Reviewed by: Demetria Mack ID: @02927552
Feedback: I think this code is very well written, variable/function names are descriptive of what they are supposed to do. The use of a trie was a very good touch and its implementation was clear and concise. Only recommendation I would make and this is subjective to me is to give a one-liner explanation of what each function's job is.

```
/**
 * Given a Boggle board and a dictionary, returns a list of available words in
 * the dictionary present inside of the Boggle board.
 * @param {string[][]} grid - The Boggle game board.
 * @param {string[]} dictionary - The list of available words.
 * @returns {string[]} solutions - Possible solutions to the Boggle board.
 * Credit to: softnami.com "Trie Tree with JavaScript" (2020)
 * Credit to: GeeksforGeeks.com "Boggle (Find all possible words in a board
 * of characters) | Set 1" (2020)
 * Credit to Dr.Burge "Lecture 13 Sept"
 * Credit to Demeteria Mack
 */

exports.findAllSolutions = function(grid, dictionary) {
    let solutions = [];

    convertCase(grid,dictionary);
    let trie = CreateTrie(dictionary);

    if(grid == null || dictionary == null){
        return solutions;
    }

    let grid_len = grid.length;


    if(grid_len==0){
        return solutions;
    }

    for(let i=0; i<grid_len;i++){
        if(grid[i].length!= grid_len ){
            return solutions;
        }
    }

    let result= new Set();
    for(let y= 0; y<grid_len ; y++){
        for(let x= 0; x<grid_len ; x++){
```

```
            let word= "";
            let checked = new Array(grid_len).fill(false).map(() => new
Array(grid_len).fill(false));
            find(word,x,y,grid,checked,trie,result);
        }
    }

    solutions= Array.from(result)     return solutions;
} Missing Semi-colon

function TrieNode(value) {
    this.value=value;
    this.children = new Array();
    this.isValidWord = false;
}; // Unnecessary semicolon


function CreateTrie(dict) {
    var root = new TrieNode(''); /// Should be indented instead of spaced &
String should be double quotes

    if(dict.length==0){
        return;
    }

    for(let words of dict){
        var node = root;
        for(let  i =0;i<words.length;i++){
            var letter = words[i];
            var ord  = letter.charCodeAt(0) - 97;
            //if a node with that  letter doesnt exist:
            var currentNode = node.children[ord];
            if(node.children[ord]== undefined){
                //create one

                var currentNode = new TrieNode(letter); // This variable was
already defined
                node.children[ord]=currentNode;

            }
            node=currentNode;
        }
        node.isValidWord=true;
    }
    return root;
}; // Remove Semi colon

function find(word,x,y,grid,checked,trie,result){
    let directions=[[0,1],[1,0],[0,-1],[-1,0],[1,1],[-1,1],[1,-1],[-1,-1]];
```

```
    if(y<0 || x<0 || y>=grid.length || x >=grid.length ||
checked[x][y]==true){
        return;
    }

    word +=  grid[x][y];
    if(checkPrefix(word,trie)){
        checked[x][y]=true;
        if (isValidWord(word,trie)){

            if(word.length>2){
                result.add(word);
            }
        }

        for(let i=0;i<8;i++){

find(word,x+directions[i][0],y+directions[i][1],grid,checked,trie,result) //
Missing semi colon
        }
    }
    checked[x][y]=false // Missing Semi Colon
}

function checkPrefix(word,trie){
    let tword='' // String should be doublequote and missing semi colon
    let currentNode=trie;

    for(let i =0;i<word.length;i++){
        if(currentNode!=undefined){
            for(let node of currentNode.children){
                if(node!=undefined && node.value==word[i]){
                    tword+=word[i];
                    currentNode=node;
                    break;
                }
            }
        }
    }
    if(word==tword){
        return true;
    }
    return false;

}

function isValidWord(word,trie){
    let tword='' // Should be double quote & Missing Semi colon
```

```
    let currentNode=trie;
    for(let i =0;i<word.length;i++){
        if(currentNode!=undefined){
            for(let node of currentNode.children){
                if(node!=undefined && node.value==word[i]){
                    tword+=word[i];
                    currentNode=node;
                    break;
                }
            }
        }
    }
    if(word==tword && currentNode.isValidWord==true){
        return true;
    }
    return false;
}

function convertCase(grid,dict){
    for(let i=0;i<grid.length;i++){
        for(let j=0;j<grid.length;j++){
            if(grid[i][j]){
                grid[i][j]= grid[i][j].toLowerCase();
            }
        }
    }

    for(let j=0;j<dict.length;j++){
        dict[j]=dict[j].toLowerCase();
    }
}

var grid = [  ['t', 'w', 'y', 'r'],
    ['e', 'n', 'p', 'h'],
    ['g', 'z', 'qu','r'],
    ['st', 'n', 't', 'a']];
var dictionary = ['art', 'ego', 'gent', 'get', 'net', 'new', 'newt', 'prat',
    'pry', 'qua', 'quart', 'quartz', 'rat', 'tar', 'tarp',
    'ten', 'went', 'wet', 'arty', 'egg', 'not', 'quar'];
console.log(exports.findAllSolutions(grid, dictionary));
```