Technical Documentation: Sudoku Application in Angular

Tommaso Spadaro

November 11, 2025

Contents

1 Introduction

This documentation describes the architecture and functionality of a web application for the Sudoku game, developed using the Angular framework. The application is structured into components, manages a complex game state, and implements backtracking algorithms both for puzzle generation and solving.

The user can interact with the application in two main modes:

- **Play Mode ('play')**: The user plays with a randomly generated puzzle.
- **Creation Mode ('create')**: The user can build their own puzzle and check if it is solvable.

2 Component Architecture

The application is divided into three main components that collaborate to create the user experience.

2.1 AppComponent

This is the root component (`app-root`) of the application.

- **Logic:** Serves as the main container.
- **Template:** Loads the `app-sudoku` component, sets a page title, and displays a footer.

2.2 SudokuComponent

This is the "brain" of the application (`app-sudoku`), orchestrating most of the business logic.

- **State Management:** Maintains the current grid state (`sudoku`), the active cell (`activeField`), the current mode (`currentMode: 'play' | 'create'`), the note mode (`noteMode`), and progress (`progress`).
- **Game Logic:** Contains methods for all user actions, including:
  - `onPlayClick()`: Generates a new puzzle.
  - `onCreateClick()`: Resets the grid for puzzle creation.
  - `insertNumber()`: Inserts a number or a note into the active cell.
  - `erase()`: Erases the contents of the active cell.
  - `hint()`: Reveals the solution for the active cell (only in 'play' mode).
  - `solveSudoku()`: Solves the entire puzzle (used in 'create' mode).
- **Algorithms:** Implements logic for:
  - **Puzzle Generation:** `generateFullSolution()` and `createPuzzleFromSolution()` use backtracking to create a valid puzzle.
  - **Solvability Check:** `checkSolvability()` and `solveWithSets()` use an optimized backtracking algorithm (with `Set`) to determine if the current puzzle has at least one valid solution.
- **Interaction:** Handles keyboard input (`@HostListener`) for numbers, backspace, and the spacebar (to toggle note mode).

2.3 GridComponent

A child component (`su-grid`), purely presentational, responsible for rendering the 9x9 grid.

- **Input/Output:** Receives the grid `sudoku` and `noteMode` as Inputs. Communicates the selected cell to the parent via Output (using two-way binding `[(activeField)]`).
- **Rendering:** Draws the 9x9 cells.
- **Dynamic Styling:** Applies dynamic CSS classes to:
  - Highlight thick borders for 3x3 boxes.
  - Mark `readonly` cells (initial clues).
  - Highlight the row, column, and cells with the same number as the active cell.
  - Indicate the state of the active cell (note or input mode).
- **Note Rendering:** If a cell has no `value` but has a `notes` array, it renders a 3x3 mini-grid showing the note numbers.

3 Data Models

The data structures are defined in the file `sudokuField.component.ts`.

- `SudokuField`: Interface defining a single cell in the grid.
  - `value?: number`: The number (1–9) entered by the user.
  - `notes?: number[]`: An array of numbers representing notes.
  - `answer: number`: The correct solution value for that cell.
  - `readonly?: boolean`: If `true`, the cell is an initial clue and cannot be modified.
- `Sudoku`: Defines the entire grid as a two-dimensional array of `SudokuField`: `SudokuField[][]`

Note: The file `cell.ts` defines a `Cell` class, but it does not appear to be actively used by the main components, which instead rely on the `SudokuField` interface.

4 Main Features

4.1 Play Mode ('play')

- At startup, a random puzzle with a predefined difficulty is generated.
- The user can enter numbers or notes (toggled via the "Note" button or spacebar).
- The function `cleanNotes()` automatically removes conflicting notes from the same row, column, or box when a number is inserted.
- A progress bar (`progress`) shows the percentage of correctly filled cells.
- The `hint()` button fills the active cell with the correct answer.
- Number buttons (1–9) are disabled when a number has already been placed 9 times on the grid.
- Upon completion (`progress === 100`), the message "You win!" is displayed.

4.2 Creation Mode ('create')

- The user starts from an empty grid.
- With every number inserted, the application runs a background solving algorithm (`checkSolvability()`) to check if the puzzle still has at least one valid solution.
- A message on screen informs the user in real time whether the puzzle is "valid" (has solutions) or "invalid" (has none).
- A "Solve" button uses the `solveSudoku()` algorithm to display a possible solution for the user's custom puzzle.