My First Express App:: Intro To Databases

Databases:: An Introduction

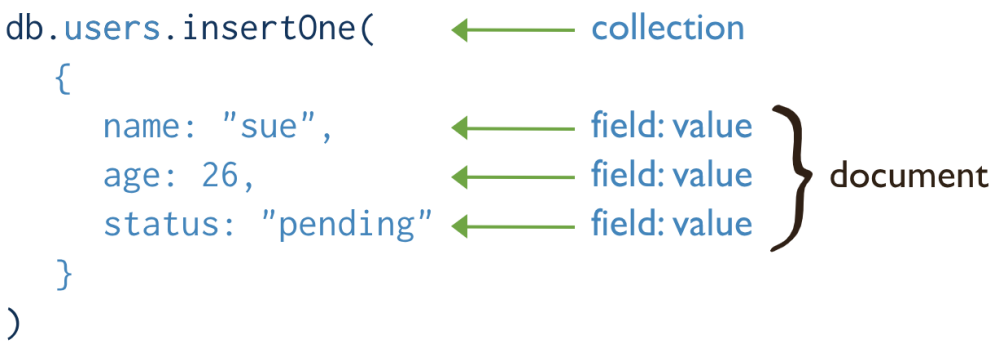
- The contact list that we have created is always there in the RAM. Whenever we restarted the server, all the changes were gone and we don't want that in real life.
- To solve this problem we have databases that store all the data permanently.
- The most popular databases are - MySql, Postgres, Firebase - {Product of Google } , Redis,MongoDb.
- Databases are the persistent storage so whenever we restart the server there is no chance of changes getting removed.
- There can be two ways of storing the data - Tabular format - {like - MySql, Postgres,} or BJSON (Binary JSON) - { like - Firebase , Redis,MongoDb }.

SQL and NoSQL Databases

- In tabular databases, we store the data in tabular format and store them in the form of rows, and columns.

EmployeeValues		
emp_nbr	emp_property	value
101	first_name	David
101	last_name	Bristol
101	date_of_birth	11/2/1946
102	first_name	Mary
102	last_name	Manning
102	date_of_birth	10/10/1951
103	first_name	Alistair
103	last_name	Ross
103	date_of_birth	5/21/1966

- In JSON - (Javascript Object Notation) databases, we store the data in the form of key-value pairs that makes accessibility faster.



Why MongoDB

We have chosen MongoDB as the database for the project as using MongoDb we get the following advantages - :

- Ease of use - we don't need to define the tabular structure inside which rows and columns.
- When we are handling very large-scale databases SQL is comparatively slower than NoSQL.
- MongoDB gives us two important stacks - MEAN and MERN stack - {Mongo, Express, Angular, React, Node }.
- MongoDB is a new technology and quite popular these days.

***EXTRA*: You can study more from the link below -**
https://docs.mongodb.com/

MongoDB :: Terms

- MongoDB stores the data in the form of JSON.
- Suppose we have to store the details of a student in the JSON format. So all the characteristics of a student will be in the form of **Field** and each of the Field has a value.
- The particular entry in the database for each student is known as a document.
- The list of students that contains all the students is known as the **Collection**.
- A database can have multiple collections.
- A collection of these **Collections** is known as the database.

Installing MongoDB, Robo3T, and Moongoosejs

- We need to install MongoDB and check whether it is installed or not.

  ***EXTRA*: You can install MongoDB according to the OS you are using.**

  https://www.mongodb.com/try/download/community

- To check that the MongoDB has been installed, type the command **{ mongo }** in the terminal. This will take you to the Mongo Shell if the MongoDB has been installed.
- We need to install Robo3T - { It is a tool to visualize the database }.

  ***EXTRA*: You can install Robo3T according to the OS you are using.**

  https://robomongo.org/

- We need to install Mongoose.
- To install mongoose we need to quit the mongo shell and navigate to the project directory.
- By using the command { npm install mongoose } in the terminal mongoose will be installed.
- To check whether it is being installed we need to check the { **package.json }** file.
- We need to set up the configuration file for mongoose.
- Finally, we have to run the server and test whether it is working or not.

Connecting to MongoDB using Mongoose

- We have to set up the configuration so that the express server can interact with the database { **MonogDB** } using the layer in between that is called **Mongoose**.
- We need to create another folder in the project and name it as { **config** }.
- Inside the config folder, we need to create a file { **mongoose.js** }.
- Inside the file, we need to require the mongoose.

  ***EXTRA*: You can read about Mongoose using the following link.**

  https://mongoosejs.com/

- We need to connect the mongoose to the database using the (.connect) method.
- We need to acquire the connection to (to check if it is successful)
- If there is an error while connecting, we throw the error.
- If there is no error then print the message successfully connected to the database.
- Include the mongoose file whenever the server is firing up inside the **{ index.js }** file.

{ **mongoose.js** }.

```
//require the library
const mongoose = require('mongoose');

//connect to the database
mongoose.connect('mongodb://localhost/contact_list_db');

//acquire the connection(to check if it's successful)
const db = mongoose.connection;

//error
db.on('error', function(err) { console.log(err.message); });

//up and running then print the message
db.once('open', function() {

    console.log("Successfully connected to the database");

});
```

{ index.js }

```
const db = require('./config/mongoose');
```

Creating the DB Schema

- In the contact list, there are two fields - names and phone.
- MongoDB is a **document** database: each record in a MongoDB collection is a **document**.
- Schema is the definition of what fields would be there in one document.
- A document needs a schema to define in mongoose. Mongoose then populates the model or the collection using the schema.
- When we are storing in an array of different documents we need to pre-define the fields that we are using.
- We need to define the schema to define the data format.
- We have to create a new folder **{ models }**.
- Inside models, we need to create a file **{ contact.js}**.
- We need to require mongoose inside the **{ contact.js}** file.
- We need different fields inside the schema and all of them should be required.
- We can give multiple validations inside every field.

{ contact.js}

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

const contactSchema = new mongoose.Schema({

    name: {
        type: String,
        required: true
    },
    phone: {
        type: String,
        required: true
    }

})


const Contact = mongoose.model('Contact', contactSchema);

module.exports = Contact;
```

- We need to require the schema inside the **{ index.js }** file.

{ index.js }

```
const db = require('./config/mongoose');

const Contact = require('./models/contact');
```

Populating the DB

- We need to create an instance of contacts inside the **{ index.js }** file.
- We have created a create function that takes two parameters as name and phone because we have defined them in the schema inside the **{ contact.js}** file.

- We need to attach the callback function to check if there is an error or not.

{ index.js }

```
app.post('/create-contact', function(req, res){

    Contact.create({
        name: req.body.name,
        phone: req.body.name
    }, function(err, newContact){
        if(err){console.log('Error in creating a contact!')
            return;}
            console.log('******', newContact);
            return res.redirect('back');
    })

});
```

Fetching Data from DB

- We have submitted the data using the form and it gets entered into the database as a document.
- We need to render the data that we have submitted on the home page { home. ejs }.
- We need to fetch all the contact in the database and store into the variable, then pass it onto the template or the view.
- After passing the data we can access it using a for a loop.
- __v in MongoDB - Whenever we change the schema of a document we can store it in the versions. Whenever we change the schema we can change the version.

{ index.js }

```
app.get('/', function(req, res){


    Contact.find({}, function(err, contacts){
        if(err){
            console.log("error in fetching contacts from db");
            return;
        }
        return res.render('home',{
            title: "Contact List",
            contact_list: contacts
        });

    })

})
app.post('/create-contact', function(req, res){

    Contact.create({
        name: req.body.name,
        phone: req.body.phone
    }, function(err, newContact){
        if(err){console.log('Error in creating a contact!')
            return;}
            console.log('******', newContact);
            return res.redirect('back');
    })


});
```

Deleting from DB

- The object id is unique to every document that is created in the database.
- We will be utilizing id to find the contact to be deleted from the contact list.

{ index.js }

```
app.get('/delete-contact/', function(req, res){
    console.log(req.query);
    let id = req.query.id

    Contact.findOneAndDelete(id, function(err){
        if(err){
            console.log('error in deleting the object');
            return;
        }
        return res.redirect('back');
    })

});
```

Summarizing

- We used express to set up the server.
- We got the concept of rendering some data or sending it back to the browser using routes and controllers.
- We use a template engine - EJS to make the website dynamic.
- We use the dynamic data to make a basic contact list app and make a post request to create another contact in the list. { HTTP Request - Get and Post}
- We have used persistent storage and set up MongoDB, Mongoose, Robo3T.
- We have used different functions of the MongoDB using mongoose to add a contact, to iterate over a contact, to fetch the list of contacts, and to delete a contact.