ADVANCED WEB APPLICATION DEVELOPMENT
WEEK 10 ASSIGNMENT

1.Explain the REST API- POST,PUT ,Updating and Deleting methods to subdocument of MongoDB.

1. REST API

- An API (Application Programming Interface) is a set of rules by which two programs can communicate with each
- other. An API can be used to return data, files, and additional information.
- In simple terms, it's an abstraction so other people can utilize the functionality of that application without knowing
  business logic.
- Let's learn it by an example: suppose you are in a restaurant where the waiter is an API, and the kitchen is the
- server. You make an order to that waiter, and then that waiter goes to the kitchen and places your order to the chef,
- and then the chef completes that order, and the waiter serves you with that order.
- So API works exactly like this, they provide an interface between two applications. You request the endpoint, and that endpoint serves you with that request. But in this whole process, your programming is communicating with another programming, and all the codes are hidden from you.
- REST stands for REpresentational State Transfer. It's an architectural style for designing an API, and it was first
  presented by Roy Fielding in 2000.
- In REST, each URL is called a request, and data that you get back is called the response. It's a designing guide, not
  a tool or any software.Before creating API, make sure the node is installed on your machine.

1. Creating An REST API with Express and Node.js

- First, create a folder where you want to store your files and folders. Let's name test-api, you can name it whatever you want.
- Now go to that directory and run npm init -y in the command line. It will initialize an empty directory to use for node.js and also create a file called package.json.
- package.json file contains much information about your project, like the name of the app, description about your
- project, author name, dependencies, and devDependecies. You can change it whenever you want.
- Above in scripts, we create a command run and we are going to use it with npm.
- When you run npm start then it will be executed as node server.js, where server.js is our main file for the app, you can name it whatever you want.
- Let's create a server.js file. But first, we have to install express, for that run given command: npm install express –save
- It will install express and save it to package.json. One thing you have noticed that one another folder is created
- named node_modules. In the node_modules folder, all the libraries and packages will be saved.
- Now let's create the server.js file and type given code.
- We have imported express and then initialized an app by calling express function, and then using the listen to the method, we have created a server on port 5000.

- When you run npm run command, it will run the server on port 5000. When you open the https://localhost:5000,
- then it will show you nothing, because we didn’t create any route yet. By route, I mean a path like https://github.com/rajeshberwal
- Before creating any route, first, understand some HTTP methods that we use in web development.
- GET: get request is used to retrieve the data from the web, e.g., when you opened our article, you make a get request.

- POST: post request is used to send data to the server, e.g., user information like passwords, usernames, or any personal information.
- PUT: used to update the data on the server
- DELETE: used to delete a piece of information from server
- So, let’s create a get request to the homepage.
- Now, run npm run command again and open http://localhost:5000 in your browser. Then you will see Hello world! on the homepage.
- In the above code / indicates our main homepage route, and in a method, we have to pass two things first, one is the route, and the other one is a middleware function.
- A middleware function is used to create functionality for that route. Now let’s create a user route.
- Whenever someone calls our user route, they will get the user’s information. We are going to store the rest of the routes in the routes folder. You can save them wherever you want. But storing paths in a routes folder is a standard way.
- To create folder routes in the current directory and create another file called users.js. You can name it whatever you
- want. In the user’s files, we are going to create some dummy users.
- And now, import this route to our main server.js file.
- Here in users.js file, we are importing router from express. And then, we created two routes, the first one is /users, and the second one is /users/{user_id}.
- When someone calls users route, then they will get all the user’s data, and when they specify the user by using id user/1 they will get data related to that user.
- And then we have exported that router. And we have imported that route in our main server.js file.
- In the server.js file, we have used it using the use method.
- It accepts two arguments, the first one is a route on that we want to show data, and the second one is a module.
- When we run the command npm start then we can see all the user's information on http://localhost:5000/users route,
- and when we specify an id http://localhost:5000/users/1, it will return data only for that user, and if the id is not available, then it will show “User not found.”
- On http://localhost:5000/users
- If id available then: http://localhost:5000/users/0 If id is not possible:
- You have created your first API using Node.js and express. You can use this knowledge to develop your
- applications. You can use other HTTP methods to increase its functionality.

2. Discuss Rest API methods for update and Delete sub documents from database with real time example.

Consider we have a JSON based database of users having the following users in a file users.json:

```
{
"user1" : {
```

```
"name" : "mahesh",
"password" : "password1",
"profession" : "teacher",
"id": 1
},
"user2" : {
"name" : "suresh",
"password" : "password2",
"profession" : "librarian",
"id": 2
},
"user3" : {
"name" : "ramesh",
"password" : "password3",
"profession" : "clerk",
"id": 3
}
}
```

Based on this information we are going to provide following RESTful APIs.

S. N.	URI	HTTP Method	POST body	Result
1	listUsers	GET	empty	Show list of all the users.
2	addUser	POST	JSON String	Add details of new user.
3	deleteUser	DELETE	JSON String	Delete an existing user.
4	:id	GET	empty	Show details of a user.

I'm keeping most of the part of all the examples in the form of hard coding assuming you already know how to pass values from front end using Ajax or simple form data and how to process them using express Request object.

List Users

Let's implement our first RESTful API listUsers using the following code in a server.js file:

```
var express = require('express');
var app = express();
var fs = require("fs");
app.get('/listUsers', function (req, res) {
fs.readFile(__dirname + "/" + "users.json", 'utf8', function (err, data) {
console.log( data );
res.end( data );
});
})
var server = app.listen(8081, function () {
var host = server.address().address
var port = server.address().port
console.log("Example app listening at http://%s:%s", host, port)
})
```

Now try to access the defined API using http://127.0.0.1:8081/listUsers on local machine. This should produce following result:

You can change given IP address when you will put the solution in production environment.

```
{
"user1" : {
"name" : "mahesh",
"password" : "password1",
"profession" : "teacher",
"id": 1
},
"user2" : {
"name" : "suresh",
"password" : "password2",
"profession" : "librarian",
"id": 2
},
"user3" : {
"name" : "ramesh",
"password" : "password3",
"profession" : "clerk",
"id": 3
}
}
```

Add User

Following API will show you how to add new user in the list. Following is the detail of the new user:

```
user = {
"user4" : {
"name" : "mohit",
"password" : "password4",
"profession" : "teacher",
"id": 4
}
}
```

You can accept the same input in the form of JSON using Ajax call but for teaching point of view, we are making it hard coded here. Following is the addUser API to a new user in the database:

```
var express = require('express');
var app = express();
var fs = require("fs");
var user = {
"user4" : {
"name" : "mohit",
"password" : "password4",
"profession" : "teacher",
"id": 4
}
}
app.get('/addUser', function (req, res) {
// First read existing users.
fs.readFile(__dirname + "/" + "users.json", 'utf8', function (err, data) {
data = JSON.parse( data );
data["user4"] = user["user4"];
console.log( data );
res.end( JSON.stringify(data));
});
})
var server = app.listen(8081, function () {
var host = server.address().address
var port = server.address().port
console.log("Example app listening at http://%s:%s", host, port)
})
```

Now try to access defined API using http://127.0.0.1:8081/addUsers on local machine. This should produce following result:

```
{ user1:
{ name: 'mahesh',
password: 'password1',
profession:  'teacher',
id: 1 },
user2:
{ name: 'suresh',
password: 'password2',
profession: 'librarian',
id: 2 },
user3:
{ name: 'ramesh',
password: 'password3',
profession: 'clerk',
id: 3 },
user4:
{ name: 'mohit',
password: 'password4',
profession:  'teacher',
id: 4 }
}
```

Show Detail

Now we will implement an API which will be called using user ID and it will display the detail of the corresponding user.

```
var express = require('express');
var app = express();
var fs = require("fs");
app.get('/:id', function (req, res) {
// First read existing users.
fs.readFile(___dirname + "/" + "users.json", 'utf8', function (err, data) {
data = JSON.parse( data );
var user = users["user" + req.params.id]
console.log( user );
res.end( JSON.stringify(user));
});
})
var server = app.listen(8081, function () {
var host = server.address().address
var port = server.address().port
console.log("Example app listening at http://%s:%s", host, port)
})
```

Now let's call above service using http://127.0.0.1:8081/2 on local machine. This should produce following result:

```
{
"name":"suresh",
"password":"password2",
"profession":"librarian",
"id":2
}
```

Delete User

This API is very similar to addUser API where we receive input data through req.body and then based on user ID we delete that user from the database. To keep our program simple we assume we are going to delete user with ID 2.

```
var express = require('express');
var app = express();
var fs = require("fs");
var id = 2;
app.get('/deleteUser', function (req, res) {
// First read existing users.
fs.readFile(__dirname + "/" + "users.json", 'utf8', function (err, data) {
data = JSON.parse( data );
delete data["user" + 2];
console.log( data );
res.end( JSON.stringify(data));
});
})
var server = app.listen(8081, function () {
var host = server.address().address
var port = server.address().port
console.log("Example app listening at http://%s:%s", host, port)
})
```

Now let's call above service using http://127.0.0.1:8081/deleteUser on local machine. This should produce following result:

```
{ user1:
{ name: 'mahesh',
password: 'password1',
profession:  'teacher',
id: 1 },
user3:
{ name: 'ramesh',
password: 'password3',
profession: 'clerk',
id: 3 }
}
```

Loading [MathJax]/jax/output/HTML-CSS/jax.j