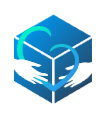
靜宜大學

Providence University



IPFS幣記

實驗室: 507 實驗室

指導教師:謝孟諺 老師

專題學生:(班級)(學號)(姓名)

(資工四B)(410919132)(林映如)

(資工四B)(410919158)(楊廷烽)

(資工四B)(410928149)(謝孟峻)

(資工四B)(410918819)(蔡侄宇)

民國112年12月14日

致謝

專題已經進入尾聲，誠摯感謝指導我們畢業專題的導師 – 謝孟諺教授。在整個製作過程中，我們遇到了許多困難，教授總是無私的指導和提供寶貴的建議，引領我們朝著正確的方向前進，並從學習中解決遇到的困難。

專題一開始，我們對 Web3 和區塊鏈技術一無所知，教授提供豐富的相關資料，並利用課後時間跟我們開會，檢視我們的進度，以確保我們能夠克服挑戰並不斷進步。在研究的過程中，我們逐漸對這個領域產生了濃厚的興趣。教授鼓勵我們深入研究區塊鏈技術的應用，並在他的指導下成功完成了兩篇學術論文。

最後，我們要感謝學校提供了優越的設備和資源，使我們得以順利完成本專題研究。系上提供的軟硬體設備和專題實驗室也對我們的研究工作有著重要的支持和貢獻。再次感謝謝孟諺教授的悉心指導和學校的支持，讓我們能夠順利完成這個畢業專題。

目錄

[一、前言 3](#_heading=h.t5nohrt9so9x)

[二、研究動機與目標 4](#_heading=h.g2m168219v42)

[(一)研究動機 4](#_heading=h.t9lvvux4vbdo)

[(二)研究目標 4](#_heading=h.yrxo5ehd4xkn)

[三、背景知識介紹 5](#_heading=h.6swtzv21c4pt)

[四、系統功能 7](#_heading=h.qa7telumj39w)

[五、研究方法及步驟 10](#_heading=h.6swtzv21c4pt)

[(一)初始概念和規劃 10](#_heading=h.4unq4shdp5vd)

[(二)系統架設 10](#_heading=h.jggfk7kky0k5)

[(三)進一步規劃和測試 12](#_heading=h.j2d6akuewex5)

[(四)最後修正與美觀 15](#_heading=h.jie513e1cxe)

[六、系統架構與設計 16](#_heading=h.zhv53ph6vhfo)

[七、安裝與實作流程 20](#_heading=h.cu5sebhebp9o)

[(一)前置作業 20](#_heading=h.5z3tp5prl26u)

[(二)部署監控與品質控管服務 20](#_heading=h.id0e6qaj237u)

[(三)前端、後端、安裝初始化 IPFS 22](#_heading=h.jq9ncipw9kgt)

[(四) Go-ethereum & BlockScout 架設 29](#_heading=h.6snx1ixhowpo)

[(五)智能合約上傳至區塊鏈 33](#_heading=h.ohtt0thahay3)

[(六) MetaMask 錢包設定(智能合約交易需要使用) 38](#_heading=h.t6r5l7r9d8au)

[八、實驗結果與展示 39](#_heading=h.on06i3s7apmz)

[九、結論與未來規劃 47](#_heading=h.sxii6fhnkxew)

[十、參考資料及網站 48](#_heading=h.b3oo4p80em29)

# 一、前言

隨著科技的迅速進展，互聯網已經成為人們分享知識和資訊的主要平台。然而，傳統的中心化平台存在著單點故障風險以及資料可信度的挑戰。為了解決這些問題，本專題建立了一個基於分散式網路和區塊鏈技術的文章發布平台，

本專題採用 IPFS (InterPlanetary File System) 作為文章的發布媒介，這使得文章能夠分散存儲在多個節點之間，確保了文章的持久性和可用性。同時，我們設計了智能合約，以實現以下三個目標(1)在區塊鏈上儲存文章的歷史紀錄，確保文章的持久性和可追溯性(2)設計符合 ERC20 協議 的加密代幣 (Appreciative Citizen 簡稱 AC 代幣) ，此代幣可以與以太幣互相轉換，設計代幣旨用於激勵創作者的創作動力(3)設計領取測試以太幣，旨在讓新用戶可以透過領取測試以太幣來熟悉平台操作介面。

此外，我們使用 Go-Ethereum 以太坊客戶端建立了一個私有區塊鏈網路，並利用 BlockScout 區塊鏈瀏覽器平台來查看智能合約交易紀錄，同時透過瀏覽器擴充程式 MetaMask 實現區塊鏈互動和加密貨幣錢包功能。平台所需的資料通過後端 NestJS 和區塊鏈提供，同時我們定義了滿足平台功能需求的資料集。未來，我們將繼續擴展平台功能，以提升使用者體驗和促進社群互動。

# 二、研究動機與目標

## (一)研究動機

隨著科技的快速發展，網路已成為人們分享知識和資訊的主要平台。然而，傳統的中心化網路平台存在一些問題，吸引我們思考如何更好地建構一個開放、去中心化的網路生態系統。

## (二)研究目標

1.單點故障與資料可信度問題:傳統網路平台依賴中心化伺服器架構，容易受到單點故障威脅，影響可用性。此外，資訊和數據可信度也受到質疑，需要更可靠的解決方案。

2.資料持久性和可用性:確保使用者產生的內容和知識的持久性和可用性對於建立可信任的知識共享平台至關重要，需要改善傳統平台在這方面的表現。

3.激勵機制:鼓勵創作者積極參與平台並貢獻高品質內容是重要目標。傳統平台通常依賴廣告收入或其他中心化激勵機制，可能導致不公平的獎勵和降低內容質量，需要更公平和激勵性更強的機制。

此專題解決了這些問題，我們探索了基於分散式技術和區塊鏈的新途徑。我們採用了 IPFS 作為分散式儲存媒介，同時在區塊鏈上儲存文章的歷史記錄，以確保內容的持久性和可追溯性。同時，設計了加密代幣( AC 代幣)作為激勵機制，鼓勵創作者積極參與平台。這些措施在改善現有網路體驗，為用戶提供更好的知識共享和互動機會。

# 三、背景知識介紹

此節將文中提到之專有名詞在此介紹:

**去中心化**

* 去中心化系統中，權力與控制不集中在單一實體或機構中，而是分散到多個節點、個體或組織中。
* 可減少單點故障風險，系統更具彈性。

**IPFS(InterPlanetary File System)星際檔案系統**

* 是一個去中心化的分佈式文件系統和傳輸協議。
* IPFS 不依賴中心化伺服器，而是通過分佈式網絡連接多個節點儲存和傳輸數據。
* 可通過添加節點來增加儲存容量。
* 使用唯一Hash值來標識文件。

**Go-ethereum**

* 是一個完整的以太坊節點，允許使用者在計算機設備上運行一個本地節點，使用者可以和區塊鏈進行交互，包刮發送交易以及部屬智能合約。
* 支持智能合約的創建、部屬和執行。
* 經過廣泛的測試和審核，確保其安全和穩定性。

**NextJS**

* 是一個 React 框架，建構於 Node.js 之上。
* 為網頁應用提供許多額外的架構、功能及優化。

**NestJS**

* 是一個強大且功能完善的 API 框架。
* 吸收其他框架的優點，例如: DI (Dependency Injection), Service Provider, AOP (Aspect Oriented Programming), Decorator 表達式

(類似 Java Spring 的 annotation)...。

**加密代幣**

* 建構於區塊鏈技術之上具有特定用途。
* 是數字形式的資產，代表不同價值和權益，可在區塊鏈進行交換與儲存。
* 使用加密技術保護用戶資產和交易。
* 代幣可基於不同標準創建，例如: ERC-20、 ERC-721 等，每種標準定義代幣特定屬性。

**智能合約**

* 是一種將雙方的協議條款，並用代碼形式在區塊鏈上運行。
* 會完全自動強制執行，公開透明不會被更改。

**MetaMask**

* 用於與以太坊區塊鏈進行互動的軟體加密貨幣錢包。

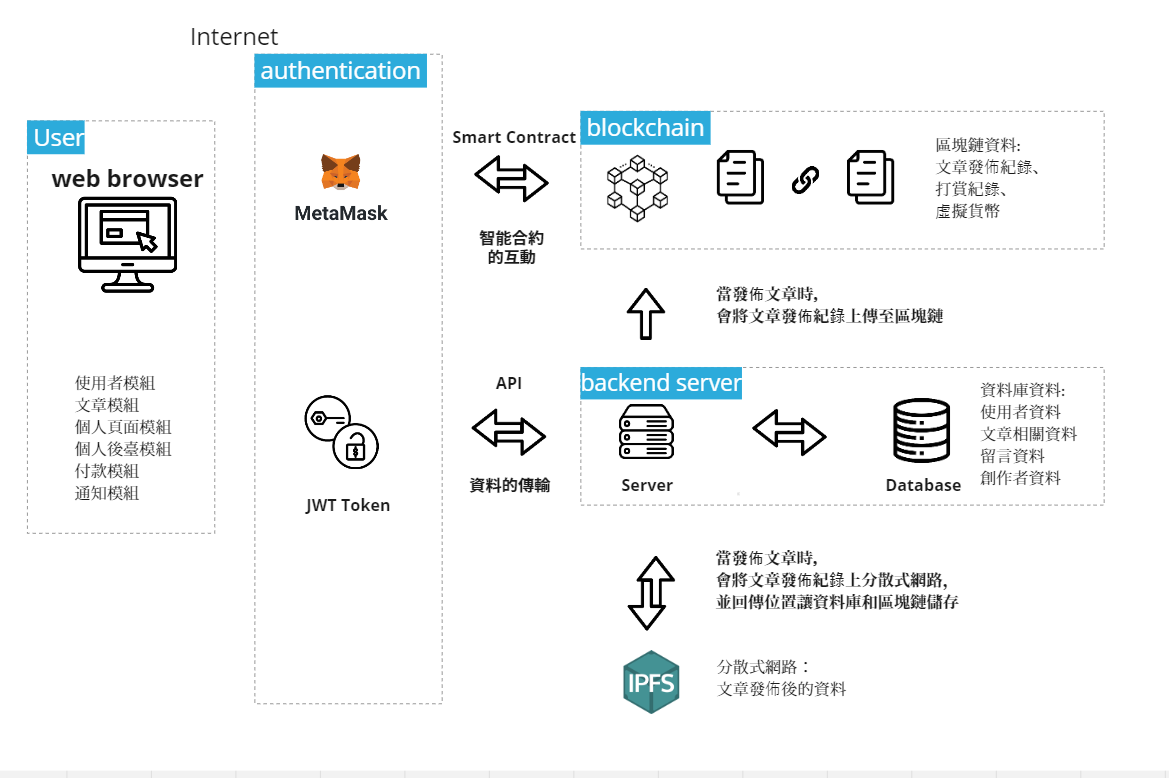
**Docker**

* 一種容器化技術，將應用程序和其依賴項打包成輕量級容器，使其獨立運行
* 可以在不同操作系統上運行，包括 Linux、Windows 和 macOS，確保應用程序在各種環境中保持一致性。

# 

# 四、系統功能

我們的系統架構包括四個關鍵模組：(1)使用者端提供使用者模組、文章模組、個人頁面模組、個人後臺模組、付款模組、通知模組(2)認證模組使用 MetaMask 錢包位址進行使用者註冊和身份驗證，確保安全性和防止未經授權的存取(3)區塊鏈模組透過智能合約處理代幣交換和文章發布紀錄(4)後端伺服器模組使用符合 RESTful API 風格來儲存用戶資料和與文章相關的信息，同時將文章上傳至 IPFS 分散式網路。這些模組共同構成了我們系統的核心功能和架構。



▲【圖一】系統平台

# 

# 

# 五、研究方法及步驟

## (一)初始概念和規劃

最初，我們以 Web3.0 的主題為基礎，試圖尋找適合開發的應用領域。我們探討了多個可能的概念，包括區塊鏈遊戲平台、虛擬貨幣購物平台等等。在經過可行性、難度、資源等多方面的考慮和評估後，我們最終決定設計一個去中心化文章平台。為實現這個平台，我們計劃利用 IPFS 、區塊鏈技術，並設計網頁前端和後端相關功能。

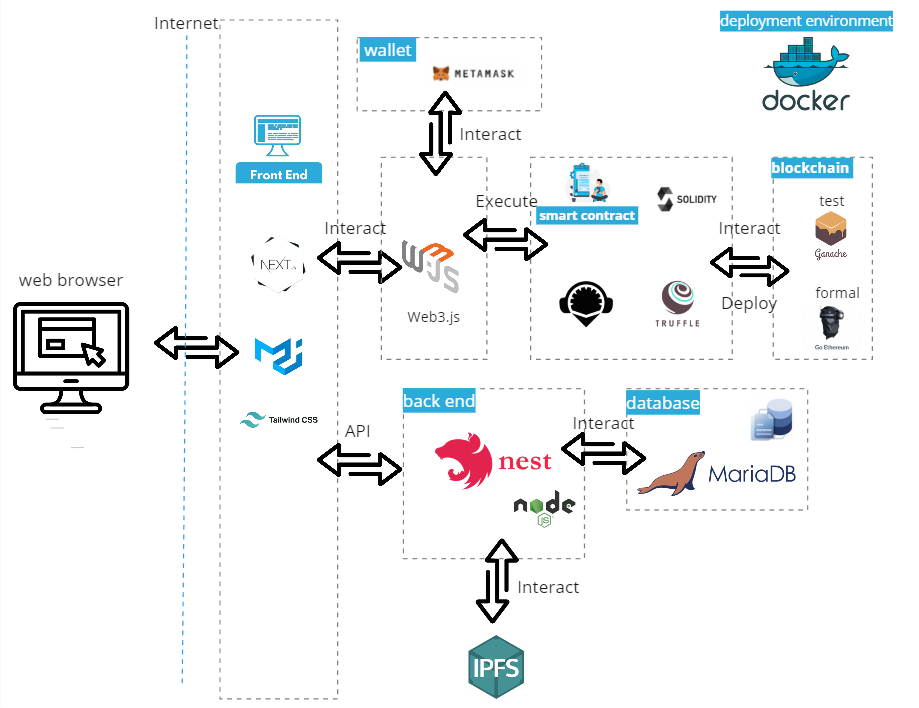
## (二)系統架設

確立方向後，我們開始收集所有相關資料，並逐步架設整個系統。

　　我們使用環境 Docker 環境建立整個系統。前端框架使用 NextJS， UI 函式庫有Tailwind CSS、 Material UI，而功能、資料處理使用 Axios 方式 介接 API，透過 Web3.js 跟 MetaMask 和智能合約互動取得授權和資料。

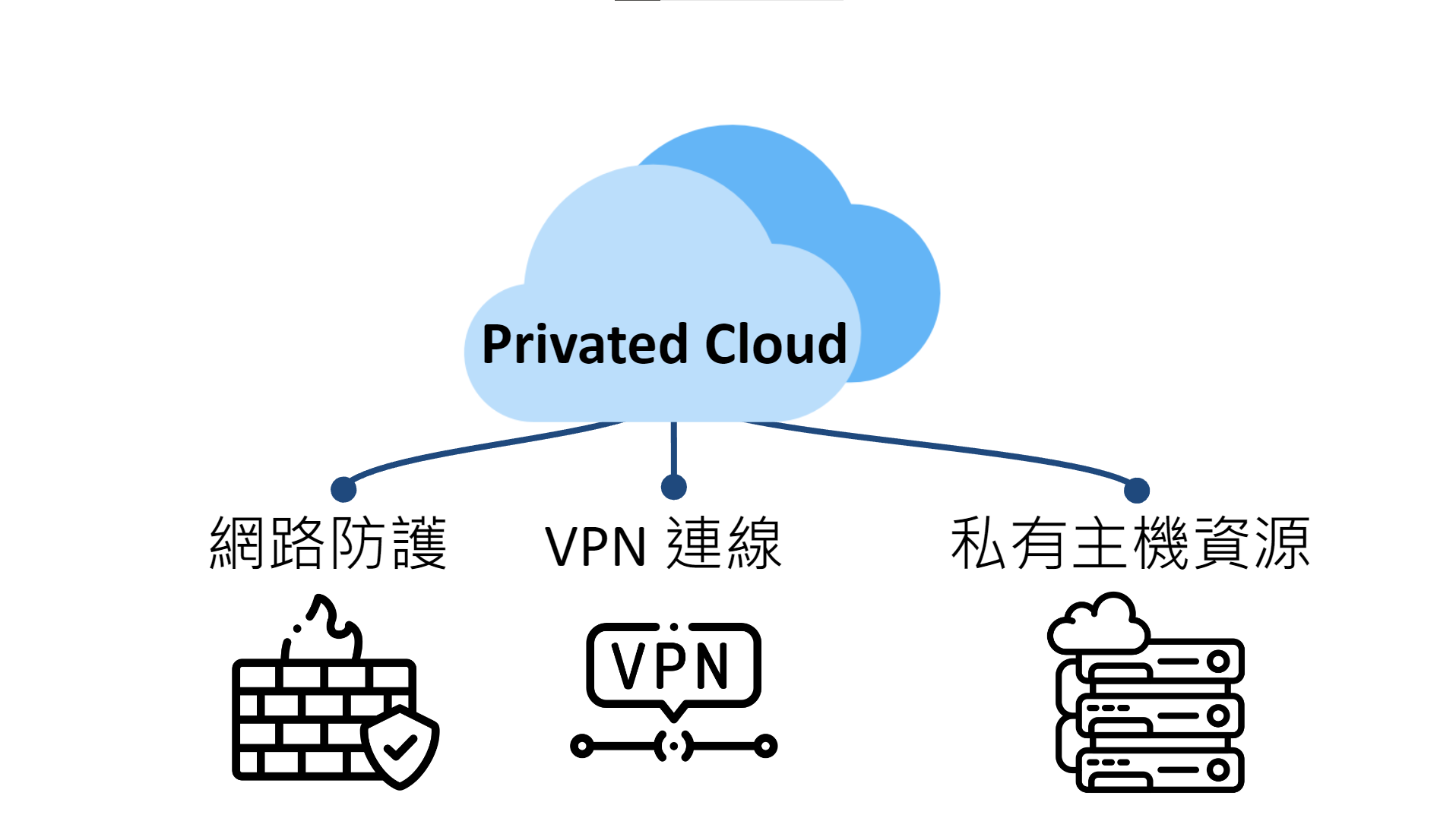
　　智能合約開發是透過 Remix - Ethereum IDE ，使用 Solidity 語言編寫，再部屬到 Go-ethereum 。

後端框架使用 NestJS ，主要以 RESTful 風格的 API 來供平台獲取必要的資料，同時也負責與 IPFS 連線，將文章上傳到 IPFS ，並搭配資料庫 MariaDB。

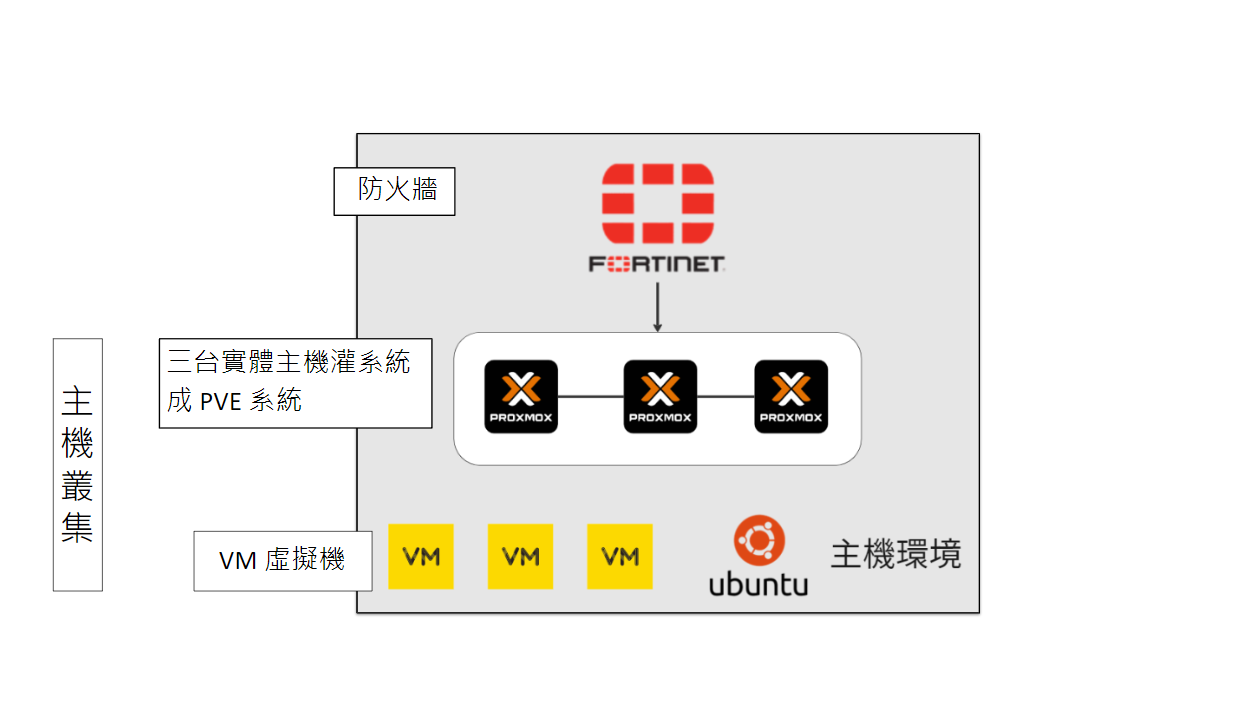


▲ 【圖二】技術應用架構圖

為了將我們的平台有一個穩定的開發與正式運行環境，便產生了私有雲架構(圖五和圖六)我們這組在實驗室將三台主機資源透過 Proxmox VE (圖七)這套開源的伺服器虛擬化環境軟體，做為所有虛擬主機架構的基底，並透過 Fortinet 防火牆(圖八)做網路層級的管理與防護。

****

▲ 【圖三】私有雲架構

****

▲ 【圖四】私有雲主機管理架構



▲ 【圖五】主機管理

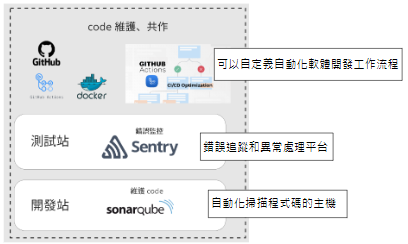


▲ 【圖六】網路管理 Fortigate 50E 畫面

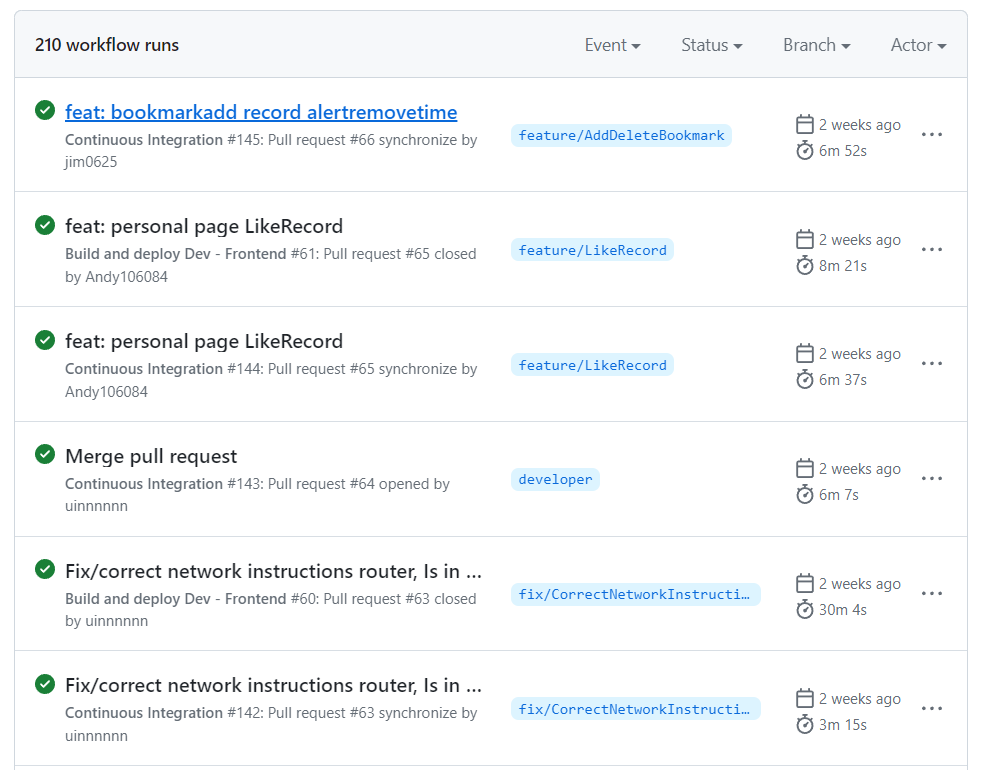
## (三)進一步規劃和測試

在不斷增加系統的各種功能時，我們將進一步規劃和實施以下措施，以確保我們的平台能夠達到預期的性能和品質。

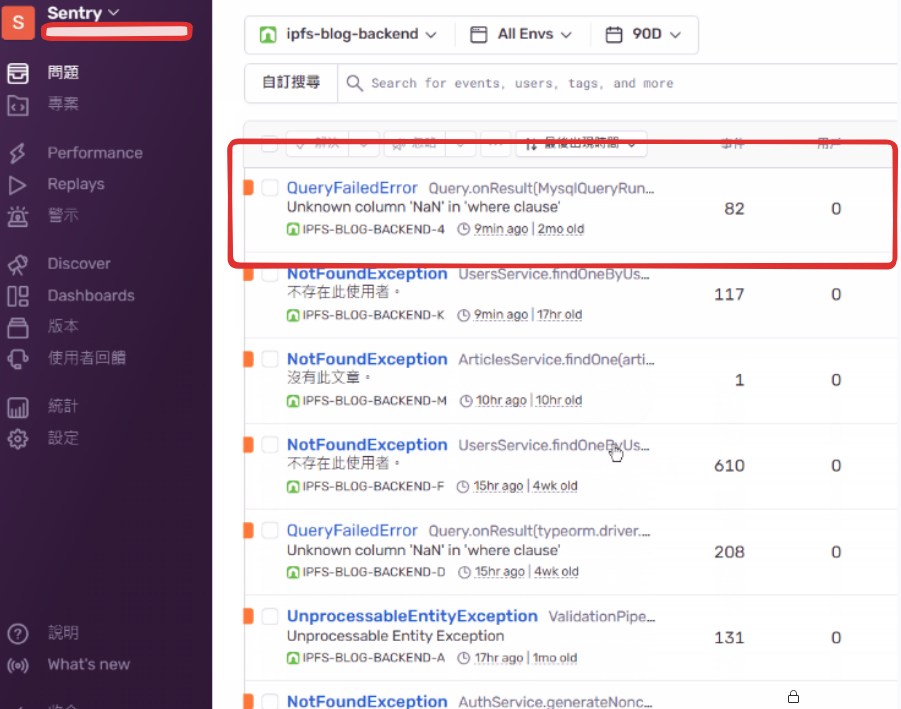
我們透過CI/CD自動化測試與部署程式碼，配置Sentry以及Sonarqube監測前、後端的代碼品質，檢查任何潛在的問題或錯誤，並於運行發生錯誤時幫助我們記錄下問題原因以及發生過程，這樣我們就能更快地找出錯誤並進行修復，有助於提高專題程式碼的穩定性和可靠性。



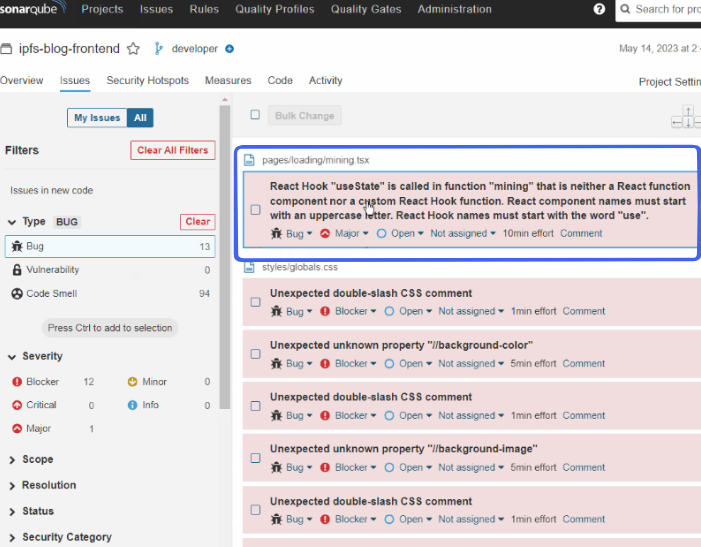
▲ 【圖七】自動化測試與品質控管應用架構圖



▲ 【圖八】GitHub Action 自動化測試與部署流程紀錄



▲ 【圖九】Sentry 檢測結果畫面



▲ 【圖十】Sonarqube檢測結果畫面

我們發現區塊鏈的交易效能有改善的空間，於是我們透過架設 eth-netstat 搭配 eth-net-intelligence-api 的網頁服務，讓我們可以實時觀察交易的處理速度、吞吐量以及相關性能指標。

從圖十二中可得知:

**平均區塊時間 ( AVG BLOCK TIME ):12.09s**

顯示生成每區塊的平均時間。該指標反映了區塊生成的速度和整體效能。

**平均網路算力 ( AVG NETWORK HASHRATE ):342.9KH/s**

表示整個網路的平均算力,即所有節點的總算力。這個數值以千哈希 (KH) 每秒為單位，反映了網路的計算資源強度。

**難度 ( DIFFICULTY ):5.02MH**

代表生成新區塊所需的工作量證明難度。這個數值反映了生成新區塊所需要的計算難度，通常使用哈希率 ( MH ) 來衡量。難度的變化可反映出網路的安全性和運作難度。這些欄位提供了有關私有鏈網路效能和狀態的重要指標。平均區塊時間展示了區塊生成的速度，平均網路算力反映了網路的計算能力，而難度則表明區塊的生成難度和網路的安全性。

一張含有 螢幕擷取畫面, 文字, 多媒體軟體, 軟體 的圖片

自動產生的描述

▲ 【圖十一】區塊鏈效能檢測畫面

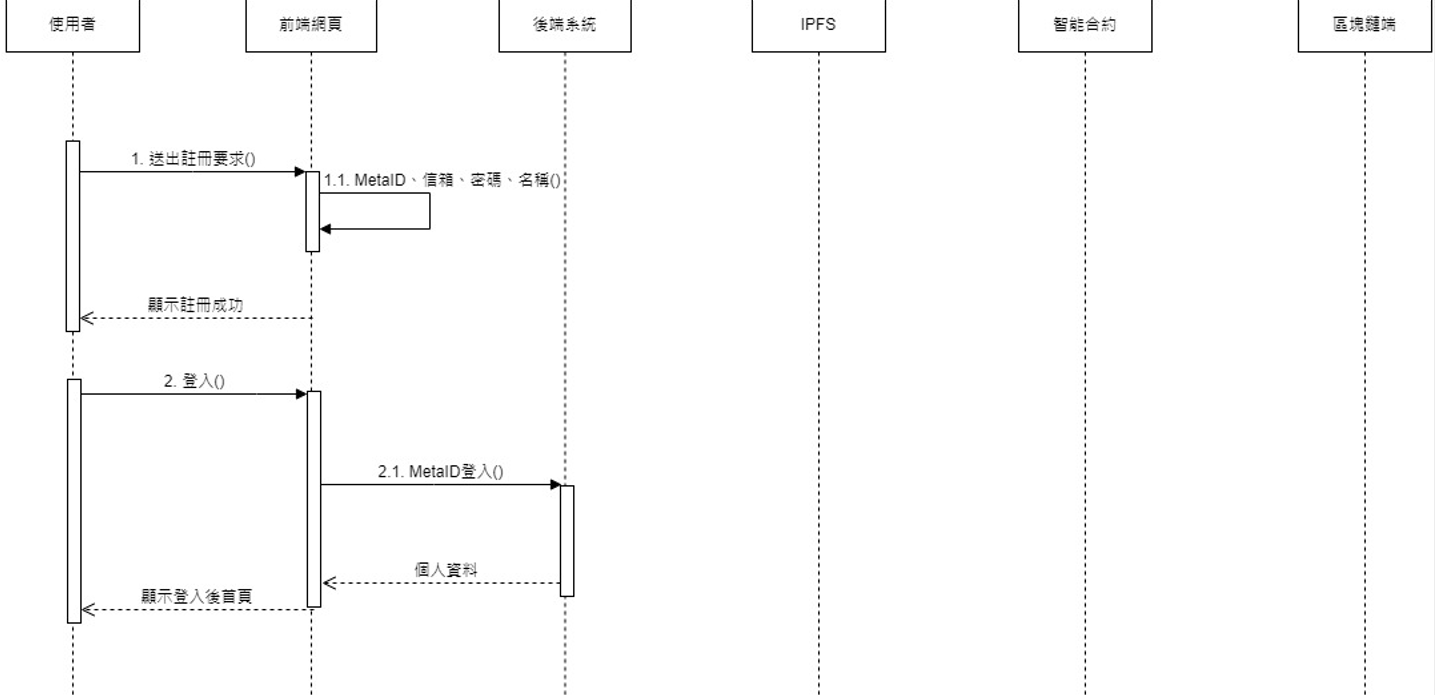
## (四)最後修正與美觀

以上工作都完成後，網站的雛型已大致完成，剩下工作只需修正網站各網頁的功能增強、版面配置和外觀美化，以達到我們認為的最佳情形。

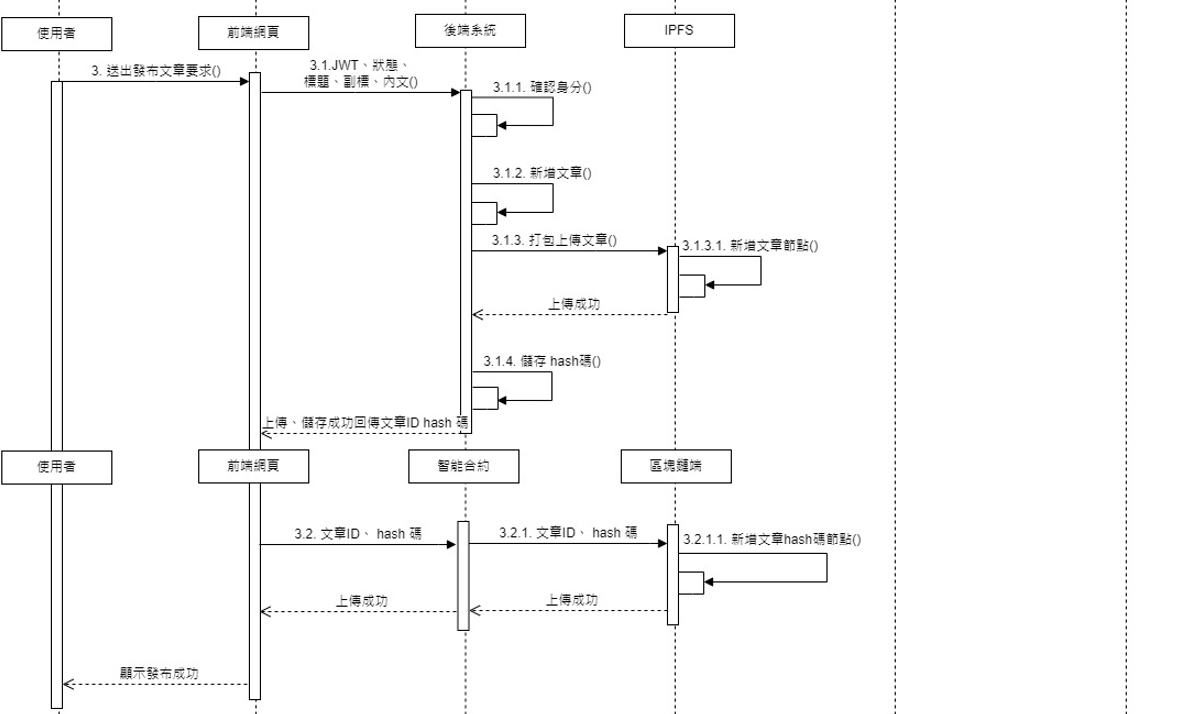
1. 使用者反饋和測試：我們將邀請使用者參與測試，收集他們的反饋和意見。
2. 響應式設計：為確保平台在不同設備和屏幕尺寸上均能正常運行，確保使用者可以在桌面、平板和手機上無縫訪問平台。
3. 性能優化：我們將進一步優化平台的性能，包括加快頁面加載速度、減少資源使用和提高伺服器反應時間，以確保用戶體驗的流暢性。
4. 區塊鏈效能優化:我們未來會使用多節點的方式運作區塊鏈，使得交易的計算壓力減輕以提升效能。

我們將確保我們的去中心化文章平台不僅擁有功能豐富的特性，還擁有吸引人的外觀和卓越的性能，以滿足未來用戶的期望並保持競爭力。

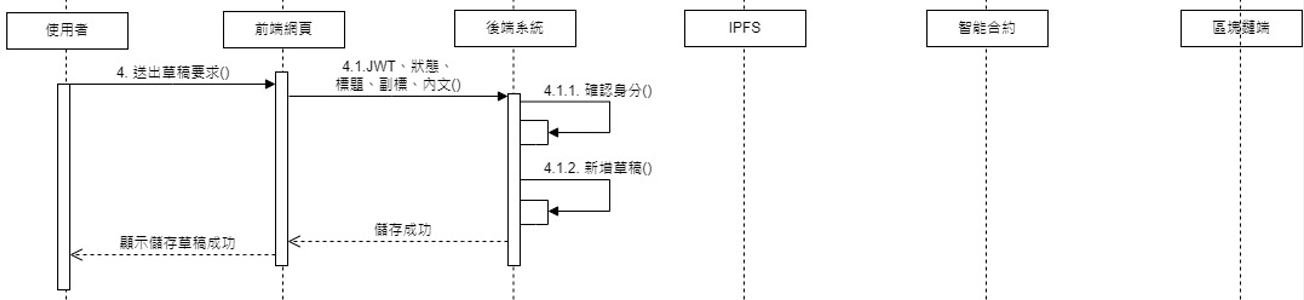
# 六、系統架構與設計



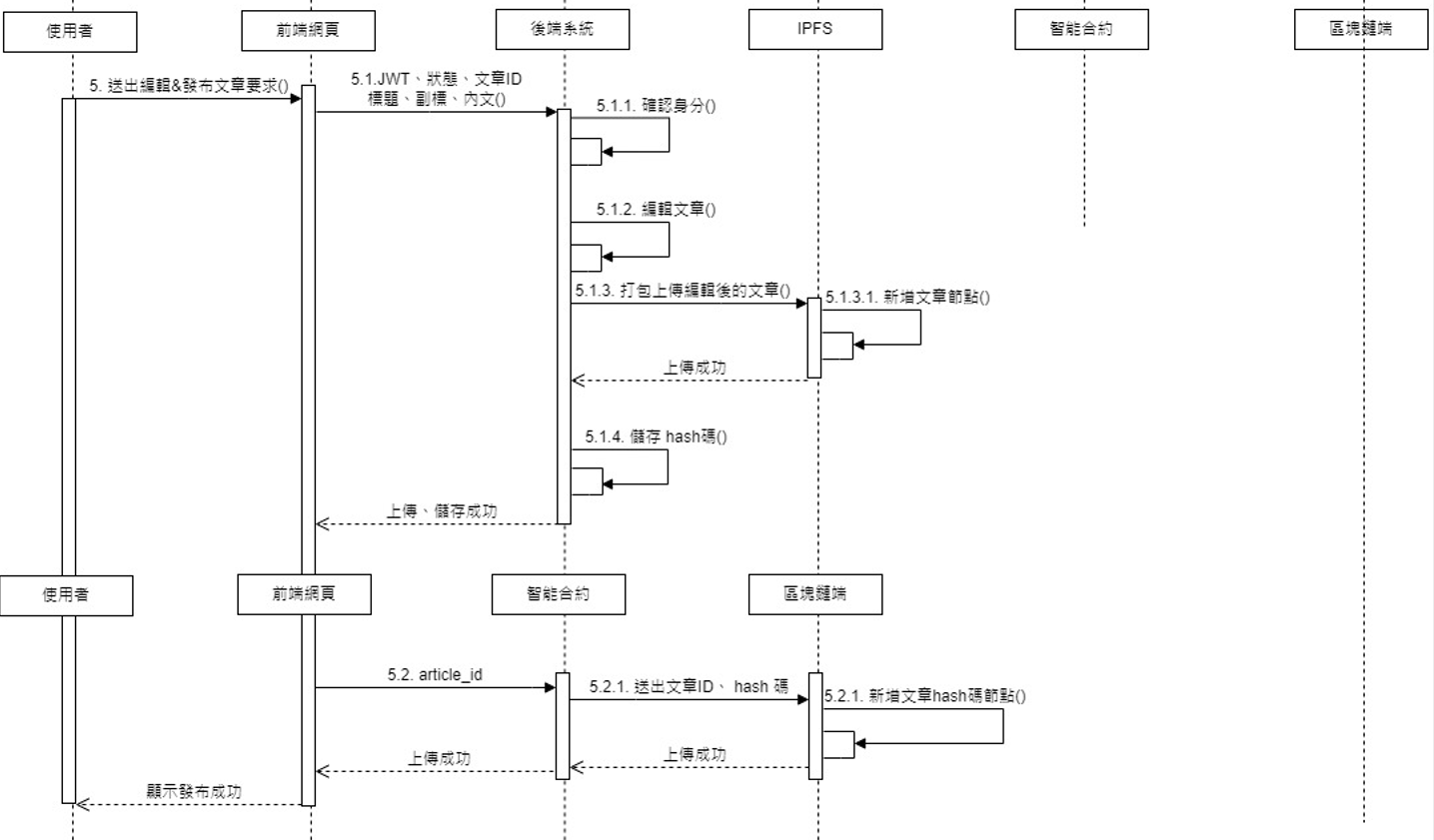
▲【圖十二】註冊與登入



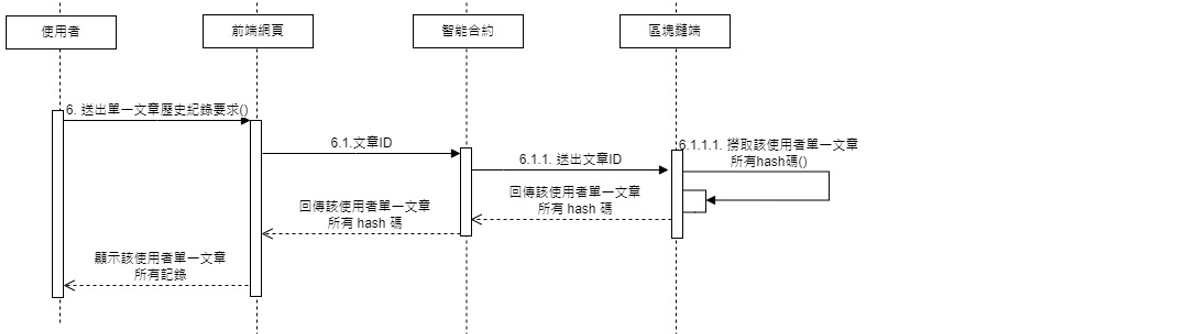
▲【圖十三】發布文章



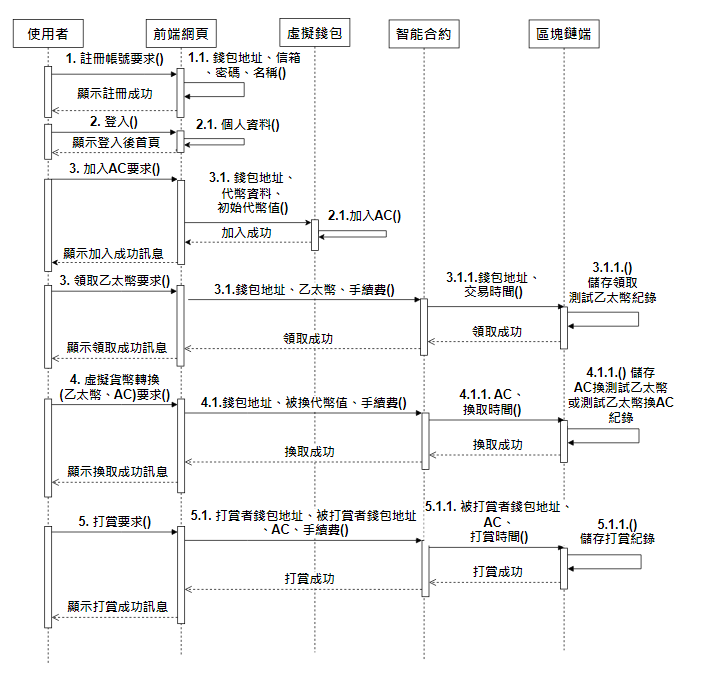
▲【圖十四】儲存草稿



▲【圖十五】編輯文章



▲【圖十六】文章歷史紀錄



▲【圖十七】加密代幣交易紀錄

# 七、安裝與實作流程

注意事項:

1.以下指令大多為 root 權限，如果 user 權限無法通過，煩請先輸入 sudo su 或在 指令之前加上 sudo ，更改為 root 權限

2.設定檔案有用紅色描述的地方可以自行調整，以下皆附上本專題的範例提供參考

## (一)前置作業

本次專題實作網頁是使用 TypeScript 語言來進行開發，採用免費且穩定之 Linux 作業系統作為平台，我們使用的 Linux 發行版本是 Ubuntu 20.04，並安裝 Docker 來進行快速建置與部署。

**安裝 Docker、Docker Compose**

先進行更新

sudo apt-get install apt-transport-https ca-certificates curl software-properties-common -y

添加 Docker 的官方 GPG 密鑰：

**拿金鑰**

wget https://download.docker.com/linux/ubuntu/gpg

**匯入**

sudo apt-key add gpg

將外部的 repository 來源匯入

sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb\_release -cs) stable"

安裝 Docker

sudo apt-get install docker-ce -y

安裝 Docker Compose

**下載Docker Compose 二進制檔**

sudo curl -L "https://github.com/docker/compose/releases/download/1.29.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

**新增可執行權限**

sudo chmod +x /usr/local/bin/docker-compose

## (二)部署監控與品質控管服務

**部屬 SonarQube 的主機**

**以下主機IP以 192.168.1.103 來做說明**

調整系統參數

sysctl -w vm.max\_map\_count=524288

sysctl -w fs.file-max=131072

ulimit -n 131072

ulimit -u 8192

**在目錄 Deploy 創建 docker-compose.yml，用於 Docker 創建前端、後端、 IPFS設定檔**

vi /SonarQube/docker-compose.yml

docker-compose.yml 設定檔內容如下

version: "3"

services:

sonarqube:

image: sonarqube:9.9.0-community

depends\_on:

- db

environment:

SONAR\_JDBC\_URL: jdbc:postgresql://db:5432/sonar

SONAR\_JDBC\_USERNAME: sonar

SONAR\_JDBC\_PASSWORD: sonar

volumes:

- sonarqube\_data:/opt/sonarqube/data

- sonarqube\_extensions:/opt/sonarqube/extensions

- sonarqube\_logs:/opt/sonarqube/logs

- sonarqube\_conf:/opt/sonarqube/conf

ports:

- "9000:9000"

db:

image: postgres:12

environment:

POSTGRES\_USER: sonar

POSTGRES\_PASSWORD: sonar

POSTGRES\_DB: sonar

volumes:

- postgresql:/var/lib/postgresql

- postgresql\_data:/var/lib/postgresql/data

volumes:

sonarqube\_data:

sonarqube\_extensions:

sonarqube\_logs:

sonarqube\_conf:

postgresql:

postgresql\_data:

**部屬 SENTRY 的主機**

**以下主機IP以 192.168.1.104 來做說明**

配置 Sentry

#拉取官網程式碼

https://github.com/getsentry/self-hosted.git

#進入資料夾

cd self-hosted

#安裝，要用最高權限

sudo ./install.sh

# 等待漫長的安裝過程

sudo docker-compose up -d

# 

## (三)前端、後端、安裝初始化 IPFS

**供使用者們的主機**

**以下主機 IP 以 192.168.1.105 來做說明**

**創建 Deploy 資料夾，供放置部署內容**

mkdir Deploy

**在目錄 Deploy 創建 docker-compose.yml ，用於 Docker 創建前端、後端、 IPFS 設定檔**

vi /Deploy/docker-compose.yml

docker-compose.yml 設定檔內容如下

version: '3'

networks:

app:

driver: bridge

web:

driver: bridge

services:

nginx:

image: nginx:latest

container\_name: web3-nginx

environment:

- HOST=${HOST}

ports:

- 80:80

depends\_on:

- frontend

- backend

networks:

- web

restart: always

tty: true

volumes:

- ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro

- ./nginx/logs:/var/log/nginx

- /etc/timezone:/etc/timezone:ro

- /etc/localtime:/etc/localtime:ro

ipfs:

image: ipfs/go-ipfs:latest

container\_name: ipfs\_host

restart: always

volumes:

- ./ipfs/export:/export

- ./ipfs/ipfs\_data:/data/ipfs

- /etc/timezone:/etc/timezone:ro

- /etc/localtime:/etc/localtime:ro

ports:

- '4001:4001'

- '4001:4001/udp'

- '8080:8080'

- '5001:5001'

db:

image: mariadb:10.6

container\_name: web3-mariadb

networks:

- app

env\_file:

- ./Backend/.env

restart: always

volumes:

- ./Backend/mysql/data:/var/lib/mysql

- /etc/timezone:/etc/timezone:ro

- /etc/localtime:/etc/localtime:ro

backend:

container\_name: web3-backend

build:

dockerfile: Dockerfile

context: ./Backend

target: production

command: /bin/sh -c "sh /app/migration-entrypoint.sh"

restart: always

networks:

- app

- web

env\_file:

- ./Backend/.env

environment:

DB\_HOST: db

depends\_on:

- db

- ipfs

volumes:

- ./Backend/outputs:/app/outputs

- ./Backend/logs:/app/logs

- /etc/timezone:/etc/timezone:ro

- /etc/localtime:/etc/localtime:ro

frontend:

container\_name: web3-frontend

build:

dockerfile: Dockerfile

context: ./Frontend

target: production

env\_file:

- ./Frontend/.env

networks:

- web

volumes:

- /etc/timezone:/etc/timezone:ro

- /etc/localtime:/etc/localtime:ro

**拉取後端原程式碼**

git clone https://github.com/IPFS-Blog/Backend.git

在後端 Backend 目錄下創建 .env，用來配置設定檔

vi /Backend/.env

.env設定檔如下

NODE\_ENV=local **(開發模式)**

DB\_HOST=127.0.0.1 **(DB 資料庫位置)**

DB\_PORT=3306 **(DB 資料庫連接埠)**

DB\_USERNAME=web3 **(DB 資料庫使用者名)**

DB\_PASSWORD=secret **(DB 資料庫使用者密碼)**

DB\_DATABASE=web3 **(DB 資料庫名)**

DB\_TIMEZONE="+08:00" **(DB 資料庫時區)**

MAIL\_HOST="smtp.gmail.com" **(MAIL 寄信主機)**

MAIL\_PORT=465 **(MAIL 寄信主機連接埠)**

MAIL\_USERNAME="XXXXXXX@gmail.com" **(MAIL 帳號)**

MAIL\_PASSWORD="XXXXXXXXX" **(MAIL 應用程式密碼)**

MAIL\_TLS\_ENABLED="false" **(MAIL TLS關閉)**

APP\_SECRET=undefine **(JWT Token 加密金鑰)**

# The main page of this system

APP\_HOST=http://192.168.1.98 **(主機位置，信件使用)**

APP\_AUTHOR\_GROUP="XXXXXXX@googlegroups.com" **(通知作者們群組)**

CORS\_WHITE="http://192.168.1.98:3000" **(開放白名單)**

CORS\_METHOD="GET,PUT,POST,DELETE,UPDATE" **(開放使用規則)**

APP\_SWAGGER\_Title="NestJS-Board API" **(SWAGGER 標題)**

APP\_SWAGGER\_Description="This is NestJS-Board API documentation." **(SWAGGER 描述)**

APP\_SWAGGER\_Version="0.0.1" **(SWAGGER 版本)**

SENTRY\_DSN="sentry\_key" **(唯一認證碼)**

IPFS\_HOST="http://192.168.1.98:5001/api/v0" **(IPFS 溝通 API)**

ROOT\_USERNAME=root **(DB 最高權限帳號)**

ROOT\_PASSWORD=example **(DB 最高權限密碼)**

MYSQL\_ROOT\_PASSWORD=example **(DB 最高權限密碼)**

MARIADB\_USER=web3 **(DB 初始化使用者帳號)**

MARIADB\_PASSWORD=secret **(DB 初始化使用者密碼)**

MARIADB\_DATABASE=web3 **(DB 初始化資料庫)**

**拉取前端原程式碼**

git clone https://github.com/IPFS-Blog/Frontend.git

在前端Frontend目錄下創建 .env，用來配置設定檔

vi /Frontend/.env

.env 設定檔如下

NEXT\_PUBLIC\_API = http://192.168.xxx.xxx:xxxx/api/v1 **(後端API主機IP)**

**(合約部分看(四)智能合約上傳至區塊鏈 操作教學)**

NEXT\_PUBLIC\_FaucetContractAddress = 0xD2eaCdb128cCA8a6916579aaf19F348CBCDF40b8 **(更改Faucet合約部屬所產合約地址)**

NEXT\_PUBLIC\_MyTokenContractAddress = 0x5baA2E5106c9F5e672844366f77Bd73e4bBB7287 **(更改MyToken合約部屬所產合約地址)**

NEXT\_PUBLIC\_ArticlesHistoryAddress = 0x5d52c23CC6Caf2773Cd86C1E09dEA7D45f3294b6 **(更改ArticlesHistory合約部屬所產合約地址)**

NEXT\_PUBLIC\_UserKey = 7xxxxxxxxxx3 **(可以看MetaMask教學部分，設定區塊鏈合約呼叫者，透過此帳戶呼叫合約可以不用省去使用者的手續費，像是Faucet領取以太幣合約)**

NEXT\_PUBLIC\_CHAINID = 0x7b **(區塊鏈ID)**

NEXT\_NODE\_ENV = production **(開發模式)**

**(監控服務設定檔)**

SENTRY\_URL = ""**(主機位置)**

SENTRY\_DSN = ""**(唯一認證碼)**

SENTRY\_ORG = "sentry"**(Sentry組織)**

SENTRY\_PROJECT = ""**(專案名)**

SENTRY\_AUTH\_TOKEN = ""**(通行權限碼)**

**(第三方 CLOUDINARY 圖片儲存服務)**

NEXT\_PUBLIC\_CLOUDINARY\_CLOUD\_NAME = xxxxxxxx **(雲端給定名稱)**

NEXT\_PUBLIC\_CLOUDINARY\_API\_KEY = xxxxxxxxxxxxx **(雲端API-KEY)**

NEXT\_PUBLIC\_CLOUDINARY\_API\_SECRET = xxxxxxxxxxxxxxxxxxxx **(雲端API密碼)**

**配置 Nginx**

**配置環境變數**

在 Deploy 目錄下創建 .env，用來配置設定檔

vi /Frontend/.env

.env 設定檔如下

HOST=192.168.1.98 **(主機對外名)**

**設定檔檔案位置**

**Deploy/nginx/nginx.conf**

worker\_processes auto;

events {

worker\_connections 1024;

}

http {

server {

listen 80;

server\_name ${HOST};

charset utf-8;

access\_log /var/log/nginx/access.log;

error\_log /var/log/nginx/error.log;

location / {

proxy\_pass http://frontend:3000; # 將請求代理到前端服務的指定端口

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $http\_x\_forwarded\_proto;

}

location /api/v1 {

proxy\_pass http://backend:3000/api/v1; # 將請求代理到後端服務的指定端口

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $http\_x\_forwarded\_proto;

}

location /api-docs {

proxy\_pass http://backend:3000/api-docs; # 將請求代理到後端服務的指定端口

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $http\_x\_forwarded\_proto;

}

location /outputs {

proxy\_pass http://backend:3000/outputs; # 將請求代理到後端服務的指定端口

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $http\_x\_forwarded\_proto;

}

}

}

**先進行初始化配置 IPFS**

**(進入資料夾，預先啟動 IPFS 節點)**

cd Deploy

sudo docker-compose up -d ipfs

**(設定跨域請求許可 - 來源)**

sudo docker exec ipfs\_host ipfs config --json API.HTTPHeaders.Access-Control-Allow-Origin '["http://192.168.1.105:5001", "http://localhost:3000", "http://127.0.0.1:5001", "https://webui.ipfs.io"]'

**(設定跨域請求許可 - 請求方式)**

sudo docker exec ipfs\_host ipfs config --json API.HTTPHeaders.Access-Control-Allow-Methods '["PUT", "POST"]'

**(重新啟動 IPFS 節點)**

sudo docker-compose down

sudo docker-compose up -d ipfs

進行初始化配置 Backend logs 位置

**(進入資料夾，預先啟動 Backend)**

cd Deploy

mkdir logs & chmod 777 logs

**啟動容器**

sudo docker-compose up -d

## (四) Go-ethereum & BlockScout 架設

開始創建一個blockchain資料夾，並在blockchain裡面創建一個data資料夾，用於存放區塊鏈設定檔案

sudo mkdir -p blockchain/data

**在 blockchain 目錄下創建 docker-compose.yml ，用於 Docker 建置區塊鏈的設定檔**

vi /blockchain/docker-compose.yml

docker-compose.yml 設定檔內容如下

version: "3.9"

services:

ethereum:

image: ethereum/client-go:stable

container\_name: ethereum

restart: always

volumes: [ "./data:/data" ]

entrypoint: /bin/sh

command: data/scripts/docker-entrypoint.sh

ports:

- "8545:8545"

- "30303:30303"

**在 data 目錄下創建 genesis.json ，設定區塊鏈的創世區塊**

vi /blockchain/data/genesis.json

genesis.json 設定檔內容如下

{

"config": {

"chainId": 123,**(此處可以自行設定區塊鏈的鏈ID)**

"homesteadBlock": 0,

"eip150Block": 0,

"eip150Hash": 0,

"eip155Block": 0,

"eip158Block": 0,

"byzantiumBlock": 0,

"constantinopleBlock": 0

},

"nonce": "0x0000000000000042",

"difficulty": "0x020",

"mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",

"coinbase": "0x0000000000000000000000000000000000000000",

"timestamp": "0x00",

"parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",

"extraData": "0x0000000000000000000000000000000000000000000000000000000000000000",

"gasLimit": "0xffffff",

"alloc": { **(此處alloc可以設定預設帳號所持有的金額，可以自行填寫)**

"0x34B605B3d13923a60a629794C15B103C44beaE1c": {

"balance": "99999999999999999999999999999"

},

"0xd8589dbd988076d18a382fb5213151477c83cbab" :{

"balance": "99999999999999999999999999999"

}

}

}

**在 data/scripts 目錄下創建 docker-entrypoint.sh ，用於 Docker 啟動執行腳本，從創世區塊設定檔案初始化區塊鏈並啟動區塊鏈網路**

vi /blockchain/data/scripts/docker-entrypoint.sh

docker-entrypoint.sh 設定檔內容如下

#!/bin/sh

geth init --datadir /data/node1 data/genesis.json

api=debug,web3,eth,txpool,ethash,net

geth --datadir /data/node1 \

--networkid 123 \ **(此處按照genesis.json設定的chainId做調整)**

--nodiscover \

--http \

--http.addr "0.0.0.0" \

--http.port 8545 \

--maxpeers 5 \

--rpc.enabledeprecatedpersonal \

--allow-insecure-unlock \

--http.corsdomain "\*" \

--http.api web3,eth,debug,personal,net,miner,admin,txpool \

--vmdebug \

--ws \

--ws.addr "0.0.0.0" \

--ws.port 8888 \

--ws.origins "\*" \

--ws.api ${api} >> /data/ethereum.log 2>&1 \

--mine \ **(此處按照預設帳號做調整，用於設定預設挖礦帳戶)**

--miner.etherbase "0xd78cbd42d7f56897d5c562ddd6b1b1837d72b54e" \

**在 blockchain 創建一個 docker-compose-no-build-geth.yml ，用於 Docker 啟動 BlockScout 的設定檔，檔案內容從官網提供的 Github Clone 做使用 (**[**https://github.com/blockscout/blockscout/blob/master/docker-compose/docker-compose-no-build-geth.yml**](https://github.com/blockscout/blockscout/blob/master/docker-compose/docker-compose-no-build-geth.yml)**)**

先從 github 複製內容過來並修改( ETHEREUM\_JSONRPC\_HTTP\_URL 需要修改)

vi blockchain/docker-compose-no-build-geth.yml

docker-compose-no-build-geth.yml 設定檔內容如下

version: '3.8'

services:

redis\_db:

extends:

file: ./services/docker-compose-redis.yml

service: redis\_db

**…中間內容省略**

environment:

ETHEREUM\_JSONRPC\_VARIANT: 'geth'

ETHEREUM\_JSONRPC\_HTTP\_URL: http://192.168.1.1:8454/**(此處修改為自己主機的IP)**

**…其餘內容省略**

**最後在 blockchain 目錄下創建 restarted.sh ，用於重啟區塊鏈以及 BlockScout 的方便腳本**

vi blockchain/restarted.sh

restarted.sh 設定檔內容如下

#!/bin/sh

docker stop $(docker ps -aq)

docker rm $(docker ps -aq)

rm -f chainblock/data/ethereum.log

rm -rf chainblock/data/node1/geth/

rm -rf chainblock/data/node1/geth.ipc/

docker-compose -f chainblock/docker-compose.yml up -d

rm -rf blockscout/

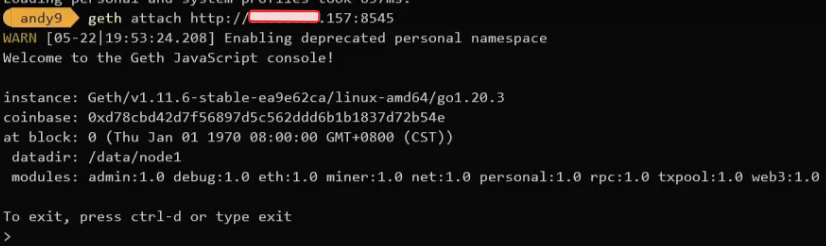
git clone https://github.com/blockscout/blockscout.git

cp docker-compose-no-build-geth.yml blockscout/docker-compose/

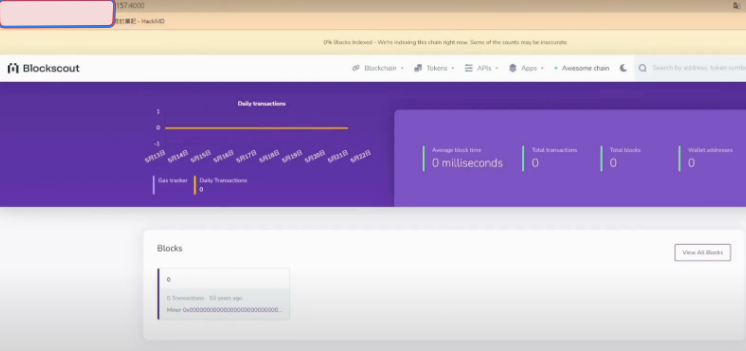
docker-compose -f blockscout/docker-compose/docker-compose-no-build-geth.yml up -d

**執行方式:**

sudo ./restarted.sh

****

▲【圖十八】連線到 Go-ethereum 畫面



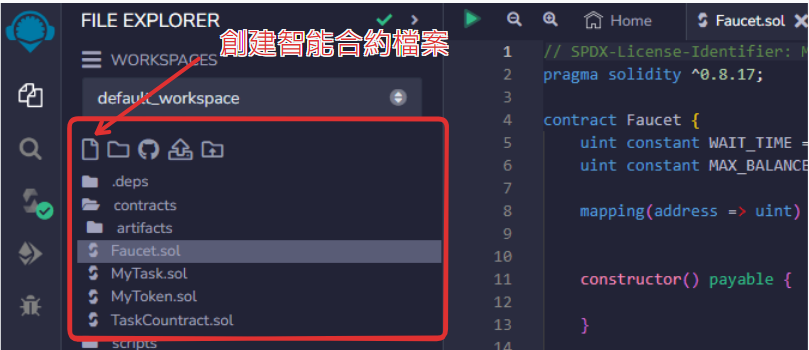
▲【圖十九】 BlockScout 區塊鏈瀏覽器網頁畫面

## (五)智能合約上傳至區塊鏈

使用 Remix 線上 IDE 撰寫、上傳智能合約(<https://remix.ethereum.org/>)

**上傳合約簡易教學:**

1. 創建智能合約檔案

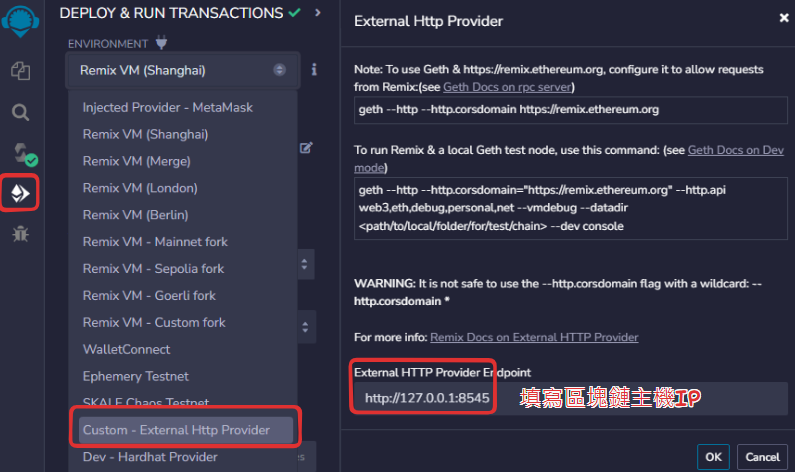


▲【圖二十】創建智能合約檔案

2. 編譯智能合約

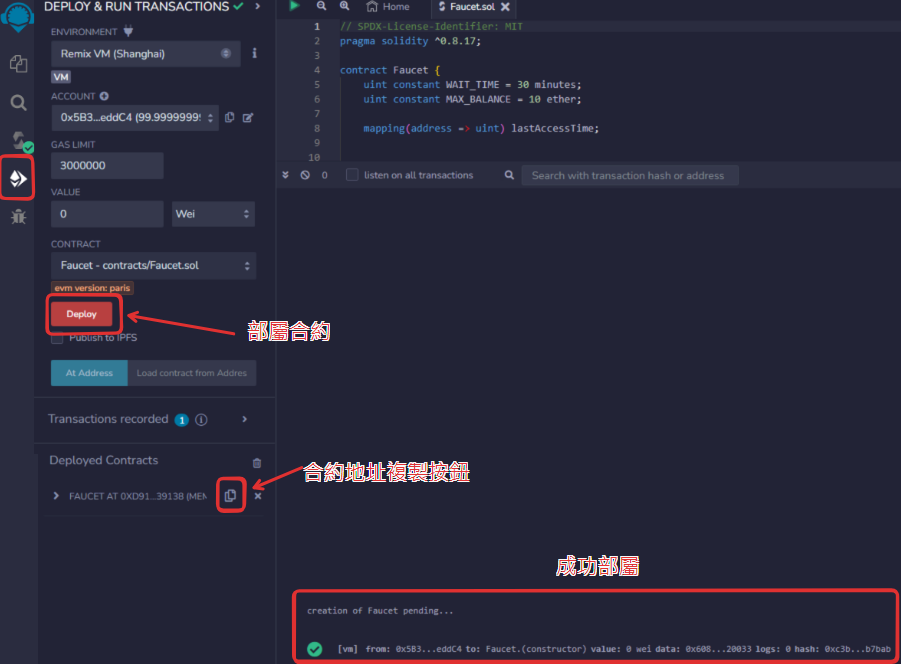


▲【圖二十一】編譯智能合約

3. 連線到區塊鏈

▲【圖二十二】連線到區塊鏈

4. 部屬合約(**複製合約地址 前端需要使用到**)



▲【圖二十三】部屬合約

**三份智能合約:**

1. Faucet.sol 區塊鏈水龍頭(提供使用者領取測試以太幣)

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

contract Faucet {

uint constant WAIT\_TIME = 30 minutes;

uint constant MAX\_BALANCE = 10 ether;

mapping(address => uint) lastAccessTime;

constructor() payable {

}

function requestTokens(address payable user) external {

require(user.balance <= MAX\_BALANCE, "Faucet balance is too high.");

require(lastAccessTime[user] + WAIT\_TIME <= block.timestamp, "Please wait before requesting tokens again.");

lastAccessTime[user] = block.timestamp;

payable(user).transfer(MAX\_BALANCE);

}

receive() external payable {

// this contract only allows receiving funds from external accounts

}

}

2. MyToken.sol 自創AC加密代幣(用於激勵/打賞創作者使用)

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.17;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract MyToken is ERC20 {

uint256 public constant EXCHANGE\_RATE = 10;

constructor() payable ERC20("Appreciate citizens", "AC") {

\_mint(address(this), 10000000 \* (10 \*\* decimals()));

}

function decimals() public view virtual override returns (uint8) {

return 0;

}

receive() external payable {}

function buyToken() external payable {

require(msg.value/ 10\*\*18 > 0, "Invalid amount of ether");

uint256 tokenAmount = msg.value/ 10\*\*18 \* EXCHANGE\_RATE;

\_transfer(address(this), msg.sender, tokenAmount);

}

function sellToken(uint256 \_tokenAmount) external {

require(\_tokenAmount > 0, "Invalid amount of token");

uint256 etherAmount = \_tokenAmount / EXCHANGE\_RATE \* 1 ether;

require(payable(address(this)).balance >= etherAmount, "Insufficient balance");

\_transfer(msg.sender, payable(address(this)), \_tokenAmount);

payable(msg.sender).transfer(etherAmount);

}

}

3. ArticleHistory.sol 文章上傳歷史紀錄

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.15;

contract ArticleRegistry {

struct Article {

string hashCode;

string creationTime;

}

mapping(uint256 => Article[]) private articles;

constructor() payable {

// 構造函數，可以接收初始資金

}

function addArticle(uint256 \_id, string memory \_hashCode, string memory \_creationTime) external {

Article memory newArticle = Article(\_hashCode, \_creationTime);

articles[\_id].push(newArticle);

}

function getArticle(uint256 \_id) external view returns (Article[] memory) {

return articles[\_id];

}

receive() external payable {

// 這個合約只允許從外部帳戶接收資金

}

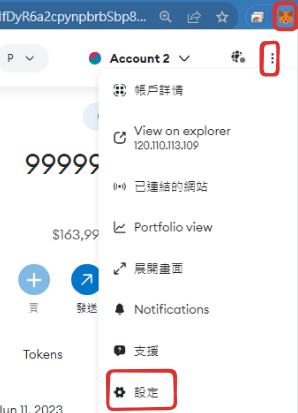
}

# 

## (六) MetaMask 錢包設定(智能合約交易需要使用)

開始先安裝 MetaMask 的 chrome 擴充套件

網址：https://reurl.cc/9RRqMX

****

▲【圖二十四】設定區塊鏈網路

# 八、實驗結果與展示

**(一) 首頁**

包含搜尋框、連接自身 MetaMask (如無申請過帳號即為註冊)，側欄包含自身後台、創建文章、水龍頭等連結。

****

▲【圖二十五】首頁

**(二) 註冊帳號**

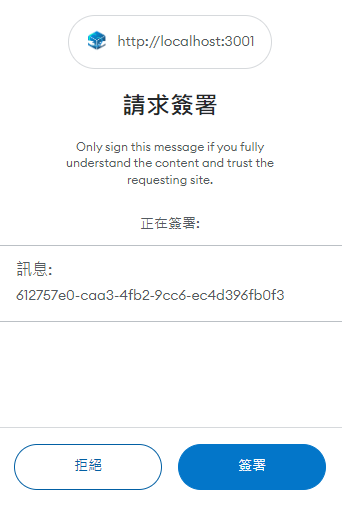
頁面 ID 為自身 MetaMask 地址，註冊時會發送驗證碼至註冊信箱，驗證碼正確即可成功註冊。

****

▲【圖二十六】註冊帳號

**(三) 登入頁面**

即為 MetaMask 授權，已註冊過帳號者，點擊首頁右上角授權後直接連結帳號。

****

▲【圖二十七】登入頁面

**(四) 個人資料頁面**

可查看自身創作文章、收藏、按讚紀錄、個人錢包等資訊。

▲【圖二十八】個人資料頁面

**(五) 修改個人資料頁面**

包含可上傳變更頭像、背景、修改個人名稱、信箱等。

****

▲【圖二十九】修改個人資料頁面

**(五) 個人錢包畫面**

包含個人 AC 代幣與 ETH 代幣，可將AC幣與私有區塊鏈 ETH 幣互相交換， AC 幣是打賞創作者的代幣。可查看代幣交換的交易紀錄。

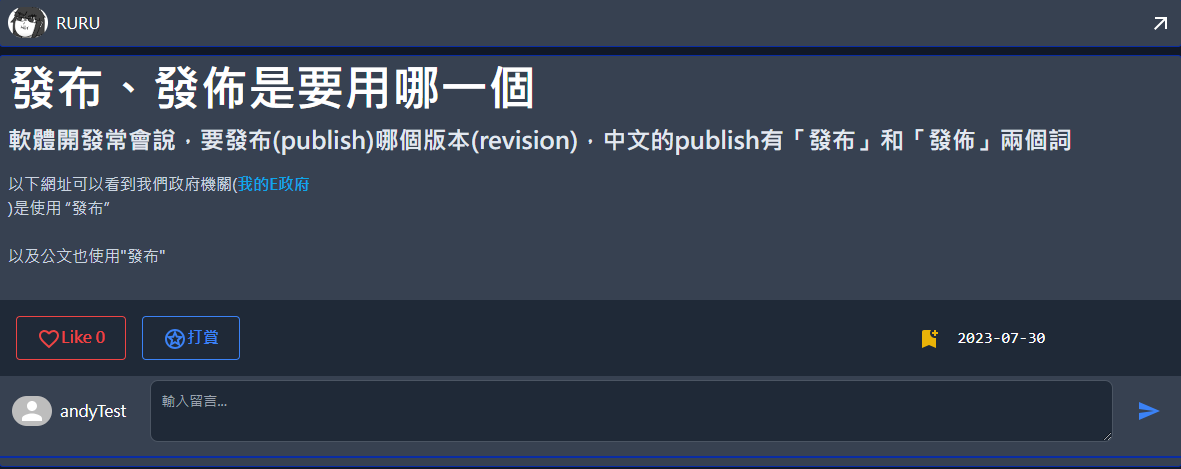
一張含有 文字, 螢幕擷取畫面, 數字, 軟體 的圖片

自動產生的描述

▲【圖三十】個人錢包畫面

**(六) 單一文章頁面**

包含作者、文章內容、留言、按讚、打賞等。



▲【圖三十一】單一文章頁面

**(七) 打賞畫面**

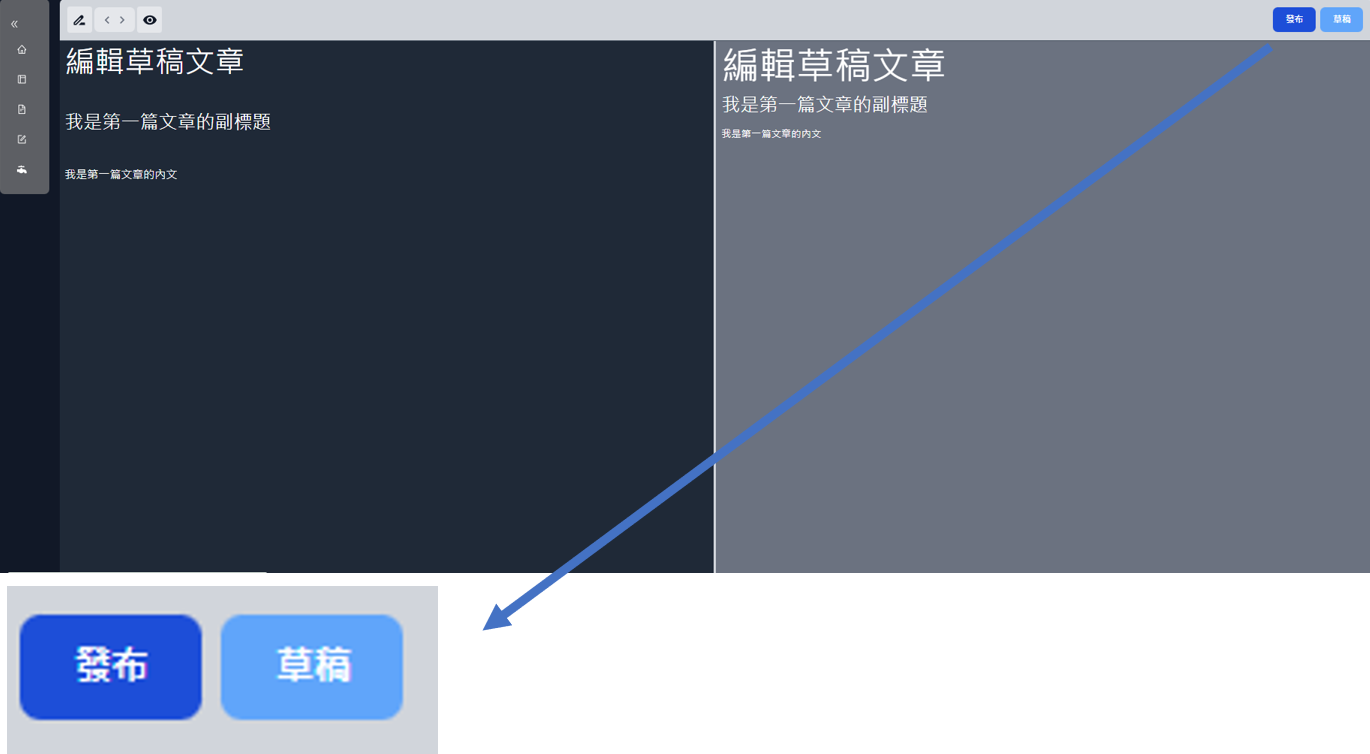
位於每篇文章底下(登入者可顯示)，透過錢包中AC代幣可打賞給該篇文章的創作者。



▲【圖三十二】打賞畫面

**(八) 發布文章或儲存草稿頁面**

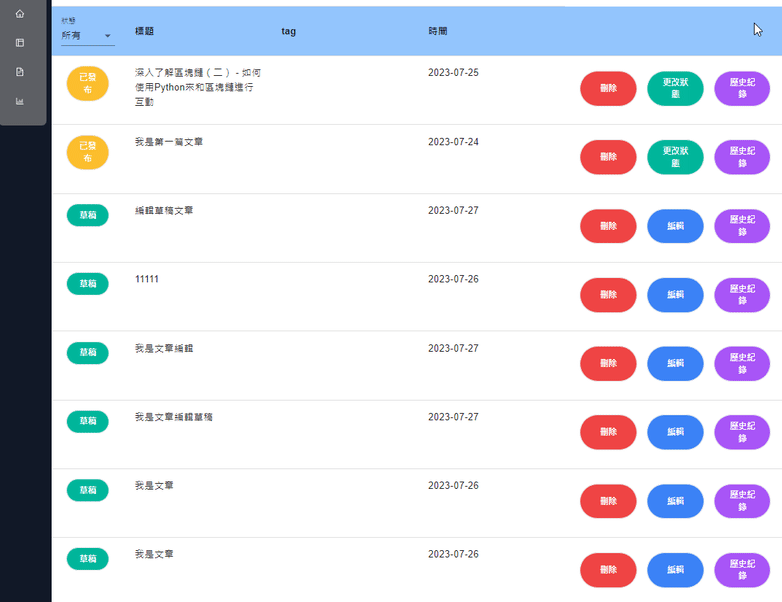
創作者可將創作好的文章進行發布或儲存成草稿，無論是何者皆可在個人後台查看該篇文章狀態。

****

▲【圖三十三】發布文章或儲存草稿頁面

**(九) 個人後台頁面**

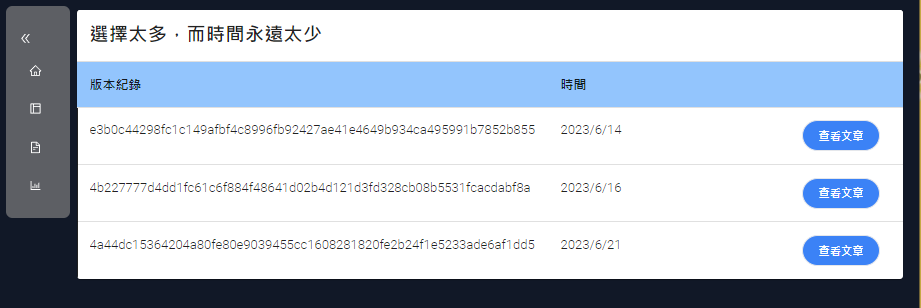
可在個人後台查看自身的創作文章狀態(10篇文章為一頁)，已發布的文章可透過更改狀態變成草稿再進行編輯。提供篩選已發布或是草稿的功能。

****

▲【圖三十四】個人後台頁面

**(十) 文章歷史紀錄頁面**

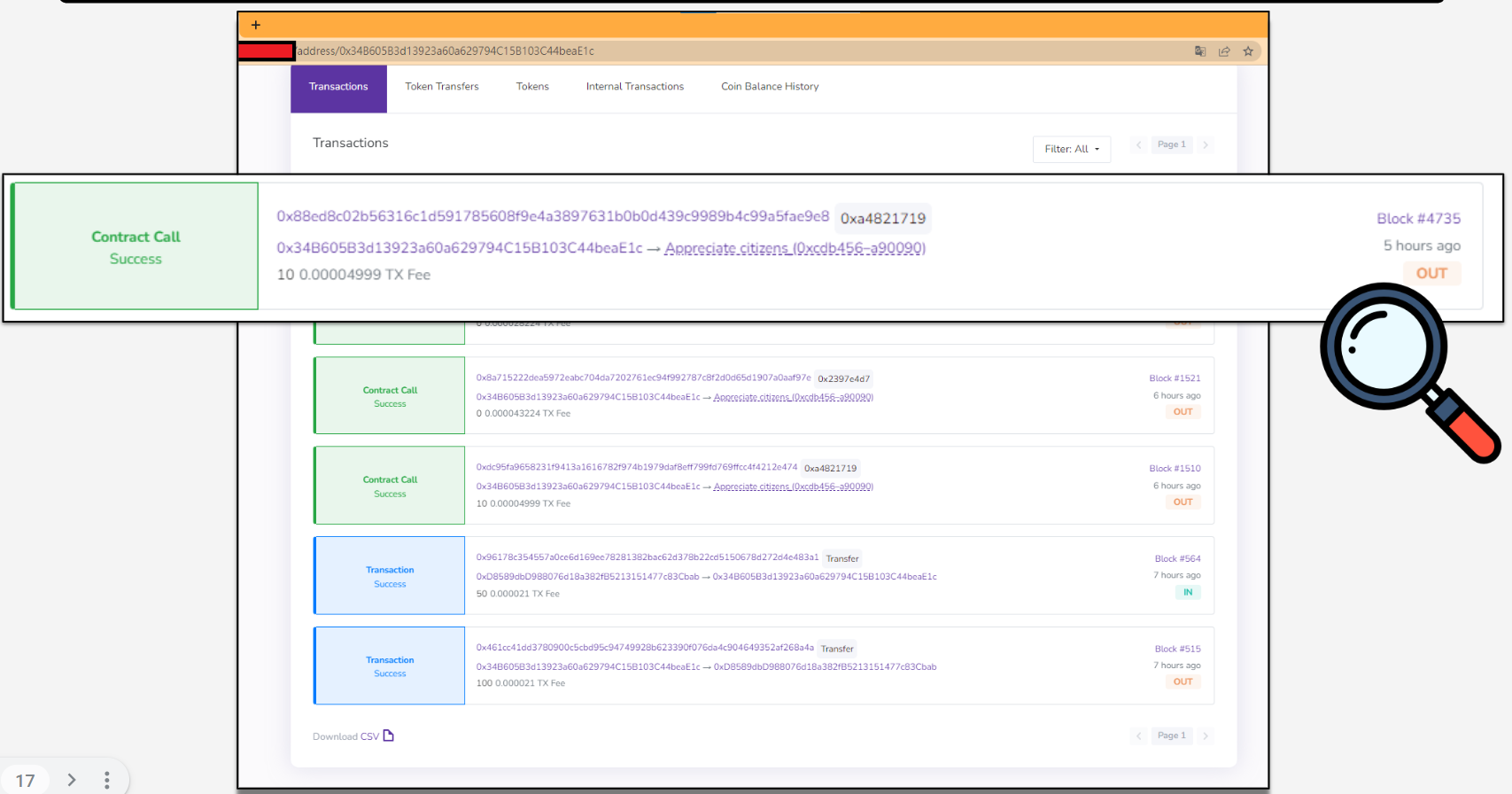
可在個人後台的文章歷史紀錄中查詢該篇文章的版本紀錄。

****

▲【圖三十五】文章歷史紀錄頁面

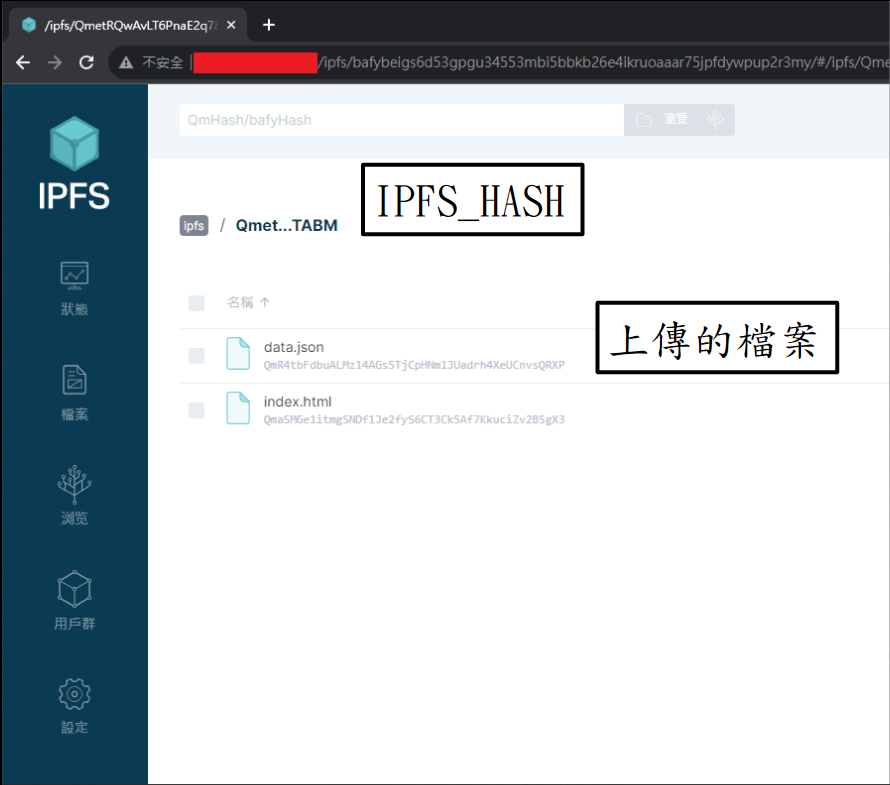
**(十一) 區塊鏈瀏覽器頁面**

在瀏覽器查詢交易，監控交易活動，瀏覽區塊鏈區塊。



▲【圖三十六】區塊鏈瀏覽器頁面

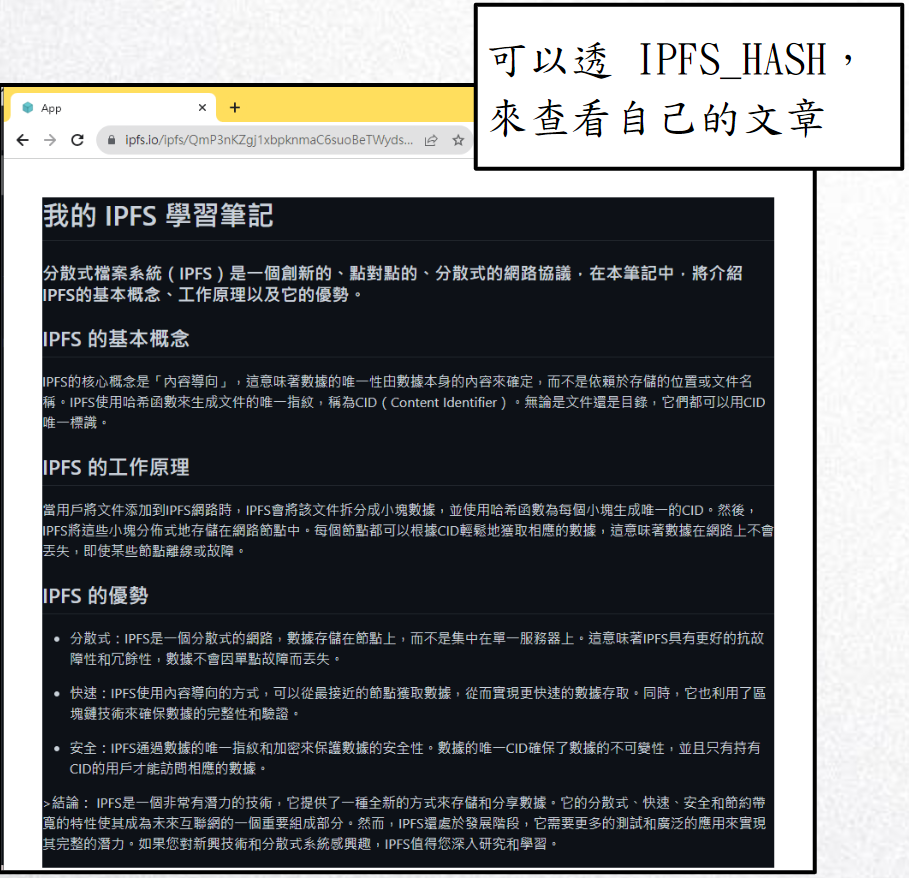
**(十二)IPFS檢索檔案頁面**

****

▲【圖三十七】IPFS 檢索檔案頁面

**(十三)透過IPFS\_HASH查看文章**

可從IPFS檢索檔案頁面中找到上傳檔案的 IPFS\_HASH ，在瀏覽器輸入 IPFS\_HASH 可查看到文章。

****

▲【圖三十八】透過IPFS\_HASH查看文章

**(十四)區塊鏈水龍頭**

提供使用者領取測試以太幣，30分鐘且錢包低於10 ETH 代幣才可以領取10 ETH。



▲【圖三十九】區塊鏈水龍頭(提供使用者領取測試以太幣)

# 

# 

# 九、結論與未來規劃

本專題成功實作了一個基於分散網絡和區塊鏈技術的文章發佈平台，該平台包括多個模組。首先，我們使用 IPFS 來確保文章的持久性和可用性。其次，智能合約在平台中扮演著關鍵角色，實現了虛擬貨幣轉換、交易和文章歷史紀錄的透明操作。最重要的是，該項專案的設計解決了傳統平台所面臨的安全和集中化問題，為使用者提供了更安全、透明和互動性的文章發佈環境。

未來，該平台可以進一步改進與改善功能，增加多媒體內容、社群互動和隱私保護功能，以滿足用戶需求，促進創作者和讀者之間的互動與交流。整體而言，我們所開發的此平台代表了未來數位內容發布的趨勢，為數位內容創作者和讀者提供更安全、透明和互動性的環境。

# 

# 十、參考資料及網站

**1. Go-ethereum 私有鏈架設 – Rogerh.eth**

[**https://medium.com/samumu-clan/用-geth-架設私有鏈-41a2baa0efd8**](https://medium.com/samumu-clan/%E7%94%A8-geth-%E6%9E%B6%E8%A8%AD%E7%A7%81%E6%9C%89%E9%8F%88-41a2baa0efd8)

**2. Go-ethereum API 文檔 - 学习软件编程** [**http://cw.hubwiz.com/card/c/geth-rpc-api/**](http://cw.hubwiz.com/card/c/geth-rpc-api/)

**3. Go-ethereum 節點配置設定 -** [**Soy**](https://blog.51cto.com/SoyTechnology)[**https://blog.51cto.com/SoyTechnology/5377127**](https://blog.51cto.com/SoyTechnology/5377127)

**4. Go-ethereum 智能合約交互 - Bob Lee**

[**https://zhuanlan.zhihu.com/p/95672004**](https://zhuanlan.zhihu.com/p/95672004)

**5.區塊鏈效能檢測服務 - Robby**

[**https://dotblogs.com.tw/explooosion/2018/07/30/200754#%E4%B8%80%E3%80%81eth-netstats**](https://dotblogs.com.tw/explooosion/2018/07/30/200754#%E4%B8%80%E3%80%81eth-netstats)

**6. Solidity 常見問題 - 瘦身小蚂蚁**

[**https://blog.csdn.net/ling1998/article/details/124335786**](https://blog.csdn.net/ling1998/article/details/124335786)

**7. MetaMask 連接到 Go-ethereum 私有鏈 - 糖糖糖糖糖糖糖**

[**https://blog.csdn.net/weixin\_44826468/article/details/121636871**](https://blog.csdn.net/weixin_44826468/article/details/121636871)

**8. web3.js - 官網**

[**https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html**](https://web3js.readthedocs.io/en/v1.2.11/web3-eth.html)

**9. ERC20 智能合約設計 - GitHub** [**https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol**](https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol)

**10. blockscout 架設 - GitHub**

[**https://github.com/blockscout/blockscout**](https://github.com/blockscout/blockscout)

**11. SonarQube - GitHub** [**https://github.com/SonarSource/docker-sonarqube/blob/master/example-compose-files/sq-with-postgres/docker-compose.yml**](https://github.com/SonarSource/docker-sonarqube/blob/master/example-compose-files/sq-with-postgres/docker-compose.yml)

**12. Sentry - 官網**

[**https://develop.sentry.dev/self-hosted/**](https://develop.sentry.dev/self-hosted/)

**13.<< Docker - 從入門到實踐>> 正體中文版** [**https://philipzheng.gitbook.io/docker\_practice/**](https://philipzheng.gitbook.io/docker_practice/)

**14. IPFS - 架設 - Frank**

[**https://www.frank.hk/blog/docker-ipfs**](https://www.frank.hk/blog/docker-ipfs)

**15. IPFS-API 指令 - 官網**[**https://docs.ipfs.tech/reference/kubo/rpc/#origin-based-security**](https://docs.ipfs.tech/reference/kubo/rpc/#origin-based-security)

**16. NestJS - 官網**

[**https://docs.nestjs.com/**](https://docs.nestjs.com/)

**17. Next.js 單元測試 - Leo Chiu**

[**https://medium.com/手寫筆記/nextjs-testing-tutorial-1-ed4b27563761**](https://medium.com/%E6%89%8B%E5%AF%AB%E7%AD%86%E8%A8%98/nextjs-testing-tutorial-1-ed4b27563761)

**18. Next.js Redux Toolkit - Wei Wei**

[**https://www.youtube.com/watch?v=IgFtEGXjl7Y**](https://www.youtube.com/watch?v=IgFtEGXjl7Y)

**19. Tailwind CSS - 官網**

[**https://tailwindcss.com/**](https://tailwindcss.com/)

**20. Next.js - 官網**

[**https://nextjs.org/**](https://nextjs.org/)