**Experiment No. 5**

**Title: Create meteor app to accept data, store and retrieve it from MongoDB.**

**Batch:B1 RollNo.:1514033 ExperimentNo.:5**

### Aim: Create meteor app to accept data through form from user, insert it in Mongodb database and display all values of collection in the form.

### --------------------------------------------------------------------------------------------

**Resources needed:**Meteor

### Theory:

**Template**

Within Meteor, templates are used to create a connection between the project’s interface and the project’s JavaScript code.

To create a template, add the following code to the very bottom of the HTML file, beneath the closing body tag:

<templatename="leaderboard">

Hello World

</template>

Here, we’re using this template tag to define a template, and the nameattribute to distinguish between the templates we create. To include the “leaderboard” template inside the interface and have it appear inside the browser, place the following tag between the body tags inside the HTML file:

**{{>**leaderboard**}}**

Obviously, this isn’t HTML. Instead, the use of double-curly braces means this is the Spacebars syntax, and Spacebars is the syntax we use in our HTML when we want something dynamic to occur. It’s the syntax that bridges the gap between the interface and the application logic.

Based on these changes, the HTML file should now resemble:

<head>

<title>Leaderboard</title>

</head>

<body>

<h1>Leaderboard</h1>

{{>leaderboard}}

</body>

<templatename="leaderboard">

Hello World

</template

To store the input from user we need collection which is then added to mongodb, the following line of cod create collection “PlayerList”:

PlayersList**=new**Mongo.Collection('players');

## Create an Event

To create an event, write the following code in javascript file:

Template.leaderboard.events({

*// events go here*

});

Here, there’s a few things going on:

First, the Template keyword searches through all of the templates in this project.

Second, the leaderboard keyword is a reference to the name of the template that we’re about to attach some events to.

Third, the events keyword is used to specify that, within the coming block of code, we want to specify one or more events. This structure is similar to how we previously defined helper functions.

Between the curly braces of the events block, create an event in the JSON format:

Template.leaderboard.events({

'click':**function**(){

*// code goes here*

}

});

### Results: (Screen shots of application development steps, program code and web browser displaying the specified message.)

**Main.html :**

<head>

<title> DF exp 5 1514033</title>

</head>

<body>

<center>

<h1>NATIONAL CORPORATION</h1>

{{> add\_item}}

{{>add}}

{{>update}}

{{>delete}}

<br><br><br>

<table border="10">

<tr><th>Employee ID</th><th>Name</th><th>Salary</th></tr>

{{#each users}}

{{>info}}

{{/each}}

</table>

</center>

</body>

<template name="add\_item">

<label>Employee ID:</label><input type="text" class="ID"><br><br>

&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;

<label>Name:</label><input type="text" class="NAME"><br><br>

&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;

<label>Salary:</label><input type="text" class="SALARY"><br><br>

</template>

<template name="info">

<tr> <td>{{employeeid}}</td><td>{{employeename}}</td><td>{{employeesalary}}</td></tr>

</template>

<template name="add">

<button class="add\_user"> Add employee </button>

</template>

<template name="update">

<button class="update\_user">Update employee</button>

</template>

<template name="delete">

<button class="delete\_user">Delete employee</button>

</template>

**Main.js:**

import {UserObj} from '../dbcode.js';

if(Meteor.isClient)

{

Template.body.helpers({

users:function(){

return UserObj.find();

}

});

Template.add.events({

'click.add\_user'(event)

{

var id =$('.ID').val();

var name =$('.NAME').val();

var salary=$('.SALARY').val();

console.log(id);

console.log(name);

console.log(salary);

UserObj.insert({employeeid:id,employeename:name,employeesalary:salary});

console.log('inserted');

}

});

Template.update.events({

'click.update\_user'(event){

var id =$('.ID').val();

var name =$('.NAME').val();

var salary=$('.SALARY').val();

console.log(id);

console.log(name);

console.log(salary);

var doc=UserObj.findOne({employeeid:id});

console.log(doc.\_id);

UserObj.update({\_id:doc.\_id},{$set:{employeesalary:salary}})

console.log('updated');

}

});

Template.delete.events({

'click.delete\_user'(event){

var id =$('.ID').val();

var name =$('.NAME').val();

var salary=$('.SALARY').val();

console.log(id);

console.log(name);

console.log(salary);

var doc=UserObj.findOne({employeeid:id});

console.log(doc.\_id);

UserObj.remove({\_id:doc.\_id});

console.log('deleted');

}

});

}

**Dbcode.js**

export const UserObj = new Mongo.Collection('userdata');

**server main.js**

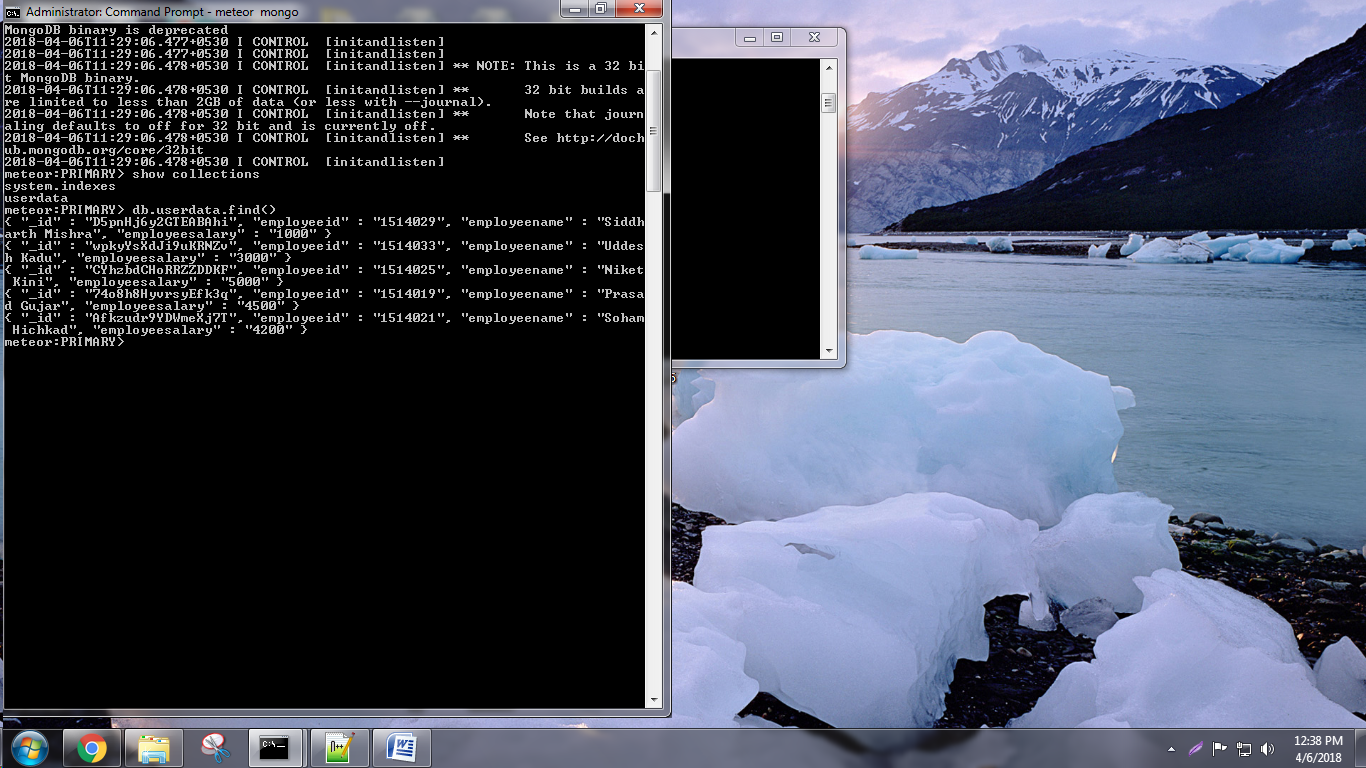
import { Meteor } from 'meteor/meteor';

import {UserObj} from '../dbcode.js';

Meteor.startup(() => {

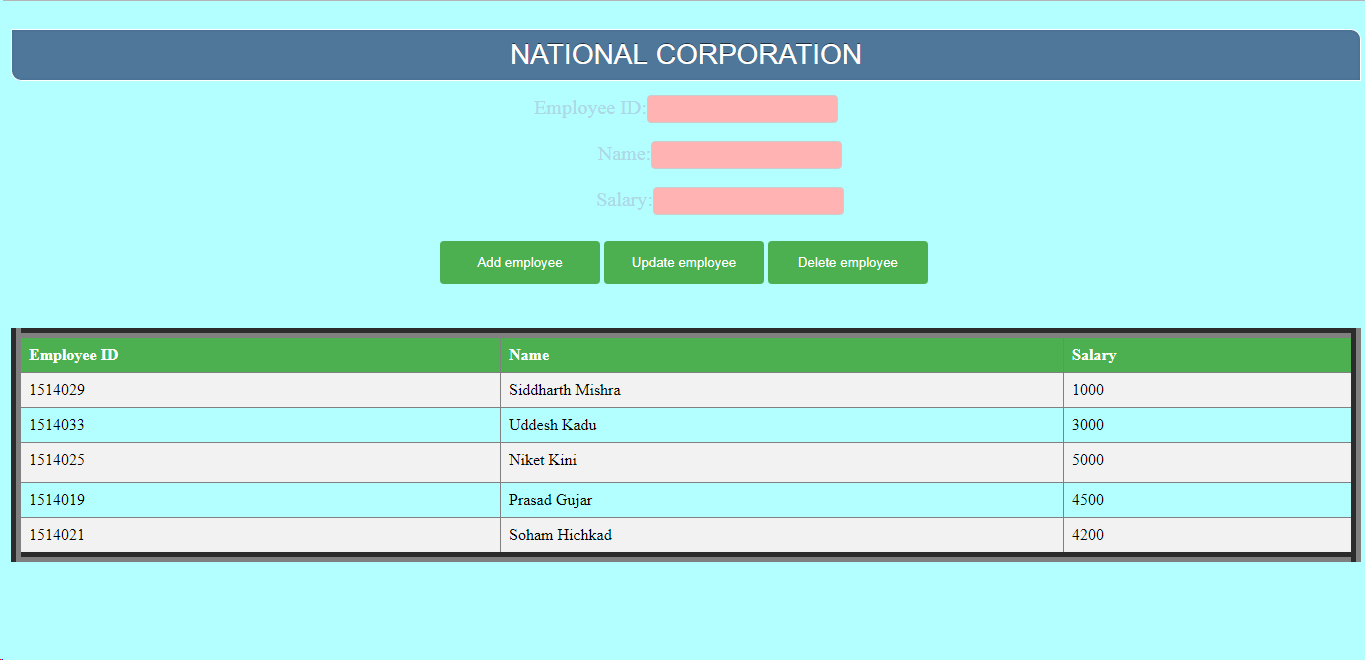
// code to run on server at startup

});

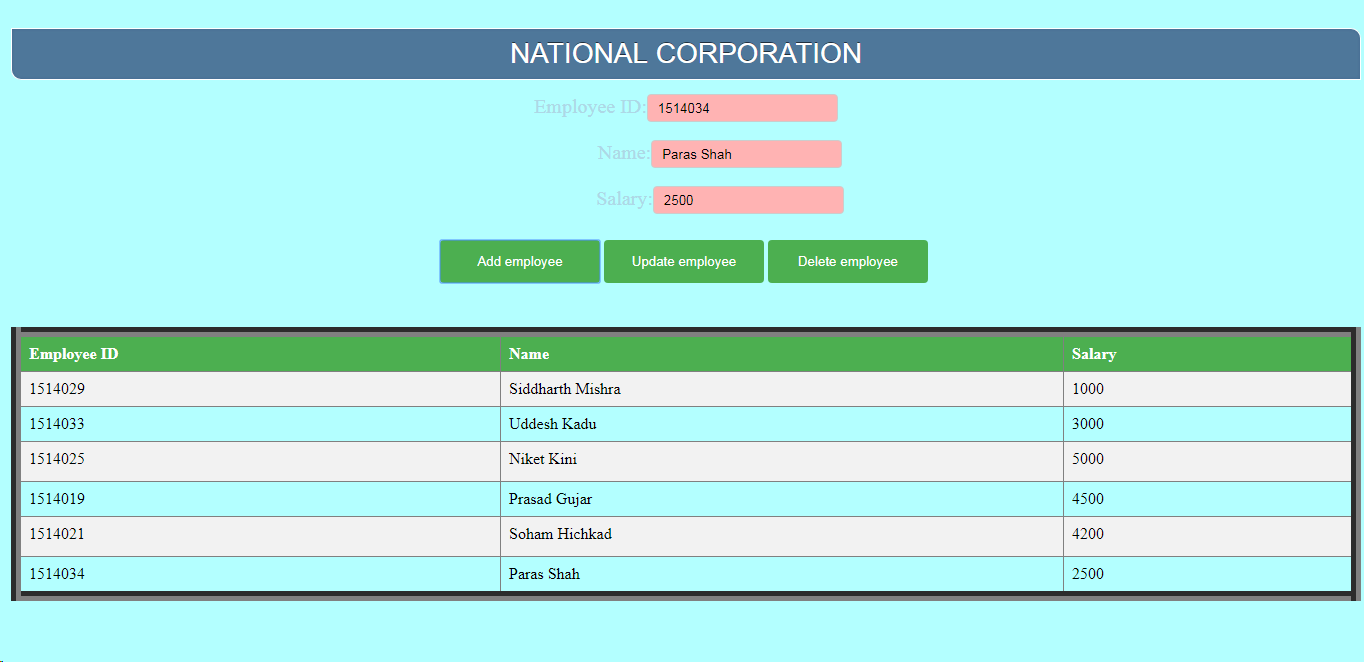


**OUTPUT :**

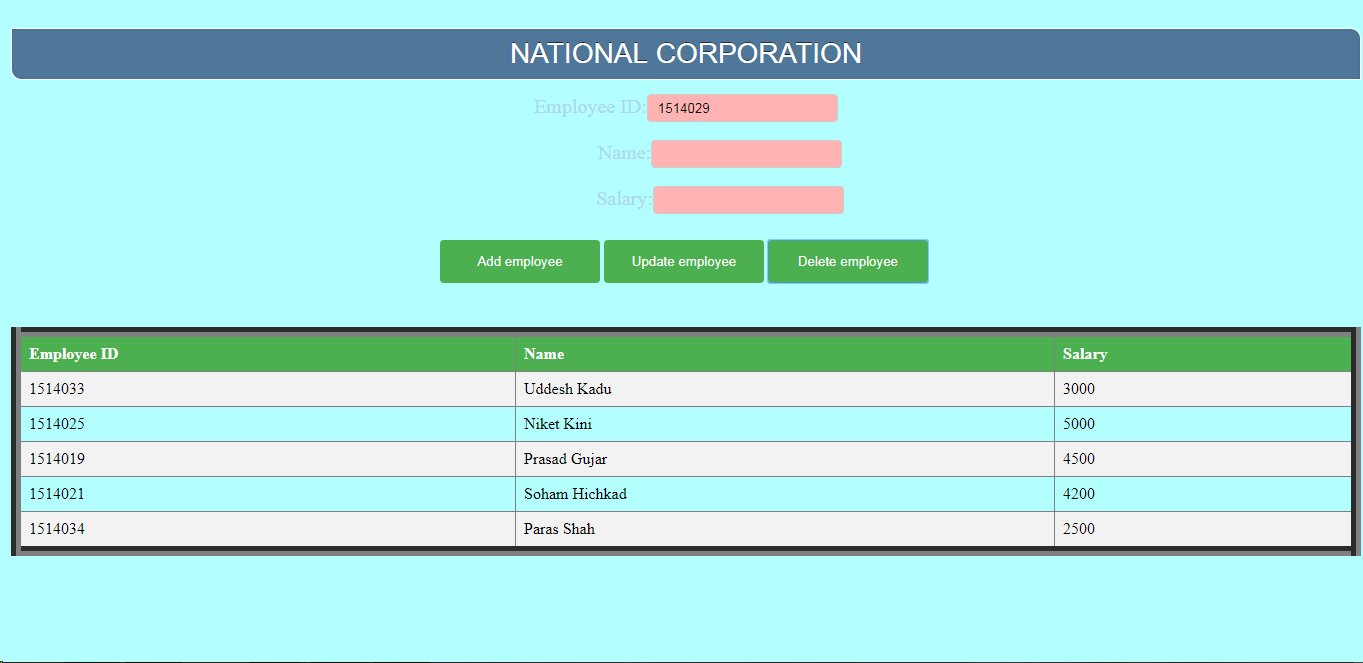
At start



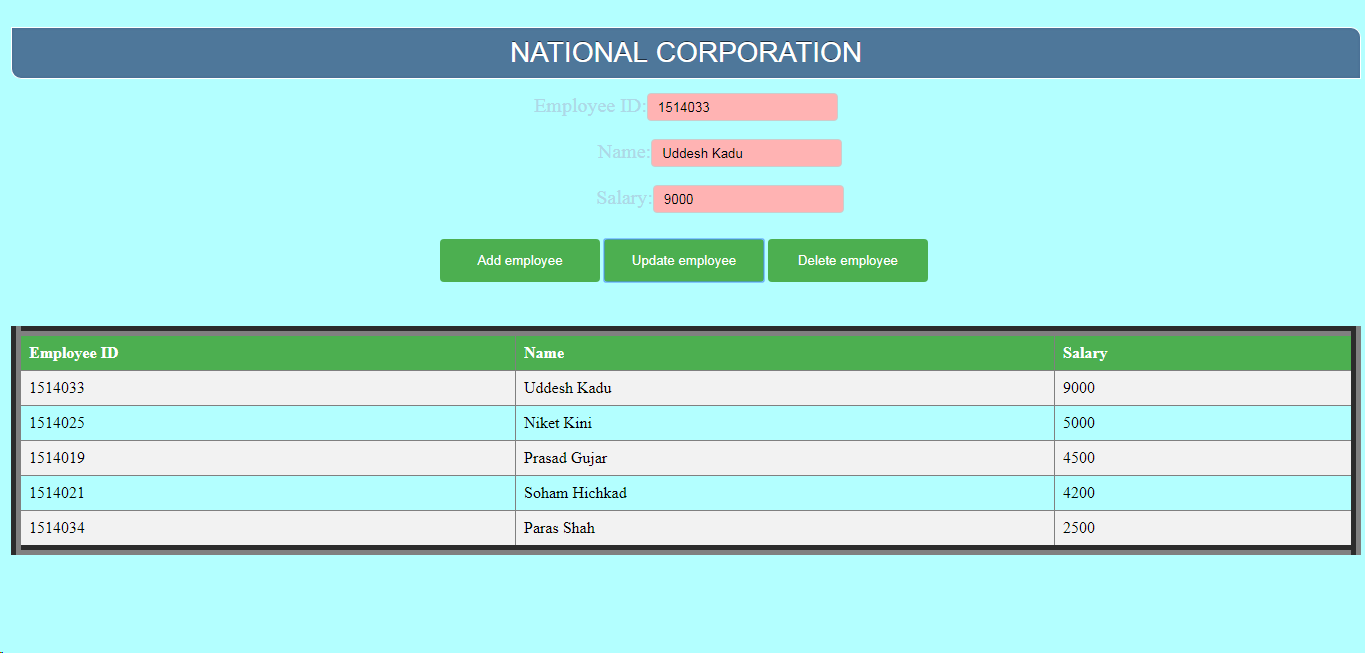
Add Employee



Delete Employee



Update Employee



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Questions:**

1. **Explain different Isopacks in meteor.**

**Ans.**

Meteor uses different types of packages. Meteor core has meteor packages. These packages are included into every app. They are known as isopacks – the isomorphic packages that work both on client and server. There are first-party packages like accounts-ui which are maintained by core team of Meteor, and they come bundled with Meteor. There are third-party isopacks developed by other users which they upload to Meteor’s package server. They are available on Atmosphere with the Meteor search command. You can develop custom packages and keep them in packages directory. And then there are NPM (Node.js Packaged Modules) –these can be used with other packages.

Meteor’s isobuild brings it all together. It combines right bits of code for each target, using right build pipeline which involves transpiling, minifying, determining load order, resolving imports and exports and more. Meteor has built-in packages to handle pre-processing and custom user authentication for Google, Facebook, Twitter, Email etc.

### Outcomes:

Develop Applications using Frameworks

**Conclusion: (Conclusion to be based on the objectives and outcomesachieved)**

Thus the Meteor application was successfully able to add update and delete value from the Mongo DB.

**Grade: AA / AB / BB / BC / CC / CD/DD**

**Signature of faculty in-charge withdate**

**References:**

1. By Isaac Strack; “Getting Started with Meteor.js JavaScript Framework”, 2nd Edition

;Packt Publishing, June 2015