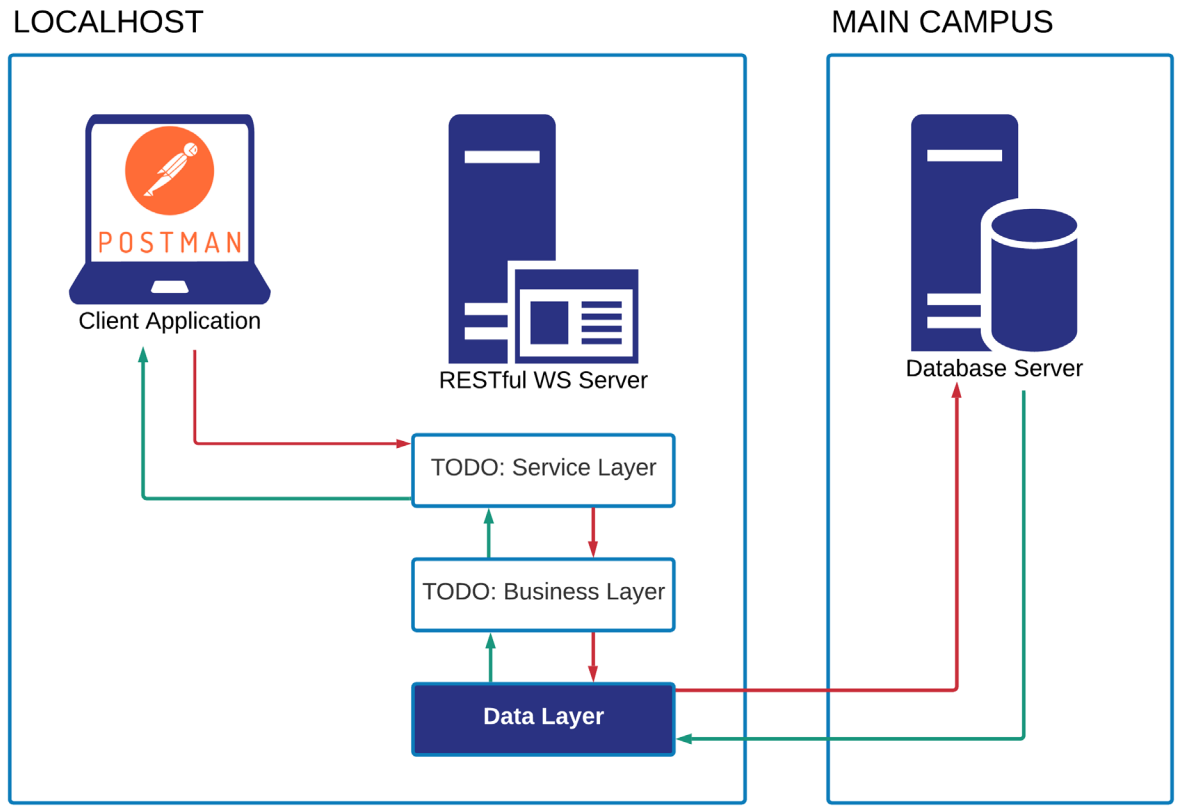
Project 3: RESTful WS in Node.js – Tracking Timecards for Employees

Your job is to write a RESTful service in Node.js for a company to allow them to track timecards for employees. You will have to build your web service (WS) following the 3-layered architecture (service layer, business layer, data layer):



Recall the workflow:

1. Postman is used to send a request
2. The request is being processed by the Service layer of your Web Service. The Service layer is implemented as a Façade to work with a more complex Business layer.
3. The Business layer is using the Data layer to get the data requested by Postman.
4. Once the data is returned from the Data layer, the Business layer formats and prepares a data structure to be returned to the Service layer.
5. Based on that data structure, the Service layer prepares and send the HTTP Response to Postman.

The data for your web service is stored in a DB, named Company, that is hosted on a main campus server. The data layer is already implemented and made available to you as the companydata module that you need to install in the node_modules folder. See the Data Layer documentation for more details.

The missing parts that you need to implement are:

- Service Layer
- Business Layer
- Postmen collection of different requests to test all the web service's API.

You will run and test your solution (web service with the Postman requests) on your computer – LOCALHOST. There is no main campus server to upload your solution to. If you are off campus, you will need to establish a secure internet connection via the VPN client first to connect to the DB.

Regarding the **Service Layer**, you need to create the service as per below, including any validation mentioned which **should be in** your **Business Layer**. You can put other things in your Business Layer if you wish. **You need to use your RIT user name for the company name whenever it is asked for**. For error output, return an appropriate informational error message (**not** the String "An appropriate error message.")

Service Layer

All methods must return a JSON String which don't have to be **formatted** as in the samples below (in other words, with no carriage returns/line feeds/tabs) but must contain the same information. Some methods take JSON as input, others take Query Parameters or Form Parameters.

All method signatures must match the ones listed. You may have to use multiple Data Layer methods to accomplish each Service Layer method.

General input validation: Refer to the EER Diagram for the database for datatypes and sizes. Any additional validation/business rules will be listed below in the appropriate method.

The first part of the request URI should be in the following format:
http://localhost:8080/YourLastNameYourFirstInitialP2 /CompanyServices

Context Path**: YourLastNameYourFirstInitialP2**
Root path for Service Layer Path**: CompanyServices**

The rest of the path including the resource method should be implemented as follows:

1) **Path:** /company
   Verb: DELETE
   Produces: application/json
   a. Deletes all Department, Employee and Timecard records in the database for the given company. You will need to pay attention to the Foreign Key Constraints.
   b. Input is your RIT user ID as a String passed as **QueryParam**
      i. company=company+name

where “company+name” is your RIT user ID

c. Output:
    i. Success:
        1. A JSON String:

```
{"success":"companyName's information deleted."}
```

        2. A JSON String:

```
{"error":"An appropriate error message."}
```

2) Path: /department
Verb: GET
Produces: application/json

d. Returns the requested Department as a JSON String.
e. Input as **QueryParams**:

company=company+name&dept_id= id

where “company+name” is your RIT user ID and
“dept_id” is the record id of the department to retrieve.

f. Output:
    i. Success:
        1. A JSON String:

```
{
        "dept_id":1,
        "company":"rituserid",
        "dept_name":"accounting",
        "dept_no":"d10",
        "location":"new york"
}
```

        2. A JSON String:

```
{"error":"An appropriate error message."}
```

3) Path: /departments
Verb: GET
Produces: application/json

g. Returns the requested list of Departments.
h. Input is your RIT user ID as a String in a **QueryParam**.
    i. company=company+name

where “company+name” is your RIT user ID

i. Output:

i. Success:

1. A JSON String:


```
[
  {
      "dept_id":1,
      "company":"rituserid",
      "dept_name":"accounting",
      "dept_no":"d10",
      "location":"new York”
  },
  {
      "dept_id":2,
      "company":"rituserid",
      "dept_name":"research",
      "dept_no":"d20",
      "location":"dallas"
  },
  {
      "dept_id":3,
      "company":"rituserid",
      "dept_name":"sales",
      "dept_no":"d30",
      "location":"chicago"   },
  {
      "dept_id":4,
      "company":"rituserid",
      "dept_name":"operations",
      "dept_no":"d40",
      "location":"boston"   }
]
```


2. A JSON String:


```
{"error":"An appropriate error message."}
```


4) Path: /department

Verb: PUT

Consumes: application/json

Produces: application/json

j. Additional Validation:

i. dept_no must be unique within your company, Suggestion: include company name as part of id.
ii. dept_id must be an existing record number for a department

k. Returns the updated Department as a JSON String.
l. Input: **JSON String**

```
{
  "company":"rituserid",
  "dept_id":5,
  "dept_name":"IT",
  "dept_no":"d11",
  "location":"rochester"
}
```

where “company” is your RIT user ID.

m. Output:
    i. Success:
        1. A JSON String:

```
{
  "success":{
      "dept_id":5,
      "company":"rituserid",
      "dept_name":"IT",
      "dept_no":"d11",
      "location":"rochester"   }
}
```

        2. A JSON String:

```
{"error":"An appropriate error message."}
```

5) Path: /department
Verb: POST
Produces: application/json

n. Additional Validation:
    i. dept_no must be unique within your company, Suggestion: include company name as part of id.
o. Returns the new Department as a JSON String.
c. Input(any values you want to change plus the record id for the Department) as FormParam:

```
"company" ="rituserid"
```

"dept_name" = "mystery"
"dept_no" = "d10"
"location" = "buffalo"

where “company” is your RIT user ID.

p. Output:
    i. Success:
        1. A JSON String:

```
{
  "success":{
      "dept_id":1,
      "company":"rituserid",
      "dept_name":"mystery",
      "dept_no":"d10",
      "location":"buffalo"   }
}
```

        2. A JSON String:

```
{"error":"An appropriate error message."}
```

6) Path: /department
Verb: DELETE
Produces: application/json

q. Returns the number of rows deleted.
r. Input as **QueryParam**:

"company" = "company name"
"dept_id" = id

where “company name” is your RIT user ID and
“id” is the record id of the department to delete.

s. Output:
    i. Success:
        1. A JSON String:

```
{
  "success":"Department 5 from rituserid deleted."
}
```

        2. A JSON String:

```
{"error":"An appropriate error message."}
```

7) Path: /employee
Verb: GET
Produces: application/json

t. Returns the requested Employee as a JSON String.
u. Input: the record id of the desired Employee as a **QueryParam**
    i. emp_id=#
v. Output:
    i. Success:
        1. A JSON String:

```
{
    "emp_id":2,
    "emp_name":"jones",
    "emp_no":"e2",
    "hire_date":"1981-04-01",
    "job":"manager",
    "salary":2975.0,
    "dept_id":2,
    "mng_id":1}
```

        2. A JSON String:

```
{"error":"An appropriate error message."}
```

8) Path: /employees
Verb: GET
Produces: application/json

w. Returns the requested list of Employees.
x. Input is your RIT user ID as a String as a **QueryParam**.
    i. company=company+name
        where “company+name” is your RIT user ID
y. Output:
    i. Success:
        1. A JSON String:

```
[
  {
    "emp_id":1,
    "emp_name":"king",
    "emp_no":"e1",
    "hire_date":"1981-11-16",
```

```
    "job":"president",
    "salary":5000.0,
    "dept_id":1,
    "mng_id":0
},
{
    "emp_id":2,
    "emp_name":"jones",
    "emp_no":"e2",
    "hire_date":"1981-04-01",
    "job":"manager",
    "salary":2975.0,
    "dept_id":2,
    "mng_id":1
},
{
    "emp_id":3,
    "emp_name":"ford",
    "emp_no":"e3",
    "hire_date":"1981-12-02",
    "job":"analyst",
    "salary":3000.0,
    "dept_id":2,
    "mng_id":2
},
{
    "emp_id":4,
    "emp_name":"smith",
    "emp_no":"e4",
    "hire_date":"1980-12-16",
    "job":"clerk",
    "salary":800.0,
    "dept_id":2,
    "mng_id":2
},
{
    "emp_id":5,
    "emp_name":"blake",
    "emp_no":"e5",
    "hire_date":"1981-04-30",
    "job":"manager",
    "salary":2850.0,
    "dept_id":3,
    "mng_id":1   },
{
    "emp_id":6,
    "emp_name":"allen",
```

```
        "emp_no":"e6",
        "hire_date":"1981-02-19",
        "job":"salesman",
        "salary":1600.0,
        "dept_id":3,
        "mng_id":5   },
    {
        "emp_id":7,
        "emp_name":"ward",
        "emp_no":"e7",
        "hire_date":"1981-02-21",
        "job":"salesman",
        "salary":1250.0,
        "dept_id":3,
        "mng_id":5
    },
    {
        "emp_id":8,
        "emp_name":"martin",
        "emp_no":"e8",
        "hire_date":"1981-09-27",
        "job":"salesman",
        "salary":1250.0,
        "dept_id":3,
        "mng_id":5
    },
    {
        "emp_id":9,
        "emp_name":"clark",
        "emp_no":"e9",
        "hire_date":"1981-06-08",
        "job":"manager",
        "salary":2450.0,
        "dept_id":3,
        "mng_id":1   }
]
```


2. A JSON String:

   {"error":"An appropriate error message."}

9) Path: /employee
Verb: POST
Consumes: Form Parameters
Produces: application/json

z. Additional validations:
   i. dept_id must exist as a Department in your company
   ii. mng_id must be the record id of an existing Employee in your company. Use 0 if the first employee or any other employee that doesn't have a manager.
   iii. hire_date must be a valid date equal to the current date or earlier (e.g. current date or in the past)
   iv. hire_date must be a Monday, Tuesday, Wednesday, Thursday or a Friday. It **cannot** be Saturday or Sunday.
   v. emp_id must be unique amongst all employees in the database. You may wish to include your RIT user ID in the employee number somehow.

aa. Returns the new Employee as a JSON String.

bb. Input as FormParam:

```
"emp_name"="french",
"emp_no"="rituserid-e1b",
"hire_date"="2018-06-16",
"job"="programmer",
"salary"=5000.0,
"dept_id"=1,
"mng_id"=2
```

cc. Output:
   i. Success:
      1. A JSON String:

```
{
  "success":{
      "emp_id":15,
      "emp_name":"french",
      "emp_no":"rituserid-e1b",
      "hire_date":"2018-06-16",
      "job":"programmer",
      "salary":5000.0,
      "dept_id":1,
      "mng_id":2   }
}
```

      2. A JSON String:

```
{"error":"An appropriate error message."}
```

10) Path: /employee
Verb: PUT
Produces: application/json

dd. Additional validations same as inserting an Employee plus emp_id must be a valid record id in the database.
ee. Returns the updated Employee as a JSON String.
ff. Input(any values you want to change plus the record id for the Employee) as **JSON string**:

```
{
  "emp_id":15,
  "emp_name":"french",
  "emp_no":"rituserid-e1b",
  "hire_date":"2018-06-16",
  "job":"programmer",
  "salary":6000.0,
  "dept_id":1,
  "mng_id":2
}
```

gg. Output:
  i. Success:
    1. A JSON String:

```
{
  "success":{
      "emp_id":15,
      "emp_name":"french",
      "emp_no":"rituserid-e1b",
      "hire_date":"2018-06-16",
      "job":"programmer",
      "salary":6000.0,
      "dept_id":1,
      "mng_id":2   }
}
```

    2. A JSON String:

```
{"error":"An appropriate error message."}
```

11) Path: /employee
Verb: DELETE
Produces: application/json

hh. Returns the that the employee deleted.
ii. Input: the record id of the Employee to delete as a **QueryParam**.

i. emp_id=#

jj. Output:
i. Success:
1. A JSON String:

```
{
  "success":"Employee 15 deleted."
}
```

2. A JSON String:

```
{"error":"An appropriate error message."}
```

12) Path: /timecard
Verb: GET
Produces: application/json

kk. Returns the requested Timecard as a JSON String.
ll. Input: the record id of the desired Timecard as a **QueryParam**
i. timecard_id=#

mm. Output:
i. Success:
1. A JSON String:

```
{
  "timecard":{
    "timecard_id":1,
    "start_time":"2018-06-14 11:30:00",
    "end_time":"2018-06-14 15:30:00",
    "emp_id":2
  }
}
```

2. A JSON String:

```
{"error":"An appropriate error message."}
```

13) Path: /timecards
Verb: GET
Produces: application/json

nn. Returns the requested list of Timecards.

oo. Input is the record id of the employee you want to see the Timecards for as a **QueryParam**.
    i. emp_id=#
pp. Output:
    i. Success:
        1. A JSON String:


```
[
  {
      "timecard_id":3,
      "start_time":"2018-06-14 11:30:00",
      "end_time":"2018-06-14 15:30:00",
      "emp_id":4
  },
  {
      "timecard_id":4,
      "start_time":"2018-06-13 11:30:00",
      "end_time":"2018-06-13 15:30:00",
      "emp_id":4
  },
  {
      "timecard_id":6,
      "start_time":"2018-06-12 11:30:00",
      "end_time":"2018-06-12 15:30:00",
      "emp_id":4   }
]
```


        2. A JSON String:


```
{"error":"An appropriate error message."}
```


14) Path: /timecard
Verb: POST
Consumes: application/json
Produces: application/json

qq. Additional validations:
    i. emp_id must exist as the record id of an Employee in your company.
    ii. start_time must be a valid date and time equal to the current date or up to 1 week ago from the current date.
    iii. end_time must be a valid date and time at least 1 hour greater than the start_time and be on the same day as the start_time.
    iv. start_time and end_time must be a Monday, Tuesday, Wednesday, Thursday or a Friday. They **cannot** be Saturday or Sunday.

v. start_time and end_time must be between the hours (in 24 hour format) of 06:00:00 and 18:00:00 inclusive.
vi. start_time must not be on the same day as any other start_time for that employee.

rr. Returns the new Timecard as a JSON String.
ss. Input(any values you want to change plus the record id for the Timecard) as **FormParam**s:

```
"emp_id"=1,
"start_time"="2018-06-15 12:30:00",
"end_time"="2018-06-15 15:30:00"
```

tt. Output:
    i. Success:
        1. A JSON String:

```
{
  "success":{
      "timecard_id":1,
      "start_time":"2018-06-14 11:30:00",
      "end_time":"2018-06-14 15:30:00",
      "emp_id":2
  }
}
```

        2. A JSON String:

```
{"error":"An appropriate error message."}
```

15) Path: /timecard
Verb: PUT
Produces: application/json

uu. Additional validations same as inserting a Timecard plus timecard_id must be a valid record id in the database.
vv. Returns the updated Timecard as a JSON String.
ww. Input:

```
{
  "timecard_id":2,
  "start_time":"2018-06-14 11:30:00",
  "end_time":"2018-06-14 15:30:00"
}
```

xx. Output:
    i. Success:
        1. A JSON String:

```
{
  "success":{
      "timecard_id":0,
      "start_time":"2018-06-15 12:30:00",
      "end_time":"2018-06-15 15:30:00",
      "emp_id":2

  }
}
```

        2. A JSON String:

```
{"error":"An appropriate error message."}
```

16) Path: /timecard
Verb: DELETE
Produces: application/json

yy. Returns the number of rows deleted.
zz. Input: the record id of the Timecard to delete as a **QueryParam**.
    i. timecard_id=#

aaa. Output:
    i. Success:
        1. A JSON String:

```
{
  "success":"Timecard 1 deleted."
}
```

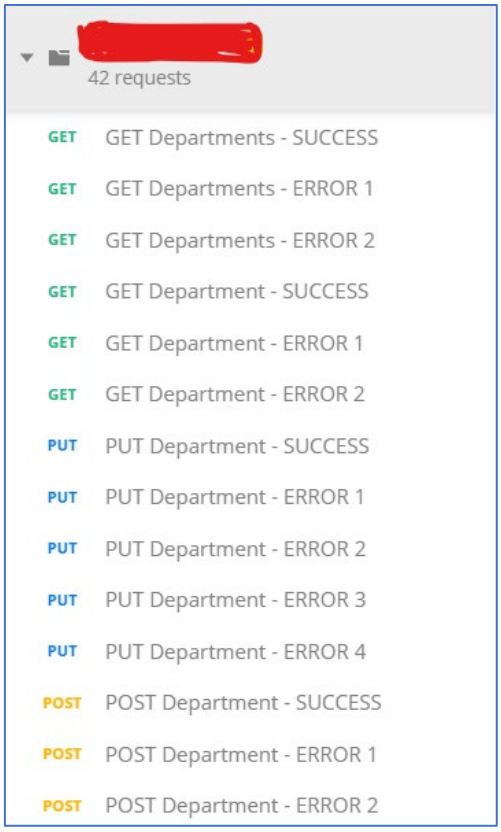
        2. A JSON String:

```
{"error":"An appropriate error message."}
```

Deliverables

Submit your zip, **YourLastNameYourFirstInitialP3.zip** , to the MyCourses Assignments folder for Project3 by the due date specified on the Assignments folder (end of week 13). The zip should contain:

1) Your project with package.json file that lists all the modules you used, so when I perform $ npm install in the command line, all the needed modules are installed including the company module. Test this for yourself before uploading.
2) A postman collection in the json format named **YourLastNameYourFirstInitialP3.postman_collection.json.**



The collection should have all the requests prepared, including the ones that will return error messages. In Postman, make sure you save all your requests before exporting those as a collection. In case you do not save prior exporting, you will end up with a collection of empty requests which are useless for our project testing.

In addition, make sure that you enough records in the database to test your app with Postman requests.

Hints

1) In addition to the Data Layer module, install any other required module using $ npm install <module name>

2) To convert from Timestamp to String (for putting in JSON String), take a look at the date-fns module (format method) or the moment.js module.

3) To test using Postman:
   a. For DELETE method, make sure any text under raw/body is deleted, all form fields are unchecked and uncheck any header fields and the correct id is set as a parameter.

b. For POST method, select x-www-form-urlencoded with fields for each item you want to pass in.
c. For PUT, make sure Content-Type header = application/json

Grading Rubrics

	Possible Points	Actual Points
All required methods with correct inputs and outputs:	40	
All validations implemented and found in Business Layer:	25	
Appropriate error messages:	5	
Correct Node.js structure and it runs:	15	
Good programming design and coding principles (DRY, etc.):	15	
Total:	100	