MCC Project 2014

The application is available at:

```
130.233.42.182:8080
```

Run the server

Assuming that mongo, node and git are installed in the system:

```
# clone the repository
git clone https://github.com/nodo/mcc-project.git
cd mcc-project

# start mongodb
mkdir -p data/db
mongod --dbpath data/db/

# install dependencies
npm install

# start the server
PORT=8080 npm start
```

The application should be now accessible at "localhost:8080".

API Description

All the APIs returns 200 OK in case of success (if not specified otherwise) and 500 Internal Server Error with error message in case of error. If you try to look for a non-existant ID (such as in /contacts/:id, /update/:id, /delete/:id) the status code will be 404.

Contact

The main data type in the application is s **contact**. Since in the future there could be other entities we have decided to start each entry point with */contacts*. An example of contact is:

```
{
  ‘name’: ‘Andrea’,
  ‘surname: ‘Nodari’,
  ‘phone’: ‘123123’,
  ‘email’: ‘thisis@sparta.fi’
 }
```

Every attribute (except from ‘name’ and ‘surname’)can be a string or an array. This is done to allow contacts with multiple emails or phone numbers. For instance we could have:

```
{
  ‘name’: ‘Nicholas’,
  ‘surname: ‘Allio’,
  ‘phone’: [‘123123’, ‘789789’],
  ‘email’: [‘thisis@sparta.fi’, ‘nothisis@athens.fi’]
 }
```

GET /contacts

Returns an array containing the list of all contacts as an array of JSON.

```
curl -i localhost:8080/contacts
```

GET /contacts/:id

Returns the contact with the given ID.

```
curl -i localhost:8080/contacts/54410c317b6b78080040c385
```

GET /contacts/serach/:term

Returns an array containing all the contacts which name or surname matches the provided "search term". The search is case insensitive.

For instance the following request:

```
curl -i localhost:8080/contacts/search/john
```

Returns an array of contacts (in JSON format). All the returned contacts have ‘name’ or ‘surname’ containing the word ‘john’.

POST /contacts

Create a new contact using the data passed as JSON to the POST request. If successful returns the newly created contact with HTTP status 201 "Created".

```
curl -i -X POST -H 'Content-Type: application/json' -d '{"name": "John",
"surname": "Smith", "phone": "123123123"}' localhost:8080/contacts
```

GET /contacts/merge

Merge the contacts with same "name" and "surname" properties without side effects. If there are conflicts while merging they are handled replacing the conflicting property value with an array of all the possible distinct values. The API handles also the case of mergin two arrays.

```
curl -i localhost:8080/contacts/merge
```

POST /contacts/merge

Same as the previous one, but this API has side effects.

```
curl -i -X POST localhost:8080/contacts/merge
```

PUT /contacts/:id

Update the contacts given a certain ID with the data passed as JSON. If successful returns the newly updated contact with HTTP status 200 "OK".

```
curl -i -X PUT -H 'Content-Type: application/json' -d '{"name": "John",
"surname": "Smith", "phone": "999"}'
http://localhost:8080/contacts/54410c317b6b78080040c385
```

DELETE /contacts/:id

Delete the contact given a certain ID. If successful returns HTTP status 204 "No Content" without a body.

```
curl -i -X DELETE localhost:8080/contacts/54410c317b6b78080040c385
```

Web interface

The second round of the assignment is available at:

```
 http://lannister.ddns.net:8080
```

The web interface it's very simple.

1. You can add a new contact by filling the form on the top left

2. The table in the middle of the page show the list of the contacts
3. You can show the details or delete a contact pressing the links on the side
4. The button "merge contacts" merge the contacts with the same name and surname
5. Before syncronize *from* and *to* Google contacts it is necessary to press the button "Get Token".
6. Afterwards it's possible to pull the contacts or push the contacts pressing the respective buttons.

Other details

The third-party source of the contacts is Google Contacts, the node package used to authenticate is "googleapis". Unfortunately it does not support Google Contacts, therefore plain https requests have been done.

The client is implemented with jQuery. The design of the system is such the client and the APIs are very loosely coupled.

Ejs is used for templating.

Zurb Foundation is used as CSS framework

Mobile application

The application is designed to be simple to use respecting the iOS guidelines. The first interaction is about asking the permission of access the local Address Book.

Once the user has accepted, he/she can visualize the contacts coming from the web application. He/She can easily perform add/remove/merge/show_details operation. The user can seamlessly utilize the address book, importing and exporting contact in the web application.

The installation requires XCode since we are still in a prototype phase. In order to use the mobile application on a physical device a iOS developer licence is needed.