# CSSE3101 – ADVANCED WEB TECHNOLOGIES

# MERN STACK DEVELOPMENT

# Objective:

In this lab activity, you are going to learn how to set up an MERN stack React application. You will learn to create a Mongodb database and collection inside the database and make the user interface(UI) to collect the data to insert into the collection in the database.

## Requirements implemented in this STUDENT INFORMATION APPLICATION:

PART 1 – APPLICATION SETUP

PART 2 – FRONT- END DEVELOPMENT USING REACT AND BOOTSTRAP

PART 3 – CREATING THE DATABASE USING MONGODB

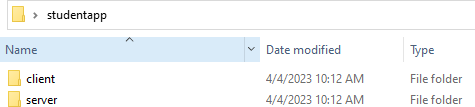
PART 4 – SERVER-SIDE DEVELOPMENT USING NodeJS and Express

PART 5 – CONNECTING YOUR APPLICATION TO MONGODB

PART 6 – CREATING MODELS

PART 7 – CREATING EXPRESS ROUTES (CRUD Operations - CREATE/Post – add student record to the database

## PART 1 – APPLICATION SETUP

1. Create the application folder structure.
2. Open the **studentapp** folder in Visual Studio (VS) Code. Open a new terminal and do the following:
   1. Inside the folder, create a folder named server.
   2. Inside the folder, create a react app using the command:

## npx create-react-app client

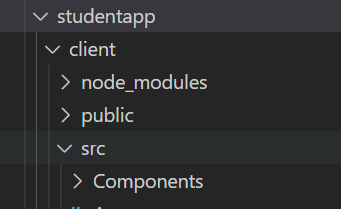
* 1. Install the required dependencies for this application using the command **npm install**. Other dependencies will be added later.

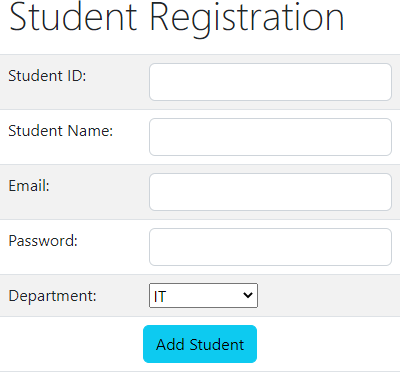
## npm install bootstrap

**npm install react-router-dom**

* 1. Start your client server using the command **npm start.**

## PART 2 – FRONT END DEVELOPMENT USING REACT AND BOOTSTRAP

1. Create a new folder named **Components** in the src folder.
2. In the Components folder, create a new file: **StudentRegister.js.** This component will allow the user to add or register a new student and will be saved to the database.
3. Create the student registration form as shown below. Use Bootstrap classes and define your own CSS to achieve the desired UI. Use Bootstrap table to create the form layout.



1. Update your **App.js, so that, it will render the StudentRegister.**
2. Import the following file in src/index.js

**import 'bootstrap/dist/css/bootstrap.css'**

## PART 3 – CREATING THE DATABASE USING MONGODB

Project Summary:

|  |  |
| --- | --- |
| **Project Name** | [**STUDENT\_INFORMATION\_SYSTEM\_APP**](https://cloud.mongodb.com/v2/6389b695e1951f1a929ba11f) |
| **Cluster Name** | studentinfosys |
| **Database Name** | studentdb |
| **Collections** | students |
| **Database username** | admin |
| **Database password** | csse3101 |

## PART 4 – SERVER-SIDE DEVELOPMENT USING NodeJS and Express

1. Go to the terminal, and in the server folder do the following:
   1. Execute the command to initialize the server. Accept the default server setting by

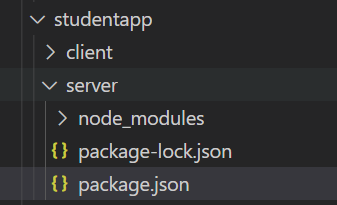
Pressing the Enter key.

## A picture containing text Description automatically generatednpm init

* 1. Install the libraries.

## npm install express

## npm install mongoose

After executing the commands your server folder will have the following files and folders:

1. In the server folder, create a file **index.js.** This file is the entry point for your server. Do the following:
   1. Write the code to create the express server.

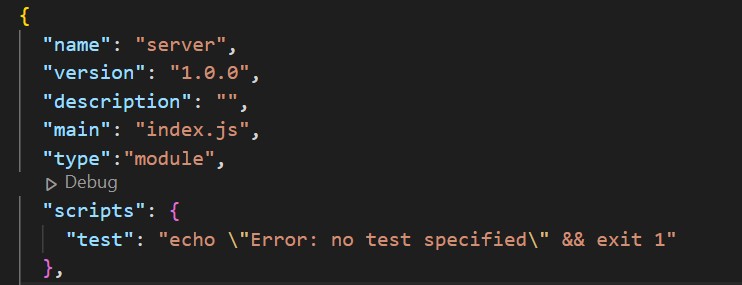
import express from "express";

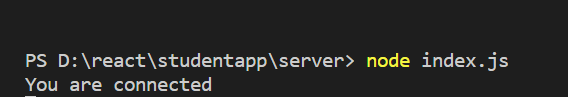
const app = express(); app.listen(3001, () => {

console.log("You are connected");

});

* 1. Edit **package.json** and add the following after the main: **"type":"module",**

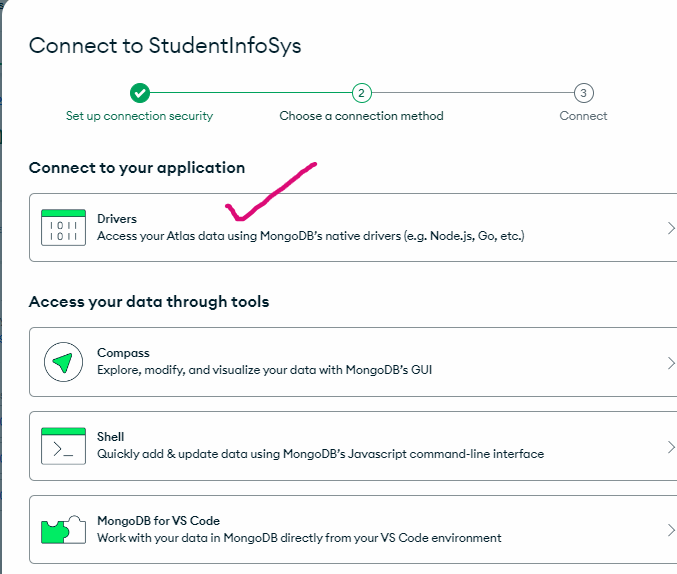


1. From the terminal, run your server.

Note: Every time you have changes in your index.js, you need to run again the server.

## PART 5 – CONNECTING YOUR APPLICATION TO MONGODB

1. Login to your MongoDb Atlas account. Go to the specific project. Select Connect Your Application.



## Copy the connection string.

## 

1. In **index.js**, do the following:
   1. Import mongoose library.
   2. Create a connectionString variable and assign the connection string generated from MongoDB Atlas.
   3. Here is an example of a connection string (specify values for password and name of the database):

## mongodb+srv://admin:<password>@studentinfosysapp.rr1cedt.mongodb.net/

**<name of the database>?retryWrites=true&w=majority**

* 1. Complete the connection code as shown below:

import express from "express"; import mongoose from "mongoose";

const app = express();

//Database connection const connectString =

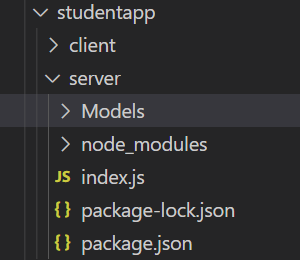
["mongodb+srv://admin:admin12345@studentinfosysapp.5pqj5zu.mo](mailto:admin12345@studentinfosysapp.5pqj5zu.mo) ngodb.net/studentDb?retryWrites=true&w=majority";

mongoose.connect(connectString);

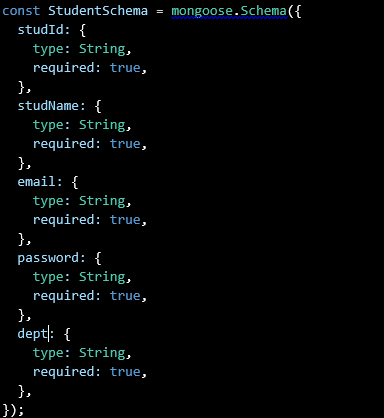
app.listen(3001, () => { console.log("You are connected");

});

## PART 6 – CREATING MODELS

1. In the server folder, create the folder **Models**. This is where the database model is saved.
2. Create a model that will save the student information. Create a new file: **Student.js** inside the Models folder and do the following:
   1. Import mongoose.
   2. Create a schema.



* 1. Add the fields in the schema.
  2. Create a model variable and bind the MongoDb collection with the schema.

**Syntax**:

mongoose.model(<Collectionname>, <CollectionSchema>)

* 1. Export the model.

## PART 7– CREATING EXPRESS ROUTES

1. In the client folder, do the following:
   1. Go to the terminal and install axios.

**npm install axios**



* 1. Update the **StudentRegister.js** and do the following:
     1. Import the following:
     2. Create state variables using useState hook for each of the field in the form.
     3. Update each state variable to be assigned with the value of the respective form elements by adding an event handler for each of the form controls.
     4. Create a function **addStudent** using arrow function syntax. This function uses Axios to submit a request to the server passing along the data to the server.

Syntax:

const nameofmethod = ()=>{

Axios.post(“Server route that handles the request”,{

JSON format data as part of the request

body

})

.then(()=>{

Action if successful

})

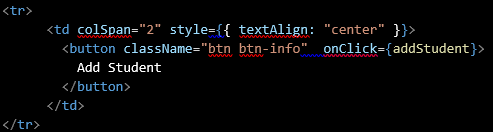
.catch(()=>{

Action if error is encountered

})

}

* + 1. Create the event handler for the button to call the **addStudent** function.



1. Go to the terminal, and in the server install cors.

**npm install cors**

1. In the server folder, update **index.js** by doing the following.
   1. Import cors.
   2. Import the needed model.
   3. Mount the needed middleware functions.
      1. Parse the incoming requests with JSON payloads
      2. Enable Cross-Origin Resource Sharing (CORS)
   4. Create the Express POST route for adding new student.

**BACK-END (server folder)**

**index.js**

import express from "express"; import mongoose from "mongoose";

import StudentModel from "./Models/Student.js"; import cors from "cors";

//declare an express object const app = express();

//middleware app.use(cors()); app.use(express.json());

//database connection const connectString =

"mongodb+srv://admin:b606110@studentinfocluster.0hmagz7.mongodb.net/studentDb?ret ryWrites=true&w=majority";

mongoose.connect(connectString);

//express POST route for adding new student app.post("/addStudent", async (req, res) => {

const student = new StudentModel({ studId: req.body.studId,

studName: req.body.studName,

email: req.body.email,

password: req.body.password,

department: req.body.department,

});

await student.save();

res.send("Record Successfully Added!");

});

Required Submission.

Once you complete the lab activity, you are required to upload the database model file student.js, server folder and src folder of the client app.