

HW： Simple Book List Viewer

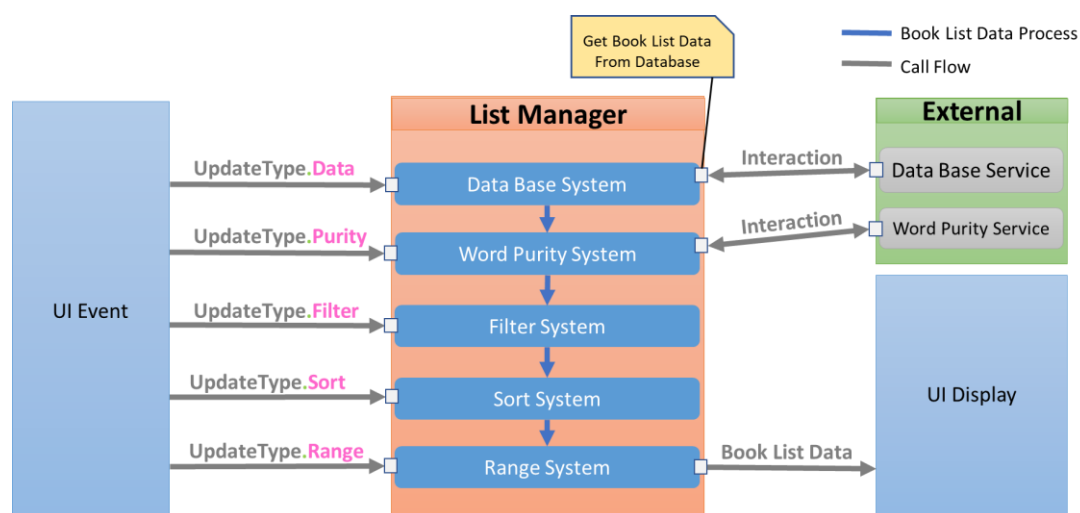
系統描述

Simple List viewer 是一個展示 Book List 的功能，並且提供些許功能讓使用者去改變顯示的 Book，例如 過濾出特定關鍵字(Filter)、排序(Sort)、範圍顯示(Range)等等功能。而這個 Book 的資訊有 ISBN、書名(Title)、作者(Author)。

這個系統會從 Data Base Service 中取出完整的 Book List。然後這些 Book List 在根據使用者的需求在前端改變要顯示的 Book 以及資訊。不過這樣就會有個問題是說當 Book 數量非常多的時，如果每次更改一次功能 Filter 就要重新把所有的功能在執行過一遍。那這樣勢必在大量資料時造成執行時上的負擔。

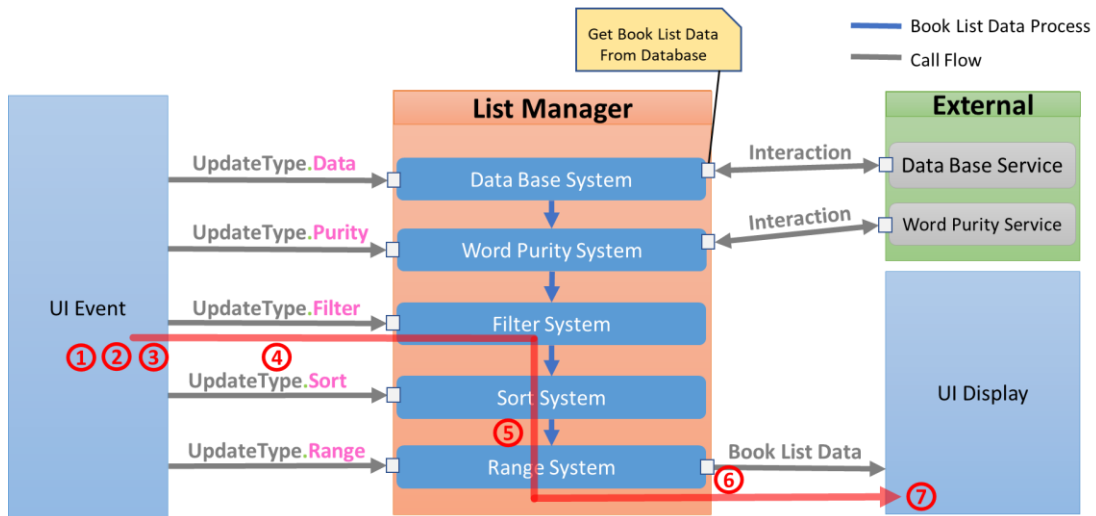
所以，為了避免這種情況，ListManager 就像一個小型的功能管理系統，可以根據更動的部分做更新。這樣可以減少不必要運算。例如要更新範圍顯示(Range)的功能可以不用在執行一次前面所有的操作，只要將前個排序功能(Sort)的結果取出然後在取出對應範圍的 Book List。

系統架構圖



例子：

假設今天 Peter 想要只看標題包含 “Ales” 關鍵字の書本，那這個系統の流程會像是這樣：



程式碼執行順序：

1. UIEvent 觸發
2. Event 取出 ListManager 中的 FilterSystem
3. filterSystem.setFilterWord(“Ales”)
4. listManager.updateResult(UpdateType.Filter)
5. 在 ListManager 中的 updateResult Method 把 FilterSystem、SortSystem、RangeSystem 的 process Method 都執行一遍
6. 呼叫 listManager.generateDisplayItemRow() 產生出用於顯示の資料
7. UI 更新資料

外部套件

這裡の外部套件有 DataBase Service 與 Word Purity Service。首先是 **DataBase Service** 實際上是模擬資料庫の互動，例如：連接、新增、刪除、取值，但是資料都是 In-Memory。

而 **Word Purity Service** 則是給可以把一個字串中の敏感字串轉換成 “*”，例如設定 “abc” 為敏感字串，而輸入 “xyzABC123abc” 則會輸出 “xyz***123***” (case-insensitive)。

照理來說對於這些外部套件，只需要知道該怎麼使用，並且 DataBaseSystem 與 WordPuriySystem 會怎麼與這兩個外部套件做互動。

環境架設

下載 NodeJS (<https://nodejs.org/en/>)

註：NodeJS 版本只要 $\geq v18.12$ 應該是都沒問題

如果你已經有 NodeJS 環境可以直接試著執行作業看看，如果不行執行在安裝此版本即可

下載後就直接一直 next 即可。

開啟作業

作業下載

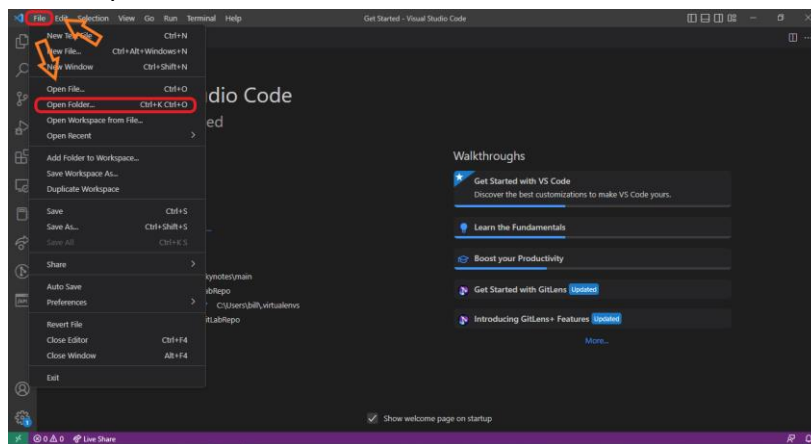
下載位置：[GOOGLE DRIVE](#)

VSCode

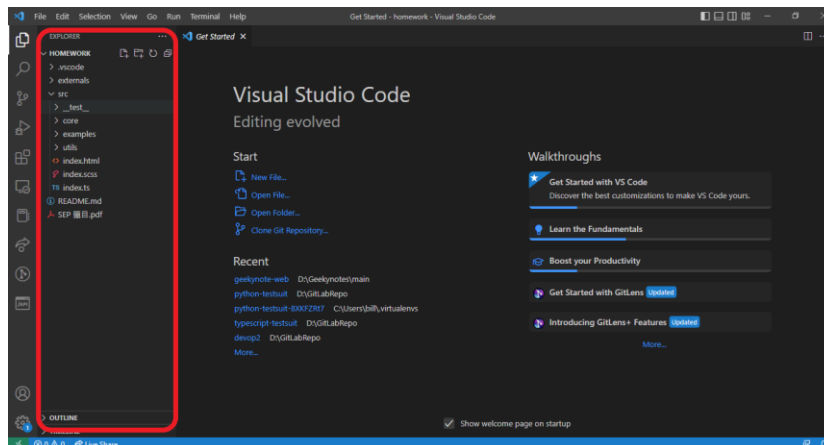
本範例使用 VS Code 作為 IDE。

VSCode 下載：[Download Visual Studio Code - Mac, Linux, Windows](#)

點擊 Open Folder 然後選擇作業的資料夾



有看到這些檔案就代表開對了

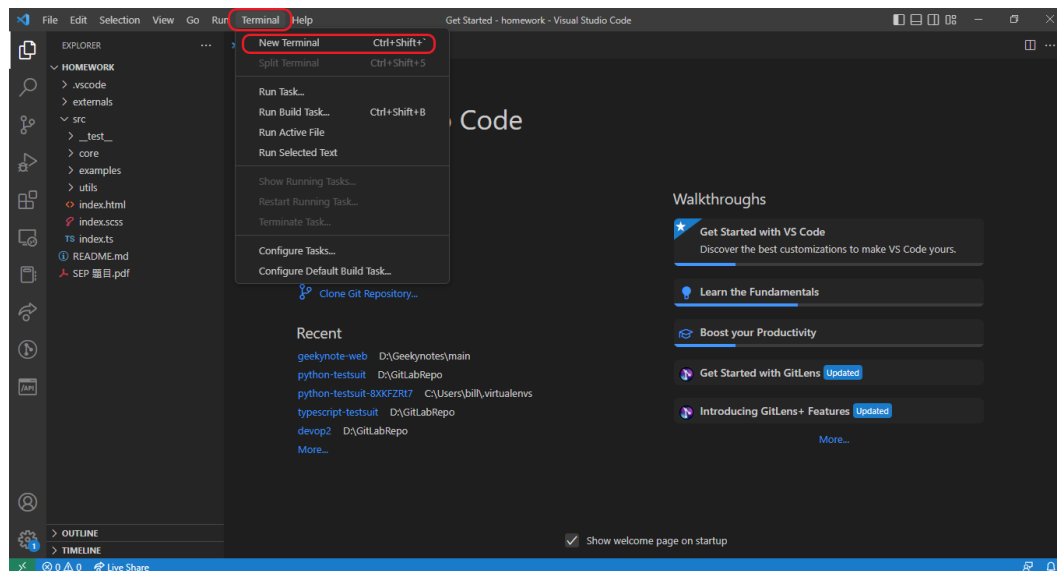


執行作業

Open Terminal

點 Terminal -> new Terminal

註：如果你用的是 PowerShell、Bash 等等就直接 cd 進到作業中



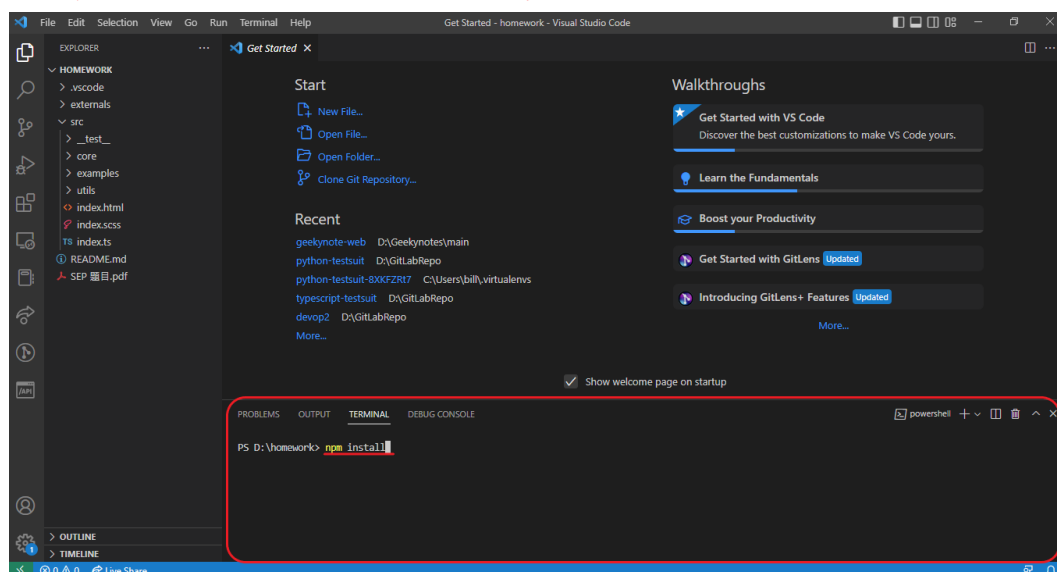
Dependency Install

在 Terminal 中打上

npm run install

安裝作業中所需的 Dependency

註：無法安裝的或有錯誤訊息的找助教幫忙釐清問題



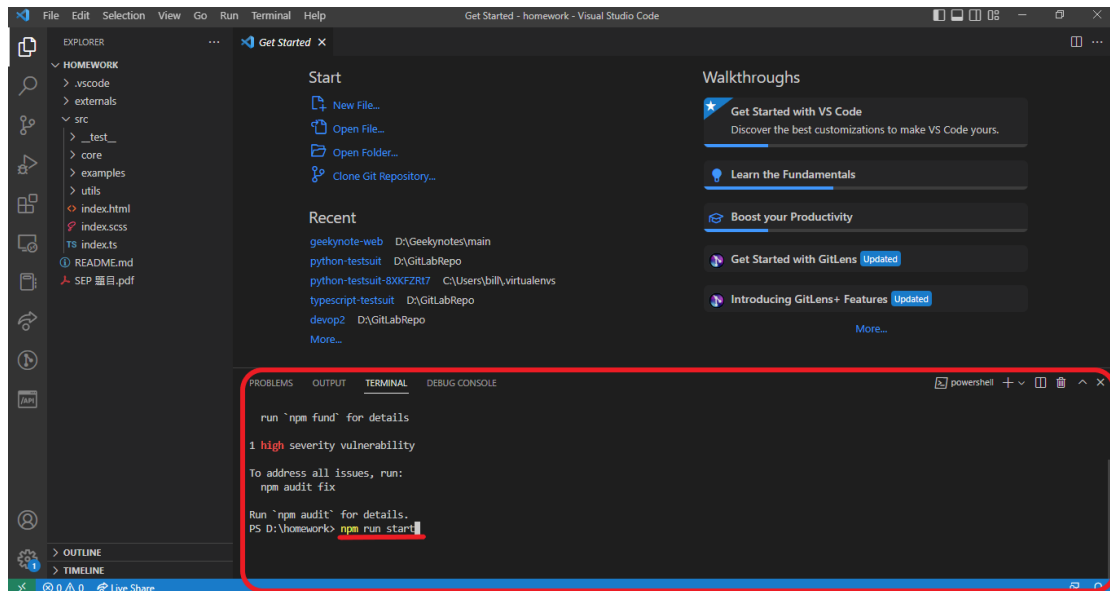
Run Application

安裝完後

在 Terminal 中打上

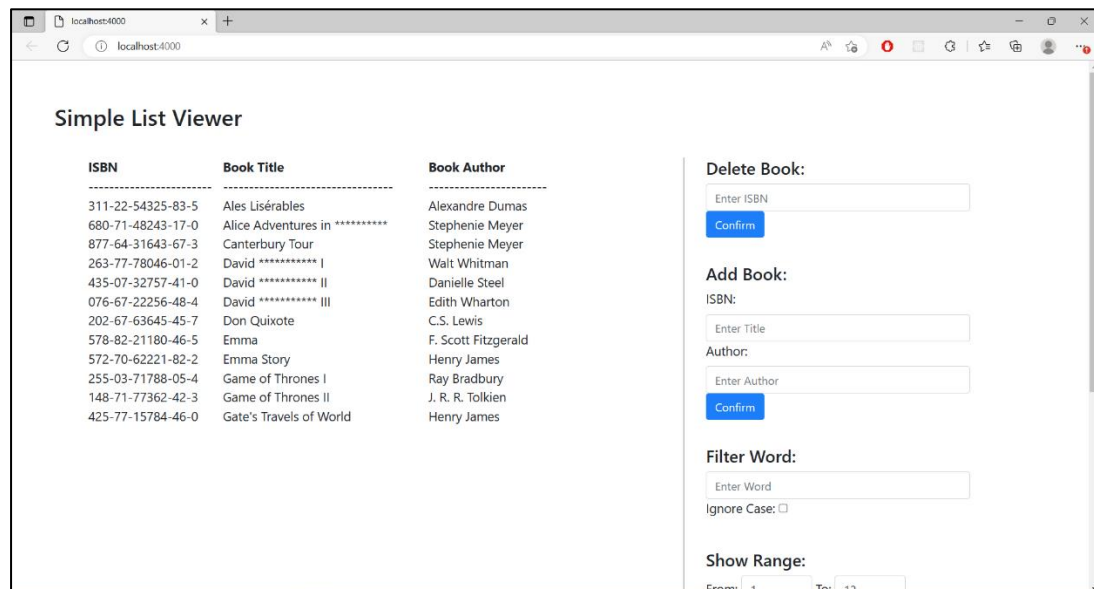
```
npm run start
```

就可以看到它會自動開啟一個網頁



網頁：

註：網頁沒有自動開啟的可以試試在瀏覽器中輸入 <http://localhost:4000/>

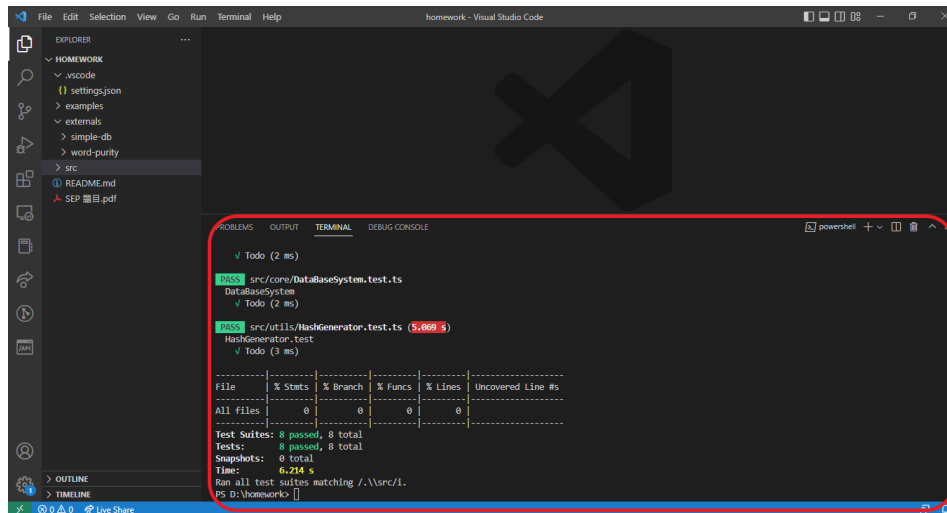


Test Application

在 terminal 或 console 中執行

```
npm run test
```

執行後的結果



The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal displays the output of running 'npm run test'. It shows two test files passing: 'src/core/DatabaseSystem.test.ts' and 'src/utils/HashGenerator.test.ts'. A summary table follows, showing 8 passed tests, 8 total tests, and 0 snapshots. The total time taken is 6.214 s. The terminal output is as follows:

```
✓ Todo (2 ms)
PASS src/core/DatabaseSystem.test.ts
DatabaseSystem
✓ Todo (2 ms)
PASS src/utils/HashGenerator.test.ts (5.069 s)
HashGenerator.test
✓ Todo (3 ms)

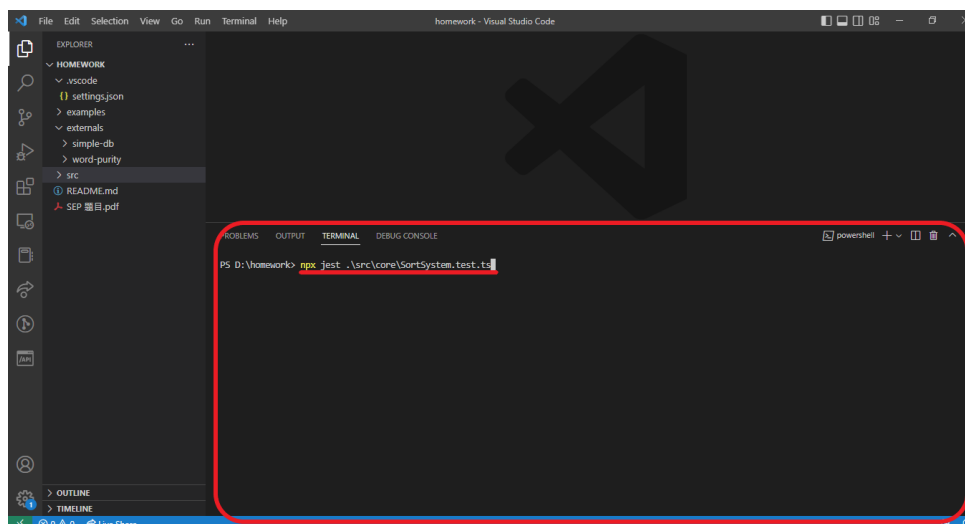
File      | % Stmts | % Branch | % Funcs | % Lines | Uncovered Line #s
-----|-----|-----|-----|-----|-----
All files |    0    |    0     |    0    |    0    |
Test Suites: 8 passed, 8 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        6.214 s
Ran all test suites matching /.\\src\\1.
PS D:\homework>
```

Test Single File

在 terminal 或 console 中執行

```
npm run test
```

這樣就可以只測試單個檔案



The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal displays the command 'npm run test' being entered. The command is highlighted in red. The terminal output is as follows:

```
PS D:\homework> npm run test
```

作業專案結構講解：

Folder Structure

註：一切都已 **VSCode** 上看的到的為主

examples	範例的 Testcase，可以在這裡找到許多幫助你完成作業的功能
externals	外部的套件，照理說你不需要讀懂內部的程式碼，之需要知道它提供了那些功能以及你的系統怎麼跟他做互動
src	這次作業主要存放 Source Code 的資料夾
src/__test__	用於存放測試時的假資料
src/core	主要的核心程式，理應幫這裡面所有的 Class、Function 撰寫測試
src/utils	其餘功能，也需要幫這裡面的 Class、Function 撰寫測試

預設測試資料

檔案位置：**src/__test__/TestingData.ts**

這裡面有已經預設好的 **Mock Data**，請妥善的利用它幫助你完成所有的測試

使用方法直接添加此行到你要測試的檔案中的第一行

然後就可以直接把 **TestBookInfo** 當成變數來使用即可

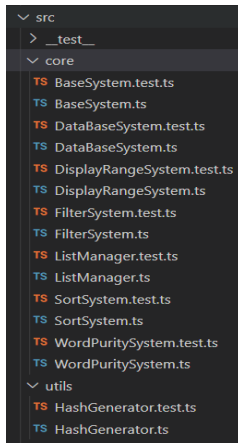
```
import { TestBookInfo } from "../__test__/TestingData";

describe("Testcase use", () => {
  test("in testcase", () => {
    const data = TestBookInfo;

    console.log(TestBookInfo);

    expect(TestBookInfo).toBeTruthy();
  });
});
```

你的任務



你的目標是為整個系統寫 Unit Test，照理說你應該只可以改動檔名後綴為 ***.test.ts** 的檔案，並為他們加上該有的 Testcase。

在 **core**、**utils** 資料夾內部的檔案。

每一個 System 都有自己對應的 Test File，檔名後綴均為 ***.test.ts**。請為這些 Test 檔案完成他們該有的單元測試。

下圖為一個最終測試完後的整體 Coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	99.5	95.65	100	100	
core	99.45	95.34	100	100	
BaseSystem.ts	100	100	100	100	
DataBaseSystem.ts	97.72	85.71	100	100	12,22
DisplayRangeSystem.ts	100	100	100	100	
FilterSystem.ts	100	100	100	100	
ListManager.ts	100	100	100	100	
SortSystem.ts	100	100	100	100	
WordPuritySystem.ts	100	100	100	100	
utils	100	100	100	100	
HashGenerator.ts	100	100	100	100	
Test Suites: 8 passed, 8 total					
Tests: 51 passed, 51 total					
Snapshots: 0 total					
Time: 8.257 s, estimated 10 s					

測試的數量不是重點，重點是可以 **Cover** 到所有的關鍵程式碼，並且各個 **Boundary**、**Negative** 都可以測試到。

另外撰寫 Testcase 的可閱讀性以及對於 Testcase 的描述也非常重要。

1. 請你設計一套 unit test cases 他能夠完整地上述的系統
2. 所謂的完整地 unit test cases 按照老師的投影片解說，每個 test cases 的設計你要能夠說出個所以然
 1. Code coverage
 2. Partition testing
 3. Boundary tests
 4. Negative tests
 5. Mock
 6. Stubs
 7. Exception Handling
 8. *Performance tests (加分，我知道，老師沒有教)
 9. *Partial Test oracle (加分，如果你找得出來的話，而且言之有理。請用註解標註 //partial oracle 讓助教可以去幫你加分。

題目 1

撰寫出 FilterSystem Class 與 SortSystem Class 的 Unit Test Cases

(確保你的測試可以 Cover 到所有的程式碼)

Class 描述：

FilterSystem 的功能是可以依據 BookInfo 中的 Title 篩選出含有關鍵字的功能

SortSystem 的功能是可以依序 BookInfo 中的 Title 去做排序

題目 2

撰寫出 DisplayRangeSystem 的 Unit Test Cases

(確保你的測試都可以做到 exception handling)

Class 規格與功能描述：

DisplayRangeSystem 用於控制 List 的顯示範圍。

StartRange 與 EndRange 必須是大於零、是個整數、且 EndRange 一定要大於等於 StartRange。

setRange Method 可以以接受數字以及字串輸入。

如果有不符合輸入的都會拋出一個錯誤

題目 3

撰寫出 HashGenerator 的 Unit Test Cases

(請使用 jest 或其他技術來控制 Math.random 的回傳值)

Class 規格與功能描述：

這個可以隨機產生出 A-Z 大寫字母的 Character

```
public g(charaterNum: number) { ..... }
```

這個可以藉由任意 “xxx-xxx-xxxx-xxx” 字串只將 x 轉換成 0-9 的任意數字

```
public simpleISBN(pattern: string) { ..... }
```

題目 4, 5

撰寫出 WordPuritySystem 的 Unit Test Cases

(請使用 Stub 來幫助你測試 WordPuritySystem，並且用 Mock 來確保有正確地與外部 Class 做互動)

Class 規格與功能描述：

WordPuritySystem 使用了 WordPurityService 外部套件，用於將 BookInfo 中敏感的字給替換成 *，不分大小寫。

例如設定 “abc” 為敏感字串，而輸入 “xyzABC123abc” 則會輸出 “xyz***123***” (case-insensitive)。

題目 6, 7

撰寫出 DataBaseSystem 的 Unit Test Cases

(請使用 Stub 來幫助你測試 DataBaseSystem，並且用 Mock 來確保有正確地與外部 Class 做互動)

Class 規格與功能描述：

DataBaseSystem 使用了 Simple Database 外部套件，用於與資料庫做簡單的互動。

DataBaseService 每個 Method 均會有可能會拋出錯誤，所以 DataBaseSystem 都要能正確的接收這些錯誤。

題目 Bonus

為 ListManager 撰寫 Unit Test Cases

(請使用 Mock 來確保有正確地與所有外部 Class 做互動)

Class 規格與功能描述：

ListManager 用於控制所有 System 的行為，並且規畫出他們執行的順序，能依照更新點來更新資料，以減少不必要的 Cost。

一個 ListViewManager 的 System 優先度是：

1. DataBaseSystem
2. WordPuritySystem
3. FilterSystem
4. SortSystem
5. DisplayRangeSystem

例如當 FilterSystem 更新時，只需要更新 SortSystem 再來更新 DisplayRangeSystem 即可。