

Creating an API using Post method

FRONT END

Step:1

Create a component with name **create-student** and add the below code

create-student.html

```
<form (ngSubmit)="create()">
  <br>
  <br>
  <div class="form-group">
    <label for="tourname">Id</label>
    <input type="number" name="id" [(ngModel)]="stud.id" class="form-
control" ></div>
  <div class="form-group">

    <label for="tourname">FirstName</label>
    <input type="text" name="firstname" [(ngModel)]="stud.firstname" class="for
m-control" ></div>
    <div class="form-group">
      <label for="tourname">LastName</label>
      <input type="text" name="lastname" [(ngModel)]="stud.lastname" class=
"form-control" ></div>
      <div class="form-group">
        <label for="tourname">Address</label>
        <input type="text" name="address" [(ngModel)]="stud.address" clas
s="form-control" ></div>
        <div class="form-group">
          <label for="tourname">Age</label>
          <input type="number" name="age" [(ngModel)]="stud.age" cl
ass="form-control" ></div>
          <div class="form-group">
            <label for="price">Class</label>
            <input type="text" name="class" [(ngModel)]="stud.class" class="form-
control" ></div>
            <div class="form-group">
```

```

        <label for="description">Section</label>
        <input type="text" name="section" [(ngModel)]="stud.section" class="form-
control" ></div>
        <button type="submit" class="btn btn-success">Save Tour</button>
</form>

```

Step:2

Create a .ts file in create-student component to define a class regarding the fields of a student form

student.ts

```

export class Student {
  id: number;
  firstname: string;
  lastname: string;
  age: number;
  address: string;
  class: string;
  section: string;
}

```

Step:3

Import the student class and service file to create-student.ts file as below and write a promise in create()

create-student.ts

```

import { Component, OnInit } from '@angular/core';
import { Student } from './student';
import { ApiService } from '../api.service';
@Component({
  selector: 'app-create-student',
  templateUrl: './create-student.component.html',
  styleUrls: ['./create-student.component.css']
})
export class CreateStudentComponent implements OnInit {
  stud : Student= {
    id : null,
    firstname: "",

```

```

    lastname: "",
    age:null ,
    address: "",
    class: "",
    section: ""
  };
  constructor(private service: ApiService) { }
  ngOnInit() {
  }
  create() {
    this.service.create(this.stud).subscribe(res => {
      console.log(res)
    })
  }
}

```

Step:4 Command to create api service : **ng g service api**

Import student class, HttpClient, HttpHeaders, HttpParams , Add headers and add an API by using post method as below

api.service.ts

```

import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders, HttpParams } from '@angular/common/http';
import { Student } from '../create-student/student';
// import {Observable} from 'rxjs/Observable';
@Injectable({
  providedIn: 'root'
})
export class ApiService {
  constructor(private _http:HttpClient) { }
  readonly rootUrl = 'http://localhost:3005/';

  create(stud : Student){
    return this._http.post(this.rootUrl+"create/create", stud , {
      observe: 'body'
    });
  }
}

```

Step:5 Update your app.module.ts as below

```
import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';
import { NgbModule } from '@ng-bootstrap/ng-bootstrap';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { CreateStudentComponent } from './create-student/create-
student.component';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
import { HttpClientModule } from '@angular/common/http';
@NgModule({
  declarations: [
    AppComponent,
    CreateStudentComponent
  ],
  imports: [
    BrowserModule,
    NgbModule,
    HttpClientModule,
    HttpClientModule,
    FormsModule,

    ReactiveFormsModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Run these commands in your vs code terminal for rectifying errors:

- ➔ npm i @angular/http
- ➔ npm i @ng-bootstrap/ng-bootstrap
- ➔ npm i ng-modules
- ➔ npm i http-client

Step:6 replace the default code in [app.component.html](#) with below code to display the buttons with router link

```

</button>

    <a [routerLink]="['/create']">
        <p>create</p>
    </a>
</button>
</router-outlet></router-outlet>

```

Step:7 add routing in **app-routing.module.ts** file as below

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { ViewStudentsComponent } from './view-students/view-students.component';
const routes: Routes = [
  { path: 'create', component: CreateStudentComponent },
];

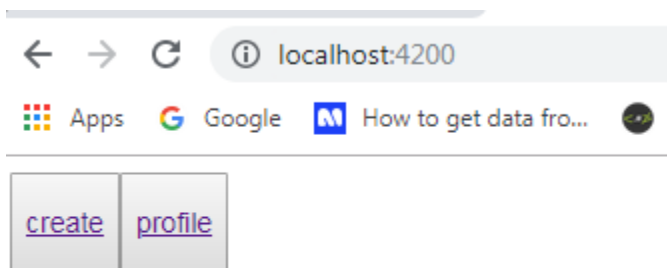
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

```

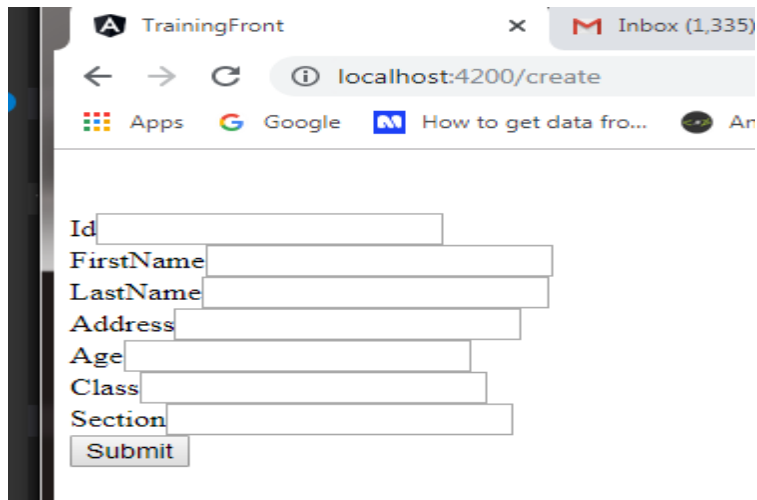
Step:8 For compiling run **ng serve**

The UI part of the above code is

1. This is the default screen we get when we run a project

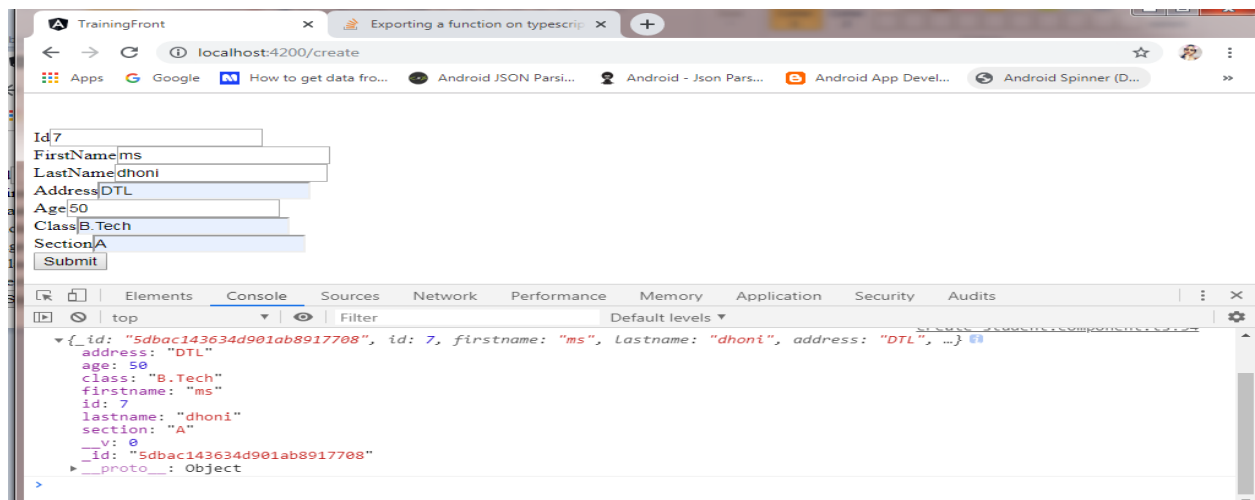


2.while clicking on create button the below screen will displays



A screenshot of a web browser showing a form titled 'TrainingFront' at the URL 'localhost:4200/create'. The form contains input fields for 'Id', 'FirstName', 'LastName', 'Address', 'Age', 'Class', and 'Section', followed by a 'Submit' button. The browser's address bar shows 'localhost:4200/create' and the page title is 'TrainingFront'.

3. Right click on screen and select inspect and go to console there u can see the entered data on a form



A screenshot of the same web browser showing the form after submission. The form fields are now populated with the following values: Id: 7, FirstName: ms, LastName: dhoni, Address: DTL, Age: 50, Class: B.Tech, and Section: A. The 'Submit' button is still visible. The browser's developer console is open, showing the following JSON object:

```
{_id: "5dbac143634d901ab8917708", id: 7, firstname: "ms", lastname: "dhoni", address: "DTL", ...}
```

Building a Simple CRUD Application with Express and Mongo DB

CRUD, Express and Mongo DB

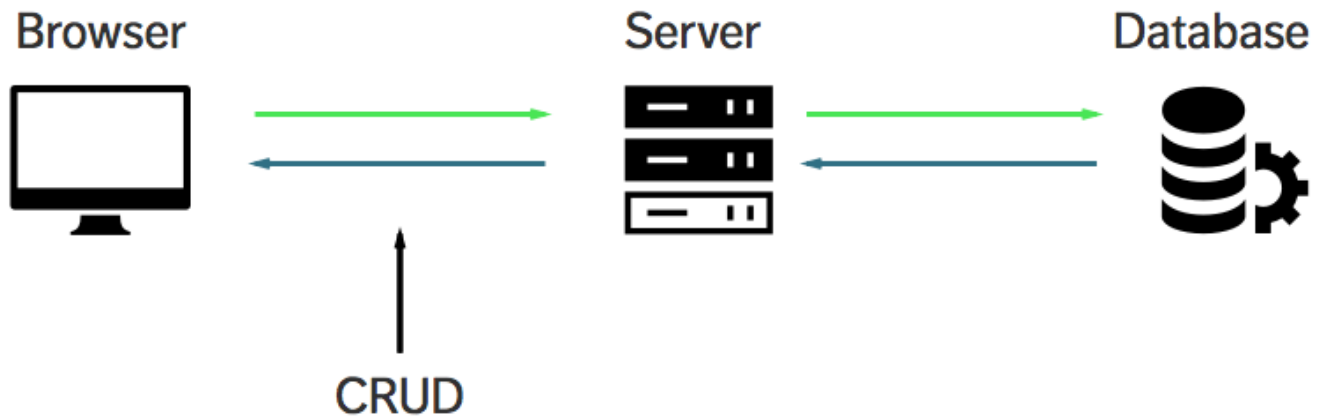
Express is a framework for building web applications on top of Node.js. It simplifies the server creation process that is already available in Node. In case you were wondering, Node allows you to use JavaScript as your server-side language.

Mongo DB is a database. This is the place where you store information for your web websites (or applications).

CRUD is an acronym for Create, Read, Update and Delete. It is a set of operations we get servers to execute (POST, GET, PUT and DELETE respectively). This is what each operation does:

- **Create (POST)** - Make something
- **Read (GET)** - Get something
- **Update (PUT)** - Change something
- **Delete (DELETE)** - Remove something

If we put CRUD, Express and MongoDB together into a single diagram, this is what it would look like:



Prerequisites:

- Mongo DB
- Node JS (version 10.9.0 or later)

To check if you have Node installed, open up your command line and run the following code:

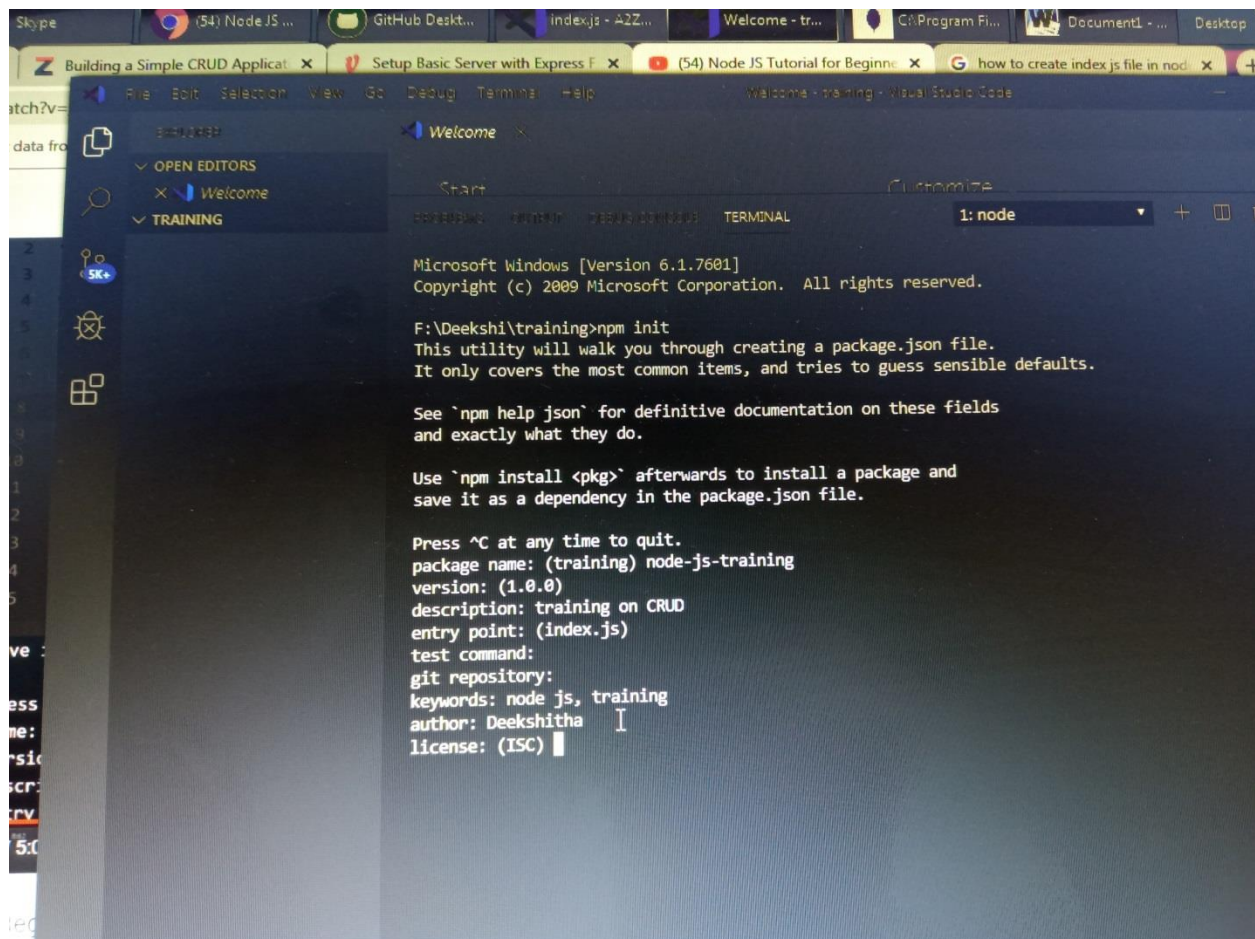
```
$ node -v
```

You should get a version number if you have Node installed. If you don't, you can install Node either by downloading the installer from [Node's website](https://nodejs.org/en/)

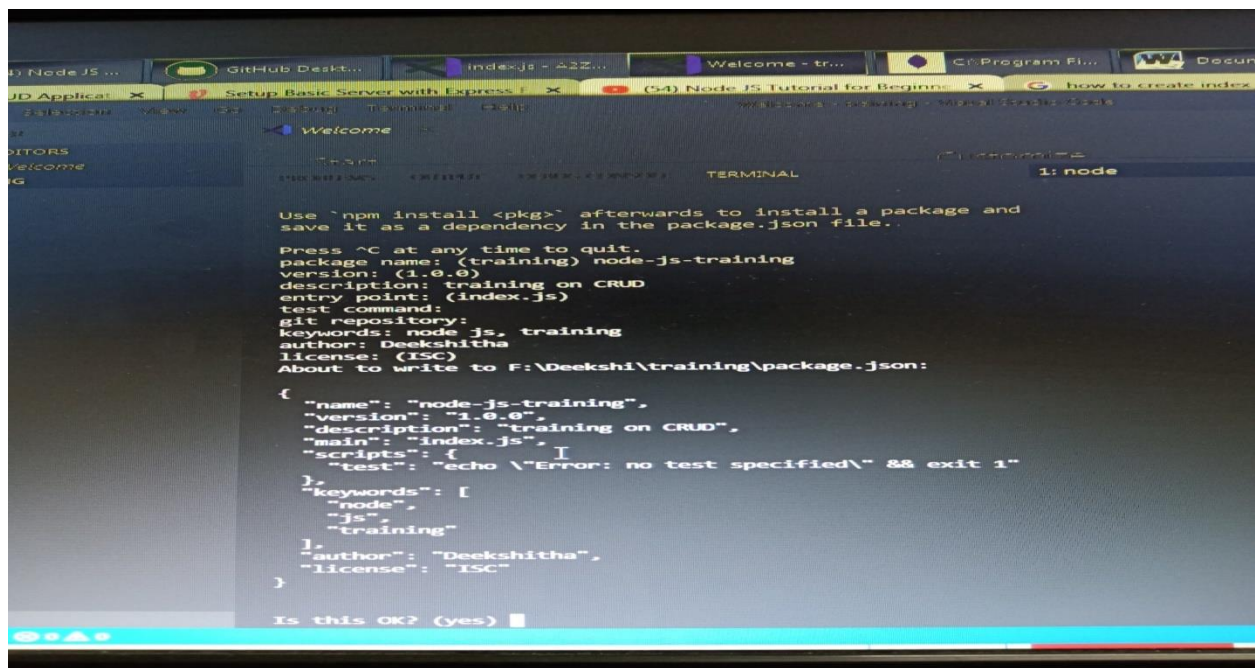
Getting started:

1. Start by creating a folder for this project.
2. Once you navigate into it, run the `npm init` command.

Some of few questions is asked by command prompt like as below:



Press **enter** after giving the above details.



Enter **yes** to get package.json file.

Installing Express :

By the use of below command you install the express.js into your project . This command is run by npm(*node package manager*).

npm install express --save

By giving the above command in terminal it will generate package.lock.json file

Create Server :

Create a file with name index.js / app.js file

1.Import express in your project by require command like below

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

2. Create app level object of Express JS

```
var app = express();
```

3. At last, write the line of code of listening to the server on 3005 port by below code of snippet.

```
app.listen(3005, ()=> console.log("listening on port 3005"));
```

MongoDB

We first have to install MongoDB through npm if we want to use it as our database.

npm install mongoose

Once installed, we can connect to MongoDB through the `mongoose` connect method as shown in the code below:

```
var mongoose = require('mongoose');
mongoose.connect("mongodb://127.0.0.1:27017/training", {useNewUrlParser: true, useUnifiedTopology: true}, (error, client) => {
```

```
if(error){
    console.log("Error occurred not able to connect");
}
});
```

Once we're done with installing express, creating server, connecting mongoose to express

Step 1:

After adding the **create** route the app.js / index.js file will be as follows :

```
var express = require('express');
var createRouter = require('./routes/create');
var bodyParser = require('body-parser')
var path = require('path');

var app = express();
app.use(bodyParser.urlencoded({ extended: false }))
// parse application/json
app.use(bodyParser.json())
// add mongoose
var mongoose = require('mongoose');
mongoose.connect("mongodb://127.0.0.1:27017/training", {useNewUrlParser: true, useUnifiedTopology: true}, (error, client) => {
    if(error){
        console.log("Error occurred not able to connect");
    }
});
//add cors
var cors = require('cors');
app.use(cors({
    origin: 'http://localhost:4200'
}));
app.use('/create', createRouter);
// catch 404 and forward to error handler

app.use(function(req, res, next) {
    next(createError(404));
});

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');
```

```
// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});
//server listening
app.listen(3005, ()=> console.log("listening on port 3005"));
module.exports = app;
```

run **npm i cors** & **npm i ejs** commands in your terminal to install cors

Step 2:

Create a folder with name views and add a files with name **error.ejs** and **index.ejs**

In **error.ejs** file include below code

```
<h1><%= message %></h1>
<h2><%= error.status %></h2>
<pre><%= error.stack %></pre>
```

In **index.ejs** file include below code

```
<!DOCTYPE html>
<html>
  <head>
    <title><%= title %></title>
    <link rel='stylesheet' href='/stylesheets/style.css' />
  </head>
  <body>
    <h1><%= title %></h1>
    <p>Welcome to <%= title %></p>
  </body>
</html>
```

Step 3: Create a Schema with name `C-stud.js` in models folder

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
// var Subscribe = mongoose.Subscribe;

var schema = new Schema({
  id : {type:Number, require:true},
  firstname: {type:String, require:true},
  lastname:{type:String, require:true},
  age:{type:Number, require:true},
  class:{type:String, require:true},
  section:{type:String, require:true},
  address:{type:String, require:true}
  // creation_dt:{type:Date, require:true}
});
module.exports = mongoose.model('create-stud', schema );
```

Step 4: create a route with name `create.js` in Routes folder

```
var express = require('express');
var router = express.Router();
var createStudent = require('../models/C-stud');
router.post('/create', function(req,res,next) {
  var createstudent = new createStudent({
    id: req.body.id,
    firstname: req.body.firstname,
    lastname: req.body.lastname,
    address: req.body.address,
    age: req.body.age,
    class: req.body.class,
    section: req.body.section,
  });

  let promise = createstudent.save();
  promise.then(function(doc) {
    return res.status(201).json(doc);
  })

  promise.catch(function(err){
```

```

    return res.status(501).json({message: 'Error adding tour'})
  })
});
module.exports = router;

```

Step 5: To run the backend use **node index.js** command in terminal

Note: ensure that mongod is started

Then u will get a msg in terminal as **“listening on port 3005”**

To see the stored data mongoDB database:

open Robo3T and choose the desired database

