**Angular NGXS State Management — Extension Assignment**

**1. Overview & Objectives**

In this task, you will set up NGXS in the Angular front end for the Patient Visit Manager. The goal is that all shared data is managed in the store, not directly in services.  
You will:

* Define and organize app state (Auth, Patients, Doctors, Visits, FeeSchedule, ActivityLog).
* Create **actions, state handlers**, and **selectors**.
* Connect the store with the .NET Core Web API so the data always stays in sync.

**2. Requirements**

**State Slices (one State class for each module):**

* **AuthState** → user info, role, token, isAuthenticated.
* **PatientState, DoctorState, VisitState** → list of entities, ids, loading/error flags.
* **FeeScheduleState, ActivityLogState** → data, lastFetched time, loading/error.

**Actions (trigger events in the store):**

* Auth → Login, Logout, RefreshToken.
* CRUD → Load[X], Create[X], Update[X], Delete[X].

**Selectors (get data from the store):**

* Provide selectors for lists, single items, loading flags, error messages, and totals.

**Data Flow**

* Components will dispatch actions.
* Store will **call Web API** and update state.
* Components will **read data only via selectors** (not from services).

**Shared State**

* Any data that multiple components need should go into the store.

**Immutability & Single Source of Truth**

* Components must not directly change shared services.
* Services can remain as **API callers only** (no state inside).

**Bonus Tasks ⭐**

* **Caching** → Don’t re-fetch data if it was already loaded recently (e.g., within the last 2 minutes), unless forced.
* **Persist State** → Use NGXS Storage Plugin so important data (like Auth) stays even after page reload.

**Submission Guidelines**

* Upload to Azure DevOps in your respective Repository
* Branch Name: **patient-visit-manager-store**
* Submission Date: **Monday, 1st September, 2025 - 11:00 AM**
* Late submission policy applies

using System;  
using System.Collections.Generic;

class Solution  
{  
    static int MinSteps = int.MaxValue;

    static bool HasAdjacentMatch(List<int> digits)  
    {  
        for (int i = 1; i < digits.Count; i++)  
        {  
            if (digits[i] == digits[i - 1])  
                return true;  
        }  
        return false;  
    }

    static List<int> SplitDigits(int number)  
    {  
        var digits = new List<int>();  
        foreach (char c in number.ToString())  
        {  
            digits.Add(c - '0');  
        }  
        return digits;  
    }

    static void Recurse(int currentNumber, List<int> currentDigits, int steps, HashSet<string> visited)  
    {  
        // Split digits of current number and add to digit list  
        var newDigits = SplitDigits(currentNumber);  
        var combinedDigits = new List<int>(currentDigits);  
        combinedDigits.AddRange(newDigits);

        // Check for adjacent match  
        if (HasAdjacentMatch(combinedDigits))  
        {  
            MinSteps = Math.Min(MinSteps, steps);  
            return;  
        }

        // Memoization key (number + last few digits)  
        string key = currentNumber + "|" + string.Join(",", combinedDigits.GetRange(Math.Max(0, combinedDigits.Count - 10), Math.Min(10, combinedDigits.Count)));  
        if (visited.Contains(key)) return;  
        visited.Add(key);

        // Try multiplying by each non-zero digit  
        foreach (int digit in newDigits)  
        {  
            if (digit == 0) continue;  
            int newNumber = currentNumber \* digit;  
            Recurse(newNumber, combinedDigits, steps + 1, visited);  
        }  
    }

    public static int MinMultiplicationsToAdjacentMatch(int startNumber)  
    {  
        MinSteps = int.MaxValue;  
        Recurse(startNumber, new List<int>(), 0, new HashSet<string>());  
        return MinSteps == int.MaxValue ? -1 : MinSteps;  
    }

    // For testing  
    static void Main()  
    {  
        int start = 8;  
        int result = MinMultiplicationsToAdjacentMatch(start);  
        Console.WriteLine($"Minimum multiplications for {start} to reach adjacent match: {result}");  
    }  
}

// See https://aka.ms/new-console-template for more information

using System.ComponentModel.Design;

public class Program

{ // directions: up-left, up, up-right,left,right, down-left, down,down-right

public static int[] D\_row = { -1, -1, -1, 0, 0, 1, 1, 1 };

public static int[] D\_col = { -1, 0, 1, -1, 1, -1, 0, 1 };

static bool DFS(char[][] board, string word, int k, int r, int c, bool[,] seen)

{

int rows = board.Length;

int cols = board[0].Length;

if (r < 0 || c < 0 || r >= rows || c >= cols)

{

return false;

}

if (seen[r, c])

{

return false;

}

if (k == word.Length - 1)

{

return true;

}

if (word[k] != board[r][c])

{

return false;

}

seen[r, c] = true;

for(int d=0; d<7; d++)

{

int nr = r + D\_row[d];

int nc = c + D\_col[d];

if (DFS(board, word, k + 1, nr, nc, seen))

{

seen[r, c] = false ;

return true;

}

}

return false;

}

static bool Exists(char[][]board,string word)

{

int r = board.Length;

int c = board[0].Length;

bool[,] seen=new bool[r,c];

for(int i = 0; i < r; i++)

{

for (int j = 0; j < c; j++)

{

if (board[i][j] == word[0] && DFS(board, word, 0, i, j, seen))

{

return true;

}

}

}

return false;

}

static void Main()

{

string[] strArr = { "asdf,fdgg,ergs,dsfg", "safas,fdfg,gfdg,dassdf,th,dsgg" };

char[][] Board=strArr[0].Split(',').Select(row=>row.ToCharArray()).ToArray();

string[] words=strArr[1].Split(',');

List<string> found= new List<string>();

foreach(string word in words)

{

if (Exists(Board, word))

{

found.Add(word);

}

}

Console.WriteLine(found.Count==words.Length);

}

}