



215

215



G+



Swift 網路程 式設計指南：

Swift 網路程式設計指南：如何使用 Alamofire

打從 2014 年 6 月發佈 Swift 語言以來，要在 Swift 中實現網路功能就有點麻煩。即便是 Swift 之父 Chris Lattner 也在 Twitter 發文表示，要解碼 JSON 還有「一段路要走」。因此許多人都在尋覓替代方案。當然也有內建的類別能夠處理基本的 JSON 剖析，但是對於開發者而言，總覺得沒有那麼友善。幸好有 Alamofire 的存在。同樣是由 AFNetworking（用過 Objective-C 的開發者大概都不陌生）的創作者所建立，Alamofire 在解碼 JSON 方面是非常好用的程式庫。

本文是一篇以假期為背景的文章，在本文中，我們將會大量探索與基礎網路程式設計有關的主題，並且建構出一個簡單 TODO App。

simple, elegant and designed forever.



forev

在本文中，你會看到如何使用與解碼 JSON、撰寫自訂的伺服器、運用 Heroku 和 MongoLab 之類的工具、理解 HTTP 方法（包括 GET、POST 和 DELETE）的運作、git、終端機，以及操作 Cocoapods。假使你覺得聽起來有點太過豐富了，泡杯咖啡且聽我繼續講下去吧。

注意：本文屬於進階文章，而且內容包羅萬象。我假設你對於 iOS 和 Swift 有非常深刻的理解。諸如表格視圖、自動佈局、委派等議題將不會深入解釋。假使你對於這些主題並不熟悉，可以回過頭去複習我們的課程，然後再返回來重拾本文。

開始動手吧

為了能夠順利運作，我用很流行的伺服器端技術 Node.js 開發了一個後台（Backend）。如果不認識 Node.js 的話，容我告訴你，它是基於 JavaScript 的執行階段環境，運用於 Google Chrome 的 V8 引擎當中。長話短說，Node.js 非常可靠、快速，而且威力強大。

針對此後台，我也一併建置了 Restify 並且使用 MongoDB。MongoDB 是一種非 SQL 資料庫，在網頁開發者之間非常流行。透過 Mongo，我們可以將相關的內容全部都儲存到資料庫當中。

當我開始深入學習 Node.js 的時候，我其實並不確定每件事情如何運作，我所造訪的許多部落格都沒有解釋背後的原理。因此，雖然我們是一個 iOS 部落格，但是我仍然會向你介紹 JavaScript，並且說明 Node.js 伺服器的運作原理。

我已經打造好網站了，獨缺從建立 API 到與 iOS App 互動的詳細逐步教學文章。但是再也不會這樣了。現在就讓我們進入網路程式設計的世界吧...

來認識一下 Node.js 吧

如前所述，Node.js 是功能非常齊全的伺服器技術，建構在 Chrome 的執行環境之上。因此具有高度非同步與非阻斷（假使你不確定這是什麼意思，簡單地說，就是主執行緒或是 App 的主要部分不會凍結卡住）的特性。多執行緒是一種避免延遲並且增加程式效率的程式設計技術。將 App 想像成高速公路。如果只有一條線道，卻有 20 台車需要通行，那麼很可能就會塞車。但是如果 3 條線道的話，而且每條都有出口和入口，那麼車流量將會降低許多。多執行緒有點這種意味。在多執行緒的環境中，程式碼是在不同的執行緒中執行，可以避免 App 壓塞，也就可以避免 App 凍結卡住了。



Node.js 是由 Joyent 開發，這是一家位於舊金山的雲端運算公司，這家公司同時也是 Node.js 的主要維護者。

假使你依舊不確定事情的運作方式，不妨用如下的方式來思考此後台：

1. 此後台可以路由你的 API（我們即將為範例 App 打造 API，就跟網路上許多其他的 API 一樣，包括我們在拙著的 [tvOS 文章](#) 當中所使用過的非常流行的 [forecast.io API](#) 一樣）。
2. MongoDB 能夠讓我們儲存所有的資料。當我們想要發佈新的訊息時，我們需要有個地方儲存該訊息。在這種情況之下，就會將東西儲存到我們的 Mongo 資料庫。

- 我們將會建構功能完整的 REST API，將會遵循 REST 協定，也就是網頁的運作方式。

我們的 MongoDB 是架在 MongoLab，而 Node.js 伺服器則是放在 Heroku。Heroku 是 Salesforce 的子公司，是架設 Node.js、Rails、Python 等服務的流行主機，而 MongoLab 則非常適合架設 Mongo 資料庫。



介紹 HTTP 動詞

在開始動手寫程式之前，你務必要瞭解 HTTP 動詞，以及如何在我們的 App 當中加以運用。

GET – GET 動詞會查詢我們的資料庫並擷取內容。GET 可以只取得一筆、多筆或是所有的資料。事實上，每次當你來到 google.com 或是瀏覽 Facebook/Twitter 的時候，都是在執行 GET 要求（只是你可能不自知而已）！

POST – POST 動詞會傳送資料到伺服器，然後儲存起來。舉例而言，當你在撰寫一篇新的 Facebook 或 Twitter 貼文然後按下發佈的按鈕時，便是在建立新的內容，而且使用的是 POST 動詞。

UPDATE – UPDATE 動詞可以讓你修改內容。當你在修改 Facebook 貼文時，使用的就是 UPDATE 動詞。

DELETE – 如同其名，DELETE 動詞可以用來刪除內容。當你針對 Facebook 或 Twitter 貼文按下刪除鈕時，便是呼叫此動詞。

這 4 個動詞就是 REST 通訊協定的基礎，同時也是網際網路運作的原理。或許你也曾經聽過 CRUD 縮寫字和這些動詞交互出現。CRUD 的 C 是建立（Create）、R 是讀取（Read）、U 是更新（Update）、D 是刪除（Delete）。如果仍看不出蛛絲馬跡的話，容我告訴

你，它們分別對應到 POST 、 GET 、 UPDATE 和 DELETE 。

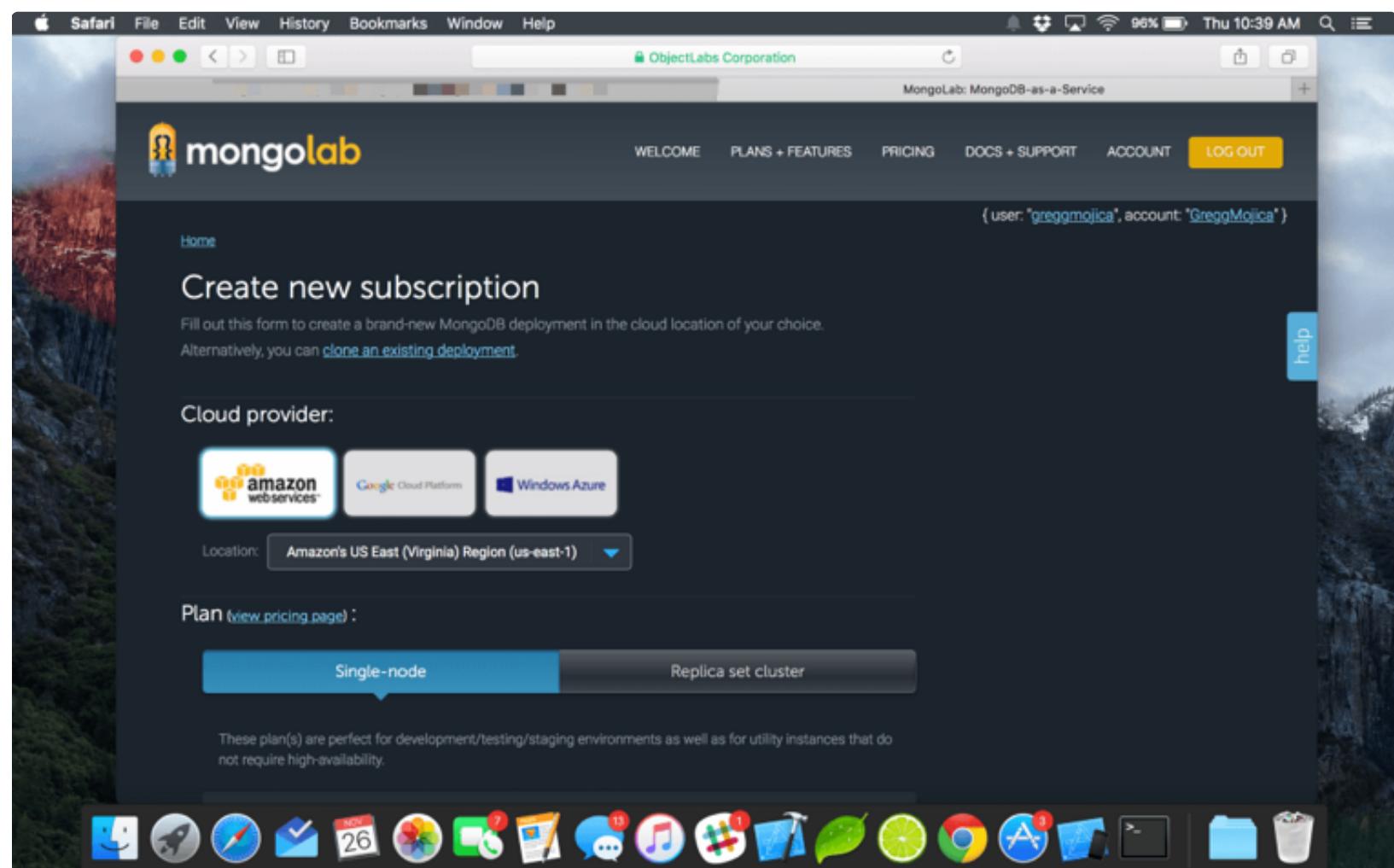
太棒了！現在我們已經對於 HTTP 通訊協定有了深刻的理解，應該已經準備好可以進入本文的教學之旅了。

設定必要的工具

在談論 MongoLab 或 Heroku 之前，請先確定你已經設定好 Node.js 了。點擊此連結，閱讀簡短的指南，然後將 Node 下載到你的電腦上。

接著，參考 npm 網站的此網頁來下載 npm 。

為了將後台設定好，我們需要在 Heroku 和 MongoLab 都擁有帳號。首先從 MongoLab 開始。點擊此連結來到 MongoLab 的網站並且建立新帳號。



建立帳號之後，回到主頁，按 Create DB 建立新 database 。

別忘了選擇 Single-node (免費方案) ，當提示視窗跳出時，填入資料庫的名稱。我將自己的資料庫取名為 **alamofire-db** (後綴 db 代表資料庫，這是常見的命名慣例，但你可能須要選

擇其他資料庫名字）。最後，按 Create new MongoDB deployment 繼續。

The screenshot shows the MongoLab MongoDB-as-a-Service interface. At the top, there are two sections for selecting a plan: 'M5 Single-node' (\$1310) and 'M6 Single-node' (\$2045). Below this, the 'High Storage Line' section lists four plans: M3 Single-node (\$500), M4 Single-node (\$1000), M5 Single-node (\$1545), and M6 Single-node (\$2180). A note states: 'Plans which offer a higher storage-to-RAM ratio than those in the Standard line and are geared towards applications that need to store large amounts of data but have more modest performance requirements. Plans come standard with 2 data nodes plus an arbiter node.' At the bottom left, it says 'MongoDB version: 3.0.x'. In the center, there's a form to enter a 'Database name:' with 'alamofire-db' typed in. To the right of the database name input is a 'Price:' section showing '\$1545 / month'. Below the price are several small icons representing different services or tools.

接著登入你的資料庫，找到 mongo URI 。

The screenshot shows the 'Database: alamofire-db' details page. It includes instructions for connecting using the mongo shell and a standard MongoDB URI. The standard MongoDB URI is highlighted with a yellow box: 'mongodb://<dbuser>:<dbpassword>@ds057954.mongolab.com:57954/alamofire-db'. Below this, it says 'mongod version: 3.0.7'. At the bottom, there are tabs for 'Collections', 'Users', 'Stats', 'Backups', and 'Tools'. The 'Collections' tab is active. There is also a 'Collections' button and an 'Add collection' button.

你接著要做的是建立新的使用者，請按下 Users 分頁標籤，然後選擇帳號和密碼。要把密碼記好喔。

The screenshot shows the MongoDB Compass interface. At the top, there are instructions for connecting via the mongo shell and a driver. Below that is a navigation bar with 'Collections', 'Users' (which is selected), and 'Stats'. A sidebar on the left lists 'Database Users' with one entry: 'gregg'. In the center, a modal window titled 'Add new database user' is open, prompting for a database username, password, and confirmation. There's also a checkbox for 'Make read-only'. At the bottom of the modal are 'Cancel' and 'Create' buttons. The top right corner of the main window shows 'mongod version: 3.0' and a 'help' link.

現在回到 mongo URI 的那個頁面，換上剛才建立的新帳號。例如：

`mongodb://<dbuser>:<dbpassword>@ds057954.mongolab.com:57954/alamofire`

置換成：

`mongodb://gregg:test@ds057954.mongolab.com:57954/alamofire-db`

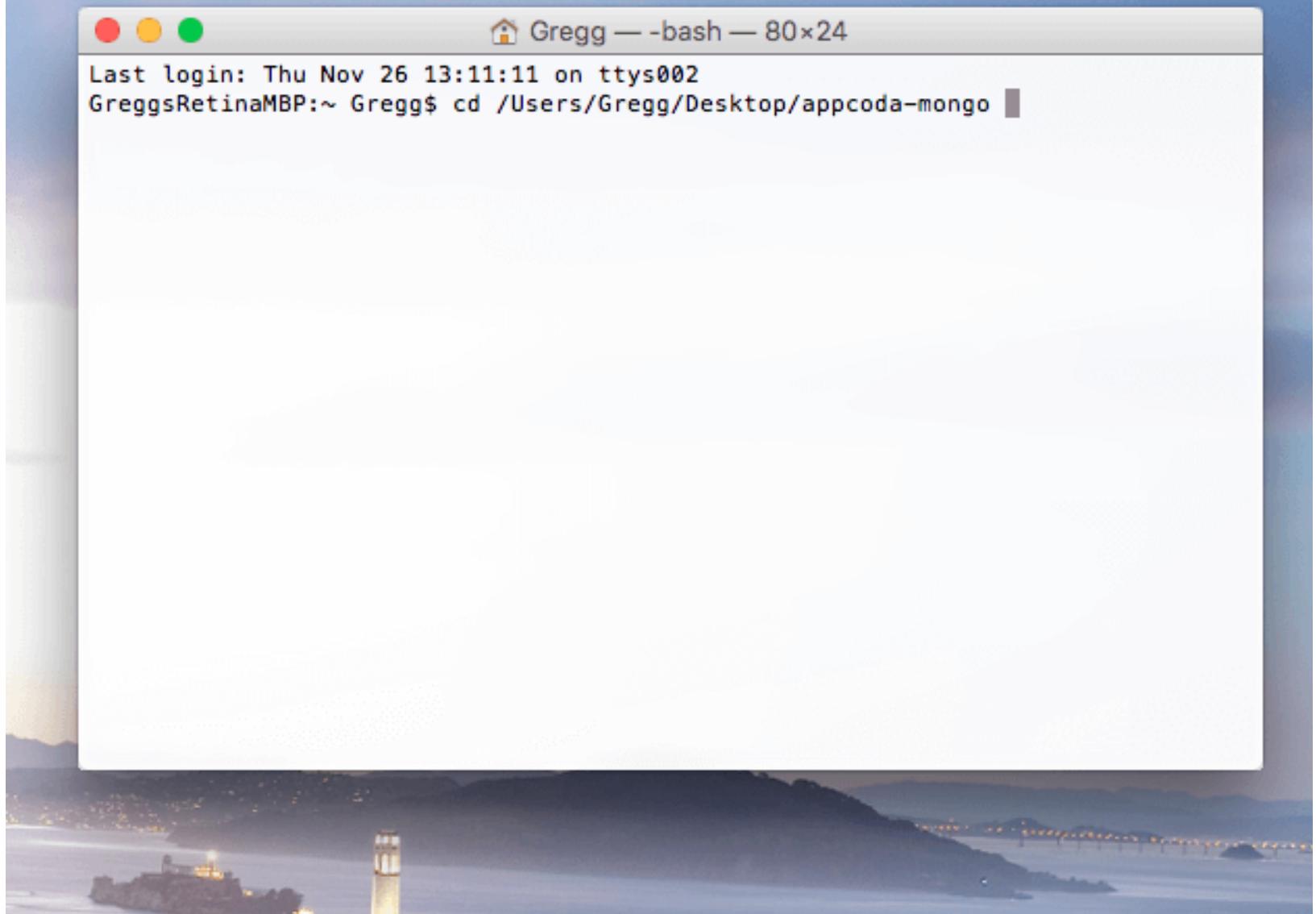
我們又往前邁進一步了！

現在來到 [Heroku.com](#)，註冊免費的帳號，做完之後再回來繼續閱讀本文。

點擊此連結以便在你的電腦上安裝 Heroku Toolbelt。Heroku Toolbelt 是一個Heroku 命令行界面工具，用來建立和管理 Heroku apps。

在安裝完畢之後，打開終端機，鍵入heroku 並以你的 Heroku 帳號登入。假使你從未使用過終端機，請別擔心。我們在本文中會很常使用到，所以看完本文之後，你會對終端機非常熟悉。

從終端機登入到 heroku 之後，使用 cd 命令（代表切換目錄）移動到你從 [GitHub](#) 下載的專案資料夾。cd 等同於在 Finder 中切換資料夾。



按下 Enter 鍵以便執行此命令。太棒了。現在我們將使用 **git** 來 **push** 東西到 heroku 。

在終端機中輸入下列的命令。

```
git init  
git add .  
git commit -m "First Commit"
```

這 3 行命令將會初始化儲存庫（ Repository ，簡稱 repo ） ，然後將檔案加入到儲存庫中，最後提交（ Commit ）或稱儲存。

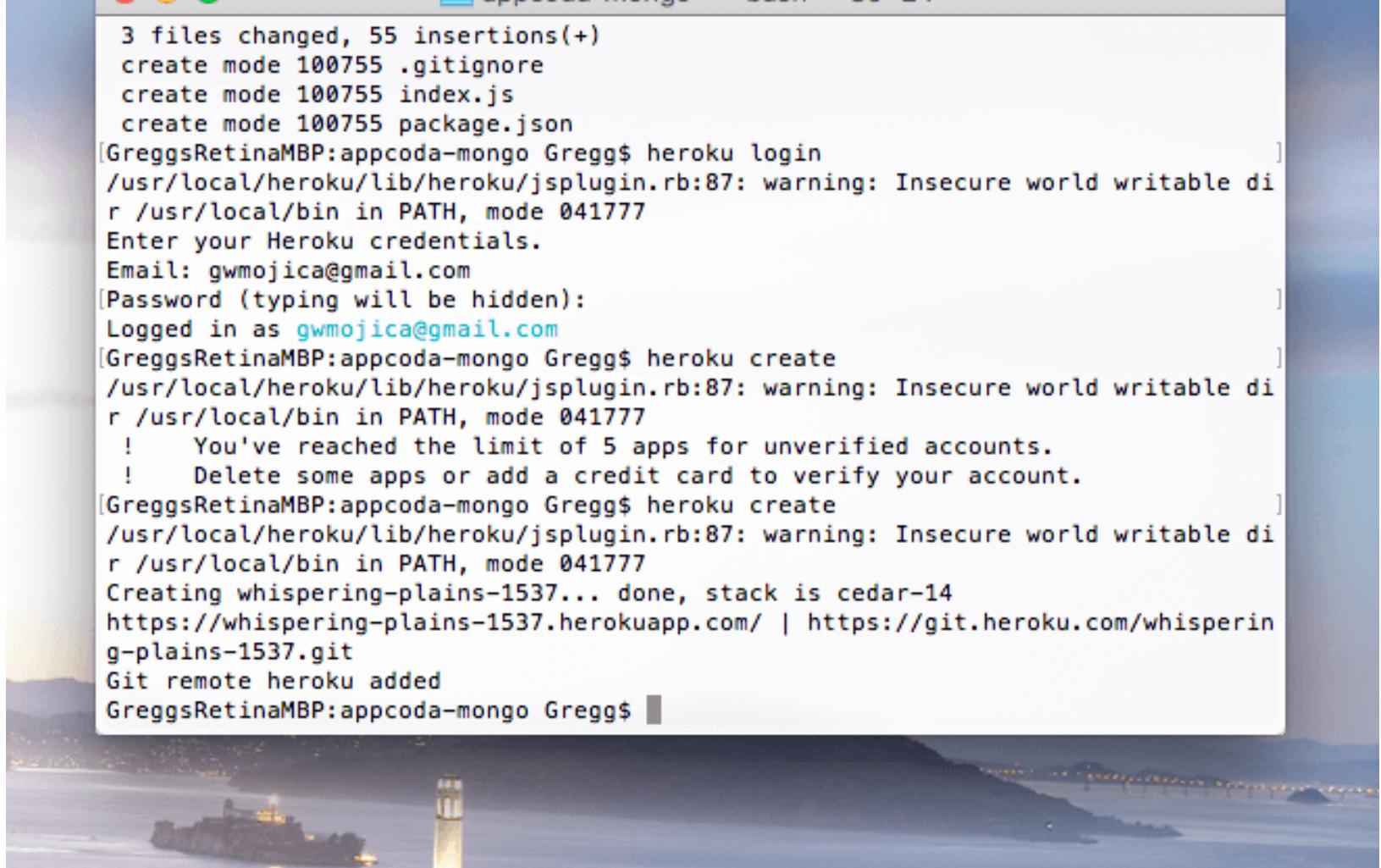
git 是非常流行的版本控制軟體，常見於許多工作流程當中。

現在你應該可以看到如下圖所示的畫面：

```
Last login: Thu Nov 26 13:11:11 on ttys002
[GreggsRetinaMBP:~ Gregg$ cd /Users/Gregg/Desktop/appcoda-mongo
GreggsRetinaMBP:appcoda-mongo Gregg$ git init
Initialized empty Git repository in /Users/Gregg/Desktop/appcoda-mongo/.git/
GreggsRetinaMBP:appcoda-mongo Gregg$ git add .
[GreggsRetinaMBP:appcoda-mongo Gregg$ git commit -m "First Commit"
[master (root-commit) a60dc7d] First Commit
 3 files changed, 55 insertions(+)
 create mode 100755 .gitignore
 create mode 100755 index.js
 create mode 100755 package.json
GreggsRetinaMBP:appcoda-mongo Gregg$ git log
```

因為安裝了 heroku toolbelt，因此可以以下 **heroku login** 命令並且輸入你的 heroku 登入帳密（也就是電子郵件和密碼）。在輸入密碼的時候，密碼會隱藏起來看不見。按下 Enter 以便繼續，如果一切正確無誤的話，會在最後面看到自己的電子郵件以醒目方式呈現。

現在輸入 **heroku create** 以便建立新的 Heroku App。Heroku 會以網域方式幫你建立新的 App。舉例而言，我的 App 是 <https://whispering-plains-1537.herokuapp.com/>。



```
appcoda-mongo — bash — 80x24
3 files changed, 55 insertions(+)
create mode 100755 .gitignore
create mode 100755 index.js
create mode 100755 package.json
[GreggsRetinaMBP:appcoda-mongo Gregg$ heroku login
/usr/local/heroku/lib/heroku/jsplugin.rb:87: warning: Insecure world writable di
r /usr/local/bin in PATH, mode 041777
Enter your Heroku credentials.
Email: gwmojica@gmail.com
>Password (typing will be hidden):
Logged in as gwmojica@gmail.com
[GreggsRetinaMBP:appcoda-mongo Gregg$ heroku create
/usr/local/heroku/lib/heroku/jsplugin.rb:87: warning: Insecure world writable di
r /usr/local/bin in PATH, mode 041777
!     You've reached the limit of 5 apps for unverified accounts.
!     Delete some apps or add a credit card to verify your account.
[GreggsRetinaMBP:appcoda-mongo Gregg$ heroku create
/usr/local/heroku/lib/heroku/jsplugin.rb:87: warning: Insecure world writable di
r /usr/local/bin in PATH, mode 041777
Creating whispering-plains-1537... done, stack is cedar-14
https://whispering-plains-1537.herokuapp.com/ | https://git.heroku.com/whisperin
g-plains-1537.git
Git remote heroku added
GreggsRetinaMBP:appcoda-mongo Gregg$
```

現在輸入 **git push heroku master** 以便將此 App 傳送至 Heroku !

如果一切正確無誤的話（也應該是這樣才對），你將看到如下圖所示的畫面（設定可能會稍有不同）。

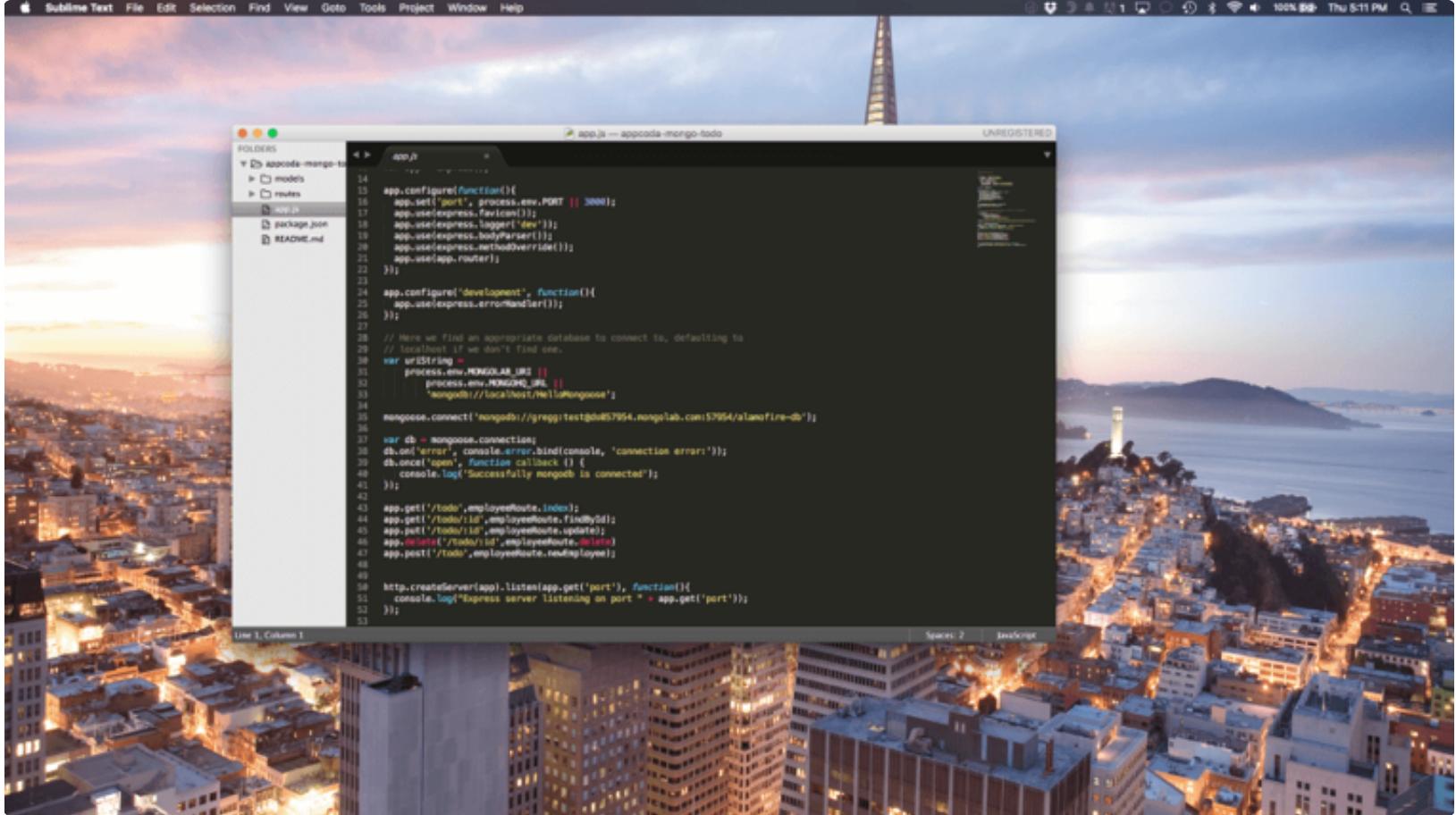
```
remote: -----> Caching build
remote:           Clearing previous node cache
remote:           Saving 2 cacheDirectories (default):
remote:             - node_modules
remote:             - bower_components (nothing to cache)
remote:
remote: -----> Build succeeded!
remote:   └── express@3.1.0
remote:     └── mongoose@4.2.8
remote:
remote:
remote: -----> Discovering process types
remote:           Procfile declares types      -> (none)
remote:           Default types for buildpack -> web
remote:
remote: -----> Compressing... done, 7.2MB
remote: -----> Launching... done, v12
remote:           https://rocky-meadow-1164.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/rocky-meadow-1164.git
  74b9184..3b945b1  master -> master
GreggsRetinaMBP:Mongoose-Heroku-Demo-master Gregg$
```

使用 Node.js 、 Express 、 MongoLab 和 Mongoose

在我們繼續之前，讓我們討論一下範例專案。開啟你最愛的文字編輯器（我使用的是 Sublime Text 2；你可以在 [SublimeText.com](#) 找到無限制的免費試用版），一起來研究範例專案的內容吧。

JavaScript 跟 Swift 非常類似，所以你應該會覺得很自在。我們將使用 express 和 mongoose 這 2 套很流行的 Node 套件。請確定你已安裝了 npm 這個 Node 套件管理工具。

Express 是一個快速、簡單的 Node.js 網頁框架，可以大幅簡化路由。你會問什麼是路由？路由是我們與網頁互動的方式。每次當你在瀏覽 google.com 的時候，實際上是連到 google.com/，也就是根索引。假使你連線到 google.com/hello，則又是另外一條路由了。稍後我們會定義希望自己的資料庫能夠接受的路由。



你可以從 expressjs.org 官網學到更多關於 Express 功能的事。

範例程式碼如下：

```
var express = require('express'); // 1
var app = express(); // 2
// 當此網頁收到 GET 要求時，便回應「hello world」
app.get('/', function(req, res) { // 3
  res.send('hello world'); //4
});
```

第 1 行設定了名為 express 的變數。第 2 行將 express 初始化給名為 app 的變數。我們會在第 3 行使用到 app (代表 Express 環境)。在此我們使用 get() 函式 (有點像 Swift 的函式)。當使用者瀏覽到 / (所有網站的預設根目錄) 的時候，將會出現「hello world」。這就是 Express 路由的範例。如欲獲得更多資訊，請參見 Express 網站，網址是 <http://expressjs.com/guide/routing.html>。

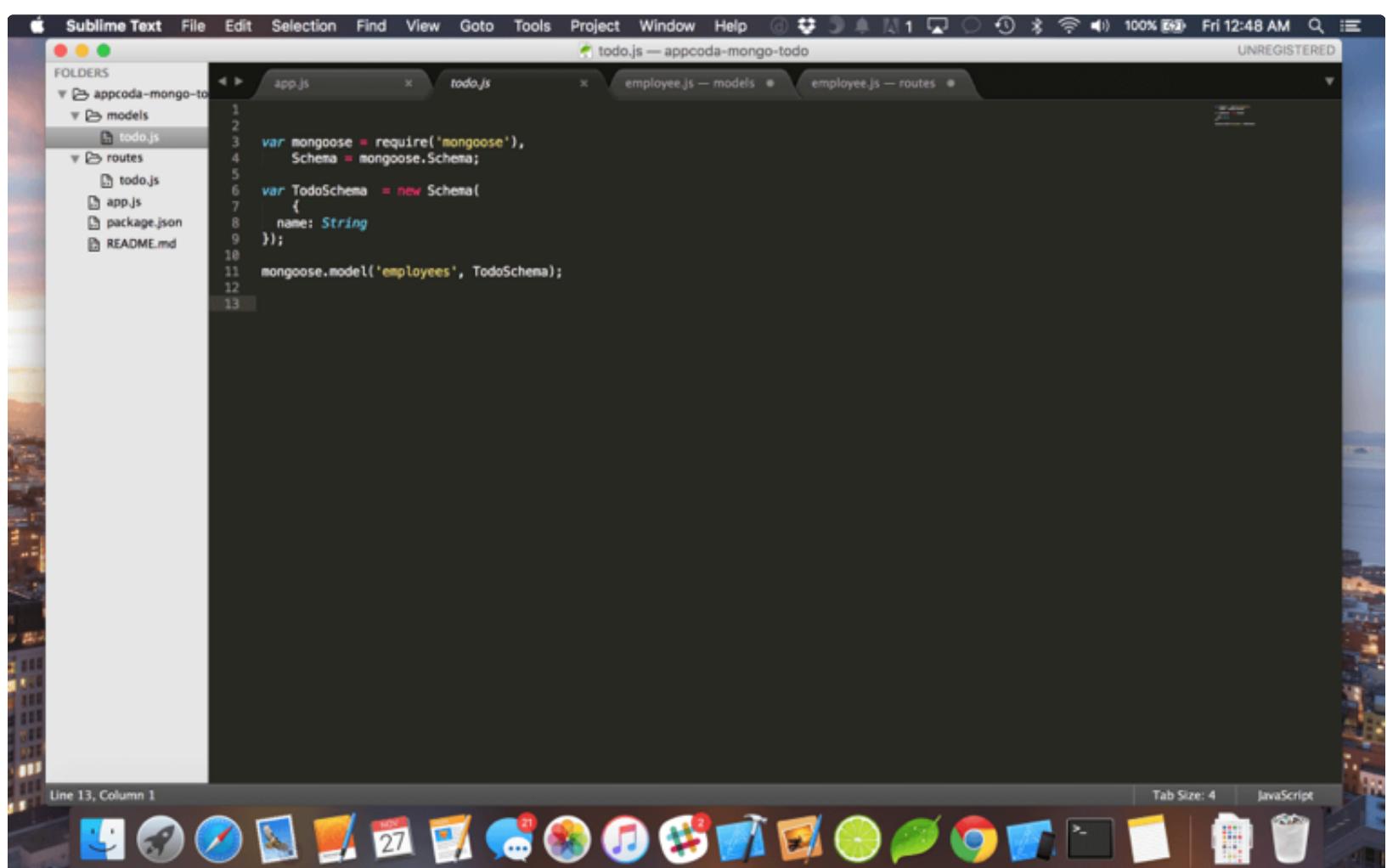
現在，我們的 Node 環境已經安裝完 Mongo 了，讓我們嘗試透過一些 cURL 要求來測試一下功能吧。cURL 是一支命令列程式，可以讓你發送 HTTP 要求。我們將先以 cURL 做實驗，再將 cURL 要求換成 AlamoFire。

Model 目錄

使用你的文字編輯器（我使用的是 Sublime）來開啟 `app.js` 檔案。如你所見，此 app 拆分成 model 和 routes 檔案（喔，當然了，還有你正在檢視的這個 `app.js` 檔案）。model 檔案構成 Schema（即資料庫結構）。讓我們來看一下此檔案（`models/todo.js`）的內容吧。

```
var mongoose = require('mongoose'),  
    Schema = mongoose.Schema;  
  
var TodoSchema = new Schema(  
{  
    name: String  
});  
  
mongoose.model('employees', TodoSchema);
```

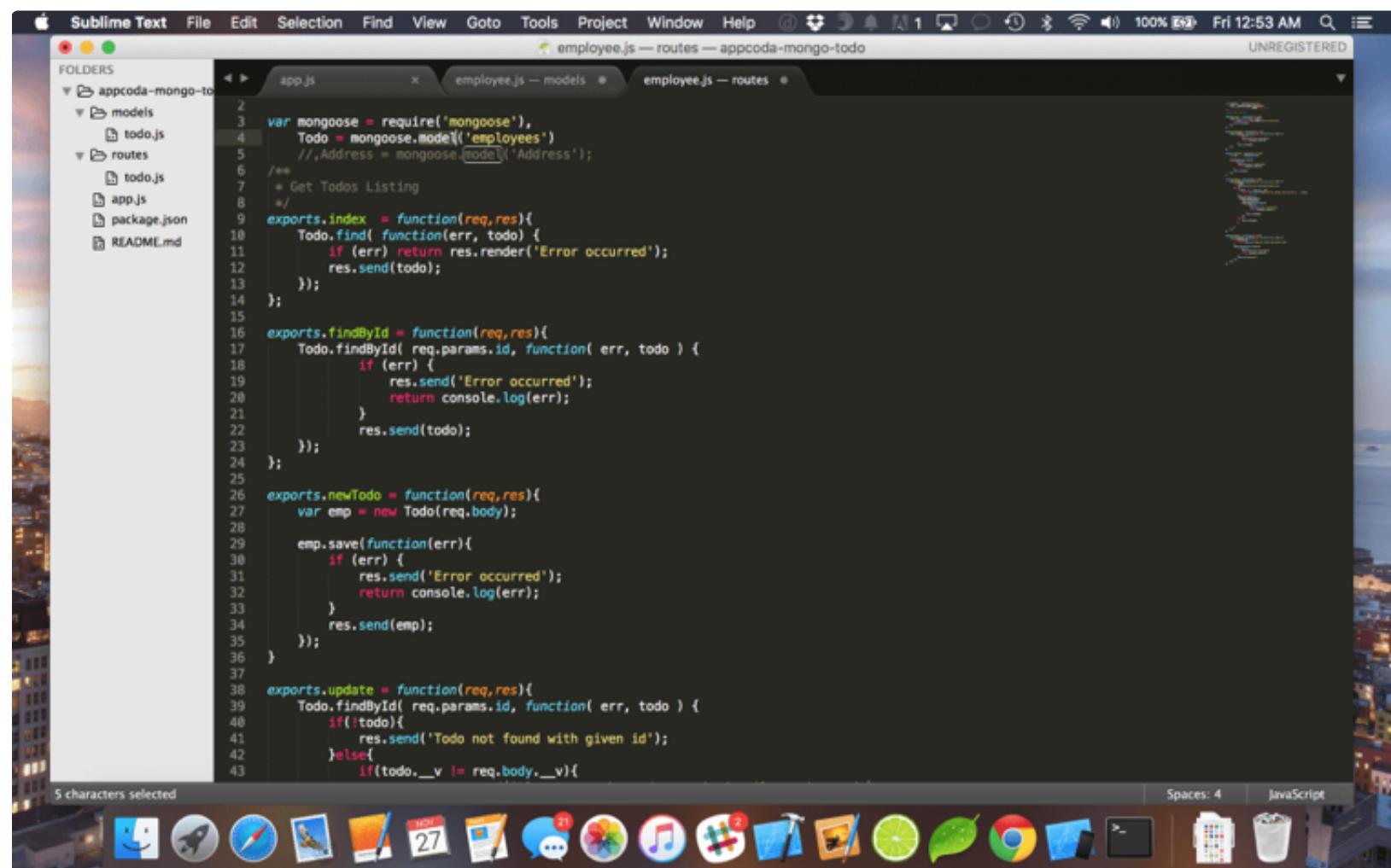
我們可以使用 `mongoose` 這套 npm 套件來串接此 App 與 Mongo。我在一開始時建立了一套員工追蹤 App，並將其模型取名為 `employees`，你儘管拿去改來用無妨。我還保留著此模型，因為本文後續章節將會需要用到！



mongoose 會將我們在 Heroku 上的 Node App 與 MongoLab 串接。真是太方便啦。

Routes 目錄

`routes` 檔案定義了要將哪些路由匯出到我們的 `app.js` 檔案。別擔心匯出的事情——這是 Node 的進階功能，本文將不會討論到。



請留意第 26 行的 `newTodo`。你可能會猜想，這個動作將會建立新的 TODO。

```
var emp = new Todo(req.body);
emp.save(function(err){
    if (err) {
        res.send('Error occurred');
        return console.log(err);
    }
    res.send(emp);
});
```

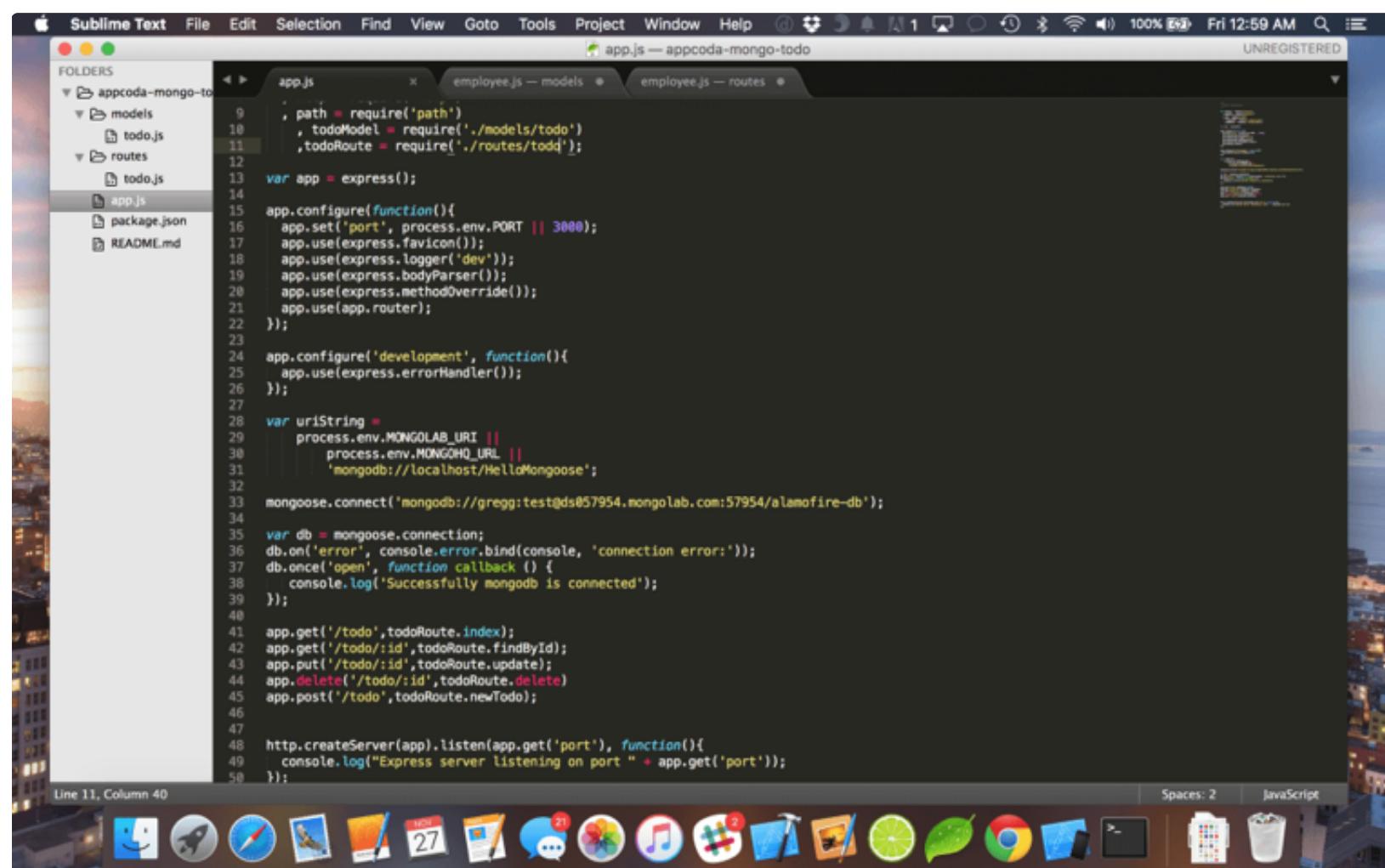
我們將 Todo 物件（定義於第 14 行，是用 mongoose 建立的連線）指派給名為 emp 的變數，並且設定 req.body （req 即 request 要求，指的是傳送給我們的資料，而 res 即 response 回應，指的則是傳回給我們的東西）。

別忘了把後續的函式也看一看喔。

神奇膠水——App.js

現在回到 app.js 檔案。這是整支 App 的主要部分，所有精華都在裡面。條列此檔案的幾個重點如下：

- 第 13 行建立 Express App
- 第 15 – 22 行設定此 App
- 第 33 行將此 App 與 mongoose 和 mongoLab 資料庫相連
- 第 35 行建立連線
- 第 41 – 45 行建立此 App 的路由，並且連接到 /routes/todo.js 檔案
- 第 48 行建立伺服器



The screenshot shows the Sublime Text editor running on a Mac OS X desktop. The window title is "app.js — appcoda-mongo-todo". The code editor displays the contents of the app.js file. The file structure on the left shows a project folder named "appcoda-mongo-todo" containing "models", "routes", and "app.js". The "app.js" file is open and shows the following code:

```
Sublime Text File Edit Selection Find View Goto Tools Project Window Help Fri 12:59 AM UNREGISTERED
FOLDERS app.js employee.js — models employee.js — routes
appcoda-mongo-todo
models
routes
app.js
package.json
README.md

app.js
9 , path = require('path')
10 , todoModel = require('./models/todo')
11 , todoRoute = require('./routes/todo');
12
13 var app = express();
14
15 app.configure(function(){
16   app.set('port', process.env.PORT || 3000);
17   app.use(express.favicon());
18   app.use(express.logger('dev'));
19   app.use(express.bodyParser());
20   app.use(express.methodOverride());
21   app.use(app.router);
22 });
23
24 app.configure('development', function(){
25   app.use(express.errorHandler());
26 });
27
28 var uriString =
29   process.env.MONGOLAB_URI ||
30   process.env.MONGOHQ_URL ||
31   'mongodb://localhost/HelloMongoose';
32
33 mongoose.connect('mongodb://gregg:test@ds057954.mongolab.com:57954/alamofire-db');
34
35 var db = mongoose.connection;
36 db.on('error', console.error.bind(console, 'connection error:'));
37 db.on('open', function callback () {
38   console.log('Successfully mongodb is connected');
39 });
40
41 app.get('/todo',todoRoute.index);
42 app.get('/todo/:id',todoRoute.findById);
43 app.put('/todo/:id',todoRoute.update);
44 app.delete('/todo/:id',todoRoute.delete);
45 app.post('/todo',todoRoute.newTodo);
46
47
48 http.createServer(app).listen(app.get('port'), function(){
49   console.log("Express server listening on port " + app.get('port'));
50 });

Line 11, Column 40
Spaces: 2 JavaScript
```

這樣你應該對於這支 JavaScript App 的運作有了基本的認識。我們畢竟是 iOS 的部落格，因此關於 JavaScript 的討論到此打住。不過建議你不妨好好研究 express 和 mongoose 。

使用 cURL

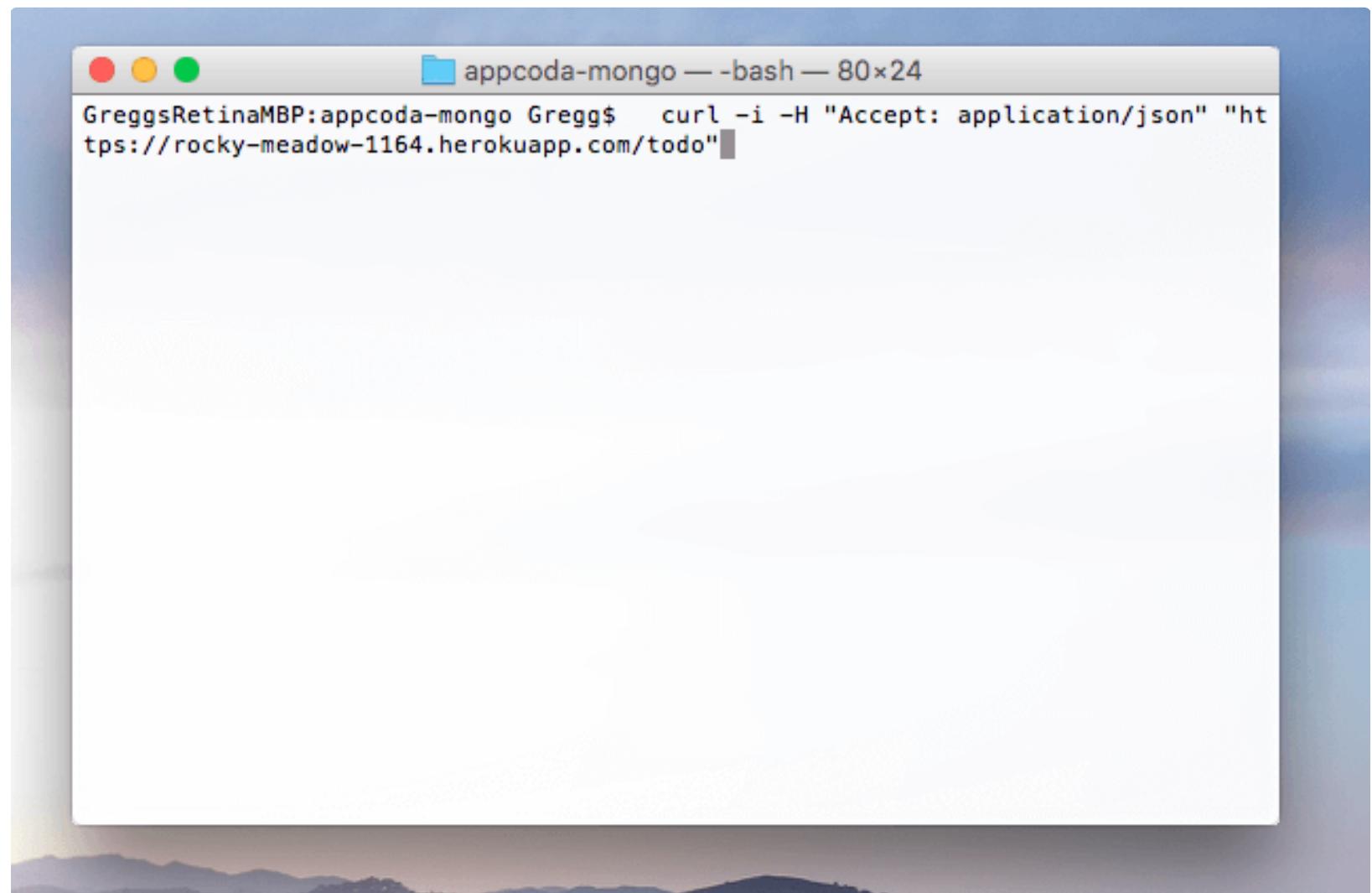
我們的 Node App 已經動起來了，現在讓我們透過 cURL 來進行一些測試吧。在完成這些測試之後，我們就可以遷移到 Alamofire 了。

GET 要求

在終端機執行下列的命令（別忘了將網址改成你自己的 Heroku 網址）。

```
curl -i -H "Accept: application/json" "https://rocky-meadow-1164.herokuapp.com/todo"
```

-i 和 -H 是旗標，它們說明了將會接受什麼資料。我們將會接受 JSON，並且將 JSON 網址附加到要求的最後面。



你應該會看到有資料傳回來。如下圖所示。

```
[GreggsRetinaMBP:appcoda-mongo Gregg$ curl -i -H "Accept: application/json" "https://rocky-meadow-1164.herokuapp.com/todo"
HTTP/1.1 200 OK
Server: Cowboy
Connection: keep-alive
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 102
Date: Thu, 26 Nov 2015 23:00:48 GMT
Via: 1.1 vegur

[{"_id": "56577e145d9a9e09007ee93f", "name": "Idiot", "__v": 0}, {"_id": "56577f258659c509000dd5f78", "name": "Buy Pre", "__v": 0}]GreggsRetinaMBP:appcoda-mongo Gregg$
```

如你所見，傳回了一些資料，那些正是我們想要的。假使你將網址換成你自己的，那麼可能不會看到任何東西，因為你的 MongoDB 中可能還沒有任何資料。

POST 要求

如果你想要在資料庫中新增資料，可以執行簡單的 POST 命令。

```
curl -H "Content-Type: application/json" -X POST -d '{"name": "Buy Pre'}
```

```
appcoda-mongo — curl -H Content-Type: application/json -X POST -d {"name": "Buy Presents"} https://rocky-meadow-1164.herokuapp.com/todo
```

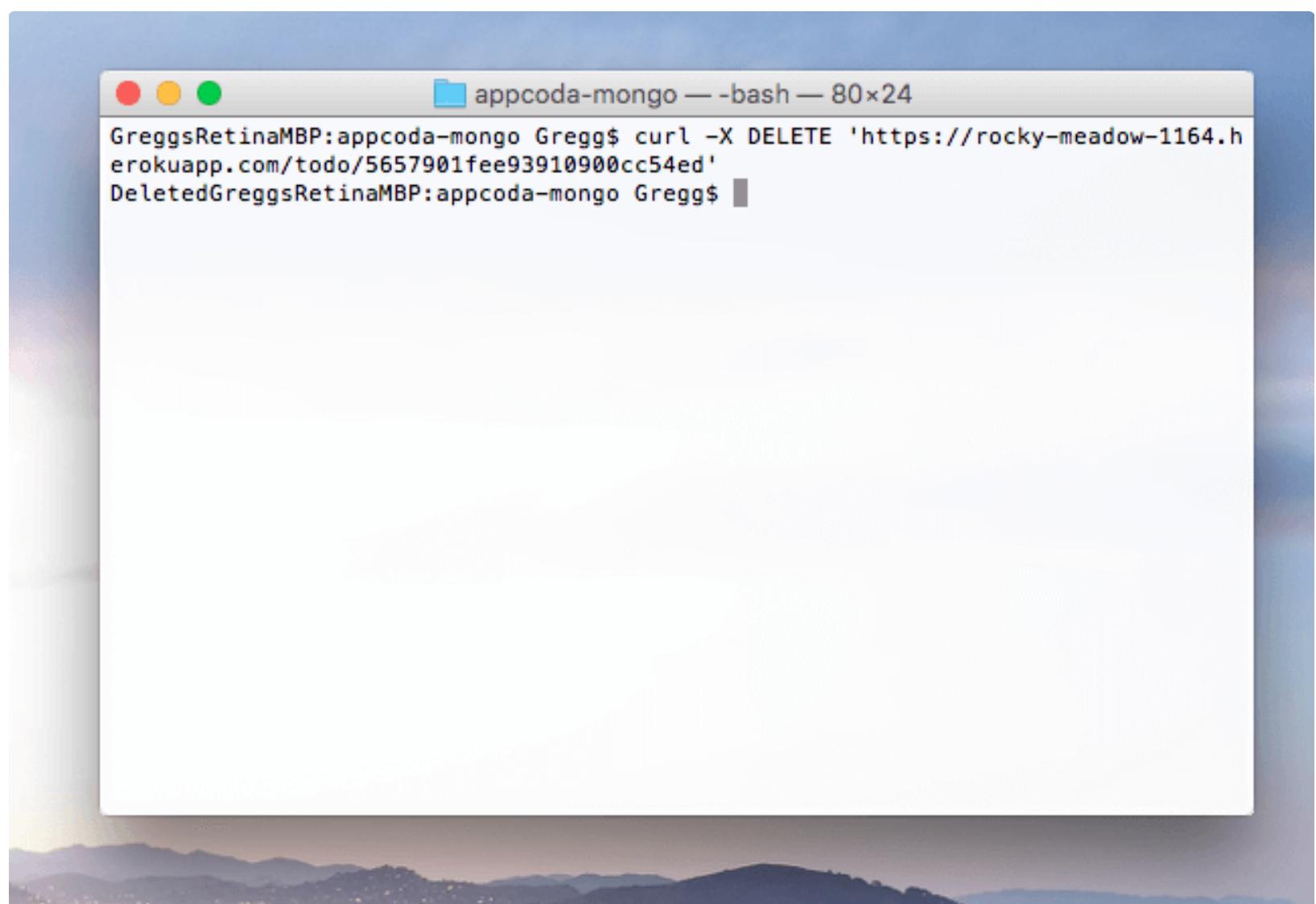
現在，再次執行稍早的 GET 命令，這回應該會看到我們剛才加入的「Buy Presents」。

```
appcoda-mongo — bash — 80x24
GreggsRetinaMBP:appcoda-mongo Gregg$ curl -H "Content-Type: application/json" -X POST -d '{"name":"Buy Presents"}' https://rocky-meadow-1164.herokuapp.com/todo
{"__v":0,"name":"Buy Presents","_id":"5657901fee93910900cc54ed"}GreggsRetinaMBP:
appcoda-mongo Gregg$ [GreggsRetinaMBP:appcoda-mongo Gregg$ curl -H "Content-Type: application/json" -X POST -d '{"name":"Buy Presents"}' https://rocky-meadow-1164.herokuapp.com/employee
[Cannot POST /employeeGreggsRetinaMBP: curl -i -H "Accept: application/json" "https://rocky-meadow-1164.herokuapp.com/todo"
HTTP/1.1 200 OK
Server: Cowboy
Connection: keep-alive
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 167
Date: Thu, 26 Nov 2015 23:06:00 GMT
Via: 1.1 vegur
[{"_id":"56577e145d9a9e09007ee93f","name":"Id: .","__v":0}, {"_id":"56577f258659c50900dd5f78","__v":0}, {"_id":"5657901fee93910900cc54ed","name":"Buy Presents","__v":0}]GreggsRetinaMBP:appcoda-mongo Gregg$ █
```



DELETE 要求

```
curl -X DELETE 'https://rocky-meadow-1164.herokuapp.com/todo/5657901fee93910900cc54ed'
```



太棒了。我們不必討論 PUT，因為在我們的 App 裡面用不到，不過 PUT 跟其他幾個方法大同小異。

在 iOS App 中啟用 Alamofire



Elegant Networking in Swift

現在讓我們從建立名為 TodoApp 的全新 Xcode 專案開始吧。因為假期是由我們自己決定的，所以我們應該要自己想辦法管理。幸運的是，我們的 Node 應用程式可以幫得上忙！

雖然你也可以手動安裝 Alamofire (透過將檔案拖曳到專案中)，不過我們打算使用 Cocoapods。Cocoapods 是一種 iOS 專案的相依性管理工具。透過使用 Cocoapods，開發者可以很輕鬆地新增框架和第三方程式庫。這是一項非常棒的資源，假使你未曾使用過的話，那麼你絕對值得嘗試看看！

如你是首次使用CocoaPods，你可以參考[這篇CocoaPods初學者指南](#)安裝 CocoaPods。

假設你已安裝CocoaPods，現在 cd 到你的專案所在位置。輸入下列的命令：

```
pod init
```

CocoaPods會製作一個 Podfile，這是 Cocoapods 每次在更新專案的 Pods (亦即不同的相依性套件) 時都會檢視的檔案。

之後，修改Podfile並加入Alamofire (黃色標誌的一行)：

```
# Uncomment this line to define a global platform for your project
# platform :ios, '9.0'

target 'ToDoApp' do
  # Comment this line if you're not using Swift and don't want to use
  use_frameworks!

  # Pods for ToDoApp
  pod 'Alamofire', '~>3.0'
end
```

輸入下列指令以便安裝 Cocoapods：

```
pod install
```

如果一切正確無誤的話，應該會看到類似下圖的畫面。

```
[GreggsRetinaMBP:TodoApp Gregg$ pod install
/Users/Gregg/.rvm/gems/ruby-2.2.1@global/gems/cocoapods-0.38.2/lib/cocoapods/com
mand.rb:130: warning: Insecure world writable dir /usr/local/bin in PATH, mode 0
41777
Updating local specs repositories

CocoaPods 0.39.0 is available.
To update use: `gem install cocoapods`
Until we reach version 1.0 the features of CocoaPods can and will change.
We strongly recommend that you use the latest version at all times.

For more information see http://blog.cocoapods.org
and the CHangelog for this version http://git.io/BaH8pQ.

Analyzing dependencies
Downloading dependencies
Installing Alamofire (3.1.3)
Generating Pods project
Integrating client project

[!] Please close any current Xcode sessions and use `TodoApp.xcworkspace` for th
is project from now on.
Sending stats
GreggsRetinaMBP:TodoApp Gregg$
```

如同命令列上的文字所建議的，你應該關閉目前的 Xcode 專案，開啟 Finder 視窗，然後選取 Cocoapods 所產生的副檔名為 `ToDoApp.xcworkspace` 的檔案。

之後，打開 `ViewController.swift`，讓我們開始寫點程式吧。

Alamofire GET 要求

首先請匯入 Alamofire：

```
import Alamofire
```

接著在 `viewDidLoad()` 裡面輸入下列的程式碼：

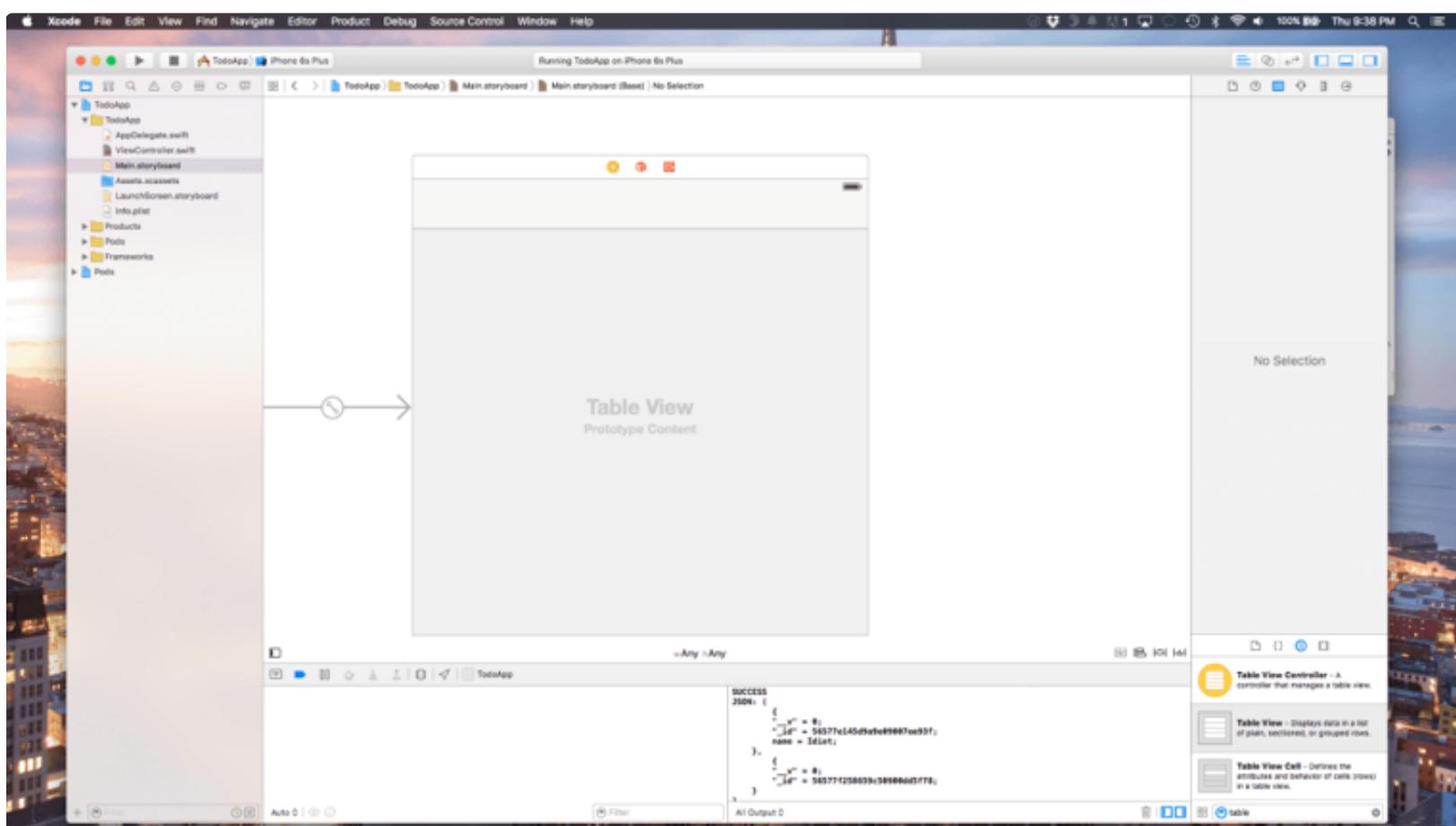
```
Alamofire.request(.GET, "https://rocky-meadow-1164.herokuapp.com/todo")
    .print(response.request) // 原始的 URL 要求
    .print(response.response) // URL 回應
    .print(response.data) // 伺服器資料
    .print(response.result) // 回應的序列化結果

    if let JSON = response.result.value {
```

```
    print("JSON: \(JSON)")
}
}
```

在第 1 行中，我們表明此為 GET 要求，並且傳入所需的 URL（別忘記將 URL 改成你的 Heroku app URL）。現在執行此 App，並且觀察傳回來的結果。如果正確無誤的話，你在 console 看到的會是 JSON 資料。

現在來到 Main.storyboard，在視圖控制器中加入表格視圖，並且嵌入到導覽控制器中。你的 Storyboard 看起來應該跟我的很像，如下圖所示（請仔細看主控台中從我們的 Node 伺服器所傳回的 JSON 資料）。



將下列的程式碼複製並貼上到你的 `ViewController.swift` 檔案中。

```
import UIKit
import Alamofire

class ViewController: UIViewController {

    @IBOutlet weak var tableView: UITableView!
    var jsonArray: NSMutableArray?
    var newArray: Array<String> = []
}
```

```

override func viewDidLoad() {
    super.viewDidLoad()

    tableView.delegate = self
    tableView.dataSource = self

    Alamofire.request(.GET, "https://still-peak-69201.herokuapp.com")
        .print(response.request) // 原始的 URL 要求
        .print(response.response) // URL 回應
        .print(response.data) // 雲端資料
        .print(response.result) // 回應的序列化結果

        if let JSON = response.result.value {
            self.jsonArray = JSON as? NSMutableArray
            for item in self.jsonArray! {
                if let name = item["name"], myName = name as? String {
                    print(myName)
                    self.newArray.append(myName)
                }
            }
        }

        self.tableView.reloadData()
    }
}
}

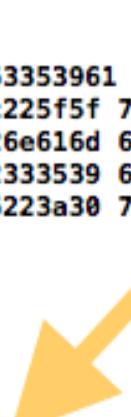
```

我初始化了 2 個陣列，並且運用一個簡單的 for 迴圈來迭代處理陣列中的元素，分別指派給 newArray 物件。

```

} })
Optional(<5b7b225f 6964223a 22353635 37633861 63353961 37663230 61303030 66643639 37222c22
6e616d65 223a2242 75792050 72657365 6e747322 2c225f5f 76223a30 7d2c7b22 5f696422 3a223536 35376338
65623539 61376632 30613030 30666436 3938222c 226e616d 65223a22 56697369 74205361 6e746122 2c225f5f
76223a30 7d2c7b22 5f696422 3a223536 35376339 32333539 61376637 30613030 30666436 3961222c 226e616d
65223a22 4d616b65 2044696e 6e657222 2c225f5f 76223a30 7d5d>)
SUCCESS
Optional(Buy Presents)
String is Buy Presents
Optional(Visit Santa)
String is Visit Santa
Optional(Make Dinner)
String is Make Dinner
New array is ["Buy Presents", "Visit Santa", "Make Dinner"]

```



小貼士：你可以透過使用 POST cURL 要求以在資料庫中增加更多的項目。

接著，在檔案開頭處的 UIViewController 壓告後面，加入 UITableViewDelegate 和 UITableViewDataSource。然後在 viewDidLoad() 中加入下列的程式碼：

```
tableView.dataSource = self  
tableView.delegate = self
```

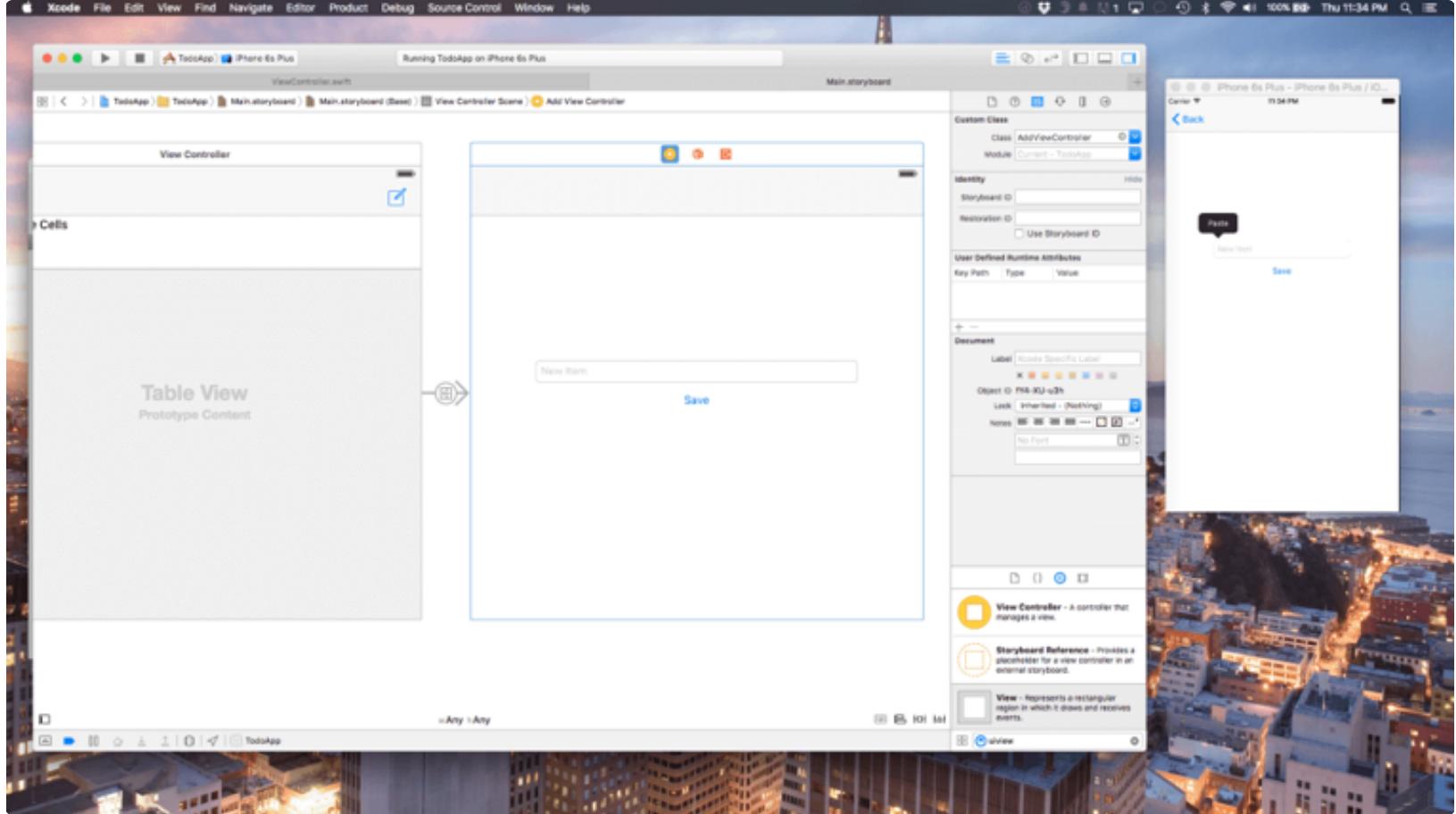
最後，加入下列這些 UITableView 委派函式：

```
func tableView(tableView: UITableView, numberOfRowsInSection section:  
    return self.newArray.count  
  
}  
  
func tableView(tableView: UITableView, cellForRowAt indexPath: IndexPath:  
    let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", fo  
  
    cell.textLabel?.text = self.newArray[indexPath.row]  
    return cell  
}
```

如你所見，我們的表格視圖已經填好資料了！

Alamofire POST 要求

現在讓我們來加入一顆編寫（Compose）按鈕，以便在清單中加入新的項目。在 Storyboard 中增加一個名為 `AddViewController` 的新類別，並且連結成 Segue。你的 Storyboard 看起來應該如下圖所示。



在你的 `AddViewController.swift` 檔案中，針對文字欄位（取名為 `textField`）建立 `IBOutlet`，同時也為儲存（Save）按鈕建立 `IBAction`。在 Save 按鈕中，輸入下列的程式碼。

```
@IBAction func saveButtonTapped(sender: AnyObject) {
    guard let todoItem = textField.text else {
        return
    }

    Alamofire.request(.POST, "https://still-peak-69201.herokuapp.com/")
    self.navigationController?.popViewControllerAnimated(true)

}
```

如你所見，Alamofire 大幅簡化了傳送 POST 要求的流程。

接著，讓我們重構一下 `ViewController.swift` 的程式碼，以便在我們儲存完新的項目之後能夠更新顯示。將你的 Alamofire GET 程式碼從 `viewDidLoad()` 抽取出來，獨立成名為 `downloadAndUpdate` 的函式：

```
func downloadAndUpdate() {
    Alamofire.request(.GET, "https://still-peak-69201.herokuapp.com/")
    print(response.request) // 原始的 URL 要求
```

```
print(response.response) // URL 回應
print(response.data)    // 伺服器資料
print(response.result)  // 回應的序列化結果

if let JSON = response.result.value {
    self.jsonArray = JSON as? NSMutableArray
    for item in self.jsonArray! {
        if let name = item["name"], myName = name as? String
            print(myName)
        self.newArray.append(myName)
    }
}

self.tableView.reloadData()
}
}
}
```

現在，加入 `viewWillAppear()` 函式，並且在其中呼叫你的函式。

```
override func viewWillAppear(animated: Bool) {
    downloadAndUpdate()
}
```

如果再次執行此 App 並且完成整個建立流程的話，此 App 每次都會重新載入新的 TODO。這是為什麼呢？

好吧，讓我們來討論一下（非常簡短）視圖控制器的生命週期。`viewDidLoad()` 是在視圖實體化時會被呼叫我到的函式，所有事情在那裡都已經做完了。問題是，當我們載入另外一個視圖（亦即 `AddViewController` 類別）到已經載入過的 `ViewController` 類別之上的時候，`viewDidLoad` 並不會再被呼叫一次（因為先前已經做過實體化了）。然而 `viewWillAppear` 則是每次當視圖顯示在螢幕上的時候，都會被呼叫我到。因為我們需要再次顯示 `ViewController.swift`，所以此函式剛好派得上用場。

Alamofire DELETE 要求

現在，讓我們在 `newArray` 變數之下加入下列的陣列：

```
var idArray: Array<String> = []
```

接著，稍微修改你的 downloadAndUpdate 函式，如下所示。

```
func downloadAndUpdate() {
    newArray.removeAll()
    idArray.removeAll()

    Alamofire.request(.GET, "https://still-peak-69201.herokuapp.com/")
        print(response.request) // 原始的 URL 要求
        print(response.response) // URL 回應
        print(response.data) // 伺服器資料
        print(response.result) // 回應的序列化結果

        if let JSON = response.result.value {
            self.jsonArray = JSON as? NSMutableArray
            for item in self.jsonArray! {
                if let name = item["name"], myName = name as? String,
                   let id = item["_id"], myId = id as? String {
                    self.newArray.append(myName)
                    self.idArray.append(myId)
                }
            }
        }

        self.tableView.reloadData()
    }
}
```

新的那幾行都以黃色標註。基本上，我們是透過迴圈在處理陣列，用來取得每個索引的 ID 然後儲存到 `idArray` 之中。此外我們也移除了陣列中的舊項目，並且加以重設。

現在加入 commitEditingStyle 函式，以便呼叫 DELETE 要求。

```
func tableView(tableView: UITableView, commitEditingStyle editingStyle: UITableViewCellEditingStyle, indexPath: NSIndexPath) {
    if editingStyle == .Delete {
        print("ID is \(idArray[indexPath.row])")

        Alamofire.request(.DELETE, "https://still-peak-69201.herokuapp.com/")
        downloadAndUpdate()
    }
}
```

```
    }  
}
```

如你所見，我們遵循自己的 Node App 的 API ， DELETE 要求都是來自於 /todo/*ID* 。再次強調，這份非常簡單的 Alamofire 函式實作會呼叫 DELETE 要求，並且刪除 TODO 項目。

現在你已經完成程式的實作了，這是一支功能完整的 Todo App ，可以在假期中使用。那麼就讓我們來為本文做個總結吧。

結語

我們在本文中探討了許多議題。從 JavaScript 的 Node 到 Express ，從 MongoDB 到 cURL ，也談到了在終端機中操作 Cocoapods ，最後則是 Alamofire ，我們討論了製作 REST API 的細節，以及網路的運作。現在你應該要知道如何：

- 打造 API
- 部署 API
- 撰寫用戶端 App
- 與 HTTP 動詞進行互動
- 使用 Cocoapods
- 使用 cURL
- 與 Node 以及 MongoDB/Express 環境進行互動
- 使用 Express 路由
- 使用 Alamofire

本文是一份龐大的教學，感謝你讀完了。供你參考，請從 [這裡](#) 下載本文完整的原始碼（包括 Node App 和 iOS App ）。

一如往常，歡迎留言寫下你的想法，我會跳出來回答所有問題。下回再見！

譯者簡介：陳佳新 – 奇步應用共同創辦人，開發自有 App 和網站之外，也承包各式案件。譯有多本電腦書籍，包括 O'Reilly 出版的 iOS 、 Android 、 Agile 和 Google Cloud 等主題，也在報紙上寫過小說。現與妻兒居住在故鄉彰化。歡迎造訪 <https://chibuapp.com> ，來信請寄到 jarsing@chibuapp.com 。

原文：[The Absolute Guide to Networking in Swift with Alamofire](#)

ALAMOFIRE

HEROKU

MONGODB

NODE.JS

SWIFT

網路程式

分享

f FACEBOOK

g+ GOOGLE+

twitter TWITTER

in LINKEDIN

SWIFT

Swift 3：你要知道的新特色和改動

[下一篇](#)

iOS

如何在 iOS App 中整合 Facebook 廣告



@GreggMojica

Gregg Mojica

軟體工程師及Gradology公司的資訊科技總監。在繁忙的工作以外，他享受於寫作、攝影和分享所見所聞。Gregg熱衷程式開發，曾在app store發佈數個apps，客戶包括美國康奈爾大學、各式地方企業和初創公司。可以在推特或LinkedIn聯絡Gregg。



1 Comment

AppCoda中文

1 Login

Recommend

Share

從最好的優先排列 ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS



jason shi (小胖) · 10個月前

感謝你的分享

^ | v · Reply · Share ·

ALSO ON APPCODA中文

透過 Firebase 與 Raspberry Pi 製作無

聯網 iOS 程式

1 comment · 一年前

宏安吳 — <https://uploads.disquscdn.com>



The image shows a modal window for signing up to receive email newsletters. It features a large orange header with a white envelope icon. The text "訂閱電子報" (Subscribe to Email Newsletter) is prominently displayed. Below the header, there is a text input field labeled "電子郵件地址" (Email Address) and a dark grey button labeled "訂閱" (Subscribe). The background of the modal is white.

訂閱電子郵件

AppCoda致力於發佈優質iOS程式教學，你不必每天上站，輸入你的電子郵件地址訂閱網站的最新教學文章。每當有新文章發佈，我們會使用電子郵件通知你。

電子郵件地址

訂閱