**FriendsManagement Web API**

**Design** **Specification**

**1. Requirement**

**Use scirpts to design six web apis for a dummy friends manamgement system.**

**API requrement are as follows:**

|  |  |  |
| --- | --- | --- |
| **Web API** | **Input** | **Expected Results** |
| Make friends | Two emails with json:  {friends:['andy@example.com','john@example.com']} | 1.Store emails into db  2. return json if successful:  { "success": true} |
| Query friends via email | { email: 'andy@example.com'} | Get email lists from db, return json:  {"success":true,  "friends":['john@example.com'],  "count":1} |
| Query common friends between two emails | {friends:['andy@example.com','john@example.com']} | Get common emails from db, return json: {"success":true,"friends":['common@example.com'],"count" : 1} |
| Subscript to an email for updates | {"requestor":"lisa@example.com","target":[john@example.com](mailto:john@example.com)} | Store emails into db, return successful json: { "success": true} |
| Block an email not for updates | {  "requestor": "andy@example.com",  "target": "john@example.com"  } | Store emails into db, return successful json: { "success": true} |
| Get qualified emails for one email’s updates | {  "sender":"john@example.com",  "text": "Hello World! kate@example.com"  } | Get emails from db and attract the emails from the text.  { "success": true  "recipients":  [  "lisa@example.com",  "kate@example.com"  ]  } |

**2. Framework selection**

Though it’s a simple design of 6 web api, it still covers all the details of a database-oriented information management system.

For scripts on server, nodejs is popular and high performance with its natural asynchnization operation on IO. With express it can set up a web site very fast.

For database selection, to choose sql or non-sql is a big discussion topic. Basically from my experiences if logic is complex and lots of foreign relationship existing, just use Sql database. If no much foreign constrait, tables can be designed to objects, nosql database is very convenient, nosql db(mango) itself api provides lots function to modeling, json convetion. To this case I chose sqlite3 as a local file for easily querying.

Framework applied: Nodejs + express4 + sqlite3

IDE: Visual studio code

Test tool: CURL

(postman has issue to work with express, post with application/json content type always failed with the error “unexpected token”.)

**3. Design**

**3.1 API URL:**

Express starts with a default port: <http://localhost:3000>

/api indicates these urls are for web api.

/friends indicates the path are related to all operations of friends.

|  |  |
| --- | --- |
| API Name | URL |
| make friends | http://localhost:3000/api/friends/MakeFriends |
| Get friends | http://localhost:3000/api/friends/GetFriends |
| Get common friends | http://localhost:3000/api/friends/GetCommonFriends |
| Subscript to an email | http://localhost:3000/api/friends/SubUpdates |
| Block an email | http://localhost:3000/api/friends/BlockUpdates |
| Get all emails for one email updates | http://localhost:3000/api/friends/GetEmailsForUpdates |

**3.2 Database schema:**

One table is enough to process the requirements to make the system as simple as possible:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table columns: | ID (integer) | ReqEmail (text) | TargetEmail  (text) | Type (integer) |
| Description | Primary key, auto incresement | Action launch email | The target email | 1 - make friends  2 - subscription  3 – block |

\*Note: Type has three different values: 1, 2,3

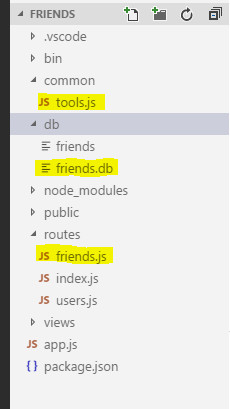
1 indicates that ReqEmail made friends with TargetEmail.

2 indicates that ReqEmail subscribed TargetEmail to receive its updates

3 indicates that ReqEmail blocked TargetEmail to hide its updates

**3.3 Project file structure**

Based on the files generated by the framework, we need to add one file for our URL mapping in Router folder. And to create a db folder for sqlite db file, and a tools.js included common functions for our api (module.exports). so the whole picture is as below:



If used nosql db, we may create a controller folder to container the model js file.

**4. Deployment**

Github address:

Steps to debug on your machine:

a. install nodejs on your machine from the address: <https://nodejs.org/en/download/>

b. download source code from github to your local folder, assuming it’s at c:\friends

c. after souce code downloads, run “cmd” and come to the path c:\friends

d. run “npm install”, the project configuration file package.json included all necessary components, npm will download all to install.

e. run npm start to launch web server. <http://localhost:3000/> can be visited for your to test.

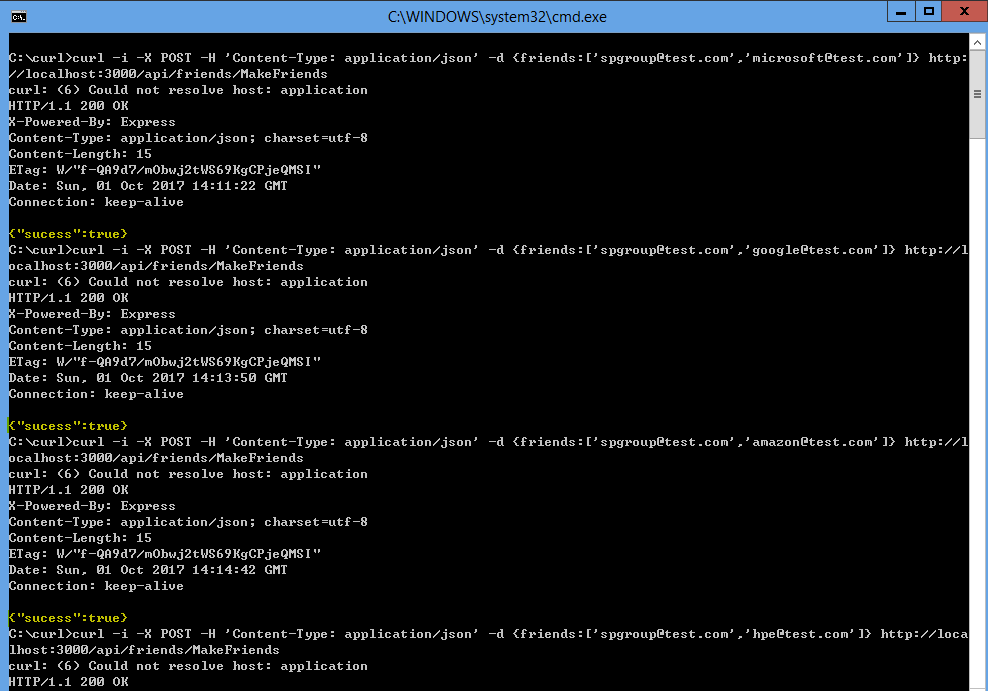
**5. Test**

**5.1 Dummy data design:**

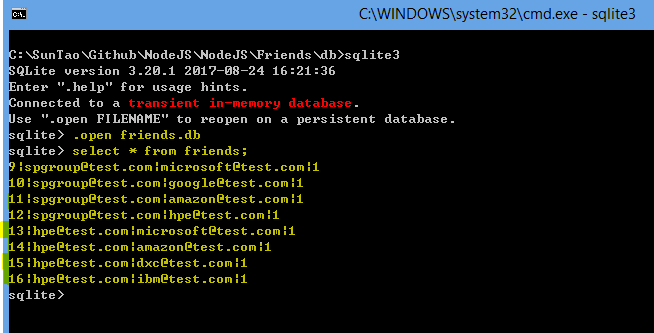
|  |  |
| --- | --- |
| **Friends:** | |
| spgroup@test.com | microsoft@test.com |
| google@test.com |
| amazon@test.com |
| hpe@test.com |
| hpe@test.com | microsoft@test.com |
| amazon@test.com |
| dxc@test.com |
| ibm@test.com |
| **Subscription** | |
| spgroup@test.com | ibm@test.com |
| **Block** | |
| spgroup@test.com | dxc@test.com |

**5.2 CURL call web api to generate datum**

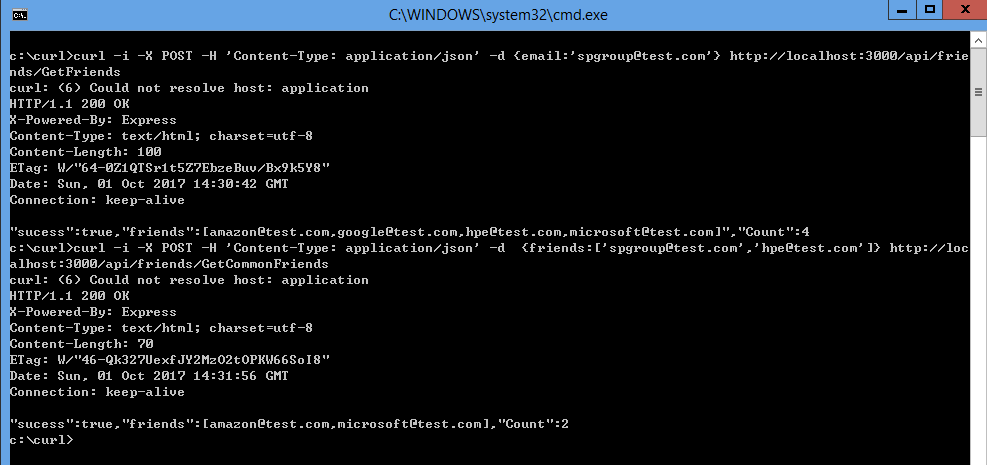
curl -i -X POST -H 'Content-Type: application/json' -d {friends:['spgroup@test.com','microsoft@test.com']} http://localhost:3000/api/friends/MakeFriends



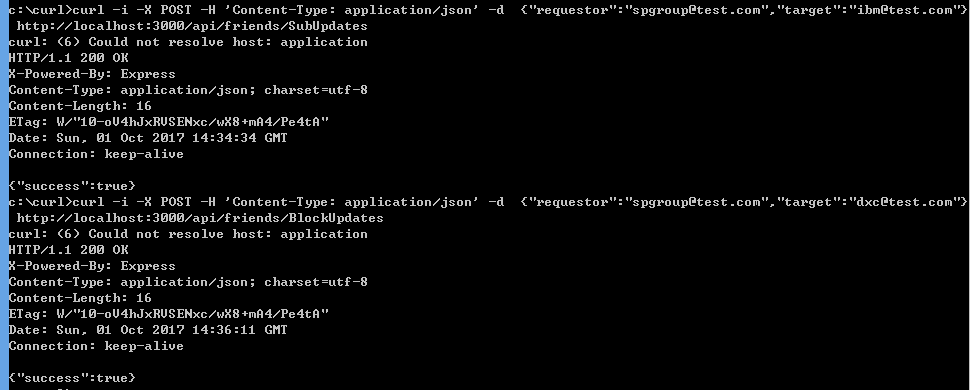
After friends information generated, we can check friends.db datum and verify the api getFriends and getCommonFriends right or not.



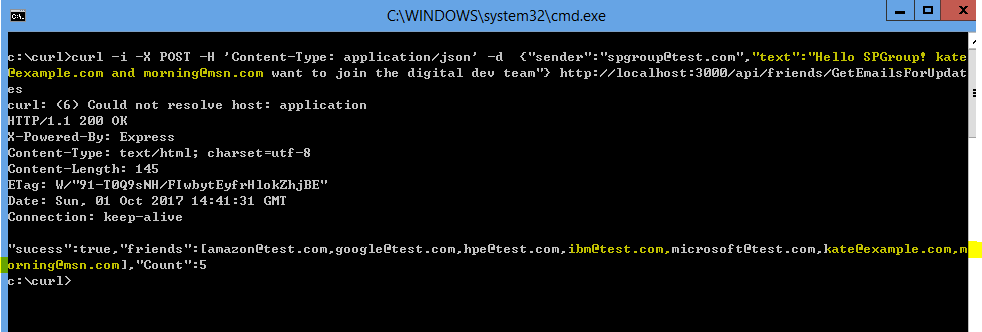
Call getFriends & getCommonFriends to see the results:



Cheers! Let’s continue to call API to add the subscription & blocks:



Now let’s verify the last API(get all emails for [spgroup@test.com](mailto:spgroup@test.com)), the logic is a little bit complex than others.



Totally right as expected, the emails extraction from text are right.

Thank you for your patient to look through the document and get here. The travel has showed our soluton works very well.

Singapore river to have a beer? Let’s go…

The attached text is curl command I used. You may copy it to run if you are interested.

curl -i -X POST -H 'Content-Type: application/json' -d {friends:['hpe@test.com','ibm@test.com']} http://localhost:3000/api/friends/MakeFriends

curl -i -X POST -H 'Content-Type: application/json' -d {email:'spgroup@test.com'} http://localhost:3000/api/friends/GetFriends

curl -i -X POST -H 'Content-Type: application/json' -d {friends:['spgroup@test.com','hpe@test.com']} http://localhost:3000/api/friends/GetCommonFriends

curl -i -X POST -H 'Content-Type: application/json' -d {"requestor":"spgroup@test.com","target":"ibm@test.com"} http://localhost:3000/api/friends/SubUpdates

curl -i -X POST -H 'Content-Type: application/json' -d {"requestor":"spgroup@test.com","target":"dxc@test.com"} http://localhost:3000/api/friends/BlockUpdates

curl -i -X POST -H 'Content-Type: application/json' -d {"sender":"spgroup@test.com","text":"Hello SPGroup! kate@example.com and morning@msn.com want to join the digital dev team"} http://localhost:3000/api/friends/GetEmailsForUpdates