

第十組

電機四 李冠儀 B10901091
資管三 嚴邦華 B11705030
資工三 李冠誼 B11902090
資工三 張宸瑋 B11902078
資工三 鄭博允 B11902038

DCManager

Data Center Management System

AGENDA

Live DEMO



System Architecture



Development Process



Unit Test



12 factors

User Requirements

#	User	Requirement	Description
1	機房管理員	自定義廠區機櫃位置與櫃號 (Unit)	可以成功創建機櫃, 可自定義櫃位標籤, 每櫃高度實務上可能會有所不同, 甚至機器本身也有高度差異, (e.g. 1 unit機器, 2 unit機器...)
2	使用者	更新機櫃與機器位置	可以上架新機器、下架機器、有移機功能
3	使用者	分配/回收機器 IP 資訊	如何讓不同服務有對應的網段去分配IP, 而且不會發生碰撞, 並可以註冊/回收IP
4	使用者	搜尋與顯示某服務/單一機器使用狀況	展示目前某一個服務/單一機器所使用機櫃位置與IP使用狀況, 以便評估是否需要擴充網段, 換修機器所在的機櫃與櫃號

User Story

What Admin Need

管理資料中心

可以指定廠區名稱（如：台中機房），管理廠區中的機房，並將機櫃實體配置到各個機房之中。

自由定義機櫃

機房管理員能自定義機房中的櫃位標籤與單位高度，支援不同高度的**Host**與**Racks**，並根據**Datacenter**高度進行管控，以利精確對應不同實體機器與空間的需求。

良好權限管理

藉由登入系統管控權限，區分**Admin**與**User**，避免**User**更改機房的**Racks, Hosts**或**Data Center**等資料。

User Story

What User Need

自由創建服務

宣告**Service**時，系統按照輸入的**Racks**數量以及**subnet**自動分配現有的**Rack**與**IP**地址，並避免**IP**衝突。當**IP**數量不足或**Rack**數量不足時通知使用者增設**Subnet**。

便利更改內容

Service 可擴充**Subnet**，擴充後自動加入可用**IP lists**。所有物件的名稱均可更改，也支援更改**Service Rack**數量, **Host**數量等，以利資料中心管理。

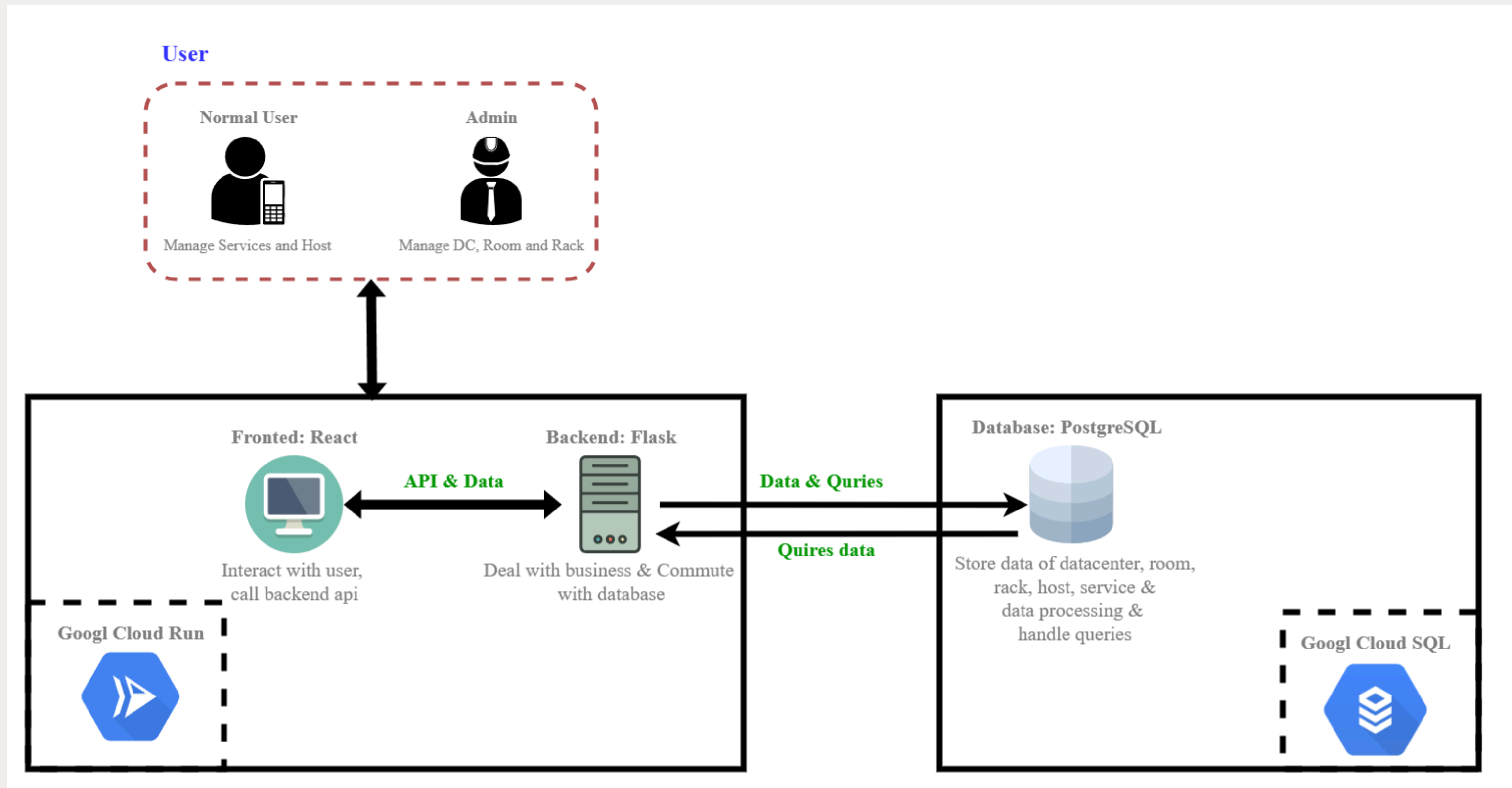
輕鬆完成移機

可以上架與下架新機器，也支援移機功能，**UI**上支援拖拉移機，也支援直接輸入座標，增加管理員操作系統的方便性。

查詢服務狀況

可直接清楚瀏覽各項資訊，也支援輸入服務/主機/資料中心等名稱查詢，包含：廠區、櫃號、**Rack**數、高度、可用**IP**數量、總**IP**數量、使用者等等。

System Architecture

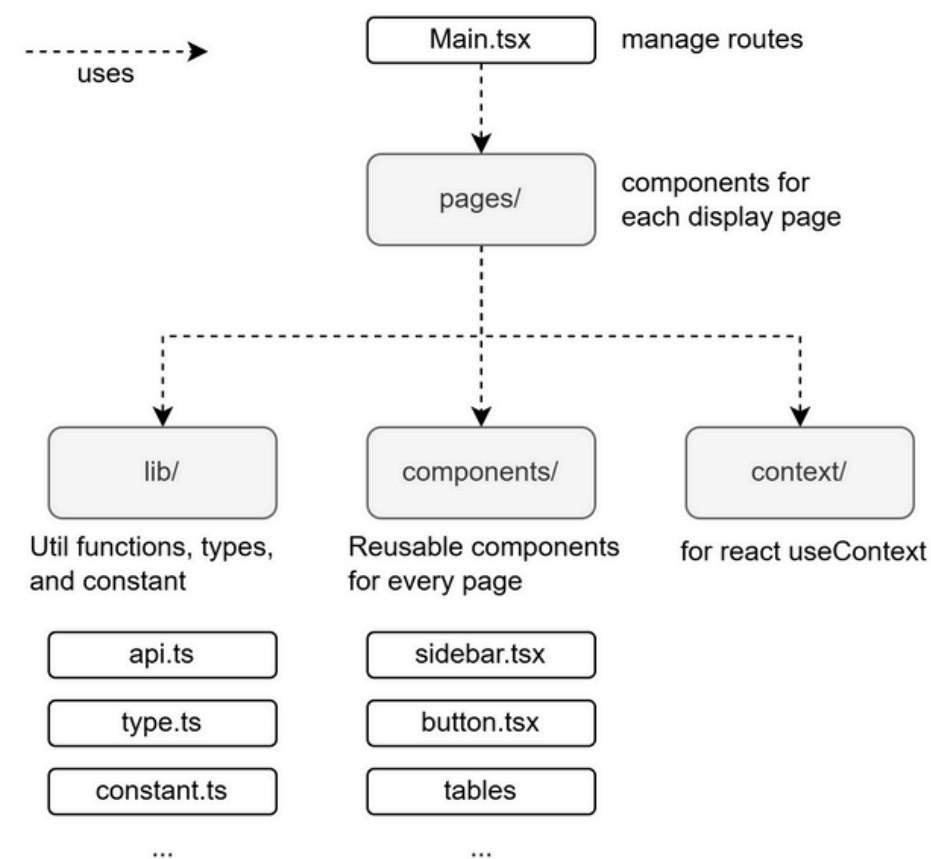


Frontend Architecture

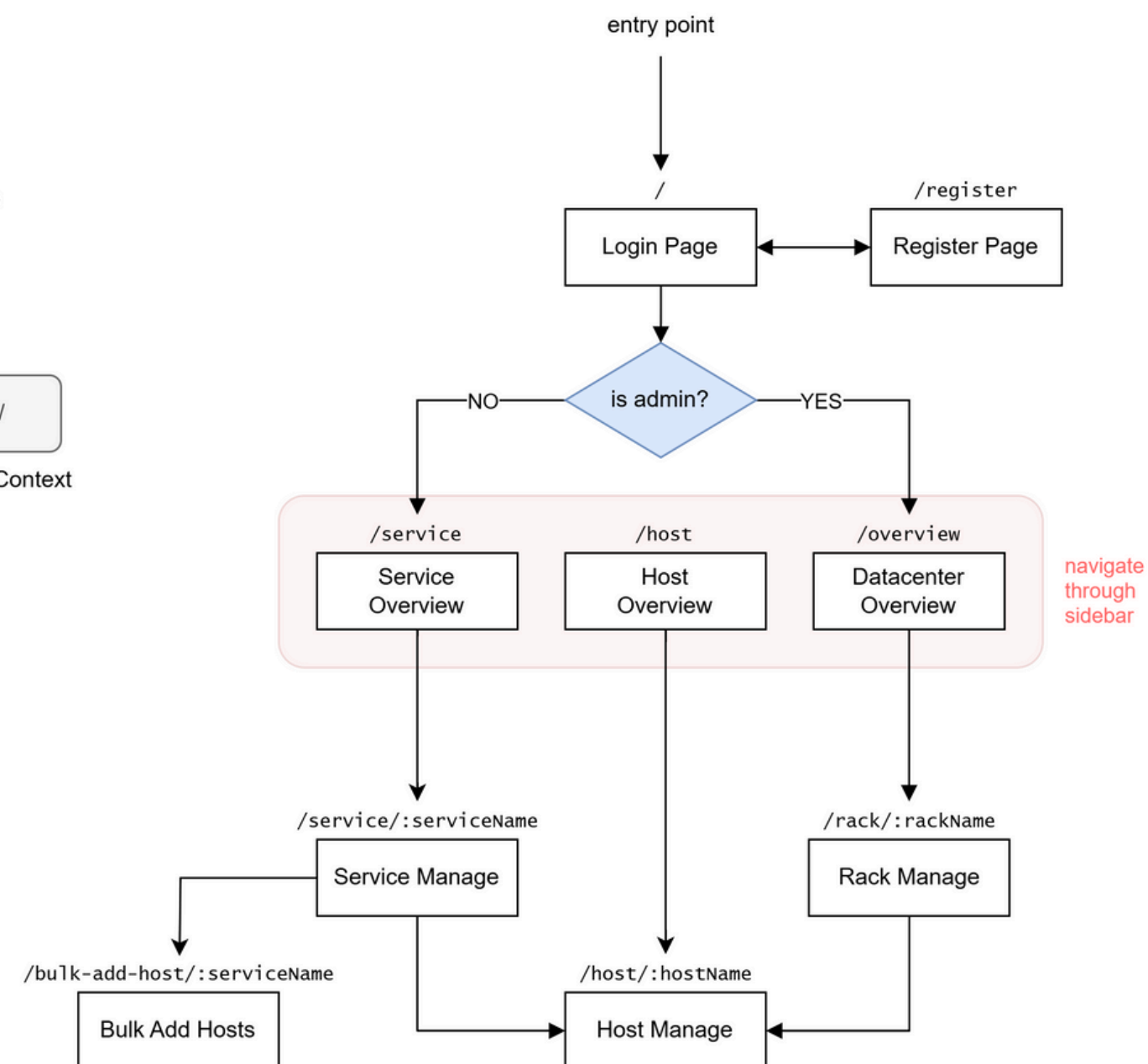
Techstack

- React
- React Router
- NodeJS
- Vite (build tool)
- Nginx (server)

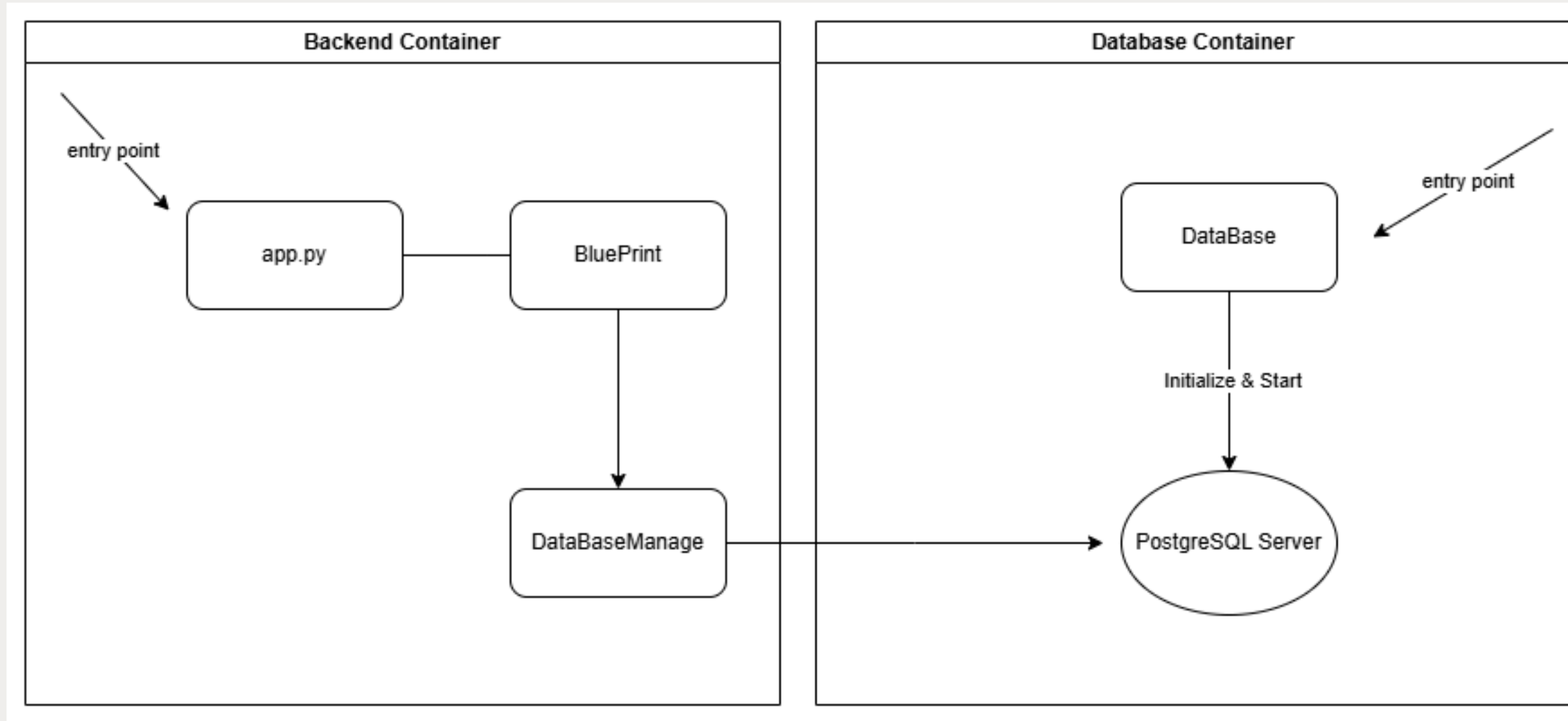
Component Design Logic



Page Navigation Logic



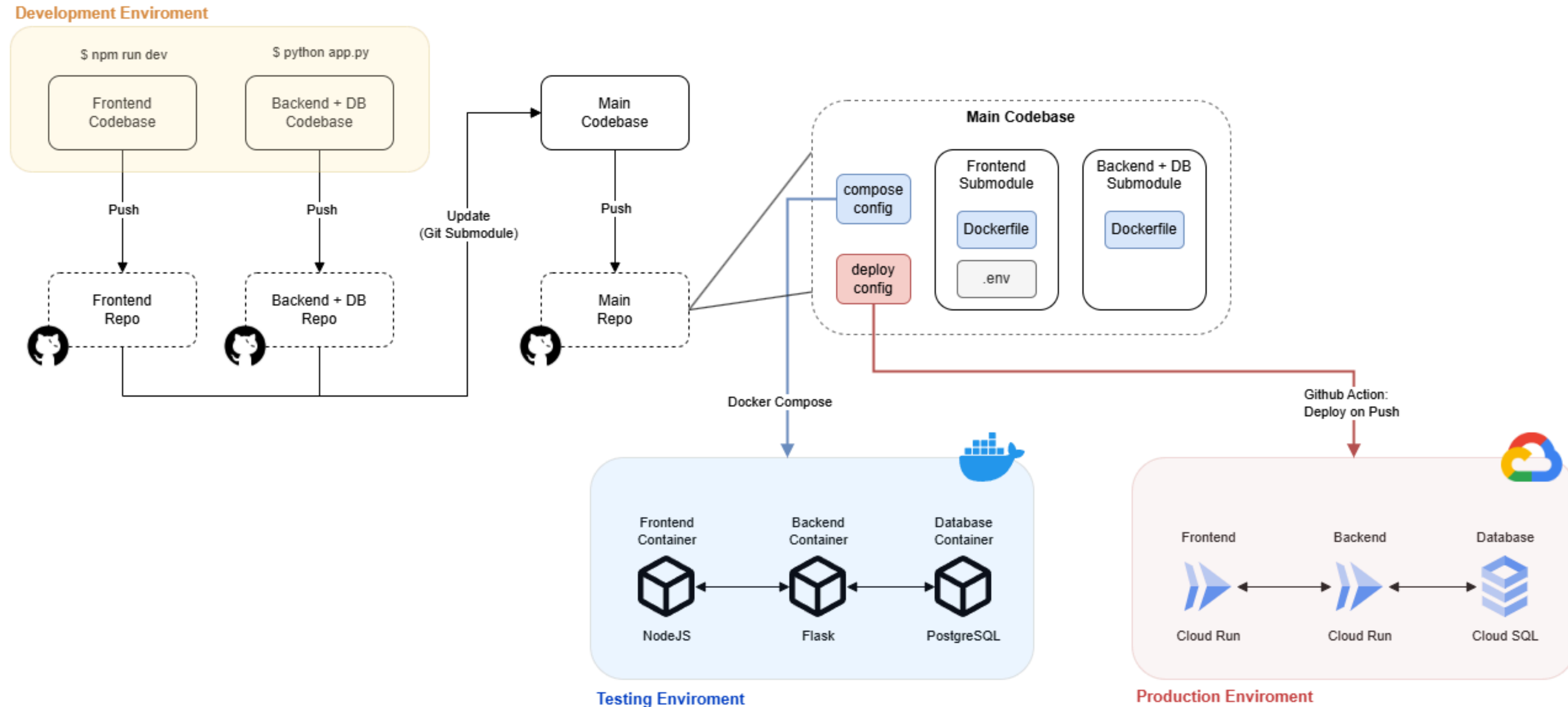
Backend Architecture



Techstack

- Flask
- Python
- PostgreSQL

Development Process



Unit Test

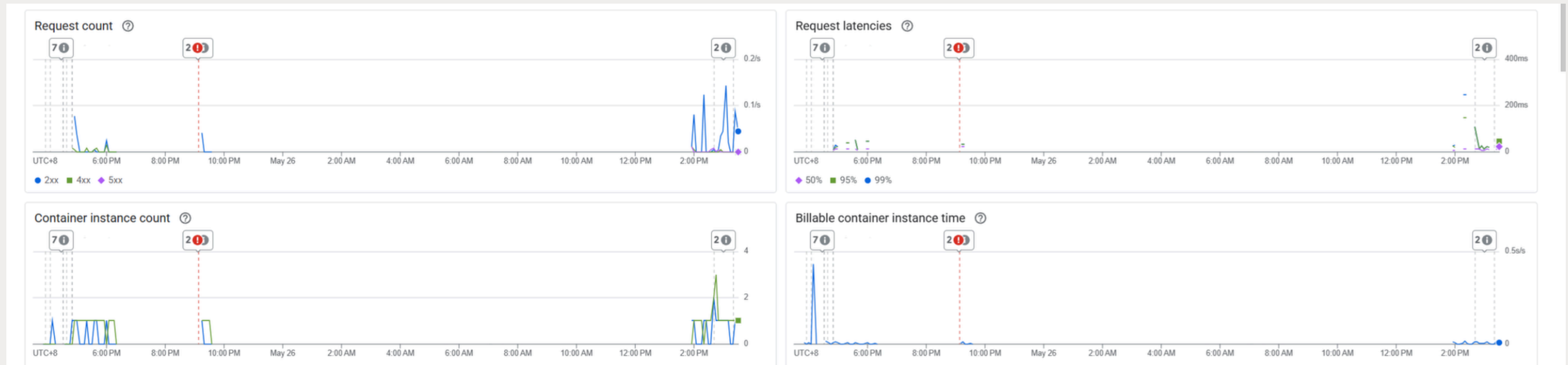
- 使用 Pytest 進行自動測試
- 涵蓋各物件為空、重複註冊、數值為0的get操作等極端測試
- 在各組件上總計涵蓋87%覆蓋率

```
----- coverage: platform linux, python 3.12.3-final-0 -----
Name                               Stmts  Miss  Cover
-----
/mnt/d/Code/DCmanager-backend/Blueprint/Auth.py           43     4    91%
/mnt/d/Code/DCmanager-backend/Blueprint/DataCenter.py      56     3    95%
/mnt/d/Code/DCmanager-backend/Blueprint/Host.py            68     9    87%
/mnt/d/Code/DCmanager-backend/Blueprint/Rack.py            78    13    83%
/mnt/d/Code/DCmanager-backend/Blueprint/Room.py            60     7    88%
/mnt/d/Code/DCmanager-backend/Blueprint/Service.py         84    14    83%
/mnt/d/Code/DCmanager-backend/Blueprint/__init__.py         0     0   100%
-----
TOTAL                                                       389    50    87%

===== 82 passed, 6 warnings
```

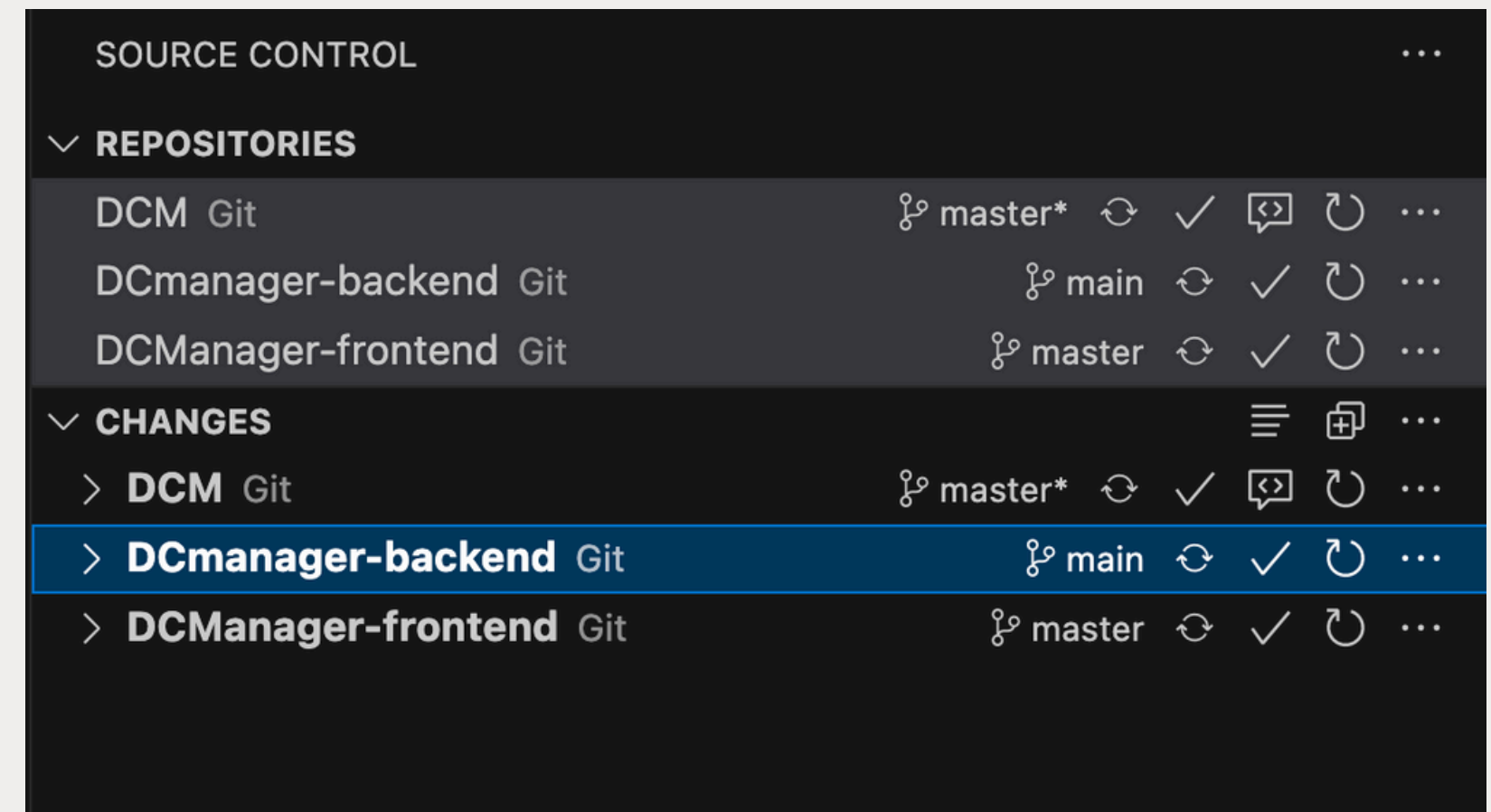
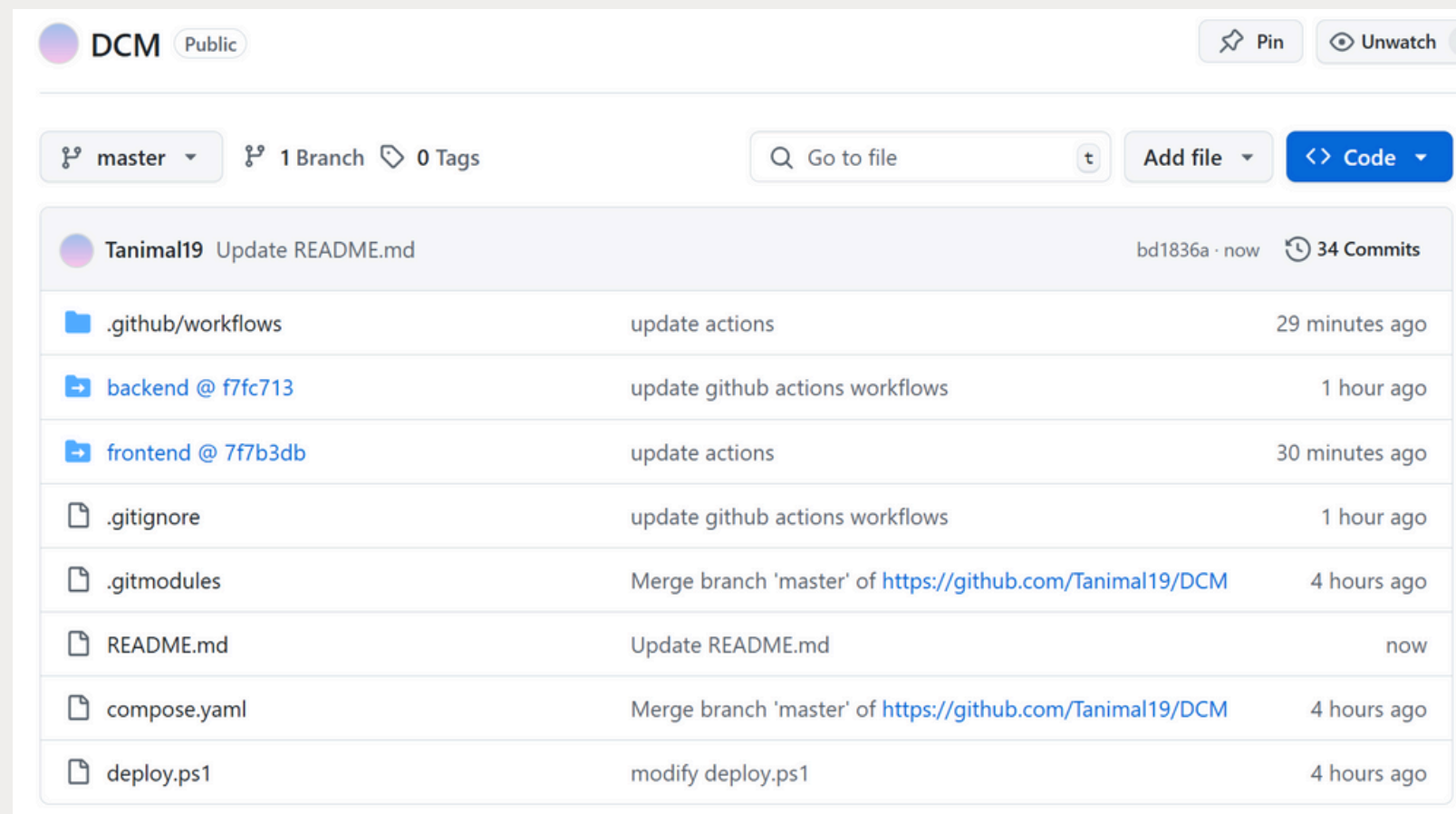
Monitor

- 部署到GCP後，自動監控各項數值，未來服務開始營運以後，還可以設定以下指標：
 - **容器 CPU / Memory 使用率**：CPU > 80% 或 Memory > 90% 超過 5 分鐘警報。
 - **HTTP 請求錯誤率 (HTTP 4xx / 5xx)**：當 5xx rate > 5% 持續超過 3 分鐘，發出警報。
 - **資料庫連線失敗 / 反應時間**：任一 SQL 查詢時間 > 1 秒者發出警報。
 - **API 延遲時間** (Latency, Response Time) 大於1秒警報
 - **容器重啟次數** (Restart Count)：任一 container 過去 10 分鐘內重啟次數 ≥ 3
 - **資源耗盡警示**：某個 service 的可用 IP 數量低於閾值（如剩不到 20%），或某 rack 剩餘空間 < 10% 發出警示。



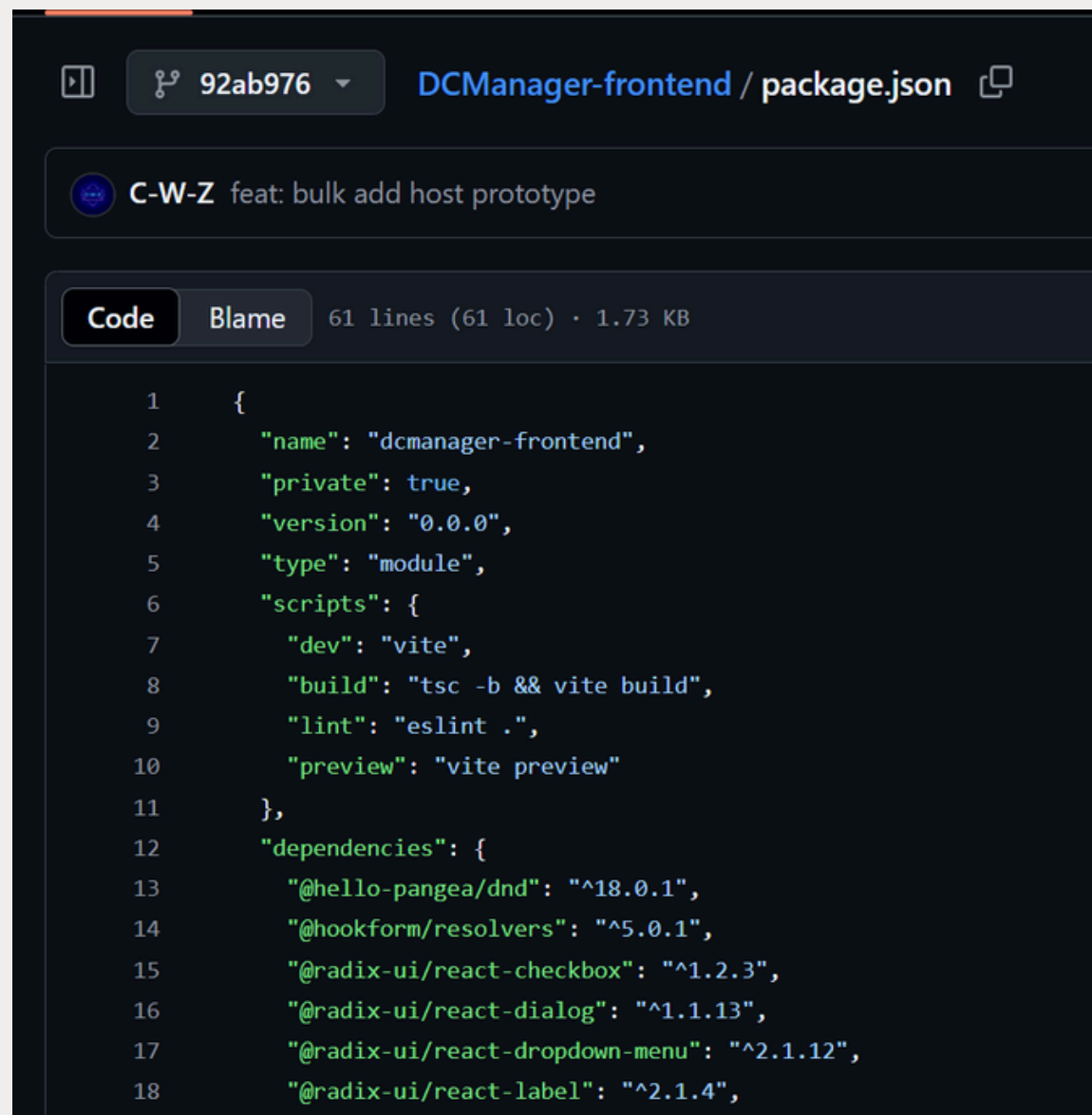
12Factors: Codebase

- 使用 Github 進行版本控制
- 前後端分成不同 Repository
- 使用 Git Submodule 將前後端整合至同一 codebase



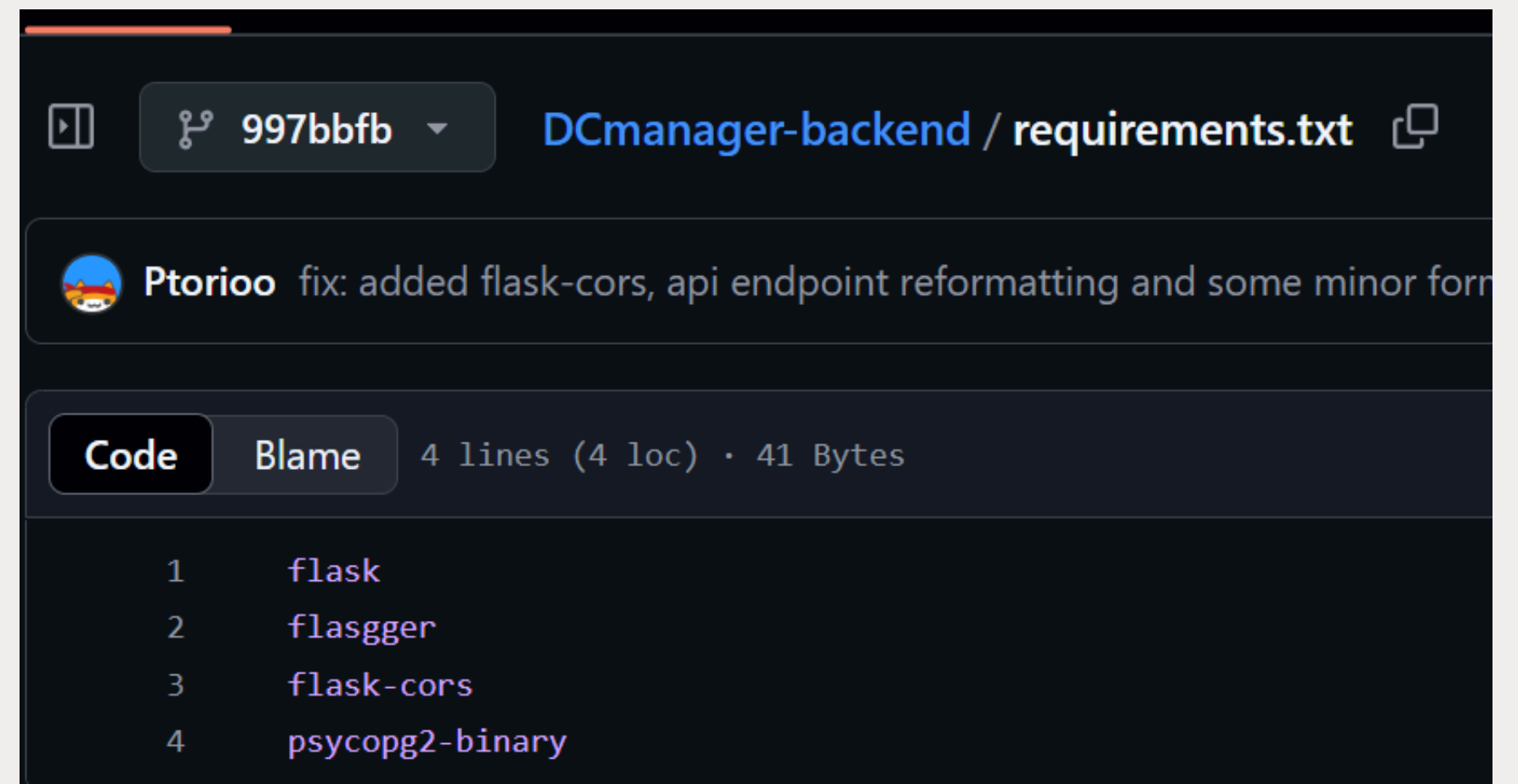
12Factors: Dependencies

- 使用工具載入依賴套件，而非直接併入 codebase
- Backend: pip requirements.txt
- Frontend: npm package.json



A screenshot of a GitHub repository showing the `package.json` file for the `DCManager-frontend` project. The file is 61 lines long (61 loc) and 1.73 KB. The code is displayed in a dark-themed editor with line numbers on the left. The file content is a JSON object with fields for `name`, `private`, `version`, `type`, `scripts`, and `dependencies`. The `scripts` field includes `dev`, `build`, `lint`, and `preview`. The `dependencies` field lists several packages with version constraints.

```
1 {
2   "name": "dcmanger-frontend",
3   "private": true,
4   "version": "0.0.0",
5   "type": "module",
6   "scripts": {
7     "dev": "vite",
8     "build": "tsc -b && vite build",
9     "lint": "eslint .",
10    "preview": "vite preview"
11  },
12  "dependencies": {
13    "@hello-pangea/dnd": "^18.0.1",
14    "@hookform/resolvers": "^5.0.1",
15    "@radix-ui/react-checkbox": "^1.2.3",
16    "@radix-ui/react-dialog": "^1.1.13",
17    "@radix-ui/react-dropdown-menu": "^2.1.12",
18    "@radix-ui/react-label": "^2.1.4",
```

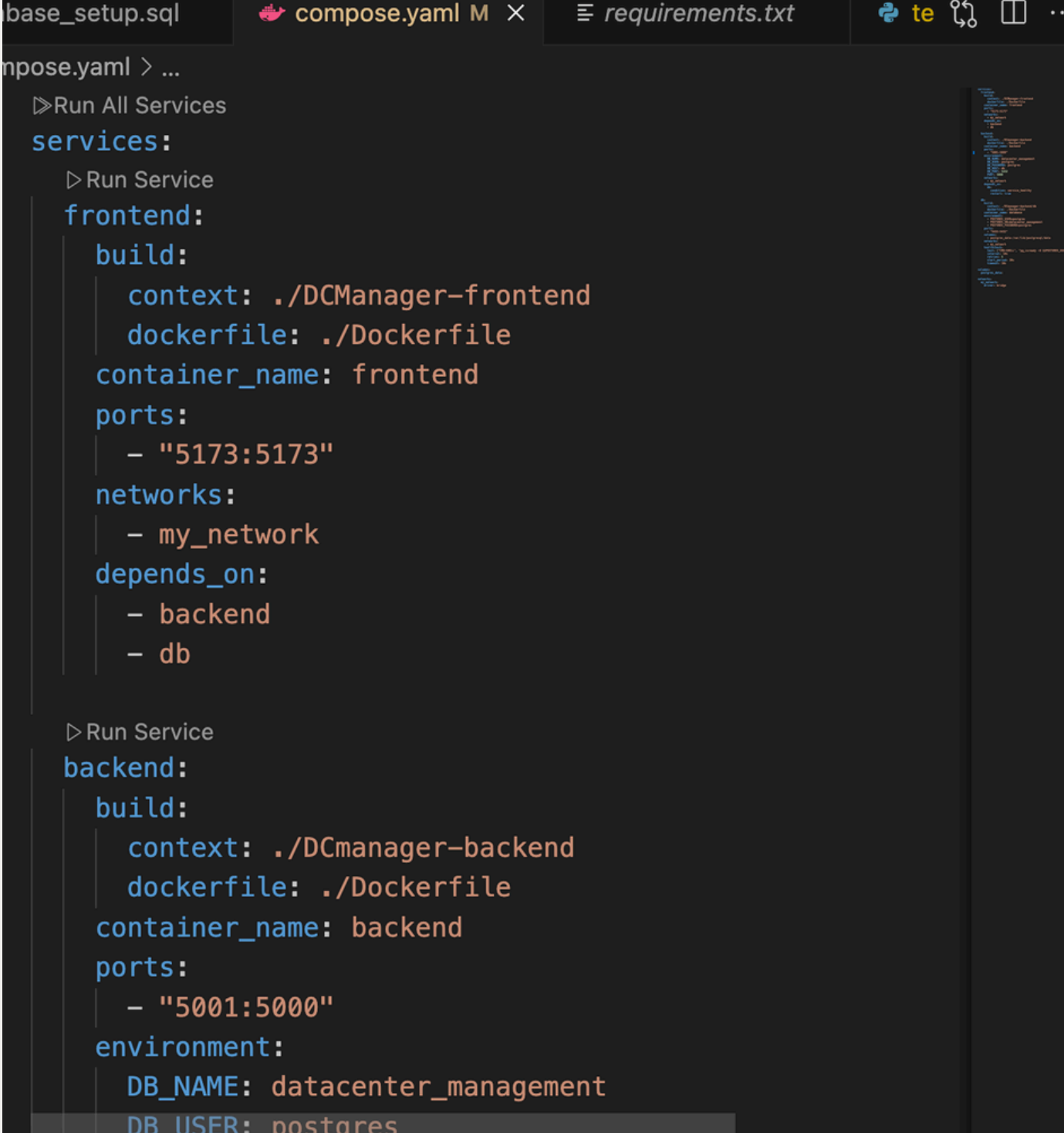


A screenshot of a GitHub repository showing the `requirements.txt` file for the `DCmanager-backend` project. The file is 4 lines long (4 loc) and 41 Bytes. The code is displayed in a dark-themed editor with line numbers on the left. The file content lists four dependencies: `flask`, `flasgger`, `flask-cors`, and `psycopg2-binary`.

```
1 flask
2 flasgger
3 flask-cors
4 psycopg2-binary
```

12Factors: Config

- 將資料庫連線資訊、PORT 號等以**環境變數**的方式定義在 .env 或是 Dockerfile 內



```
base_setup.sql  compose.yaml M x  requirements.txt  te  ..
compose.yaml > ...
  > Run All Services
services:
  > Run Service
  frontend:
    build:
      context: ./DCManager-frontend
      dockerfile: ./Dockerfile
    container_name: frontend
    ports:
      - "5173:5173"
    networks:
      - my_network
    depends_on:
      - backend
      - db
  > Run Service
  backend:
    build:
      context: ./DCmanager-backend
      dockerfile: ./Dockerfile
    container_name: backend
    ports:
      - "5001:5000"
    environment:
      DB_NAME: datacenter_management
      DB_USER: postgres
```


12Factors: Backing services

- 將 SQL 資料庫從前後端獨立出來，並使用 socket 方式連接，減少資料庫替換時造成的程式修改成本

```
backend > DataBaseManage > connection.py > test_connection
1  import os
2  import psycopg2
3  from psycopg2.pool import SimpleConnectionPool
4
5  # Database connection configuration
6  DB_CONFIG = {
7      "dbname": os.environ.get("DB_NAME", "datacenter_management"),
8      "user": os.environ.get("DB_USER", "postgres"),
9      "password": os.environ.get("DB_PASSWORD", "postgres"),
10     "host": os.environ.get("DB_HOST", "localhost"),
11     "port": int(os.environ.get("DB_PORT", "5433")),
12 }
13
14 # Create a connection pool
15 pool = SimpleConnectionPool(1, 10, **DB_CONFIG)
16
```

12Factors: Build, Release, Run

- 使用 Github Actions 自動在特定 Branch 更新時打包程式碼，並 deploy 到 GCP 上

The screenshot shows the GitHub Actions interface for a workflow named "Build and Deploy to Cloud Run". The workflow is in a "Success" state. The left sidebar shows the "Summary" tab selected, with "Jobs" listed below it, including a "deploy" job. The main content area shows the "deploy.yml" file, which is triggered by a "workflow_dispatch" event. The "deploy" job is shown as successful with a duration of 2m 10s.

← Build and Deploy to Cloud Run

✓ Build and Deploy to Cloud Run #5

Summary

Jobs

✓ deploy

Run details

Usage

Workflow file

Re-run triggered 38 minutes ago

Status

Total duration

Artifacts

Tanimal19 1c4771e master Success 2m 23s -

deploy.yml

on: workflow_dispatch

✓ deploy 2m 10s

12Factors: Processes

- 採用 Flask 搭配 RESTful API，後端對於每個 request 皆為獨立運行，且無狀態

12Factors: Concurrency

- Flask 可以搭配 Gunicorn 達成多進程並行，以應對高流量情境

12Factors: Port binding

- 使用 Dockerfile 對各個服務指定 port，確保各服務對外的接口固定:

Frontend: 5173 → 80

Backend: 5000 → 5000

Database: 5432

```
compose.yaml
1  services:
2    frontend:
3      build:
4        context: ./frontend
5        dockerfile: ./Dockerfile
6      container_name: frontend
7      ports:
8        - "5173:5173"
9      networks:
10       - my_network
11      depends_on:
12        - backend
13        - db
14
15    backend:
16      build:
17        context: ./backend
18        dockerfile: ./Dockerfile
19      container_name: backend
20      ports:
21        - "5000:5000"
22      environment:
23        DB_NAME: datacenter_management
24        DB_USER: postgres
25        DB_PASSWORD: postgres
26        DB_HOST: db
27        DB_PORT: 5432
28        PORT: 5000
```

12Factors: Disposability

- GCP Cloud Run，可以透過增加 minimum instances 數量來避免冷啟動帶來的延遲

12Factors: Dev/Prod Parity

- Dev 環境使用 Docker Container 進行測試
- Prod 環境則將 Container 直接放到 Cloud Run 上
- Dev/Prod 使用相同的 Dockerfile 以確保環境一致

12Factors: Logs

- GCP, Docker運行皆可不斷產生Log，監控各項數值與狀況

12Factors: Admin processes

- 部分管理員程序，如Reset DB，可直接Run admin/management tasks as one-off processes，如下圖的Script

```
OPEN EDITORS
database_setup.sql DCmanager-backen...
testcrud.py DCmanager-backend/Da... 1
datacentermanager.py DCmanager... 1
servicemanager.py DCmanager-ba... 2
connection.py DCmanager-backend... 2
x reset_db.py DCmanager-backen... 2, U
hostmanager.py DCmanager-backe... 1
Host.py DCmanager-backend/BluePrint

DCM
v DCmanager-backend
v DataBaseManage
datacentermanager.py
hostmanager.py 1
rackmanager.py
roommanager.py
servicemanager.py 2
testcrud.py 1
usermanager.py
> db
v scripts
reset_db.py 2, U

DCmanager-backend > scripts > reset_db.py > ...
15
16 @click.command()
17 @click.option('--yes', is_flag=True, help='Confirm reset without
18 def reset_db(yes):
19     if not yes:
20         confirm = input("! This will delete all data. Are you su
21         if confirm.lower() != 'y':
22             print("X Canceled.")
23             return
24
25     try:
26         conn = psycopg2.connect(**DB_CONFIG)
27         cursor = conn.cursor()
28
29         print("🔄 Resetting tables...")
30
31         cursor.execute("TRUNCATE hosts, racks, ips RESTART IDENTI
32
33         conn.commit()
34         print("✅ Database reset completed.")
35
```

謝謝大家

Github: <https://github.com/Tanimal19/DCM>

Demo Site: <https://frontend-566579704717.asia-east1.run.app/>