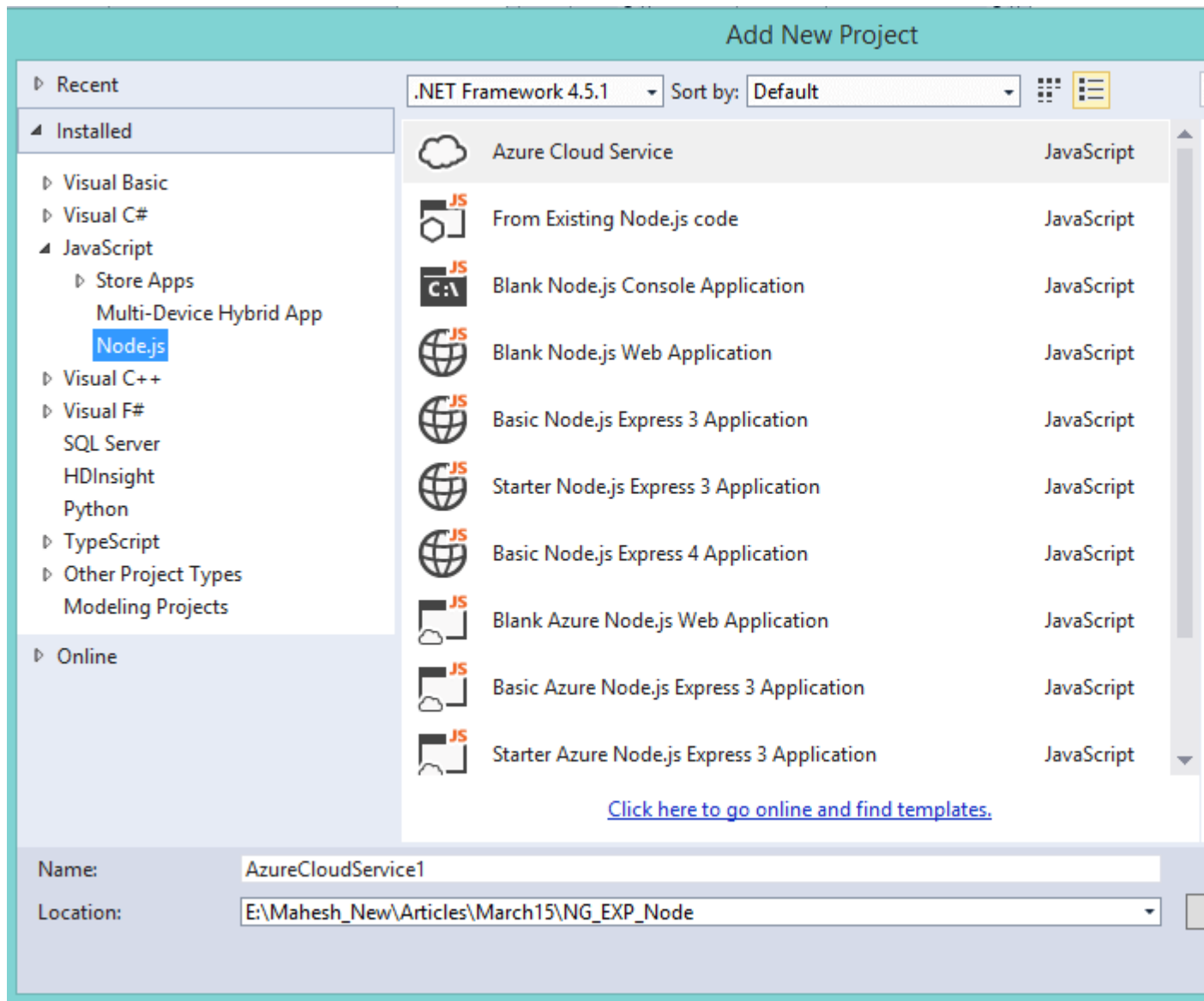
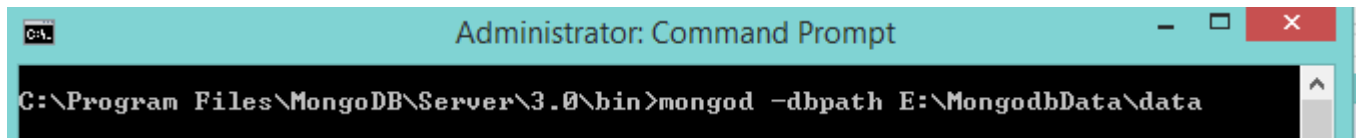


## Developing a REST API with Node.js and Consume it in an ASP.NET MVC application

In this section, we will create a REST API using Node.js and then consume this API in an ASP.NET MVC application. To implement this solution, we will use the Free [Visual Studio 2015 Community Edition](#). Node.js templates can be added to Visual Studio 2015 using [Node.js tools for Visual Studio \(NTVS\)](#). Once this extension is installed, Visual Studio will show Node.js templates as shown here:

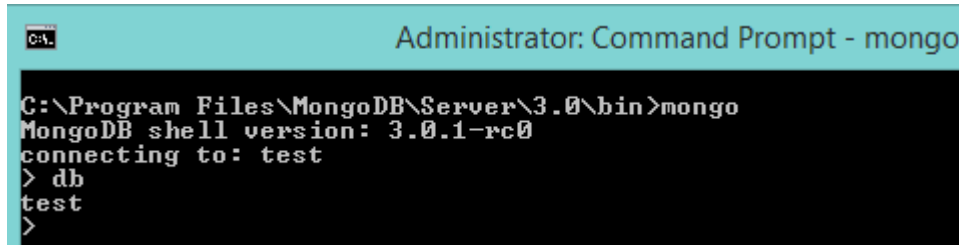


Our application will make use of MongoDB to store data. We can install MongoDB for windows by downloading it from this [link](#). We need to install and configure MongoDB using the steps from the link [here](#). Mongo can be installed in the %Program Files% folder and the *MongoDB* folder inside it. Using *mongod -dbpath* command, a connection with the MongoDB can be started as shown here:



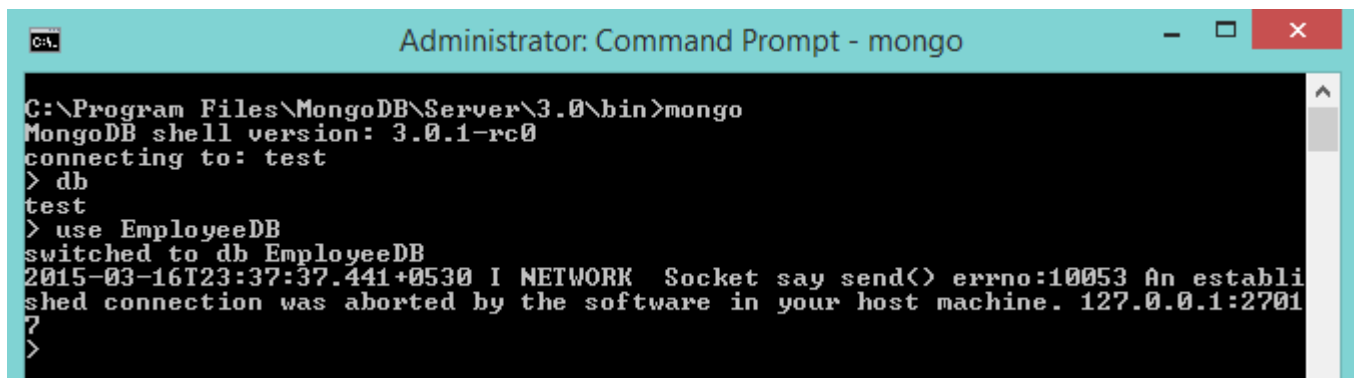
```
Administrator: Command Prompt
C:\Program Files\MongoDB\Server\3.0\bin>mongod -dbpath E:\MongodbdData\data
```

Using *mongo* command from the command prompt, the MongoDB shell can be connected with the default *test* database as shown in the following figure.



```
Administrator: Command Prompt - mongo
C:\Program Files\MongoDB\Server\3.0\bin>mongo
MongoDB shell version: 3.0.1-rc0
connecting to: test
> db
test
>
```

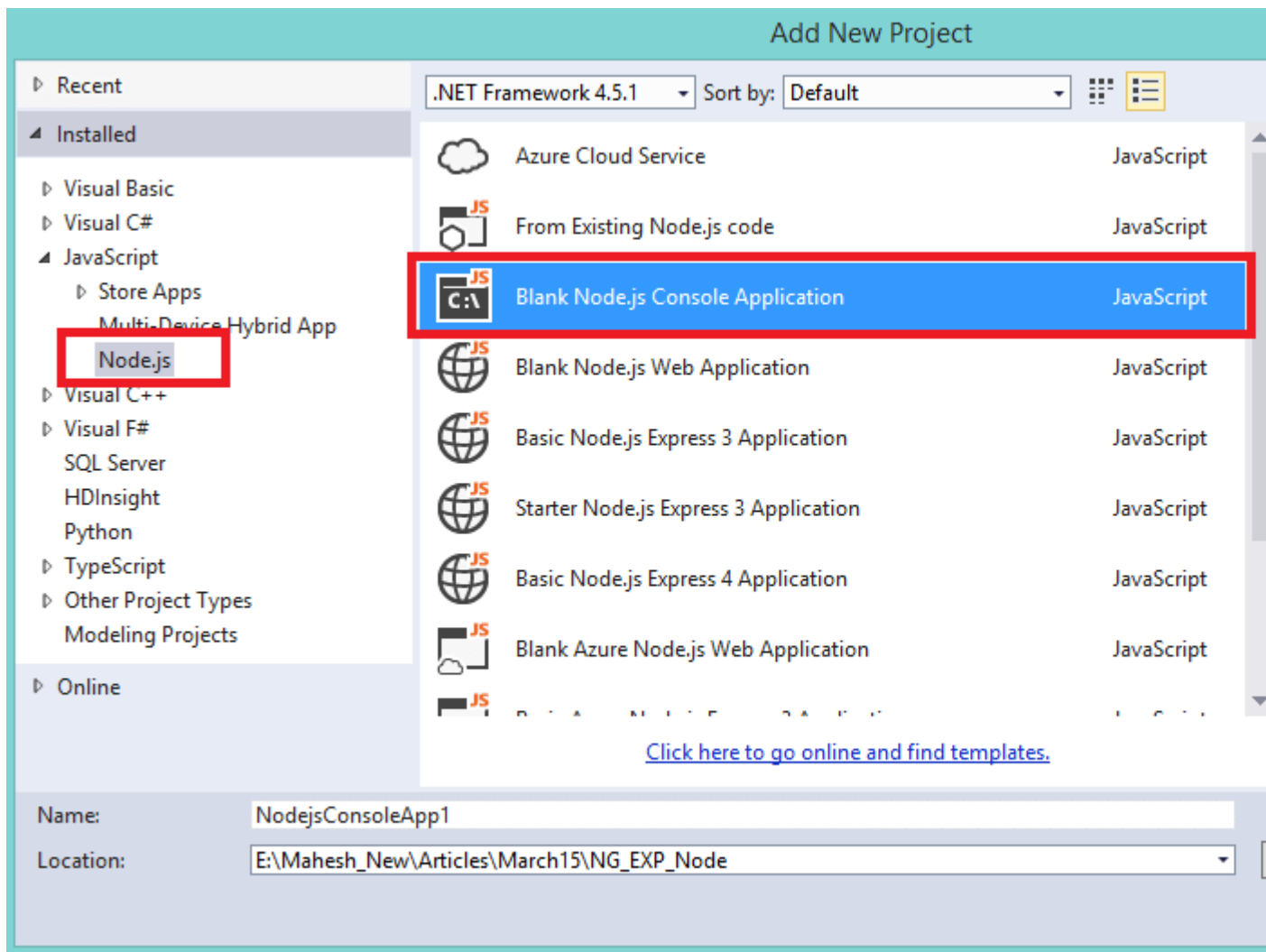
The New database can be created using the command *use*



```
Administrator: Command Prompt - mongo
C:\Program Files\MongoDB\Server\3.0\bin>mongo
MongoDB shell version: 3.0.1-rc0
connecting to: test
> db
test
> use EmployeeDB
switched to db EmployeeDB
2015-03-16T23:37:37.441+0530 I NETWORK  Socket say send() errno:10053 An establish
ed connection was aborted by the software in your host machine. 127.0.0.1:2701
?
>
```

The command *use EmployeeDB* creates a new database named EmployeeDB.

Step 1: Open Visual Studio and create a new blank Node.js console application and name it 'NG\_EXP\_Node' as shown in the following figure.



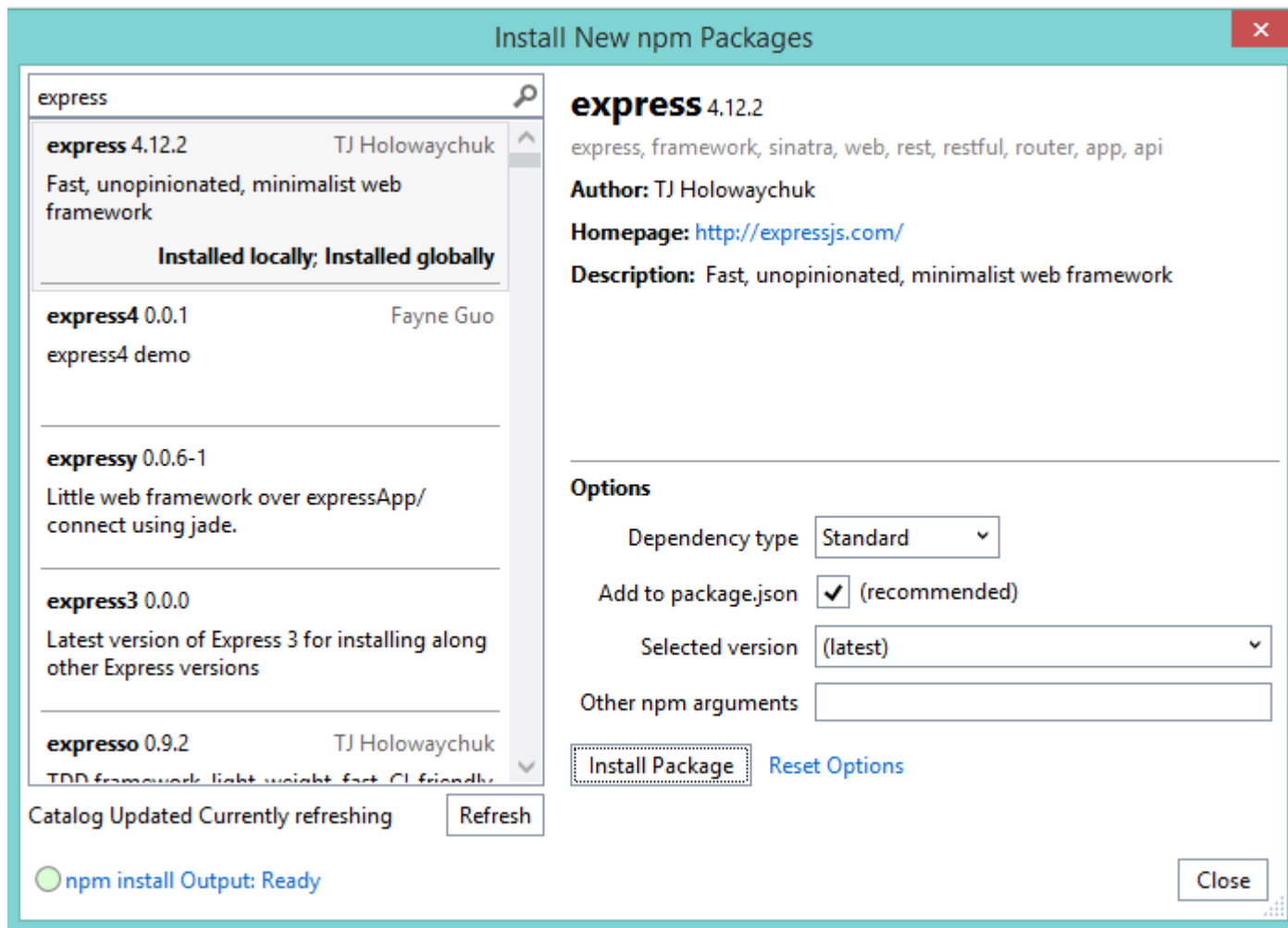
This step will create an app.js file by default in the project.

Step 2: Since we will be using MongoDB, Express and Mongoose in the project, we need to install these packages first.

*The Mongoose npm package allows to connect to MongoDB and helps to define the schema for storing data in MongoDB. This package also provides functions for performing CRUD operations against MongoDB.*

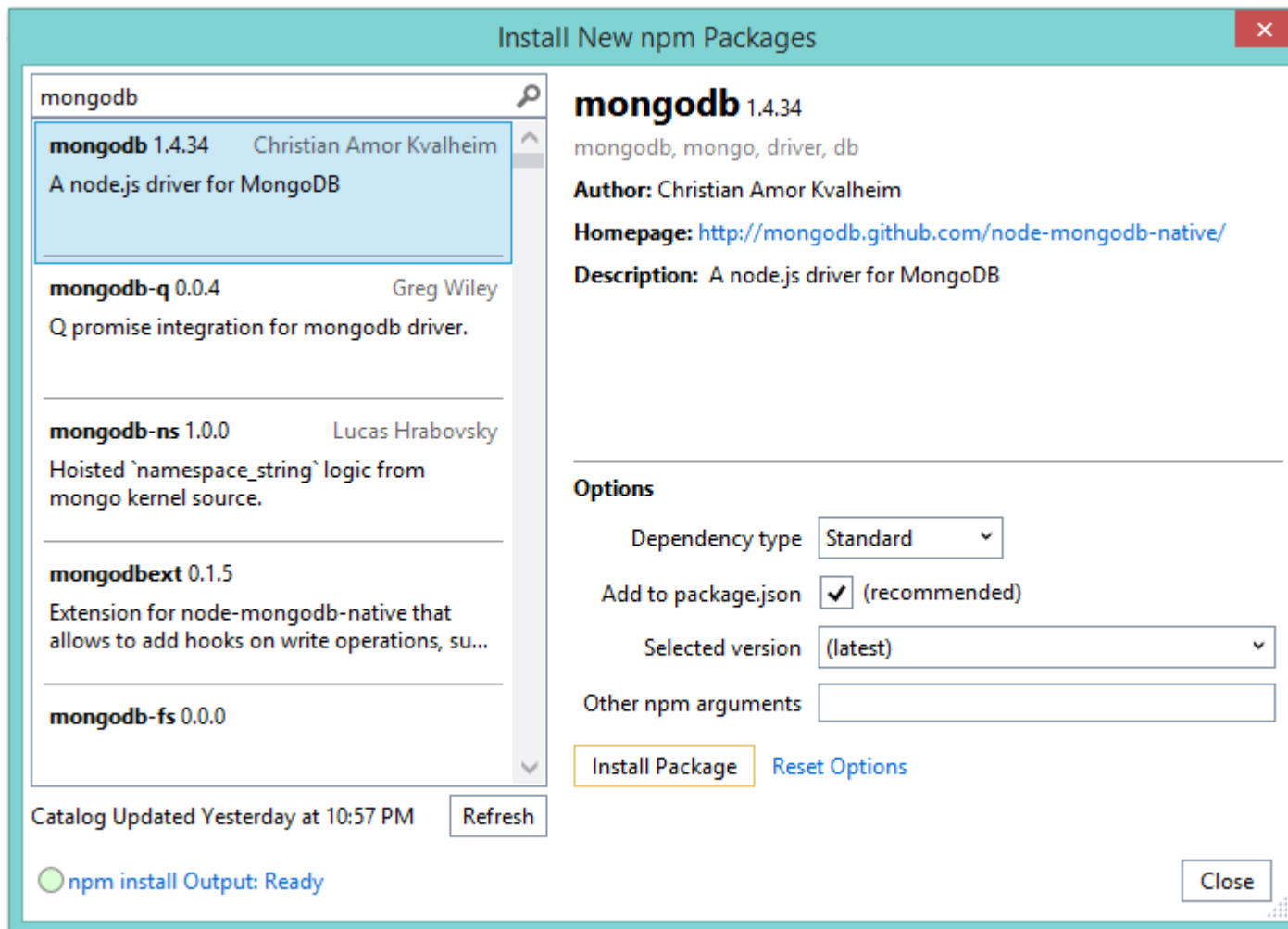
Right-click on the project and select the *Install new npm packages....* option which brings up the package window. Note that if you are doing this for the first time, it will first list all packages. Luckily the window has a *Search for package* text box using which the necessary packages can be searched. Add the packages as shown in the following figure:

Express.js



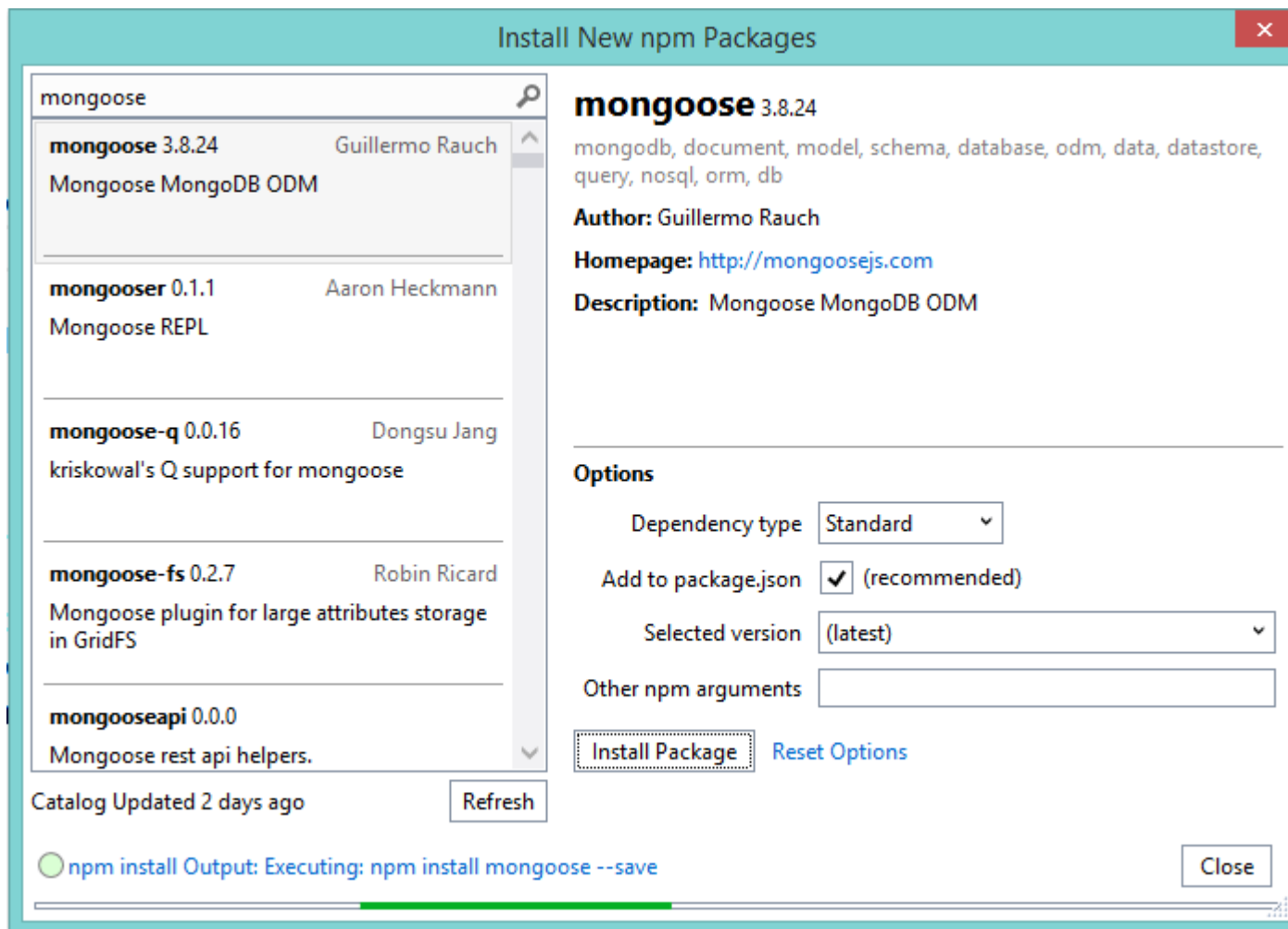
Click on Express and click on 'Install Package' to install the Express package.

MongoDB



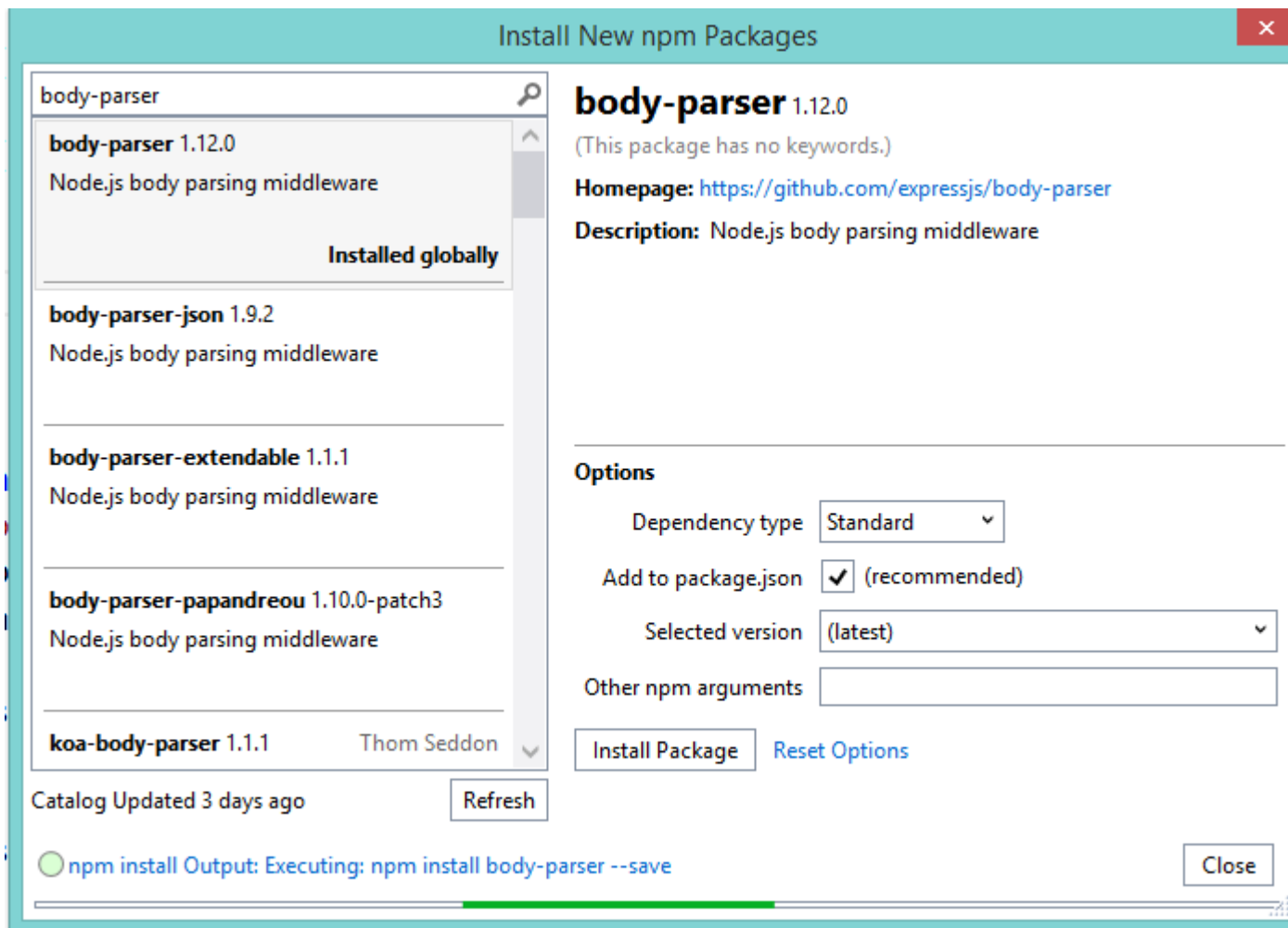
This is the driver for the MongoDB database access.

Mongoose



This will install the Mongoose npm package.

Body Parser



Body parser is a tool that gives you easy access to values submitted using a form. After these packages are added, the package.json file in the project will be updated with the following entries:

```
{
  "name": "NG_EXP_Node",
  "version": "0.0.0",
  "description": "NG_EXP_Node",
  "main": "app.js",
  "author": {
    "name": "Mahesh",
    "email": ""
  },
  "dependencies": {
    "body-parser": "^1.12.1",
    "bson": "^0.2.19",
    "express": "^4.12.2",
    "mongodb": "^1.4.34",
    "mongoose": "^3.8.25"
  }
}
```

Step 3: In the project, add a new JavaScript file named ReadWrite.js with the following code:

```
var mongooseDrv = require('mongoose'); //Get the Mongoose Driver
connection with the MongoDB
```

```
mongooseDrv.connect('mongodb://localhost/EmployeeDB');
```

```

var db = mongooseDrv.Connection; //The Connection

if (db == 'undefined') {
    console.log("The Connecion issues");
}

//The Schema for the Data to be Stored
var EmployeeInfoSchema = mongooseDrv.Schema({
    EmpNo: String,
    EmpName: String,
    Salary: String,
    DeptName: String,
    Designation: String
});

var EmployeeInfoModel = mongooseDrv.model('EmployeeInfo', EmployeeInfoSchema);

//retrieve all records from the database
exports.get = function (req, resp) {
    EmployeeInfoModel.find().exec(function (error, res) {
        if (error) {
            resp.send(500, {error:error});
        } else {
            resp.send(res);
        }
    });
};

//Add a new Record in the Employee Model
exports.add = function (request, response) {

    var newEmp = { EmpNo: request.body.EmpNo, EmpName: request.body.EmpName, Salary:
request.body.Salary, DeptName: request.body.DeptName, Designation:
request.body.Designation};
    EmployeeInfoModel.create(newEmp, function (addError, addedEmp) {
        if (addError) {
            response.send(500, { error: addError });
        }
        else {
            response.send({ success: true, emp: addedEmp });
        }
    });
};

```

The above JavaScript code uses the *Mongoose* driver to connect to the EmployeeDB database created in MongoDB. This also defines the *EmployeeInfoSchema* object containing the schema for storing Employee information. The methods *get()* and *add()* are used to read and write data received from the client application respectively.

Step 4: In the project, add the following code in app.js to load the necessary modules for Http utilities and REST APIs.

```

var body_Parser = require('body-parser'); //Load Modules for bodyparser
var path = require('path'); //Load 'path' the File base module
var express = require('express'); //Load express module
var http = require('http'); //Load Http module for Http operation

```



```

var app = express(); //The Express object

app.use(body_Parser());

var rwOperation = require('./ReadWrite.js'); //Load the File

var communicationPort = 8080; //The Communication port

//The REST API

app.get('/EmployeeList/api/employees', rwOperation.get);

app.post('/EmployeeList/api/employees', rwOperation.add);

```

```
app.listen(communicationPort);
```

This code loads *express* module for creating REST APIs. The `get()` and `post()` methods on the *express* application object are used to expose REST APIs to perform HTTP GET and POST methods respectively.

Step 5: Open the command prompt with Admin (Run as Administrator) rights. Navigate to the project folder and run the following command.

```
<project path="">:>node app.js
</project>
```

This will start the Node.js environment.

Step 6: In the same solution, add a new empty ASP.NET MVC application with the name *Node\_MVCCClient*. In this project, add JavaScript references for jQuery, Bootstrap and Angular.js using NuGet.

Step 7: In the Controllers folder, add a new empty MVC controller of the name *NodeAppController*. Scaffold an empty Index view from the Index method of this controller.

Step 8: In the Scripts folder add a new folder of name *MyScripts*. In this folder, add the following JavaScript files:

module.js

```

var app;
(function () {
    app = angular.module('mynodemodule', []);
})();

```

service.js

```

app.service('mynodeservice', function ($http) {
    this.get = function() {
        var res = $http.get("http://localhost:8080/EmployeeList/api/employees");
        return res;
    }

    this.post = function (emp) {
        var res = $http.post("http://localhost:8080/EmployeeList/api/employees", emp);
        return res;
    }

});

```

The above code defines methods for making a call to the REST APIs using Node.js application.

Controller.js

```
app.controller('mynodecontroller', function ($scope, mynodeservice) {

    //The Employee Object used to Post
    $scope.Employee = {
        _id:"",
        EmpNo: 0,
        EmpName: "",
        Salary: 0,
        DeptName: "",
        Designation:""
    }

    loaddata();
    //Function to load all records
    function loaddata() {
        var promise = mynodeservice.get();

        promise.then(function (resp) {
            $scope.Employees = resp.data;
            $scope.message = "Call is Successful ";
        }, function (err) {
            $scope.message = "Error in Call" + err.status
        });
    };

    //Function to POST Employee
    $scope.save = function () {
        var promisePost = mynodeservice.post($scope.Employee);

        promisePost.then(function (resp) {
            $scope.message = "Call is Successful";
            loaddata();
        }, function (err) {
            $scope.message = "Error in Call" + err.status
        });
    };

    $scope.clear = function () {
        $scope.Employee.EmpNo = 0;
        $scope.Employee.EmpName = "";
        $scope.Employee.Salary = 0;
        $scope.Employee.DeptName = "";
        $scope.Employee.Designation = "";
    };

});
```

The above code contains functions for reading all data using REST APIs and performing POST operations.

Step 9: In the Index.cshtml add the following markup

```
<script src=""~/Scripts/jquery-2.1.3.min.js""></script>
<script src=""~/Scripts/bootstrap.min.js""></script>
<script src=""~/Scripts/angular.min.js""></script>
<script src=""~/Scripts/MyScripts/module.js""></script>
```

```
<script src="../../../Scripts/MyScripts/service.js"></script>
<script src="../../../Scripts/MyScripts/controller.js"></script>
```

```
<h2>Index</h2>
```

```
<table class="table table-bordered">
```

```
  <tbody><tr>
```

```
    <td>
```

```
      EmpNo
```

```
    </td>
```

```
    <td>
```

```
      <input type="text" class="form-control" ng-model="Employee.EmpNo"/>
```

```
    </td>
```

```
  </tr>
```

```
  <tr>
```

```
    <td>
```

```
      EmpName
```

```
    </td>
```

```
    <td>
```

```
      <input type="text" class="form-control" ng-model="Employee.EmpName">
```

```
    </td>
```

```
  </tr>
```

```
  <tr>
```

```
    <td>
```

```
      Salary
```

```
    </td>
```

```
    <td>
```

```
      <input type="text" class="form-control" ng-model="Employee.Salary">
```

```
    </td>
```

```
  </tr>
```

```
  <tr>
```

```
    <td>
```

```
      DeptName
```

```
    </td>
```

```
    <td>
```

```
      <input type="text" class="form-control" ng-model="Employee.DeptName">
```

```
    </td>
```

```
  </tr>
```

```
  <tr>
```

```
    <td>
```

```
      Designation
```

```
    </td>
```

```
    <td>
```

```
      <input type="text" class="form-control" ng-model="Employee.Designation">
```

```
    </td>
```

```
  </tr>
```

```
  <tr>
```

```
    <td>
```

```
      <input type="button" value="New" class="btn btn-primary" ng-
```

```
click="clear()"/>
```

```
    </td>
```

```
    <td>
```

```
      <input type="button" value="Save" class="btn btn-success" ng-
```

```
click="save()"/>
```

```
    </td>
```

```
  </tr>
```

```
</tbody></table>
```

```
<table class="table table-bordered table-condensed">
```

```

<thead>
  <tr>
    <td>
      ID
    </td>
    <td>
      EmpNo
    </td>
    <td>
      EmpName
    </td>
    <td>
      Salary
    </td>
    <td>
      DeptName
    </td>
    <td>
      Designation
    </td>
  </tr>
</thead>
<tbody ng-repeat="Emp in employees">
  <tr>
    <td>{{Emp._id}}</td>
    <td>{{Emp.EmpNo}}</td>
    <td>{{Emp.EmpName}}</td>
    <td>{{Emp.Salary}}</td>
    <td>{{Emp.DeptName}}</td>
    <td>{{Emp.Designation}}</td>
  </tr>
</tbody>
</table>
<div>
  {{Message}}
</div>

```

This markup contains Angularjs databinding using various directives.

Run the application and you should get the following result:

# Index

EmpNo	<input type="text" value="0"/>
EmpName	<input type="text"/>
Salary	<input type="text" value="0"/>
DeptName	<input type="text"/>
Designation	<input type="text"/>
<input type="button" value="New"/>	<input type="button" value="Save"/>

ID	EmpNo	EmpName	Salary
5506b2f8ce2d7d7824f9e775	101	TS	45000
5506b8a20c857f801c393cb9	102	MS	47000

Enter values in the TextBoxes and click on the *Save* button. The table will display the newly added record.

## CONCLUSION

This article demonstrated how Node.js provides server-side JavaScript features for REST APIs and how it can be consumed by any client application. This article aimed at helping ASP.NET developers learn the essentials of Node.js web development using the MEAN stack.