08. More on Backend + Database

Ric Huang / NTUEE

(EE 3035) Web Programming

上星期跟大家介紹了 backend 的基本知識

並且用 React/Axios/Express/Node

打造了一個簡單的全端 project

(HW#5: Number Guessing Game)

這星期將會帶大家更深入了解 backend 的操作機制

並且介紹 database 的基本知識

以及 mongoDB 的簡單實作

希望讓大家都有能力自行建置一個全端的 project

Outline

- Express + HTTP revisit
- RESTful APIs
- Introduction to Database
- MongoDB + Mongoose ORM
- With React + Axios

我們今天會針對每一個 topics

逐一/漸進的建立幾個 projects

建議大家可以把這幾個 projects 獨立整理到一個目錄

以利比較與學習

01. Express + HTTP Revisited

準備 project 的 package.json

1. Create project directory (e.g. 01.express-http)

```
mkdir 01.express-http && cd 01.express-http
```

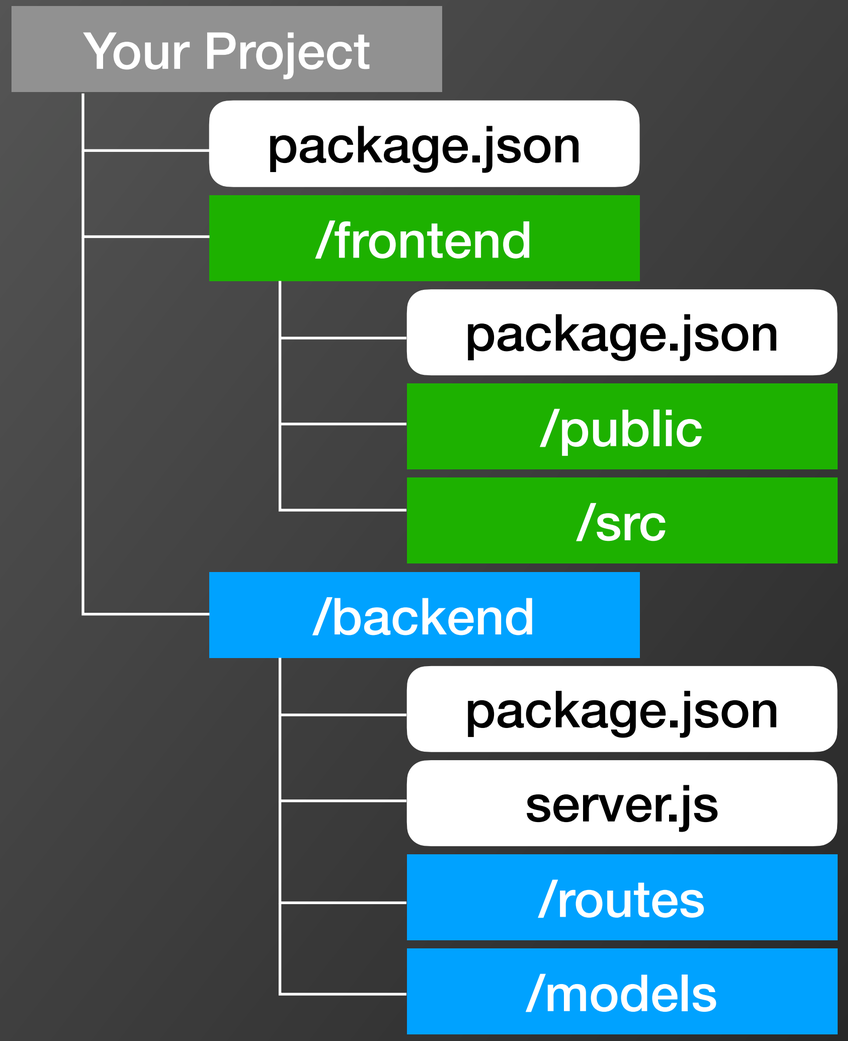
2. Create a “package.json” for this project

```
yarn init -y
```

3. Add these lines to “package.json”

```
  "scripts": {
    "start": "cd frontend && yarn start",
    "server": "cd backend && yarn server"
  }
```

Project File Structure



Project entry

Frontend root directory

Frontend related packages

index.html, CSS files

index.js, App,js, axios.js

Backend root directory

Backend related packages

Server entry

Define service routes

Define DB schema

Let’s prepare a simple Express server

1. Create directory for backend

```
mkdir backend
```

2. Create a “package.json” for backend

```
cd backend
yarn init -y
```

3. Install “express” locally

```
yarn add express --save
```

Let's prepare a simple Express server

4. Create and edit "server.js"

```
import express from 'express';
const app = express();
const port = process.env.PORT || 4000;
app.get('/', (req, res) => {
  res.send('Hello, World!');
});
app.listen(port, () =>
  console.log(`Example app listening on port ${port}!`),
);
```

5. Add these lines to "package.json"

```
  "scripts": {
    "server": "nodemon server.js"
  },
```

Test the project...

Under the project directory —

- Start server

```
yarn server
```

- If it complains about some module missing, install it.

What do you see?

Test the project...

- You will probably see this error message:

```
(node:19449) Warning: To load an ES module, set
"type": "module" in the package.json or use
the .mjs extension.
```

- Add this line to "package.json":

```
  "type": "module",
```

Restart the server. What do you see?

Test the project...

- 這行 'Hello, World!' 印到哪裡去了？

```
app.get('/', (req, res) => {
  res.send('Hello, World!');
});
```

A closer look at the backend

- 試著用 browser 打看 “localhost:4000” 看看

==> What do you see?

- 所以你看到的是 frontend 還是 backend?
  (咦，我們不是還沒有寫 front-end?)
- 事實上，當你用 browser 打開 “localhost:4000” 的時候，你就是用 browser (當作 front-end client) 向 server 打了個 GET 的 HTTP request，也就是觸發了 “server.js” 的這行的回應：

```
app.get('/', (req, res) => { res.send('Hello, World!'); });
```

Express server

Define “GET” API

符合路徑時動作

“GET” request from browser
(在 server.js console.log 試試看)

A ref to the response from server

Send a “Hello, World!” From server

Add more APIs to “server.js”

- 把這幾行加進去 —

```
app.get('/', (req, res) => {
  res.send('Received a GET HTTP method');
});

app.post('/', (req, res) => {
  res.send('Received a POST HTTP method');
});

app.put('/', (req, res) => {
  res.send('Received a PUT HTTP method');
});

app.delete('/', (req, res) => {
  res.send('Received a DELETE HTTP method');
});
```

加在 "Hello, World" 前面 or 後面試試看

有看到任何的變化嗎？Post/Put/Delete 做了什麼？

cURL: A command-line URL tool

- 安裝 “curl”，讓我們用 command line 來跟 backend 互動

```
> curl http://localhost:4000
// default 是 GET method
```

- 試試看 POST / PUT / DELETE

```
> curl -X POST http://localhost:4000
> curl -X PUT http://localhost:4000
> curl -X DELETE http://localhost:4000
```

試試看 routing

- 把這幾行加到 “server.js”（加前面 or 加後面？有差嗎？）

```
app.post('/users', (req, res) => {
  res.send('POST HTTP method on users resource');
});

app.put('/users/:userId', (req, res) => {
  res.send(
    `PUT HTTP method on users/${req.params.userId} resource`,
  );
});
```

- 用 curl 呼叫看看 —

```
> curl -X POST http://localhost:4000/users
> curl -X PUT http://localhost:4000/users/12
```

POST data to server

- POST / PUT methods 理論上是要把資料寫到 server 上 (的資料庫)，試試看這樣傳給 server:

```
curl -X POST http://localhost:4000 \
     -H "Content-Type:application/json" \
     -d '{"text":"Hi again, World"}'
```

- 有看到 “Hi again,World” 印在任何地方嗎？
  - Why not?
  - 試試看在 app.post() 中加上 “console.log(req)
  - What do you see?

HTTP Body Parser [ref]

- HTTP 傳過去的資訊是不是很複雜？

==> 所以我們要善用套件 (e.g. body-parser)

- 在 “server.js” 中適當的地方加入

```
import bodyParser from 'body-parser';

// Parses the text as JSON and exposes the resulting
// object on req.body.
app.use(bodyParser.json());
```

- 然後在 app.post() 中加入：

```
console.log(req.body.text);
```

- 就可以看到 “Hi again, World” 了！

01. Express + HTTP Revisited: Learnings

- 如何從頭建立一個 backend (Express) server project
- 如何使用 Command Line Tool “Curl” 去操作 HTTP requests (GET/POST/PUT/DELETE)

Quick Recap on HW#5

- HW#5 總共有三個 APIs:
  - start: 按下“start”按鈕進入遊戲模式，server 必須隨機產生一個亂數
  - guess: client 端猜一個數字，送到後端，比對大小，再由後端回傳結果與建議
  - restart: 按下“restart”按鈕，重新開始遊戲，server 必須再隨機產生一個亂數

這三個 APIs 其實都沒有寫資料到後端，所以你覺得他們應該是：GET/POST/PUT/DELETE 哪一種 APIs 呢？

Recap: 常見的 HTTP Request Methods (ref)

- GET — The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.
- POST — The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server.
- PUT — The PUT method replaces all current representations of the target resource with the request payload.
- DELETE — The DELETE method deletes the specified resource.

Recap: An Analogy of HTTP Request Methods (ref)

- 假設現在我們要點餐，
- 我們必須先知道菜單是甚麼（get），
- 我們會向服務生點餐（post），
- 我們想要取消剛才點的餐點（delete），
- 我們想要重新點一次（put），
- 我們想要加點甜點和飲料（patch）。

Quick Recap on HW#5

- HW#5 總共有三個 APIs:
  - start: 按下“start”按鈕進入遊戲模式，server 必須隨機產生一個亂數 => POST
  - guess: client 端猜一個數字，送到後端，比對大小，再由後端回傳結果與建議 => GET
  - restart: 按下“restart”按鈕，重新開始遊戲，server 必須再隨機產生一個亂數 => POST

雖然有時候你用錯 request method 也是可以 work,
但它很可能會造成 client/server 端的溝通困難

所以前端與後端要有良好的
溝通協定/介面

RESTful API

- API (application programming interface), 顧名思議就是定義好一個介面的協定來讓溝通的兩端得以獨立的開發，而不會受到任何一方改進、除錯的影響
- 在網路服務前後端的溝通，最有名、廣為大家遵循的 APIs 就是 RESTful APIs
- 不過嚴格來說，RESTful 並不是一個 API 的規範，而是一種 style

REST

- 表徵性狀態傳輸 (REpresentational State Transfer)
- 2000 年就由 Roy Fielding 在博士論文提出
- 但在 Rails 1.2 實作後才紅起來
- 是設計「風格」而不是標準
- 符合 REST 設計風格的 Web API 稱為 RESTful API

RESTful

- ✓ 資源是由 URI 指定
- ✓ 對資源的操作包括獲取、創建、修改、刪除，剛好對應 GET、POST、PUT、DELETE
- ✓ 依照不同需求給予不同格式的回應
- ✓ 利用 Cache 增加性能
- ✓ 正確的 HTTP status code

資源是由 URI 指定

- Bad examples —

```
/getDevice
/getDeviceList
/getDevices
/getMyDevice
/getOtherDevice
/updateDeviceList
/addNewDevice
```

- Why bad?
  - 不統一，不好記憶，重寫會不一樣
  - get / post 無法看出來
  - 參數如何傳遞？

RESTful APIs

- Good examples —

```
GET /devices
POST /devices/
GET /devices/123
PUT /devices/123
DELETE /devices/123
```

- Why better?
  - 不同人依舊會實作相同 API
  - 清楚表示資源關係

More examples of RESTful APIs

```
GET http://stackoverflow.com/questions
POST http://stackoverflow.com/questions/389169
GET http://stackoverflow.com/tags
PUT http://stackoverflow.com/users
DELETE http://stackoverflow.com/users/3012290
```

RESTful with Parameters

- Filter
- Sorting
- Searching

```
GET /tickets?q=return&state=open&sort=-prority,created_at
```

Recall: HTTP 動詞 (Request 動作)

取得資源清單 -> GET /resources

新增資源 -> POST /resources

取得單一資源 -> GET /resources/:id

修改單一資源 -> PUT /resources/:id

刪除單一資源 -> DELETE /resources/:id

更新資源部份內容 -> PATCH /resources

只回傳 HTTP header -> HEAD /resources/:id

URI 名詞（指定資源）

- 相對於 HTTP 動詞，URI 就是名詞了
- URI 由 prefix + API endpoint 組成。Prefix 的部份可有可無，例如 /api 或 /api/v1，API endpoint 的設計，幾個重要原則如下(ref)：
  - 一般資源用複數名詞，例如 /books 或 /books/123
  - 唯一資源（亦即對client而言只有一份的資源）用單數名詞。例如 /user/subscriptions，其中 user 是指目前驗證的使用者，所以用單數
  - 資源的層級架構，可以適當反應在 API endpoint 設計上，例如 /books/123/chapters/2
  - URI components 都用小寫

Recall: 回傳狀態碼

- API回傳的結果，應使用適當的HTTP狀態碼
  - 1xx -> 訊息
  - 2xx -> 成功
  - 3xx -> 重導向
  - 4xx -> 用戶端錯誤
  - 5xx -> 伺服器端錯誤

常見 Status Codes

```
200 OK
--
301 Moved Permanently
302 Found
304 Not Modified
--
400 Bad Request
401 Unauthorized
403 Forbidden
404 Not Found
405 Method Not Allowed
408 Request Timeout
--
500 Internal Server Error
502 Bad Gateway
503 Service Unavailable
504 Gateway Timeout
```

Error Status in HW#5

- In "server/routers/guess.js"

```
if (!guessed || guessed < 1 || guessed > 100) {
  res.status(400).send({ msg: 'Not a legal number.' })
}
```

- In "src/axios.js"
  - 需要判斷 client side 是否有正確接到回傳訊息
  - If not, how to catch it?
  - 有需要在 client side 就做好 input 的檢查嗎？
    If so, server side 還需要檢查嗎？

HTTP Header

- 用戶端送出 API 請求時，要帶一些 http header 讓 server 端了解目的
  - Authorization: 認證資訊
  - Accept: 能夠接受的回應內容類型 (Content-Type)，屬於內容協商的一環

```
# 用 Accept 跟 Server 溝通格式
接收 純文字 -> Accept: text/plain
接收 HTML  -> Accept: text/html
接收 JSON  -> Accept: application/json
```

- 至於 API 回傳結果的 HTTP header，沒甚麼特別之處，按照一般原則處理即可（例如Content-Type、Content-Length、ETag、Cache-Control…）

HTTP Body

- JSON, 純文字，或 XML 格式
- 善用一些 node.js 套件，例如：body-parser

【RESTful 是風格，不是規範】

可以適當放寬設計 (或稱之為：RESTish)

避免工程師的鄙視鏈

到目前為止，server 的資料都只是存在程式裡，

如果遇到 server 當機之類的情況，資料就全不見了。

因此，大部分正式的 server 實作，

都會把資料存在「資料庫」（Databse）裡。

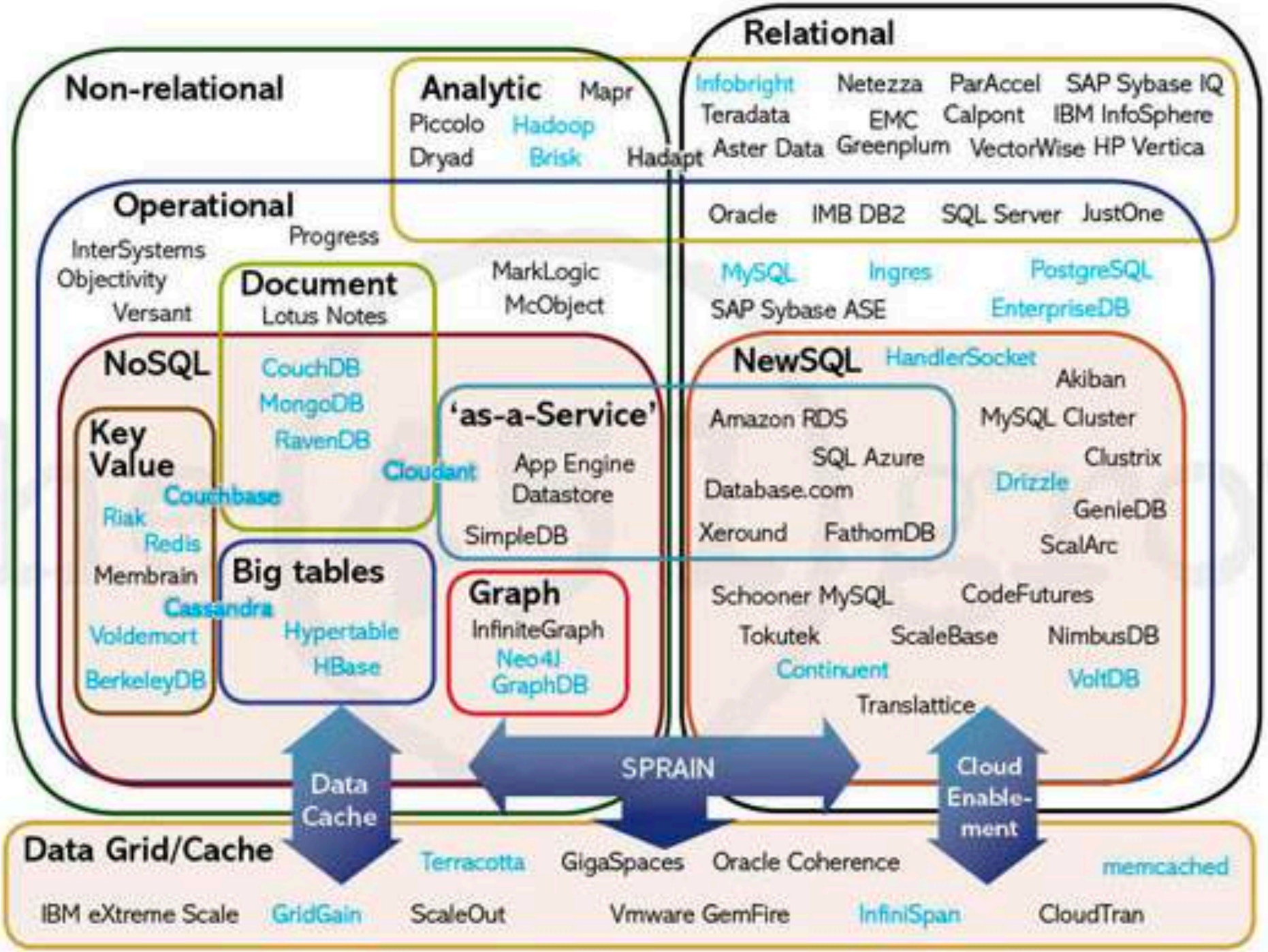
What is a "Database"?

- Database? Data Structure? File?
- What is database? What is a DataBase Management System (DBMS)?

Think...

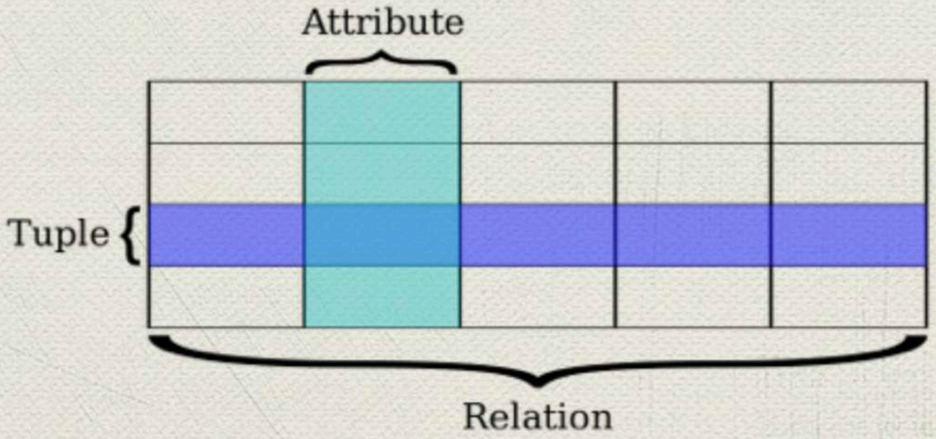
- 資料如何儲存在電腦裡面？檔案？記憶體？
- 如何找到特定一筆資料？如何找到符合特定條件的資料？如何統計符合特定條件的資料？
- 如何讓多人的資料可以被多工的操作？如何保證不會有讀寫覆蓋、資料不一致、或者是資料遺失等問題？
- 如何保證資料不會被竊取、竄改？

Types of Database



Relational Database Management System (RDBMS)

- Relational? 關聯式資料庫?
- 比方說，一個 Excel Table
- RDBMS 架構
  - Server（伺服器）
  - Database（資料庫）
  - Table（資料表）
  - Column（欄位） // Attribute
- 每個 row (tuple) 代表一筆資料
  - Query 符合某個欄位的某些條件



SQL: **S**tructured **Q**uery **L**anguage

- 通俗唸法：See-Quel
- 用來 query structured database (通常指 Relational DB) 的一種語言，但後來人們會用 SQL Database 來指 relational DB
- 常見的 SQL DB 包含：
  - MySQL: 早期 open, 但後來被 Oracle 收購
  - MS SQL: by Microsoft
  - PostgreSQL: open source
  - IBM DB2
  - SQLite: free/Lite software
  - MariaDB: MySQL founder 後來出來復刻的 free 版本

不過，時間關係，我們今天
先無法仔細講 RDB/SQL
我們先來介紹 HW#6 會用
到的 NoSQL/MongoDB

TO SQL or NOSQL,
That is the question!

【NOSQL】

(Not only SQL)

非關係型、分散式、不提供 ACID 的

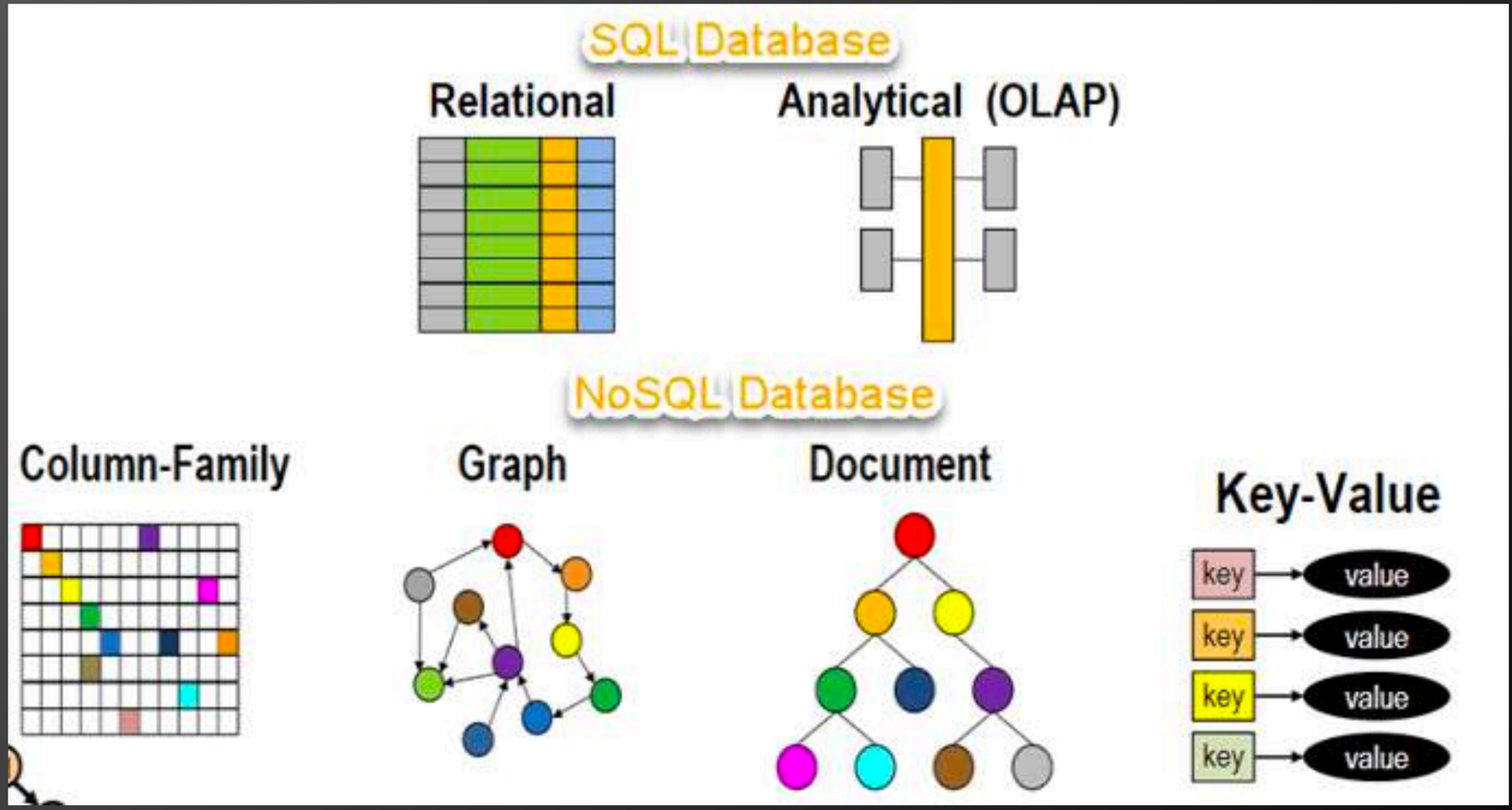
資料庫設計模式

資料庫設計的 ACID 原則

- 為了確保資料庫 transactions 的正確性，在設計資料庫系統 (DBMS) 時會希望保有以下 ACID 原則：

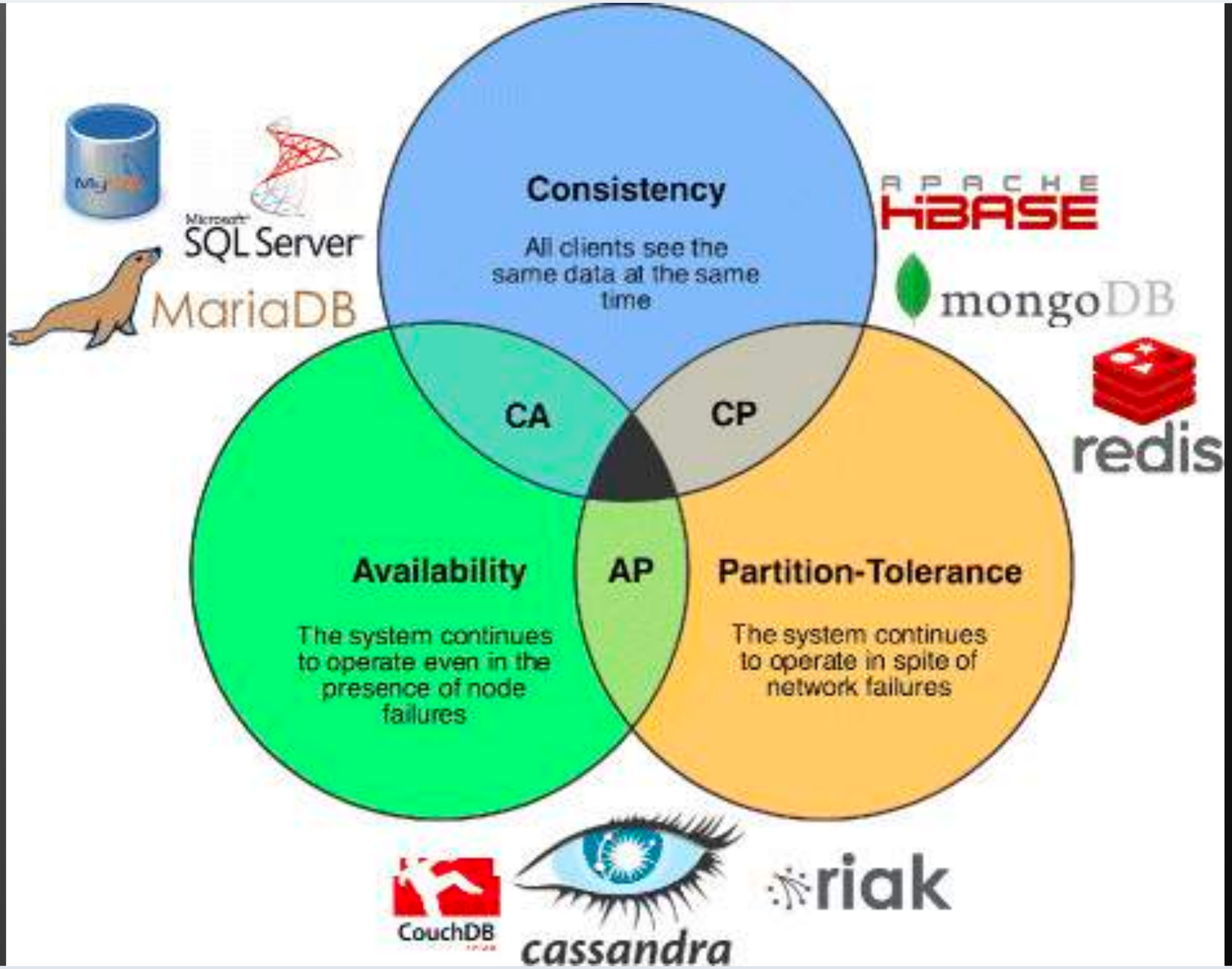
1. Atomicity（原子性）：資料庫的 transactions 不會被斷在完成一半的階段，如果 error, 應該要 roll back 回去前一個動作
2. Consistency（一致性）：任何的 transactions 要維持資料庫的完整性與一致性，不正確的存取不應該造成資料庫的破壞
3. Isolation（隔離性）：允許多個 transactions 可以同時 (concurrent) 進行且不互相干擾，其結果應與其循序執行相同
4. Durability（持久性）：Transactions 結束後，對資料的修改就是永久的，即便系統故障也不會丟失。

SQL or NOSQL

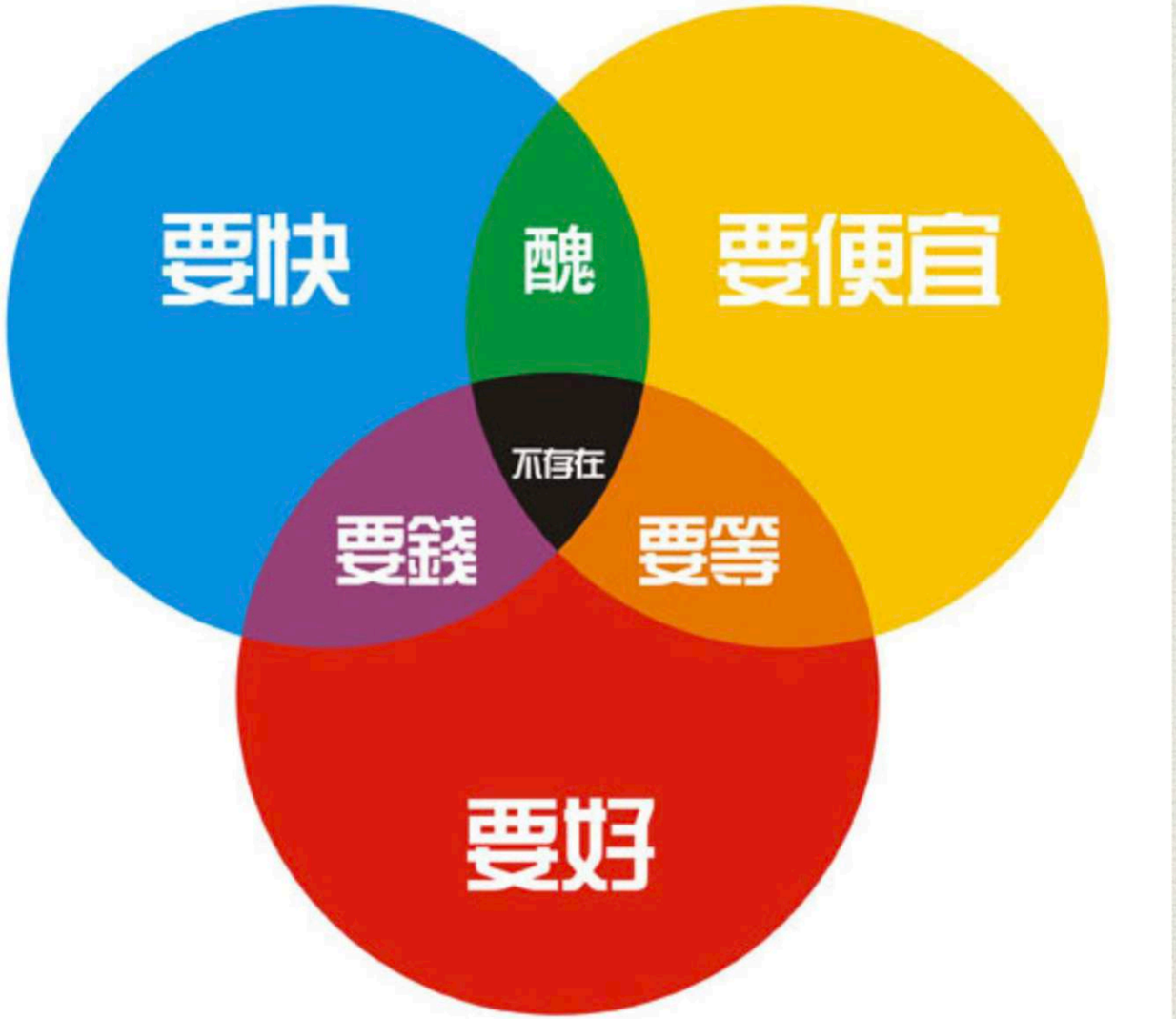

SQL Database
Relational
Analytical (OLAP)
NoSQL Database
Column-Family
Graph
Document
Key-Value
key
value
key
value
key
value
key
value

NOSQL 那麼多種，要怎麼選？

- Brewer's CAP Theorem
  - Consistency 一致性
  - Availability 可用性
  - Partition tolerance 分區容忍
  - 根據定理，分布式系統只能滿足三項中的兩項而不可能滿足全部三項。理解CAP理論的最簡單方式是想像兩個節點分處分區兩側。允許至少一個節點更新狀態會導致數據不一致，即喪失了C性質。如果為了保證數據一致性，將分區一側的節點設置為不可用，那麼又喪失了A性質。除非兩個節點可以互相通信，才能既保證C又保證A，這又會導致喪失P性質。

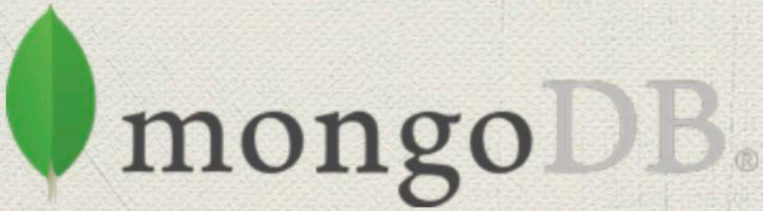

Consistency
All clients see the same data at the same time
Availability
The system continues to operate even in the presence of node failures
Partition-Tolerance
The system continues to operate in spite of network failures
CA
CP
AP
SQL Server
MariaDB
APACHE HBASE
mongoDB
redis
CouchDB
cassandra
riak

讓人想到這張經典圖...



MongoDB

- MongoDB 是 10gen 這家公司開發的一個 NoSQL Database，屬於 Document-Oriented Database 這一類型，希望能夠結合 Relational Database 與 Key/Value Database 雙方的優點，很適合用在 Web 應用程式、Internet 架構的環境底下。



Ref: http://www.codedata.com.tw/database/mongodb-tutorial-1-setting-up-cloud-env

MongoDB Terminologies

RDBMS	MongoDB
Database	Database
Table	Collection
Record/Row	Document
Column	Field
Primary Key	_id

Ref: http://www.codedata.com.tw/database/mongodb-tutorial-1-setting-up-cloud-env

MongoDB Document Format

- 寫法看起來就跟 JSON 一樣。_id 欄位就是 Primary Key，其他都是一般的欄位。

```
{
    "_id": "10280",
    "city": "NEW YORK",
    "state": "NY",
    "pop": 5574,
    "loc": [-74.016323, 40.710537]
}
```

Ref: http://www.codedata.com.tw/database/mongodb-tutorial-1-setting-up-cloud-env

MongoDB Query Example

- 查詢 OA 出版社出版的所有書籍資料db.books.find({pubId: “OA"});
- 查詢所有定價超過 50 元美金的書籍資料db.books.find({listPrice: {$gte: 50}});
- 查詢 OA 出版社出版的所有書籍中，書名出現過 Java 的書籍資料
  db.books.find({pubId: "OA", title: /.*Java.*/g});
- 查詢 OA 或 PH 出版社出版的所有書籍資料
  db.books.find({$or: [{pubId: "OA"}, {pubId: "PH"}]}); 或
  db.books.find({pubId: {$in: ["OA", “PH"]}});
- Query 預設會傳回所有欄位，如果要查詢 OA 出版社出版的所有書籍，只需要書名與定價資料
  db.books.find({pubId: "OA"}, {title:1, listPrice: 1, _id: 0});

Ref: http://www.codedata.com.tw/database/mongodb-tutorial-2-query-language/

MongoDB Query Example

- 如果要查詢所有書籍，不想要出版日期 db.books.find(null, {releaseDate: 0});
- 搭配 sort 與 limit、skip 等方法，可以控制輸出的順序 — 如果要查詢所有書籍，只需要書名與定價資料，根據定價由小排到大或由大排到小 db.books.find(null, {title:1, listPrice: 1, _id: 0}).sort({listPrice: 1});
- 如果要查詢所有書籍，根據定價由小排到大，只要前兩筆 db.books.find().sort({listPrice: 1}).limit(2);
- 透過 count 之類的方法，可以輕易達到彙總的功能 — 如果要查詢 OA 出版社出版的所有書籍總數 db.books.count({pubId: "OA"});

Ref: http://www.codedata.com.tw/database/mongodb-tutorial-2-query-language/

其他應該要知道的 NoSQL

- **Redis**: in-memory 的 key-value database，因此常常被用在需要快取（Cache）一些資料的場合，加快資料存取速度。
- **Riak**: Basho公司基於Amazon的Dynamo理論推廣開發的 key-value distributed database. Based on Erlang, 一開始就支持分布式、容错應用程序。没有主節點的概念，因此在處理故障方面有更好的彈性。
- **Cassandra**: 一套開源分散式NoSQL資料庫系統。它最初由Facebook開發，用於儲存收件箱等簡單格式資料，集Google BigTable的資料模型與Amazon Dynamo的完全分散式架構於一身。Facebook於2008將Cassandra 開源，此後，由於Cassandra良好的可延伸性和效能，被許多知名網站所採用，成為了一種流行的分散式結構化資料儲存方案。

如何在 Web Program 中使用 DB?

- **ORM (Object Relational Mapping)**
  - 理想上，在寫 web program 的時候，我們希望 focus on web 操作的邏輯與資料流，不想要去管 DB 如何儲存、以及如何使用等語法細節。
  - ORM 概念上像是創建了一個可在編程語言裡使用的「虛擬物件資料庫」，讓DB的存取與查詢都可以用比較高階、物件化的方式進行
  - 只不過，現實上，DB 的種類太多，在 Web Programming 技術不斷推陳出新的情況下，似乎沒有人有那個力氣/時間打造出一個通用、大家都認可、願意使用的 high-level ORM

02. Mongoose

開啟另外一個 project 來測試

1. Create project directory (e.g. 02.mongoose)

```
mkdir 02.mongoose && cd 02.mongoose
```

2. Create a “package.json” for this project

```
yarn init -y
```

3. Add these lines to “package.json”

```
  "scripts": {
    "start": "cd frontend && yarn start",
    "server": "cd backend && yarn server"
  }
```

Let’s prepare a simple DB service

1. Create directory for backend

```
mkdir backend
```

2. Create a “package.json” for backend

```
cd backend
yarn init -y
```

3. Install “mongoose” locally

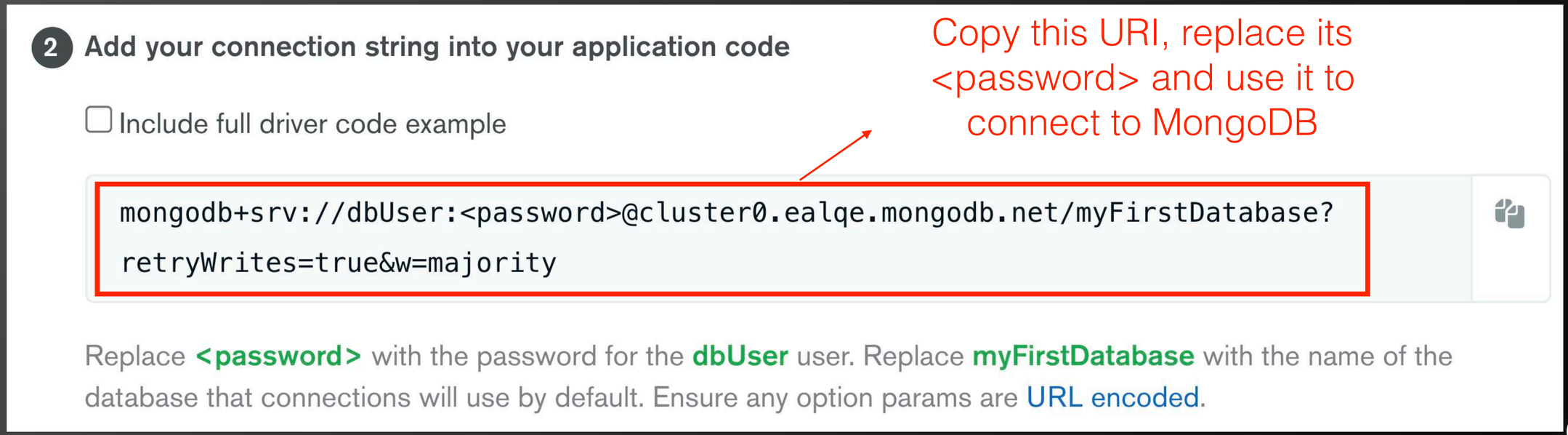
```
yarn add mongoose --save
```

How and Where to Serve a MongoDB?

- 理論上一個網路服務應該要把伺服器程式佈建到一個使用者可以透過 URL access 得到的機器，然後把 DB 放在 server 可以連線得到、持續開機的機器/硬碟裡
- 然而，我們在開發的時候，為了方便 debug 起見，會把 server 與 client 都先放在 localhost, 而 DB 就存放在 memory or 一個臨時的檔案裡，但這樣就會產生一個問題，當 server 重啟的時候，DB 的資料還會在嗎？
    - 建議找一個雲端(免費)的 DB service 來存放資料

MongoDB Atlas

- MongoDB 的官方提供了一個免費的服務：Atlas
- Sign up 之後，會被要求創立一個 organization, 然後建立一個 project, 最後建立一個 cluster (i.e. a MongoDB).
- Cloud service 可選擇 Google Cloud (因為他們在台灣有 node)
- 選擇 "Connect your application", you will see:



Let's prepare a simple DB service

4. Create and edit "server.js"

```
import mongoose from 'mongoose';
mongoose
  .connect('mongodb://...', {
    useNewUrlParser: true,
    useUnifiedTopology: true,
  })
  .then((res) => console.log("mongo db connection created"));
```

把剛剛在 Mongoose Atlas copy 的 URI 貼在這裡，改掉 <password>

5. Add these lines to "package.json"

```
  "type": "module",
  "scripts": {
    "server": "nodemon server.js"
  },
```

把 MongoDB URI/password 放在 code 裡？

- 你不會想把 URI/Password 放在 code 裡的！
- (Solution) 把他們放在一個檔案裡 (.env)，然後用 "dotenv" 這個套件去把他們讀進來，存到 "process.env.MONGO_URL"

新增一個 ".env" 檔案，把 URI/Password copy 在這裡 —

```
MONGO_URL=mongodb+srv://
dbUser:[redacted]@cluster0.ealq
e.mongodb.net/myFirstDatabase?
retryWrites=true&w=majority
```

In “server.js”, 把 ‘mongoldb://…" 改成 —

```
mongoose.connect(
  process.env.MONGO_URL, {
  useNewUrlParser: true,
  useUnifiedTopology: true
});
```

把 MongoDB URI 放在 code 裡？

- 但你也不會想要把 .env check in 進去，所以我們會把它加入 .gitignore 裡頭去
- 但問題是，別人 pull 下來後是沒有 .env 檔的，這樣他要怎麼才會知道要設定哪些環境變數呢？
- 使用 "dotenv-defaults" instead of "dotenv"

```
yarn add dotenv-defaults --save
```

- 增加一個 .env.defaults 檔案，裏頭放一個空的環境變數

```
MONGO_URL=
```

- 在 server.js 加上：

```
import dotenv from "dotenv-defaults";
dotenv.config();
```

- 執行的時候 dotenv-defaults 會去找 .env 來讀取環境變數資訊

Test the project...

Under the project directory —

- Start server

```
yarn server
```

- If it complains about some module missing, install it.

What do you see?

How to write data to DB?

Define MongoDB Schema

- 如前所述，MongoDB 的儲存格式就像是 JSON format 一樣，因此，在 JS 裡頭就用 object 的格式來定義即可
- Create a file “models/user.js”

```
import mongoose from 'mongoose'
import User from 'models/user.js'

const Schema = mongoose.Schema;
const UserSchema = new Schema({
  id: Number,    // Number is shorthand for {type: Number}
  name: String
});
const User = mongoose.model('User', UserSchema);

export default User;
```

Modify “server.js”

- 新增底下兩個 DB mutation functions

```
const saveUser = async (id, name) => {
  const existing = await User.findOne({ name }); { name: name }
  if (existing) throw new Error(`data ${name} exists!!`);
  try {
    const newUser = new User({ id, name });
    console.log("Created user", newUser);
    return newUser.save();
  } catch (e) { throw new Error("User creation error: " + e); }
};

const deleteDB = async () => {
  try {
    await User.deleteMany({});
    console.log("Database deleted");
  } catch (e) { throw new Error("Database deletion failed"); }
};
```

Filter 條件. Same as

Modify “server.js”

- To operate DB through Mongoose APIs (ref):

```
const db = mongoose.connection;
db.on("error", (err) => console.log(err));
db.once("open", async () => {
  await deleteDB();
  await saveUser(57, "Ric");
  await saveUser(108, "Sandy");
  await saveUser(77, "Peter");
});
```

Restart DB. What do you see?

Check on Mongoose Atlas!!

Now you did:

01. Express + HTTP

02. Mongoose

And we did in HW#5:

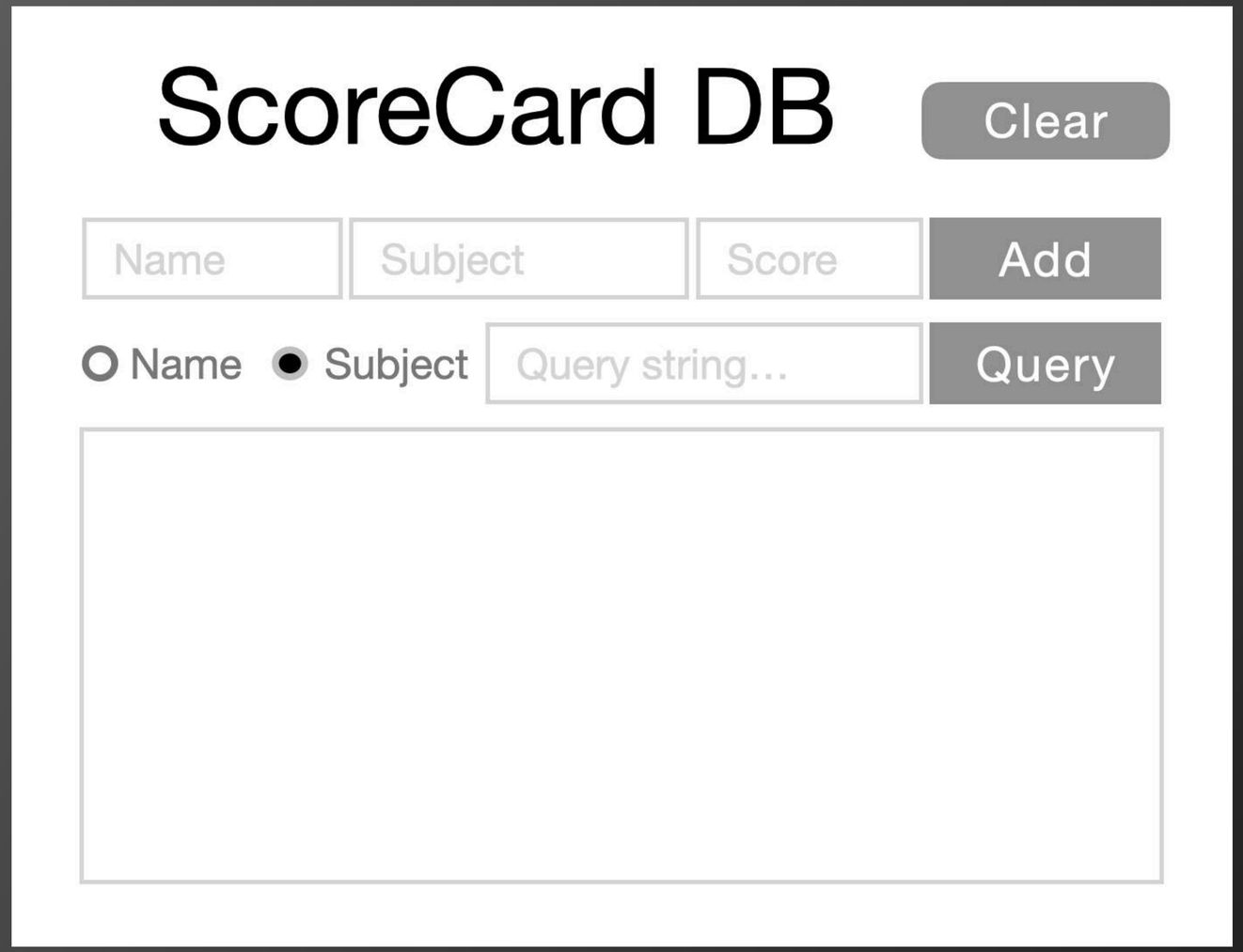
React + Axios + Express

Let's put them together as HW#6!!

HW#6 說明

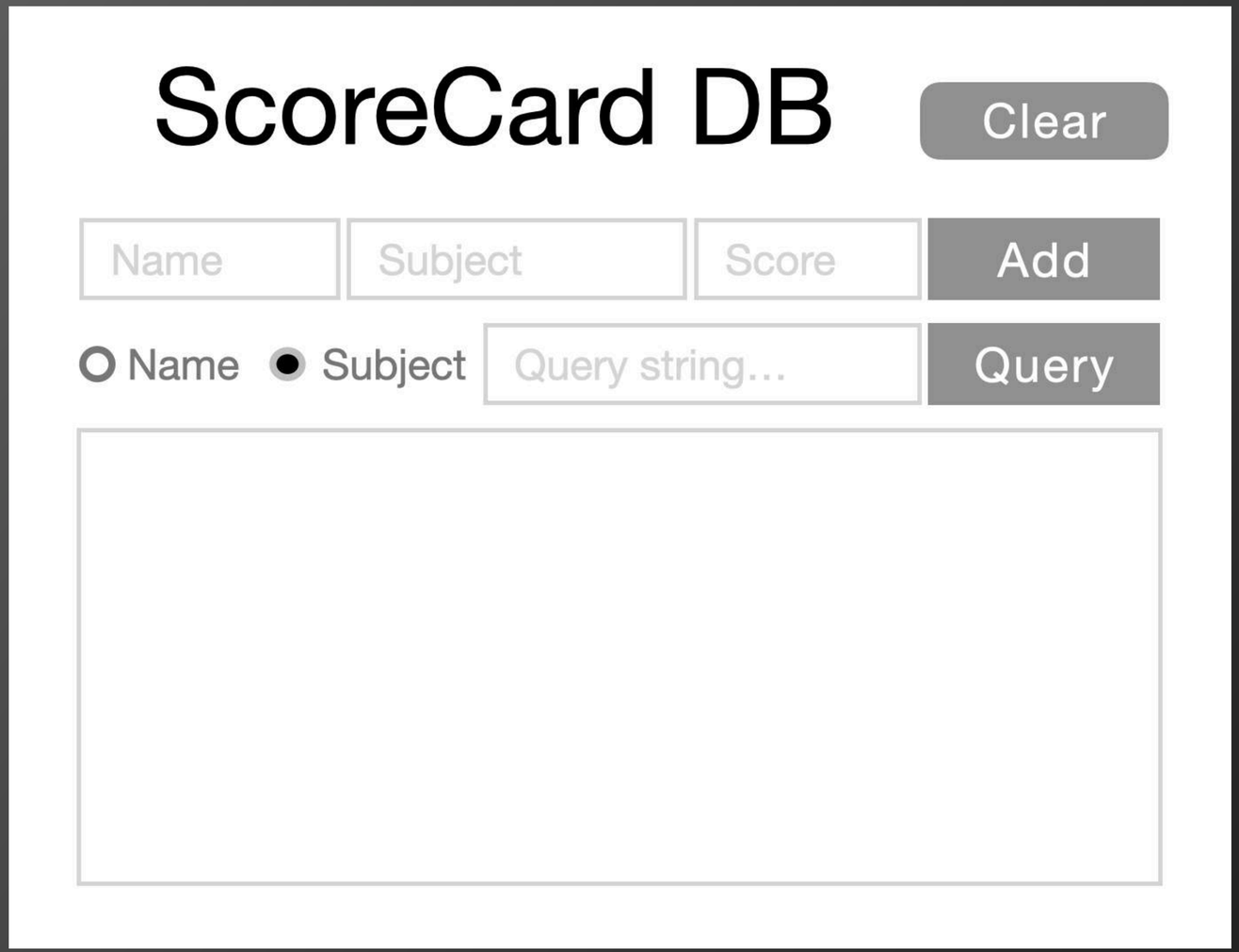
- 目標：從 frontend React APP 透過簡單 UI 輸入 { id, name }, 用 Axios 傳給 backend Express, 然後寫入 MongoDB
- Getting Started:
  - 在 wp1092 底下，創建 hw6 目錄
  - 在 hw6 底下，創建 own 目錄
  - 利用 "create-react-app" 建立 “frontend” 目錄
  - 利用前面講義的說明，建立包含 express 以及 mongoose (with User schema) 的 backend

Manipulate User DB thru React APP



- 如左圖，當 "Clear" 按鈕被按下去後，會送出 DELETE request 把 DB 清空，然後在下方空白處印出 "Database cleared"。
- 當 "Name", "Subject" 以及 "Score" 三個輸入匡都被輸入值之後，按下 "Add" 按鈕，會把資料寫到 DB
  - 如果 {Name, Subject} paired value 已經存在 DB，則取代原先在 DB 的這筆資料，然後在下方空白處印出 "Updating (Name, Subject, Score)”
  - 否則，在 DB 新增一筆資料，印出 "Adding (Name, Subject, Score)"

Manipulate User DB thru React APP



- 當 Query string 不為空字串的情況下，當 Query 按鈕按下去後，會根據 Name/Subject 二選一的 radio button 選擇 Name or Subject 作為 query 條件，然後根據輸入 Query String 的內容去 DB 把符合條件的資料搜尋回來，在下方空白處印出搜尋出來的結果 (一筆一行，格式不限)。如果找不到，就印出 "QueryType (QueryString) not found!"

感謝聆聽！

Ric Huang / NTUEE

(EE 3035) Web Programming