



COLLEGE OF ENGINEERING AND COMPUTER STUDIES

Final Exam: CRUD OPERATION

A requirement submitted to the College of Engineering and Computer Studies as a requirement for the completion of the Professional Elective 3 – IoT Fundamentals

EMPLOYEE CRUD OPERATION USING NODE JS AND MONGOOSE

Submitted By

Jan Dale Austria

Denzel Joseph Tolosa

Jan Myrone Alano

Junell Roxas

Joshua Molino

Dave Pagkaliwagan

Course & Section

BSIT 3-1

Date

July 2021



TABLE OF CONTENTS

- I. INTRODUCTION
- II. METHODS
- III. RESULT
- IV. DISCUSSION



I. Introduction

Nowadays, different IT systems are made by people who use different programming languages to build a system and different database application to know the users information, one of which is that they use a database to find out the system or we can see what they are inputting data or information that attach in the database, many database applications can also be used, and nowadays many IT experts or people use XAMPP SQL database, because it is easy to use, and other database app is the mongo Db, we all know the mongo db. it manages relationships between data, provides schema validation, and is used to translate between objects in code and the representation of those objects in MongoDB. and in programming language many types of programming language can be used to access database, such as JavaScript, C#, and node.js and other programming languages.

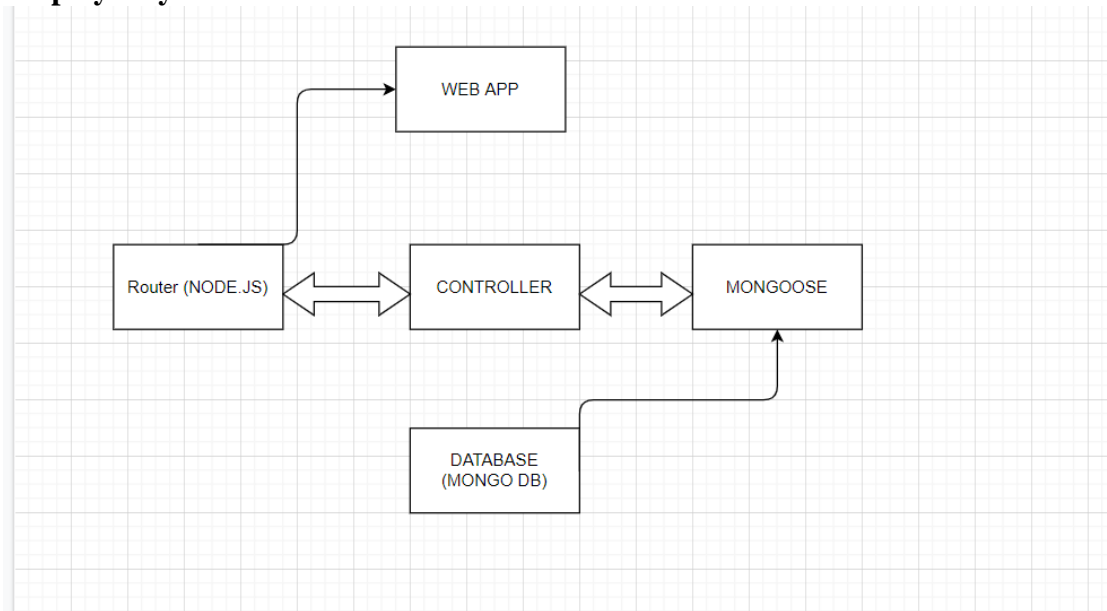
Many also now see in a system is the CRUD operation, We all know when we hear CRUD operation what immediately ran through our minds was CREATE, READ, UPDATE and DELETE, this are use in the programming, like in the XML, Database, and SQL, first what is CREATE in the CRUD operation, it performs the INSERT to create a new record of data, and the READ in the CRUD operation is to GET method, the read can know the source of the information, and if we create a table, it reads the table record based on the primary key, the UPDATE in the CRUD operation is to update the records, and it performs the UPDATE statement on the table based on the primary key for a record specified in WHERE clause statement, the DELETE in the CRUD operation is to delete a data, and row specified in WHERE CLAUSE, and I shared my experience in the CRUD operation, my first experience is I used the language of XML, in the first place, It is hard to code because you have no idea what is CRUD, and my professor discuss to us, and I think he can use C# and XML, and I follow him to get an idea about the CRUD operation, when he finished to create the CRUD operation, I decided to do the CRUD.

And as BSIT students we want to make a simple system, we will create CRUD operation using mongoose. And node.js, we can use this because this is the best tool that we use in our project because mongoose is an object document modeling (ODM) layer that sits on top of Node's MongoDB driver. If you are coming from SQL, it's similar to an ORM for a relational database. While it is not required to use Mongoose with the Mongo, here are four reasons why using Mongoose with MongoDB is generally a good idea.

II. Methods

Planning Phase

Employee system



The router receives http requests from the web application system and forwards them to the controller, which in turn creates, reads, updates, and deletes data models defined with the mongoose library, the user profiles and sessions information will reside in a mongo DB, the router also receives data from the controller and bundles it in http responses that it sends to the web system.

To create the CRUD operation, we use to download some files in the visual studio, and the programming language that we used is the NODE.js and in the database we used mongoose.

Steps for download in package.json.

- 1) NPM init
- 2) And we input server.js.
- 3) Npm I --s express@4.16.4 mongoose@5.3.4 express-handlebars@3.0.0 body parser@1.18.3

Steps for download in nodemon

- 1) Npm install nodemon
- 2) Auditfix
- 3) Nodemon server.js



After download the necessary files, we proceed in the coding.

CODE OF THE SYSTEM

List.hbs:

```

<h3><a class="btn btn-secondary" href="/employee"><i class="fa fa-plus"></i>
Create New</a> Employee List</h3>
<table class="table table-striped">
  <thead>
    <tr>
      <th>Full Name</th>
      <th>Email</th>
      <th>Contact Number</th>
      <th>Address</th>
      <th></th>
    </tr>
  </thead>
  <tbody>
    <#each list>
      <tr>
        <td>{{fullName}}</td>
        <td>{{email}}</td>
        <td>{{mobile}}</td>
        <td>{{city}}</td>
        <td>
          <a href="/employee/{{this._id}}"><i class="fa fa-pencil fa-lg" aria-
hidden="true"></i></a>
          <a href="/employee/delete/{{this._id}}" onclick="return confirm('Are you
sure to delete this record ?');"><i class="fa fa-trash fa-lg" aria-
hidden="true"></i></a>
        </td>
      </tr>
    </each>
  </tbody>
</table>

```

AddorEdit.hbs:

```

<h3>{{viewTitle}}</h3>

<form action="/employee" method="POST" autocomplete="off">
  <input type="hidden" name="_id" value="{{employee._id}}">
  <div class="form-group">
    <label>Full Name</label>
    <input type="text" class="form-control" name="fullName"
placeholder="Full Name" value="{{employee.fullName}}">

```



```

<div class="text-danger">
  {{employee.fullNameError}}</div>
</div>
<div class="form-group">
  <label>Email</label>
  <input type="text" class="form-control" name="email"
placeholder="Email" value="{{employee.email}}">
  <div class="text-danger">
    {{employee.emailError}}</div>
</div>
<div class="form-row">
  <div class="form-group col-md-6">
    <label>Contact Number</label>
    <input type="text" class="form-control" name="mobile"
placeholder="Contact Number" value="{{employee.mobile}}">
  </div>
  <div class="form-group col-md-6">
    <label>Address</label>
    <input type="text" class="form-control" name="city"
placeholder="Address" value="{{employee.city}}">
  </div>
</div>
<div class="form-group">
  <button type="submit" class="btn btn-success"><i class="fa fa-laptop"></i>
Submit</button>
  <a class="btn btn-dark" href="/employee/list"><i class="fa fa-users"></i>
View All</a>
</div>
</form>
MainLayout.hbs:
<!DOCTYPE html>

<html>

<head>
  <title>Employee Data with CRUD and NodeJS</title>
  <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
integrity="sha384-
MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdk
nLPMO"
crossorigin="anonymous">
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-
awesome/4.7.0/css/font-awesome.min.css">
</head>

```



```
<body class="bg-success">
  <div class="row">
    <div class="col-md-6 offset-md-3" style="background-color: #fff;margin-top:
220px;padding:20px;">
      {{{body}}}
    </div>
  </div>
</body>
```

```
</html>
```

Server.js:

```
require('./models/db');

const express = require('express');
const path = require('path');
const Handlebars = require('handlebars');
const exphbs = require('express-handlebars');
const bodyparser = require('body-parser');
const {allowInsecurePrototypeAccess} = require('@handlebars/allow-prototype-
access');

const employeeController = require('./controllers/employeeController');

var app = express();
app.use(bodyparser.urlencoded({
  extended: true
}));
app.use(bodyparser.json());
app.set('views', path.join(__dirname, '/views/'));
app.engine('hbs', exphbs({ extname: 'hbs', defaultLayout: 'mainLayout',
layoutsDir: __dirname + '/views/layouts/', handlebars:
allowInsecurePrototypeAccess(Handlebars) }));
app.set('view engine', 'hbs');

app.listen(3000, () => {
  console.log('Express server started at port : 3000');
});

app.use('/employee', employeeController);
```

Employeecontroller.js:

```
const express = require('express');
```



```
var router = express.Router();
const mongoose = require('mongoose');
const Employee = mongoose.model('Employee');

router.get('/', (req, res) => {
  res.render("employee/addOrEdit", {
    viewTitle: "Insert Employee"
  });
});

router.post('/', (req, res) => {
  if (req.body._id == "")
    insertRecord(req, res);
  else
    updateRecord(req, res);
});

function insertRecord(req, res) {
  var employee = new Employee();
  employee.fullName = req.body.fullName;
  employee.email = req.body.email;
  employee.mobile = req.body.mobile;
  employee.city = req.body.city;
  employee.save((err, doc) => {
    if (!err)
      res.redirect('employee/list');
    else {
      if (err.name == 'ValidationError') {
        handleValidationError(err, req.body);
        res.render("employee/addOrEdit", {
          viewTitle: "Insert Employee",
          employee: req.body
        });
      }
      else
        console.log('Error during record insertion : ' + err);
    }
  });
}

function updateRecord(req, res) {
  Employee.findOneAndUpdate({ _id: req.body._id }, req.body, { new: true }, (err, doc) => {
    if (!err) { res.redirect('employee/list'); }
  });
}
```




```
else {
  if (err.name == 'ValidationError') {
    handleValidationError(err, req.body);
    res.render("employee/addOrEdit", {
      viewTitle: 'Update Employee',
      employee: req.body
    });
  }
  else
    console.log('Error during record update : ' + err);
}
});
}

router.get('/list', (req, res) => {
  Employee.find((err, docs) => {
    if (!err) {
      res.render("employee/list", {
        list: docs
      });
    }
    else {
      console.log('Error in retrieving employee list : ' + err);
    }
  });
});

function handleValidationError(err, body) {
  for (field in err.errors) {
    switch (err.errors[field].path) {
      case 'fullName':
        body['fullNameError'] = err.errors[field].message;
        break;
      case 'email':
        body['emailError'] = err.errors[field].message;
        break;
      default:
        break;
    }
  }
}

router.get('/:id', (req, res) => {
```



```
Employee.findById(req.params.id, (err, doc) => {
  if (!err) {
    res.render("employee/addOrEdit", {
      viewTitle: "Update Employee",
      employee: doc
    });
  }
});

router.get('/delete/:id', (req, res) => {
  Employee.findByIdAndRemove(req.params.id, (err, doc) => {
    if (!err) {
      res.redirect('/employee/list');
    }
    else { console.log('Error in employee delete :' + err); }
  });
});

module.exports = router;
db.js:
const mongoose = require('mongoose');

mongoose.connect('mongodb://localhost:27017/EmployeeDB', {
  useUnifiedTopology: true,
  useNewUrlParser: true, }, (err) => {
  if (!err) { console.log('MongoDB Connection Succeeded.') }
  else { console.log('Error in DB connection : ' + err) }
});

require('./employee.model');

employee.model.js:
const mongoose = require('mongoose');
var employeeSchema = new mongoose.Schema({
  fullName: {
    type: String,
```



```

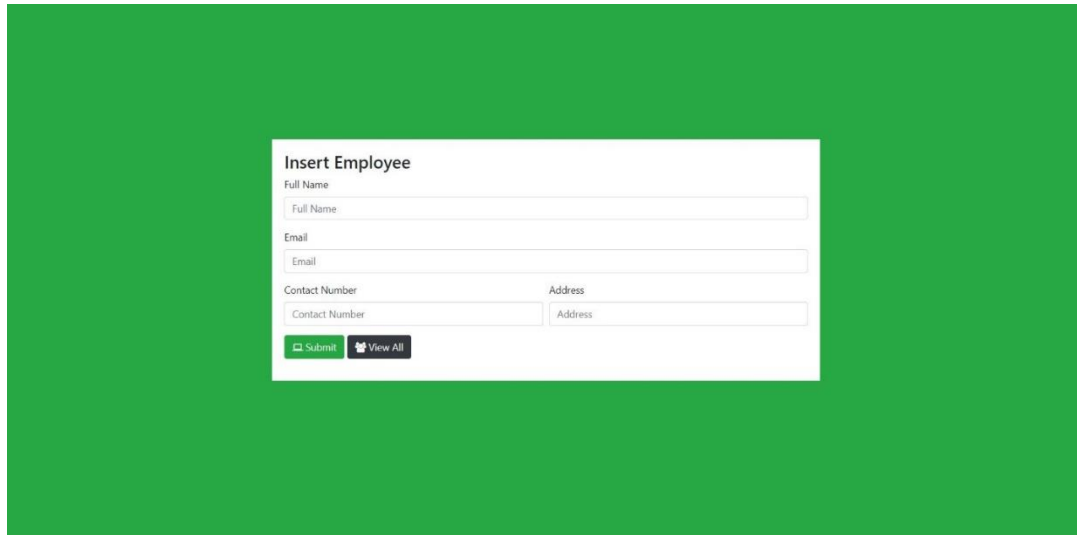
    required: 'This field is required.'
  },
  email: {
    type: String,
    required: 'This field is required.'
  },
  mobile: {
    type: String,
    required: 'This field is required.'
  },
  city: {
    type: String,
    required: 'This field is required.'
  }
});
// Custom validation for email
employeeSchema.path('email').validate((val) => {
  emailRegex = /^[^<>()\[\]\\\.,;:\s@"]+(\.[^<>()\[\]\\\.,;:\s@"]+)*("\.[^"]*"|'[^']*')@[([0-9]{1,3}\.){0-9}[0-9]{1,3}\.([0-9]{1,3}\.){0-9}[0-9]{1,3}|([a-zA-Z-0-9]+\.)+[a-zA-Z]{2,})$/;
  return emailRegex.test(val);
}, 'Invalid e-mail.');
```

```
mongoose.model('Employee', employeeSchema);
```

III. Results of the system

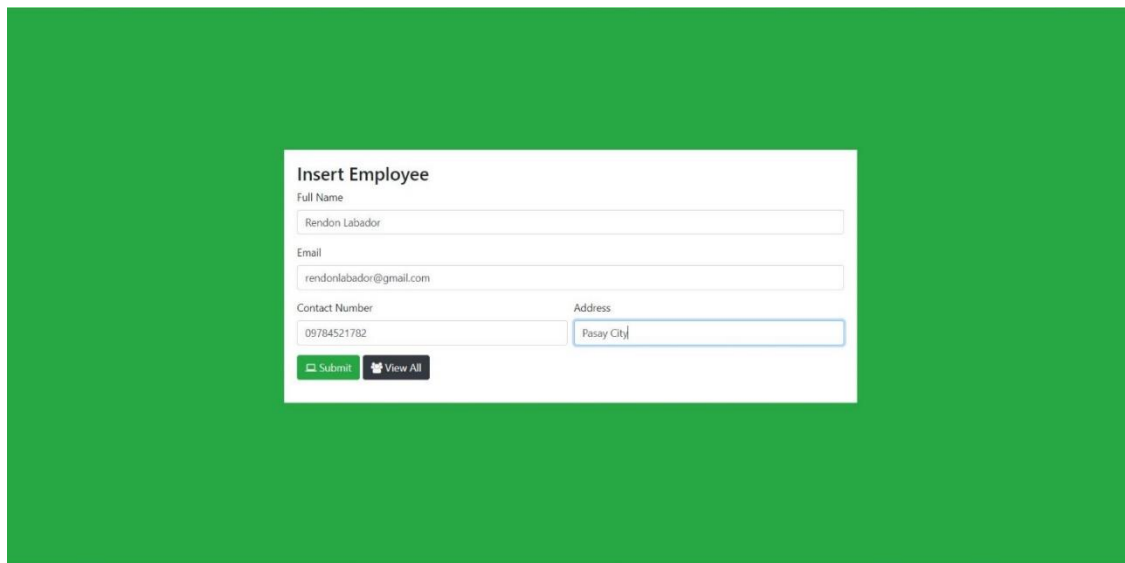
CRUD Operation.

UI OF THE SYSTEM:



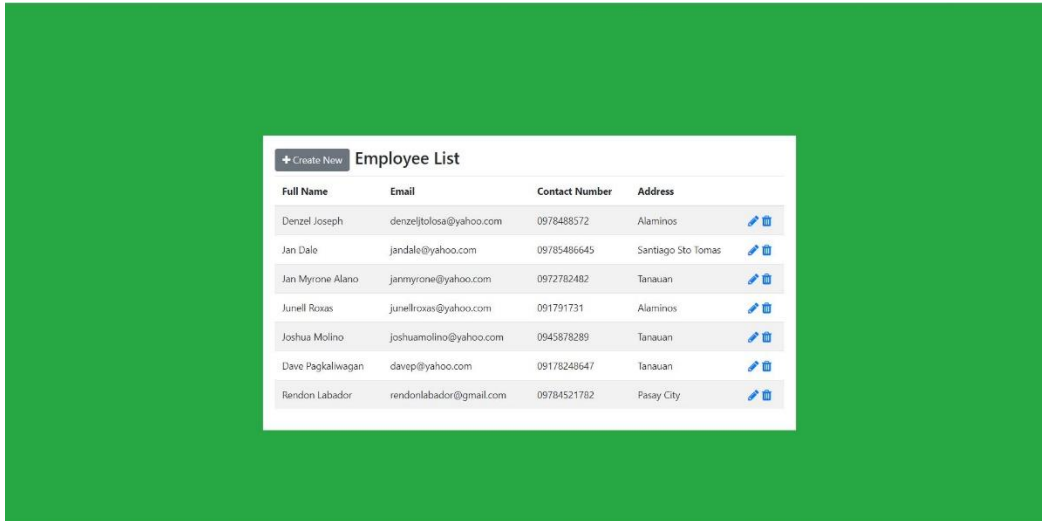
The screenshot shows a web form titled "Insert Employee" on a green background. The form has the following fields: "Full Name" (text input), "Email" (text input), "Contact Number" (text input), and "Address" (text input). At the bottom, there are two buttons: "Submit" and "View All".








ADDING A RECORD TO THE SYSTEM:



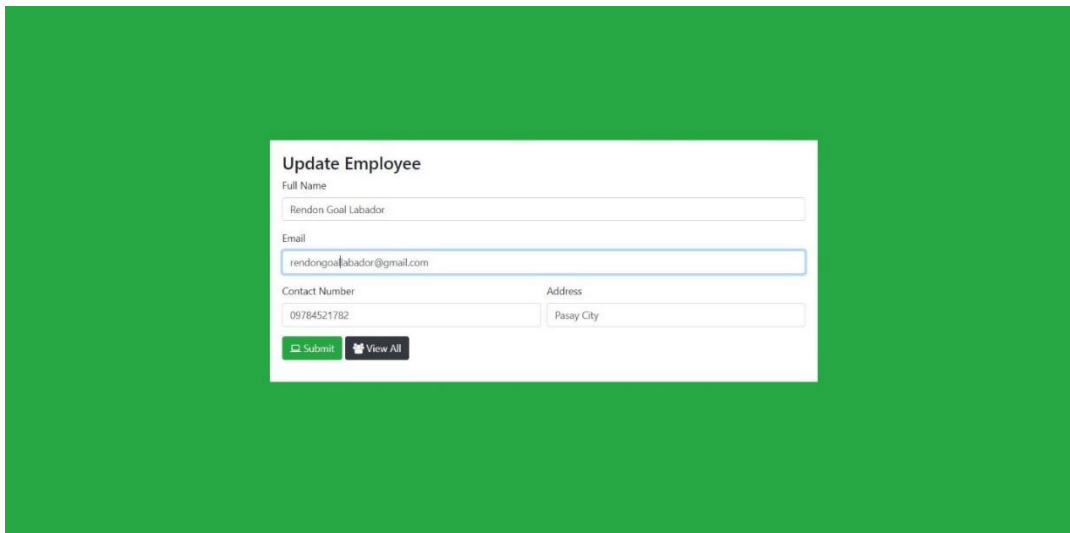
The screenshot shows the same "Insert Employee" form with data entered into the fields: "Full Name" is "Rendon Labrador", "Email" is "rendonlabador@gmail.com", "Contact Number" is "09784521782", and "Address" is "Pasay City". The "Submit" and "View All" buttons are visible at the bottom.

LIST AFTER ADDING THE RECORD:



Full Name	Email	Contact Number	Address	
Denzel Joseph	denzeljtolosa@yahoo.com	0978488572	Alaminos	
Jan Dale	jandale@yahoo.com	09785486645	Santiago Sto Tomas	
Jan Myrone Alano	janmyrone@yahoo.com	0972782482	Tanauan	
Junell Roxas	junellroxas@yahoo.com	091791731	Alaminos	
Joshua Molino	joshuamolino@yahoo.com	0945878289	Tanauan	
Dave Pagkaliwagan	davep@yahoo.com	09178248647	Tanauan	
Rendon Labador	rendonlabador@gmail.com	09784521782	Pasay City	

UPDATING THE RECORD:



Update Employee

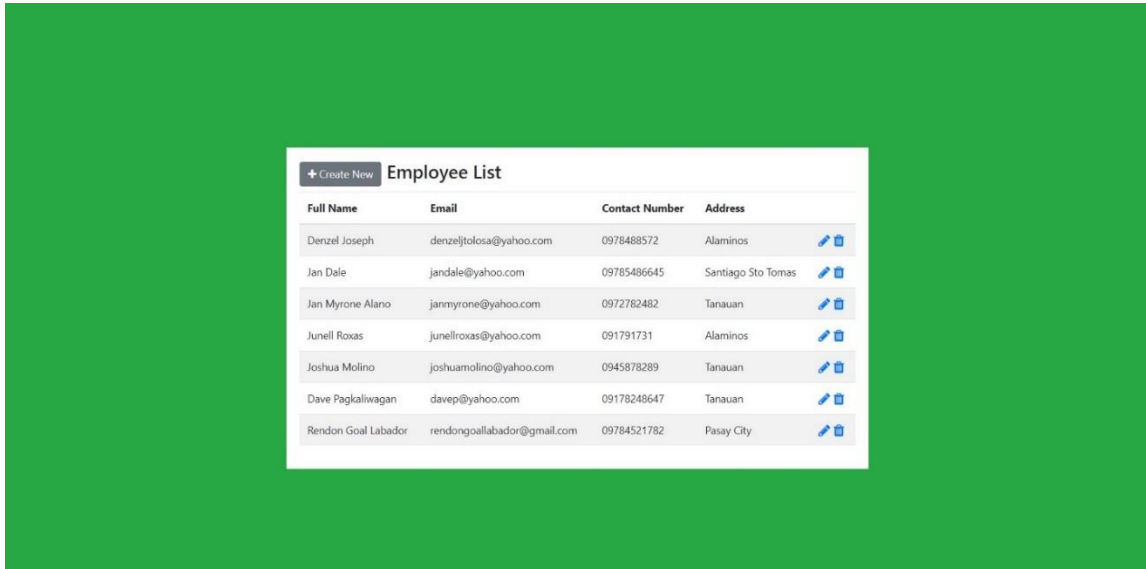
Full Name

Email

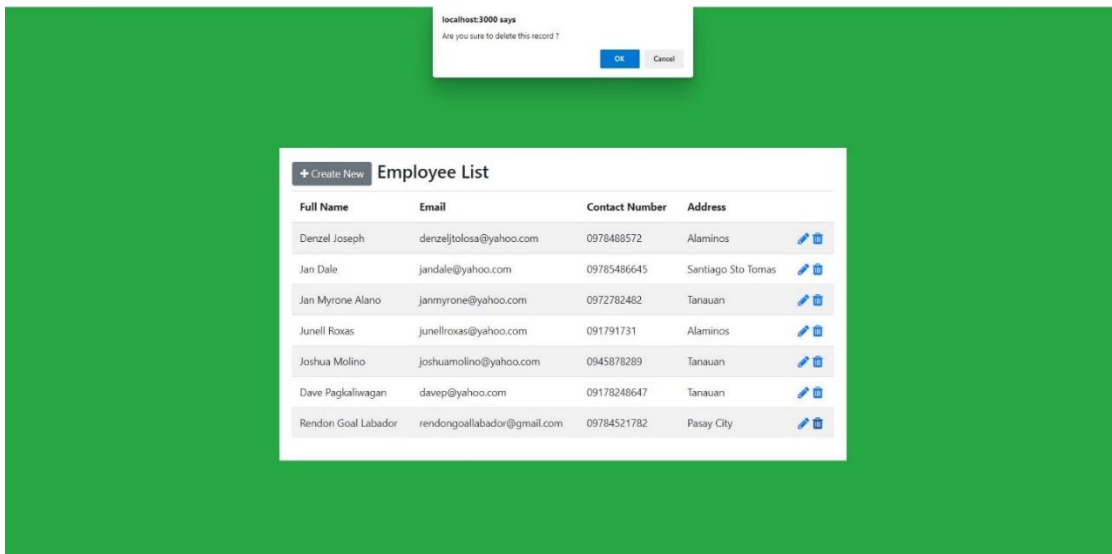
Contact Number

Address

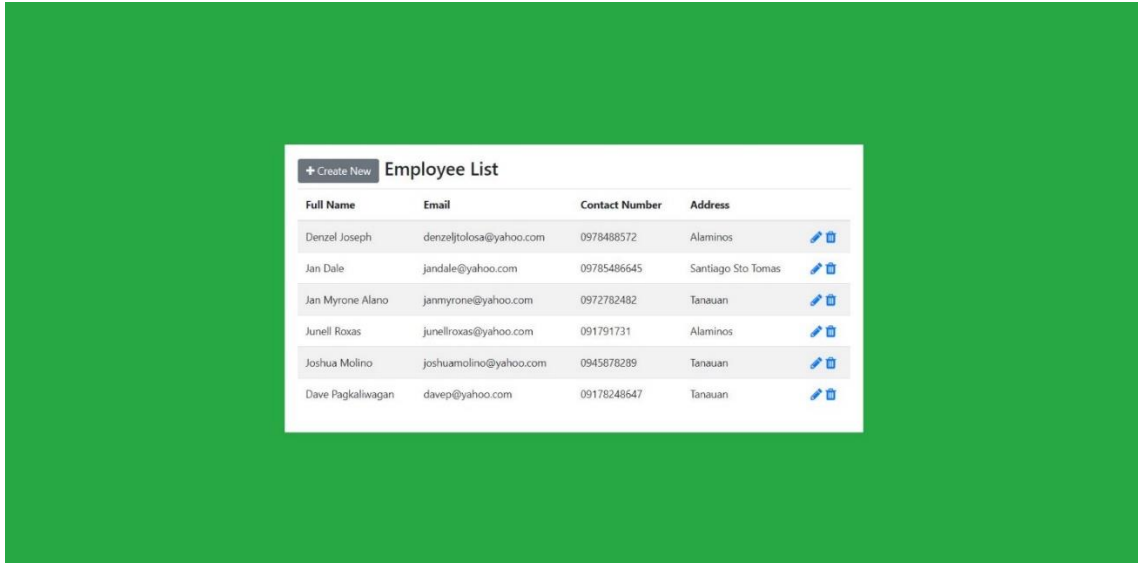
AFTER SUCCESSFULLY UPDATING THE RECORD:



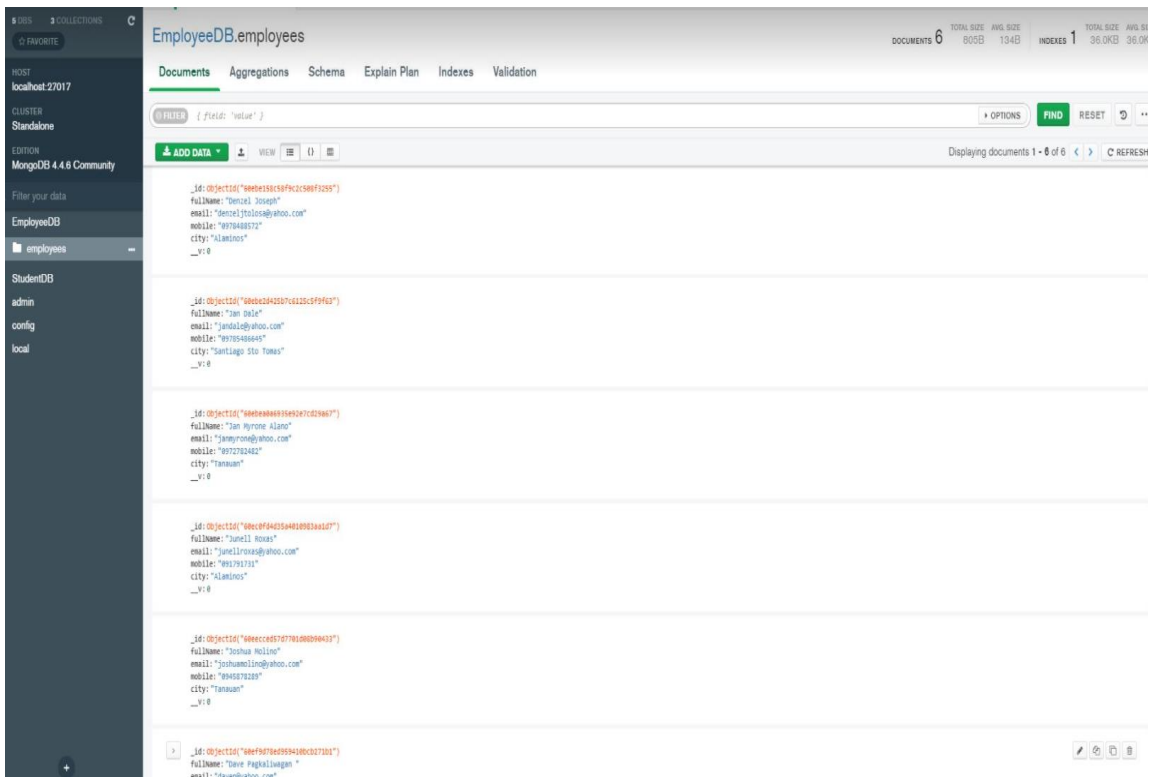
DELETING THE RECORD:



AFTER SUCCESSFULLY DELETE THE RECORD:



MONGO DB





VI. Discussion of the process system.

In the first process we show our UI design for our simple project, and next process is to add record, the next process we show the record, the next process is to update the record, and we show the results of updating the record, and the last is the delete, we show the delete button to remove record, and after that, it is successfully deleted the records, after that we show the database, It shows the data of the employee.