1 Technical Solution

1.1 Server

1.1.1 Place cell

This code generates the completed Sudoku grid recursively by first shuffling a list of numbers 1 to 9. Then it iterates through the list, and it will place the first valid number it reaches. If there is no legal place-able number for that tile, it will backtrack and continue from where it left off on a previous tile

```
def place_cell(sudoku_grid, c=0):
    column_number, row_number = divmod(c, 9)  # returns column,row
    numbers = [count for count in range(1, 10)]
    random.shuffle(numbers)
    for Number in numbers:
        if ((Number not in sudoku_grid[round_(c, 9):round_(c, 9) + 9]) and
                (Number not in sudoku_grid[row_number::9]) and
                all(Number not in sudoku_grid[round_(row_number, 3) + ((round_(column_number, 3) + count) * 9):
                    round_(row_number, 3) + 3 + ((round_(column_number, 3) + count) * 9)]
                    for count in range(0, 3))):  # checks grid
            sudoku_grid[c] = Number
            if c + 1 >= 81 or place_cell(sudoku_grid, c + 1):
                return sudoku_grid
    else:
        sudoku_grid[c] = None
        return None
```

```
def generate_key(grid):
    full_list = []
    check_list = []
    for Index_Tile in range(0, 9):
        for Row in range(0, 3):
            for Tile_Row in range(3):
                check_list[Row * 9 + Tile_Row * 3:Row * 9 + (Tile_Row + 1) * 3] = \
                    grid[(((Index_Tile // 3) + Row) * 9 + Tile_Row * 3 + Index_Tile % 3) * 3:
                         (((Index_Tile // 3) + Row) * 9 + Tile_Row * 3 + 1 + Index_Tile % 3) * 3]
        for Column in range(1, 3):
            for Tile_Row in range(3):
                check_list[27 + (Column - 1) * 9 + Tile_Row * 3:27 + (Column - 1) * 9 + (Tile_Row + 1) * 3] = \
                    grid[((Index_Tile // 3) * 9 + Tile_Row * 3 + (Index_Tile + Column) % 3) * 3:
                         ((Index_Tile // 3) * 9 + Tile_Row * 3 + 1 + (Index_Tile + Column) % 3) * 3]
        check_lists = [check_list,
                       [check_list[9:18] + check_list[:9] + check_list[18:]][0],
                       [check_list[18:27] + check_list[:18] + check_list[27:]][0],
                       [check_list[27:36] + check_list[:27] + check_list[36:]][0],
                       [check_list[36:] + check_list[:36]][0]]
        tile_indexed = []
        for StartIndex in range(len(check_lists)):
            index = [x for x in range(1, 10)]
            for _ in itertools.repeat(None, 3):
                change_dict = {index[x]: x_value for x, x_value in enumerate(check_lists[StartIndex][:9])}
                tile_indexed.append([change_dict[int(x)] for x in check_lists[StartIndex][9:]])
                index = rotation(index, 90, 2)
        full_list.append(tile_indexed)
    return full_list
    }
```
