:** Extend the system to monitor live processes and detect deadlocks dynamically.
* **Integration with Operating Systems:** Enhance compatibility with OS-level process management.
* **Advanced Recovery Strategies:** Implement priority-based resolution methods for better system stability.
* **Machine Learning for Prediction:** Use AI to predict potential deadlocks before they occur.
* **Web-based Interface:** Develop a cloud-based version for remote deadlock analysis and monitoring.
* **Enhanced Visualization:** Improve graphical representations with interactive 3D models and animations.

## 8.Source Code:

