

# 軟體工程期末報告

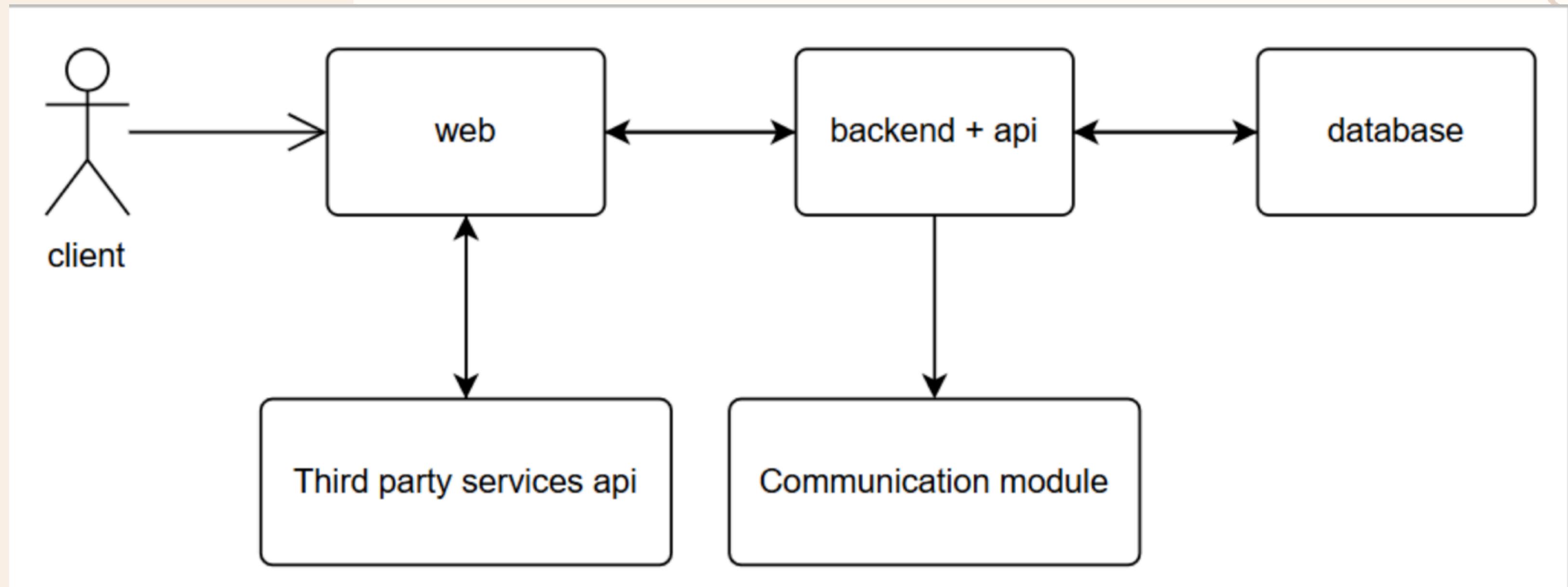
台科大點餐系統組

# OUTLINE

- 01 Project Introduction
- 02 Waterfall
- 03 Unit Testing(backend)
- 04 Agile
- 05 Unit Testing(frontend)
- 06 Refactoring

# Project Introduction

# 台科點餐系統 Structure Overview



# 台科點餐系統

## 主畫面



## 搜尋畫面

The search interface displays a list of restaurants in the Daan District, Taipei, with their details and a map view.

- 美德耐 台科大活動中心第一餐廳**  
4.4 ★★★★★ (1,183) · \$1-200  
餐廳 · 基隆路四段43號B1  
已打烊 · 開始營業時間：週一07:00  
內用 · 外帶
- 帝一味自助餐**  
4.2 ★★★★★ (103) · \$1-200  
餐廳 · 基隆路四段43號第三學生餐廳  
已打烊 · 開始營業時間：週一10:30  
內用 · 外帶
- 七樂燒臘**  
4.0 ★★★★★ (29) · \$1-200  
中國菜 · 基隆路四段43號  
已打烊 · 開始營業時間：週一11:00  
內用 · 外帶
- 炸物三兄弟 (台科後餐鹹酥雞)**  
4.3 ★★★★★ (98) · \$1-200  
炸物串與串炸餐廳 · 基隆路四段41巷68弄2號  
已打烊 · 開始營業時間：週一18:00  
內用 · 外帶 · 無外送服務

# 台科點餐系統

登入畫面

登入

  
  
   
[忘記密碼](#)  
—— 以其他登入方式 ——  
  
[返回](#)

註冊畫面

註冊

  
  
  
[返回](#)

# 台科點餐系統

## 已完成訂單畫面

< 已完成訂單

摩斯漢堡 NTD 220

下訂於 2024年9月10日 星期二 13:00

1X 超級大麥海洋珍珠堡  
1X 超級大麥薑燒珍珠堡  
1X 紅茶 ...

重新下訂

## 訂單詳情畫面

<

訂單 #1111111111

外送於 11月20日 15:23

訂購於 摩斯漢堡	小計 \$ 178
外送到 台北市大安區基隆路四段43號	外送服務費 \$ 35
	折扣 -\$ 60
	平台 \$ 5
	總計 \$ 278
超級大麥海洋珍珠堡 x1	\$ 69
超級大麥薑燒珍珠堡 x1	\$ 69
紅茶 x2	\$ 40

付款方式  
線上結帳  
\$ 278

重新下訂

# 台科點餐系統

餐廳簡介畫面

摩斯漢堡

★★★★★ 5.0(3000+則評分)

⌚ 目前營業中 星期一 - 星期日 8:00 - 19:00

📍 106台北市大安區基隆路四段43號

人氣精選 精選組合餐 快速組合餐 單點點心 單點飲品

**人氣精選**

超級大麥燒肉珍珠堡組合餐 \$170  
組合餐內含點心兩份及飲品選擇一份 | 使用.....

超級大麥海洋珍珠堡組合餐 \$170  
組合餐內含點心兩份及飲品選擇一份 | 使用.....

店內搜尋

愛心圖示

放大鏡圖示

喜愛餐廳畫面

位置 ▾ 購物車

喜爱餐廳

摩斯漢堡	評分 4.7
摩斯漢堡	台北市大安區 0912345678
摩斯漢堡	評分 4.7



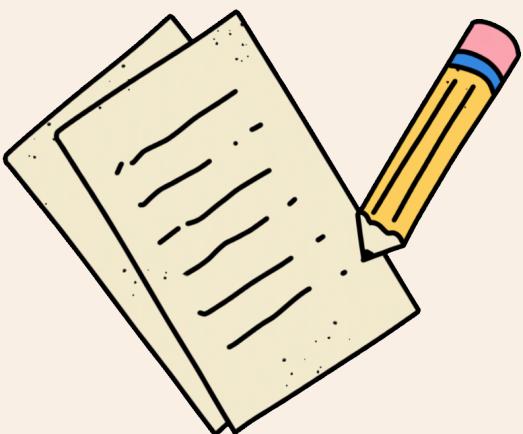
# Waterfall

# 文件介紹

PRD:需求分析文件

SDS: 解釋該系統的目的和功能

API文件: 描述提供給前端的接口



Name	Last commit message	Last commit date
...		
SDS_Document.pdf	add sds	last month
Waterfall PRD.docx	Add files via upload	last month
docs.go	fix bug	last month
swagger.json	fix bug	last month
swagger.yaml	fix bug	last month

# PRD

## Product Requirement Document

### 開發軟體簡介

● 主題：台科點餐系統  
● 動機：由於學餐在用餐尖峰時段總是大排長龍，不僅浪費學生和教職員的寶貴時間，更降低了用餐品質。因此，我們希望開發一個線上點餐系統，讓消費者可以在到達餐廳前提前下單，減少排隊時間，並讓餐廳能更有效率地管理訂單和客流，這不僅提升了顧客的用餐體驗，也幫助餐廳優化營運流程。  
● 系統概述：本點餐系統旨在連接顧客與餐廳，提供線上選餐、下單和評價等功能。顧客可透過系統瀏覽餐廳和菜單，並在高峰時段提前下單，避免排隊等待；同時可保存「我的最愛」餐廳，方便快速點餐。餐廳方則能管理訂單、查看顧客評價，並分析訂單數據以優化營運。系統支援多設備使用，提供直觀且流暢的用戶體驗，提升用餐和訂餐的效率。

### 組員

B11005121 許苑真	B11115055 游騰楓
B11115016 Jin Woojun	B11130005 鍾昊成
B11115021 徐家萱	B11130034 游傳智
B11115049 黃羿華	

### 文件更改歷程

日期	更改內容
2024/11/02	初始版本
2024/11/06	需求更新
2024/11/07	最終版本

### 目標使用者及需求

● 一般用戶：在校學生及教職員，包含本國生及外籍生，尋找餐廳、儲存最愛、並進行點餐的個人用戶，希望加速點餐流程，提升用餐品質。  
● 餐廳業主：學生餐廳的管理者，尋求平台來管理訂單、評價和顧客互動，希望能更有系統性的管理顧客訂單及自家餐廳資訊。

### 顧客需求、市場行銷與商業模型

● 已知顧客與需求：顧客包括餐廳和尋求簡化點餐與評價體驗的個人用戶。潛在使用者調查的反饋顯示，他們對個性化的餐廳和菜單推薦特別有興趣。  
● 量化的顧客數據：大部分用戶希望有「最愛餐廳」的功能，而部分用戶則希望擁有便捷的評價系統。  
● 商業模型與定價：計畫以 SaaS 的模式提供服務，為餐廳設立標準定價方案，並可能提供高級顧客分析功能作為附加選項。  
● 預期成果：我們預計在推出後的前三個月內，將有約 20 家餐廳和 500 名個人用戶註冊此功能。此外，透過精簡且易於使用的評價選項，我們預測在 6 個月內餐廳評價互動率將增加 10%。

### 關鍵指標

說明：關鍵指標的設計目的在於衡量系統在使用者及餐廳端的成功程度和使用率。這些指標幫助我們了解系統的市場反應和用戶參與度。透過分析這些數據，開發團隊能持續優化功能，提升用戶體驗，並確保解決顧客問題的效率。此外，關鍵指標也能讓餐廳了解在平台上的表現，藉此制定更合適的營運策略。這些指標不僅是產品成效的衡量工具，也是持續改進的依據。

系統的關鍵指標包含：

- 註冊的餐廳與用戶帳戶數量
- 每月活躍用戶（MAU）與每日活躍用戶（DAU）
- 每家餐廳與每位用戶的訂單量
- 每家餐廳的平均評分
- 餐廳和個人用戶的留存率
- 解決顧客問題所需的時間

### 用詞與文案

說明：用詞與文案部分設計了清晰而直接的詞句，方便用戶理解和使用系統功能，關鍵用語是為了讓不同技術背景的用戶快速找到所需功能。此外，針對餐廳業主，我們設計了「Restaurant Dashboard」與「Customer Analytics」，讓他們在管理店家資訊、訂單和顧客數據時更加便捷。這些精心設計的文案不僅提升了系統的易用性，還傳達了每個功能的核心價值，使用戶體驗更加直觀。

關鍵用語：

- 「Order Now」：迅速從選定餐廳下單的行動呼籲。
- 「Favorites」：用戶可以儲存偏好的餐廳，方便快速存取的區域。
- 「My Reviews」：用戶查看、管理和編輯餐廳評價的區域。
- 「Recommended for You」：根據用戶偏好提供的個性化餐廳和菜單建議。
- 「Restaurant Dashboard」：餐廳業主用來管理店家資訊、訂單與評價的集中控制面板。
- 「Customer Analytics」：供餐廳業主使用的高級功能，能查看顧客見解與數據趨勢。
- 「Explore Restaurants」：讓用戶探索新餐飲選擇的瀏覽區域。

### Functional Requirements

- 帳號管理：系統必須支援使用電子郵件和密碼的帳號註冊，此外，帳號管理功能應包含帳戶啟用、登入及密碼重置，確保用戶能輕鬆、安全地管理個人資料。
- 餐廳管理：餐廳業者應該能在系統中輕鬆新增、更新餐廳資訊及菜單，確保餐廳內容即時、準確地展示給用戶。管理功能需簡單易用，支持餐廳業者靈活調整店內資訊，以因應變動的營運需求。
- 訂單與我的最愛：用戶應能在系統中輕鬆下單、收藏喜愛的餐廳，並有效管理個人訂單。此功能讓用戶更快速地查看訂單狀態、追蹤已下訂的餐廳，同時滿足他們的收藏需求，提供便捷且個性化的訂餐體驗。
- 評價系統：用戶必須具備新增、修改和查看餐廳回饋的功能，這使用戶能分享用餐體驗並給予評價。系統需支援多樣化的回饋功能，讓用戶不僅能評分，還能撰寫評論，促進餐廳與顧客之間的雙向交流。
- 易用性與導覽：介面設計應易於訪問且方便導航，提供清晰的操作指示，讓所有技術水平的用戶都能輕鬆使用系統，確保用戶在點餐、收藏及查看評價等操作上能快速上手，提升整體使用體驗。
- 多語言支援：考慮到來自不同地區的用戶需求，系統應支援多種語言，包括英文，並應具備增設其他語言的彈性。此功能能讓用戶在偏好語言下流暢操作，增加產品的全球適應性及用戶滿意度。

# SDS

Design Specification of online ordering platform		
Doc #	Version: 2024	Page 1 / 22

TABLE OF CONTENTS	
1	Introduction 1
1.1	Document conventions 1
2	Software architecture overview 2
3	Software design specification 3
3.1	Frontend 3
3.1.1	Component design description 3
3.1.2	Workflows and algorithms 4
3.1.4	Component secure design description 10
3.1.6	Software requirements mapping 11
3.2	Backend 11
3.3	Database 19
4	Responsibilities table 21

**1 Introduction**  
線上點單平台的出現，徹底改變了餐飲業中商家與顧客的互動方式。平臺透過簡化點單流程，為顧客提供便利且易用的體驗，同時幫助商家更有效率地管理營運。

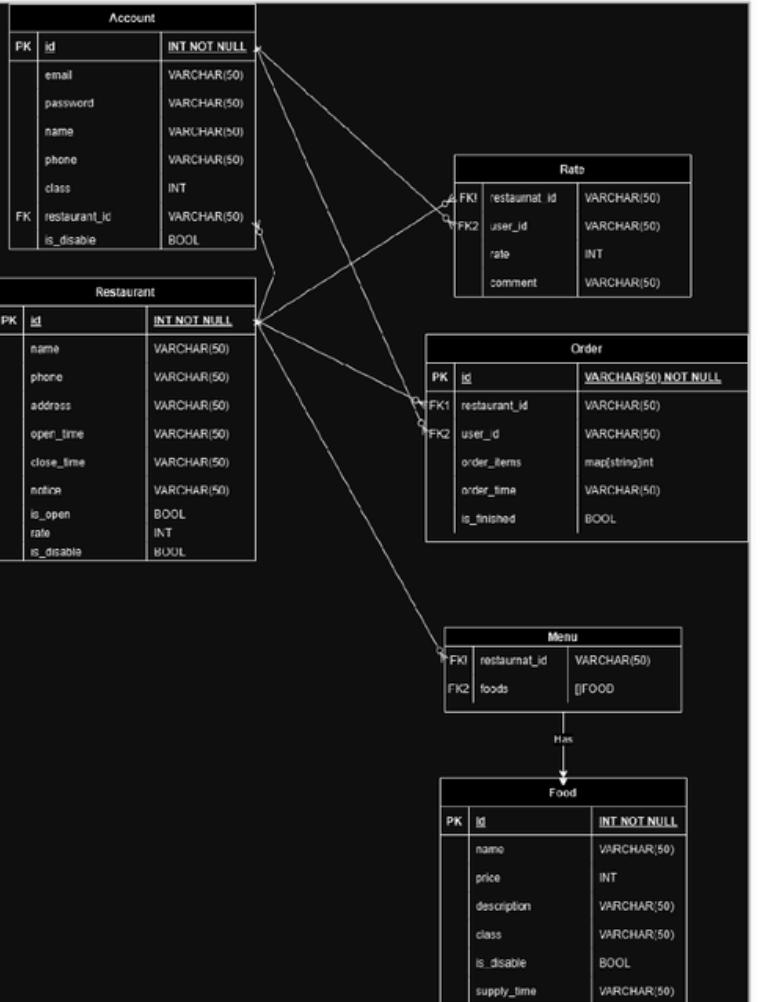
我們的線上點單平臺旨在為顧客與商家之間搭起一座無縫的橋樑，提供流暢且高效的點餐流程。此系統專為現代消費者而設計，滿足他們對便利性、速度和靈活性的需求。平臺功能包括即時菜單更新、安全的支付處理和訂單追蹤，旨在提升使用者滿意度，並改善商家營運效率。

本系統適用於所有類型的餐廳商家，並提供可自訂的功能以符合不同商家的需求。平臺支援桌面和行動裝置，確保跨設備的便利性，為終端用戶提供穩定的使用體驗。

**1.1 Document conventions**  
本文件統一採用Cambria字體撰寫，標題字型大小為12px，子標題及內文皆為11px，大小標題皆保持粗體。

Design Specification of online ordering platform		
Doc #	Version: 2024	Page 19 / 22

### 3.3 Database



Design Specification of online ordering platform		
Doc #	Version: 2024	Page 20 / 22

### Account

Account 表用於存儲所有帳戶的相關資訊。此表與 Restaurant, Rate 和 Order 表具有關聯性。

- 屬性:
  - id (PK)**: 帳戶的唯一識別碼。
  - email**: 用戶的電子郵件地址。
  - password**: 帳戶的密碼。
  - name**: 帳戶名稱。
  - phone**: 用戶的聯絡電話號碼。
  - class**: 帳戶的分類(如用戶類型)。
  - restaurant\_id (FK)**: 該帳戶所屬餐廳的識別碼。
  - is\_disable**: 帳戶是否被禁用的標記。

### Restaurant

Restaurant 表用於存儲餐廳的相關資訊。此表與 Account, Rate 和 Menu 表具有關聯性。

- 屬性:
  - id (PK)**: 餐廳的唯一識別碼。
  - name**: 餐廳名稱。
  - phone**: 餐廳的聯絡電話號碼。
  - address**: 餐廳地址。
  - open\_time**: 營業開始時間。
  - close\_time**: 營業結束時間。
  - notice**: 餐廳的公告訊息。
  - is\_open**: 餐廳是否營業的標記。
  - rate**: 餐廳的平均評分。
  - is\_disable**: 餐廳是否被禁用的標記。

### Rate

Rate 表用於存儲對餐廳的評分與評論。此表與 Restaurant 和 Account 表具有關聯性。

- 屬性:
  - restaurant\_id (FK)**: 評分餐廳的識別碼。
  - user\_id (FK)**: 評分用戶的識別碼。
  - rate**: 評分分數(整數)。
  - comment**: 用戶的評論內容。

### Order

# Swagger

可以透過簡單的編寫註解，就可以透過golang中的Swagger函式庫，自動生成API文件，還可以支援一些基本的測試。

```
// Login
// @Summary 登入帳號
// @Tags user module
// @param user body object{email:string,password:string} false "用戶登入資料"
// @Success 200 {string} json{"code", "message", "token"}
// @Router /user/login [post]
func Login(c *gin.Context) {
```

# Swagger

**user module**

**POST** /user/forget 忘記密碼

**GET** /user/info 獲取用戶信息

**POST** /user/login 登入帳號

**POST** /user/register 註冊新用戶

**POST** /user/verify 驗證帳號

**POST** /user/login 登入帳號 Try it out

**Parameters**

Name	Description
user object (body)	用戶登入資料 Example Value   Model

```
{ "email": "string", "password": "string" }
```

Parameter content type

**Responses** Response content type

Code	Description
200	code", "message", "token" Example Value   Model

```
"string"
```

# Go 單元測試

利用golang自帶的testing套件來實現單元測試

```
func TestRegister(t *testing.T) {
    router := setupRouter()

    userInput := models.UserBasic{
        Email:    "test@example.com",
        Password: "password123",
        Name:     "Test User",
        Phone:    "123456789",
        Roles:    "user",
    }

    jsonValue, _ := json.Marshal(userInput)
    req, _ := http.NewRequest("POST", "/user/register", bytes.NewBuffer(jsonValue))
    req.Header.Set("Content-Type", "application/json")

    w := httptest.NewRecorder()
    router.ServeHTTP(w, req)

    assert.Equal(t, http.StatusOK, w.Code)

    var response MockResponse
    json.Unmarshal(w.Body.Bytes(), &response)
    assert.Equal(t, "註冊成功", response.Message)
}
```

```
func TestLogin(t *testing.T) {
    router := setupRouter()

    loginInput := map[string]string{
        "email":    "test@example.com",
        "password": "password123",
    }

    jsonValue, _ := json.Marshal(loginInput)
    req, _ := http.NewRequest("POST", "/user/login", bytes.NewBuffer(jsonValue))
    req.Header.Set("Content-Type", "application/json")

    w := httptest.NewRecorder()
    router.ServeHTTP(w, req)

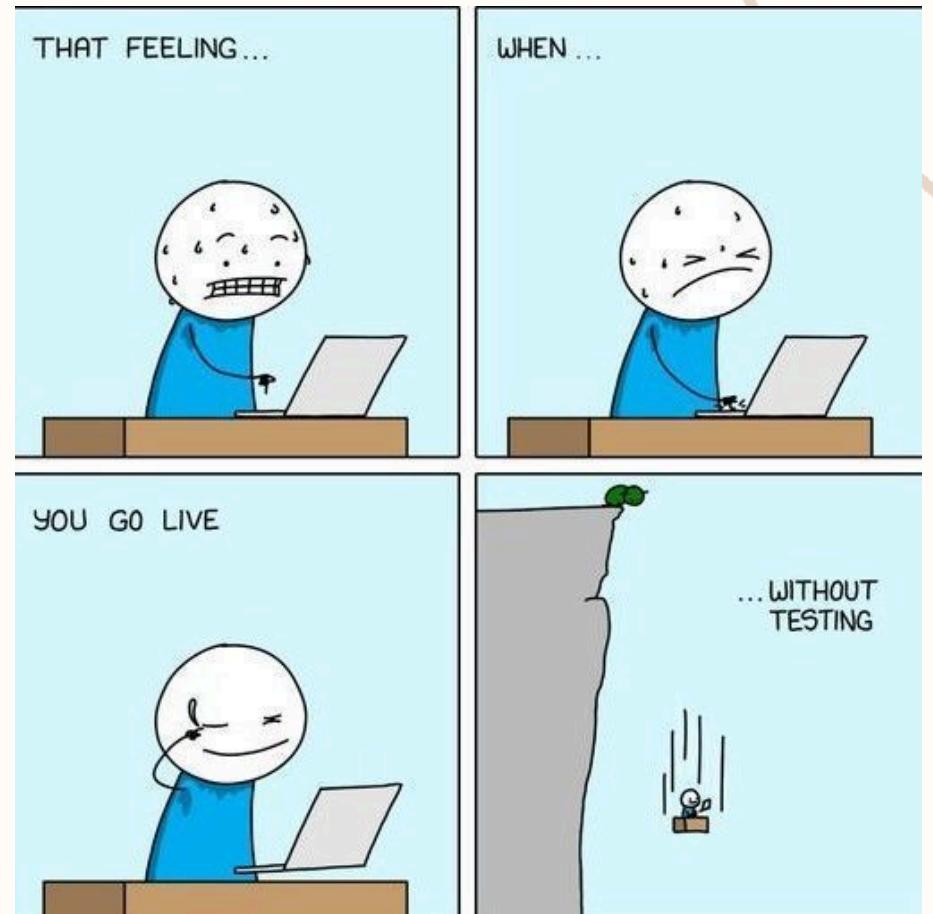
    assert.Equal(t, http.StatusOK, w.Code)

    var response map[string]interface{}
    json.Unmarshal(w.Body.Bytes(), &response)
    assert.Equal(t, "登入成功", response["message"])
    assert.NotEmpty(t, response["token"])
}
```

# Go 單元測試

測試功能是否正常

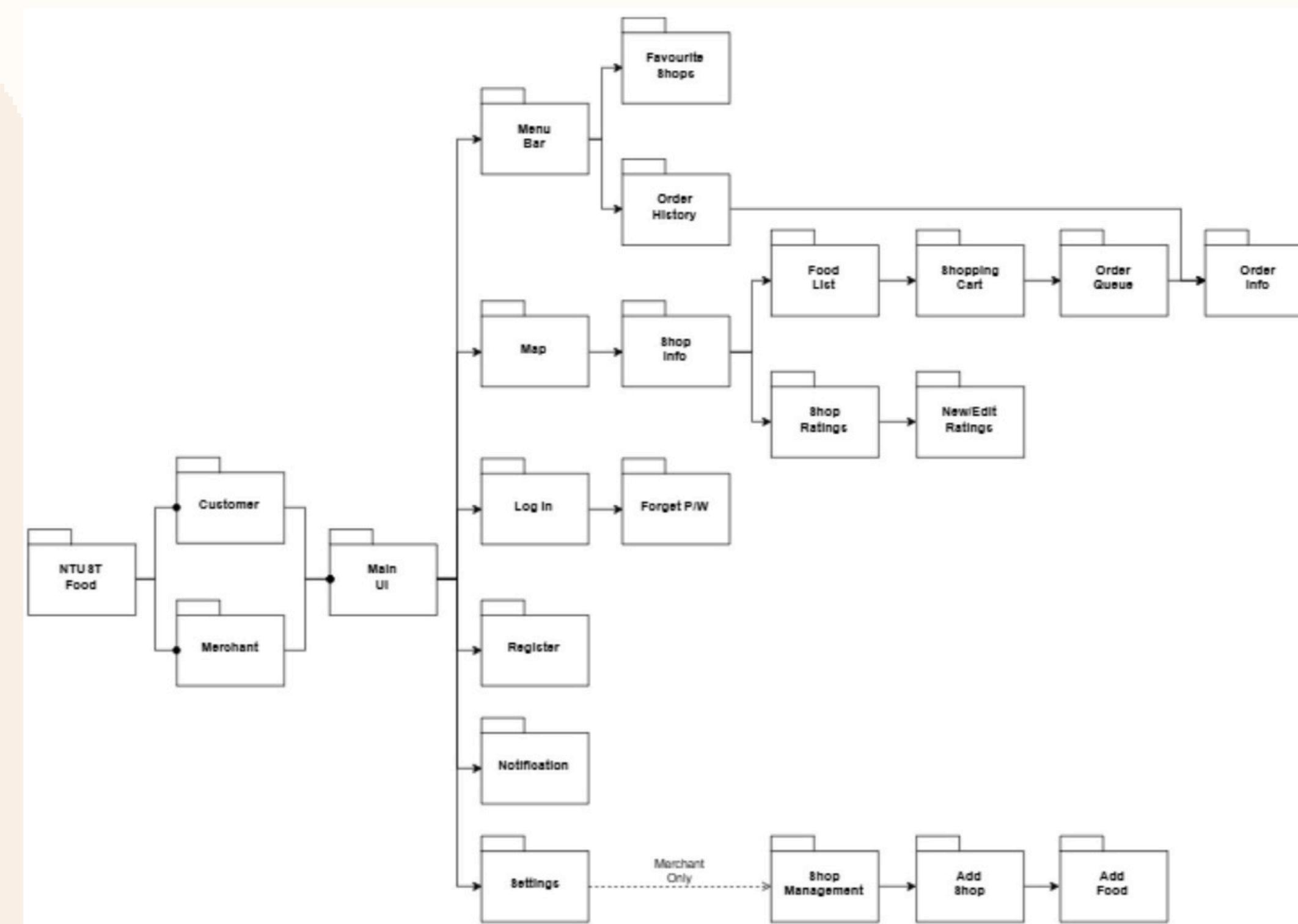
```
PS C:\Users\user\Desktop\SE_project-main\SE_project-main> go test ./controller -run TestRegister
ok      orderfood/controller 0.223s
PS C:\Users\user\Desktop\SE_project-main\SE_project-main> go test ./controller -run TestLogin
ok      orderfood/controller 0.152s
PS C:\Users\user\Desktop\SE_project-main\SE_project-main>
```



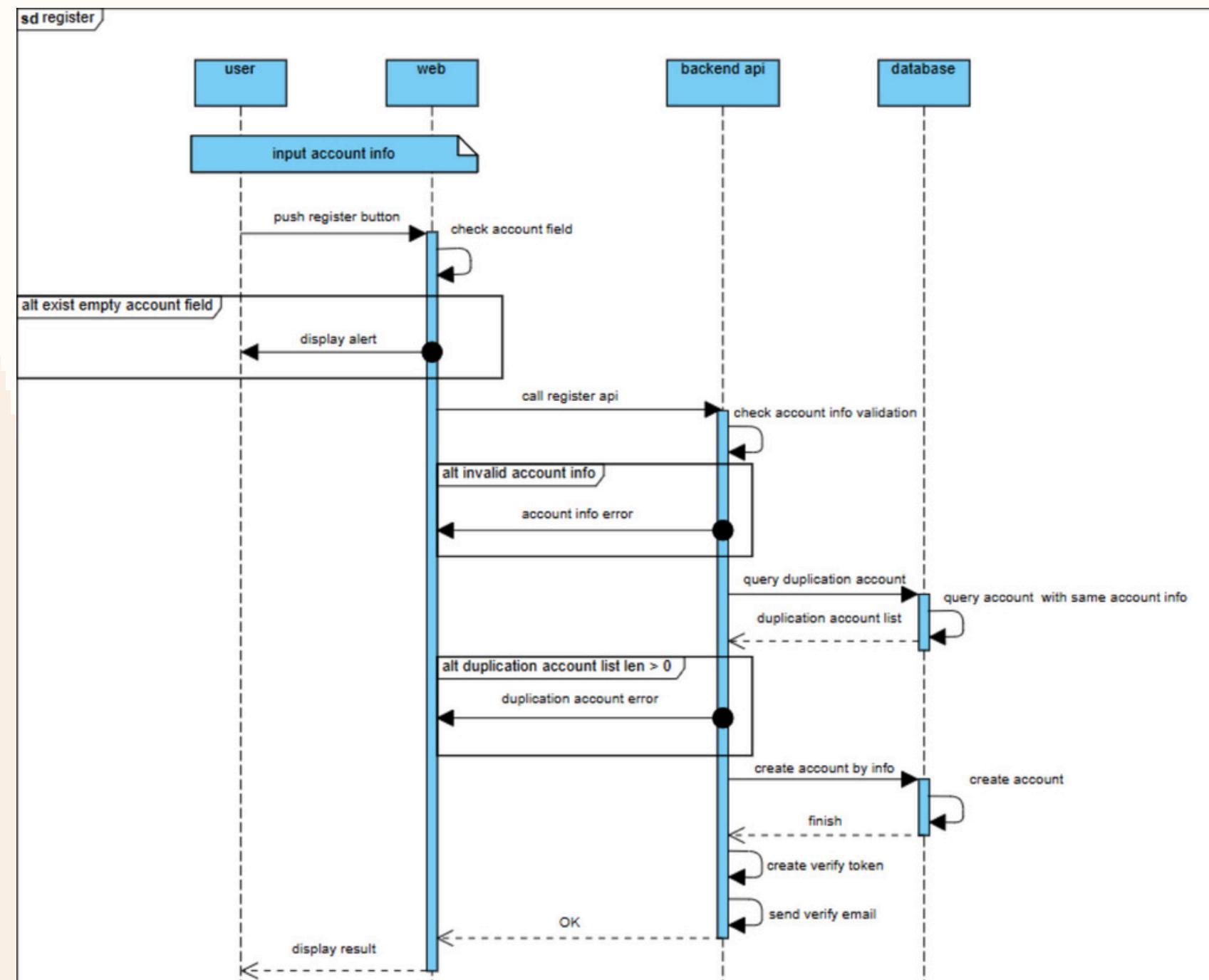


Agile

# 台科點餐系統 Component design



# 台科點餐系統 Sequential Diagram



# Jira testing setting



# Jira testing setting

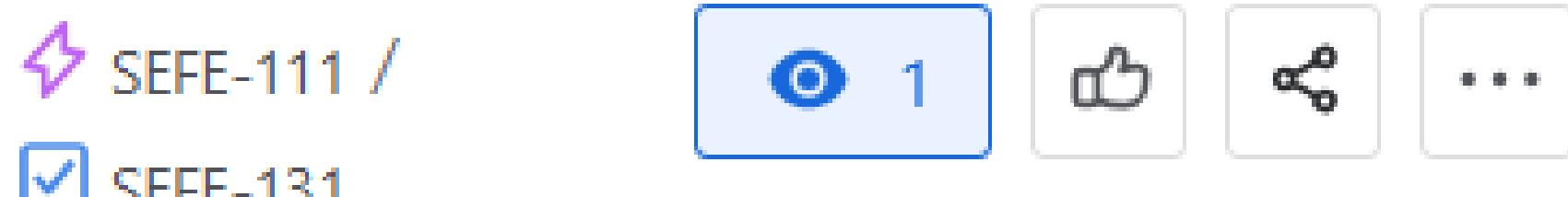
## 1. 在衝刺專案設置Testing看板

The screenshot shows a Jira board titled "NTUST Food" with the following details:

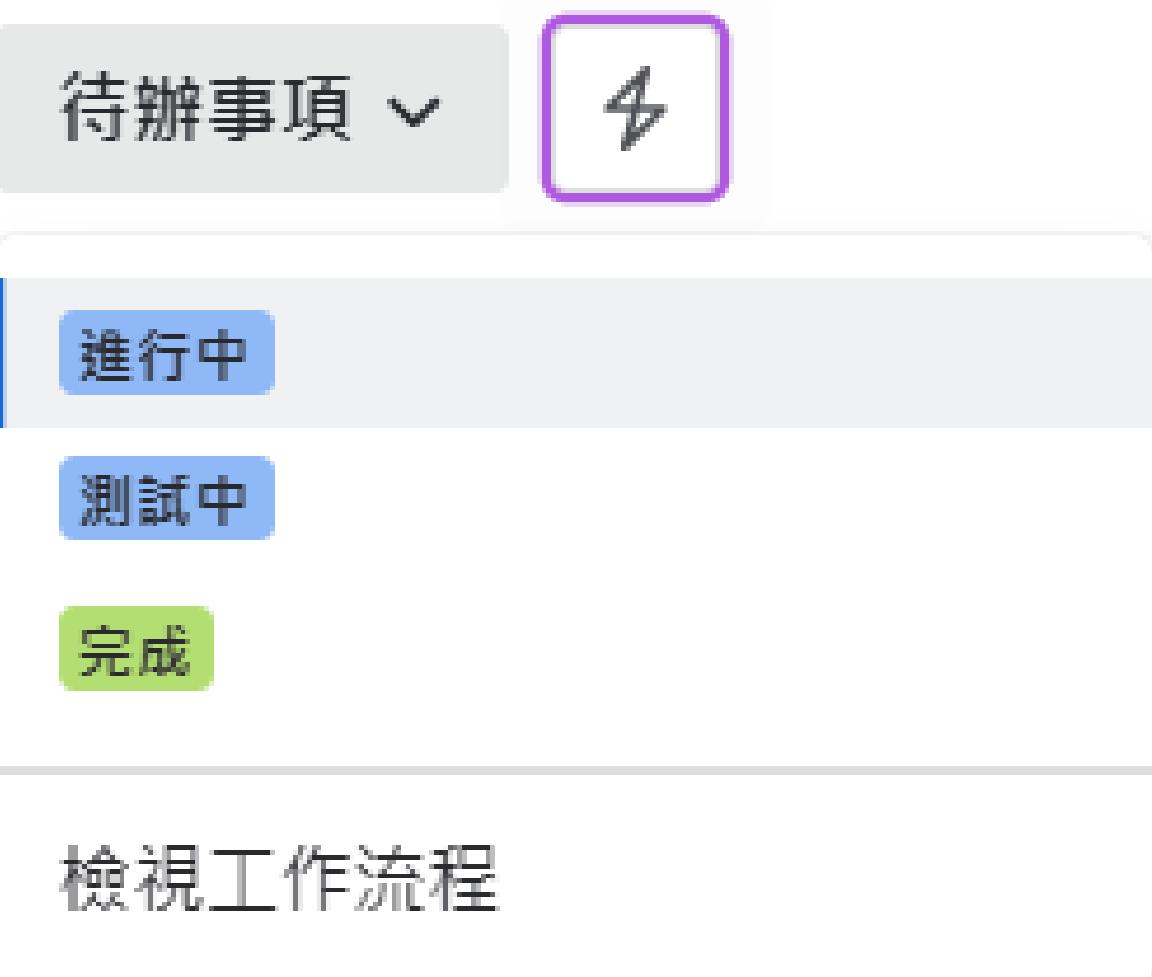
- Top Navigation:** 摘要 (Summary), 時間軸 (Timeline), 待辦事項 (Backlog), 看板 (Board), 行事曆 (Calendar), 報告 (Report), 列表 (List), 頁面 (Page), 捷徑 (Shortcuts), and a plus sign (+).
- Search Bar:** 搜尋看板 (Search Board) with a magnifying glass icon.
- Filter Buttons:** 大型工作 (Large Work) and 衝刺 (Sprint).
- Columns:** TO DO (3 items), IN PROGRESS (1 item), TESTING (6 items, highlighted with a red border), and DONE (1 item).
- TO DO Column:** 菜單瀏覽畫面 (查詢餐廳功能, SEFE-131), 餐廳資訊畫面繪製 (前端畫面繪製, SEFE-98), 菜單瀏覽畫面繪製 (前端畫面繪製, SEFE-132). Each item has a person icon.
- IN PROGRESS Column:** 餐廳資訊畫面 (查詢餐廳功能, SEFE-119). Each item has a person icon.
- TESTING Column:** 抽屜 (使用者工具, SEFE-12), 工具列 (使用者工具, SEFE-19), API 整合測試 (SEFE-168), 登入畫面 (登入/註冊功能, SEFE-113), 註冊畫面 (登入/註冊功能, SEFE-113). The "抽屜" item is highlighted with a red border. Each item has a person icon.
- DONE Column:** 登入/註冊功能 (SEFE-113). Each item has a checkmark icon.

# Jira testing setting

2. 每個任務上的狀態列表會與看板同步更新
3. 將正在衝刺中的待測試任務狀態設為測試中



菜單瀏覽畫面



# Jira testing setting

4. 衝刺中的所有任務都會依照狀態擺放於對應的看板，可以在上方過濾查看特定衝刺的任務

5. 測試結束將任務狀態改為完成，任務會移至 DONE 看板

# Jira testing setting

6. 在jira應用程式搜尋katalon並安裝

The screenshot shows the Katalon - Test Automation for Jira app page. At the top, there's a logo consisting of a black square with a white diagonal line and a green square below it. Next to the logo, the app name "Katalon - Test Automation for Jira" is displayed in bold black text, followed by "by Katalon Inc." Below the name, there are three sections: "OVERALL RATINGS" showing a 3.1/4 rating with four orange stars and 34 reviews; "INSTALLS" showing 1,924 installations with a downward arrow icon; and "TRUST SIGNALS" showing the "CLOUD FORTIFIED" badge. In the top right corner, there are two buttons: a blue "Added" button with a checkmark and a white "Manage app" button. At the bottom of the screenshot, there are three navigation tabs: "Overview" (which is underlined in blue), "Privacy & Security", and "Support".

A simple and powerful software testing management and test automation solution for Jira with GPT powered manual test generation

# Jira testing setting

專案 / NTUST Food / 新增 大型工作 / SEFE-168

## API 整合測試

+ 新增 應用程式

描述

Katalon Manual Tests (BETA)

編輯描述

+ Katalon Manual Tests (BETA)

### Katalon Manual Tests (BETA)

Test Case 1: Verify successful API integration

No	Step	Expected result	Test data
1	Send a GET request to the API endpoint with valid parameters	The API returns a 200 OK status with the expected data structure	Endpoint: /api/v1/resource, Parameters: {"id": 123}
2	Validate the response data against the expected schema	The response data matches the expected JSON schema with all required fields present	Expected Schema: {"id": "number", "name": "string", "status": "string"}
3	Send a POST request to the API endpoint with valid data	The API returns a 201 Created status and the resource is added successfully	Endpoint: /api/v1/resource, Payload: {"name": "Test Resource", "status": "active"}
4	Verify the newly created resource by sending a GET request	The API returns a 200 OK status with the correct resource data	Endpoint: /api/v1/resource/124

> Test Case 2: Handle API error responses gracefully

測試中

4

詳細資料

受託人

未指派

指派給我

標籤

無

父系

無

Team

無

Sprint

Testing

Story point estimate

無

開發

建立分支

# Jira testing setting

如何將測試結果自動上傳到Jira ?

# Jira testing setting

## 1. 創建Jira api token

### 建立 API 權杖

為您的權杖指定名稱，以描述其功能，並選取到期日。

Required fields are marked with an asterisk \*

Name \*

NTUST Food

例如：APItoken1\_updated\_daily\_report

Expires on \*

2024/12/26



For your security, tokens can last no longer than a year.

建立 API 權杖即表示您接受 Atlassian 開發者條款，且同意隱私政策。

取消

建立

# Jira testing setting

2. 藉由測試工具生成Junit.xml
3. 撰寫自動上傳腳本

```
const JIRA_URL = 'https://sefrontend.atlassian.net'  
const ISSUE_KEY = 'SEFE-170'  
const EMAIL = 'ian523411756@gmail.com'  
const API_TOKEN = 'ATATT3xFfGF0rTUFKMkWR-Yay3EsUkBIXBT5MiagIV  
  
const axios = require('axios'); 62.8k (gzipped: 23.2k)  
const fs = require('fs');  
const xml2js = require('xml2js'); 98.4k (gzipped: 22.1k)  
  
const reportFile = './test/test_results/junit.xml';  
const reportContent = fs.readFileSync(reportFile, 'utf-8');  
  
xml2js.parseString(reportContent, (err, result) => {  
    if (err) throw err;  
  
    const testsuite = result.testsuites.testsuite[0];  
    const passed = testsuite.$.tests - testsuite.$.failures;  
    const failed = testsuite.$.failures;  
    const duration = testsuite.$.time;
```

```
const comment = {
  body: {
    type: "doc",
    version: 1,
    content: [
      {
        type: "paragraph",
        content: [
          {
            text: ` 📈 測試結果報告\n✓ 通過測試: ${passed}\n✗ 失敗測試: ${failed}\n⌚ 測試時間: ${duration}s`,
            type: "text"
          }
        ]
      }
    ]
  }
};

// 發送至 Jira API
axios.post(
  `${JIRA_URL}/rest/api/3/issue/${ISSUE_KEY}/comment`,
  { ...comment },
  {
    auth: { username: EMAIL, password: API_TOKEN },
    headers: { 'Content-Type': 'application/json' },
  }
)
.then(() => console.log('✓ 測試結果已成功上傳至 Jira!'))
.catch(error => console.error('✗ 上傳失敗:', error.response.data));
});
```

# Jira testing setting

## 4. 運行腳本

```
D:\github\SE_project_frontend>node ./test/autoUpload.js  
✓ 測試結果已成功上傳至 Jira!
```

# Jira testing setting

## 5. 檢查上傳之report comment是否正確

### 測試結果

+ 新增

◎ 應用程式

### 描述

編輯描述

### 活動

顯示: 全部 留言 歷程記錄 工作記錄



新增留言...

 看起來不錯!

 需要協助嗎?

專業秘訣：按下 **M** 進行留言



山羊 2 分鐘前

 測試結果報告

 通過測試: 5

 失敗測試: 1

 測試時間: 0.012719s

編輯 · 刪除 · 



前端 Unit Testing + Api  
Integration Testing

# 前端 Unit Testing

前面描述如何將測試報告自動上傳到jira

現在講解我們如何生成測試報告(以nuxt3 framework 為例)

# 前端 Unit Testing

Nuxt3 framework 可使用vitest做為測試工具，並將測試報告輸出成junit.xml

## Login component 單元測試範例

```
describe('LoginComponent 測試', () => {
  it('應該渲染 HTML 結構', () => {
    const wrapper = mount(Login) // 渲染組件

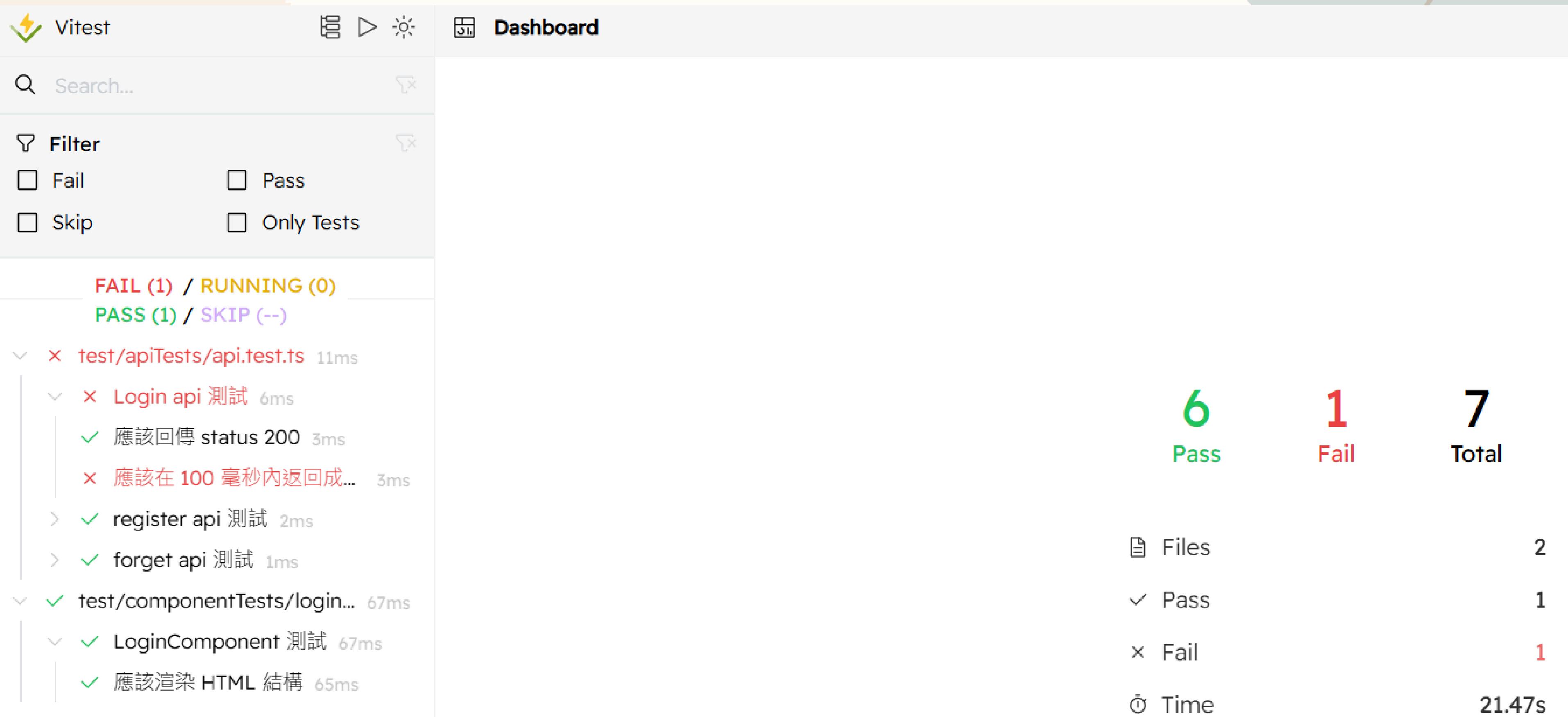
    // 測試 HTML 結構
    expect(wrapper.html()).toContain('email')
    expect(wrapper.html()).toContain('password')

    // 檢查是否渲染某個元素，例如 button
    expect(wrapper.find('button')).exists().toBe(true)

    // 輸出完整 HTML 結構(可選)
    console.log(wrapper.html())
  })
})
```

# 前端 Unit Testing

## 測試結果視覺化



# Api Intergration Testing

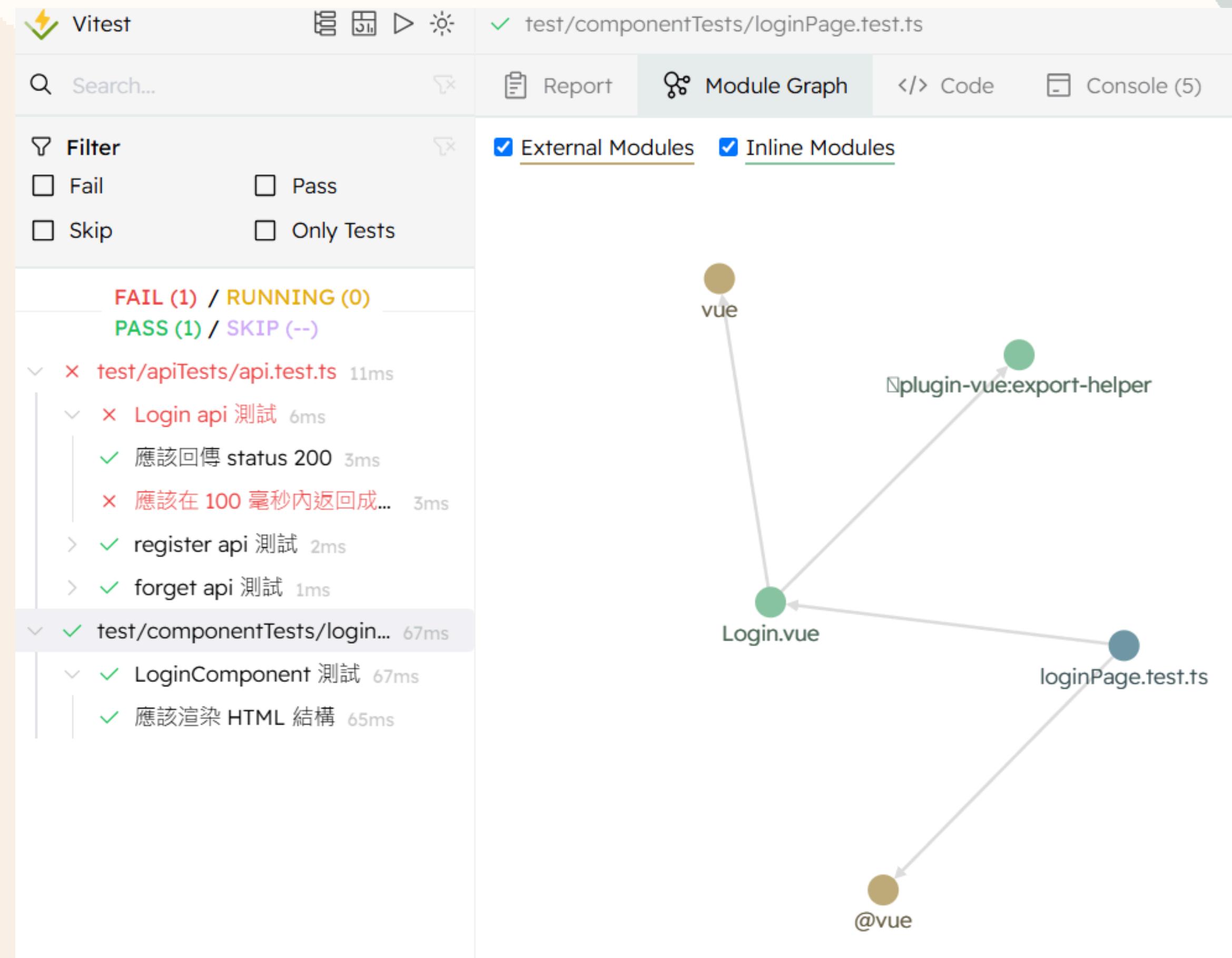
測試結果視覺化

The screenshot shows the Vitest interface with the following details:

- Top Bar:** Vitest logo, file navigation icons, file path: `test/apiTests/api.test.ts`.
- Search Bar:** Search input field.
- Filter Buttons:** Fail, Pass, Skip, Only Tests.
- Status Summary:** FAIL (1) / RUNNING (0), PASS (1) / SKIP (--).
- Test Suite:** `x test/apiTests/api.test.ts` (11ms)
  - Test Case:** `x Login api 測試` (6ms)
    - Assertion:** 應該回傳 status 200 (3ms) - Passed
    - Assertion:** 應該在 100 毫秒內返回成... (3ms) - Failed (AssertionError: expected 779.0535 to be less than 100)
  - Test Case:** `> register api 測試` (2ms) - Passed
  - Test Case:** `> forget api 測試` (1ms) - Passed
- Other Test Suites:** `✓ test/componentTests/login...` (67ms), `✓ LoginComponent 測試` (67ms), `✓ 應該渲染 HTML 結構` (65ms) - All passed.

# 前端 Unit Testing

測試元件依賴圖





# Refactoring

# Refactoring\_1: User Login

Original Code has the following issues:

1. Poor Naming
2. Lack of Encapsulation
3. Hard-coded Error Messages

```
1 func Login(c *gin.Context) {
2     // Parse input
3     var u models.UserBasic
4     err := c.BindJSON(&u)
5     if err != nil {
6         c.JSON(400, gin.H{"msg": "Bad input"})
7         return
8     }
9
10    if u.Email == "" || u.Password == "" {
11        c.JSON(400, gin.H{"msg": "Missing fields"})
12        return
13    }
14
15    // Fetch user from DB
16    user, fetchErr := utils.FindUserByEmail(u.Email)
17    if fetchErr != nil {
18        c.JSON(500, gin.H{"msg": "Fetch user failed"})
19        return
20    }
21    if user == nil || !user.Exists() {
22        c.JSON(404, gin.H{"msg": "User not found"})
23        return
24    }
25
26    // Decode user
27    var dbUser models.UserBasic
28    decodeErr := user.DataTo(&dbUser)
29    if decodeErr != nil {
30        c.JSON(500, gin.H{"msg": "User decode error"})
31        return
32    }
33
34    // Check password
35    passErr := bcrypt.CompareHashAndPassword([]byte(dbUser.Password), []byte(u.Password))
36    if passErr != nil {
37        c.JSON(401, gin.H{"msg": "Wrong password"})
38        return
39    }
40
41    // Generate token
42    tk, tkErr := utils.GenerateToken(dbUser.ID)
43    if tkErr != nil {
44        c.JSON(500, gin.H{"msg": "Token generation failed"})
45        return
46    }
47
48    // Respond success
49    c.JSON(200, gin.H{
50        "msg": "Success",
51        "token": tk,
52    })
53 }
```

original

```
1 func Login(c *gin.Context) {
2     // 使用 c.BindJSON 來從請求中解析 JSON 到 UserBasic 物件
3     var userInput models.UserBasic
4     if err := c.BindJSON(&userInput); err != nil {
5         response.Fail(c, nil, "無效的輸入格式")
6         return
7     }
8
9     // 打印輸入資料
10    println("email: ", userInput.Email)
11    println("password: ", userInput.Password)
12
13    // 檢查輸入資料
14    if userInput.Email == "" || userInput.Password == "" {
15        response.Fail(c, nil, "請輸入完整資料")
16        return
17    }
18
19    // 處理並開啟文件
20    user, err := utils.FindUserByEmail(userInput.Email)
21
22    if err != nil {
23        response.Fail(c, nil, "查詢用戶失敗")
24        return
25    }
26
27    // 檢查用戶是否存在
28    if user == nil || !user.Exists() {
29        response.Fail(c, nil, "用戶不存在")
30        return
31    }
32
33    // 使用 User 延伸方法驗證密碼
34    var userBasic models.UserBasic
35    if err := user.DataTo(&userBasic); err != nil {
36        log.Println("Failed to decode document into User struct: %v", err)
37        response.Fail(c, nil, "帳號解析失敗")
38        return
39    }
40
41    // 驗證密碼
42    if err := bcrypt.CompareHashAndPassword([]byte(userBasic.Password), []byte(userInput.Password)); err != nil {
43        response.Fail(c, nil, "密碼錯誤")
44        return
45    }
46
47    // 發放token
48    token, err := utils.GenerateToken(userBasic.ID)
49    if err != nil {
50        response.Fail(c, nil, "token發放失敗")
51        return
52    }
53
54    response.Success(c, gin.H{
55        "token": token,
56    }, "登入成功")
57 }
```

refactored

# Refactoring\_2: User Registration

```
1 func Register(c *gin.Context) {
2     // 使用 c.BindJSON 來從請求中解碼 JSON 到 UserBasic 結構體
3     var userInput models.UserBasic
4     if err := c.BindJSON(&userInput); err != nil {
5         response.Fail(c, nil, "無效的輸入格式")
6         return
7     }
8
9     // 驗證是否有缺失的字段
10    if userInput.Email == "" || userInput.Password == "" || userInput.Name == "" || userInput.Phone == "" || userInput.Roles == "" {
11        response.Fail(c, nil, "請輸入完整資料")
12        return
13    }
14
15    // 檢查用戶是否已經存在
16    user, err := utils.FindUserByEmail(userInput.Email)
17    if err != nil {
18        response.Fail(c, nil, "查詢用戶失敗")
19        return
20    }
21
22    if user != nil && user.Exists() {
23        response.Fail(c, nil, "用戶已存在")
24        return
25    }
26
27    // 加密密碼
28    hasedPassword, err := bcrypt.GenerateFromPassword([]byte(userInput.Password), bcrypt.DefaultCost)
29    if err != nil {
30        response.Fail(c, nil, "加密密碼失敗")
31        return
32    }
33
34    // 生成 UUID 並設置到用戶結構體中
35    UUID := uuid.New().String()
36    userInput.ID = UUID
37    userInput.Password = string(hasedPassword)
38    userInput.IsEnabled = true // 註冊時設置為默認禁用
39
40    // 將用戶數據儲存到 Firestore
41    _, err = utils.FirestoreClient.Collection("users").Doc(UUID).Set(c, userInput)
42    if err != nil {
43        response.Fail(c, nil, "註冊失敗")
44        return
45    }
46
47    response.Success(c, nil, "註冊成功")
48 }
```

original

```
1 func Register(c *gin.Context) {
2     var userInput models.UserBasic
3     if !CheckInputFormat(c, userInput) {return}
4     if !CheckMissingFields(c, userInput) {return}
5     if !CheckUserExistence(c, userInput.Email) {return}
6     if !EncryptPassword(c, &userInput) {return}
7     if !StoreUserDataInFirestore(c, userInput) {return}
8     Response(c, true, "註冊成功", nil)
9 }
10
11 func CheckInputFormat(c *gin.Context, userInput *models.UserBasic) bool {
12     if err := c.BindJSON(userInput); err != nil {
13         Response(c, false, "無效的輸入格式", nil)
14         return false
15     }
16     return true
17 }
18
19 func CheckMissingFields(c *gin.Context, userInput *models.UserBasic) bool {
20     if userInput.Email == "" || userInput.Password == "" || userInput.Name == "" || userInput.Phone == "" || userInput.Roles == "" {
21         Response(c, false, "請輸入完整資料", nil)
22         return false
23     }
24     return true
25 }
26
27 func CheckUserExistence(c *gin.Context, email string) bool {
28     user, err := utils.FindUserByEmail(email)
29     if err != nil {
30         Response(c, false, "查詢用戶失敗", nil)
31         return false
32     }
33
34     if user != nil && user.Exists() {
35         Response(c, false, "用戶已存在", nil)
36         return false
37     }
38     return true
39 }
40
41 func EncryptPassword(c *gin.Context, userInput *models.UserBasic) bool {
42     hasedPassword, err := bcrypt.GenerateFromPassword([]byte(userInput.Password), bcrypt.DefaultCost)
43     if err != nil {
44         Response(c, false, "加密密碼失敗", nil)
45         return false
46     }
47     userInput.Password = string(hasedPassword)
48     return true
49 }
50
51 func SetUserStructure(userInput *models.UserBasic) {
52     UUID := uuid.New().String()
53     userInput.ID = UUID
54     userInput.IsEnabled = true
55 }
56
57 func StoreUserDataInFirestore(c *gin.Context, userInput models.UserBasic) bool {
58     _, err := utils.FirestoreClient.Collection("users").Doc(userInput.ID).Set(c, userInput)
59     if err != nil {
60         Response(c, false, "註冊失敗", nil)
61         return false
62     }
63     return true
64 }
65
66 func Response(c *gin.Context, success bool, message string, data interface{}) {
67     if success {
68         response.Success(c, data, message)
69     } else {
70         response.Fail(c, data, message)
71     }
72 }
73 }
```

refactored

# Refactoring\_3: Require\_Input

```
<template>
  <v-row class="h-full" :class="ac.getThemeStyle('primary', 'background')">
    <v-spacer v-if="viewport.isGreater Than('tablet')"/>
    <div class="flex flex-col flex-grow justify-center items-center p-8">
      <div class="mb-8" :class="ac.getThemeStyle('primary', 'title')">
        {{ mode == 'login' ? '登入' : (mode == 'register' ? '註冊' : '重設密碼') }}
      </div>
      <template v-if="mode == 'login'">
        <div class="flex flex-col w-full">
          <div v-if="showTitle" class="mb-2" :class="ac.getThemeStyle('primary', 'inputTitle')">
            <div class="w-full mb-2" :class="ac.getThemeStyle('primary', 'input0').concat([getInputPlaceholder('email')])">
              <input class="w-full" :class="ac.getThemeStyle('primary', 'inputI')" type="text" :placeholder="placeholderEmail" />
            </div>
            <div v-if="!check" class="text-red-500 text-sm pl-6 mb-2">{{remind}}{{remind}}
```

original

1. Repeated <input> tags
2. Hard-coded processing
3. Readability Issue

```
<template>
  <v-row class="h-full" :class="ac.getThemeStyle('primary', 'background')">
    <v-spacer v-if="viewport.isGreater Than('tablet')"/>
    <div class="flex flex-col flex-grow justify-center items-center p-8">
      <div class="mb-8" :class="ac.getThemeStyle('primary', 'title')">
        {{ mode == 'login' ? '登入' : (mode == 'register' ? '註冊' : '重設密碼') }}
      </div>
      <template v-if="mode == 'login'">
        <RequireInput placeholder="email" v-model:val="email" v-model:trigger="sendCheck"/>
        <RequireInput placeholder="password" v-model:val="password" v-model:trigger="sendCheck" type="password"/>
        <div class="flex flex-row mt-4">
          <button class="h-16 w-40 text-3xl mr-4" :class="ac.getThemeStyle('primary', 'button')" @click="switchMode('register')">註冊</button>
          <button class="h-16 w-40 text-3xl" :class="ac.getThemeStyle('secondary', 'button')" @click="login()">>登入</button>
        </div>
        <button class="mt-8" :class="ac.getThemeStyle('third', 'button')" @click="switchMode('reset')">>忘記密碼</button>
      </template>
      <template v-else-if="mode == 'register'">
        <RequireInput placeholder="name" v-model:val="name" v-model:trigger="sendCheck"/>
        <RequireInput placeholder="email" v-model:val="email" v-model:trigger="sendCheck"/>
        <RequireInput placeholder="phone" v-model:val="phone" v-model:trigger="sendCheck"/>
        <RequireInput placeholder="password" v-model:val="password" v-model:trigger="sendCheck" type="password"/>
        <RequireInput placeholder="check password" v-model:val="secondPassword" v-model:trigger="sendCheck" :sameValue="true"/>
        <button class="h-16 w-full text-3xl" :class="ac.getThemeStyle('secondary', 'button')" @click="register()">>{{'註冊' /*寄送註冊確認信*/}}</button>
      </template>
    </div>
  </v-row>
```

refactored

-- Method & Class Extraction

# 工作分配

B11115055 游騰楓	報告、後端程式
B11115049 黃羿華	報告、前端程式
B11130034 游傳智	重構、文件
B11130005 鍾昊成	重構、文件
B11115021 徐家萱	單元測試、後端程式
B11115016 Jin Woojun	前端文件、報告
B11005121 許菀真	前端文件、報告

謝謝！

Q&A