Node.js+Express and MongoDB+Mongoose

2020/21 COMP3322 Modern Technologies on WWW

Content

- Introduction to MongoDB
- Introduction to Mongoose.js

Express Database Integration

- Express can make use of or connect to a number of database systems.
  - Cassandra, Couchbase, CouchDB, LevelDB, **MySQL**, **MongoDB**, Neo4j, Oracle, PostgreSQL, Redis, SQL Server, SQLite, & ElasticSearch
- To connect to database, we need to install the corresponding Node.js driver to your project and add it to your app.
- MongoBD is one of the commonly used database systems around. We are going to explore the basic features of MongoDB and how it is being used in the Express app.

Introduction to MongoDB

- MongoDB is an example of **NoSQL**.
- NoSQL database stands for "Not Only SQL" or "Not SQL.
  - Traditional RDBMS uses SQL syntax to store and retrieve data. However, the system response time becomes slow when connects to massive volumes of data.
  - **NoSQL database is non-relational**, so it scales out better than relational databases as they are designed with web applications in mind.
- MongoDB is a document database.
  - There are several types of NoSQL databases.

Introduction to MongoDB

- Each Mongo database contains **collections** of **documents**.
  - A **collection** is like a table in RDBMS.
  - A **document** is like a record in RDBMS.
- A document has zero or more fields. **Fields** are analogous to columns in relational databases.
  - Each field is a **name-value pair** in a document.
  - The **number of fields** in each document can be different from each other even they are under the same collection.
  - The document structure is more like the JSON object of JavaScript.
  - It provides more flexibility since all records are **not restricted by the same column names and types** defined across the entire table.

Install MongoDB

- To install MongoDB on your platform:
  - https://www.mongodb.com/try/download/community
  - Read the installation guide
    - https://docs.mongodb.com/guides/server/install/

- The **mongo shell** is an interactive JavaScript interface to MongoDB.
  - We can use the mongo shell to query and update data as well as perform administrative operations.

Create and Insert Database (Mongo Shell)

- The "use" command is used to create a database in MongoDB. **If the database does not exist a new one will be created**.

```
> use studentDB
switched to db studentDB
>
```

- **Adding documents** using insert() command.
  - Within the "insert" command, add the required Field Name and Field Value for the document which needs to be created.
  - The "insert" command can also be used to **insert multiple documents** into a collection at one time.
  - It returns an object that contains the status of the operation.
    - A WriteResult object for single inserts.
    - A BulkWriteResult object for bulk inserts.

https://docs.mongodb.com/manual/reference/method/db.collection.insert/

```
> db.srecords.insert([
...   {
...     "name": "Tony Stark",
...     "number": "3015111111",
...     "age": 27,
...     "email": "tonystark@hku.hk"
...   },
...   {
...     "name": "Peter Parker",
...     "number": "3015222222",
...     "age": 24,
...     "email": "peterparker@hku.hk"
...   },
...   {
...     "name": "Bruce Banner",
...     "number": "3015333333",
...     "age": 21,
...     "email": "brucebanner@hku.hk"
...   },
... ])
```

```
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 3,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
> show dbs
admin      0.000GB
config     0.000GB
local      0.000GB
studentDB  0.000GB
>
```

If your database is empty, it won't be showed by 'show dbs'

Add Collections

- The easiest way to create a collection is to insert a record (which is nothing but a document consisting of Field names and Values) into a collection. **If the collection does not exist a new one will be created.**

```
> db.courses.insert(
... {
...   "code": "COMP3322",
...   "title": "Modern Technologies in WWW"
... }
... )
WriteResult({ "nInserted" : 1 })
> show collections
courses
srecords
> db
studentDB
```

MongoDB ObjectID

- By default when inserting documents in the collection, if you don't add a field name with the '_id' in the field name, then MongoDB will **automatically add** an Object id field.
- This becomes the **primary key** of the document which is unique within the collection.
- If you want to specify your own id as the _id of the collection, then you need to explicitly define this while creating the collection.

```
db.courses.insert(
  {
    "_id": 3322,
    "code": "COMP3322",
    "title": "Modern Technologies in WWW"
  }
)
```

https://docs.mongodb.com/manual/reference/method/db.collection.find/

Performing Queries

- MongoDB provides a function called find () which is used for retrieval of documents from a MongoDB database.
- To **get all documents** in the collection:

```
> db.srecords.find()
{ "_id" : ObjectId("5be7e0030409f5c94b87c89e"), "name" : "Tony Stark",
"number" : "3015111111", "age" : 27, "email" : "tonystark@hku.hk" }
{ "_id" : ObjectId("5be7e0030409f5c94b87c89f"), "name" : "Peter Parker",
"number" : "3015222222", "age" : 24, "email" : "peterparker@hku.hk" }
{ "_id" : ObjectId("5be7e0030409f5c94b87c8a0"), "name" : "Bruce Banner",
"number" : "3015333333", "age" : 21, "email" : "brucebanner@hku.hk" }
```

https://docs.mongodb.com/manual/reference/operator/query/

Perform Queries

- We can use **criteria's or conditions** to retrieve specific data from the database.
- To find a student whose name is "Peter Parker" in the collection:

```
> db.srecords.find({name: "Peter Parker"})
{ "_id" : ObjectId("5be7e0030409f5c94b87c89f"), "name" : "Peter Parker", "number"
: "3015222222", "age" : 24, "email" : "peterparker@hku.hk" }
```

- To find the students whose age is greater than 22:

```
> db.srecords.find({age: {$gt:21}})
{ "_id" : ObjectId("5be7e0030409f5c94b87c89e"), "name" : "Tony Stark", "number"
: "3015111111", "age" : 27, "email" : "tonystark@hku.hk" }
{ "_id" : ObjectId("5be7e0030409f5c94b87c89f"), "name" : "Peter Parker",
"number" : "3015222222", "age" : 24, "email" : "peterparker@hku.hk" }
```

Perform Queries

- We can specify an optional 2nd argument to determines which fields are returned in the matching documents.
  - To find the students whose age is greater than 22 and only return the name and number fields:

```
> db.srecords.find({age: {$gt:21}}, {name: 1, number: 1})
{ "_id" : ObjectId("5e9e5eb99f75aad39c545708"), "name" : "Tony Stark", "number"
: "3015111111" }
{ "_id" : ObjectId("5e9e5eb99f75aad39c545709"), "name" : "Peter Parker",
"number" : "3015222222" }
```

- The 2nd parameter takes a document of the following form:
  - { field1: <value>, field2: <value> ... }
  - The <value> can be any of the following:
    - 1 for including the field in the return documents; 0 for excluding the field; can be an expression using a Projection Operators.

https://docs.mongodb.com/manual/reference/method/js-cursor/

Perform Queries

- To get the number of documents in a collection, user count()

```
> db.srecords.count()
4
```

- To limit the number of returned documents, use limit()

```
> db.srecords.find().limit(2)
{ "_id" : ObjectId("5be7e0030409f5c94b87c89e"), "name" : "Tony Stark", "number" :
"3015111111", "age" : 27, "email" : "tonystark@hku.hk" }
{ "_id" : ObjectId("5be7e0030409f5c94b87c89f"), "name" : "Peter Parker", "number" :
"3015222222", "age" : 24, "email" : "peterparker@hku.hk" }
```

- To order the returned documents in ascending (1) or descending (-1) order of a field, use sort()

```
> db.srecords.find().sort({number: 1}).limit(2)
{ "_id" : ObjectId("5be7e9890409f5c94b87c8a1"), "name" : "James Bond", "number" :
"3015007007", "age" : 26, "email" : "jamesbond@hku.hk" }
{ "_id" : ObjectId("5be7e0030409f5c94b87c89e"), "name" : "Tony Stark", "number" :
"3015111111", "age" : 27, "email" : "tonystark@hku.hk" }
```

Update Documents

- The update() method updates the values in the existing document.

```
> db.courses.find()
{ "_id" : ObjectId("5be7ea1a0409f5c94b87c8a2"), "code" : "COMP3322", "title" :
"Modern Technologies in WWW" }
> db.courses.update({code: "COMP3322"}, {$set: {code: "COMP3322A"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.courses.find()
{ "_id" : ObjectId("5be7ea1a0409f5c94b87c8a2"), "code" : "COMP3322A", "title" :
"Modern Technologies in WWW" }
```

https://docs.mongodb.com/manual/reference/method/db.collection.update/

https://docs.mongodb.com/manual/reference/operator/update/

Update Documents

- The save() method replaces the existing document with the document passed in save() method.

```
> db.courses.save(
... { "_id" : ObjectId("5be7ea1a0409f5c94b87c8a2"),
... "code": "COMP3230A", "title": "Principles of Operating Systems"})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.courses.find()
{ "_id" : ObjectId("5be7ea1a0409f5c94b87c8a2"), "code" : "COMP3230A", "title" :
"Principles of Operating Systems" }
```

https://docs.mongodb.com/manual/reference/method/db.collection.remove/

Delete Documents

- The remove() method is used to remove documents from a collection.
- Either all of the documents can be removed from a collection or only those which matches a specific condition.

```
> db.courses.count()
3
> db.courses.remove({"code": "COMP3322A"})
WriteResult({ "nRemoved" : 1 })
> db.courses.count()
2
> db.courses.remove({})
WriteResult({ "nRemoved" : 2 })
> db.courses.count()
0
```

Rename and Delete Collections

- To rename a collection, use .renameCollection() method

```
> db.srecords.renameCollection(“stdrecords")
```

- To delete a collection from the database, use .drop() method

```
> db.srecords.drop()
```

Mongoose

- Mongoose is an **object document mapper** (ODM) library on Node.
  - It is an object associated with NoSQL database. As the name suggests it **maps documents** in a MongoDB database **to objects** in the program.
- Mongoose provides a straight-forward, schema-based solution to model your application data.
- It provides a simple validation and query API to interact with the MongoDB database and it makes development fast.

Using Mongoose

- Here are the actions to use MongoDB in your Express code using Mongoose:
  - Include Mongoose in your express app

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

  - Connect to your MongoDB server to access the target database

```
mongoose.connect('mongodb://localhost/studentDB');
```

DeprecationWarning: current URL string parser is deprecated, and will be removed in a future version. To use the new parser, pass option { useNewUrlParser: true } to MongoClient.connect.

```
mongoose.connect('mongodb://localhost:27017/studentDB', {useNewUrlParser: true});
```

https://mongoosejs.com/docs/connections.html

  - Define a schema for your object. Then use the schema to create a model that links to a collection defined in the connected MongoDB.
  - With the model, we can make use of the Mongoose API to search for specific data, add new data, update the data and delete the data related to this model.

Mongoose Schemas

- Mongoose uses schemas to model the data an application wishes to store and manipulate in MongoDb.
- Mongoose supports various data types in the schema:
  - String
  - Number
  - Date
  - Buffer – allows us to save binary data, e.g, image file
  - Boolean
  - Mixed – is an “anything goes” type, which means no defined structure
  - ObjectId – a type for storing the object id that links to another document
  - Array – a data type allows us to store JavaScript-like arrays

https://mongoosejs.com/docs/schematypes.html

Define a Mongoose Schema

```
var Schema = mongoose.Schema;

var studentSchema = new Schema({
  name: String,
  number: String,
  age: Number,
  email: String,
  courses: [ {code: String, title: String} ]
});
```

Mongoose provides a way to validate data before you save the data to DB.
We can design our own validation functions or just catch the error threw by the built-in validation.

```
var studentSchema = new Schema({
  name: String,
  number: {type: String, minlength: 10, maxlength: 10},
  age: { type: Number, min: 17, max: 28 },
  email: String,
  courses: [ {code: String, title: String} ]
});
```

https://mongoosejs.com/docs/models.html

Create a Mongoose Model from the Schema

- Based on the defined schema, we **register a model** with Mongoose so that we can use it throughout our application.
- An instance of a model is called a document. Models are responsible for creating and reading documents from the underlying MongoDB database.

```
var srecord = mongoose.model("srecord", studentSchema);
```

Returns a mongoose model object

The first argument is the singular name of the collection in the database, i.e, **the system is expecting** there is a collection with the name "srecord**s**"

```
var srecord = mongoose.model("record", studentSchema, "srecords");
```

https://mongoosejs.com/docs/api/document.html#document_Document-save

Saving Documents - (Create)

- To save a document to the connected MongoDB:
  - Create an instance of the model object
  - Assign contents to the instance
  - Call the model.save() with the callback function

```
var newRecord = new srecord({
   name: "Harry Potter",
   number: "3015987321",
   age: 17,
   email: "harrypotter@hku.hk"
 });
 newRecord.save((err, result) => {
   if (err) {
     console.log("Database error: "+err);
   } else {
     console.log("Record added");
   }
 });
```

https://mongoosejs.com/docs/queries.html

Searching Data - (Retrieve)

- Mongoose provides several different functions to find data from a collection. The methods are:
  - .find() – find documents match the given condition(s)

```
//return all documents from the collection
srecord.find((err, result) => { })
//return document(s) that match - number: 3015222222
srecord.find({number: "3015222222"}, (err, result) => { })
//return document(s) that has age greater than or equal to 20
srecord.find({age: {$gte: 20}}, (err, result) => { })
```

Data is returned as the 2nd argument of the callback function, which is an array of documents

    - We can ask for returning specific fields only

```
//return all documents and only the "name" and "number" fields
srecord.find({}, 'name number', (err, result) => { })
```

  - .findOne() - returns the **first document** that matches the given condition(s), otherwise returns null
  - .findById() - find **a single document** by its "_id" field.

https://mongoosejs.com/docs/api.html#query_Query-sort

Searching Data

- The find function call also be chained to other query methods, such as where, and, or, limit, sort, any, etc.

```
srecord.find().sort({number: -1}).limit(3).exec((err, result) => { })
```

-1: descending
1: ascending

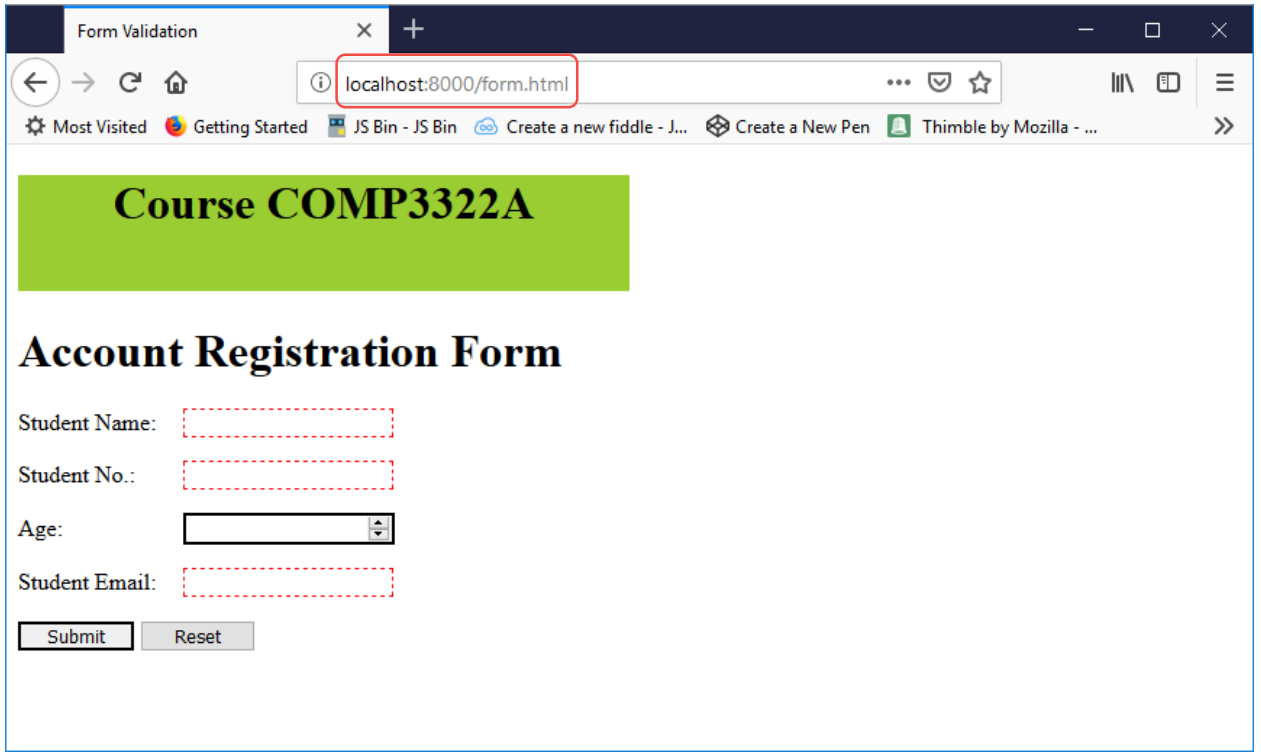
limit the no. of return

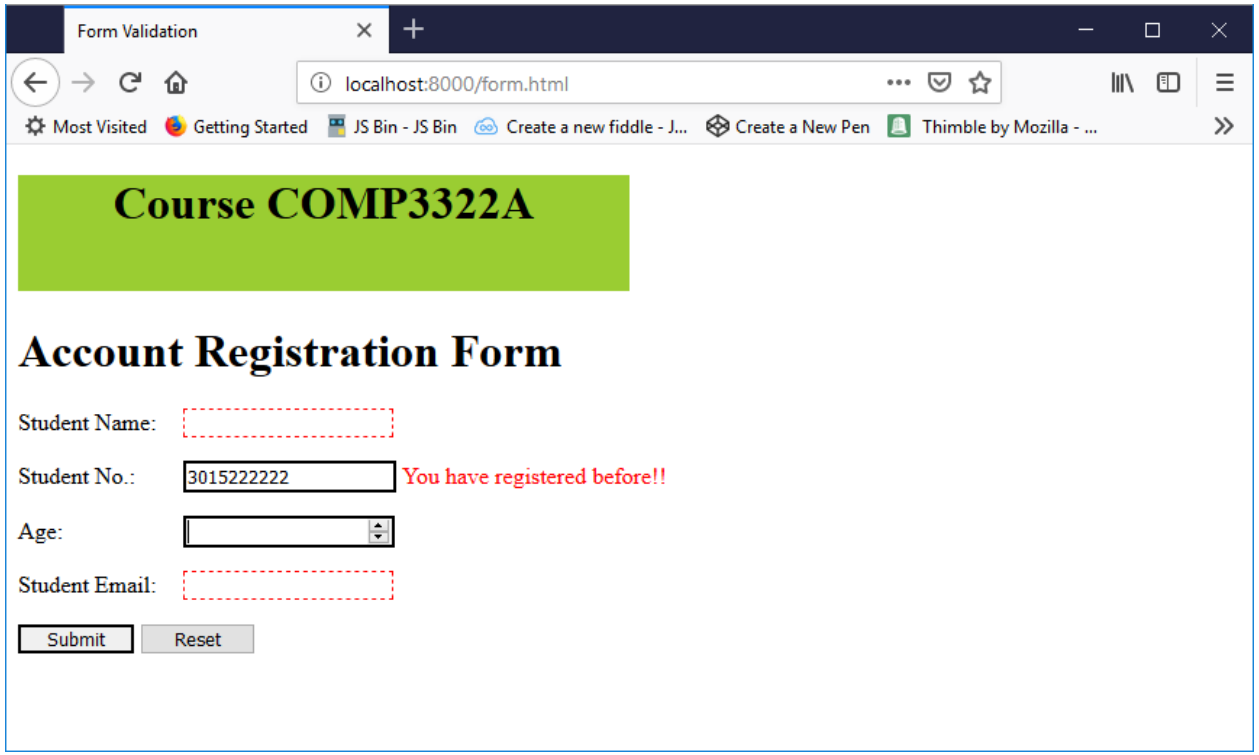
execute the query

Update and Delete

- Mongoose provides several methods to find-update and find-delete
  - .find**ById**AndDelete() - Finds a matching document, removes it, passing the found document (if any) to the callback.
  - .find**ById**AndUpdate() - Finds a matching document, updates it according to the update arg, and returns the found document (if any) to the callback.
  - .find**One**AndDelete()
  - .find**One**AndUpdate()
  - .deleteMany() - Deletes all of the documents that match conditions from the collection.
  - .deleteOne() - Deletes the first document that matches conditions from the collection.
  - .replaceOne() - replace the entire document
  - .updateMany()
  - .updateOne()

Demo – Our Account Registration Form





Form Validation

localhost:8000/form.html

Most Visited | Getting Started | JS Bin - JS Bin | Create a new fiddle - J... | Create a New Pen | Thimble by Mozilla - ...

Course COMP3322A

Account Registration Form

Student Name: Harry Potter

Student No.: 3015888888

Age: 18

Student Email: harrypotter@hku.hk

Submit Reset

Account List

localhost:8000/acclist.html

Most Visited | Getting Started | JS Bin - JS Bin | Create a new fiddle - J... | Create a New Pen | Thimble by Mozilla - ...

Course COMP3322A

back

Current Account List

Name : James Bond
Number : 3015007007
Age : 26
Email : jamesbond@hku.hk

Name : Tony Stark
Number : 3015111111
Age : 27
Email : tonystark@hku.hk

Name : Peter Parker
Number : 3015222222
Age : 24
Email : peterparker@hku.hk

Name : Bruce Banner
Number : 3015333333
Age : 21
Email : brucebanner@hku.hk

Name : Harry Potter
Number : 3015888888
Age : 18
Email : harrypotter@hku.hk

DBPug

Name

- public
  - style
    - style.css
  - js
    - jquery-3.3.1.js
- data
- views
  - form.pug
  - acclist.pug
  - use.pug
  - header.pug
- node_modules
- index.js
- package-lock.json
- package.json

style.css

```
label {
  display: inline-block;
  width: 110px;
}
.bttn{
  display: inline-block;
  width: 80px;
}
input:invalid {
  border: 1px dashed red;
}

input:valid {
  border: 2px solid black;
}

#chkReg {
  color: red;
}

#header {
  background-color: yellowgreen;
  width: 50%;
  height: 5rem;
  position: relative;
}

#header h1 {
  text-align: center;
}

#header a {
  position: absolute;
  right: 2px;
}
```

header.pug

```
div#header
  h1 Course COMP3322A
  block link
```

use.pug

```
extends header.pug

block link
  a(href='/form.html') back
```

acclist.pug

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Account List</title>
  <script src='/js/jquery-3.3.1.js'></script>
  <link rel="stylesheet" type="text/css" href="/style/style.css">
</head>
body
    include use.pug
    h1 Current Account List
    each record in data
      p
        = "Name : "+record.name
        br
        = "Number : "+record.number
        br
        = "Age : "+record.age
        br
        = "Email : "+record.email
```

form.pug

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Form Validation</title>
  <script src='/js/jquery-3.3.1.js'></script>
  <link rel="stylesheet" type="text/css" href="/style/style.css">
</head>
<body>
  include header.pug
  <h1>Account Registration Form</h1>
  <form id="RegForm" action="acclist.html" method="post">
    <p>
      <label for="name">Student Name:</label>
      <input type="text" id="name" name="name" maxlength="50" required>
    </p>
    <p>
      <label for="number">Student No.:</label>
      <input type="text" id="number" name="number" maxlength="10" pattern="3015[0-
9]{6}" required>
      <span id="chkReg"></span>
    </p>
    <p>
      <label for="age">Age:</label>
      <input type="number" id="age" name="age" min="17" max="30" step="1"
pattern="[0-9]+">
    </p>
    <p>
      <label for="email">Student Email:</label>
      <input type="email" id="email" name="email" required>
    </p>
    <input class="bttn" type="submit" value="Submit">
    <input class="bttn" type="reset">

  </form>
```

```
script.
    $("#email").on("input", function () {
      if (this.validity.typeMismatch) {
        this.setCustomValidity("Enter a valid email address");
      } else {
        this.setCustomValidity("");
      }
    });

    $("#number").on("input", function () {
      if (this.validity.patternMismatch) {
        this.setCustomValidity("Must be 10 digits starts with 3015");
      } else {
        this.setCustomValidity("");
      }
    });

    $("#number").on("blur", function () {
      $.get("check",
        {"number": this.value},
        function (data, status) {
          if (status == "success") {
            $("#chkReg").html(data);
          }
        }
      );
    });

</body>
</html>
```

index.js

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

app.use(express.static("public"));
app.set("view engine", "pug");
app.set("views", "views");

app.use(express.urlencoded({extended: false}));

//connect to MongoDB
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/studentDB', (err) => {
  if (err)
    console.log("MongoDB connection error: "+err);
  else
    console.log("Connected to MongoDB");
});

//Set the Schema
var mySchema = new mongoose.Schema({
  name: String,
  number: String,
  age: Number,
  email: String
});
//Create my model
var srecord = mongoose.model("srecord", mySchema);

//handle the route GET /form.html
app.get('/form.html', (req, res) => {
```

```
//handle the route GET /form.html
app.get('/form.html', (req, res) => {
  res.render('form');
});

//handle the route GET /check, which is initiated by AJAX
app.get('/check', (req, res) => {
  if (req.query.number) {
    let num = req.query.number;
    console.log("Check existence of record: "+num);
    srecord.find({number: num}, (err, result) => {
      if (err) {
        console.log("Query error: "+err);
        res.end();
      } else {
        if (result.length > 0) {
          res.send("You have registered before!!");
        } else
          res.end();
      }
    });
  } else
    res.end();
});

//handle the route POST /acclist.html
app.post('/acclist.html', (req, res) => {
  let newRecord = new srecord({
    name: req.body.name,
    number: req.body.number,
    age: req.body.age,
```

```
//handle the route POST /acclist.html
app.post('/acclist.html', (req, res) => {
  let newRecord = new srecord({
    name: req.body.name,
    number: req.body.number,
    age: req.body.age,
    email: req.body.email
  });
  newRecord.save((err, result) => {
    if (err) {
      console.log("Database error: "+err);
      res.sendStatus(500);
    } else {
      console.log("Record added");
      //retrieve all records
      srecord.find().sort({number: 1}).exec((err, result) => {
        if (err) {
          console.log("Database error: "+err);
          res.sendStatus(500);
        } else {
          res.render('acclist', {data: result});
        }
      });
    }
  });
});

app.listen(8000, () => {
  console.log('Example app listening on port 8000!')
});
```

Readings

- An Introduction to Mongoose for MongoDB and Node.js
  - https://code.tutsplus.com/articles/an-introduction-to-mongoose-for-mongodb-and-nodejs--cms-29527

References

- MongoDB
  - https://docs.mongodb.com/manual/
- Mongoose
  - https://mongoosejs.com/
- Introduction to Mongoose for MongoDB
  - https://medium.freecodecamp.org/introduction-to-mongoose-for-mongodb-d2a7aa593c57
- MDN - Express Tutorial Part 3: Using a Database (with Mongoose)
  - https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/mongoose