**實作細節**

**Server:**

* **使用函式庫:**

[**uWebSockets**](https://github.com/uNetworking/uWebSockets)

[**SQLiteCpp**](https://github.com/SRombauts/SQLiteCpp)

[**nlohmann/json**](https://github.com/nlohmann/json)

* **資料傳遞格式:**[**Reversi Doc**](https://cypress-insect-afc.notion.site/Reversi-doc-1b7d35cbd4ab8089b5e4d701ce12bbf4)
* **遊戲模式:**
* **本地雙人遊玩:**提供本地兩個人一起遊玩。
* **人機對戰:**與機器人遊玩，機器人會計算能夠翻最多棋子的步數並放置。
* **線上對戰:**與線上玩家對戰，點下配對後會將玩家放入配對佇列中，直到有第二位玩家加入。
* **遊戲機制:**

為了確保玩家能夠即開即玩，若是尚未遊玩過的玩家會被伺服器分配一組id，並且由客戶端保存，這個遊戲id代表玩家的在伺服器的身分，同時保存了玩家的遊戲資料。

玩家進入遊戲可以選擇模式，每個模式都有不同的編碼，或是玩家也可以選擇曾經參與過的對局，並且繼續遊玩或是重看回放。

若是在玩家連線對局，在放置的時候會讓該名使用者的倒數計時器減掉這次放置的時間，並向前端更新，而實際的倒數計時則由前端進行，一旦前端倒數結束，則會通知伺服器結束該對局。

* **架構:**

1. **伺服器介面:**

透過map解析客戶端發送的各種資料，提供相應的處理及回覆。使用者登入時，調用uWebSockets內的getUserData存取使用者指針，保存當前使用者資訊。

1. **資料庫:**

透過SQLiteCpp建立與資料庫的互動，我們包裝了各種靜態方法，如保存、註冊等，提供伺服器介面調用。

1. **遊戲邏輯:**
2. **遊戲流程**

先初始化棋盤和遊戲裡各種參數，並且找出第一位玩家所可以放置的有效棋子，以及其對應到的可吃棋子。

當玩家放置棋子的時候，會去先檢測先根據前一步或初始化時所找出的有效格子來判斷玩家所點擊的位置是否合法，如果合法的話再去執行翻轉棋子的邏輯，這之後就分別計算分數和計算下一個人的有效格子以及可吃棋子。

如果下一個人沒有任何有效格子，就切換玩家再去找該玩家的有效格子，如果也沒有，就代表雙方都沒有可以下棋的地方。

最後判斷輸贏，並更新變數，遊戲就結束了。

1. **技術細節**

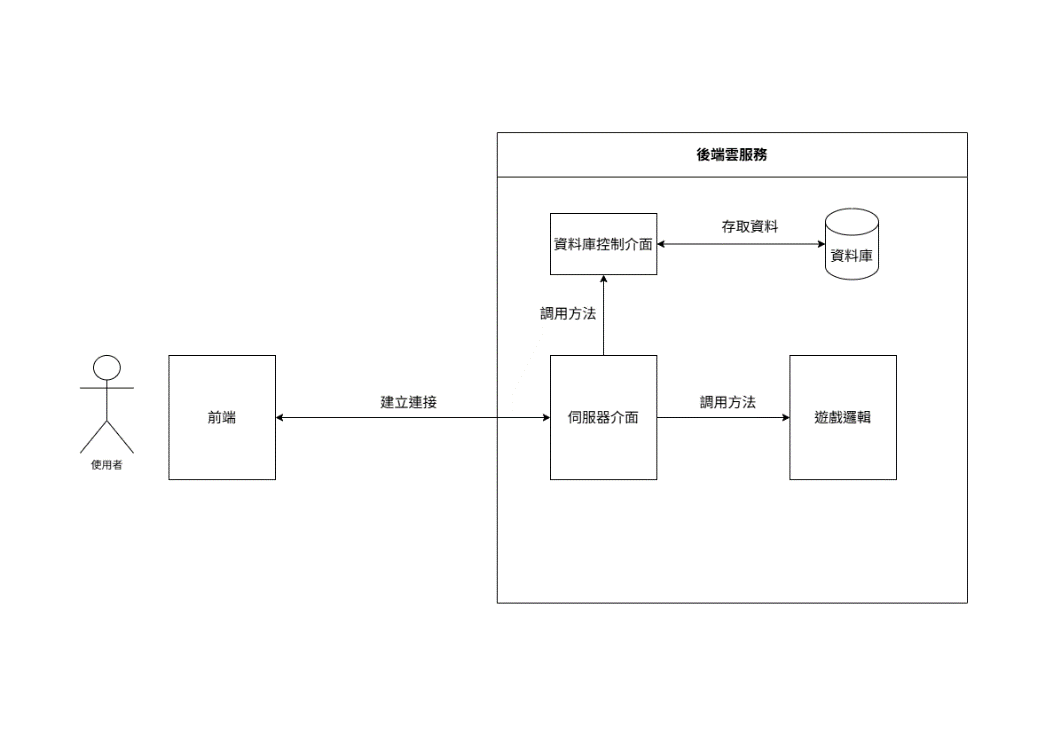
棋盤上的資料格式為: 0:空白格/1:黑棋/2:白旗

1. **初始化棋盤:**

棋盤為二微陣列，首先將其盤上所有原素設定成0，再將中間的4格設定成規則上的初始棋子(2顆黑棋、2顆白棋)。

1. **計算有效格子:**
   * + 1. 從對方的棋子去檢查周圍8格有沒有空格
       2. 取一個向量(對方棋子 - 空格)
       3. 用上一步的向量取反向量的方向去找有沒有我方的棋子
       4. 用(x,y)的方式加入到有效格子validSquare的陣列中
2. **計算可吃棋子:**
3. 從去檢查每一個有效格子周圍8格有沒有敵方棋子
4. 取一個向量(有效格子 – 敵方格子)
5. 用上一步的向量的方向去找所有敵方的棋子，並且確保此方向上一定有我方棋子
6. 用(x,y)的方式將所有敵方棋子的座標加入到以當前的有效格子為key的map中
7. **放置棋子:**
8. 從去檢查每一個有效格子周圍8格有沒有敵方棋子
9. 取一個向量(有效格子 – 敵方格子)
10. 用上一步的向量的方向去找所有敵方的棋子，並且確保此方向上一定有我方棋子
11. 將在這個方向上所有敵方棋子翻轉
12. **判斷輸贏:**
13. 場上沒有任何空白格(0)
14. 雙方均沒有有效格子
15. 其中一方時間到
16. **計算分數:**

用迴圈去計算白方和黑方的棋子數量。

****

架構圖

**Client:**

1. **接收資料**

透過 webSocket 事件接收後端的更新、回傳資料，並使用 react 狀態管理系統儲存狀態。

1. **介面**

使用 react 搭配 css 完成基本 UI 設計，使用 react 狀態管理系統達成頁面切換效果 (本質上為單網頁渲染)。

1. **事件發送**

在登入、加入配對、加入遊戲、放置棋子等等互動中向伺服器發送 webSocket 事件，和伺服器進行交互。