ChatGPT

Containers 架構設計說明文件

本文件說明你目前建立的 `containers` 系統邏輯與架構目標，定位為一套 **可在 CDN 模式下使用的模組化應用框架**，支援：動態註冊、模組懶載入、依賴注入、策略模式、事件處理等擴展能力。

📦 架構總覽

1. Container 核心功能

- **register(key, module)**：註冊模組，支援元件、store、策略、service、plugin 等
- **resolve(key)**：取用已註冊模組，具 DI / Mock injection 功能
- **registerComponent(app, module, options?)**：自動註冊 Vue 元件與對應樣式

2. 支援模組類型

類型	範例	描述
元件 (component)	`PaginationComponent`	UI元件，自帶樣式與註冊邏輯
Store	`tokenStore`, `menuStore`	Pinia風格 Store，可注入 DI
策略 (strategy)	`MockApiStrategy`, `LiveApiStrategy`	封裝請求邏輯或策略切換
指令 (command)	`AddCommand`, `UndoCommand`	可執行/回復的命令模式模組
服務 (service)	`fetchService`, `popupBridge`	提供資料或橋接能力的純邏輯函式

3. 設計理念

- **模組化（Modular）**：每個功能單元皆為獨立模組，可封裝、載入、替換
- **注入式（Injectable）**：依賴統一從 container 中解析，利於 mock / 測試
- **懶加載（Lazy Loadable）**：頁面級模組以 async import 動態載入
- **插件化（Pluggable）**：install() 標準化所有模組掛載方式，支援 app.use()

檔案結構建議

```
containers/
├── index.js          # container 初始化與 API（register/resolve）
├── registerComponent.js  # 元件註冊工具（自動掛樣式）
├── utils/            # 架構服務層函式，如 getBaseUrl, injectCss
│   └── componentUtils.js  # 共用 injectCss, getBaseUrl
├── stores/           # 所有 store 模組
│   └── tokenStore.js
├── strategies/       # API 請求策略或 mock 模組
│   └── mockApiStrategy.js
```

```
├── commands/       # 命令模式模組
│   └── addCommand.js
└── services/       # popup, fetch 這類邏輯型函式
    └── popupBridge.js
```

模組撰寫標準

install() 模式（元件 / plugin 模組）

```
// pagination-module/index.js
import { PaginationComponent } from './component.js'
import { injectCss, getBaseUrl } from '@/containers/utils/componentUtils.js'

export default {
 install(app, options = {}) {
  const cssPath = options.css ?? `${getBaseUrl(import.meta.url)}/style.css`
  injectCss(cssPath)
  app.component('PaginationComponent', PaginationComponent)
 },
}
```

store 模組

```
export const tokenStore = () => {
 const token = ref(localStorage.getItem('token'))
 const setToken = (val) => {
  token.value = val
  localStorage.setItem('token', val)
 }
 return { token, setToken }
}
```

開發建議

哪些該放在 container？

是否放入 containers	判斷依據
要放入	需被註冊、注入、mock、替換、策略切換
不建議放入	單純工具函式，如 formatDate、isEmailValid 等

可考慮建立 `/lib/` 或 `/shared/utils/` 統一放置純工具函式。

下一步建議

1. 抽出一個 `registerPlugin()`，讓所有可註冊模組符合統一規格（可含 install、meta）
2. 建立 container config 系統，支援註冊策略、環境 mock、延遲載入
3. 撰寫一份 container 自動註冊機制（例如掃描路徑或讀取 json 設定）

如需轉為 CLI 專案腳手架或前端 mini framework，也可以逐步拆出 build config、plugin 機制、模組描述格式等，進入完整模組化架構。