**OrderPally 前端前測**

**以下題⽬的回答請上傳到 github**

**題⽬ 1**

**請使⽤ Vue2 或 Vue3 設計⼀個 List Component, 需求如下：**component 有下列 properties

props: {

data: {

type: Array,

required: true

}

}

component 可接受以下三種格式的資料並呈現於畫⾯上：

Type 1

[

{

name: "Anna",

year: 10,

class: "A"

},

{

name: "Mark",

year: 10,

class: "M",

}

]

Type 2

[

{

title: "Javascript Info", category: "book",

rating: 10,

},

{

title: "Fantasy",

category: "movie",

rating: 5,

}

]

```

Type 3

[

{

winner: 'Amy',

rank: 1

},

{

winner: 'Bob',

rank: 2

},

{

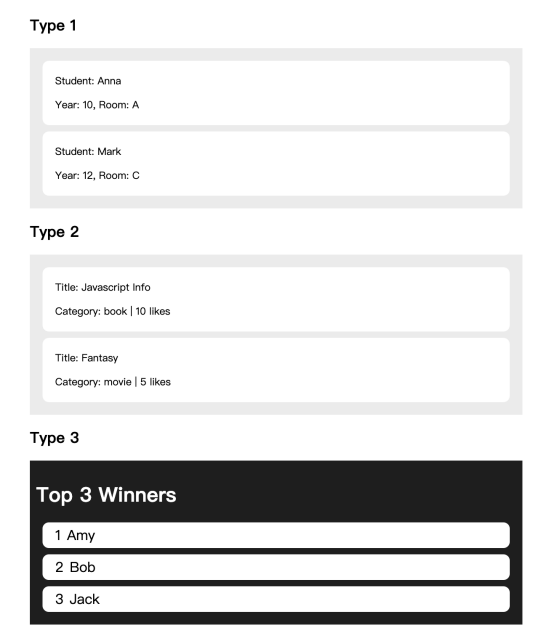
winner: 'Jack',

rank: 2

}

]

**畫⾯參考:**

**題⽬ 2**

有⼀份食譜列表結構如下：列表的每個元素皆為 Array, 第⼀個元素是菜名, 其他元素為這份食譜要⽤到的原料。

請列出各原料能做出哪些菜，原料需按照字⺟順序排序, 舉例： **Example 1**

// input

recipes = [

["Salad", "Tomato", "Cucumber", "Salad", "Sauce"], ["Pizza", "Tomato", "Sausage", "Sauce", "Dough"], ["Quesadilla", "Chicken", "Cheese", "Sauce"], ["Sandwich", "Salad", "Bread", "Tomato", "Cheese"] ]

solution(recipes)

// output

[

["Cheese", "Quesadilla", "Sandwich"],

["Salad", "Salad", "Sandwich"],

["Sauce", "Pizza", "Quesadilla", "Salad"], ["Tomato", "Pizza", "Salad", "Sandwich"] ]

**Example 2**

// input

recipes = [

["Pasta", "Tomato Sauce", "Onions", "Garlic"], ["Chicken Curry", "Chicken", "Curry Sauce"], ["Fried Rice", "Rice", "Onions", "Nuts"], ["Salad", "Spinach", "Nuts"],

["Sandwich", "Cheese", "Bread"],

["Quesadilla", "Chicken", "Cheese"]

]

solution(recipes)

// output

[

["Cheese", "Quesadilla", "Sandwich"],

["Chicken", "Chicken Curry", "Quesadilla"], ["Nuts", "Fried Rice", "Salad"],

["Onions", "Fried Rice", "Pasta"]

]

**請完成以下 function**

function solution(recipes) {

// TODO

}

2-1 recipes 保證在以下範圍內

1 ≤ recipes.length ≤ 500, 2 ≤ recipes[i].length ≤ 10, 1 ≤ recipes[i][j].length ≤ 50. 2-2 原料與菜名字⾸必為⼤寫(如上述範例)

**題⽬ 3**

**3-1 請說明 Flux, Redux, Vuex 之間的差異**

**Flux**是單向資料流的設計概念, 方向為 Action => Dispatcher => Store => View, 以下說明各角色負責的功能：  
  
1. Action: 規範所有改變資料的動作  
  
2. Dispatcher: 將目前發生的行為, 告知給所有已註冊的Store, Dispatcher會提供API讓Store註冊callback function, 這個callback function會讓Store得知是否有新Action  
  
3. Store:   
 a. 存放資料和業務邏輯, 根據業務需求會分成多個Store管理, 有可能發生只更新部分store而造成錯誤  
 b. 更新資料只能經由Action和Dispatcher, 並且提供API讓View註冊listener, 當資料更新時會讓View得知  
 c. 提供getter API 讓人取得資料  
  
4. View: 根據監聽Store資料渲染UI和監聽使用者操作事件來觸發Action

**Redux**是根據Flux實作的狀態管理庫, 以下說明各角色負責的功能：  
1. Action: 同Flux, 規範所有改變資料的動作, 並且由View發出  
  
2. Store: 存放狀態, 整個應用只會有一個Store相較Flux管理較簡單, 並且提供以下管理state的API:  
 a. getState: 取得當前state  
  
 b. dispatch(action): Redux沒有Dispatcher, 而是在Store提供了用來觸發state改變的dispatch函式, 是View發出Action的唯一途徑  
  
 c. subscribe(listener): 設置state變化的監聽函式, 若把View的更新函式作為listener傳入, 則可觸發View自動渲染  
  
3. Reducer: 存在於Store的同步純函數, 根據action.type更新state, 並返回新的state替換原本的state, (preState, action) => newState  
4. View: 同Flux

**Vuex**也是根據Flux實作的Vue狀態管理庫, 跟Redux一樣整個應用只會有一個Store, 以下說明各角色負責的功能：1. Action: 屬於Store的一部份  
 a. 可以在這裡對state異步操作, 並透過 store.commit 調用Mutaions處理狀態變化  
 b. View可以透過 store.dispatch 觸發Action  
  
2. Mutations: 屬於Store的一部份, vuex中唯一改變state的途徑, 而且只能是同步操作  
  
3. State: 屬於Store的一部份, 為Vuex的單一數據源, 負責更新View  
  
4. View: 同Flux

**3-2 請說明單向資料流與雙向資料流在處理資料更新上的異同**

單向資料流: 更新資料一律從源頭開始, 無法從目標將資料直接返回源頭

雙向資料流: 資料可以從源頭流向目標, 可以從目標流向源頭, 但操作會更為複雜

**3-3 這些設計主要想要解決什麼樣的問題? 優缺點為何？**

為了解決全域狀態管理

優點: 使程式碼具有可測試性、可維護性和可擴展性

缺點: 增加程式複雜度, 通常適用於大型應用程式, 對小型程式來說過於繁重