|  |
| --- |
|  |

Assignment 7 – RESTful API

By

Tech Squad Group

Web UI Design CMPE 280, Tue & Th 10:30am – 11:45am

Professor Ron Mak

November 10th, 2018

**Instructions**

1. Install node from <https://nodejs.org/en/> and mongo database from <https://docs.mongodb.com/master/administration/install-community/>
2. Unzip the zip file we uploaded on Canvas to any folder
3. Navigate to Server folder
4. Run “npm install” to install all needed npm modules
5. Run “npm start” command to start node server
6. You can see Node Backend Application on <http://localhost:5000> URL.
7. For the purpose of this Assignment #7, we need to only run the backend Node.js Backend using the Express.js framework
8. Start mongo database with the mongod command
9. Add records to the database by pasting below url into browser

<http://localhost:5000/addrecords>

**Introduction**

We used Node.js backend, Express.js framework, and MongoDB for this assignment. We used Monk as client connector to the database.

We can regard our application as a service (SAAS) to display talent, add talent, etc. We are exposing APIs for these functions.

An API is different from a user interface because an API is a programmatic API. Our service uses an industry standard programmatic interface so that other applications can interact with our application

We want to make sure that we follow industry standard for these APIs. That standard is called REST. REST stands for Representational State Transfer.

We want to design routing so that every route will contain all the information necessary to identify the talent that the user wants to work with.

We are going to expose our restful APIs through Mongo database.

A database uses CRUD operations: Create, Read, Update, and Delete

We can map these to HTTP verbs as follows

* Create -> POST
* Read -> GET
* Update ->PUT
* Delete ->DELETE

Express fakes PUT and DELETE since a web browser can only issue GET and POST.

The below table shows our 7 APIs and how they compare to REST standards.

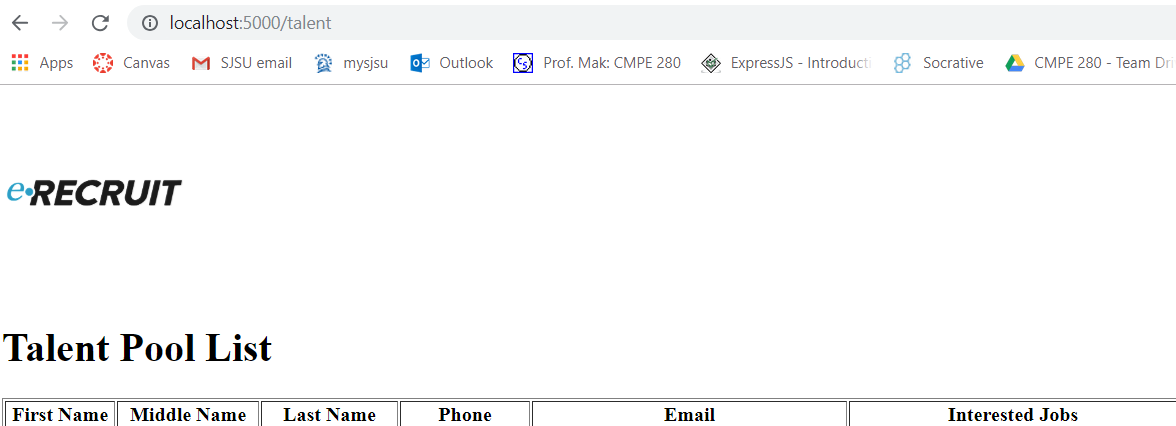
|  |  |  |  |
| --- | --- | --- | --- |
| **RESTful action** | **Http** | **Method and URI** | **Operation on resource** |
| index | GET | GET /talent | List all talent |
| show | GET | GET /talent/:lname | Show one talent |
| new | GET | GET /addTalent | Show a form to create a new talent |
| edit | GET | GET /updateTalent | Show a form to edit an existing talent |
| create | POST | POST /addTalent | Create a new talent |
| update | PUT | PUT /updateTalent | Update an existing talent |
| destroy | DELETE | DELETE /deleteTalent | Delete an existing talent |

**Screenshots of actions**

The following section displays a screenshot of all actions as they occur on the screen.

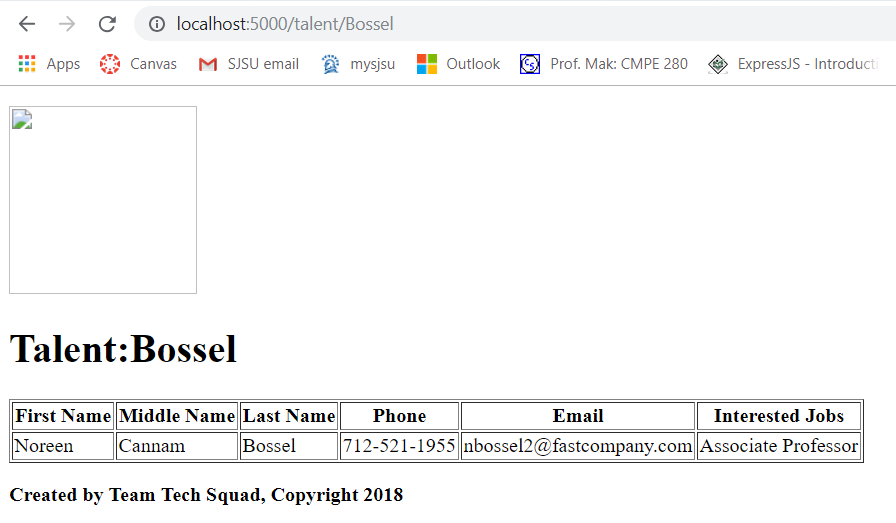
In the implementation section **we also display screenshot of all routes and controller action code** to explain the implementation.

List all talent (1)

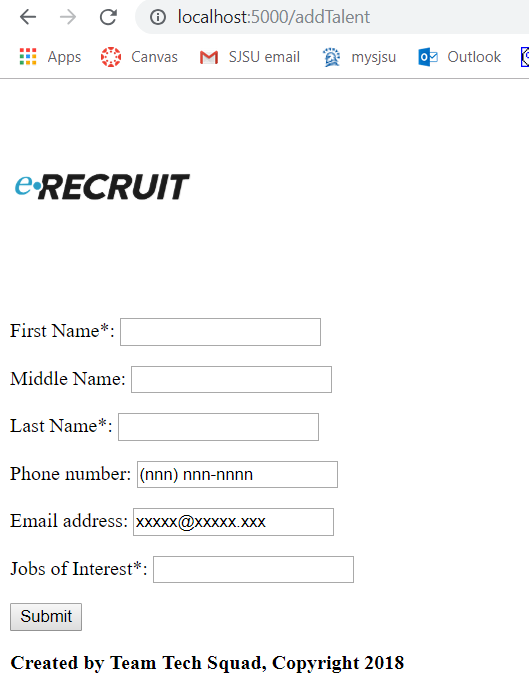




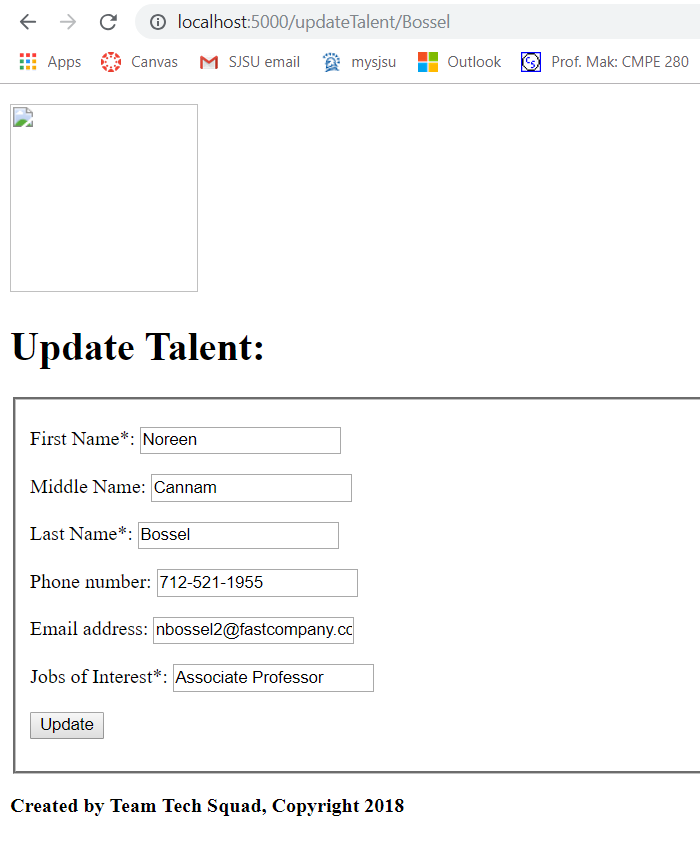
Show one talent (2)



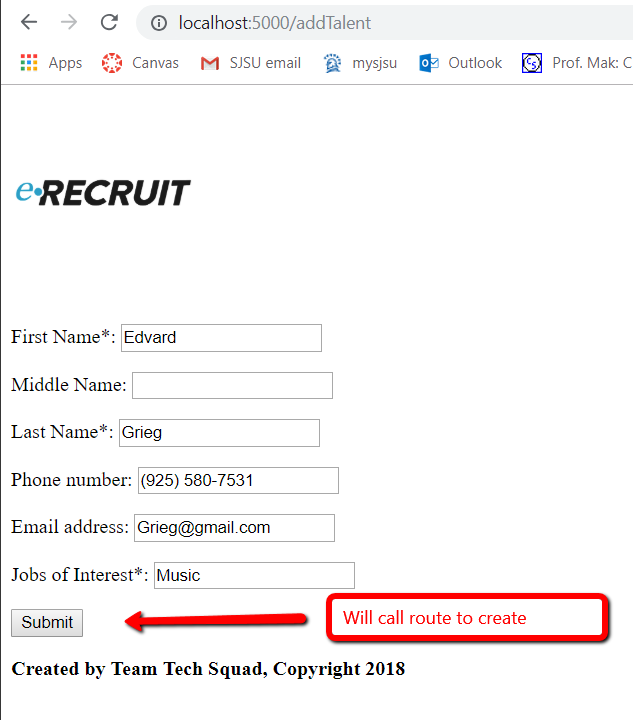
Show a form to create a new talent (3)

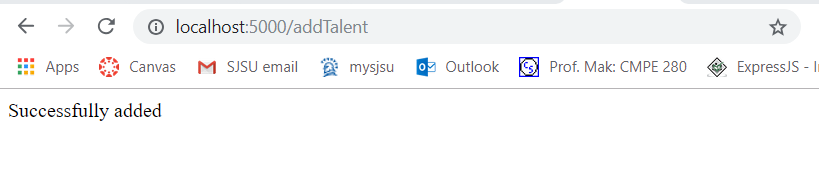


Show a form to edit an existing talent (4)

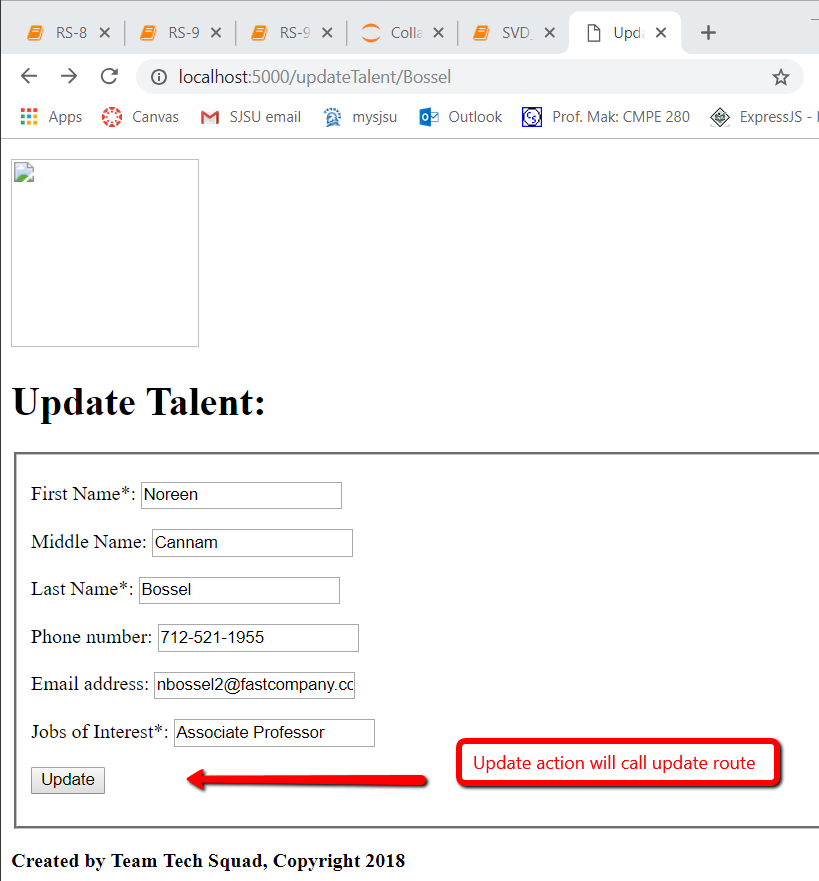
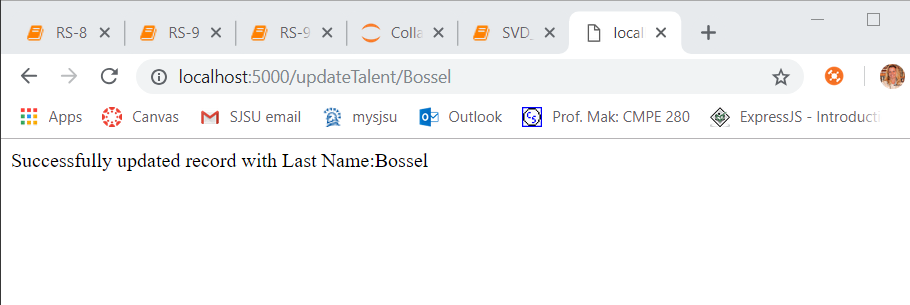


Create a new talent from form data (5)

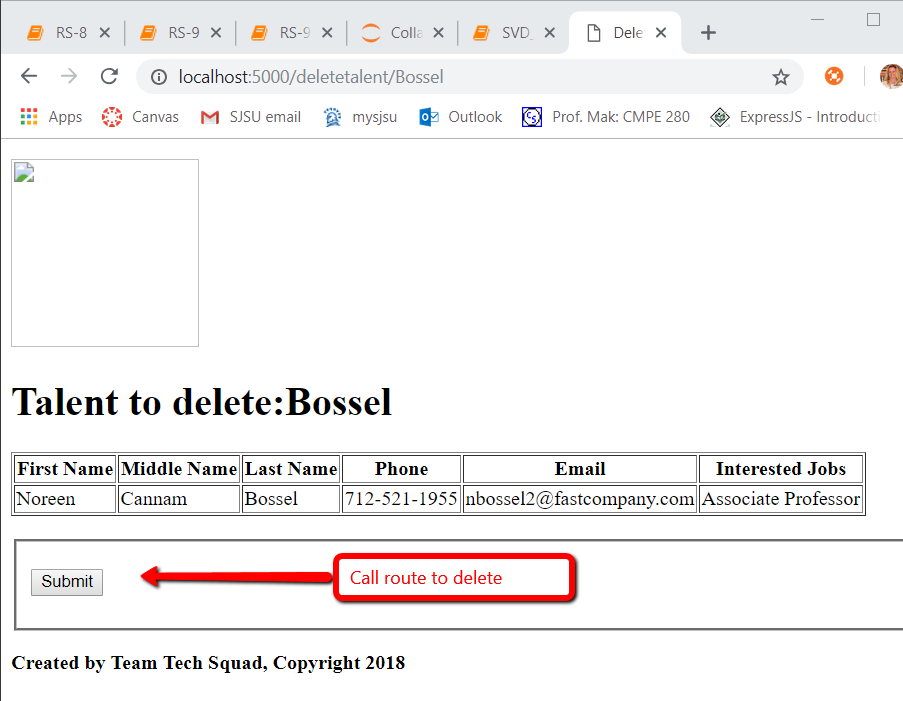


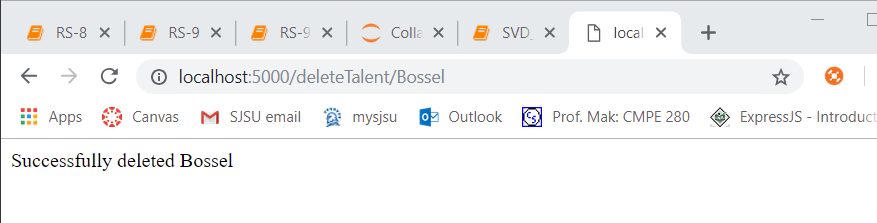


Update a talent from form data (6)

Delete talent (7)



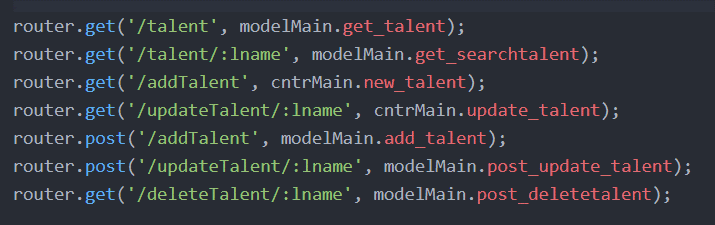


**Explanation of implementation**

Below table shows the relationship between the apis and the routes we created

|  |  |  |  |
| --- | --- | --- | --- |
| **RESTful action** | **Http** | **Description** | **Route implementation** |
| index | GET | List all talent | router.get('/talent', modelMain.get\_talent); |
| show | GET | Show one talent | router.get('/talent/:lname', modelMain.get\_searchtalent) |
| new | GET | Show a form to create a new talent | router.get('/addTalent', cntrMain.new\_talent) |
| edit | GET | Show a form to edit an existing talent | router.get('/updateTalent/:lname', cntrMain.update\_talent); |
| create | POST | Create a new talent | router.post('/addTalent', modelMain.add\_talent); |
| update | PUT | Update an existing talent | router.post('/updateTalent/:lname', modelMain.post\_update\_talent); |
| destroy | DELETE | Delete an existent talent | router.get('/deleteTalent/:lname', modelMain.post\_deletetalent); |

Below is a screenshot of our routes in index.js



We have faked “put” http verb for updating talent by using post.

We have also faked “delete” http verb for deleting talent by using get.

We have to do this since the browser only understands get and post. Express lets us do this “faking”

Below is a screenshot of each of the associated controller code where we will explain the implementation of each

1. Display talent.

This code displays all the talent in our database. It renders template displayTalent.hbs. We are using handlebar as our views template



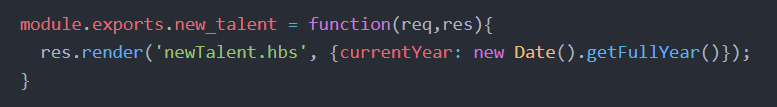
2. Show talent.

This code renders template displayTalent.hbs and selects one talent to display based on last name (lname).



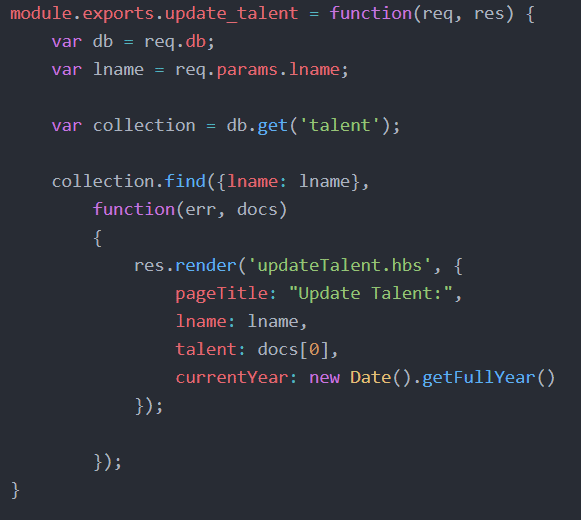
3. Display form to create a new talent

This code renders template newTalent.hbs so that users can enter new talent.



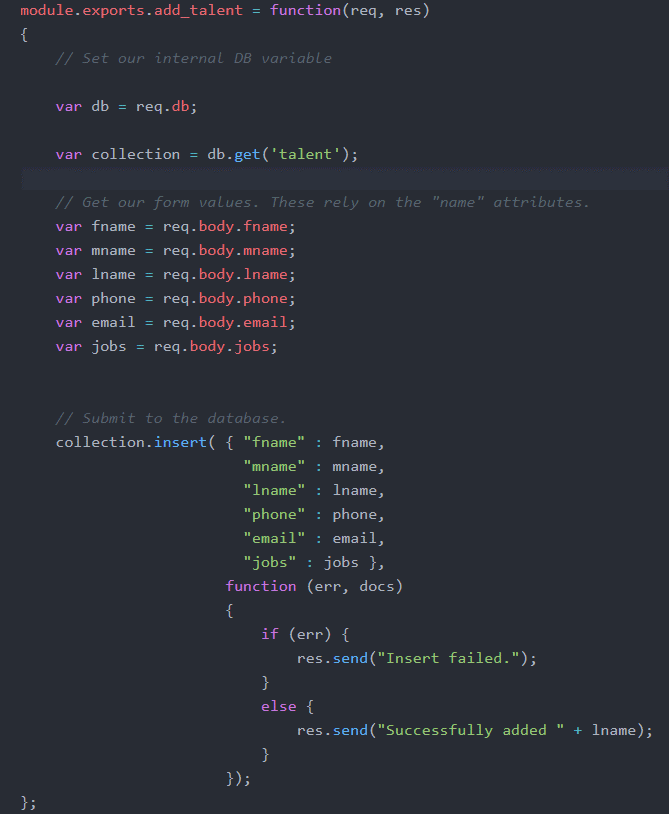
4. Display form to edit a talent

This code renders template updateTalent.hbs to display form to edit a talent. It edits a particular talent based on last name (lname).



5. Create new talent from form data

Below code adds new talent to the database when user his the submit button



6. Update a user from form data

Below code updates talent from form data



7. Delete a talent

Below code deletes a talent from the database

