Task: N-Queens Problem Solver

Task Description

The **N-Queens problem** is a classic **backtracking algorithm challenge** where you must place `N` queens on an `N×N` chessboard **so that no two queens attack each other**.

Your task is to **implement the N-Queens solver in JavaScript** and create a **React UI** that:

Allows users to **input `N` (size of the board)**
Finds a solution using the **Backtracking Algorithm**
Displays the solution visually on a chessboard
Handles edge cases (like `N=2` and `N=3`, which have no solutions)

Requirements

1. Implement **a function to solve the N-Queens problem** using **backtracking**.
2. Create a **React UI** to input the board size (`N`).
3. Show the **chessboard with queens placed correctly**.
4. Allow users to **see different solutions** if multiple exist.

Steps to Follow

Understand the N-Queens Problem

- The goal is to **place N queens** on an `N×N` board such that:
  No two queens are in the **same row**
  No two queens are in the **same column**
  No two queens are in the **same diagonal**

Implement the Backtracking Algorithm

- Start placing queens **row by row**.
- If a queen can be placed **safely**, move to the next row.
- If no safe spot is found, **backtrack** and try another position.
- Continue until all `N` queens are placed successfully.

Build the React UI

- Add an **input box** for the user to enter `N`.
- Display the **chessboard dynamically** based on `N`.

- Highlight **queen positions** in the solution.
- Show a **"Solve" button** to trigger the algorithm.

Example Walkthrough

For `N = 4`

A possible solution:

```
css
CopyEdit
. Q . .
. . . Q
Q . . .
. . Q .
```

This means:

- The **first queen** is placed at `(0,1)`.
- The **second queen** is placed at `(1,3)`.
- The **third queen** is placed at `(2,0)`.
- The **fourth queen** is placed at `(3,2)`.