

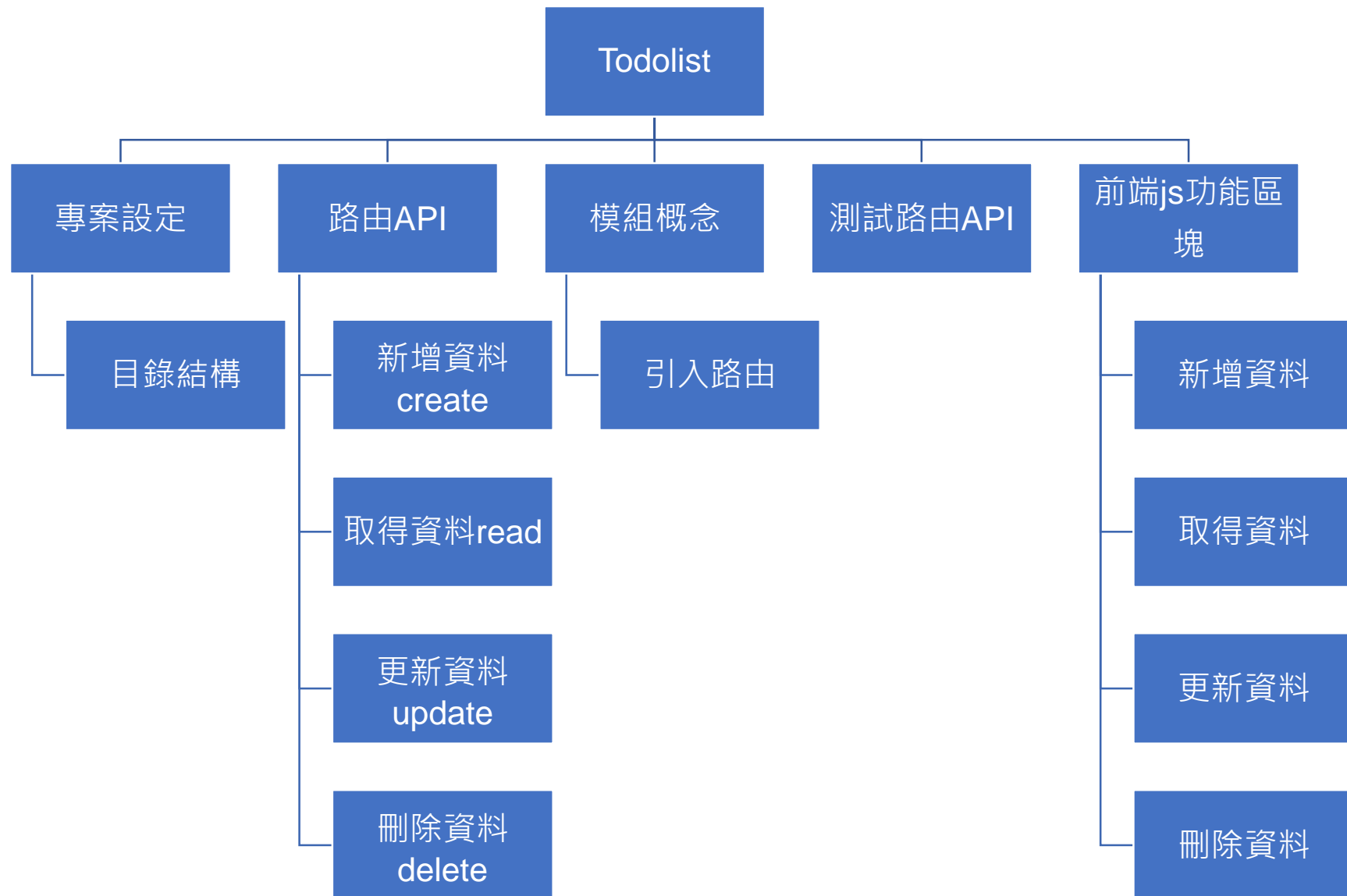
Node.js
express-generator

學習目標

- 了解中介軟體(middleware)
- 具備設定路由(routes)中介軟體能力
- 學習前後端請求與回覆關係與流程
- 學習測試路由API功能

學習任務

- 完成Todolist小專案
 - 路由
 - 請求
 - 回覆
 - Ajax



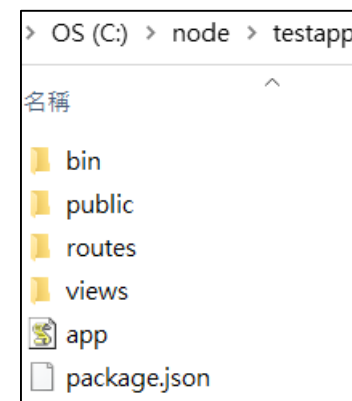
安裝Express應用程式產生器

- `npm i express-generator -g`

```
PS C:\node> npm install express-generator -g
npm WARN deprecated mkdirp@0.5.1: Legacy versions of mkdirp are no longer supported. Please update to mkdirp 1.x. (Note that the API surface has changed to use Promises in 1.x.)
C:\Users\USER\AppData\Roaming\npm\express -> C:\Users\USER\AppData\Roaming\npm\node_modules\express-generator\bin\express-cli.js
+ express-generator@4.16.1
added 10 packages from 13 contributors in 1.029s
```

- 建立一個testapp專案
 - 在C硬碟下建立一個node的資料夾
 - 日後關於node的專案皆存放於此
 - 在c:node> 輸入 **express testapp**
 - 安裝成功後的畫面如右圖

```
C:\node> express testapp
```



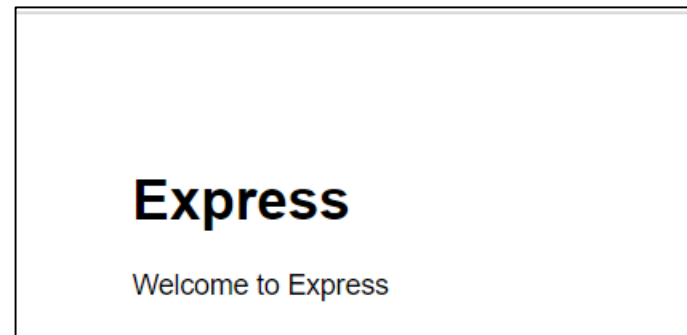
安裝相關套件

- 進入testapp透過指令npm install安裝相關套件
 - cd testapp
- 輸入 npm install
 - 此處後面沒有指定的套件名稱，將會依據package.json檔案內的清單進行安裝。









```
{
  "name": "testapp",
  "version": "0.0.0",
  "private": true,
  > Debug
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "cookie-parser": "~1.4.4",
    "debug": "~2.6.9",
    "express": "~4.16.1",
    "http-errors": "~1.6.3",
    "jade": "~1.11.0",
    "morgan": "~1.9.1"
  }
}
```

如何運行專案

- cmd內輸入npm start
- 打開瀏覽器輸入localhost:3000
- 執行結果如右圖



Express檔案結構說明

 bin	專案啟動設定位置，預設是bin/www，負責啟動Node.js server並呼叫app.js
 node_modules	已安裝套件，此專案必要的模組，透過npm安裝會儲存在此
 public	放置可公開存取的靜態檔案，預設為JS、CSS、images，可依需求增加
 routes	路由位置設定，主要負責傳遞資料、設定API路由路徑
 views	預設頁面位置，也就是顯示網頁畫面，預設檔案格式.jade，Jade為樣板引擎
 app	程式進入點，後續詳細說明
 package.json	專案相關資訊檔，例如專案名稱、版本、描述
 package-lock.json	檢驗套件版本，當node_modules或package.json發生變動會自動生成文件，並且檢驗文件的版本是否相符合。

Express 基礎概念

- Express專案啟動設定的位置，預設為bin資料夾下的www檔案，負責啟動Node.js Server，同時呼叫app.js啟動專案程式。
- 打開www後可見程式碼上皆有註解說明每行程式的目的。

```
JS www x
node > testapp > bin > JS www > ...
1  #!/usr/bin/env node
2
3  /**
4   * Module dependencies.
5   */
6
7  var app = require('../app');
8  var debug = require('debug')('testapp:server');
9  var http = require('http');
10
11  /**
12   * Get port from environment and store in Express.
13   */
14
15  var port = normalizePort(process.env.PORT || '3000');
16  app.set('port', port);
17
18  /**
19   * Create HTTP server.
20   */
21
22  var server = http.createServer(app);
23
24  /**
25   * Listen on provided port, on all network interfaces.
26   */
27
```

專案www進入點

- 引入必要檔案，也就是app.js，此處不需要加上副檔名。

```
var app = require('../app');
```

- 引入http模組

```
var http = require('http');
```

- 設定連接埠port

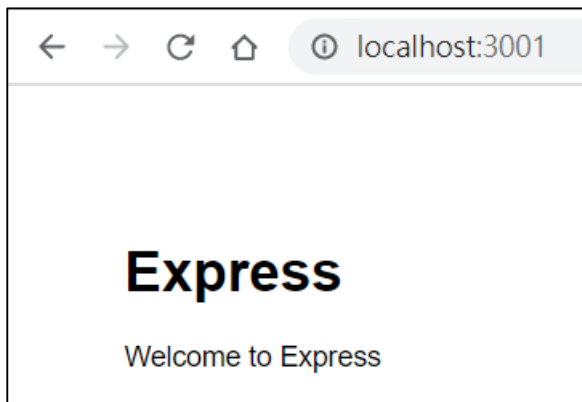
```
var port = normalizePort(process.env.PORT || '3000');  
app.set('port', port);
```

- 若存在環境變量process.env.PORT則使用，無則使用3000
- 此處port更改為3001測試

- 每次修改後程式必須要重新啟動程式，先按下**ctrl+c**，畫面詢問是否要終止，在按下**Y**表示確定，再重新重啟專案即可。

```
PS C:\node\testapp> npm start  
  
> testapp@0.0.0 start C:\node\testapp  
> node ./bin/www  
  
GET / 304 152.082 ms - -  
GET /stylesheets/style.css 304 1.140 ms - -  
要終止批次工作嗎 (Y/N)? y  
PS C:\node\testapp> █
```

- 修改port後畫面如圖



package.json

- package.json此檔案主要存放專案的相關訊息，以JSON格式儲存。相關資訊主要分成三部分

基本上不需要做額外的變動

```
{
  "name": "testapp",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "cookie-parser": "~1.4.4",
    "debug": "~2.6.9",
    "express": "~4.16.1",
    "http-errors": "~1.6.3",
    "jade": "~1.11.0",
    "morgan": "~1.9.1"
  }
}
```

專案描述

自訂指令

專案依賴安裝的套件

scripts

- **Scripts**區塊是用來存放開發者自訂的指令，當要執行自訂指令時，只要透過**npm**來運行就可以，在**Express**專案中已經預設一個自訂指令。

```
"scripts": {  
  "start": "node ./bin/www"  
},
```

- **Start**是啟動專案時**npm start**使用的指令，目的是取代 **node ./bin/www** 一長串的指令，透過簡單的**npm start**輸入即可。換句話說 **npm start = node ./bin/www**

自訂測試指令

```
"scripts": {  
  "start": "node ./bin/www",  
  "test": "echo This is a test message!"  
},
```

- 新增一個**test**指令，會列印出文中的訊息，儲存完成後，在cmd視窗中執行**npm test**即會顯示

```
PS C:\node\testapp> npm test  
  
> testapp@0.0.0 test C:\node\testapp  
> echo This is a test message!  
  
This is a test message!
```

dependencies

- 此區塊主要紀錄目前專案依賴所需的套件名稱以及版本。
- 自動安裝方式
 - 若在此專案下執行`npm install`則會自動建立`node_modules`資料夾，並依據`dependencies`中的套件自動安裝，開發者不需要手動安裝。
- 解除安裝套件
 - `npm uninstall express` 加上要刪除的套件名稱
 - 解除完成後`dependencies`內的套件名稱也會一併更新
- 自動記錄新增加的套件名稱
 - 未來專案若需要增加新的套件，在安裝過程中加`--save`參數，便會自動加入到`dependencies`中。

App.js 程式進入點

- app.js扮演整個專案的程式進入點。
- 由於此專案是使用express產生器自動建立，因此app.js內預設許多功能。

```
1  var createError = require('http-errors');
2  var express = require('express');
3  var path = require('path');
4  var cookieParser = require('cookie-parser');
5  var logger = require('morgan');
6
7  var indexRouter = require('./routes/index');
8  var usersRouter = require('./routes/users');
9
10 var app = express();
11
12 // view engine setup
13 app.set('views', path.join(__dirname, 'views'));
14 app.set('view engine', 'jade');
15
16 app.use(logger('dev'));
17 app.use(express.json());
18 app.use(express.urlencoded({ extended: false }));
19 app.use(cookieParser());
20 app.use(express.static(path.join(__dirname, 'public')));
21
22 app.use('/', indexRouter);
23 app.use('/users', usersRouter);
24
25 // catch 404 and forward to error handler
26 app.use(function(req, res, next) {
```


App.js 引入套件模組

- `express` 模組用來產生應用程式。
- `path` 模組用來處理專案中的檔案路徑。
- `cookie-parser` 模組用來處理分析網頁中暫存的資料 `cookie`。
- `logger` 變數是引入 `morgan` 模組並命名為 `logger`，用來記錄 HTTP 請求的訊息。
- **Routes**：專案預設在 `app.js` 將原有的路由檔案引入並作為預設，若自訂路由程式，也需要將此路由引入

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
```

建立express模組

- 引入完模組後建立express模組，並命名為app

```
var app = express();
```

- 透過app.set設定樣板引擎，在此使用的是jade

```
// view engine setup  
app.set('views', path.join(__dirname, 'views'));  
app.set('view engine', 'jade');
```

- Jade已經被更名改用pug，需要安裝pug
 - 一開始安裝即指定
 - `express --view=pug myapp`

Margon模組

- logger先前提到是用來記錄HTTP相關的請求

```
app.use(logger('dev'));
```

- 將logger加入到app中，其中dev字串是margan模組定義的紀錄格式，當運行專案開啟瀏覽器進入頁面後，在CMD中可看到HTTP的請求 GET/ 304

```
PS C:\node\testapp> npm start

> testapp@0.0.0 start C:\node\testapp
> node ./bin/www

GET / 304 172.024 ms - -
GET /stylesheets/style.css 304 1.438 ms - -
GET / 304 9.908 ms - -
GET /stylesheets/style.css 304 0.544 ms - -
```

Morgan模組 – 內建預設格式

- dev

```
:method :url :status :response-time ms - :res[content-length]
```

- common

```
:remote-addr - :remote-user [:date[clf]] ":method :url HTTP/:http-version" :status  
:res[content-length]
```

```
::1 - - [22/Jul/2020:03:10:54 +0000] "GET / HTTP/1.1" 304 -  
::1 - - [22/Jul/2020:03:10:54 +0000] "GET /stylesheets/style.css HTTP/1.1" 304 -
```

- short

```
:remote-addr :remote-user :method :url HTTP/:http-version :status :res[content-length]  
- :response-time ms
```

```
::1 - GET / HTTP/1.1 304 - - 150.386 ms  
::1 - GET /stylesheets/style.css HTTP/1.1 304 - - 1.613 ms
```

- tiny

```
:method :url :status :res[content-length] - :response-time ms
```

Margon模組 – 自訂格式

```
//morgan自訂格式
logger.format('myFormat', ':method :url :status :response-time :remote-addr');
app.use(logger('myFormat'));
```

- 時間紀錄中沒有填入ms，因此請求回覆的時間不會顯示ms

```
GET / 304 142.337 ::1
GET /stylesheets/style.css 304 1.189 ::1
```

```
GET / 200 9.231 ::ffff:127.0.0.1
GET /stylesheets/style.css 200 2.221 ::ffff:127.0.0.1
```

Express中介軟體模組(module)

- 加入express模組範例

```
app.use(express.json());  
app.use(express.urlencoded({ extended: false }));  
app.use(express.static(path.join(__dirname, 'public')));
```

- 其中app.use(中介軟體模組名稱)為使用中介軟體的指令
- 第一和二種使用json()和urlencoded()透過POST可以解析JSON和urlencoded資料格式。
 - urlencoded為常見HTTP資料傳輸一種資料格式，其為key1=value1&key2=value2
- 第三種則使用加入靜態檔案的目錄位置，讓使用者可以透過public資料夾取得靜態資料。
 - __dirname是node.js的關鍵字，代表的是啟動server所在的實體目錄
 - path.join是path模組提供的方法，主要是避免路徑打錯字

Node.js 中的路徑問題

- 絕對路徑: 最常看到的有三種，都是要使用path套件喔
 - __dirname: 回傳執行此js檔案所在資料夾的絕對路徑
 - __filename: 回傳執行此js檔的絕對路徑，包含檔案名稱
 - process.cwd(): 回傳執行node指令時所在的資料夾之絕對路徑
 - ./: 與process.cwd()相同
 - ../:

```
console.log(`__dirname -> ${__dirname}`);  
console.log(`__filename -> ${__filename}`);  
console.log(`process.cwd() -> ${process.cwd()}`);  
console.log(`./ -> ${path.resolve('./')}`);  
console.log(`../ -> ${path.resolve('../')}`);
```

```
__dirname -> C:\Workspace\space_nodejs\todolist  
__filename -> C:\Workspace\space_nodejs\todolist\app.js  
process.cwd() -> C:\Workspace\space_nodejs\todolist  
./ -> C:\Workspace\space_nodejs\todolist  
../ -> C:\Workspace\space_nodejs
```

中介軟體模組有哪些呢？

協力廠商中介軟體

以下是部分的 Express 中介軟體模組：

- [body-parser](#)：即先前的 `express.bodyParser`、`json` 和 `urlencoded`。另請參閱：
 - [body](#)
 - [co-body](#)
 - [raw-body](#)
- [compression](#)：即先前的 `express.compress`
- [connect-image-optimus](#)：可提供最佳影像的 Connect/Express 中介軟體模組。如果可能的話，請將影像切換成 `.webp` 或 `.jxr`。
- [connect-timeout](#)：即先前的 `express.timeout`
- [cookie-parser](#)：即先前的 `express.cookieParser`
- [cookie-session](#)：即先前的 `express.cookieSession`
- [csurf](#)：即先前的 `express.csrf`
- [errorhandler](#)：即先前的 `express.errorHandler`
- [express-debug](#)：低調的開發工具，可在您的應用程式中新增標籤，內含範本變數 (locals)、現行階段作業、有用的要求資料等相關資訊。
- [express-partial-response](#)：Express 中介軟體模組，會根據 `fields` 查詢字串，使用 Google API 的 Partial Response 來濾除 JSON 回應部分。
- [express-session](#)：即先前的 `express.session`
- [express-simple-cdn](#)：Express 中介軟體模組，會將 CDN 用於靜態資產，並支援多部主機（例如：`cdn1.host.com`、`cdn2.host.com`）。
- [express-slash](#)：Express 中介軟體模組，供嚴格看待尾端斜線的人員使用。
- [express-stormpath](#)：Express 中介軟體模組，供使用者執行儲存、鑑別、授權、SSO 和資料安全。
- [express-uncapitalize](#)：中介軟體模組，可將含有大寫的 HTTP 要求重新導向至標準小寫形式。

<https://expressjs.com/zh-tw/resources/middleware.html>

其他模組

- cookie-parser 模組

- 引入cookie-parser模組後，下一步一樣要加入cookie-parser模組

```
app.use(cookieParser());
```

- 可以用來處理並分析網頁中暫存的cookie資料

- router模組

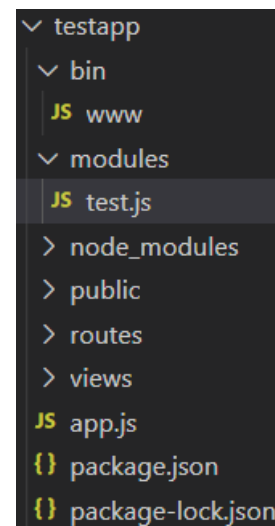
- 同樣透過app.use將路由加入

```
app.use('/', indexRouter);  
app.use('/users', usersRouter);
```

- 第一個參數/此參數為網頁請求的路徑名稱
 - 第二個參數是最一開始引入的路由

模組機制

- 模組(Module)可以讓程式開發者撰寫的程式在其他檔案中引用並使用，可以依照不同的需求或功能劃分不同的模組。
- 後續的新增、修改、刪除和維護程式時就依照不同的模組撰寫。
- 建立模組
 - 預設在testapp專案下有一個node_modules資料夾用來存放此專案相依模組。
 - 避免搞混，在專案下新增一個modules資料夾，並在此資料夾下新增一個test.js檔案，開始建立我們一個設計的模組。



建立一個具有計算功能的模組

- test.js

```
function addition(a, b) {  
    return a+b;  
}  
//將add方法輸出成模組  
module.exports = addition;
```

引入add模組並測試

- 開啟route下的index.js檔案，引入剛建立的test.js中的add模組

```
var express = require('express');
var router = express.Router();
var test = require('../modules/test');

/* GET home page. */
router.get('/', function(req, res, next) {
  // res.render('index', { title: 'Express'
  res.render('index', {title: 'test(2,3)'});
});

module.exports = router;
```

頁面結果

5

Welcome to 5

test.js 加入多種方法

- 同一個模組中可以加入多種方法，例如加入一個乘法的方法

```
function addition(a, b) {  
    return a+b;  
}  
function multiply(a, b){  
    return a*b;  
}  
//將add方法輸出成模組  
module.exports = {  
    add: addition,  
    multi: multiply  
};
```

修改index.js

```
res.render('index', {title: test.multi(2,3)});
```

6

Welcome to 6

不管是app.js或是路由程式，最後一行都是以模組方式輸出，此方式是為了讓其他程式可以使用此模組。

```
module.exports = app;
```

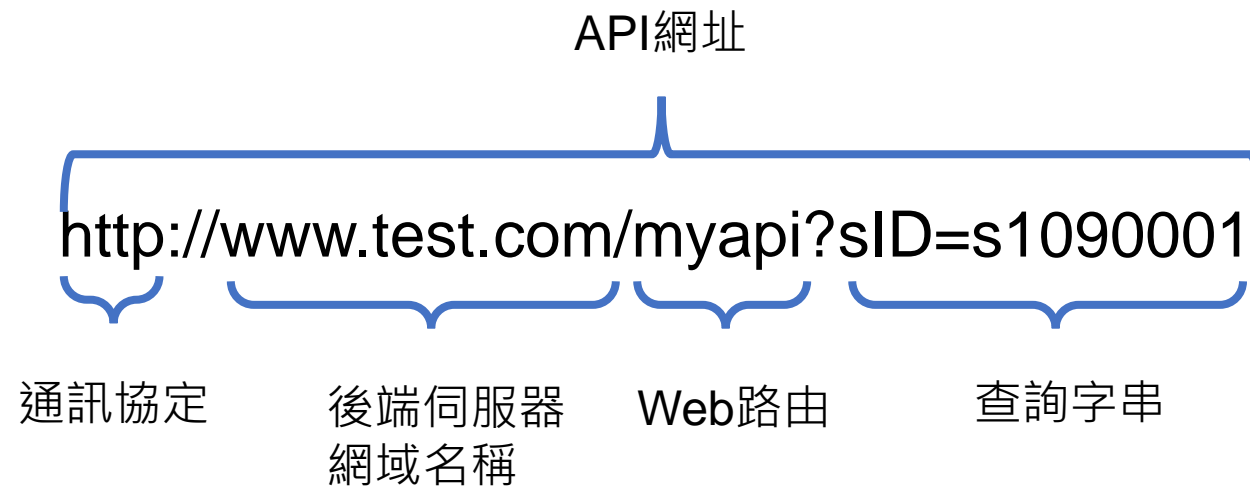
```
module.exports = addition;
```

```
module.exports = router;
```

路由是什麼？跟HTTP有何關係？怎麼使用？

- HTTP request

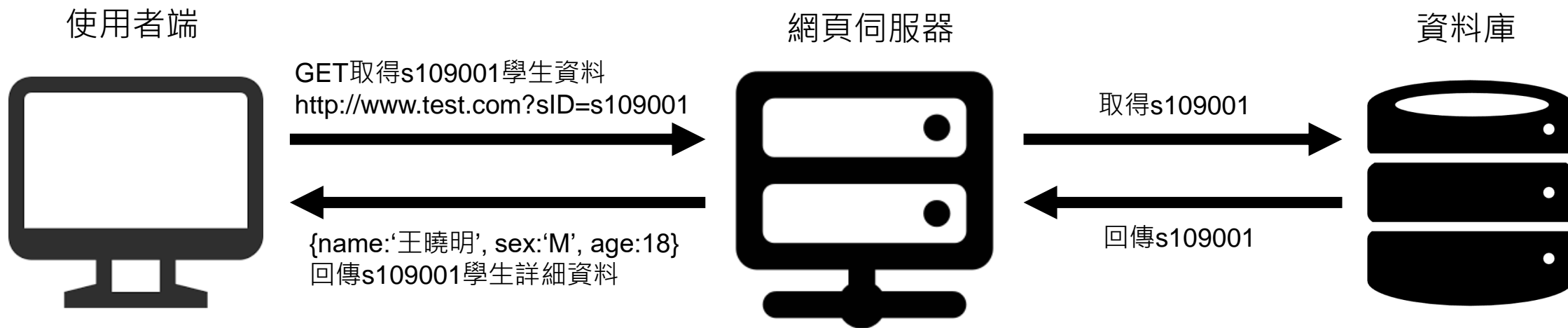
- HTTP是網路資料通訊的協定，主要是讓前端瀏覽器可以使用URL網址來請求後端伺服器，並透過網路進行資料請求與交換。
- URL網址是前後端溝通的橋樑，也是一種API。



HTTP請求方法

- 發送HTTP請求，其格式主要分為Header和Body
 - Header主要紀錄請求的資訊
 - Body放置要傳遞的內容
- 常見的請求方法
 - GET
 - 請求資訊公開的，只有Header沒有Body
 - POST
 - 請求資訊是包起來的，具有Header和Body，較為機密的資訊存放在Body中

GET請求流程



`http://www.test.com?sID=s109001`

?是連接網址與查詢字串的符號

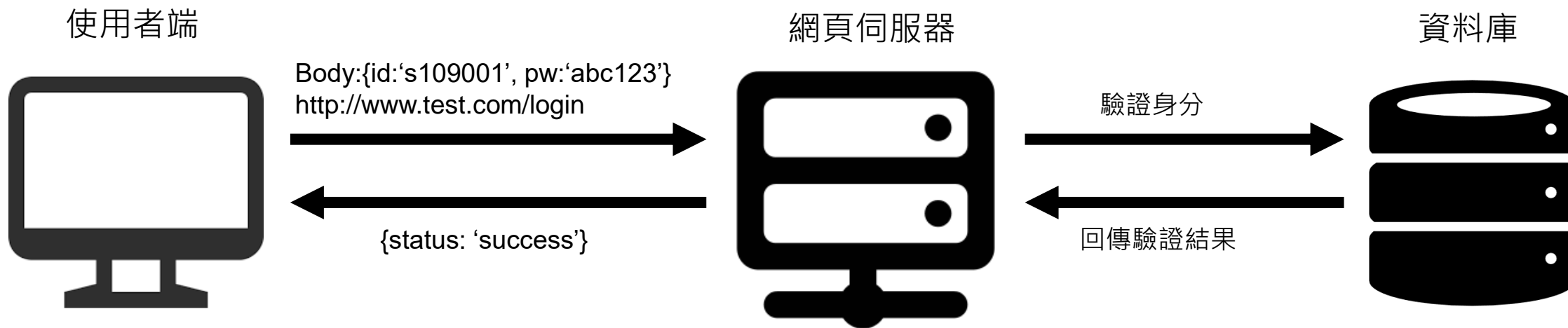
Header設定大致如下

url: 'http://www.test.com?sID=s109001'
Type: 'GET'

多個條件查詢以&符號連結

`http://www.test.com?sAge=18&sSex=F`

POST



- **POST**安全性較高，通常用在修改資料、資料驗證、取得特定資料等等，比如登入portal需要驗證身分，若使用**GET**方式將帳密公開在**URL**上，有安全性較低。

Header和Body設定大致如下
url: 'http://www.test.com/login'
type: 'POST'
data: {id:'s109001', pw:'abc123'}

Route 概念

- 路由在前端與後端互相傳遞資料時扮演引導者的角色，將前端的請求導向正確的函數進行處理，最後將結果回傳至前端顯示。
- 範例：自訂路由myRoute.js(在routes資料夾下新增此檔案)

```
var express = require('express');
var router = express.Router();

router.get('/', function(req, res){
    //codes
});

router.post('/', function(req, res){
    //codes
})
```

引入express模組

在express模組下使用Router()方法新增一個路由器叫做router

在get方法中傳入兩個參數
第1個參數字串是表示路由的路徑
第2個參數則是callback回調函數

```
router.get('/', function(req, res){  
    //codes  
});
```

- 路由路徑表示網頁發送請求時，透過請求的路徑進入方法處理。
- 當前端使用router.get()方法透過API請求，網域名稱後方要加上路由路徑/才能順利取得。

<http://www.test.com/>

- 若將路由路徑設定為/myRouter，則路徑設定如下

```
router.get('/myRouter', function(req, res){  
    //codes  
});
```

<http://www.test.com/myRouter>

- 多層路由設定

```
router.get('/myRouter/about', function(req, res){  
    //codes  
});
```

<http://www.test.com/myRouter/about>

路由路徑加上參數

- 加上參數目的
 - 讓後端依據前端請求的參數進行判斷，回傳所需的資料。
- 使用方式
 - 加上:冒號及參數名稱即可
- 例如

```
router.get('/myRouter:s109001', function(req, res){  
    //codes  
});
```

<http://www.test.com/myRouter/s109001>

參數請求 vs. 搜尋請求

- 兩種方式的概念類似，都是傳遞想要的參數，讓後端可以進行判斷，進而回傳所需要的資料。
- 細微差異僅在於實作過程不同
 - 參數請求API `http://www.test.com/myRouter/s109001`
 - 搜尋請求API `http://www.test.com/myRouter?sID=s109001`
- 不管哪一種方式在後端中如何取得搜尋值或參數，則必須透過第二個參數，也就是回調函數。

回調函數

- 當網頁API的請求符合路由路徑且存在時，就會使用此函數，並且執行此方法中的程式碼。
- 在回調函數中江傳入兩個參數，分別是req請求以及res回覆。
 - req請求(request)：表示請求的物件，包含請求參數和內容等等。
 - 常用的req物件屬性

物件屬性	描述
req.body	取得POST請求中Body的內容
req.query	取得GET或是POST請求中要查詢的內容
req.params	取得GET或是POST請求中，API帶入的參數

req.body

- req.body如何取得POST請求的內容？
- 在app.js中的

```
app.use(express.json());  
app.use(express.urlencoded({ extended: false }));
```
- 此段程式碼就是讓POST可以解析Body中的JSON和urlencoded的資料格式，若沒有此段程式碼，則req.body所取得的資料將會是undefined。
- 例如：
 - 前端發送一個POST請求

```
{sID: 's109001', sPW: 'abc123'}
```
 - 後端接收請求，要使用內容則透過req.body

```
req.body.sID; //s109001  
req.body.sPW; //abc123
```

req.query

- 例如：

- 前端要查詢s109001學生的詳細資料

`http://www.test.com/myRouter?sID=s109001`

- 後端要取得學生id進一步查詢該位學生，則透過req.query

```
router.get('/myRouter', function(req, res){  
    console.log(req.query.sID); //s109001  
});
```


req.params

- API帶入參數為s109001

`http://www.test.com/myRouter/s109001`

- 後端要取得則使用req.params方式。

```
router.get('/myRouter:sID', function(req, res){  
  console.log(req.params.sID); //s109001  
});
```

res

- **res回覆(response)**：表示回覆的物件，用來回應請求，以下為常用的**res**物件屬性。

物件屬性	描述
res.send()	回傳回覆訊息，格式可以是字串、陣列、物件或JSON
res.json()	回傳回覆訊息，格式只能是JSON

- **res.send()**

```
res.send({"Hi, how are you?});  
res.send([1,2,3]);
```

- **res.json()**

```
res.json({"sID": "s109001", "sAge": 18});
```

概念補充 middleware 中介軟體

- Express is a routing and middleware web framework that has minimal functionality of its own: An Express application is essentially a series of middleware function calls.
 - 中介軟體扮演資料庫與應用程式之間的溝通橋樑，透過不同種類中介軟體可以讓資料傳輸更加有效率，同時可以依照需求對資料進行不同的處理。
- 類型
 - Application-level middleware (`app.use()`、`app.get()`、`app.post()`)
 - Router-level middleware (`express.Router()`)
 - Error-handling middleware (`app.use(function(err, req, res, next))`)
 - Built-in middleware (`express.static('public')`)
 - Third-party middleware (`const bodyParser = require('body-parser')`)

範例：實作自訂路由

(myRoute.js)

```
var express = require('express');
var router = express.Router();

router.get('/intro', function(req, res){
  res.send("Hello Node.js!!!");
});

module.exports = router;
```

使用get()方法設定路由的路徑為/intro
進入後使用res.send()方法將字串送出

(app.js)

```
var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');
var myRouter = require('./routes/myRoute'); //引入
```

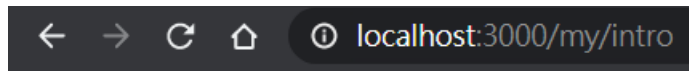
引入myRoute

```
app.use('/', indexRouter);
app.use('/users', usersRouter);
app.use('/my', myRouter);
```

將myRouter加入app應用中，路徑參數設定為/my

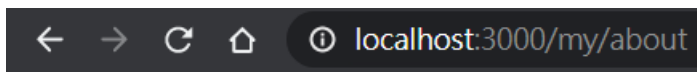
頁面顯示

<http://localhost:3000/my/intro>



Hello Node.js!!!

- 練習製作一個/my/about，在頁面上顯示自己的姓名



我是王曉明!!!

POST請求測試

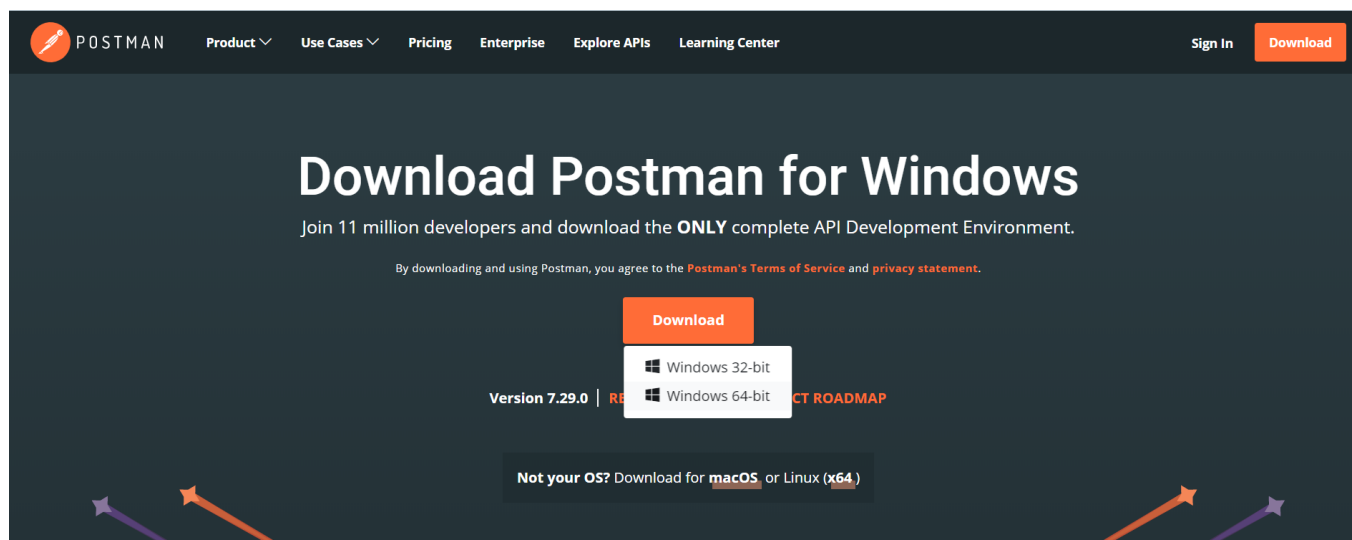
- 在myRoute.js的GET方法下方新增一個POST方法

```
router.post('/sum', function(req, res){  
  var x = parseInt(req.body.num1);  
  var y = parseInt(req.body.num2);  
  var sum = x + y;  
  res.json({mySum: sum});  
})
```

- 當使用者進入/sum路徑時，會傳送兩個參數計算總合後，以json方式送出計算結果。

使用Postman測試

- 瀏覽器可以測試剛剛製作的GET請求，但對於POST請求無法進行測試。
- 到Postman網站下載桌面應用程式





New

Import

Runner



My Workspace ▾



Invite



Sign In



Filter

History

Collections

APIs

☐ Save Responses

You haven't sent any requests

Any request you send in this workspace will appear here.



Show me how

Launchpad



No Environment ▾



Good morning!

Let's start the day off right. Use Launchpad to start something new, pick up where you left off, or explore some resources to help you master Postman.

Start something new

[GET Create a request](#)[Create a collection ▾](#)[Create an environment](#)[Create an API](#)[View More](#)

Customize

☐ Dark mode☒ Enable Launchpad[More settings](#)

What's new with Postman

Resources

Events

Tell us how you use Postman and you could win a \$50 Amazon Gift Card!

Take this short survey on how you use Postman and you could win a \$50 Amazon gift card (gift card drawing for US residents only).

[Take the Survey](#)

Securely Using API Keys in Postman

To help you use API keys as effectively as possible, this blog post walks you through some common pitfalls (and how to avoid them)...

[Read the Blog Post](#)

API 101: Learning with Open Access SOAP APIs

Unless your organization is brand-new, chances are you're developing and maintaining legacy codebases that rely on SOAP. If...

[Read the Blog Post](#)[Show More](#)



New

Import

Runner



My Workspace ▾

Invite



Sign In



Filter

History

Collections

APIs

+ New API



Sign in to create APIs

APIs define related collections and environments under a consistent schema.

Sign in to create APIs

Launchpad

POST Untitled Request



No Environment ▾



Untitled Request

Comments 0

POST



Enter request URL

Send



Save



GET

POST



PUT

PATCH

DELETE

COPY

HEAD

OPTIONS

LINK

UNLINK

PURGE

LOCK

UNLOCK

PROPFIND

VIEW

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies Code

VALUE

DESCRIPTION



Bulk Edit

Value

Description



Hit Send to get a response



Find and Replace



Console



Bootcamp





New

Import

Runner



My Workspace ▾

Invite



Sign In



Filter

History

Collections

APIs

+ New API



Sign in to create APIs

APIs define related collections and environments under a consistent schema.

Sign in to create APIs

Launchpad

POST localhost:3000/my/sum



No Environment ▾



Untitled Request

Comments 0

POST ▾

localhost:3000/my/sum

Send ▾

Save ▾

Params

Authorization

Headers (7)

Body

Pre-request Script

Tests

Settings

Cookies Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Response ▾



Hit Send to get a response



Postman

File Edit View Help

New Import Runner +

My Workspace Invite

Filter

History Collections **APIs**

+ New API

Sign in to create APIs

APIs define related collections and environments under a consistent schema.

Sign in to create APIs

Launchpad POST localhost:3000/my/sum

Untitled Request

Comments 0

POST localhost:3000/my/sum

Send Save

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

none form-data **x-www-form-urlencoded** raw binary GraphQL

	KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/>	num1	1	
<input checked="" type="checkbox"/>	num2	2	
	Key	Value	Description

Body Cookies Headers (6) Test Results

Status: 200 OK Time: 5 ms Size: 222 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "mySum": 3
3 }
```

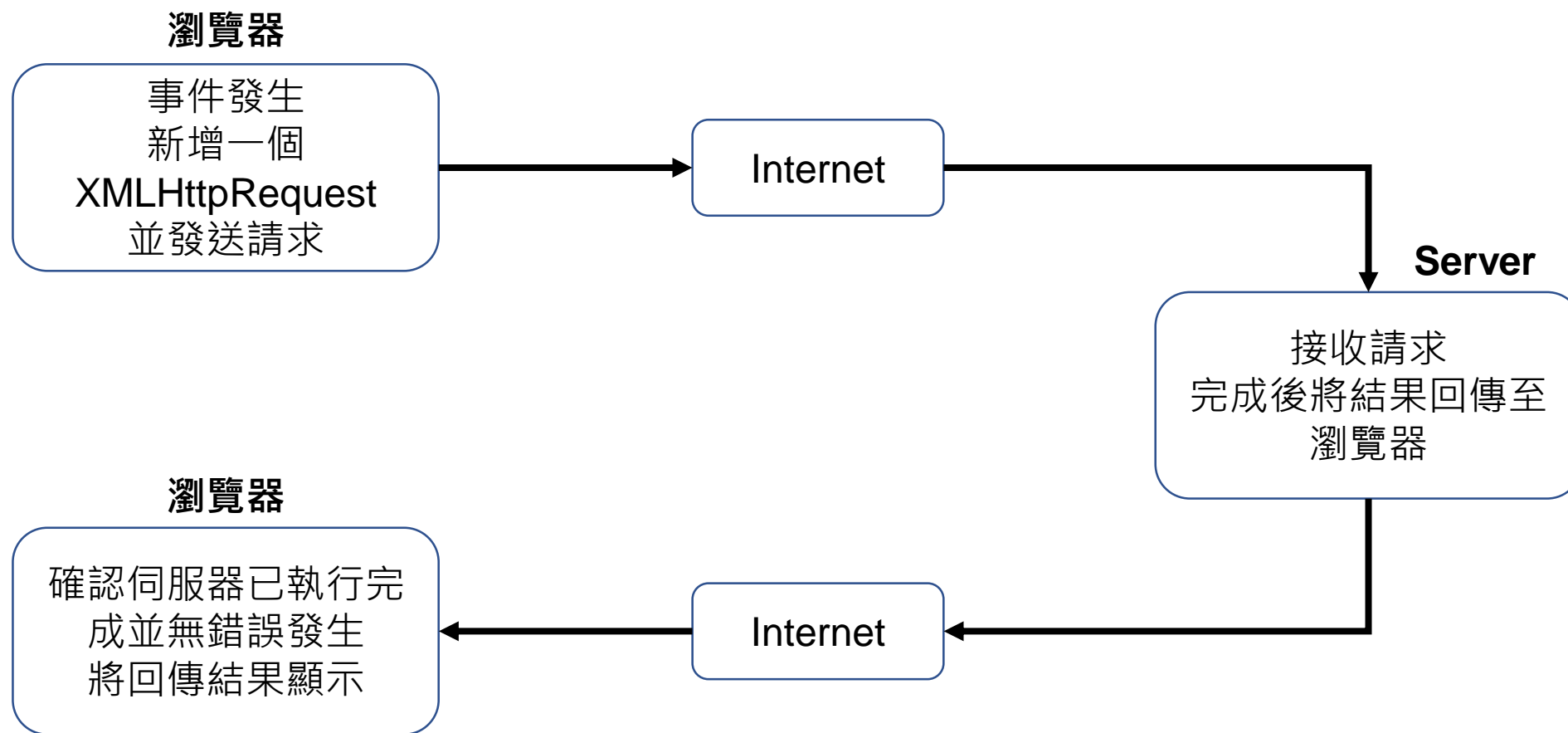
Find and Replace Console

Bootcamp

如何讓前端與後端進行資料傳輸？

- 一般而言，過去前端執行資料傳輸給後端時時，大部分需要將整個網頁內容傳至伺服器端，若傳送僅有個位或少數欄位時，這樣的方式會特別浪費時間。
- 我們將使用**JS**進行非同步傳輸，不需要將整個頁面傳送至伺服器端，便可透過**JS**動態更新頁面，以實現及時資料傳輸的效果，這樣的方式稱之為**AJAX (Asynchronous JavaScript and XML)**
- **AJAX**主要用來即時傳送資料到後端並處理接收的資料，同時不會影響前端使用者的操作，也不需要更新頁面的情況下，使用者可以得到後端回傳的結果。

AJAX資料傳輸過程



XMLHttpRequest是撰寫AJAX的核心角色，用來送出Http請求的物件。

AJAX語法

- 右邊程式碼是一個\$.ajax的語法範例，從中可見\$.ajax傳入的是一個物件，此物件中包含許多屬性與方法。

```
$.ajax(  
    url: "API",  
    contentType: "application/x-www-form-urlencoded",  
    data: "DATA",  
    type: "POST",  
    datatype: "資料型態",  
    success: function(res){  
  
    },  
    error: function(){  
  
    }  
);
```

\$.ajax()傳入物件的屬性與方法

參數名稱	說明
url	指定要進行呼叫的位址
contentType	前端傳送至後端的資料型態 預設值: application/x-www-form-urlencoded application/json、text/xml
data	傳送至後端的資料，相對應HTTP POST請求中的Body，因此此參數對GET請求是無效的。
dataType	後端回傳前端的資料型態(xml、html、json、text)
type	請求方式，主要為GET、POST
success	請求成功時所執行的函數
error	請求失敗時所執行的函數

以上是常用的參數介紹，並非每個參數都必須填寫。

AJAX 簡易寫法

- GET請求

```
var API="http://www.test.com";  
$.get(API, function(data, status){  
    //codes  
})
```

欲存取的網站儲存至API變數

使用\$.get方法傳入API變數，去向後端請求資料，在此不論成功與否，皆會執行function中的程式碼。

- POST請求

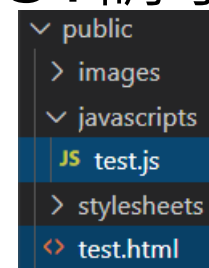
```
var API="http://www.test.com";  
var payload={id:'John', pw:'abc123'};  
$.post(API, payload, function(data, status){  
    //codes  
})
```

建立一個帳密的物件payload

第1個參數傳入API變數
第2個參數傳入要傳送給後端的Body資料
第3個參數為完成請求後要執行的程式碼

如何整合前後端？

- 範例目標：建立一個向後端發送加總計算的POST請求。
- 第一步：建立一個發送的頁面
 - 在public/javascripts資料夾下新增一個test.js
 - 在public資料夾下新增一個test.html



```
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<script src="javascripts/test.js"></script>
```

```
<body>
  <h1>前後端整合範例</h1>
  <div>num1:<input type="text" id="num1"></div>
  <div>num2:<input type="text" id="num2"></div>
  <div>使用POST請求
    <input type="button" id="btnSumPOST" value="POST加總">
    <input type="text" id="resPOST" placeholder="POST結果">
  </div>
</body>
```

前後端整合範例

num1:

num2:

test.js 撰寫加總按鈕程式

前後端整合範例

num1:

num2:

test.js

```
$("#btnSumPOST").click(function(){  
    var x = $("#num1").val();  
    var y = $("#num2").val();  
    var API = "http://localhost:3000/my/sum";  
    var data = {num1: x, num2: y};  
    $.post(API, data, function(res){  
        $("#resPOST").val(res.mySum);  
    });  
});
```

myRoute.js

```
router.post('/sum', function(req, res){  
    var x = parseInt(req.body.num1);  
    var y = parseInt(req.body.num2);  
    var sum = x + y;  
    res.json({mySum: sum});  
})
```

前後端整合範例

num1: 1

num2: 2

使用GET()方法改寫

test.html

```
<div>使用GET請求&nbsp;&nbsp;&nbsp;&nbsp;&  
    <input type="button" id="btnSumGET" value="GET加總">  
    <input type="text" id="resGET" placeholder="GET結果">  
</div>
```

test.js

```
$("#btnSumGET").click(function(){
    var x = $("#num1").val();
    var y = $("#num2").val();
    var API = "http://localhost:3000/my/sum?num1="+x+"&num2="+y;
    $.get(API, function(res){
        $("#resGET").val(res.mySum);
    })
})
```

myRoute.js

```
router.get('/sum', function(req, res){
    var x = parseInt(req.query.num1);
    var y = parseInt(req.query.num2);
    var sum = x + y;
    res.json({mySum: sum});
})
```

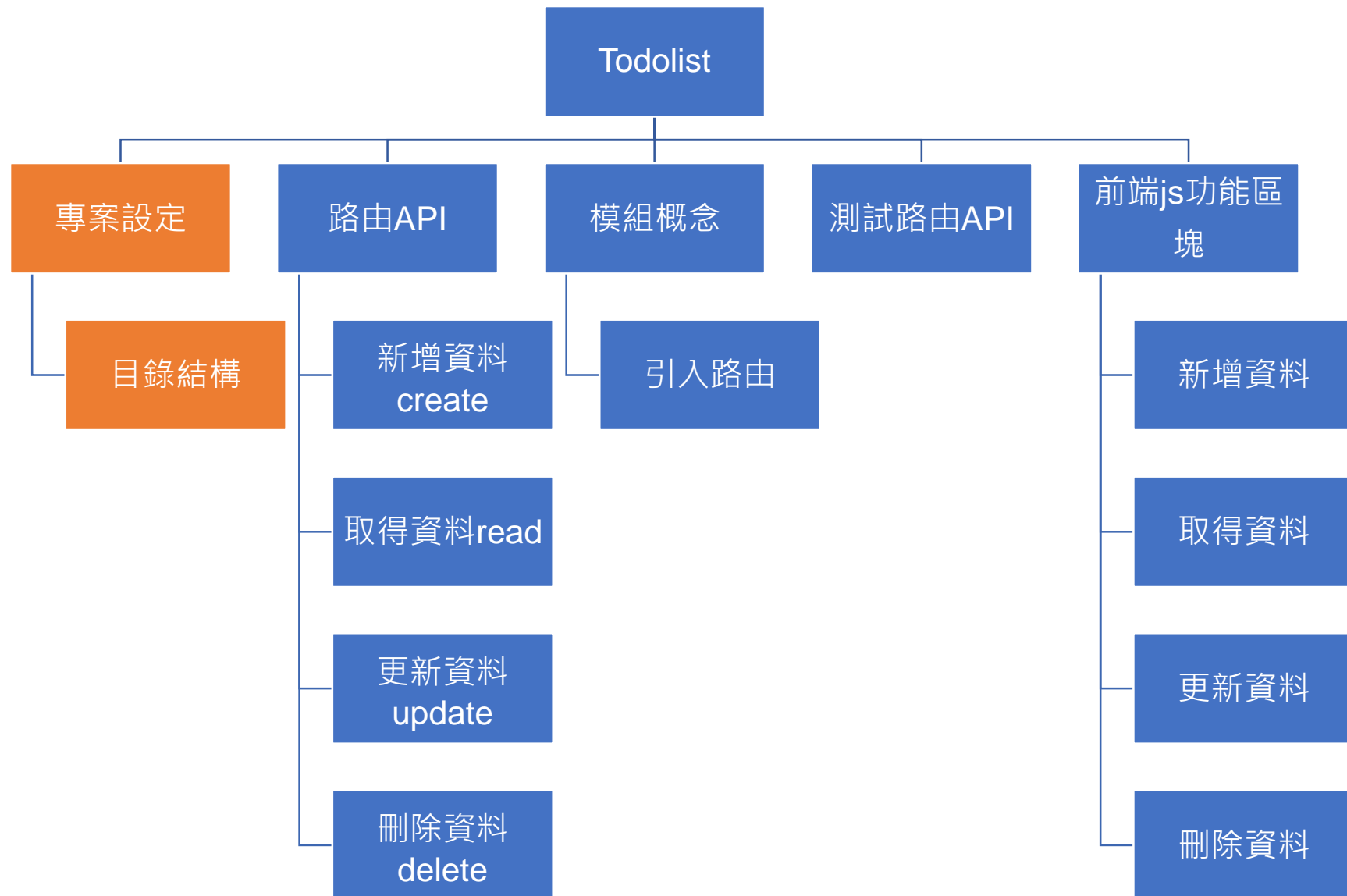
GET()畫面結果

前後端整合範例

num1:	1	
num2:	3	
使用POST請求	POST加總	4
使用GET請求	GET加總	4

TodoList小專案

- 前述
 - 上一次的小專案僅針對前端部分進行設計，我們將延續上次的小專案，加入本次所學習的`node.js`和`express`，實作新增、修改、刪除代辦事項，以及改變狀態的API，並且取代原本在前端的相關操作。



Todolist 專案 - 專案設定

- 建立TodoList專案
 - 指令express todolist
 - 進入todolist資料夾下
 - 安裝所需模組

```
C:\node> express todolist
```

```
C:\node> cd .\todolist\
```

```
C:\node\todolist> npm install
```

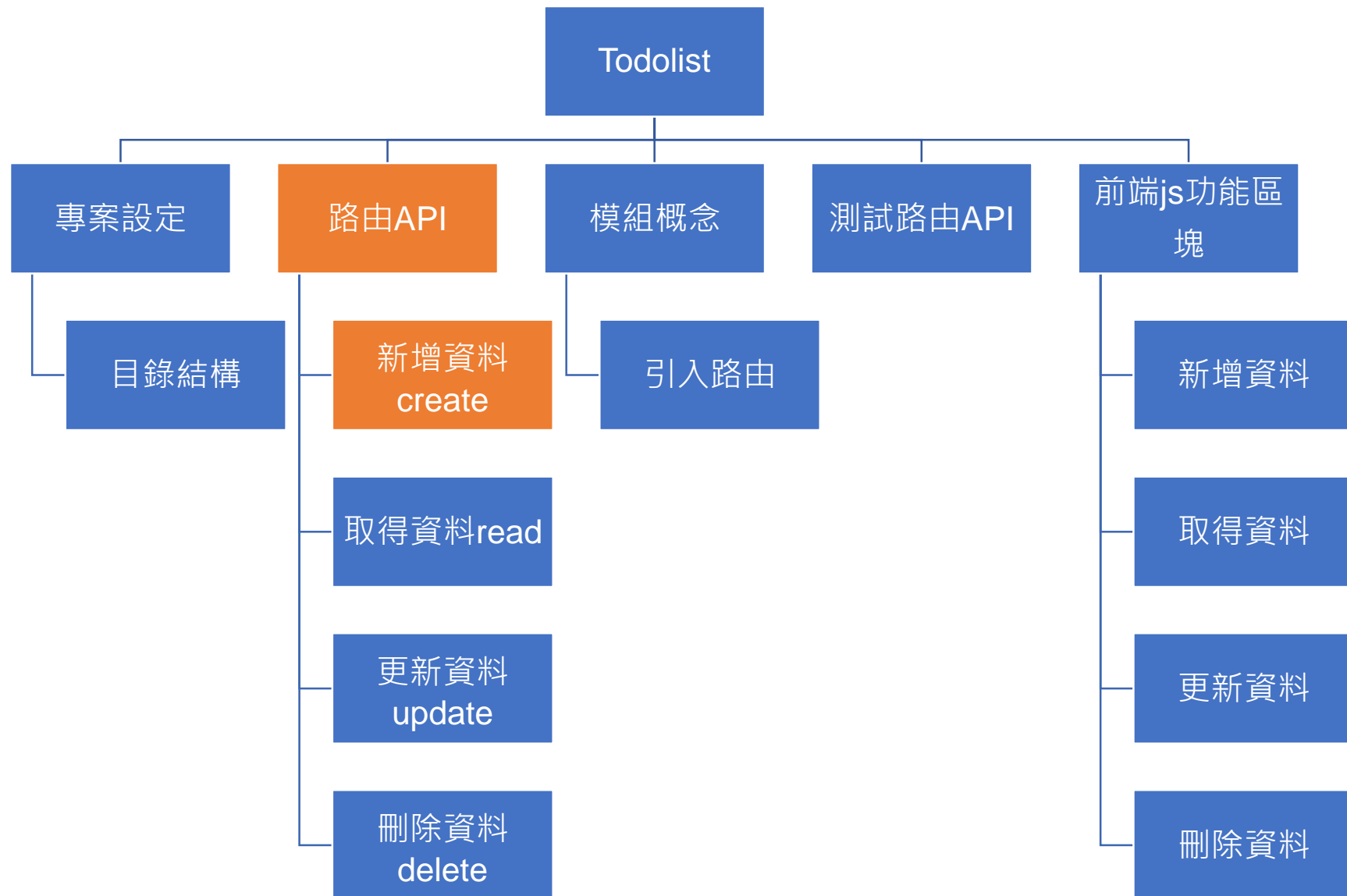
Todolist 專案目錄

```
▼ todolist
  > bin
  > node_modules
  ▼ public
    > images
    ▼ javascripts
      JS todolist.js
    ▼ stylesheets
      # style.css
      <> todolist.html
  ▼ routes
    JS api.js
    JS index.js
    JS users.js
  > views
  JS app.js
  {} package.json
  {} package-lock.json
```

新增todolist的JS檔案

新增todolist網頁

新增api

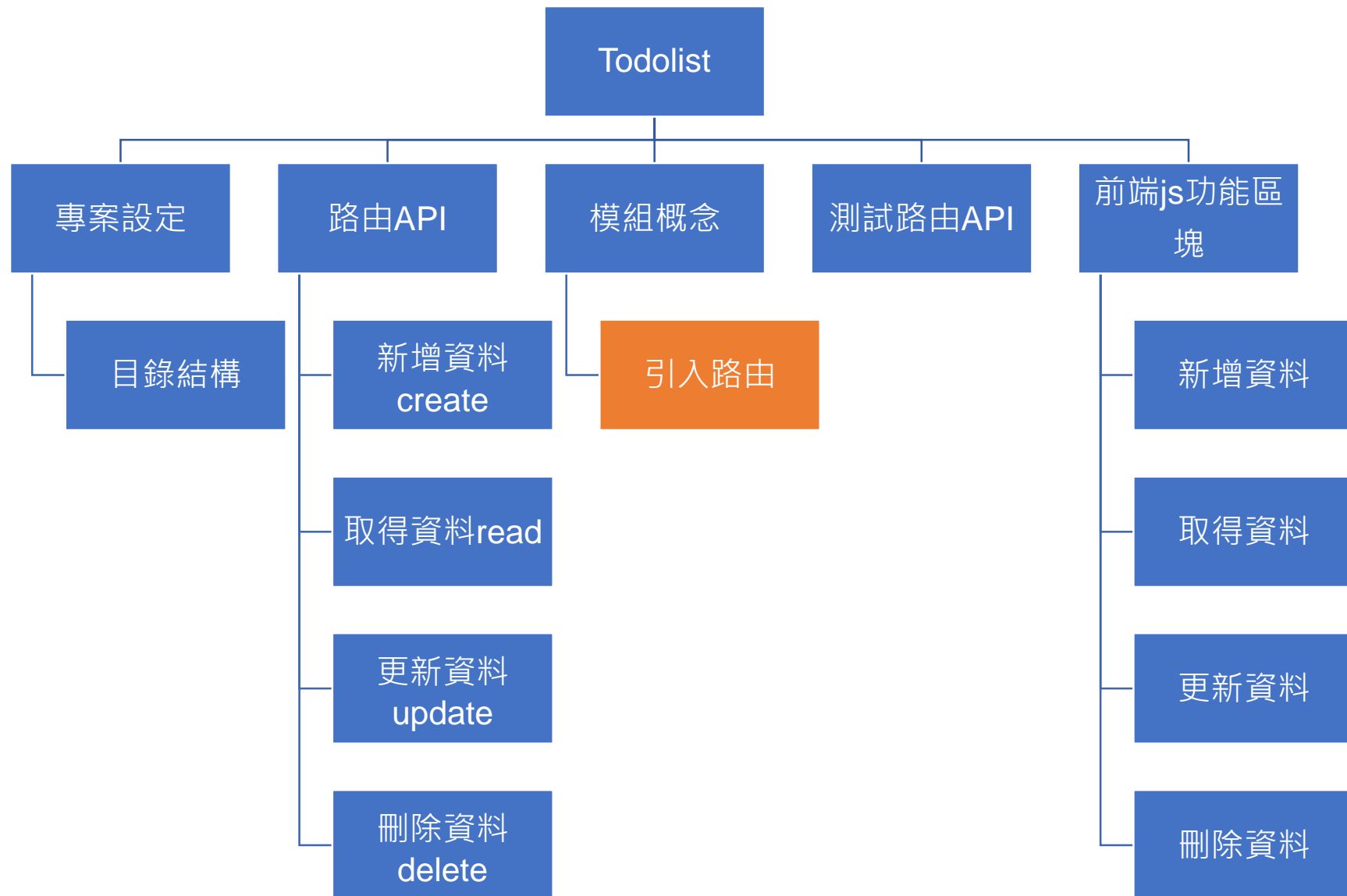


Routes – Create

新增待辦事項 /api.js

- 此api.js路由檔案將處理todolist待辦事項的取得、新增、修改刪除以及狀態更改等等。

```
var express = require('express');
var router = express.Router();
var allList = []; //存放所有待辦事項
var id = 1; //紀錄待辦事項的索引值
router.post('/addList', function(req, res){
  var newTodo = {
    "id": id,
    "title": req.body.title,
    "msg": req.body.msg,
    "status": false
  };
  allList.push(newTodo);
  id++;
  res.json({"status":0, "msg":"success", "data":newTodo});
});
module.exports = router;
```



app.js引入api.js

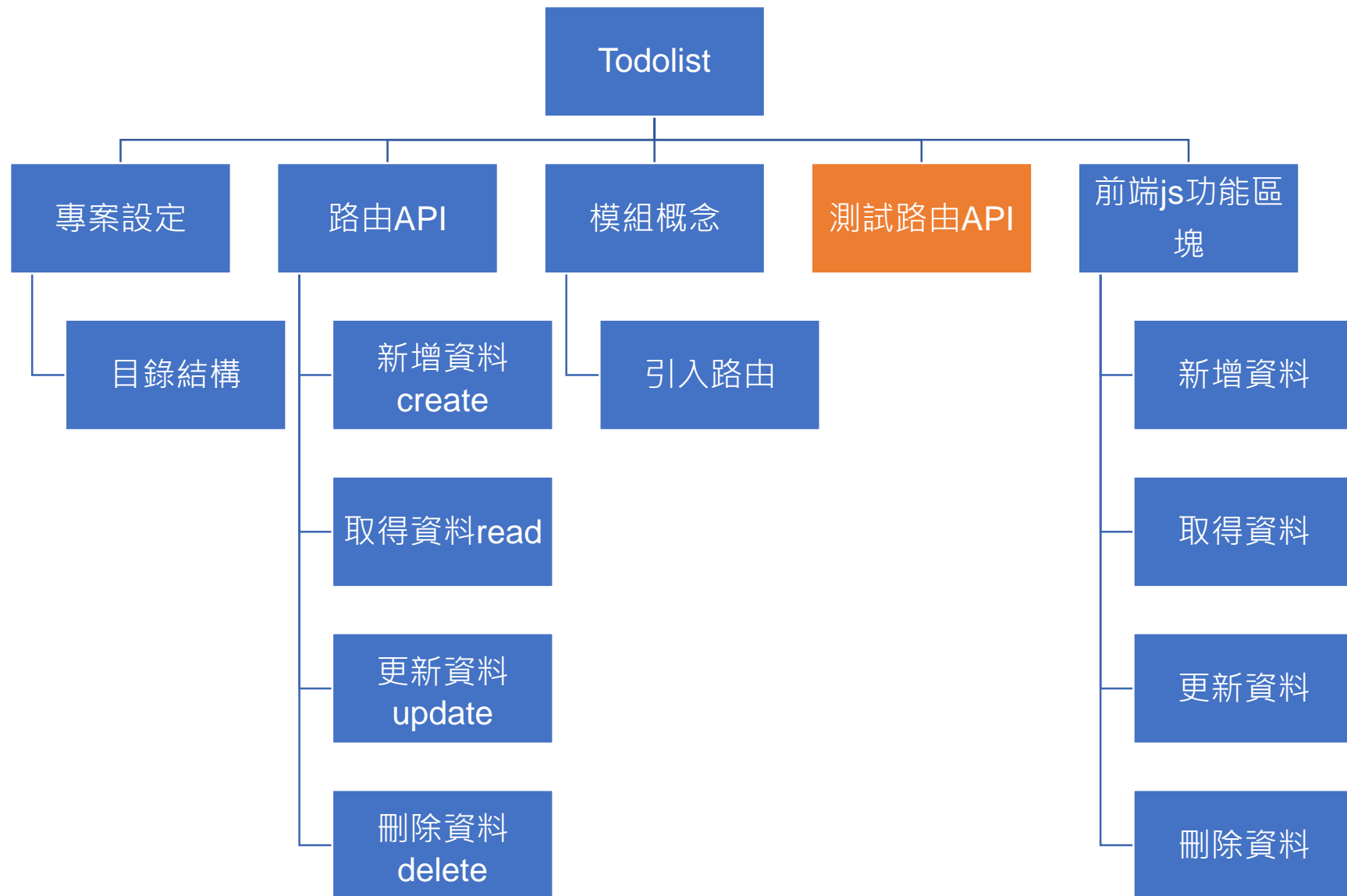
```
var indexRouter = require('./routes/index');  
var usersRouter = require('./routes/users');  
var api = require('./routes/api');
```

引入剛剛寫好的api.js

```
app.use('/', indexRouter);  
app.use('/users', usersRouter);  
app.use('/api', api);  
app.use('/public', express.static('public'));
```

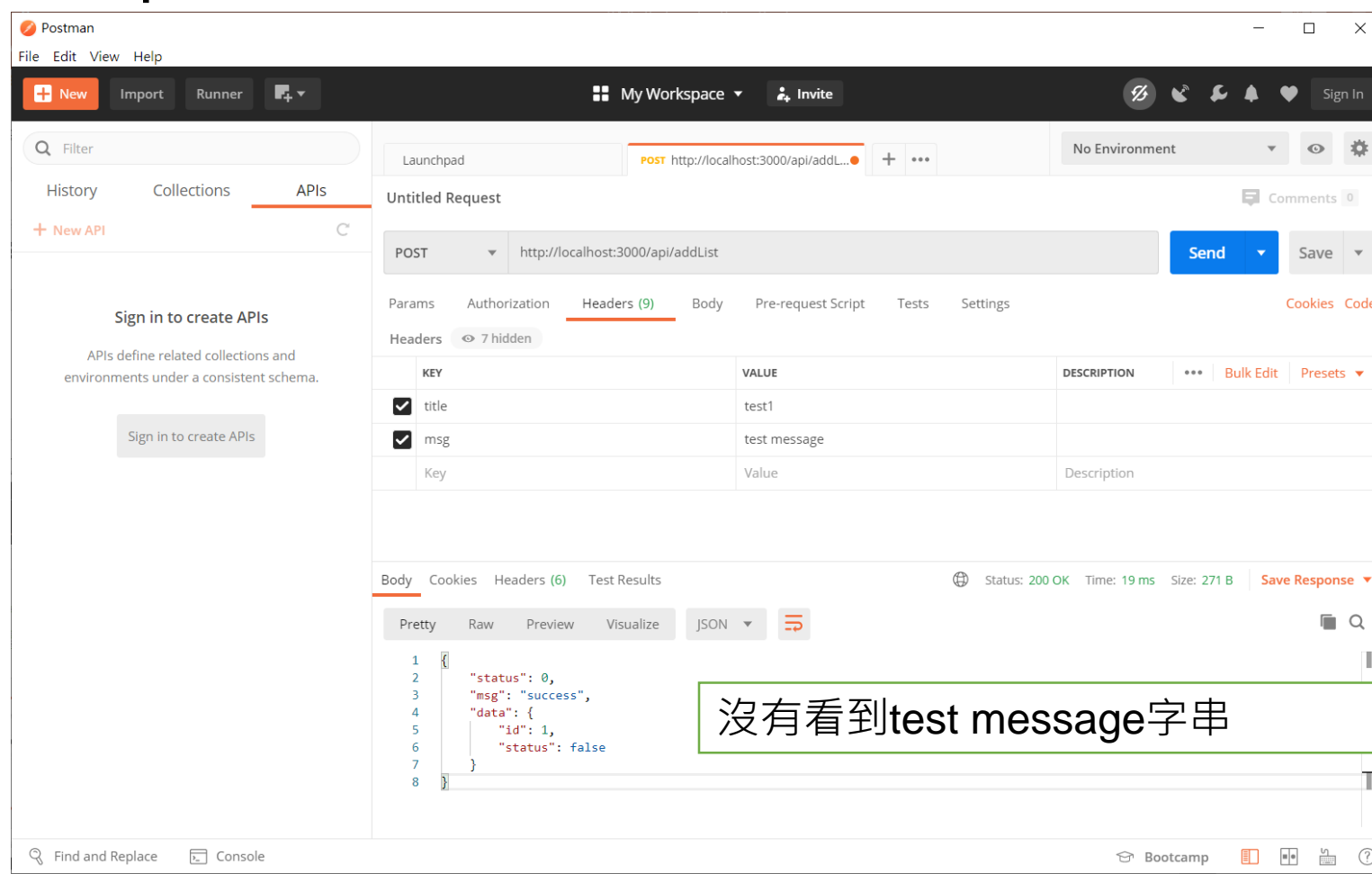
設定api路由路徑

使用express.static將public此資料夾對外開放，讓其他程式可以存取相關靜態資源



使用postman測試剛剛撰寫的路由程式

- 記得先啟動npm start



移置到body

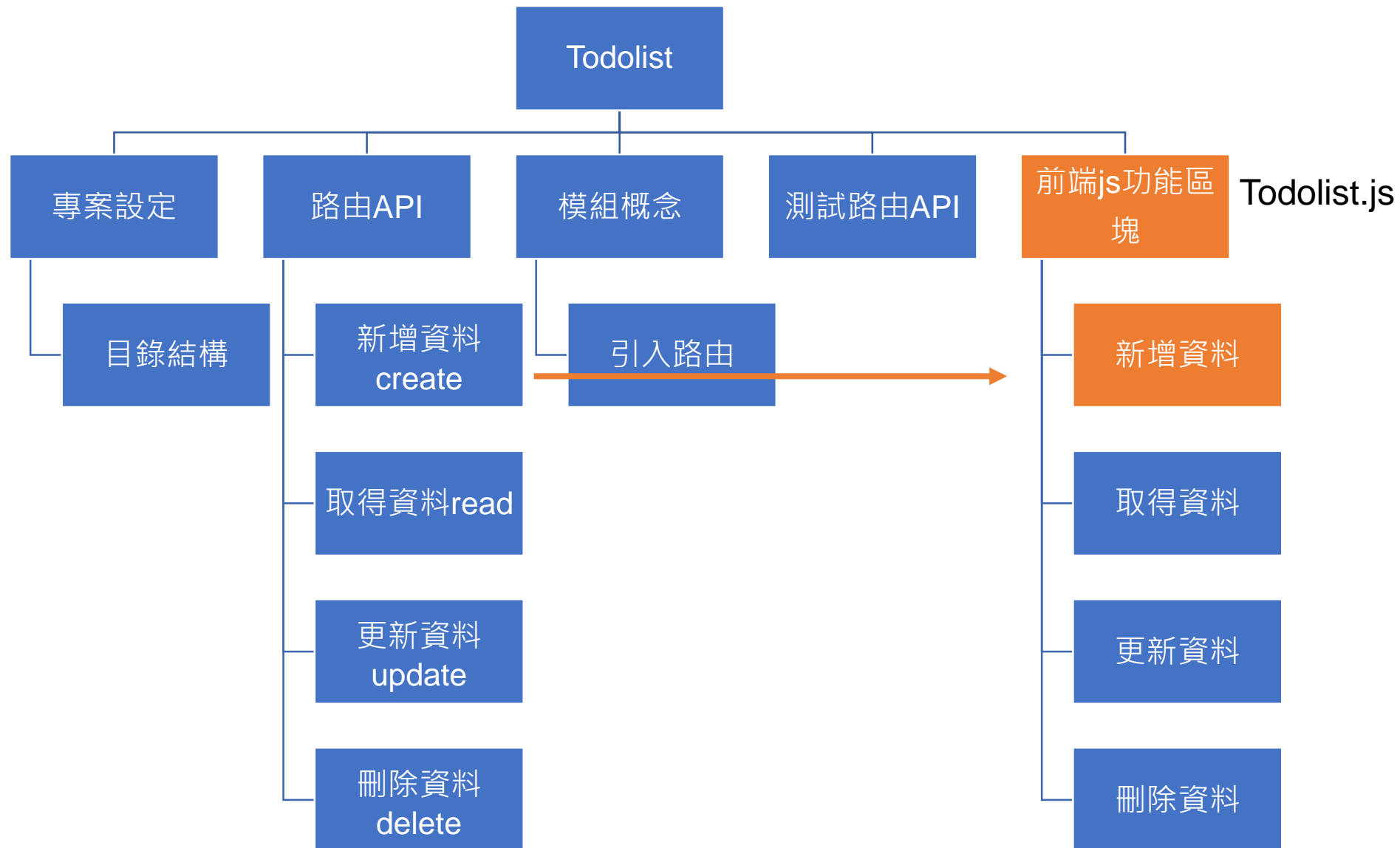
The screenshot displays the Postman application interface. The top navigation bar includes the Postman logo, a menu (File, Edit, View, Help), and buttons for 'New', 'Import', 'Runner', and 'My Workspace'. The left sidebar shows 'History', 'Collections', and 'APIs' tabs, with a 'Sign in to create APIs' prompt. The main workspace is titled 'Untitled Request' and shows a POST request to 'http://localhost:3000/api/addList'. The 'Headers' tab is active, displaying a table with headers 'title' and 'msg'. The 'Body' tab is also visible, showing a JSON response.

Headers

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> title	test1	
<input checked="" type="checkbox"/> msg	test message	
Key	Value	Description

Body

```
1 {
2   "status": 0,
3   "msg": "success",
4   "data": {
5     "id": 1,
6     "title": "test",
7     "content": "test message",
8     "status": false
9   }
10 }
```

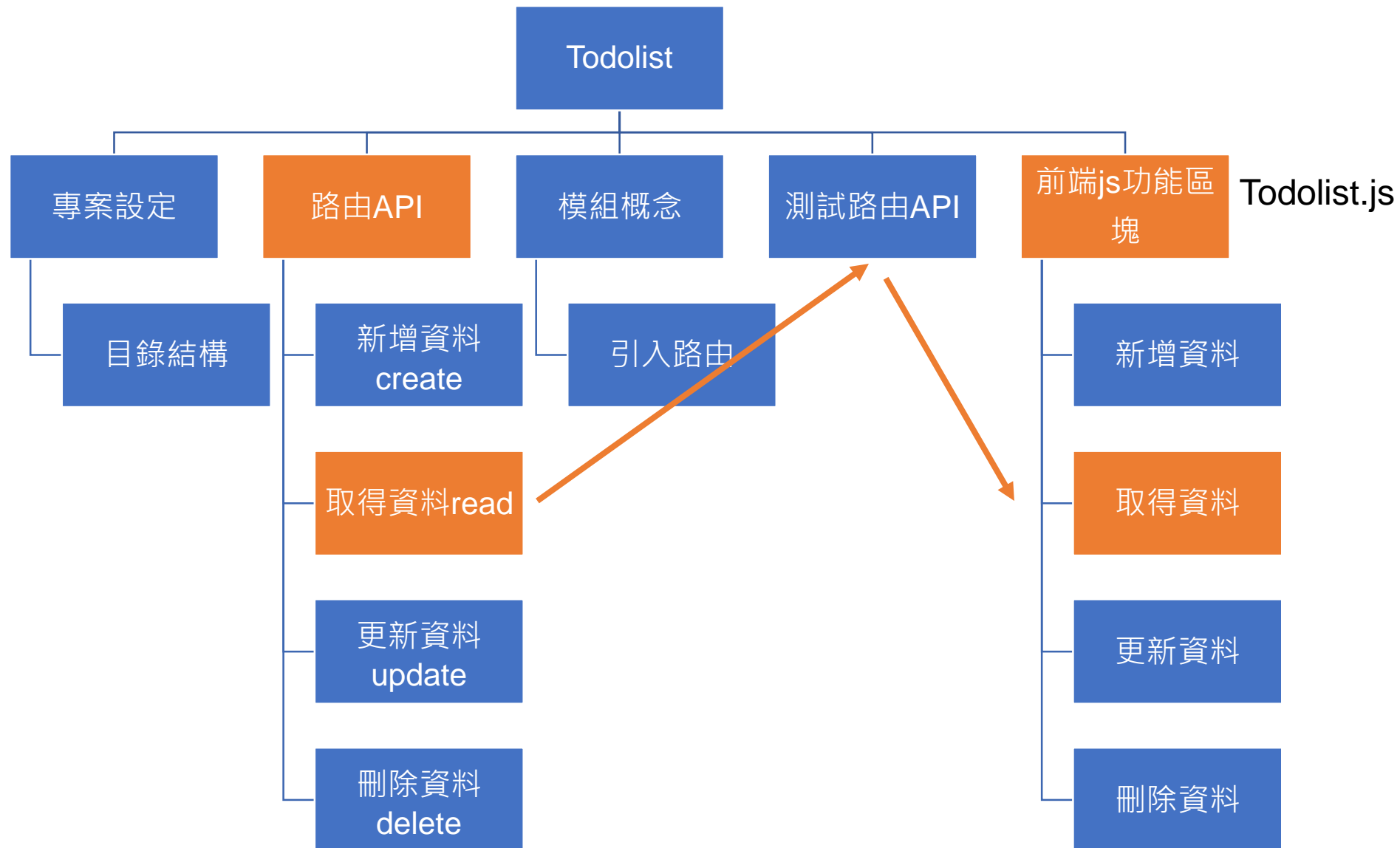


Todolist.js開始撰寫addList方法

```
var todolist = [];  
var id = 1;  
function addList(){  
    var title = $('#title').val();  
    var msg = $('#msg').val();  
    if(title == "" || msg == "") {  
        alert("請輸入標題和內容!");  
    } else {  
        var api = "http://localhost:3000/api/addList";  
        var data = {"title":title, "msg":msg};  
        $.post(api, data, function(res){  
            newList(res.data);  
            $('#title').val('');  
            $('#msg').val('');  
        });  
    }  
}
```

使用POST方法呼叫addList API，並且將title和msg傳給後端處理

newList與之前使用方式一樣，複製到此程式區塊下方



Routes – Read

新增待辦事項 /api.js 顯示待辦事項頁面(api.js)

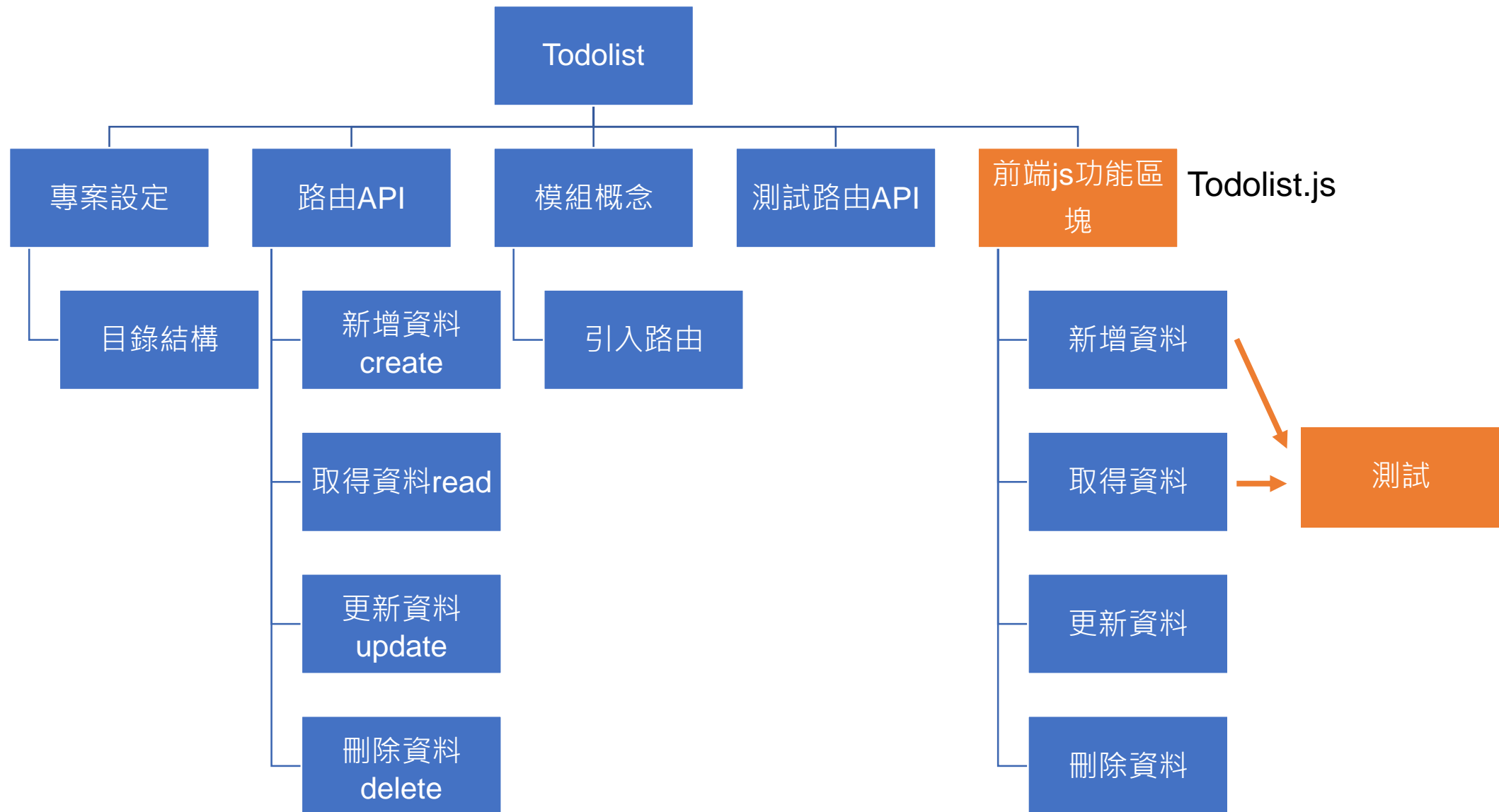
- 在api.js中建立取得所有待辦事項API的路由以及方法
 - 定義路由為GET方法，設定路徑為getList，最後以json格式回傳。

```
router.get('/getList', function(req, res){  
    res.json(allList);  
})
```

前端 todolist.js

- 從路由getList取得回傳JSON資料後，在todolist.js要處理這些待辦事項

```
getList();  
function getList(){  
    var api="http://localhost:3000/api/getList";  
    $.get(api, function(data, status){  
        for(var i=0; i<data.length; i++){  
            newList(data[i]);  
        }  
    })  
}
```



執行專案npm start

- 網址列輸入<http://localhost:3000/todolist.html>

TodoList

標題

內容

確認

<input type="checkbox"/>	標題	待辦事項	修改	更新	刪除
--------------------------	----	------	----	----	----

新增一筆待辦事項測試

TodoList

標題

內容

確認

<input type="checkbox"/>	標題	待辦事項	修改	更新	刪除
<input type="checkbox"/>	繳交報告	網頁期中企劃案報告書	修改	刪除	

概念整理

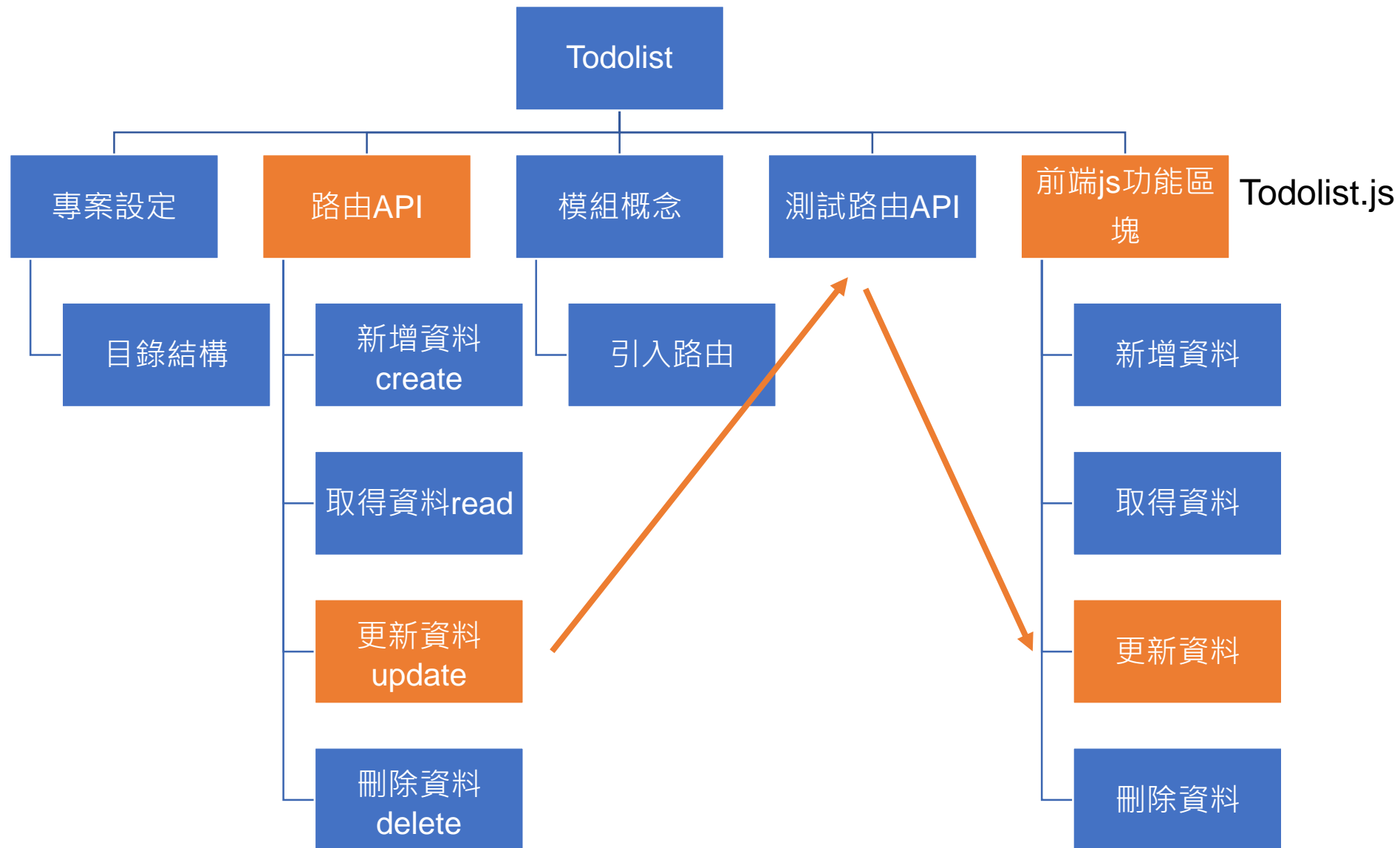
```
let todolist = [];  
let id = 1;  
function addList(){  
  let title = $('#title').val();  
  let msg = $('#msg').val();  
  if(title == "" || msg == "") {  
    alert("請輸入標題和內容!");  
  } else {  
    let newTodo = {  
      'id': id,  
      'title': title,  
      'msg': msg,  
      'status': false  
    };  
    todolist.push(newTodo);  
    newList(newTodo);  
    id++;  
    $('#title').val('');  
    $('#msg').val('');  
  }  
}  
function newList(data) {  
  let status = (data.status)? "checked":"";  
  let content =  
    `<div class="input-group mb-3" id="${data.id}">  
      //...省略  
    </div>`  
  $('#.container').append(content);  
}
```

api.js

```
var express = require('express');  
var router = express.Router();  
var allList = []; //存放所有待辦事項  
var id = 1; //紀錄待辦事項的索引值  
router.post('/addList', function(req, res){  
  var newTodo = {  
    "id": id,  
    "title": req.body.title,  
    "msg": req.body.msg,  
    "status": false  
  };  
  allList.push(newTodo);  
  id++;  
  res.json({"status":0, "msg":"success", "data":newTodo});  
});  
router.get('/getList', function(req, res){  
  res.json(allList);  
})
```

todolist.js

```
var todolist = [];  
var id = 1;  
getList();  
function addList(){  
  //...省略  
  $.post(api, data, function(res){  
    newList(res.data)  
    //...省略  
  });  
}  
function getList(){  
  //...省略  
}  
function newList(data) {  
  //...省略  
}
```



Routes – Update

修改待辦事項 api.js

```
//修改與更新待辦事項
router.post('/updateList', function(req, res){
  var id = req.body.id;
  var index = allList.findIndex(element => element.id == id);
  allList[index].title = req.body.title;
  allList[index].msg = req.body.msg;
  res.json({"status":0, "msg":"success"});
})
```

修改待辦事項 todolist.js

- 新增function editList(id)，此區塊程式碼與先前相同
- 新增updateList(id)：用來確認修改好的待辦事項

```
//更新待辦事項
function updateList(id){
    var title = $("#title"+id).val();
    var msg = $("#msg"+id).val();
    var API = "http://localhost:3000/api/updateList";
    var data = {"id":id, "title":title, "msg":msg};
    $.post(API, data, function(res){
        if(res.status ==0){
            $('#btnEdit'+id).removeClass("d-none");
            $('#btnRemove'+id).removeClass("d-none");
            $('#btnUpdate'+id).addClass("d-none");
            $('#title'+id).attr("readonly", true);
            $('#msg'+id).attr("readonly", true);
        }
    });
}
```

測試進入修改狀態是否正確

TodoList

標題

標題

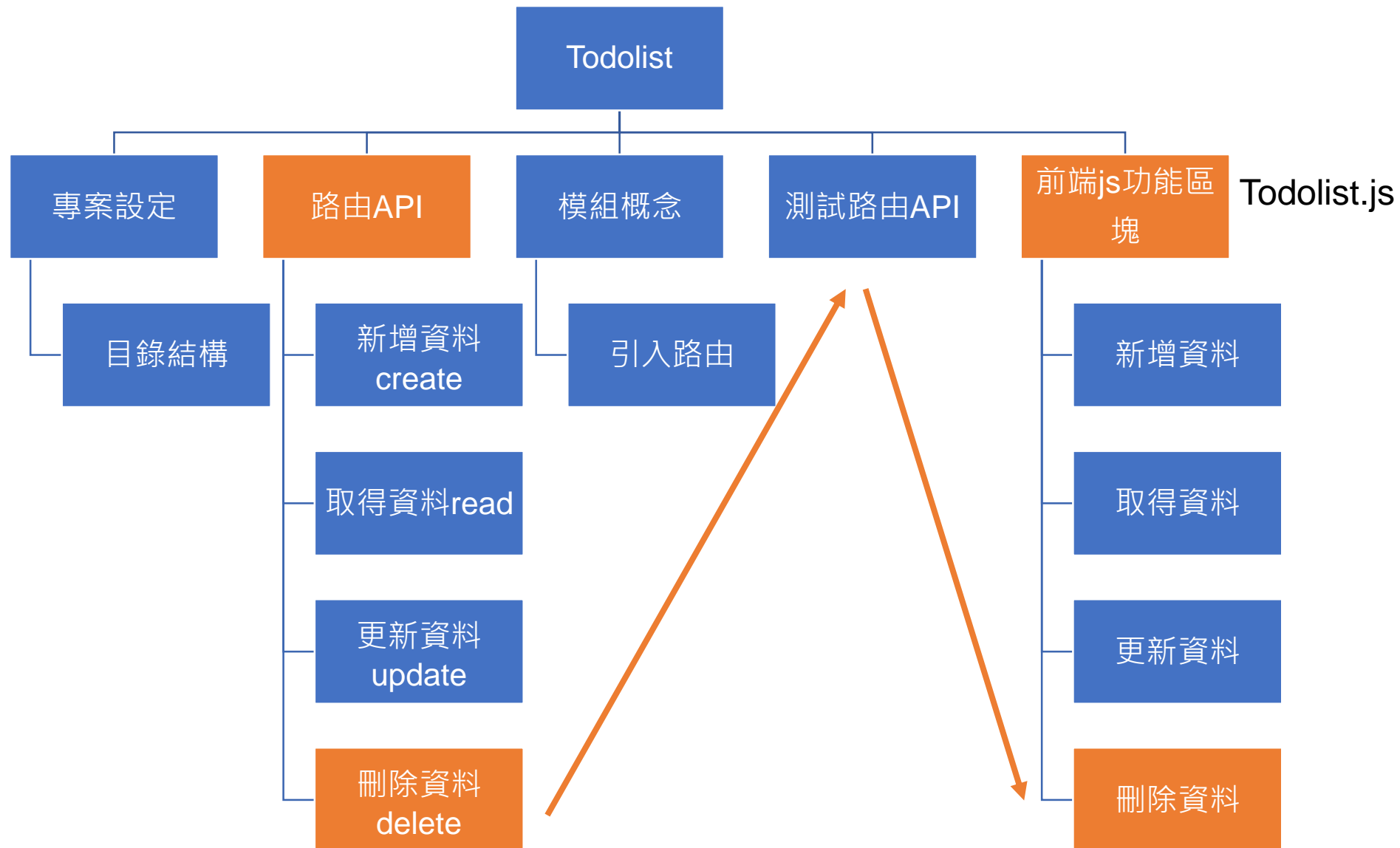
內容

輸入待辦事項

確認

<input type="checkbox"/>	標題	待辦事項	修改	更新	刪除
<input type="checkbox"/>	繳交報告	網頁期中企畫報告書_修改	更新		

```
{status: 0, msg: "success"}  
  msg: "success"  
  status: 0  
  ▶ proto : Object
```



刪除待辦事項 api.js

```
//刪除待辦事項
router.post('/removeList', function(req, res){
  var id = req.body.id;
  var index = allList.findIndex(element => element.id == id);
  allList.splice(index, 1);
  res.json({"status":0, "msg":"success"});
});
```

刪除待辦事項 todolist.js

```
//刪除待辦事項
function removeList(id){
    var API = "http://localhost:3000/api/removeList";
    var data = {"id":id};
    $.post(API, data, function(res){
        if(res.status == 0){
            $('#'+id).remove();
            alert("刪除成功!!!");
        }
    })
}
```

測試刪除是否正確

ToDoList

標題

內容

確認

<input type="checkbox"/>	標題	待辦事項	修改	更新	刪除
<input type="checkbox"/>	繳交報告1	網頁期中企畫報告書1	修改		刪除
<input type="checkbox"/>	繳交報告2	網頁期中企畫報告書2	修改		刪除
<input type="checkbox"/>	繳交報告3	網頁期中企畫報告書3	修改		刪除

localhost:3000 顯示

刪除成功!!!

確定

ToDoList

標題

內容

確認

<input type="checkbox"/>	標題	待辦事項	修改	更新	刪除
<input type="checkbox"/>	繳交報告1	網頁期中企畫報告書1	修改		刪除
<input type="checkbox"/>	繳交報告3	網頁期中企畫報告書3	修改		刪除

改變待辦事項外觀設計(刪除線) api.js

```
// 改變待辦事項外觀設計
router.post('/checkStatus', function(req, res){
  var id = req.body.id;
  var index = allList.findIndex(element => element.id == id);
  allList[index].status = allList[index].status? "false":"true";
  res.json({"status":0, "msg":"success"});
})
```


改變待辦事項外觀設計(刪除線) todolist.js

```
//改變待辦事項外觀設計
function checkStatus(id, checkStatus){
    var API = "http://localhost:3000/api/checkStatus";
    var data = {"id":id, "status":checkStatus.checked};
    $.post(API, data, function(res){
        if(res.status == 0){
            if(checkStatus.checked){
                $('#title'+id).addClass("textDelete");
                $('#msg'+id).addClass("textDelete");
                $('#btnEdit'+id).addClass("d-none");
            } else {
                $('#title'+id).removeClass("textDelete");
                $('#msg'+id).removeClass("textDelete");
                $('#btnEdit'+id).removeClass("d-none");
            }
        }
    });
}
```

測試外觀設計(刪除線)是否正確

TodoList

標題

標題

內容

輸入待辦事項

確認

<input type="checkbox"/>	標題	待辦事項	修改	更新	刪除
<input checked="" type="checkbox"/>	繳交報告1	網頁期中企畫報告書1			刪除
<input checked="" type="checkbox"/>	繳交報告2	網頁期中企畫報告書2			刪除
<input checked="" type="checkbox"/>	繳交報告3	網頁期中企畫報告書3			刪除

補充

```
/* GET home page. */
router.get('/', function (req, res) {
  res.sendFile(path.join(__dirname, '../public', 'todolist.html'));
  res.redirect('todolist.html')
});

module.exports = router;
```

- 訪問首頁時，**res.sendFile()**提供靜態**todolist.html**檔案，此方式雖可行，但是正式專案中盡量避免使用，**sendFile()**函式先讀取完檔案系統後，才能夠取得檔案資料，一個人讀取還好，若多個人則會有明顯的延遲，可以直接使用**express**中提供靜態檔案服務中介軟體，直接導向**public**中所需檔案即可。