Diploma in

IT, Networking and Cloud

Elective Module 3 Web Development using JS Frameworks

Lab Manual

Table of Contents

Learning Outcome

After completing this module, the student should be **able to understand how to install Angular Js.**

To meet the learning outcome, a student has to complete the following activities

1. Installation of node js and angular js (1.5 Hrs)
2. Creating a Angular Js Application (3 Hrs)

Activity 1

Aim: Installing Angular Js.

Learning outcome: Able to understand basic how to install angular js.

Duration: 1.5 hour

List of Hardware/Software requirements:

1. Operating system - Windows 10/11 or Linux
2. Command prompt
3. Internet connectivity

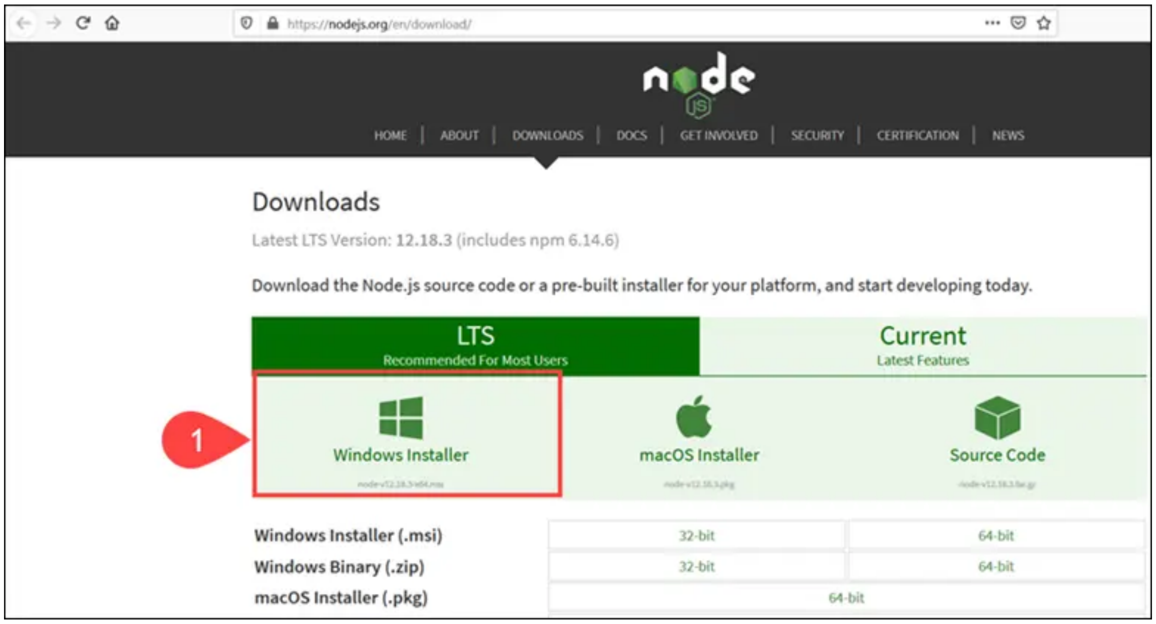
Code/Program/Procedure (with comments):

Step 1: Install Node.js

Angular bases its build environments on Node.js, and many of its features depend on NPM packages. Conveniently, the Node Package Manager (NPM) client is part of the default Node.js package.

To install Node.js:

1. Visit the official Node.js page and download the latest Node.js **Windows Installer**.


https://nodejs.org/en/download/
HOME | ABOUT | DOWNLOADS | DOCS | GET INVOLVED | SECURITY | CERTIFICATION | NEWS
LTS Recommended For Most Users
Current Latest Features
1
Windows Installer
macOS Installer
Source Code

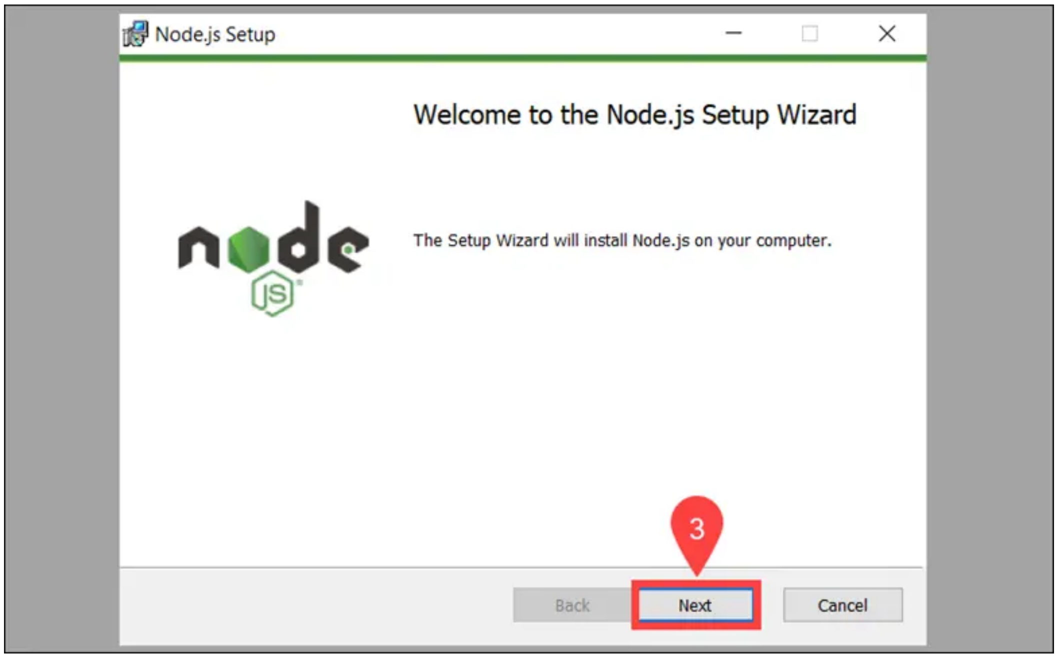
Downloads

Latest LTS Version: 12.18.3 (includes npm 6.14.6)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

Windows Installer (.msi)	32-bit	64-bit
Windows Binary (.zip)	32-bit	64-bit
macOS Installer (.pkg)	64-bit	

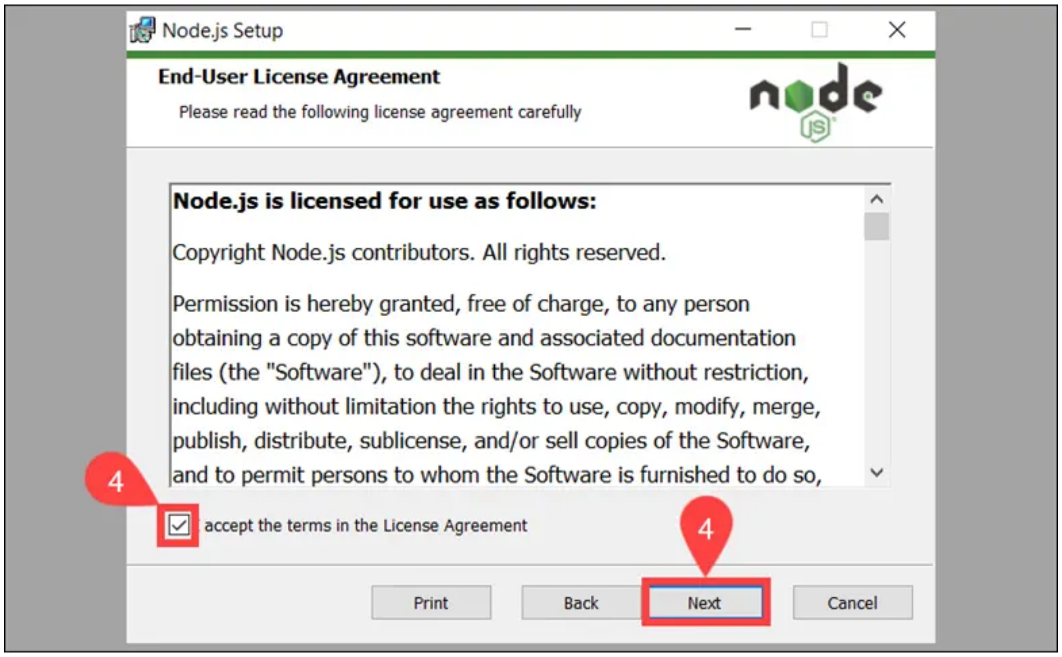
2. Access the download location and double-click the Windows Installer Package.
3. Select **Next** on the initial Node.js Setup Wizard screen.



Welcome to the Node.js Setup Wizard

The Setup Wizard will install Node.js on your computer.

4. **Accept** the *License Agreement* and click **Next.**


Node.js Setup

End-User License Agreement

Please read the following license agreement carefully

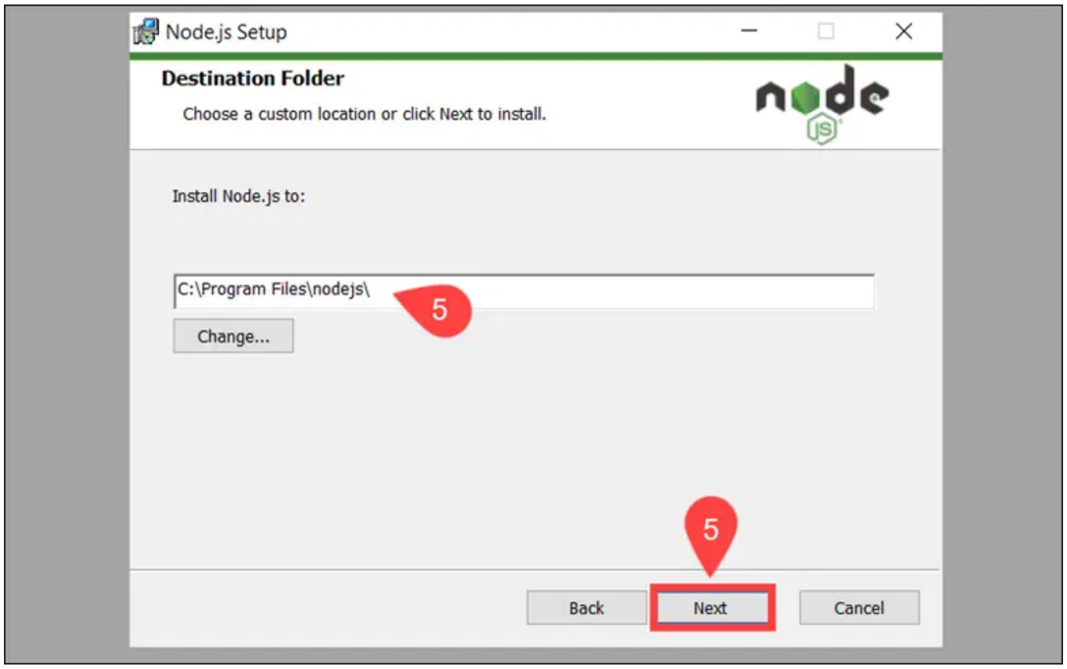
Node.js is licensed for use as follows:

Copyright Node.js contributors. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so,

[✓] accept the terms in the License Agreement

5. Define the **destination folder** for the installation and select **Next.**


Node.js Setup

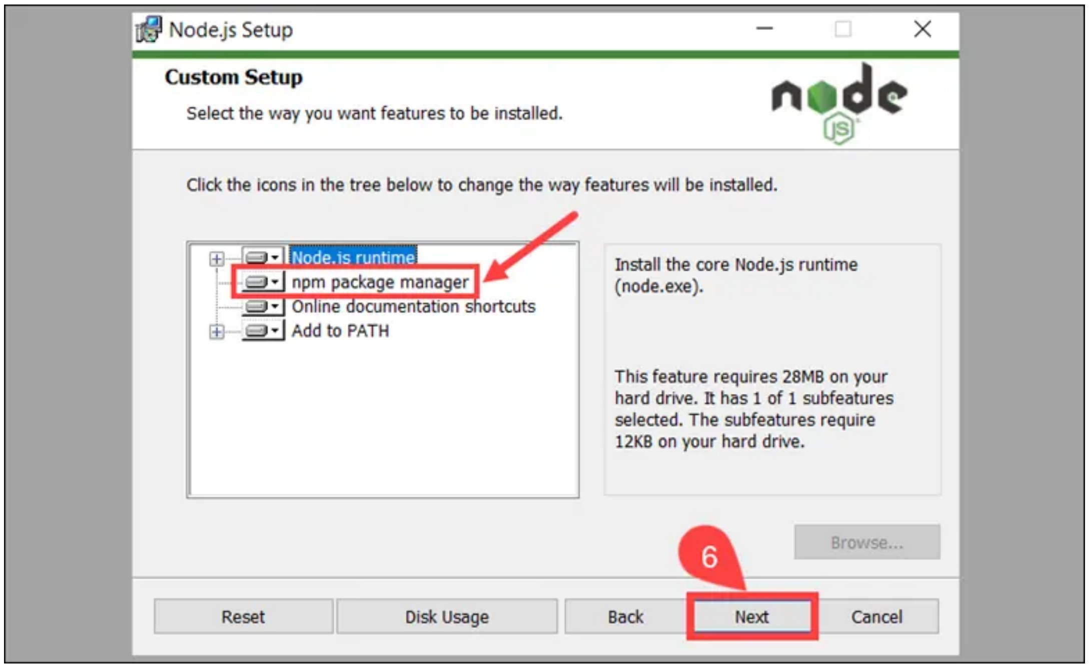
Destination Folder

Choose a custom location or click Next to install.

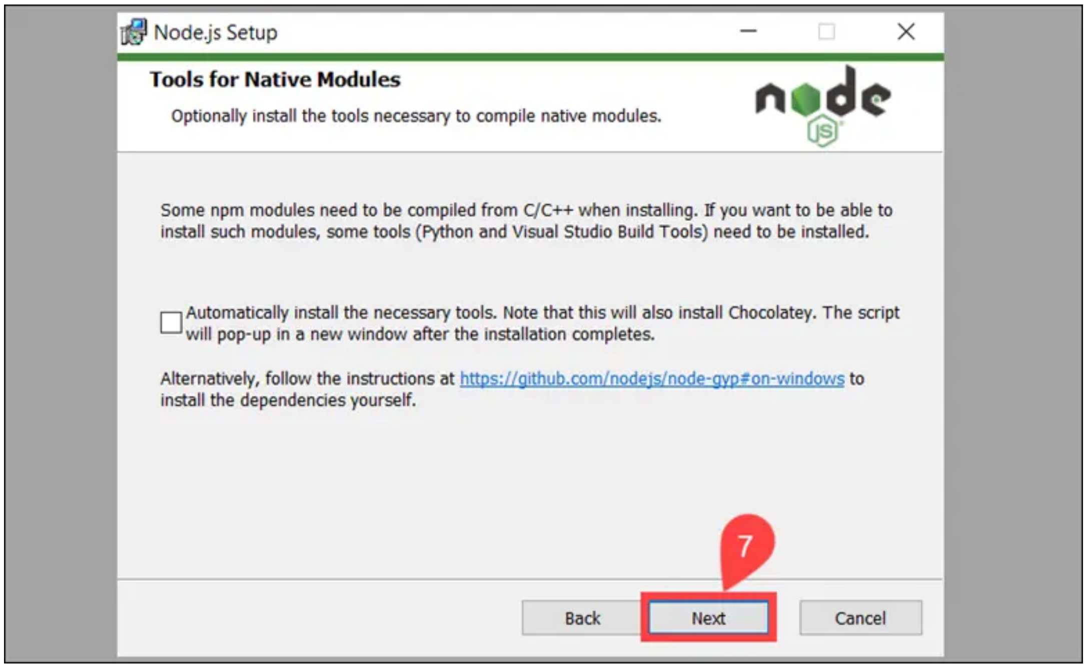
Install Node.js to:

C:\Program Files\nodejs\

6. You can customize how to install available features. Make sure that the **npm package manager** is part of the installation bundle. Click **Next** to proceed.


Node.js Setup
Custom Setup
Select the way you want features to be installed.
Click the icons in the tree below to change the way features will be installed.
Node.js runtime
npm package manager
Online documentation shortcuts
Add to PATH
Install the core Node.js runtime (node.exe).
This feature requires 28MB on your hard drive. It has 1 of 1 subfeatures selected. The subfeatures require 12KB on your hard drive.
Browse...
6
Reset
Disk Usage
Back
Next
Cancel

7. Check the box if you would like to install tools for compiling native modules automatically. They are not mandatory and require an additional 3 GB of space. Select **Next** to continue.


Node.js Setup

Tools for Native Modules

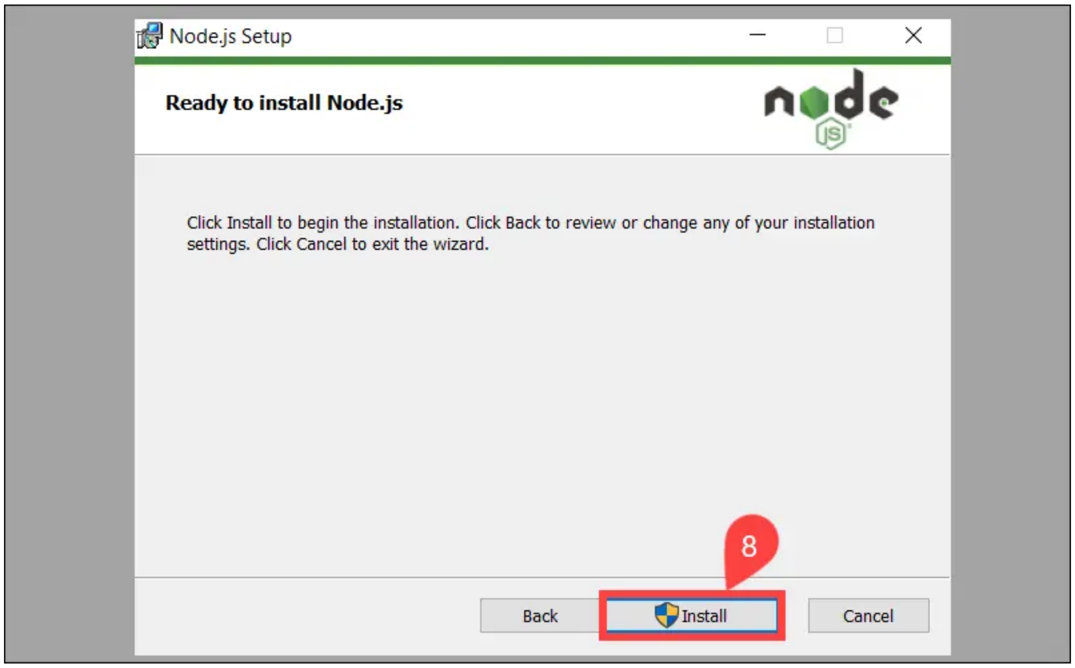
Optionally install the tools necessary to compile native modules.

Some npm modules need to be compiled from C/C++ when installing. If you want to be able to install such modules, some tools (Python and Visual Studio Build Tools) need to be installed.

☐ Automatically install the necessary tools. Note that this will also install Chocolatey. The script will pop-up in a new window after the installation completes.

Alternatively, follow the instructions at https://github.com/nodejs/node-gyp#on-windows to install the dependencies yourself.

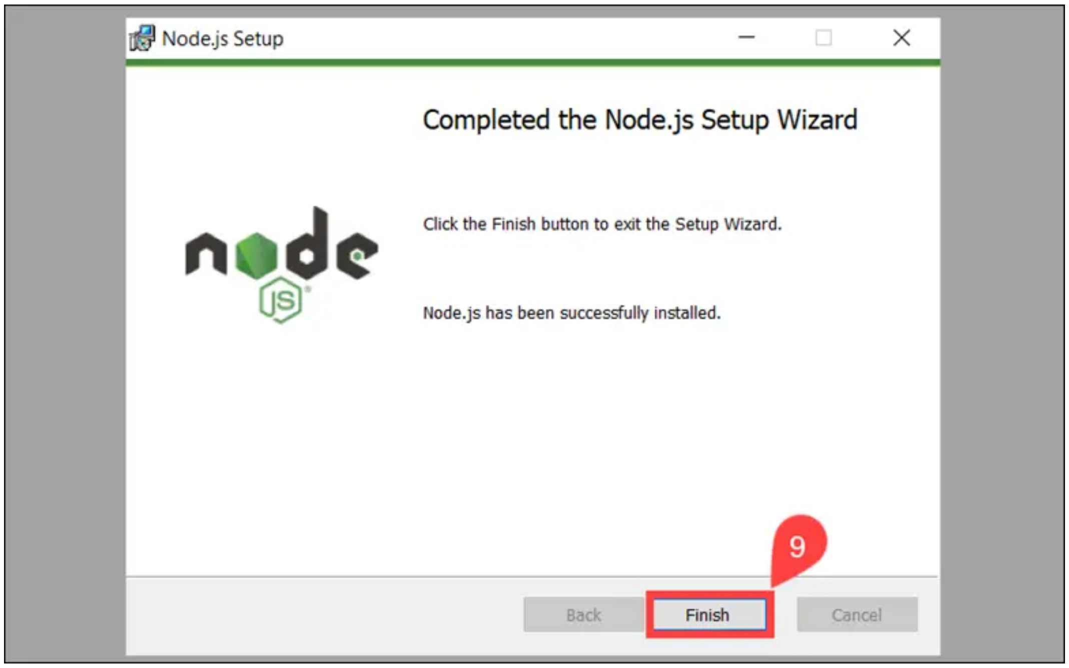
8. Click **Install** to start the installation process.



Ready to install Node.js

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

9. Once the installation is complete, select **Finish** to exit the Setup Wizard.

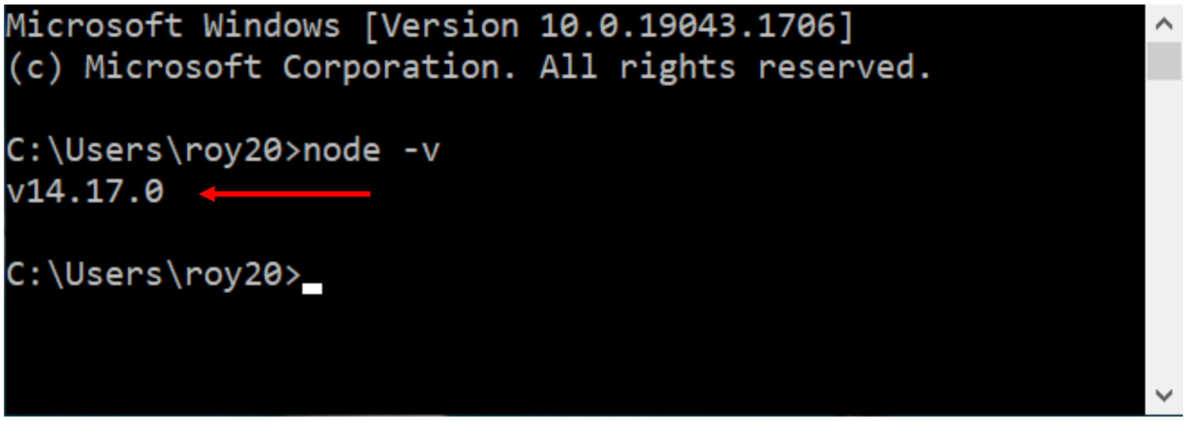


Completed the Node.js Setup Wizard

Click the Finish button to exit the Setup Wizard.

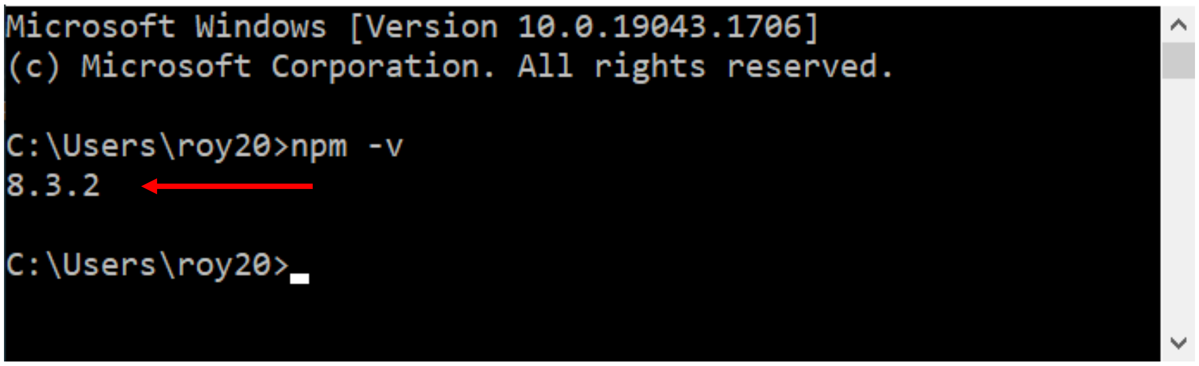
Node.js has been successfully installed.

10. Access the Windows Command Prompt (or PowerShell) and check the Node.js version:
    node -v
    The system confirms that Node.js **v14.17.0** is installed.



```
Microsoft Windows [Version 10.0.19043.1706]
(c) Microsoft Corporation. All rights reserved.

C:\Users\roy20>node -v
v14.17.0

C:\Users\roy20>_
```

11. To verify the NPM version, use the following command:
    npm -v
    The output shows you have installed NPM version 8.3.2.

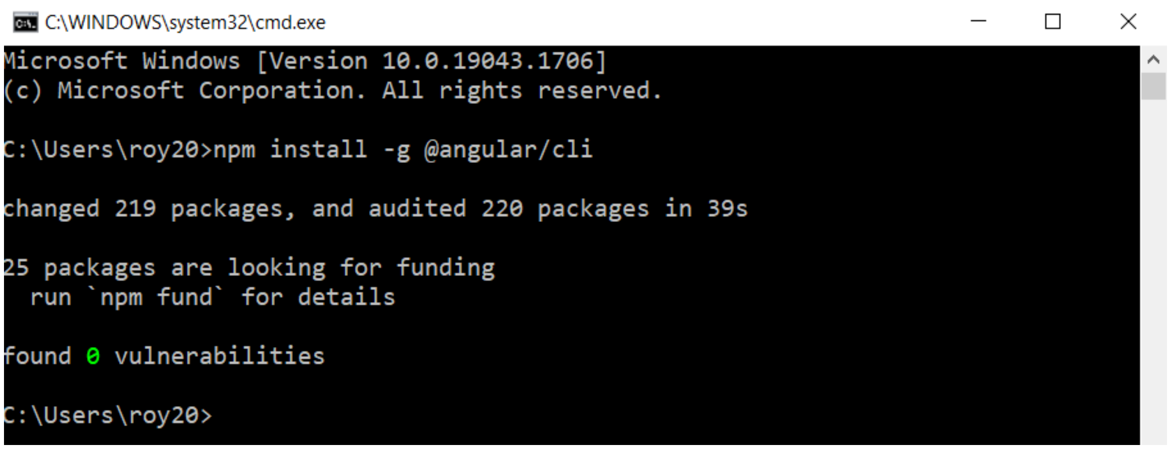


```
Microsoft Windows [Version 10.0.19043.1706]
(c) Microsoft Corporation. All rights reserved.

C:\Users\roy20>npm -v
8.3.2

C:\Users\roy20>_
```

Step 2: Install Angular CLI

The Angular command-line interface (CLI) tool allows you to initialize, develop, and manage your Angular applications. You can use the NPM package manager to install the Angular CLI.

1. Access the Windows Command Prompt and enter the following command:

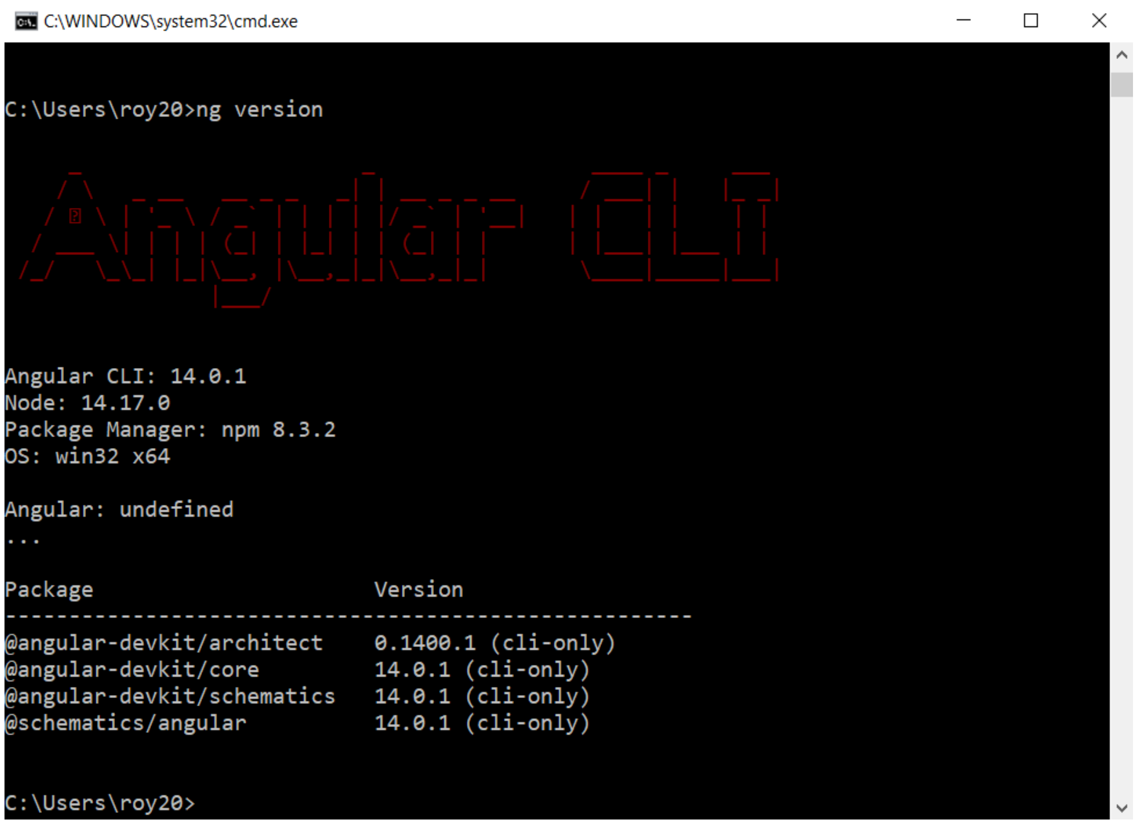

C:\WINDOWS\system32\cmd.exe

```
Microsoft Windows [Version 10.0.19043.1706]
(c) Microsoft Corporation. All rights reserved.

C:\Users\roy20>npm install -g @angular/cli

changed 219 packages, and audited 220 packages in 39s

25 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

C:\Users\roy20>
```

2. Once all packages have been added, verify the installed version:

ng version

The Angular CLI version is **14.0.1**


C:\WINDOWS\system32\cmd.exe

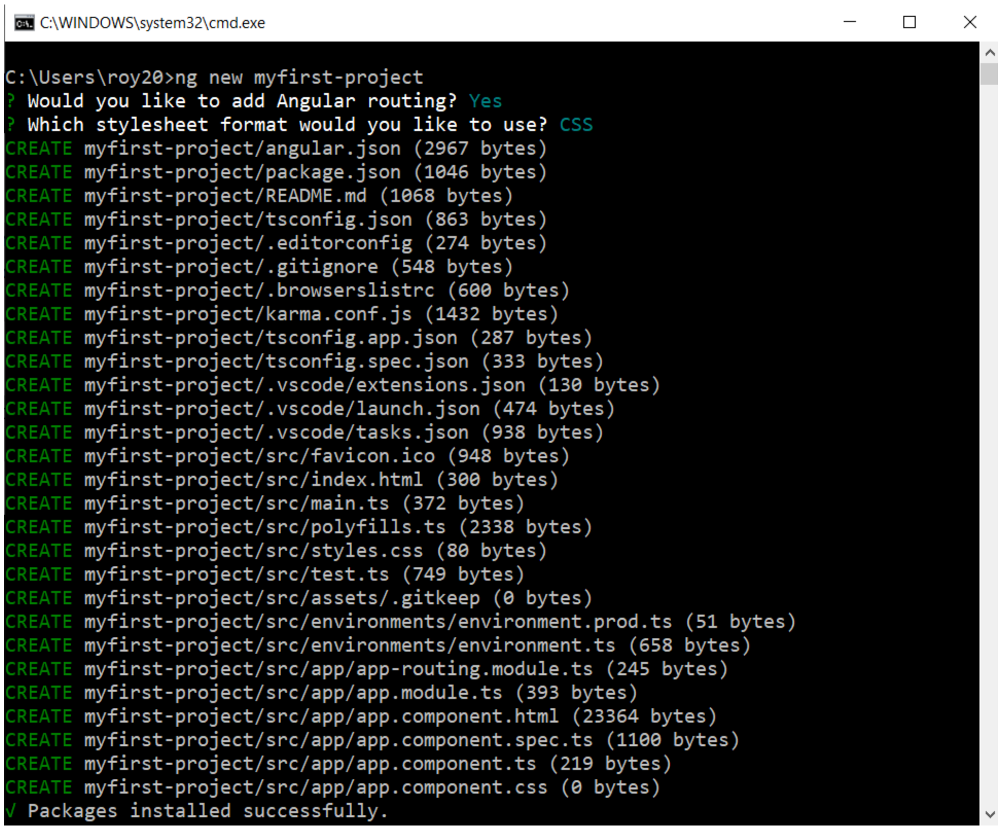
```

C:\Users\roy20>ng version

Angular CLI

Angular CLI: 14.0.1
Node: 14.17.0
Package Manager: npm 8.3.2
OS: win32 x64

Angular: undefined
...

Package                      Version
------------------------------------------------------
@angular-devkit/architect    0.1400.1 (cli-only)
@angular-devkit/core         14.0.1 (cli-only)
@angular-devkit/schematics   14.0.1 (cli-only)
@schematics/angular          14.0.1 (cli-only)


C:\Users\roy20>
```

Step 3: Create Angular Project

1. Use the Angular CLI to start a new Angular project. It this example, the name of the project is kitchen-sink. You are free to use the name of your choice for your project name. Type the following command in your Windows Command Prompt:

ng new myfirst-project

Before proceeding, you can customize the application. Define if you would like to use Angular routing and choose a stylesheet format. These settings can be edited at a later point.



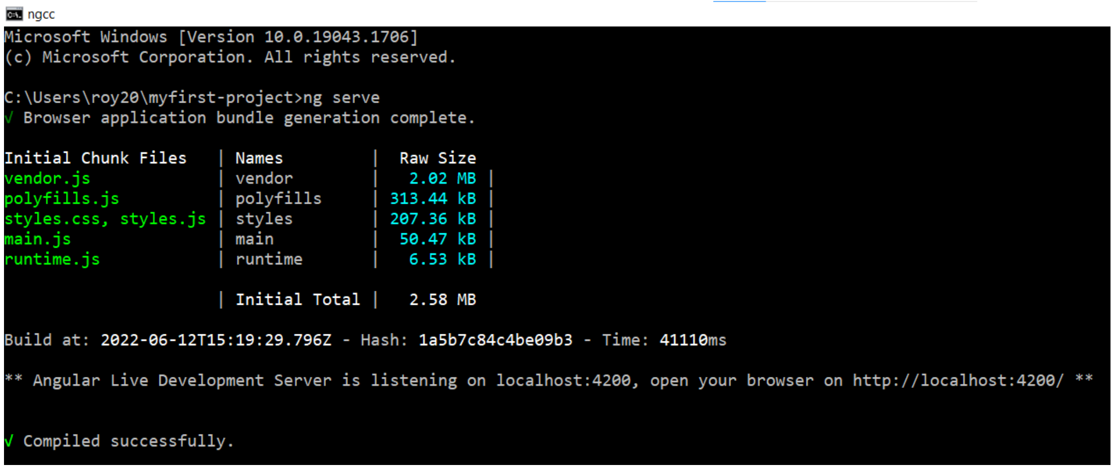
```
C:\WINDOWS\system32\cmd.exe

C:\Users\roy20>ng new myfirst-project
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
CREATE myfirst-project/angular.json (2967 bytes)
CREATE myfirst-project/package.json (1046 bytes)
CREATE myfirst-project/README.md (1068 bytes)
CREATE myfirst-project/tsconfig.json (863 bytes)
CREATE myfirst-project/.editorconfig (274 bytes)
CREATE myfirst-project/.gitignore (548 bytes)
CREATE myfirst-project/.browserslistrc (600 bytes)
CREATE myfirst-project/karma.conf.js (1432 bytes)
CREATE myfirst-project/tsconfig.app.json (287 bytes)
CREATE myfirst-project/tsconfig.spec.json (333 bytes)
CREATE myfirst-project/.vscode/extensions.json (130 bytes)
CREATE myfirst-project/.vscode/launch.json (474 bytes)
CREATE myfirst-project/.vscode/tasks.json (938 bytes)
CREATE myfirst-project/src/favicon.ico (948 bytes)
CREATE myfirst-project/src/index.html (300 bytes)
CREATE myfirst-project/src/main.ts (372 bytes)
CREATE myfirst-project/src/polyfills.ts (2338 bytes)
CREATE myfirst-project/src/styles.css (80 bytes)
CREATE myfirst-project/src/test.ts (749 bytes)
CREATE myfirst-project/src/assets/.gitkeep (0 bytes)
CREATE myfirst-project/src/environments/environment.prod.ts (51 bytes)
CREATE myfirst-project/src/environments/environment.ts (658 bytes)
CREATE myfirst-project/src/app/app-routing.module.ts (245 bytes)
CREATE myfirst-project/src/app/app.module.ts (393 bytes)
CREATE myfirst-project/src/app/app.component.html (23364 bytes)
CREATE myfirst-project/src/app/app.component.spec.ts (1100 bytes)
CREATE myfirst-project/src/app/app.component.ts (219 bytes)
CREATE myfirst-project/src/app/app.component.css (0 bytes)
√ Packages installed successfully.
```

The system takes a few moments to create the project. Once it finishes, you see the “Packages installed successfully” message.

2. Access your project’s root folder (myfirst-project in this example) from the Windows Command Prompt and enter the following command:

ng serve

The system proceeds to generate the environment for your Angular application.



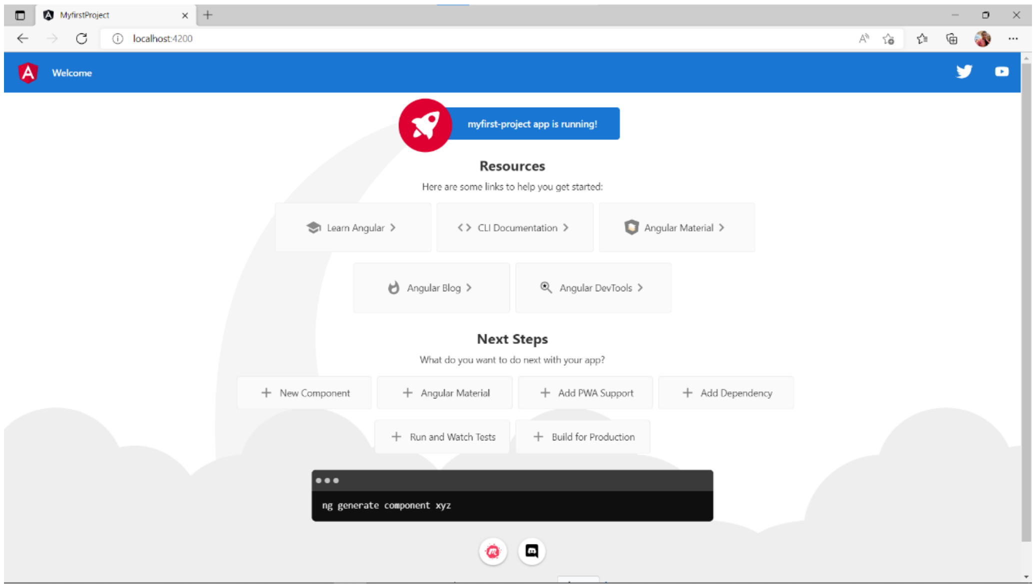
```
ngcc
Microsoft Windows [Version 10.0.19043.1706]
(c) Microsoft Corporation. All rights reserved.

C:\Users\roy20\myfirst-project>ng serve
√ Browser application bundle generation complete.

Initial Chunk Files    | Names          |  Raw Size
vendor.js              | vendor         |   2.02 MB |
polyfills.js           | polyfills      | 313.44 kB |
styles.css, styles.js  | styles         | 207.36 kB |
main.js                | main           |  50.47 kB |
runtime.js             | runtime        |   6.53 kB |

                       | Initial Total |   2.58 MB

Build at: 2022-06-12T15:19:29.796Z - Hash: 1a5b7c84c4be09b3 - Time: 41110ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **


√ Compiled successfully.
```

Keep the Windows Command Prompt running. You can continue using the terminal, and the changes you make will be reflected in the application within your browser.

3. Use any browser to access your Angular local development server on localhost:4200:

http://localhost:4200/

You can now start creating new Angular components.



```
ng generate component xyz
```

Activity 2

Aim: Create an Angular application which add, multiply, divide the two input numbers from textbox.

Learning outcome: Able to understand how to create an angular application.

Duration: 3 hours

List of Hardware/Software requirements:

1. Operating System – Windows 10/11 or Linux
2. Command Prompt/Power Shell
3. Internet Connectivity, Browser (Chrome or Edge)
4. Text Editor – Notepad / IDE – Visual Studio Code/Any IDE
5. NodeJS, Angular CLI

Code/Program/Procedure (with comments):

1. Open Visual Studio.
2. Let's create a simple AngularJS web application step by step and understand the basic building blocks of AngularJS.
3. First, create an HTML document with <head> and <body> elements, as show below.

Example: HTML Template

```
<!DOCTYPE html>

<html>

   <head>

   </head>

<body>
```

```
</html>
```

4. Include angular.js file in the head section (you have learned how to download angular library in the previous activity). You can take a reference from the CDN also.
   Example: Include AngularJS Library.

```
<!DOCTYPE html>

<html>

<head>

    <title>AngularApp</title>

    <script src= " https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.8.3/angular.js
"></script>

</head>

<body>


</body>

</html>
```

5. Here, we will be creating a simple calculator application which will add, subtract, multiply, divide two numbers and display the result.

```
<!Doctype html>

<html lang="en">

<head>

  <meta charset="utf-8">

  <title>AngularApp</title>

  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">

  <script src=
"https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.8.3/angular.js"></script>

</head>

<body ng-app>

 <h1>First AngularJS Application</h1>

    Enter Numbers to Add:

    <input type="number" ng-model="Num1" /> + <input type="number" ng-
model="Num2" /> = <span>{{Num1 + Num2}}</span>  <br><br>

    Enter Numbers to Subtract:

    <input type="number" ng-model="Num3" /> - <input type="number" ng-
model="Num4" /> = <span>{{Num3 - Num4}}</span>  <br><br>

    Enter Numbers to Multiply:

    <input type="number" ng-model="Num5" /> x <input type="number" ng-
model="Num6" /> = <span>{{Num5 * Num6}}</span> <br><br>

    Enter Numbers to Divide:

    <input type="number" ng-model="Num7" /> / <input type="number" ng-
model="Num8" /> = <span>{{Num7 / Num8}}</span>

</body>

</html>
```

The above example is looks like HTML code with some strange attributes and braces such as ng-app, ng-model, and {{ }}. These built-in attributes in AngularJS are called **directives**.

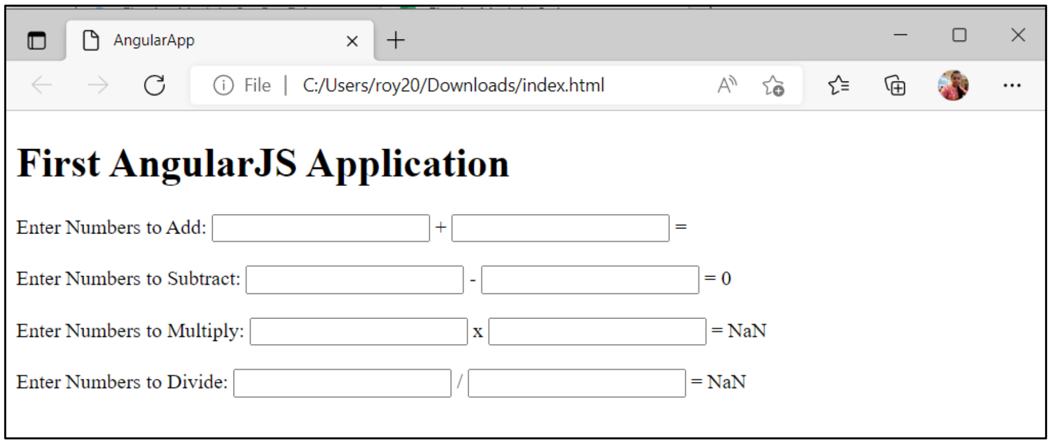
The expressions are written inside the ***{{ expression }}***.

ng-model directive helps to bind the value in html control. the above solution we use "Num1" it will contain the first input value."Num2" contain the second input value.

ng-app directive defines the angular js application.

6. Save as index.html

Output/Results snippet:



First AngularJS Application

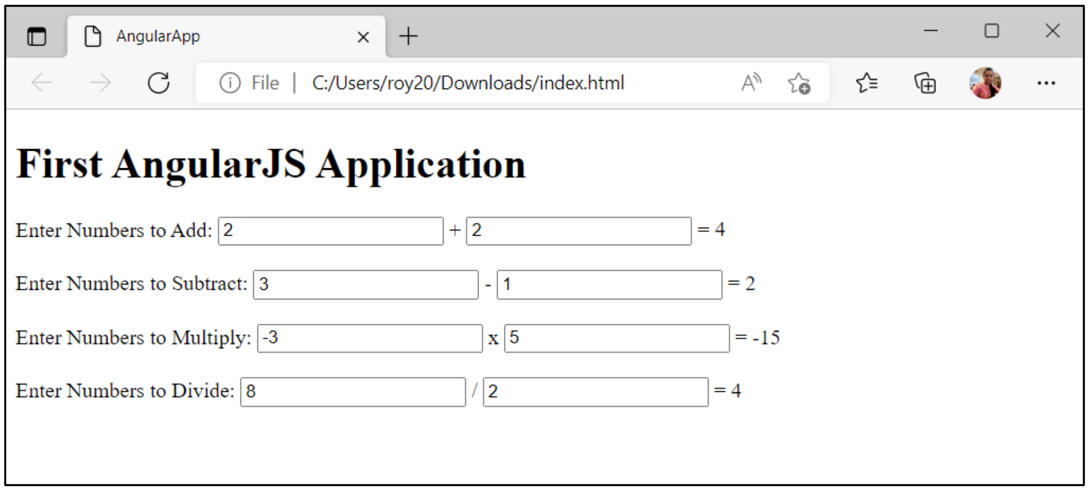
Enter Numbers to Add: + =

Enter Numbers to Subtract: - = 0

Enter Numbers to Multiply: x = NaN

Enter Numbers to Divide: / = NaN

User will enter two numbers in two separate textboxes and the result will be displayed immediately, as shown below.



First AngularJS Application

Enter Numbers to Add: 2 + 2 = 4

Enter Numbers to Subtract: 3 - 1 = 2

Enter Numbers to Multiply: -3 x 5 = -15

Enter Numbers to Divide: 8 / 2 = 4

References:

- https://dzone.com/articles/multiplication-of-two-numbers-using-angular-js

Learning Outcome

After completing this module, the student should be **Able to develop the real time scenarios based on Node JS applications.**

To meet the learning outcome, a student has to complete the following activities

1. Create an Angular application which can validate the email accepted from user
2. Create an Angular application with form validation

Activity 1

Aim: Create an Angular application which can validate the email accepted from user

Learning outcome: Able to develop the real time scenarios based on Node JS applications.

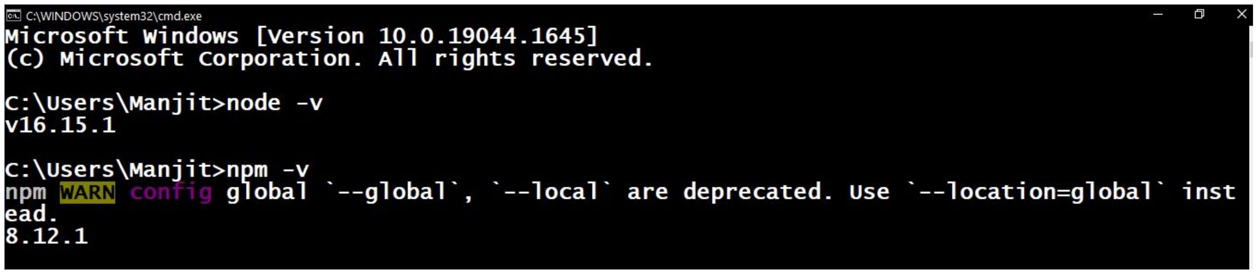
Duration: 3 Hour

List of Hardware/Software requirements:

6. Operating System – Windows 10/11 or Linux
7. Command Prompt/Power Shell
8. Internet Connectivity, Browser (Chrome or Edge)
9. Text Editor – Notepad / IDE – Visual Studio Code/Any IDE
10. NodeJS, Angular CLI

Code/Program/Procedure (with comments):

Step 1: To verify the node, npm version, use the following command:



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.19044.1645]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Manjit>node -v
v16.15.1

C:\Users\Manjit>npm -v
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` inst
ead.
8.12.1
```

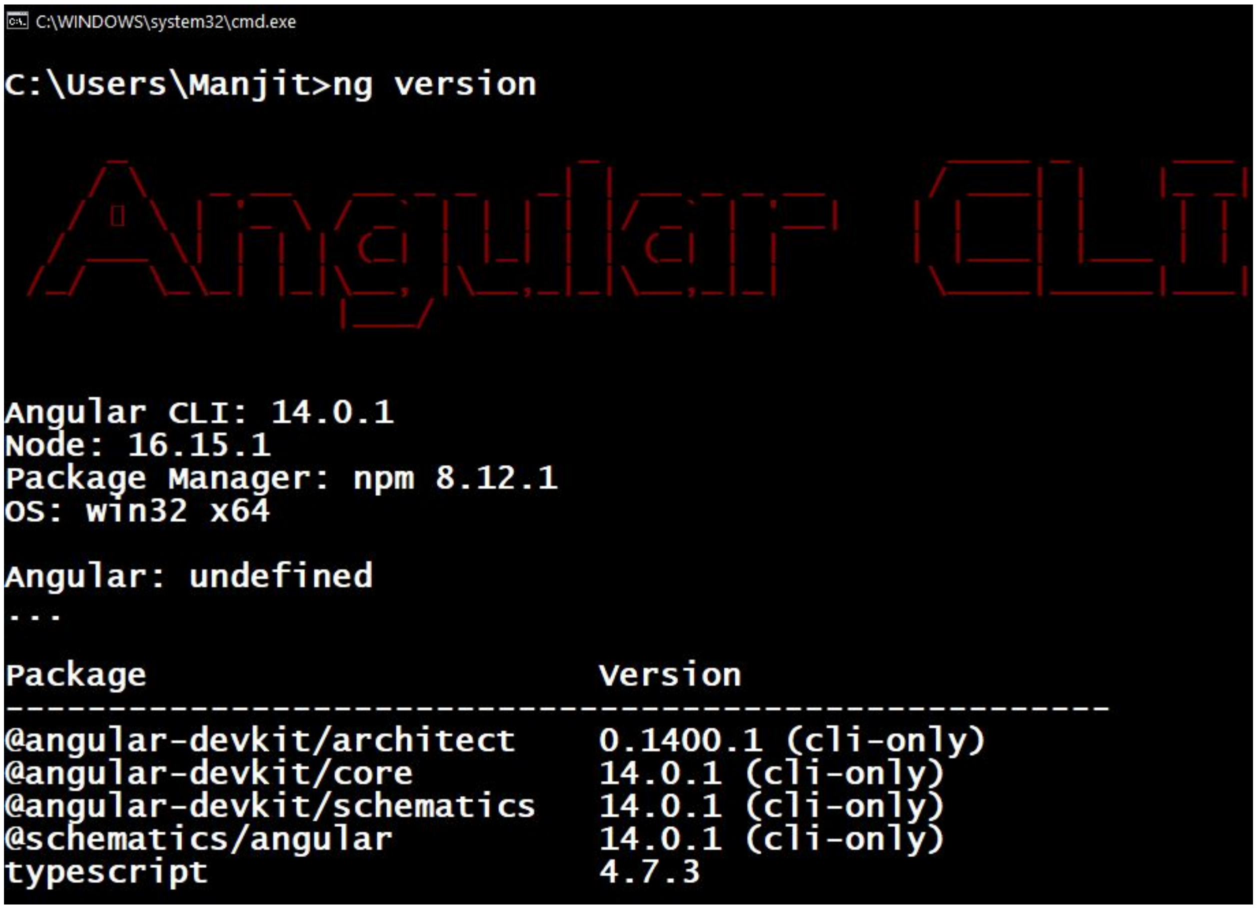
Step 2: Installing Angular CLI

The Angular command-line interface (CLI) tool allows you to initialize, develop, and manage your Angular applications. You can use the NPM package manager to install the Angular CLI.

Access the Windows Command Prompt and enter the following command:

First step, where we'll have to install latest version of Angular CLI 14.0.1

$ npm install -g @angular/cli

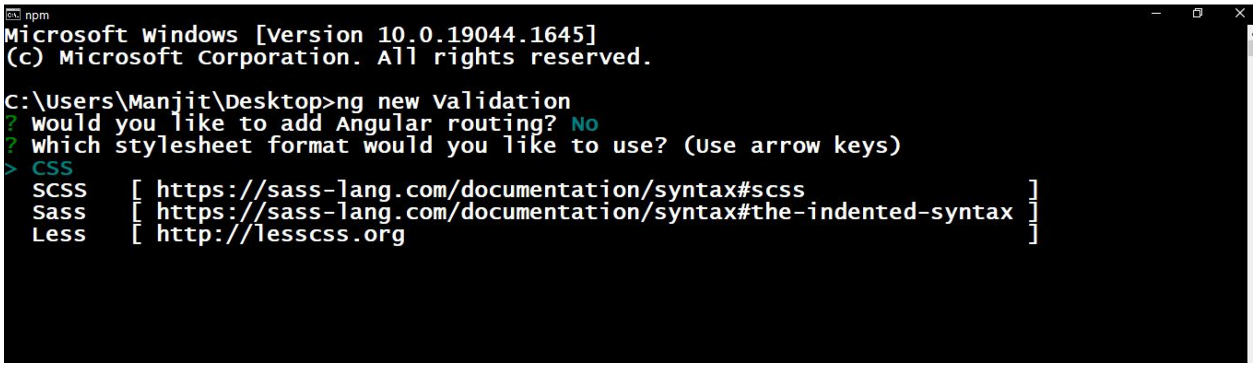


```
C:\WINDOWS\system32\cmd.exe

C:\Users\Manjit>ng version

Angular CLI: 14.0.1
Node: 16.15.1
Package Manager: npm 8.12.1
OS: win32 x64

Angular: undefined
...

Package                         Version
---------------------------------------------------------
@angular-devkit/architect       0.1400.1 (cli-only)
@angular-devkit/core            14.0.1 (cli-only)
@angular-devkit/schematics      14.0.1 (cli-only)
@schematics/angular             14.0.1 (cli-only)
typescript                      4.7.3
```

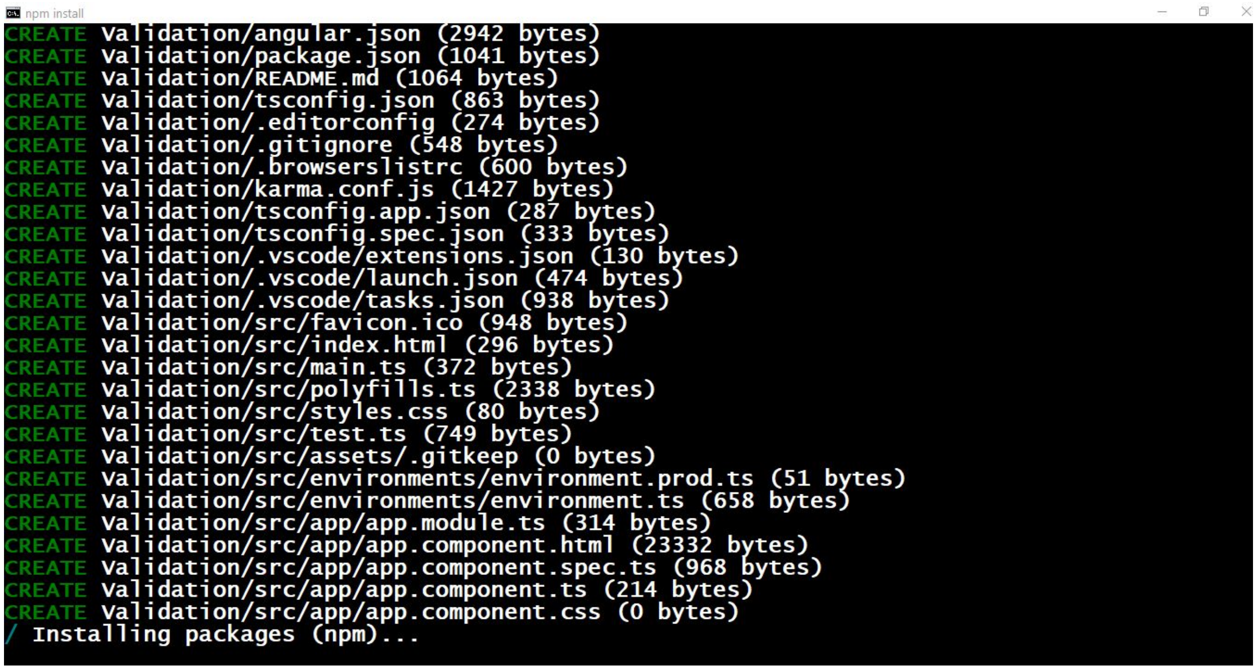
Step 3: Creating your Angular 12 Project

- In this second step, we will use Angular CLI to start our Angular Project
- Go to CMD or Terminal and use this command:

ng new Validation



```
Microsoft Windows [Version 10.0.19044.1645]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Manjit\Desktop>ng new Validation
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
  SCSS   [ https://sass-lang.com/documentation/syntax#scss                ]
  Sass   [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less   [ http://lesscss.org                                             ]
```



```
CREATE Validation/angular.json (2942 bytes)
CREATE Validation/package.json (1041 bytes)
CREATE Validation/README.md (1064 bytes)
CREATE Validation/tsconfig.json (863 bytes)
CREATE Validation/.editorconfig (274 bytes)
CREATE Validation/.gitignore (548 bytes)
CREATE Validation/.browserslistrc (600 bytes)
CREATE Validation/karma.conf.js (1427 bytes)
CREATE Validation/tsconfig.app.json (287 bytes)
CREATE Validation/tsconfig.spec.json (333 bytes)
CREATE Validation/.vscode/extensions.json (130 bytes)
CREATE Validation/.vscode/launch.json (474 bytes)
CREATE Validation/.vscode/tasks.json (938 bytes)
CREATE Validation/src/favicon.ico (948 bytes)
CREATE Validation/src/index.html (296 bytes)
CREATE Validation/src/main.ts (372 bytes)
CREATE Validation/src/polyfills.ts (2338 bytes)
CREATE Validation/src/styles.css (80 bytes)
CREATE Validation/src/test.ts (749 bytes)
CREATE Validation/src/assets/.gitkeep (0 bytes)
CREATE Validation/src/environments/environment.prod.ts (51 bytes)
CREATE Validation/src/environments/environment.ts (658 bytes)
CREATE Validation/src/app/app.module.ts (314 bytes)
CREATE Validation/src/app/app.component.html (23332 bytes)
CREATE Validation/src/app/app.component.spec.ts (968 bytes)
CREATE Validation/src/app/app.component.ts (214 bytes)
CREATE Validation/src/app/app.component.css (0 bytes)
/ Installing packages (npm)...
```

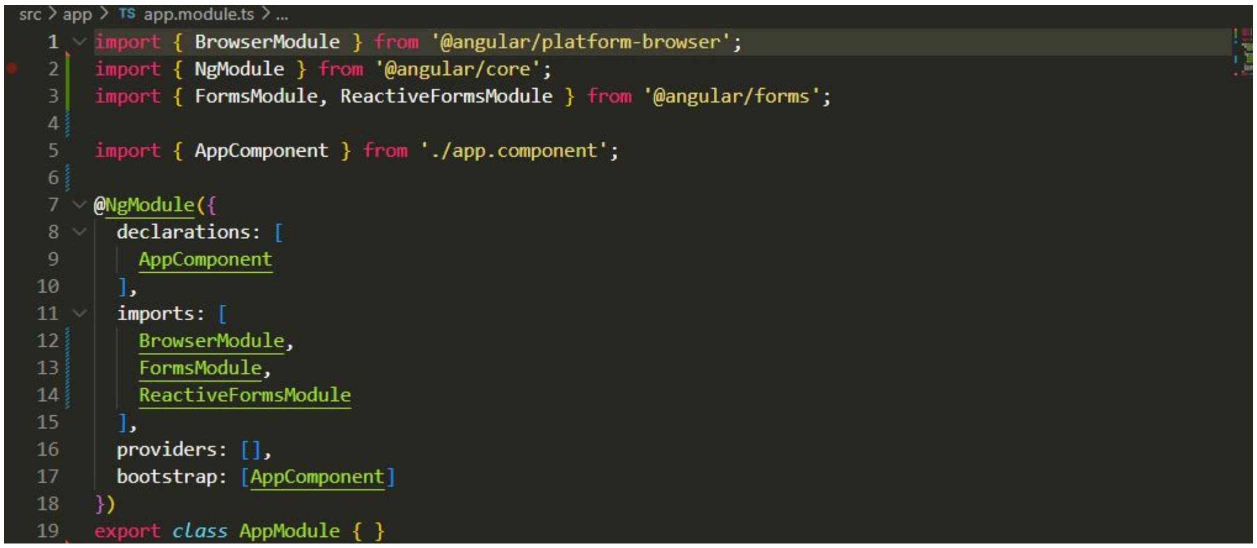
Once all packages have been added.

- This CLI will ask you “whether you would like to add Angular routing” Say Yes.
- It will ask “which stylesheet format you would like to use”. Choose CSS.
- Now your project is ready Angular CLI will generate required files and folders along with NPM packages and routing too.

Step 4: Import FormsModule

If you want to create form in angular app then you need to import FormsModule from @angular/forms library. so let's add following code to app.module.ts file.

src/app/app.module.ts

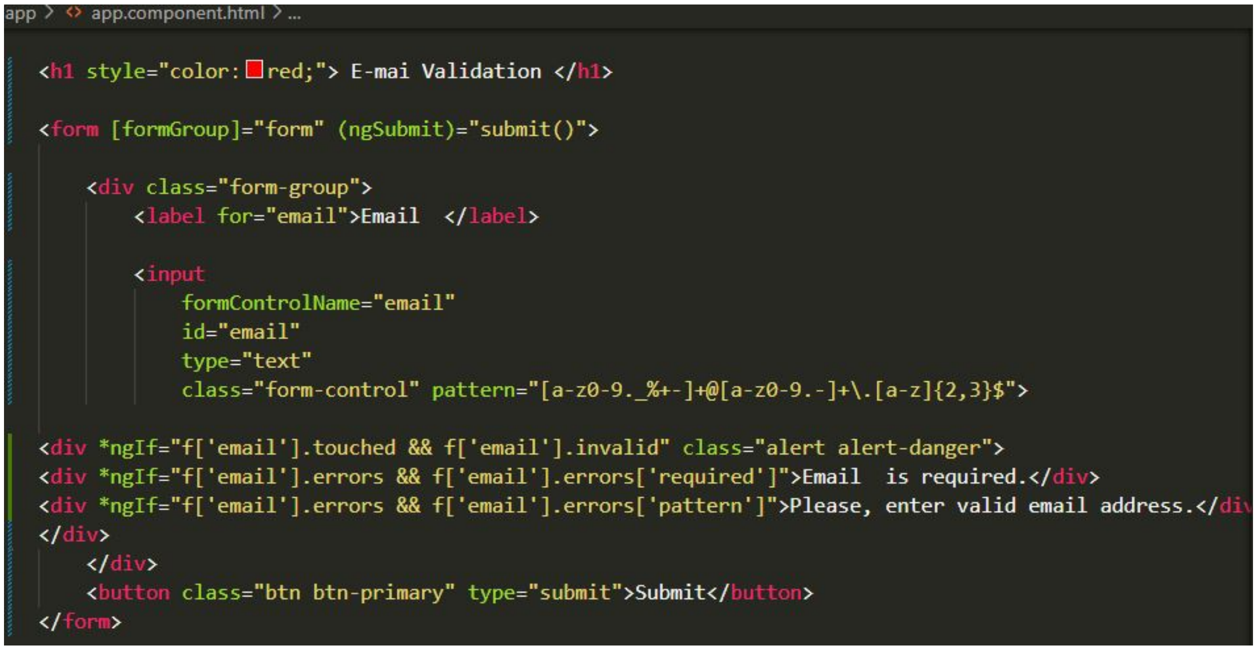


src > app > TS app.module.ts > ...

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    ReactiveFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Step 5: Form with ngModel

In this step, we will write code of html form with ngModel. so, add following code to app.component.html file



app > <> app.component.html > ...

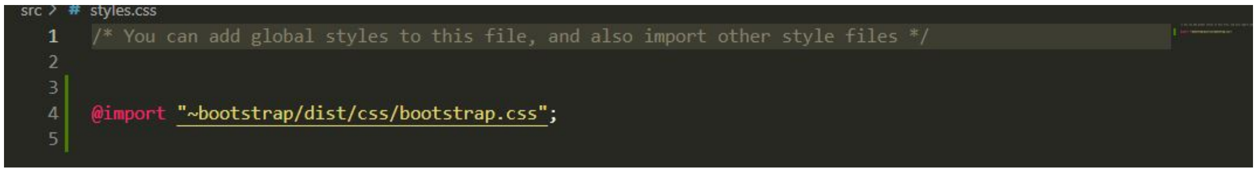
```
<h1 style="color: red;"> E-mai Validation </h1>

<form [formGroup]="form" (ngSubmit)="submit()">

    <div class="form-group">
        <label for="email">Email  </label>

        <input
            formControlName="email"
            id="email"
            type="text"
            class="form-control" pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,3}$">

<div *ngIf="f['email'].touched && f['email'].invalid" class="alert alert-danger">
<div *ngIf="f['email'].errors && f['email'].errors['required']">Email  is required.</div>
<div *ngIf="f['email'].errors && f['email'].errors['pattern']">Please, enter valid email address.</di
</div>
    </div>
    <button class="btn btn-primary" type="submit">Submit</button>
</form>
```

In this solution, you need to just install bootstrap on your angular 12 and import css file to style.css file. this is only for css importing. so, you can run command bellow:

npm install bootstrap --save

You need to import your bootstrap css on style.css file as like bellow:

src/style.css



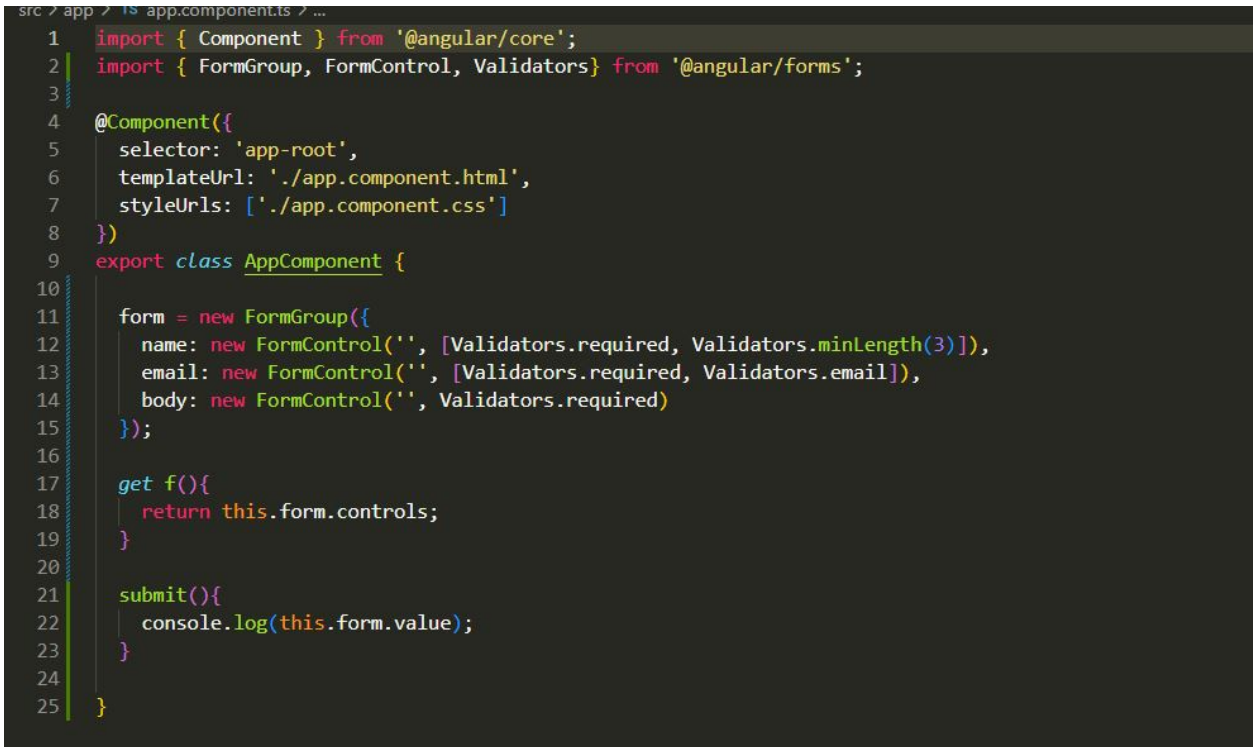
src > # styles.css

```
/* You can add global styles to this file, and also import other style files */


@import "~bootstrap/dist/css/bootstrap.css";

```

Step 6: updated Ts File

In ts file. we will write submit() and get all input fields values. so let's add following code to app.component.ts file.

src/app/app.component.ts


src > app > TS app.component.ts > ...

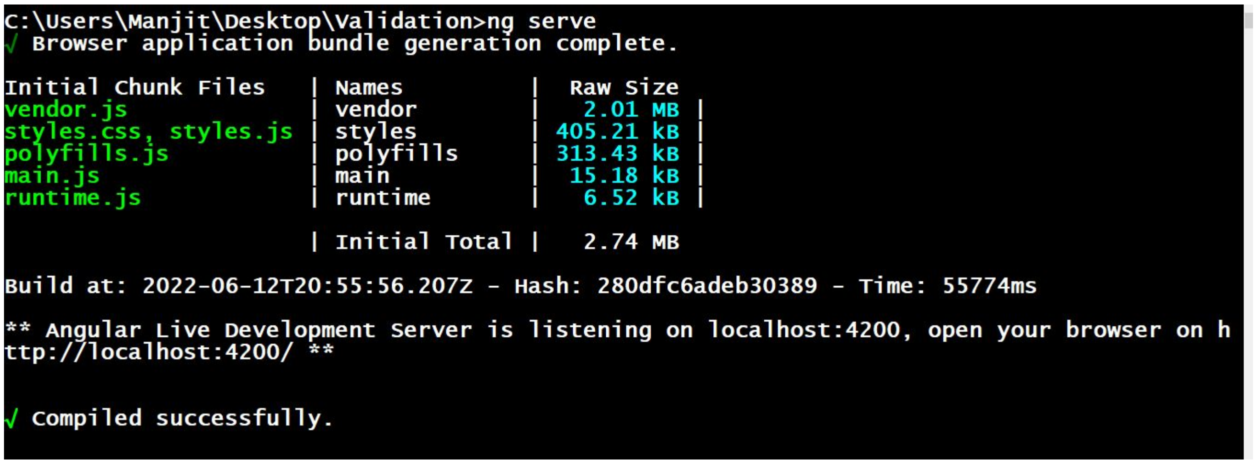
```typescript
import { Component } from '@angular/core';
import { FormGroup, FormControl, Validators} from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  form = new FormGroup({
    name: new FormControl('', [Validators.required, Validators.minLength(3)]),
    email: new FormControl('', [Validators.required, Validators.email]),
    body: new FormControl('', Validators.required)
  });

  get f(){
    return this.form.controls;
  }

  submit(){
    console.log(this.form.value);
  }

}
```

Step 7: Run your application

The system proceeds to generate the environment for your Angular application.

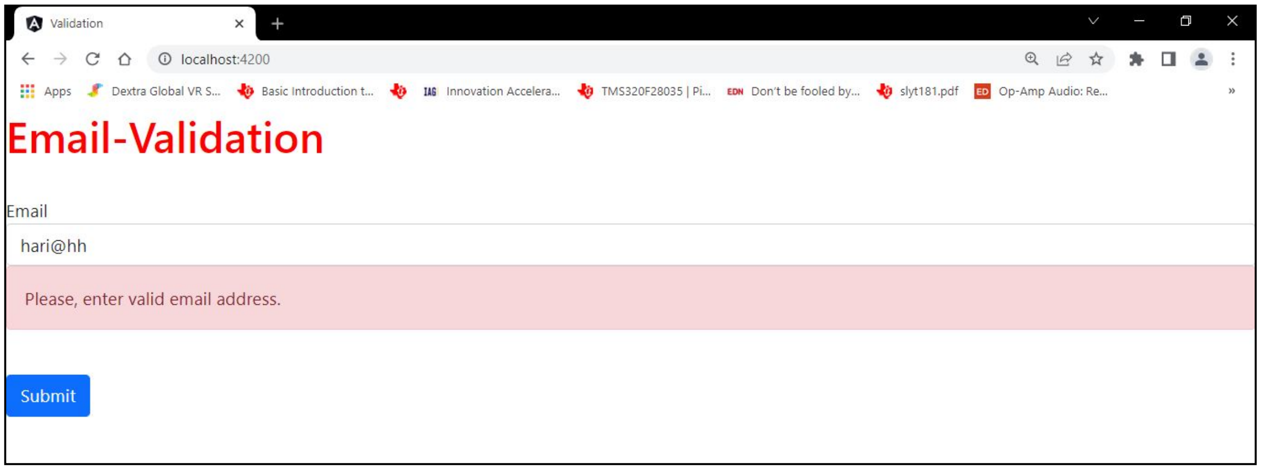
Keep the Windows Command Prompt running. You can continue using the terminal, and the changes you make will be reflected in the application within your browser.

Use any browser to access your Angular local development server on localhost:4200:



```
C:\Users\Manjit\Desktop\Validation>ng serve
√ Browser application bundle generation complete.

Initial Chunk Files    | Names          |  Raw Size
vendor.js              | vendor         |   2.01 MB |
styles.css, styles.js  | styles         | 405.21 kB |
polyfills.js           | polyfills      | 313.43 kB |
main.js                | main           |  15.18 kB |
runtime.js             | runtime        |   6.52 kB |

                       | Initial Total |   2.74 MB

Build at: 2022-06-12T20:55:56.207Z - Hash: 280dfc6adeb30389 - Time: 55774ms

** Angular Live Development Server is listening on localhost:4200, open your browser on h
ttp://localhost:4200/ **


√ Compiled successfully.
```

Output/Results snippet:



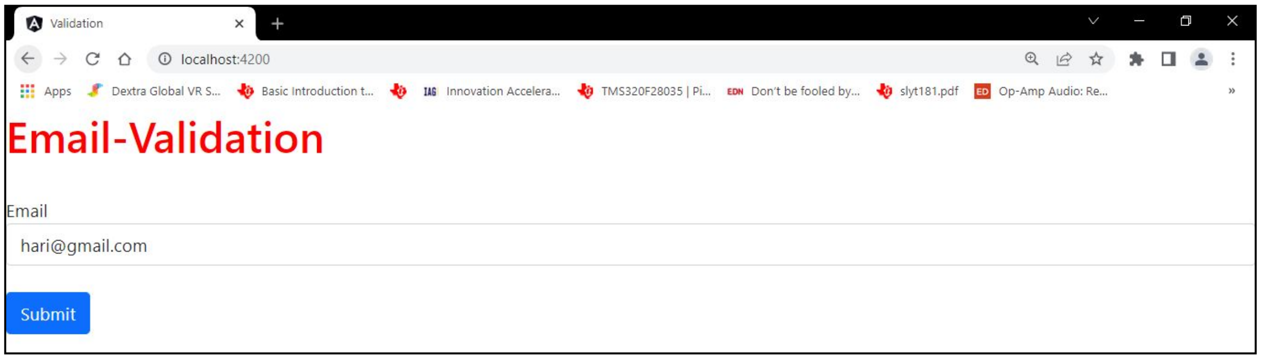
Email-Validation

Email

hari@hh

Please, enter valid email address.

Submit



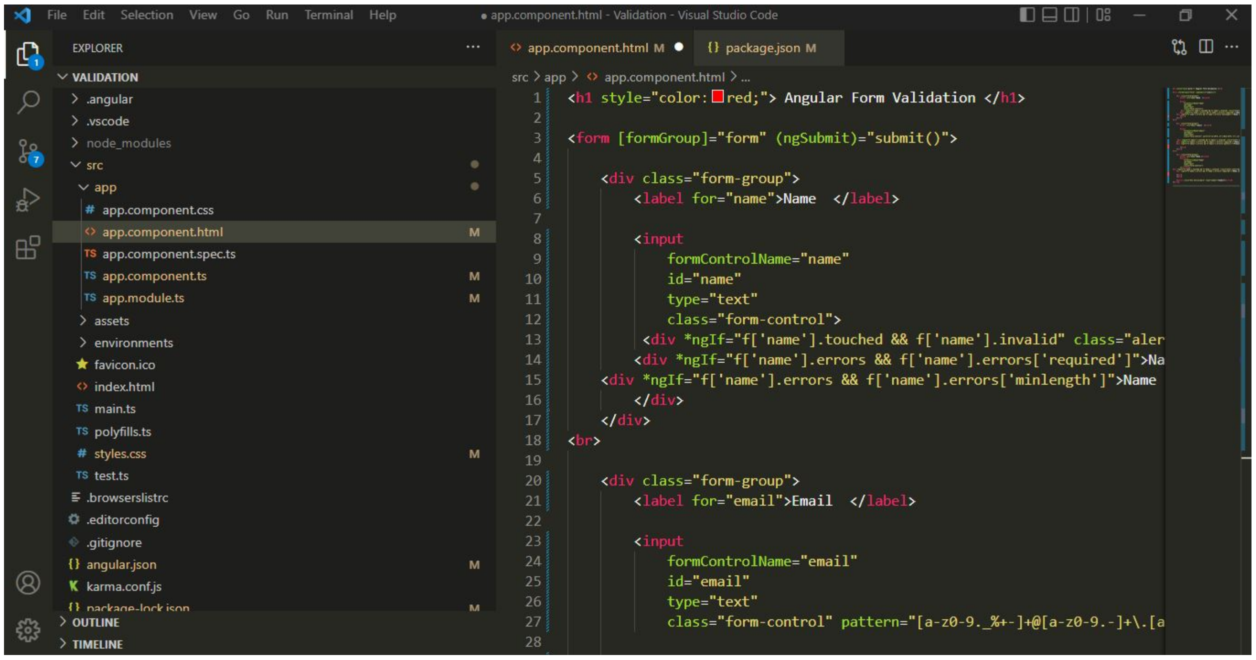
Email-Validation

Email

hari@gmail.com

Submit

Project Directory:



```
<h1 style="color:red;"> Angular Form Validation </h1>

<form [formGroup]="form" (ngSubmit)="submit()">

    <div class="form-group">
        <label for="name">Name  </label>

        <input
            formControlName="name"
            id="name"
            type="text"
            class="form-control">
         <div *ngIf="f['name'].touched && f['name'].invalid" class="aler
        <div *ngIf="f['name'].errors && f['name'].errors['required']">Na
    <div *ngIf="f['name'].errors && f['name'].errors['minlength']">Name
        </div>
    </div>
<br>

    <div class="form-group">
        <label for="email">Email  </label>

        <input
            formControlName="email"
            id="email"
            type="text"
            class="form-control" pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a

```

References:

- https://www.itsolutionstuff.com/post/angular-12-reactive-forms-validation-exampleexample.html

  .

Activity 2

Aim: Create an Angular application with form validation

Learning outcome: Able to develop the real time scenarios based on Node JS applications.

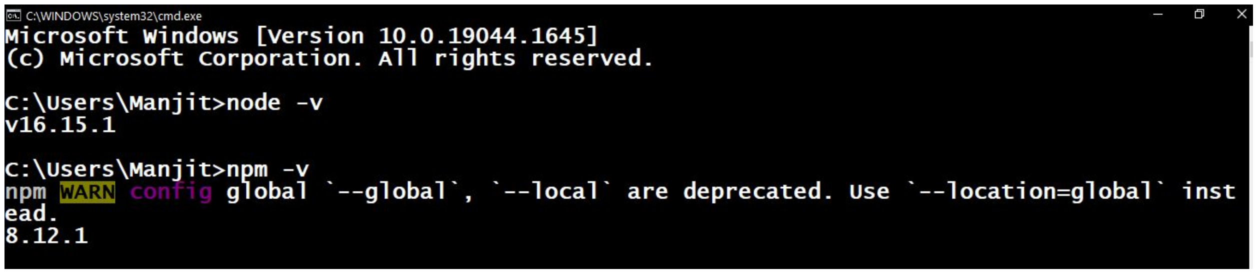
Duration: 3 Hour

List of Hardware/Software requirements:

1. Operating System – Windows 10/11 or Linux
2. Command Prompt/Power Shell
3. Internet Connectivity, Browser (Chrome or Edge)
4. Text Editor – Notepad / IDE – Visual Studio Code/Any IDE
5. NodeJS, Angular CLI

Code/Program/Procedure (with comments):

Step 1: To verify the node, npm version, use the following command:



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.19044.1645]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Manjit>node -v
v16.15.1

C:\Users\Manjit>npm -v
npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` inst
ead.
8.12.1
```

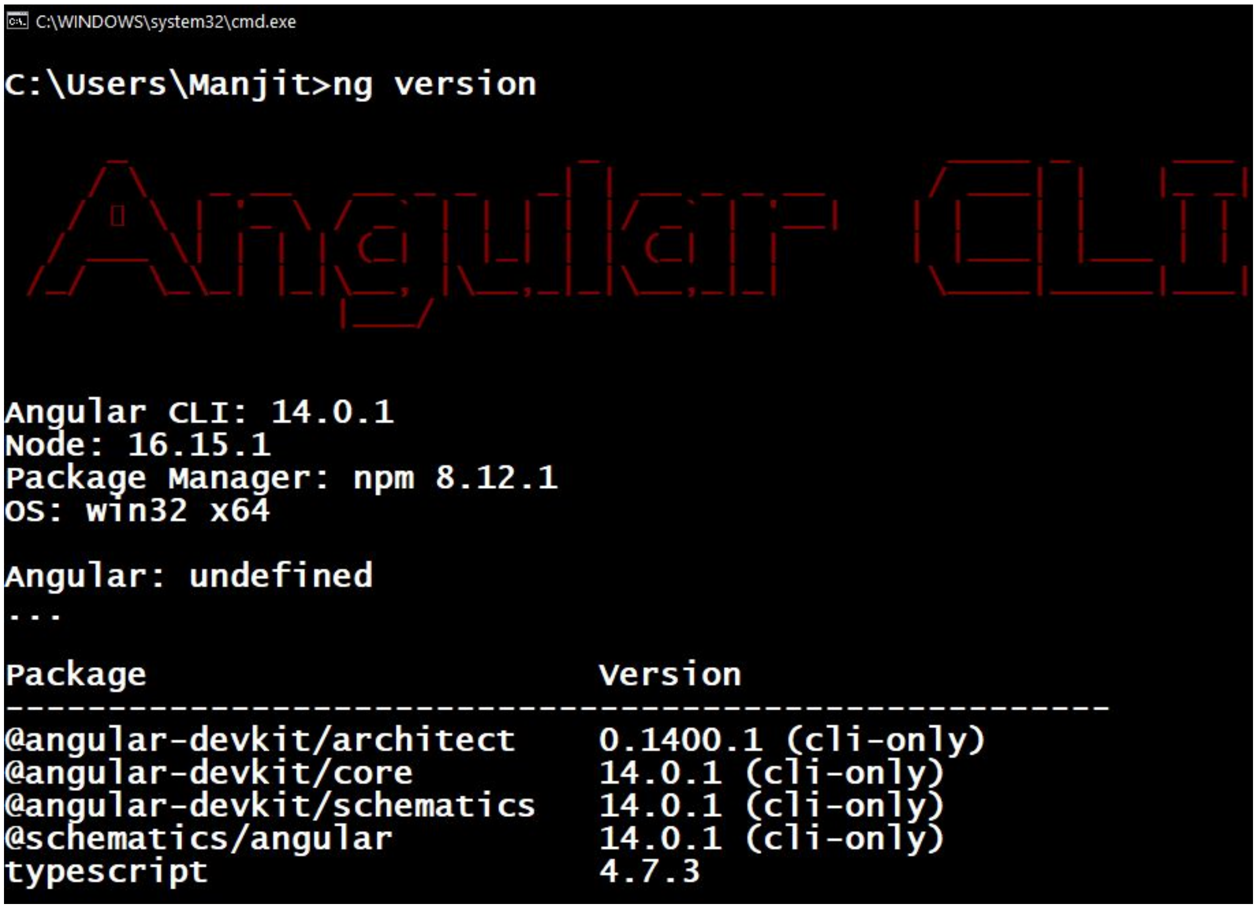
Step 2: Installing Angular CLI

The Angular command-line interface (CLI) tool allows you to initialize, develop, and manage your Angular applications. You can use the NPM package manager to install the Angular CLI.

Access the Windows Command Prompt and enter the following command:

First step, where we'll have to install latest version of Angular CLI 14.0.1

$ npm install -g @angular/cli

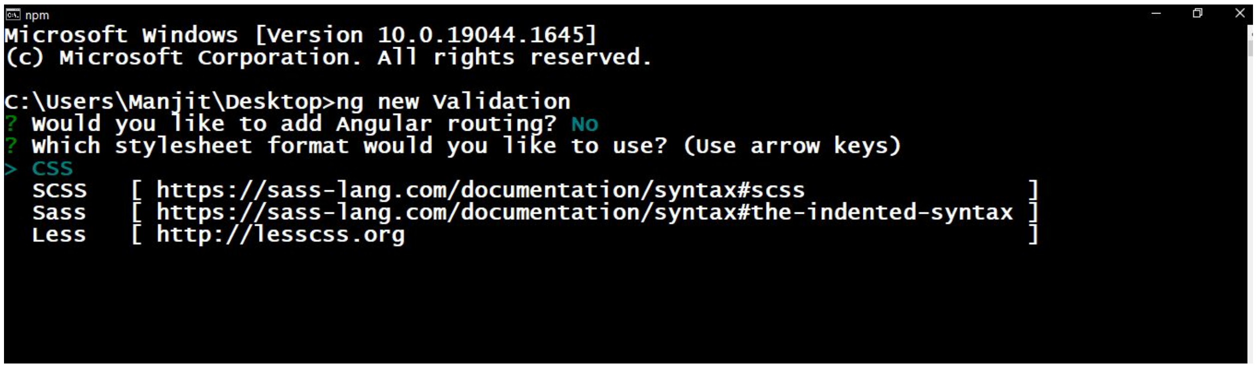


```
C:\Users\Manjit>ng version

Angular CLI: 14.0.1
Node: 16.15.1
Package Manager: npm 8.12.1
OS: win32 x64

Angular: undefined
...

Package                        Version
------------------------------------------------------
@angular-devkit/architect      0.1400.1 (cli-only)
@angular-devkit/core           14.0.1 (cli-only)
@angular-devkit/schematics     14.0.1 (cli-only)
@schematics/angular            14.0.1 (cli-only)
typescript                     4.7.3
```

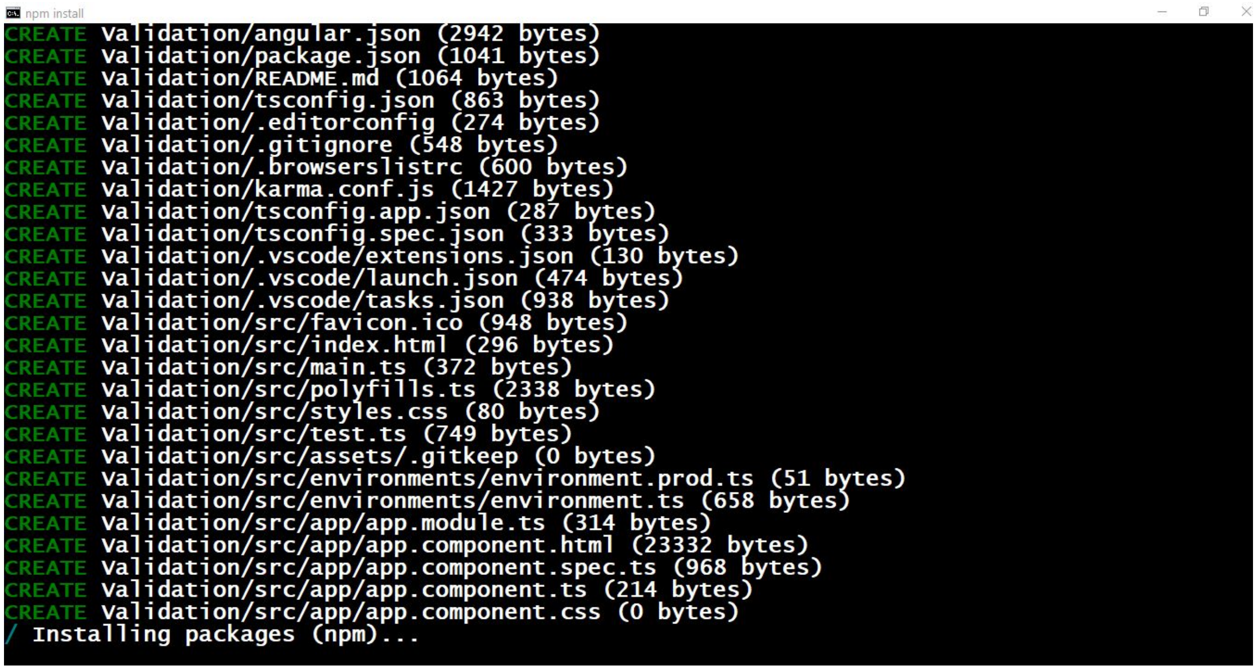
Step 3: Creating your Angular 12 Project

- In this second step, we will use Angular CLI to start our Angular Project
- Go to CMD or Terminal and use this command:

ng new Validation



```
Microsoft Windows [Version 10.0.19044.1645]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Manjit\Desktop>ng new Validation
? Would you like to add Angular routing? No
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
  SCSS   [ https://sass-lang.com/documentation/syntax#scss                ]
  Sass   [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less   [ http://lesscss.org                                             ]
```



```
CREATE Validation/angular.json (2942 bytes)
CREATE Validation/package.json (1041 bytes)
CREATE Validation/README.md (1064 bytes)
CREATE Validation/tsconfig.json (863 bytes)
CREATE Validation/.editorconfig (274 bytes)
CREATE Validation/.gitignore (548 bytes)
CREATE Validation/.browserslistrc (600 bytes)
CREATE Validation/karma.conf.js (1427 bytes)
CREATE Validation/tsconfig.app.json (287 bytes)
CREATE Validation/tsconfig.spec.json (333 bytes)
CREATE Validation/.vscode/extensions.json (130 bytes)
CREATE Validation/.vscode/launch.json (474 bytes)
CREATE Validation/.vscode/tasks.json (938 bytes)
CREATE Validation/src/favicon.ico (948 bytes)
CREATE Validation/src/index.html (296 bytes)
CREATE Validation/src/main.ts (372 bytes)
CREATE Validation/src/polyfills.ts (2338 bytes)
CREATE Validation/src/styles.css (80 bytes)
CREATE Validation/src/test.ts (749 bytes)
CREATE Validation/src/assets/.gitkeep (0 bytes)
CREATE Validation/src/environments/environment.prod.ts (51 bytes)
CREATE Validation/src/environments/environment.ts (658 bytes)
CREATE Validation/src/app/app.module.ts (314 bytes)
CREATE Validation/src/app/app.component.html (23332 bytes)
CREATE Validation/src/app/app.component.spec.ts (968 bytes)
CREATE Validation/src/app/app.component.ts (214 bytes)
CREATE Validation/src/app/app.component.css (0 bytes)
/ Installing packages (npm)...
```

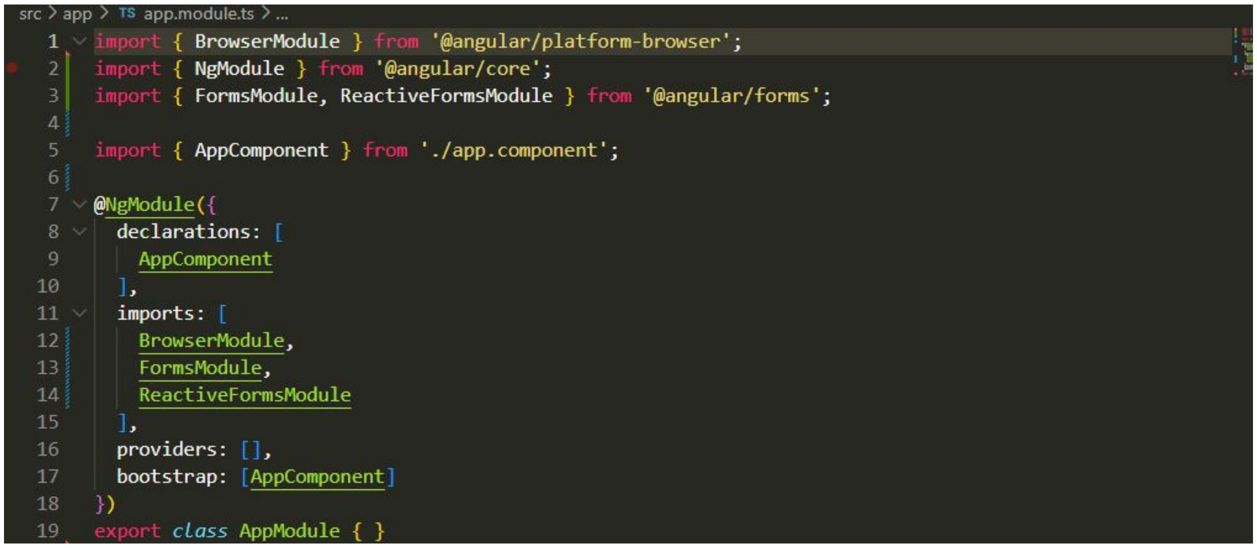
Once all packages have been added.

- This CLI will ask you "whether you would like to add Angular routing" Say Yes.
- It will ask "which stylesheet format you would like to use". Choose CSS.
- Now your project is ready Angular CLI will generate required files and folders along with NPM packages and routing too.

Step 4: Import FormsModule

If you want to create form in angular app then you need to import FormsModule from @angular/forms library. so let's add following code to app.module.ts file.

src/app/app.module.ts

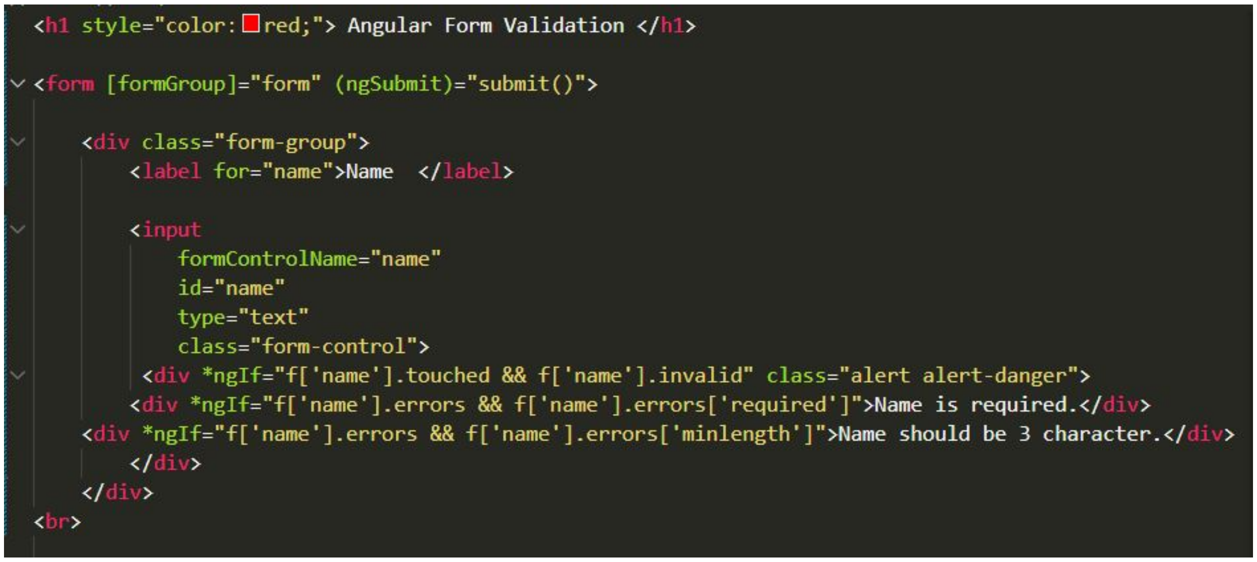


src > app > TS app.module.ts > ...

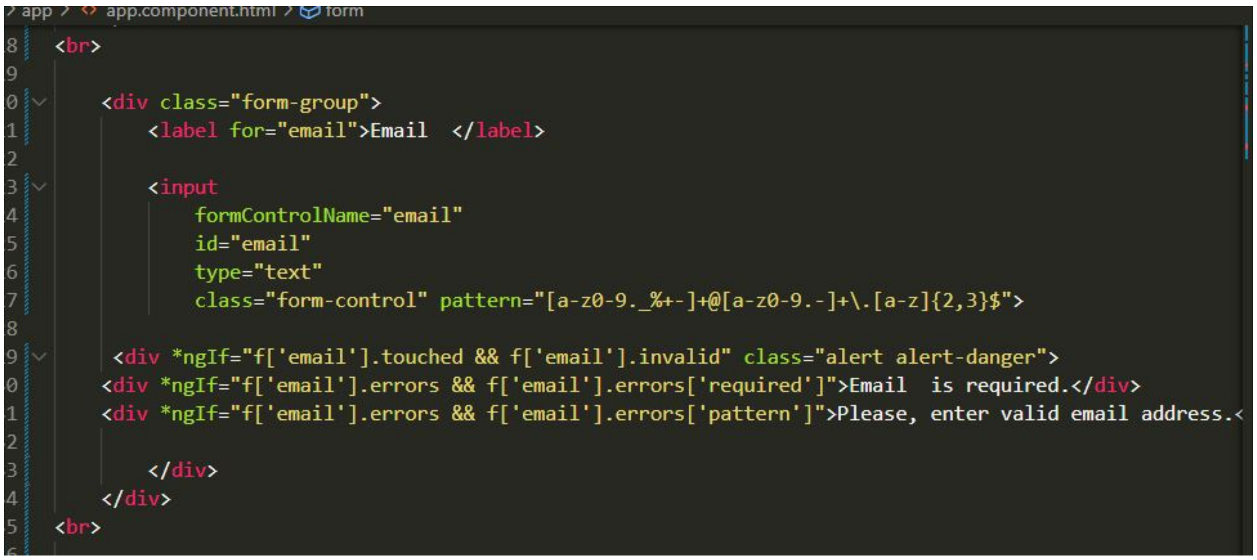
```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    ReactiveFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Step 5: Form with ngModel

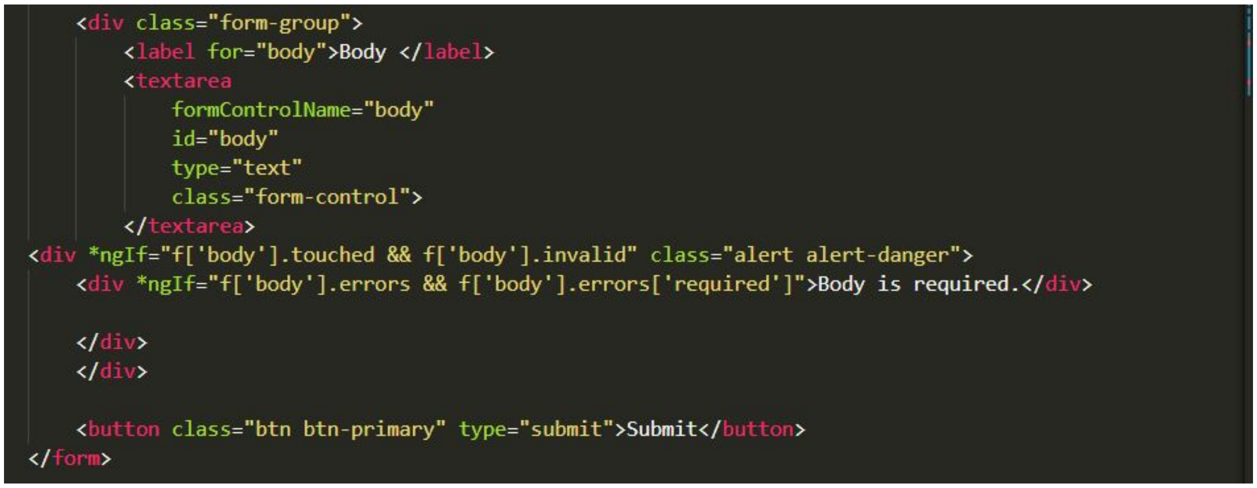
In this step, we will write code of html form with ngModel. so, add following code to app.component.html file



```
<h1 style="color: red;"> Angular Form Validation </h1>

<form [formGroup]="form" (ngSubmit)="submit()">

    <div class="form-group">
        <label for="name">Name  </label>

        <input
            formControlName="name"
            id="name"
            type="text"
            class="form-control">
         <div *ngIf="f['name'].touched && f['name'].invalid" class="alert alert-danger">
        <div *ngIf="f['name'].errors && f['name'].errors['required']">Name is required.</div>
    <div *ngIf="f['name'].errors && f['name'].errors['minlength']">Name should be 3 character.</div>
        </div>
    </div>
<br>
```



```
  <br>

      <div class="form-group">
          <label for="email">Email  </label>

          <input
              formControlName="email"
              id="email"
              type="text"
              class="form-control" pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,3}$">

       <div *ngIf="f['email'].touched && f['email'].invalid" class="alert alert-danger">
      <div *ngIf="f['email'].errors && f['email'].errors['required']">Email  is required.</div>
      <div *ngIf="f['email'].errors && f['email'].errors['pattern']">Please, enter valid email address.<

          </div>
      </div>
  <br>
```



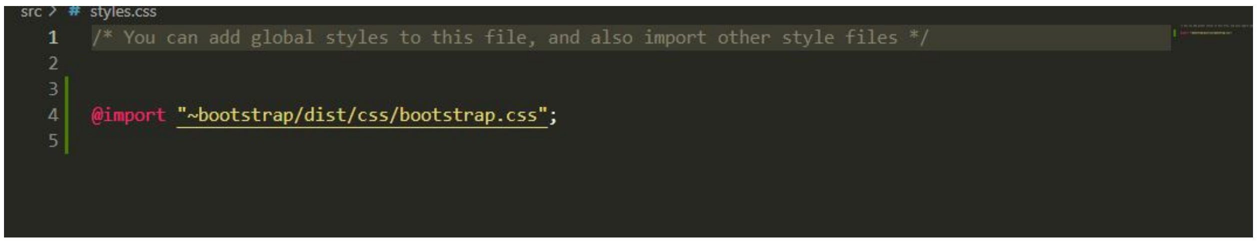
```
    <div class="form-group">
        <label for="body">Body </label>
        <textarea
            formControlName="body"
            id="body"
            type="text"
            class="form-control">
        </textarea>
<div *ngIf="f['body'].touched && f['body'].invalid" class="alert alert-danger">
    <div *ngIf="f['body'].errors && f['body'].errors['required']">Body is required.</div>

    </div>
    </div>

    <button class="btn btn-primary" type="submit">Submit</button>
</form>
```

In this solution, you need to just install bootstrap on your angular 12 and import css file to style.css file. this is only for css importing. so, you can run command bellow:

npm install bootstrap --save

You need to import your bootstrap css on style.css file as like bellow:

src/style.css



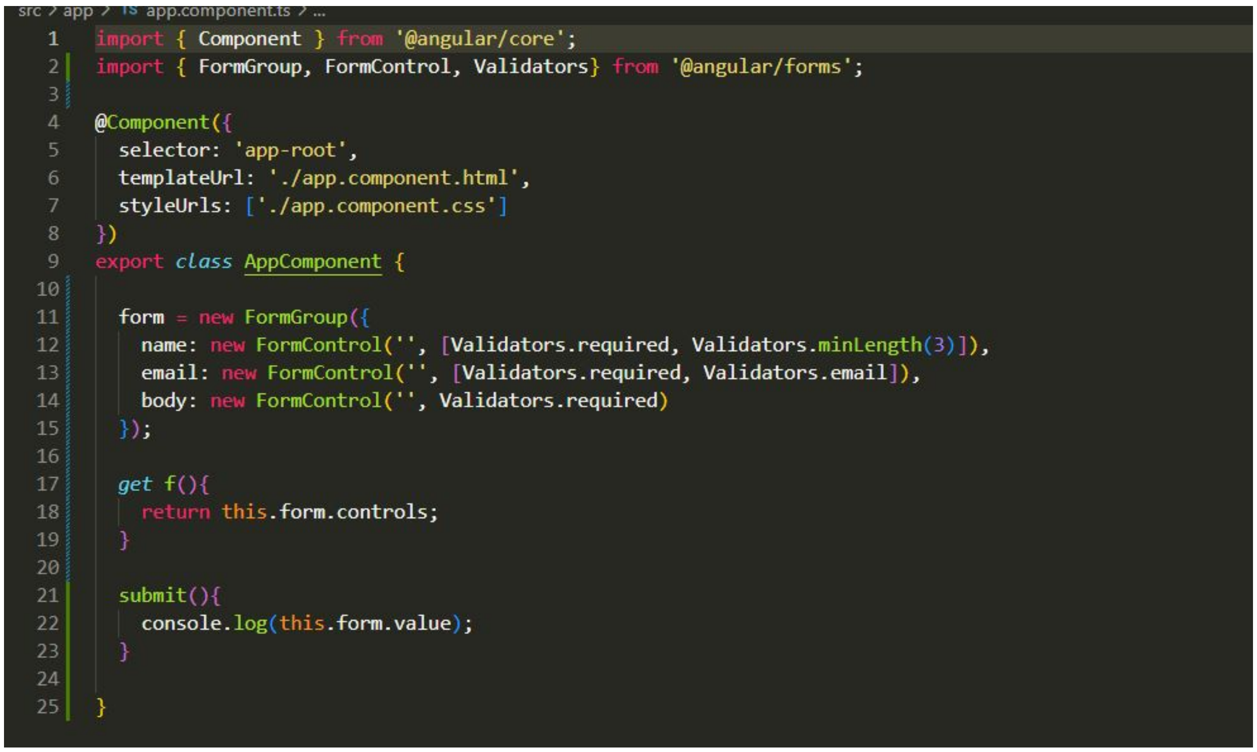
src > # styles.css

```css
/* You can add global styles to this file, and also import other style files */


@import "~bootstrap/dist/css/bootstrap.css";

```

Step 6: updated Ts File

In ts file. we will write submit() and get all input fields values. so let's add following code to app.component.ts file.

src/app/app.component.ts



src > app > TS app.component.ts > ...

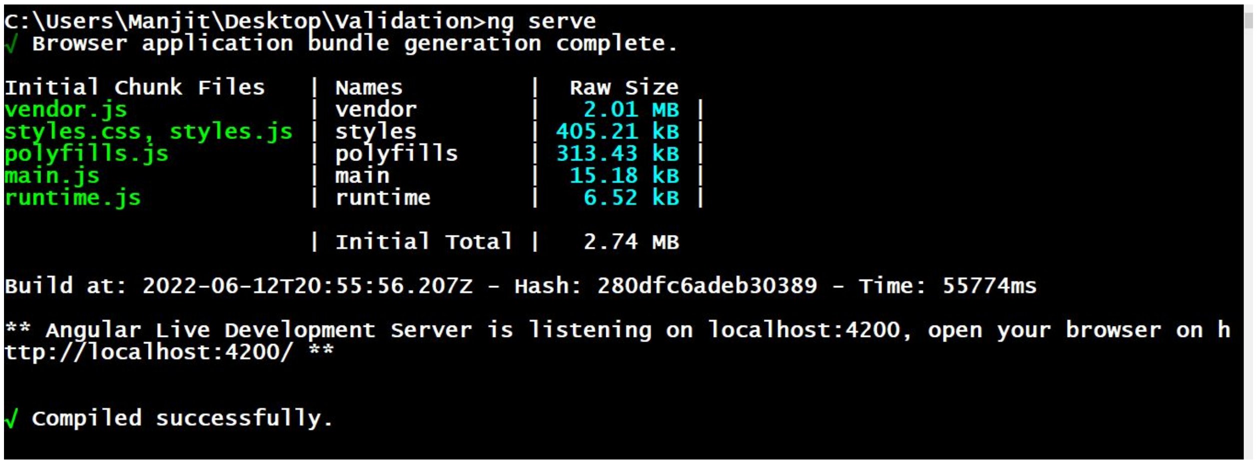
```typescript
import { Component } from '@angular/core';
import { FormGroup, FormControl, Validators} from '@angular/forms';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  form = new FormGroup({
    name: new FormControl('', [Validators.required, Validators.minLength(3)]),
    email: new FormControl('', [Validators.required, Validators.email]),
    body: new FormControl('', Validators.required)
  });

  get f(){
    return this.form.controls;
  }

  submit(){
    console.log(this.form.value);
  }

}
```

Step 7: Run your application

The system proceeds to generate the environment for your Angular application.

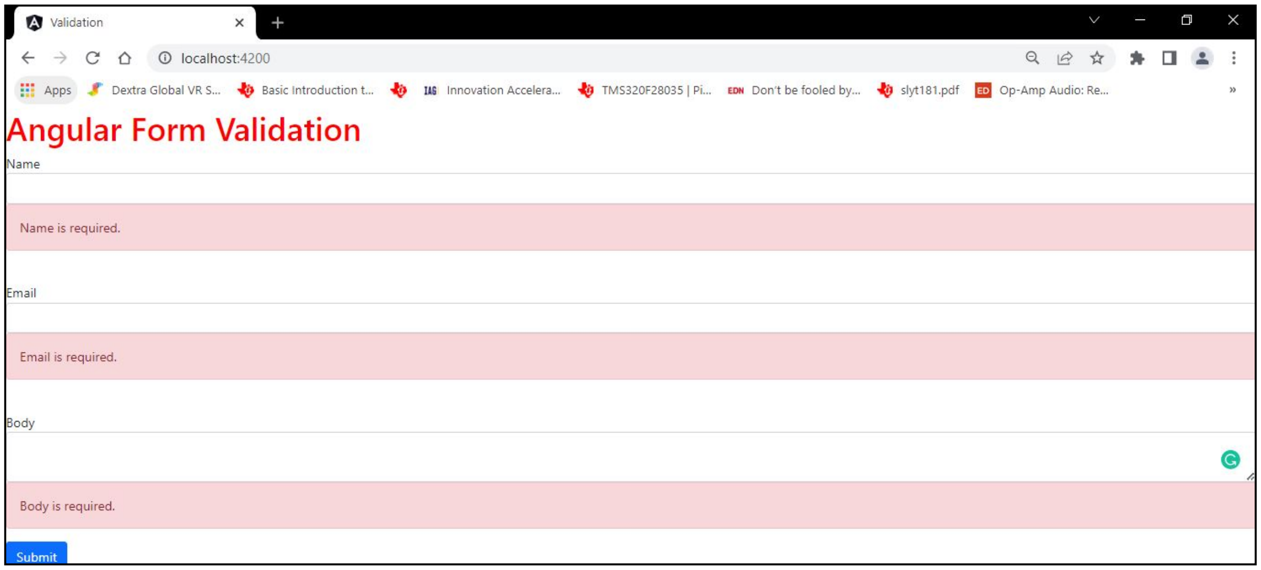
Keep the Windows Command Prompt running. You can continue using the terminal, and the changes you make will be reflected in the application within your browser.

Use any browser to access your Angular local development server on localhost:4200:



```
C:\Users\Manjit\Desktop\Validation>ng serve
√ Browser application bundle generation complete.

Initial Chunk Files     | Names          |  Raw Size
vendor.js               | vendor         |   2.01 MB |
styles.css, styles.js   | styles         | 405.21 kB |
polyfills.js            | polyfills      | 313.43 kB |
main.js                 | main           |  15.18 kB |
runtime.js              | runtime        |   6.52 kB |

                        | Initial Total |   2.74 MB

Build at: 2022-06-12T20:55:56.207Z - Hash: 280dfc6adeb30389 - Time: 55774ms

** Angular Live Development Server is listening on localhost:4200, open your browser on h
ttp://localhost:4200/ **


√ Compiled successfully.
```

Output/Results snippet:



Angular Form Validation

Name

Name is required.

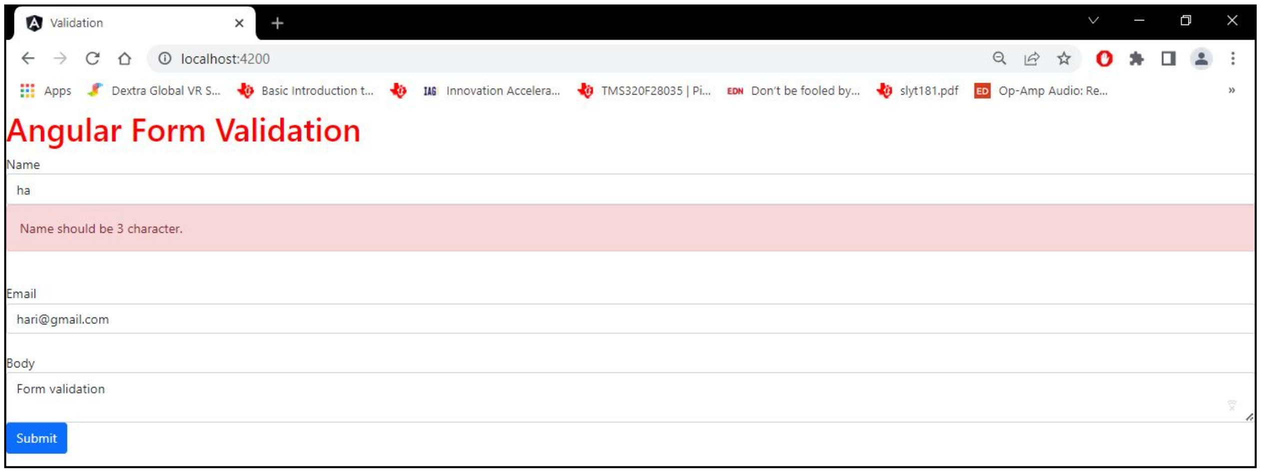
Email

Email is required.

Body

Body is required.

Submit


Validation

localhost:4200

Apps   Dextra Global VR S...   Basic Introduction t...   Innovation Accelera...   TMS320F28035 | Pi...   Don't be fooled by...   slyt181.pdf   Op-Amp Audio: Re...

Angular Form Validation

Name

ha

Name should be 3 character.

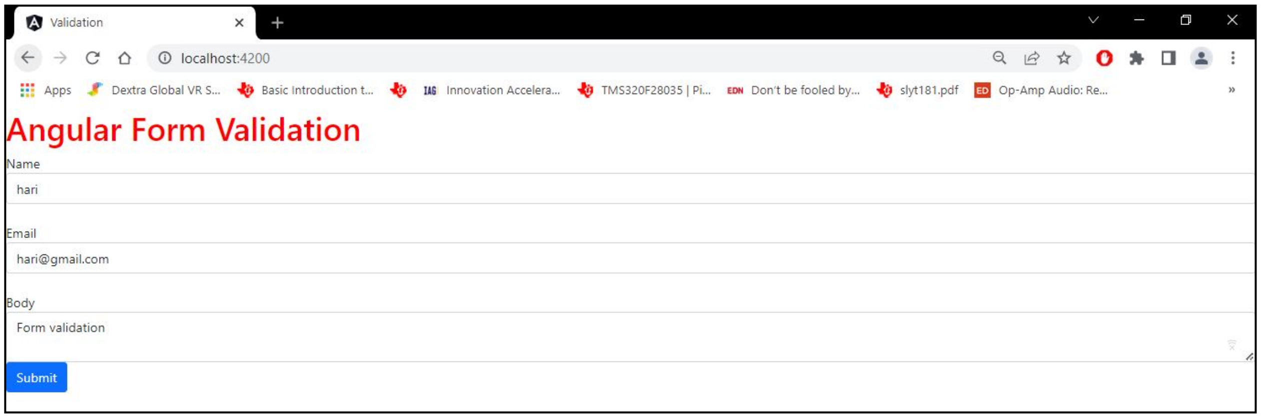
Email

hari@gmail.com

Body

Form validation

Submit


Validation

localhost:4200

Apps   Dextra Global VR S...   Basic Introduction t...   Innovation Accelera...   TMS320F28035 | Pi...   Don't be fooled by...   slyt181.pdf   Op-Amp Audio: Re...

Angular Form Validation

Name

hari

Email

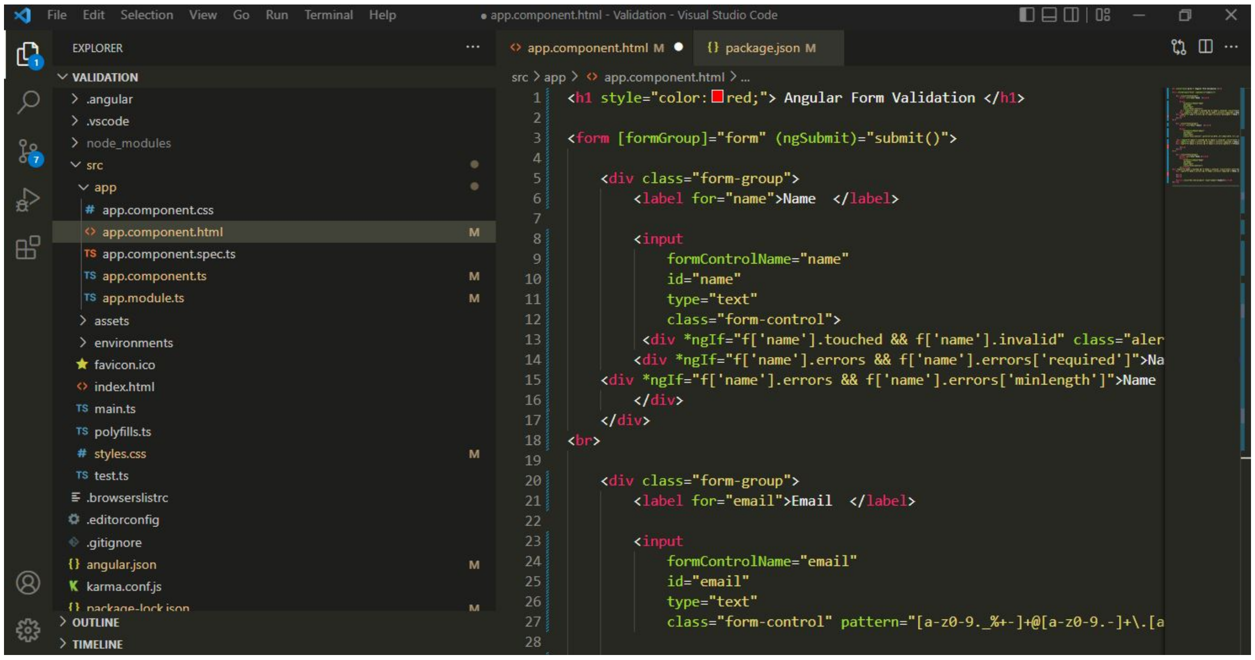
hari@gmail.com

Body

Form validation

Submit

Project Directory:



```
<h1 style="color:red;"> Angular Form Validation </h1>

<form [formGroup]="form" (ngSubmit)="submit()">

    <div class="form-group">
        <label for="name">Name  </label>

        <input
            formControlName="name"
            id="name"
            type="text"
            class="form-control">
         <div *ngIf="f['name'].touched && f['name'].invalid" class="aler
        <div *ngIf="f['name'].errors && f['name'].errors['required']">Na
    <div *ngIf="f['name'].errors && f['name'].errors['minlength']">Name
        </div>
    </div>
<br>

    <div class="form-group">
        <label for="email">Email  </label>

        <input
            formControlName="email"
            id="email"
            type="text"
            class="form-control" pattern="[a-z0-9._%+-]+@[a-z0-9.-]+\.[a

```

References:

- https://www.itsolutionstuff.com/post/angular-12-reactive-forms-validation-exampleexample.html

Learning Outcome

Able to develop the real time scenarios based on Node JS applications.

1. Create an Angular application which can create Captcha.
2. Create Responsive Web Application using Angular.

Activity 1

Aim: Create an Angular application which can create Captcha.

Learning outcome: Able to develop the real time scenarios based on Node JS applications.

Duration: 5 hour

List of Hardware/Software requirements:

1. Git
2. Node.js and npm
3. Angular CLI
4. IDE (e.g. Visual Studio Code)

Code/Program/Procedure (with comments):

Create the Angular application

1. Let's create the application with the Angular base structure using the @angular/cli with the route file and the SCSS style format.

```
ng new angular-recaptcha-v2

? Would you like to add Angular routing? Yes

? Which stylesheet format would you like to use? SCSS   [ https://sass-
lang.com/documentation/syntax#scss                ]

CREATE angular-recaptcha-v2/README.md (1064 bytes)

CREATE angular-recaptcha-v2/.editorconfig (274 bytes)

CREATE angular-recaptcha-v2/.gitignore (604 bytes)

CREATE angular-recaptcha-v2/angular.json (3291 bytes)

CREATE angular-recaptcha-v2/package.json (1082 bytes)

CREATE angular-recaptcha-v2/tsconfig.json (783 bytes)
CREATE angular-recaptcha-v2/.browserslistrc (703 bytes)

CREATE angular-recaptcha-v2/karma.conf.js (1437 bytes)

CREATE angular-recaptcha-v2/tsconfig.app.json (287 bytes)

CREATE angular-recaptcha-v2/tsconfig.spec.json (333 bytes)

CREATE angular-recaptcha-v2/src/favicon.ico (948 bytes)

CREATE angular-recaptcha-v2/src/index.html (304 bytes)

CREATE angular-recaptcha-v2/src/main.ts (372 bytes)

CREATE angular-recaptcha-v2/src/polyfills.ts (2820 bytes)

CREATE angular-recaptcha-v2/src/styles.scss (80 bytes)

CREATE angular-recaptcha-v2/src/test.ts (788 bytes)

CREATE angular-recaptcha-v2/src/assets/.gitkeep (0 bytes)

CREATE angular-recaptcha-v2/src/environments/environment.prod.ts (51 bytes)

CREATE angular-recaptcha-v2/src/environments/environment.ts (658 bytes)

CREATE angular-recaptcha-v2/src/app/app-routing.module.ts (245 bytes)

CREATE angular-recaptcha-v2/src/app/app.module.ts (393 bytes)

CREATE angular-recaptcha-v2/src/app/app.component.scss (0 bytes)

CREATE angular-recaptcha-v2/src/app/app.component.html (24617 bytes)

CREATE angular-recaptcha-v2/src/app/app.component.spec.ts (1115 bytes)

CREATE angular-recaptcha-v2/src/app/app.component.ts (225 bytes)

✓ Packages installed successfully.
```

2. Install and configure the Bootstrap CSS framework. Do steps 2 and 3 of the post Adding the Bootstrap CSS framework to an Angular application.

3. Configure the siteKey variable with the Google reCAPTCHA key in the src/environments/environment.ts and src/environments/environment.prod.ts files as below.

```
recaptcha: {

  siteKey: '6LfKNi0cAAAAACeYwFRY9_d_qjGhpiwYUo5gNW5-',

},
```

2. Install the ng-recaptcha library.

```
npm install ng-recaptcha
```

3. Import the FormsModule, RecaptchaFormsModule, RecaptchaModule modules. Configure the Google reCAPTCHA key. Change the app.module.ts file and add the lines as below.

```
import { FormsModule } from '@angular/forms';

import { RECAPTCHA_SETTINGS, RecaptchaFormsModule, RecaptchaModule,
RecaptchaSettings } from 'ng-recaptcha';

import { environment } from '../environments/environment';

imports: [

  BrowserModule,

  FormsModule,

  AppRoutingModule,

  RecaptchaModule,

  RecaptchaFormsModule,

],

providers: [

  {
    provide: RECAPTCHA_SETTINGS,

    useValue: {

      siteKey: environment.recaptcha.siteKey,

    } as RecaptchaSettings,

  },

],
```

4. Remove the contents of the AppComponent class from the src/app/app.component.ts file. Import the NgForm component and create the send method as below.

```
import { Component } from '@angular/core';

import { NgForm } from '@angular/forms';

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.scss'],

})

export class AppComponent {


  token: string|undefined;


  constructor() {

    this.token = undefined;

  }
  public send(form: NgForm): void {

    if (form.invalid) {

      for (const control of Object.keys(form.controls)) {

        form.controls[control].markAsTouched();

      }

      return;

    }


    console.debug(`Token [${this.token}] generated`);

  }


}
```

5. Remove the contents of the src/app/app.component.html file. Add the re-captcha component as below.

```
<div class="container-fluid py-3">

  <h1>Angular reCAPTCHA v2</h1>

  <form #form="ngForm">

    <div class="row mt-3">

      <div class="col-sm-12 mb-2">

        <re-captcha id="recaptcha" name="recaptcha" #recaptcha="ngModel"
[(ngModel)]="token" required [class.is-invalid]="recaptcha.invalid && (recaptcha.dirty ||
recaptcha.touched)"></re-captcha>
```

```
        <div *ngIf="recaptcha.invalid && (recaptcha.dirty || recaptcha.touched)"
class="invalid-feedback">

          <div *ngIf="recaptcha.errors?.['required']">This field is required.</div>

        </div>

      </div>

    </div>

    <div class="row">

      <div class="col-sm-12 mb-2">

        <button type="button" class="btn btn-sm btn-primary"
(click)="send(form)">Send</button>

      </div>

    </div>

  </form>

</div>
```

6. Add the style in the src/app/app.component.scss file as below.

```
re-captcha.is-invalid > div {

  border: 1px solid #dc3545 !important;

  border-radius: 0.2rem;

}
```

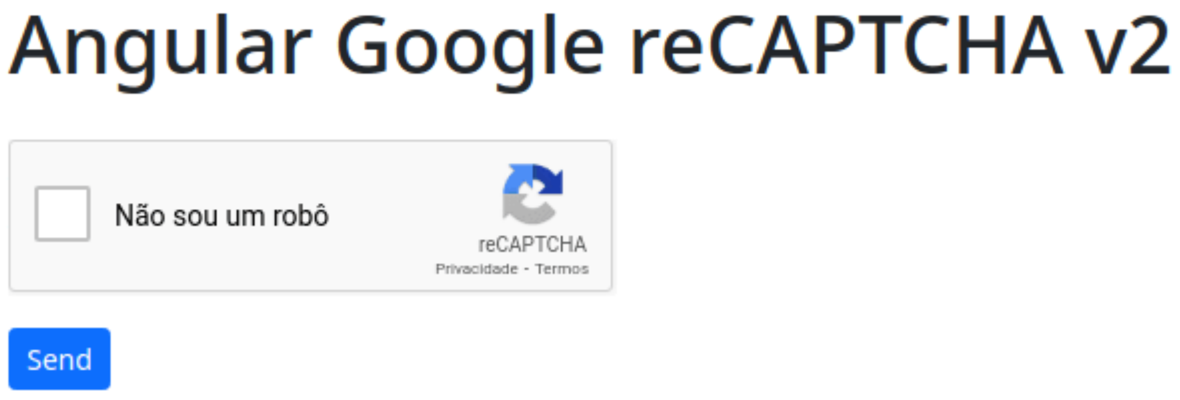
7. Run the application with the command below.

```
npm start

> angular-recaptcha-v2@0.0.0 start

> ng serve
```

```
✓ Browser application bundle generation complete.


Initial Chunk Files | Names         |      Size
vendor.js           | vendor        |   2.73 MB
styles.css          | styles        | 266.58 kB
polyfills.js        | polyfills     | 128.52 kB
scripts.js          | scripts       |  76.67 kB
main.js             | main          |  15.54 kB
runtime.js          | runtime       |   6.64 kB


| Initial Total |   3.21 MB
Build at: 2021-08-28T12:35:07.166Z - Hash: 0612b9d911a0acdf2b42 - Time: 10102ms
** Angular Live Development Server is listening on localhost:4200, open your browser
on http://localhost:4200/ **
✓ Compiled successfully.
```

8. Ready! Access the URL http://localhost:4200/ and check if the application is working. See the application working on GitHub Pages and Stackblitz.

Output/Results snippet:



References:

- https://dev.to/rodrigokamada/adding-the-google-recaptcha-v2-to-an-angular-application-1o7o
- https://github.com/rodrigokamada/angular-recaptcha-v2.

Activity 2

Aim: Create an Angular application which can create Captcha.

Learning outcome: Able to develop the real time scenarios based on Node JS applications.

Duration: 5 hours

List of Hardware/Software requirements:

1. Git
2. Node.js and npm
3. Angular CLI
4. IDE (e.g. Visual Studio Code)

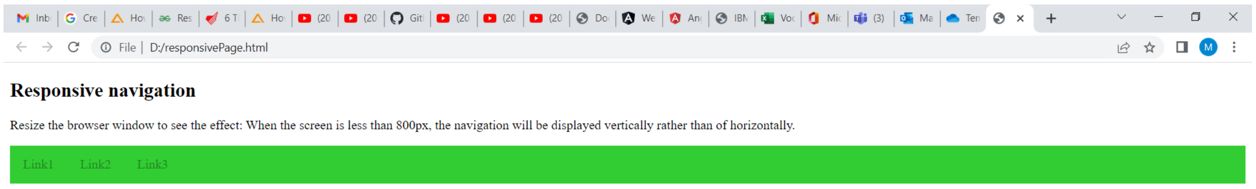
Code/Program/Procedure (with comments)

```
<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<meta name="viewport"

        content="width=device-width,

                        initial-scale=1">

<style>

* {

box-sizing: border-box;

}
```

```
/* Style the top navigation bar */

.topnav {

overflow: hidden;

background-color: #32CD32;

}



.topnav a {

float: left;

display: block;

color: #228B22;

text-align: center;

padding: 14px 16px;

text-decoration: none;

}


/* Change color on hover */

.topnav a:hover {

background-color: #C0C0C0;

color: black;

}
```

```
@media screen and (max-width: 800px) {
.topnav a {
        float: none;
        width: 100%;
}
}
</style>
</head>
<body>


<h2>Responsive navigation </h2>
<p>Resize the browser window to see the effect:
When the screen is less than 800px,
the navigation will be displayed vertically
rather than of horizontally.</p>
<div class="topnav">
<a href="#">Link1</a>
<a href="#">Link2</a>
<a href="#">Link3</a>
</div>
</body>
</html>
```

Output/Results snippet:

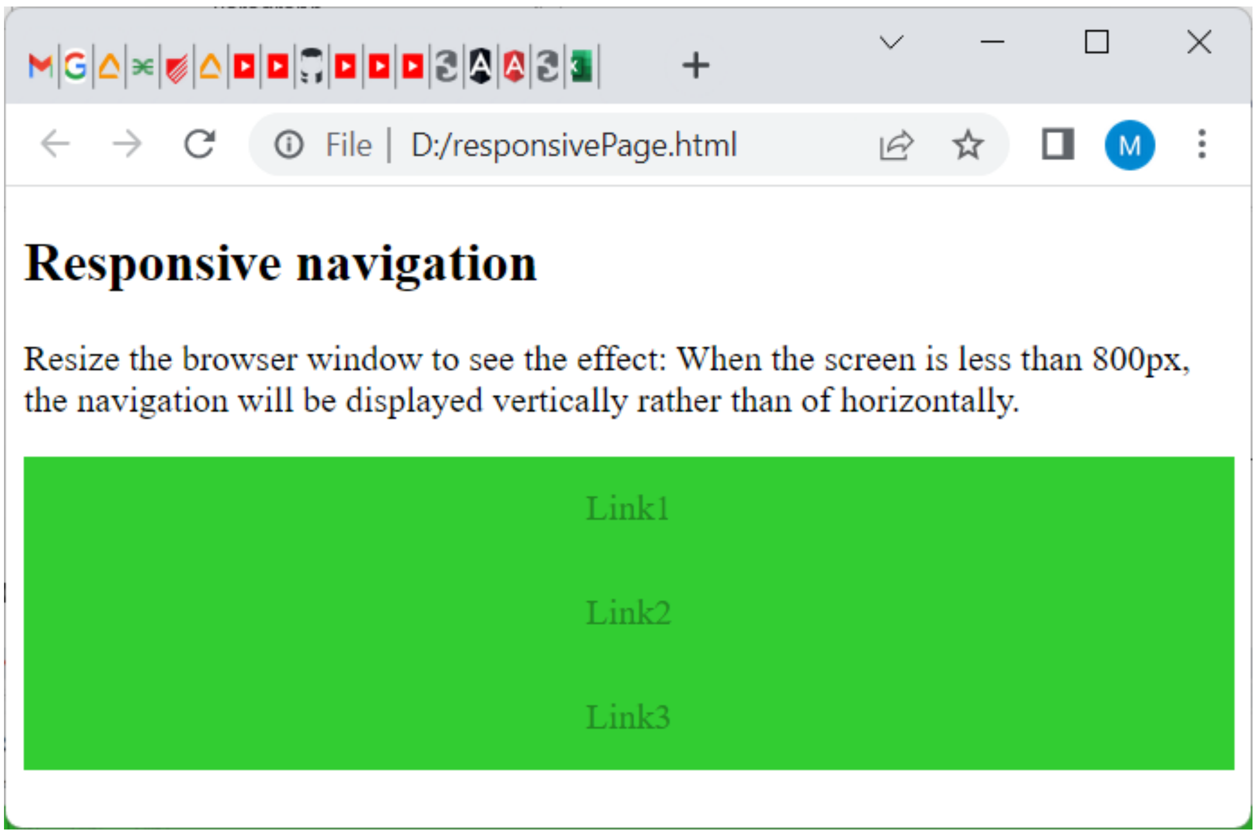


File | D:/responsivePage.html

Responsive navigation

Resize the browser window to see the effect: When the screen is less than 800px, the navigation will be displayed vertically rather than of horizontally.

Link1 Link2 Link3



File | D:/responsivePage.html

Responsive navigation

Resize the browser window to see the effect: When the screen is less than 800px, the navigation will be displayed vertically rather than of horizontally.

Link1

Link2

Link3

References:

- https://www.geeksforgeeks.org/responsive-page-in-angularjs/

Learning Outcome

After completing this module, the student should be **able to understand how to install Angular Js.**

To meet the learning outcome, a student has to complete the following activities

1. Create an Angular application using Node JS with complete templating system.
2. Create a basic Program with Node**.**

Activity 1

Aim: Create an Angular application using Node JS with complete templating system

Learning outcome: Able to develop the real time scenarios based on Node JS applications.

Duration: 2 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Windows OS / Linux OS - Ubuntu 18.04 LTS
2. NodeJS, ExpressJS, VS Code installed

Code/Program/Procedure (with comments):

Install Angular CLI

Angular provides many libraries and packages for application development. You can install libraries required for your application using Angular CLI (Command Line Interface). Angular CLI is also used to generate, build, run, and deploy Angular application.

To install the Angular CLI globally using NPM, open a terminal/command window, and enter the following command:

```
npm install -g @angular/cli@latest
```

Create Angular 2 Application

The following creates a new angular application named "FirstAngularApp" in the AngularApps folder.

```
D:\AngularApps> ng new FirstAngularApp
```

To open this project in VS Code, navigate to the project folder in the terminal/command window and type **code.**

```
D:\AngularApps\FirstAngularApp\>code .
```

Run Angular Application

Open the terminal in VS Code from menu Terminal -> New Terminal, and type ng serve -o command and press enter

Angular 2 Components

Angular Component = HTML Template + Component Class + Component Metadata

HTML Template

HTML template is nothing but a regular HTML code with additional Angular specific syntax to communicate with the component class.

Generate Angular Component using Angular CLI

Use the following CLI command to generate a component.

ng generate component greet

Now, open greet.component.ts file in VS Code, and you will see the following code.

```
import { Component, OnInit } from '@angular/core';

@Component({

  selector: 'app-greet',

  templateUrl: './greet.component.html',

  styleUrls: ['./greet.component.css']

})

export class GreetComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {
  }}
```

Now, let's add a property and method in the component class.

```
export class GreetComponent implements OnInit {

  constructor() { }

  ngOnInit(): void {

  }

  name: string = "Steve";

  greet(): void {

      alert("Hello " + this.name);

  };

}
```

Open greet.component.html file, remove existing code and add the following code.

```
<div>

    Enter Your Name: <input type="text" value={{name}} /> <br />

    <button (click)="greet()">Greet Me!</button>

</div>
```

In the above HTML template, we used name property in the {{ }} interpolation to display its value and greet() function as click event.

Bootstrapping Component

Now, it's time to load our component, but before that, we need to host our application and load the root component. This process is called bootstrapping.

We will load our greet component into the root component in two steps.

1. Declare a component in the root module.

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


import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { GreetComponent } from './greet/greet.component'; //import GreetComponent


@NgModule({
  declarations: [
    AppComponent,
    GreetComponent  // <- include GreetComponent in declarations
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
```

```
export class AppModule { }
```

2. Add component tag in the root HTML template.

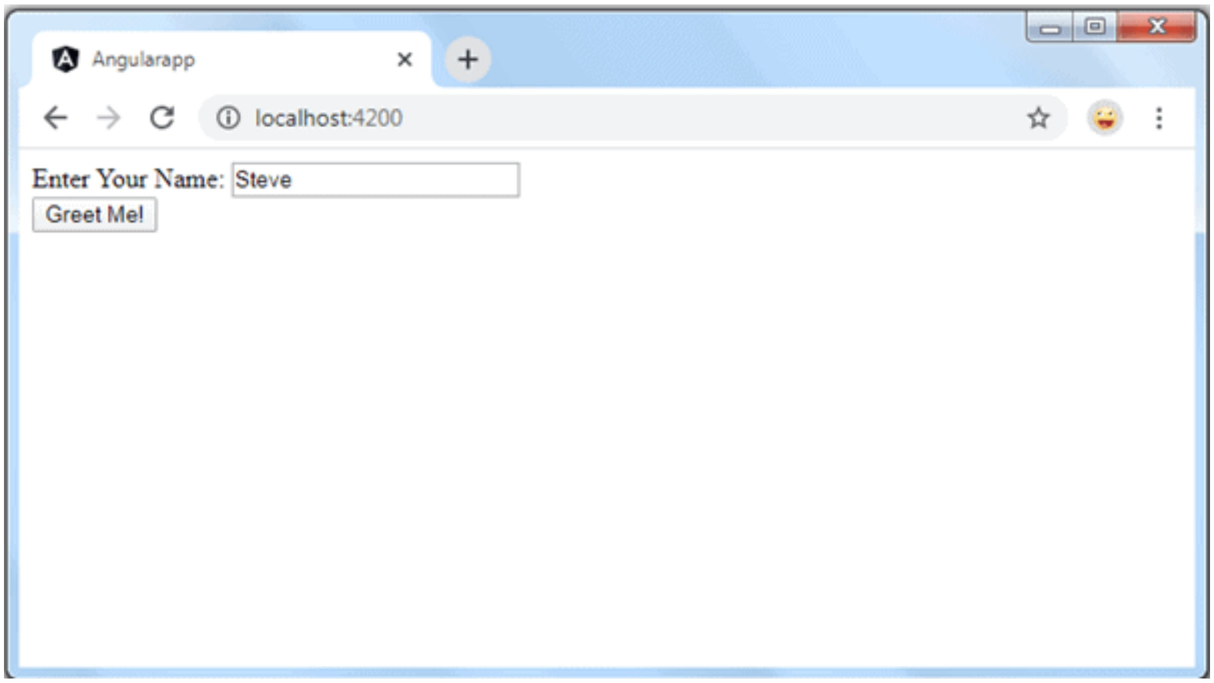
```
<div>
    <app-greet></app-greet>
</div>
```

Run the app by executing npm start command in the terminal of VS Code. After successful compilation, open the browser and enter http://localhost:4200.

We can also create a single component file greet.component.ts if the HTML code of a component is less. Use the template parameter in the @Component decorator to include HTML of the component. The following greet component gives the same result.

```
import { Component } from '@angular/core';

@Component({
    selector: "app-greet",
    template: `<div>
    Enter Your Name: <input type="text" value={{name}} /> <br/>
    <button (click)="greet()">Greet Me!</button>
    </div>`
})

export class GreetComponent {

    name: string = "Steve";
    greet(): void {
        alert("Hello " + this.name);
    };
}
```

Output/Results snippet:



Enter Your Name: Steve

References:

- https://www.tutorialsteacher.com/angular/create-angular-application

Activity 2

Aim: Create a basic Program with Node

Learning outcome: Able to develop the real time scenarios based on Node JS applications.

Duration: 5 hours

List of Hardware/Software requirements:

1. Git
2. Node.js and npm
3. Angular CLI
4. IDE (e.g., Visual Studio Code

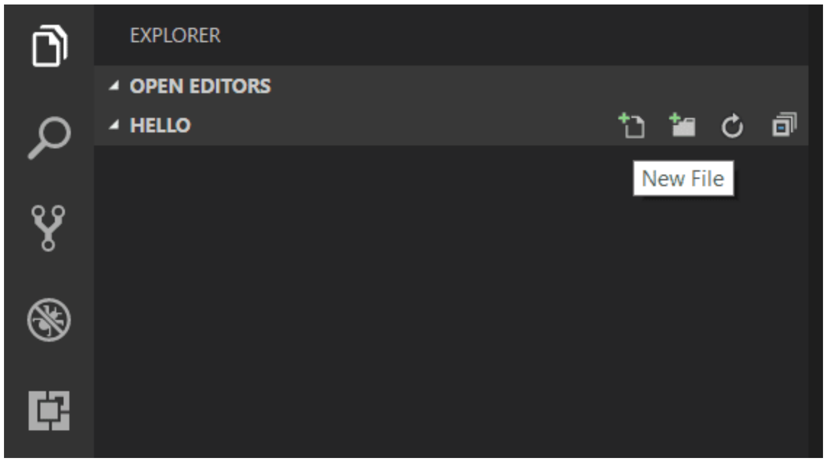
Code/Program/Procedure (with comments):

Let's get started by creating the simplest Node.js application, "Hello World".

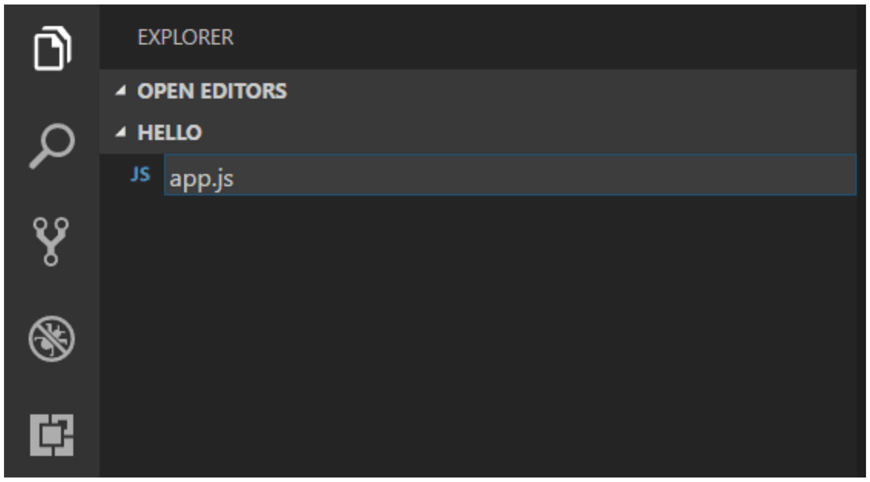
Create an empty folder called "hello", navigate into and open VS Code:

```
mkdir hello

cd hello

code .
```

From the File Explorer toolbar, press the New File button



And name the file app.js:

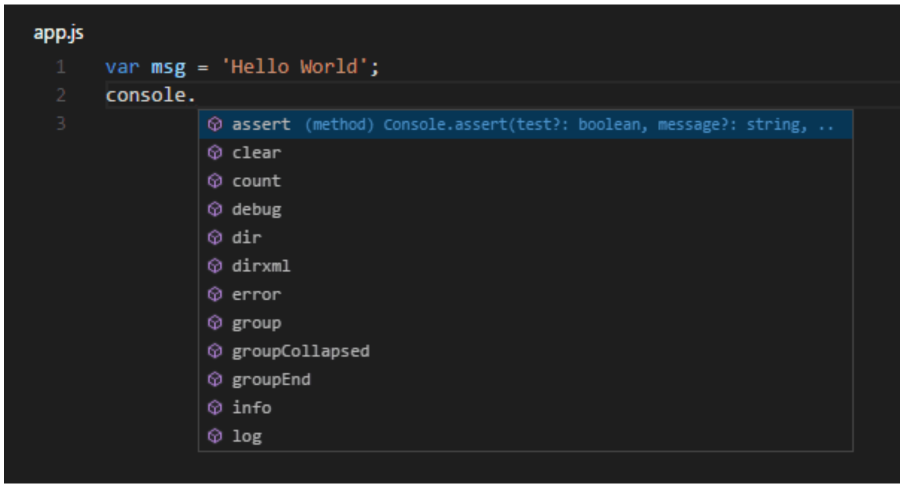


By using the .js file extension, VS Code interprets this file as JavaScript and will evaluate the contents with the JavaScript language service. Refer to the VS Code JavaScript language topic to learn more about JavaScript support.

Create a simple string variable in app.js and send the contents of the string to the console:

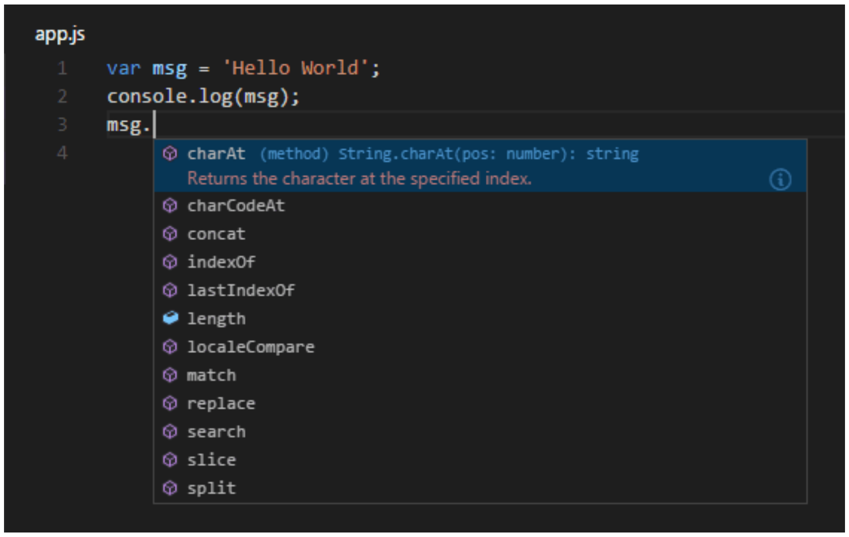
```
var msg = 'Hello World';

console.log(msg);
```

Note that when you typed console. IntelliSense on the console object was automatically presented to you.



```
app.js
var msg = 'Hello World';
console.
  assert  (method) Console.assert(test?: boolean, message?: string, ..
  clear
  count
  debug
  dir
  dirxml
  error
  group
  groupCollapsed
  groupEnd
  info
  log
```

Also notice that VS Code knows that msg is a string based on the initialization to 'Hello World'. If you type msg. you'll see IntelliSense showing all of the string functions available on msg.



```
app.js
var msg = 'Hello World';
console.log(msg);
msg.
  charAt  (method) String.charAt(pos: number): string
  Returns the character at the specified index.
  charCodeAt
  concat
  indexOf
  lastIndexOf
  length
  localeCompare
  match
  replace
  search
  slice
  split
```

After experimenting with IntelliSense, revert any extra changes from the source code example above and save the file (Ctrl+S).

Running Hello World#

It's simple to run app.js with Node.js. From a terminal, just type:

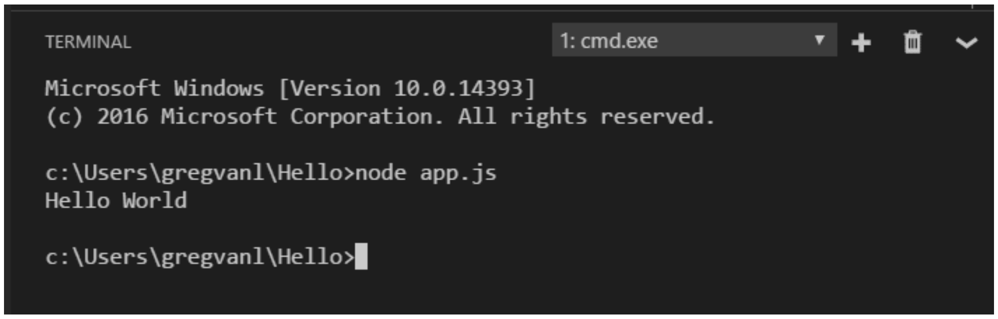
node app.js

You should see "Hello World" output to the terminal and then Node.js returns.

Integrated Terminal#

VS Code has an integrated terminal which you can use to run shell commands. You can run Node.js directly from there and avoid switching out of VS Code while running command-line tools.

View > Terminal (Ctrl+` with the backtick character) will open the integrated terminal and you can run node app.js there:



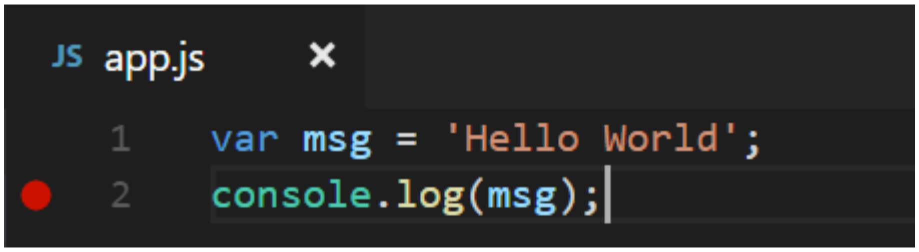
```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

c:\Users\gregvanl\Hello>node app.js
Hello World

c:\Users\gregvanl\Hello>
```

For this walkthrough, you can use either an external terminal or the VS Code integrated terminal for running the command-line tools.

Debugging Hello World#

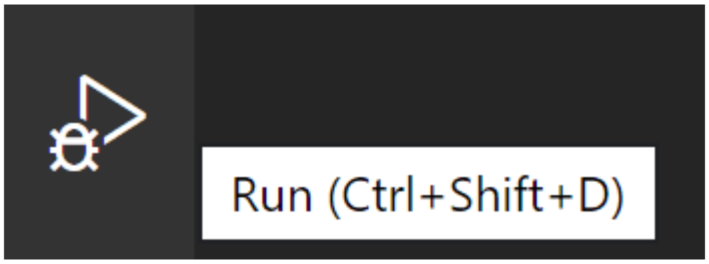
As mentioned in the introduction, VS Code ships with a debugger for Node.js applications. Let's try debugging our simple Hello World application.

To set a breakpoint in app.js, put the editor cursor on the first line and press F9 or click in the editor left gutter next to the line numbers. A red circle will appear in the gutter.

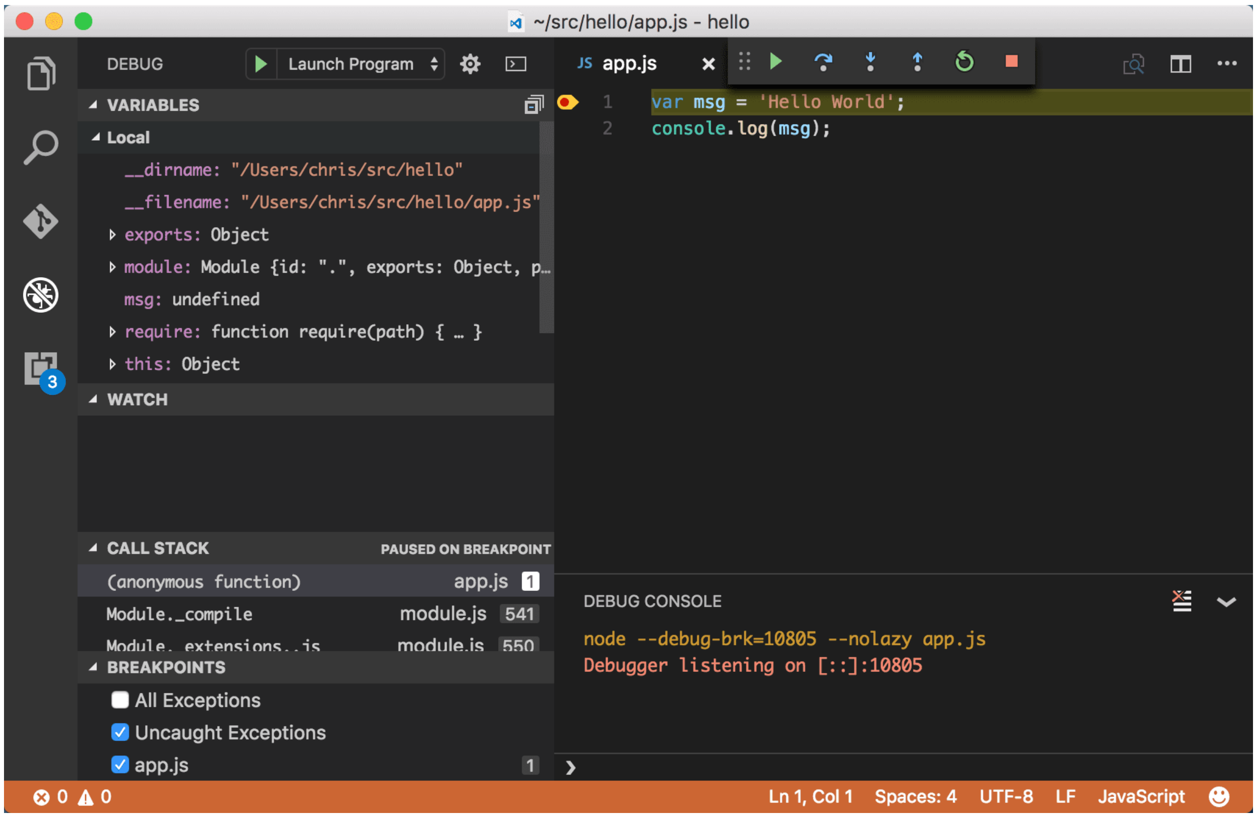


```
1  var msg = 'Hello World';
2  console.log(msg);
```

To start debugging, select the Run View in the Activity Bar:



You can now click Debug toolbar green arrow or press F5 to launch and debug "Hello World". Your breakpoint will be hit and you can view and step through the simple application. Notice that VS Code displays a different colored Status Bar to indicate it is in Debug mode and the DEBUG CONSOLE is displayed.



```
var msg = 'Hello World';
console.log(msg);
```

```
__dirname: "/Users/chris/src/hello"
__filename: "/Users/chris/src/hello/app.js"
exports: Object
module: Module {id: ".", exports: Object, p…
msg: undefined
require: function require(path) { … }
this: Object
```

```
(anonymous function)      app.js 1
Module._compile           module.js 541
Module._extensions..js    module.js 550
```

All Exceptions
Uncaught Exceptions
app.js 1

```
node --debug-brk=10805 --nolazy app.js
Debugger listening on [::]:10805
```

0 0 Ln 1, Col 1 Spaces: 4 UTF-8 LF JavaScript

Now that you've seen VS Code in action with "Hello World", the next section shows using VS Code with a full-stack Node.js web app.

An Express application#

Express is a very popular application framework for building and running Node.js applications. You can scaffold (create) a new Express application using the Express Generator tool. The Express Generator is shipped as an npm module and installed by using the npm command-line tool npm.

Install the Express Generator by running the following from a terminal:

```
npm install -g express-generator
```

The -g switch installs the Express Generator globally on your machine so you can run it from anywhere.

We can now scaffold a new Express application called myExpressApp by running:

```
express myExpressApp --view pug
```

This creates a new folder called myExpressApp with the contents of your application. The --view pug parameters tell the generator to use the pug template engine.

To install all of the application's dependencies (again shipped as npm modules), go to the new folder and execute npm install:

```
cd myExpressApp

npm install
```

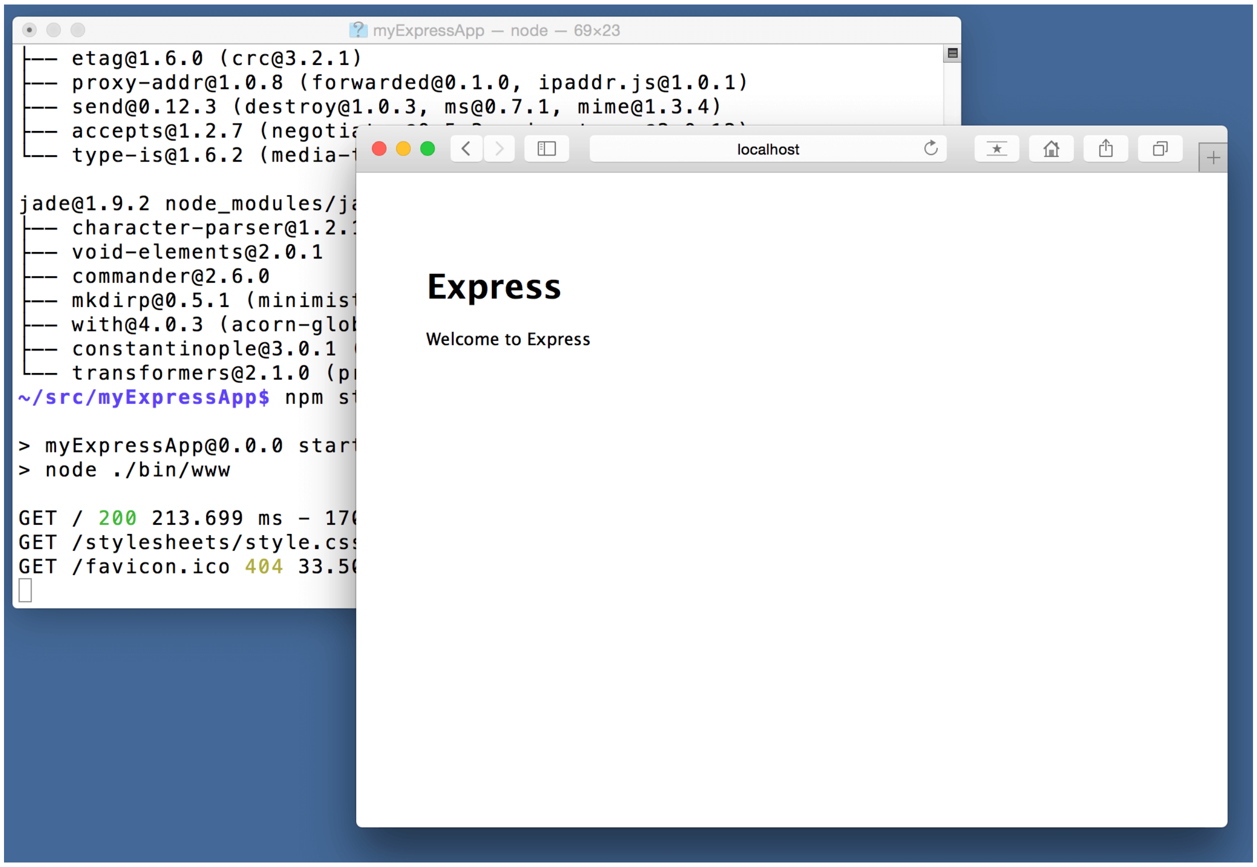
At this point, we should test that our application runs. The generated Express application has a package.json file which includes a start script to run node ./bin/www. This will start the Node.js application running.

From a terminal in the Express application folder, run:

```
npm start
```

The Node.js web server will start and you can browse to http://localhost:3000 to see the running application.

Your first Node Express App



```
├── etag@1.6.0 (crc@3.2.1)
├── proxy-addr@1.0.8 (forwarded@0.1.0, ipaddr.js@1.0.1)
├── send@0.12.3 (destroy@1.0.3, ms@0.7.1, mime@1.3.4)
├── accepts@1.2.7 (negotia
└── type-is@1.6.2 (media-t

jade@1.9.2 node_modules/ja
├── character-parser@1.2.1
├── void-elements@2.0.1
├── commander@2.6.0
├── mkdirp@0.5.1 (minimist
├── with@4.0.3 (acorn-glob
├── constantinople@3.0.1 (
└── transformers@2.1.0 (pr
~/src/myExpressApp$ npm st

> myExpressApp@0.0.0 start
> node ./bin/www

GET / 200 213.699 ms - 170
GET /stylesheets/style.css
GET /favicon.ico 404 33.50
```

Express

Welcome to Express

Great code editing#

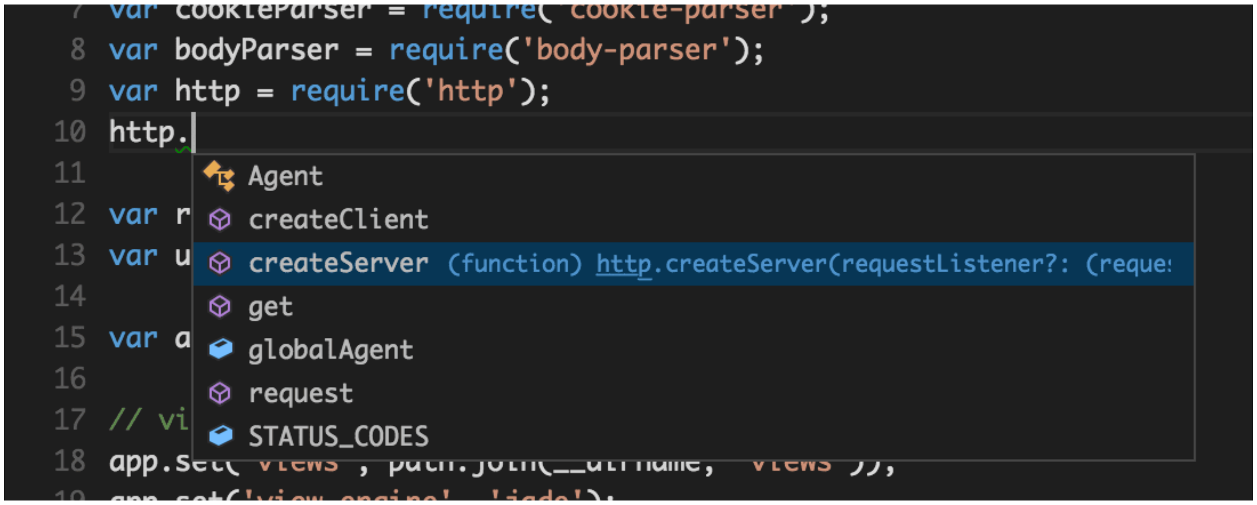
Close the browser and from a terminal in the myExpressApp folder, stop the Node.js server by pressing CTRL+C.

Now launch VS Code:

```
code .
```

The Node.js and Express documentation does a great job explaining how to build rich applications using the platform and framework. Visual Studio Code will make you more productive in developing these types of applications by providing great code editing and navigation experiences.

Open the file app.js and hover over the Node.js global object __dirname. Notice how VS Code understands that __dirname is a string. Even more interesting, you can get full IntelliSense against the Node.js framework. For example, you can require http and get full IntelliSense against the http class as you type in Visual Studio Code.

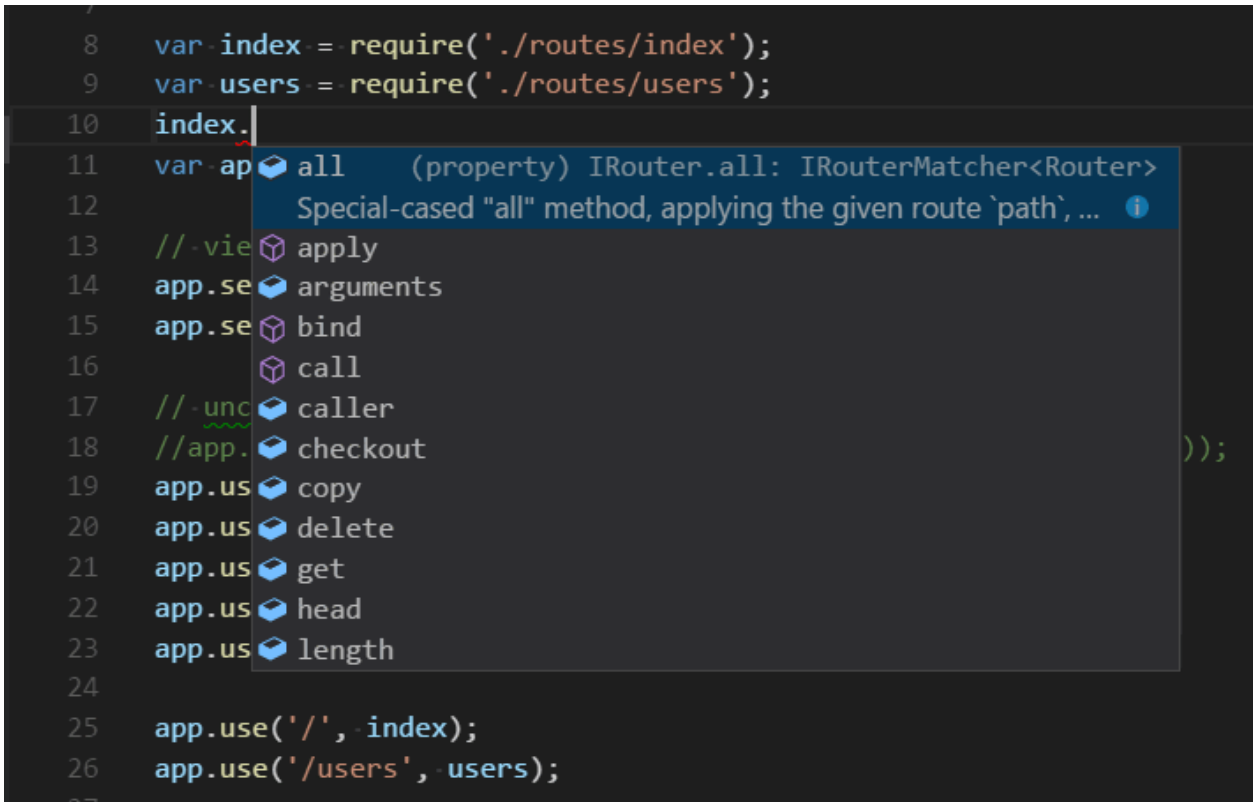


```
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');
var http = require('http');
http.

var r
var u

var a

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

```
Agent
createClient
createServer  (function) http.createServer(requestListener?: (reque
get
globalAgent
request
STATUS_CODES
```

VS Code uses TypeScript type declaration (typings) files (for example node.d.ts) to provide metadata to VS Code about the JavaScript based frameworks you are consuming in your application. Type declaration files are written in TypeScript so they can express the data types of parameters and functions, allowing VS Code to provide a rich IntelliSense experience. Thanks to a feature called Automatic Type Acquisition, you do not have to worry about downloading these type declaration files, VS Code will install them automatically for you.

You can also write code that references modules in other files. For example, in app.js we require the ./routes/index module, which exports an Express.Router class. If you bring up IntelliSense on index, you can see the shape of the Router class.



```
var index = require('./routes/index');
var users = require('./routes/users');
index.
var ap
```

all (property) IRouter.all: IRouterMatcher<Router>
Special-cased "all" method, applying the given route `path`, ...
apply
arguments
bind
call
caller
checkout
copy
delete
get
head
length

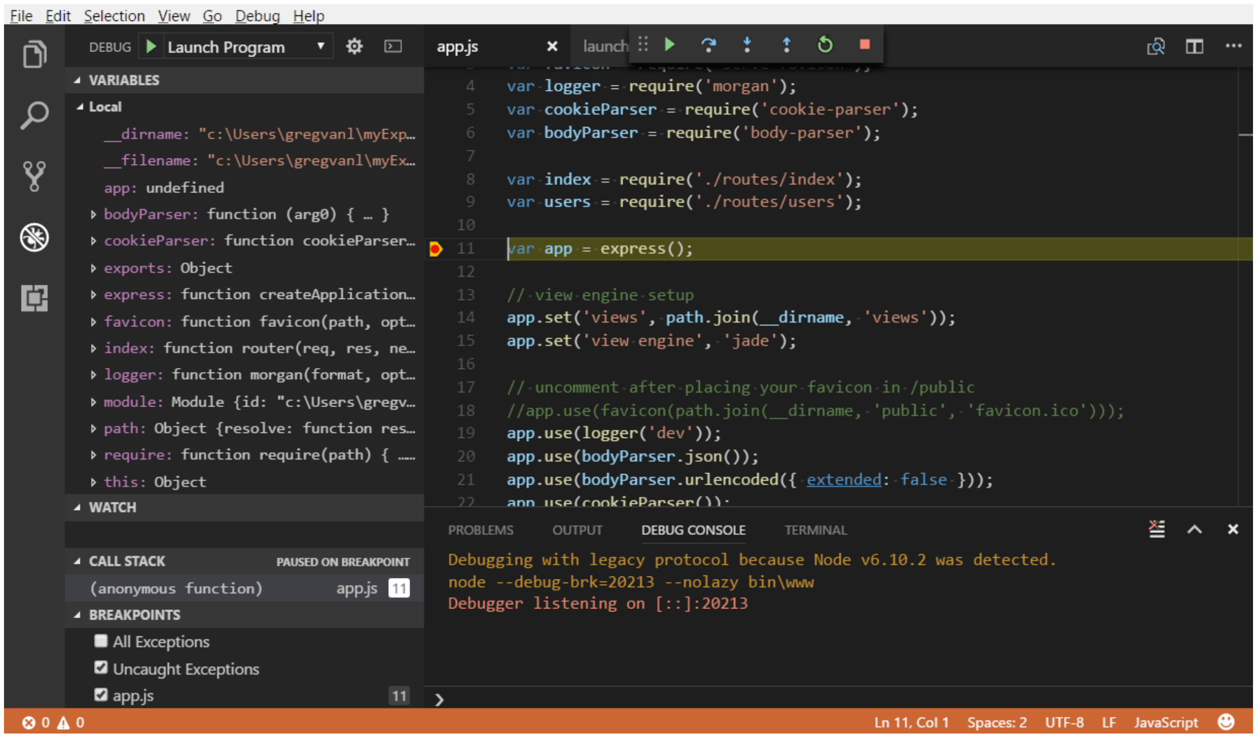
```
app.use('/', index);
app.use('/users', users);
```

Debug your Express app#

You will need to create a debugger configuration file launch.json for your Express application. Click on the Run icon in the Activity Bar and then the Configure gear icon at the top of the Run view to create a default launch.json file. Select the Node.js environment by ensuring that the type property in configurations is set to "node". When the file is first created, VS Code will look in package.json for a start script and will use that value as the program (which in this case is "${workspaceFolder}\\bin\\www) for the Launch Program configuration.

```
{

  "version": "0.2.0",

  "configurations": [

    {
      "type": "node",
      "request": "launch",
      "name": "Launch Program",
      "program": "${workspaceFolder}\\bin\\www"
    }
  ]
}
```

Save the new file and make sure Launch Program is selected in the configuration dropdown at the top of the Run view. Open app.js and set a breakpoint near the top of the file where the Express app object is created by clicking in the gutter to the left of the line number. Press F5 to start debugging the application. VS Code will start the server in a new terminal and hit the breakpoint we set. From there you can inspect variables, create watches, and step through your code.



File Edit Selection View Go Debug Help
DEBUG Launch Program
app.js
launch
VARIABLES
Local
WATCH
CALL STACK PAUSED ON BREAKPOINT
BREAKPOINTS
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
__dirname: "c:\Users\gregvanl\myExp…
__filename: "c:\Users\gregvanl\myEx…
app: undefined
bodyParser: function (arg0) { … }
cookieParser: function cookieParser…
exports: Object
express: function createApplication…
favicon: function favicon(path, opt…
index: function router(req, res, ne…
logger: function morgan(format, opt…
module: Module {id: "c:\Users\gregv…
path: Object {resolve: function res…
require: function require(path) { ….
this: Object
```

```
(anonymous function)    app.js 11
```

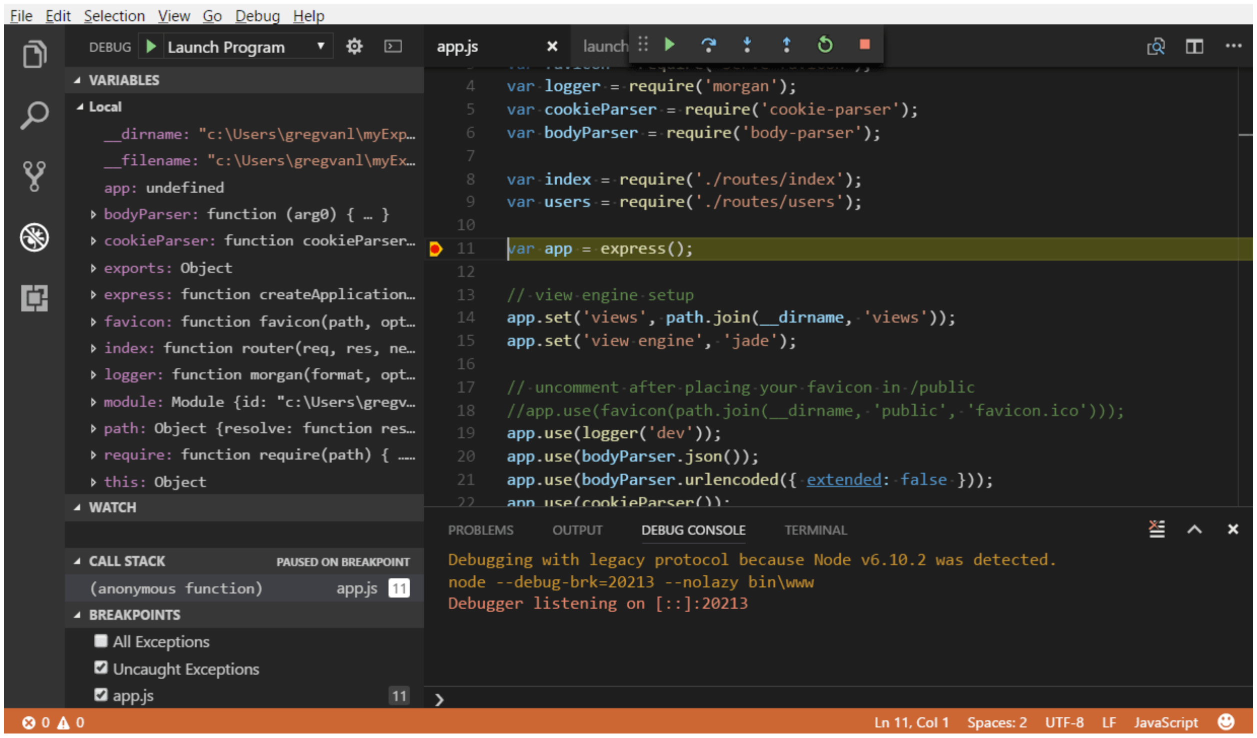
```
☐ All Exceptions
☑ Uncaught Exceptions
☑ app.js    11
```

```
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var index = require('./routes/index');
var users = require('./routes/users');

var app = express();

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

// uncomment after placing your favicon in /public
//app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
```

```
Debugging with legacy protocol because Node v6.10.2 was detected.
node --debug-brk=20213 --nolazy bin\www
Debugger listening on [::]:20213
```

```
⊗ 0 ⚠ 0        Ln 11, Col 1   Spaces: 2   UTF-8   LF   JavaScript
```

Output/Results snippet:



```
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var index = require('./routes/index');
var users = require('./routes/users');

var app = express();

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

// uncomment after placing your favicon in /public
//app.use(favicon(path.join(__dirname, 'public', 'favicon.ico')));
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(cookieParser());
```

```
VARIABLES
  Local
    __dirname: "c:\Users\gregvanl\myExp…
    __filename: "c:\Users\gregvanl\myEx…
    app: undefined
    bodyParser: function (arg0) { … }
    cookieParser: function cookieParser…
    exports: Object
    express: function createApplication…
    favicon: function favicon(path, opt…
    index: function router(req, res, ne…
    logger: function morgan(format, opt…
    module: Module {id: "c:\Users\gregv…
    path: Object {resolve: function res…
    require: function require(path) { ….
    this: Object
WATCH
CALL STACK                PAUSED ON BREAKPOINT
  (anonymous function)    app.js 11
BREAKPOINTS
  ☐ All Exceptions
  ☑ Uncaught Exceptions
  ☑ app.js                11
```

```
Debugging with legacy protocol because Node v6.10.2 was detected.
node --debug-brk=20213 --nolazy bin\www
Debugger listening on [::]:20213
```

File Edit Selection View Go Debug Help
DEBUG Launch Program
app.js
launch
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Ln 11, Col 1 Spaces: 2 UTF-8 LF JavaScript

References:

1. https://code.visualstudio.com/docs/nodejs/nodejs-tutorial#_great-code-editing

Learning Outcome

After completing this module, the student should be **Able to develop the real time scenarios based on Node JS applications.**

To meet the learning outcome, a student has to complete the following activities

3. Create Module (Function) & export, import using Node
4. Create a Web Application using Express JS
5. Create RestAPI using express JS

Activity 1

Aim: Create Module (Function) & export, import using Node

Learning outcome: Able to develop the real time scenarios based on Node JS applications.

Duration: 3 Hour

List of Hardware/Software requirements:

1. Operating System – Windows 10/11 or Linux
2. Command Prompt/Power Shell
3. Text Editor – Notepad / IDE – Visual Studio Code/Any IDE
4. NodeJS

Code/Program/Procedure (with comments):

1. Creating a Module: Modules are created in Node.js are JavaScript files. Every time a new file with .js extension is created, it becomes a module.

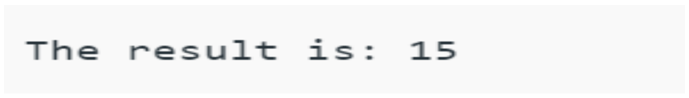
2. Create a file func.js

```
function add (x, y) {
  return x + y;
}
function subtract (x, y) {
  return x - y;
}
// Adding the code below to allow importing
// the functions in other files
module.exports = { add }
```

3. Create a file **main.js** which will import the **func.js** module.

```
// Importing the func.js module
// The ./ says that the func module is in the same directory as the main.js file
const f = require('./func');
// Require returns an object with add() and stores it in the f variable
// which is used to invoke the required
const result = f.add(10, 5);
console.log ('The result is:', result);
```

4. Run the code using **node main.js**



```
The result is: 15
```

References:

- https://www.geeksforgeeks.org/import-and-export-in-node-js/?ref=rp
- https://javascript.plainenglish.io/create-a-single-page-website-using-node-js-and-express-js-a0b53e396e4f

Activity 2

Aim: Create a Web Application using Express JS

Learning outcome: Able to develop the real time scenarios based on Node JS applications.

Duration: 4 Hour

List of Hardware/Software requirements:

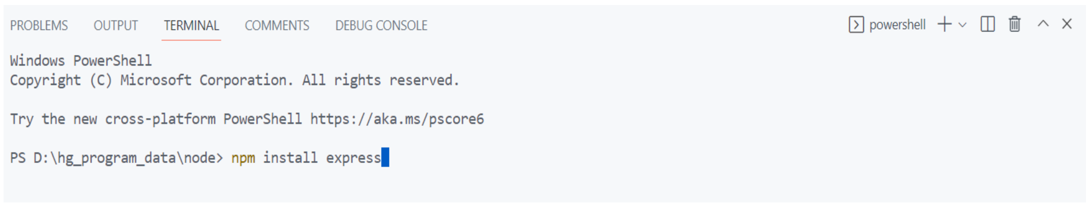
1. Operating System – Windows 10/11 or Linux
2. Command Prompt/Power Shell
3. Text Editor – Notepad / IDE – Visual Studio Code/Any IDE
4. NodeJS
5. Express JS

Code/Program/Procedure (with comments):

1. Make a file app.js using Visual Studio Code.

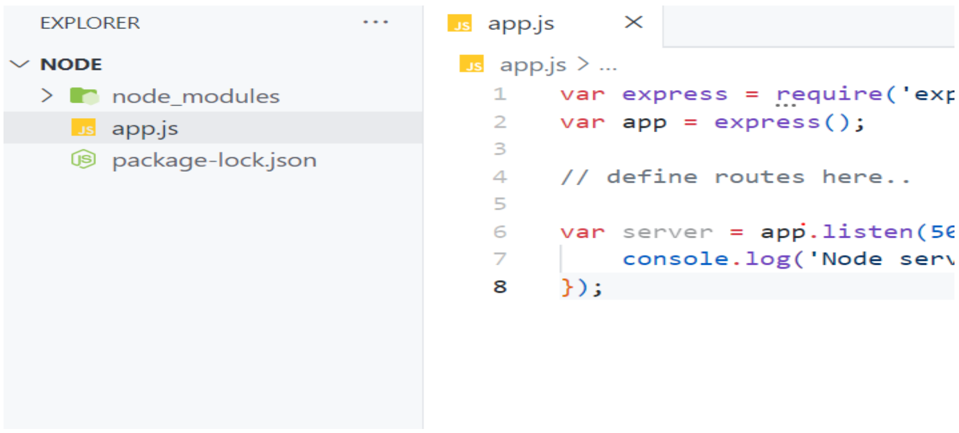
```
var express = require('express');
var app = express();

// define routes here..

var server = app.listen(5000, function () {
console.log('Node server is running..');
});
```

2. Open terminal in that specific folder to install all modules as mentioned below



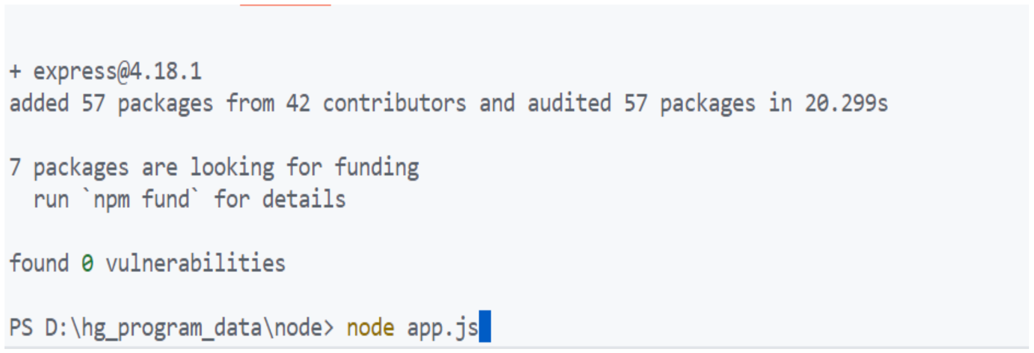
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS D:\hg_program_data\node> npm install express
```

3. Now we can see all required modules are installed.



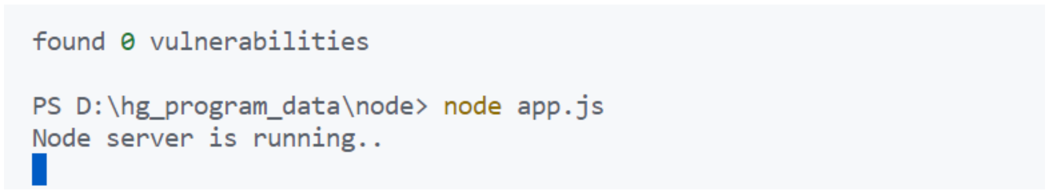
```
var express = require('exp
var app = express();

// define routes here..

var server = app.listen(50
    console.log('Node serv
});
```

4. Now run the app using **node app.js**



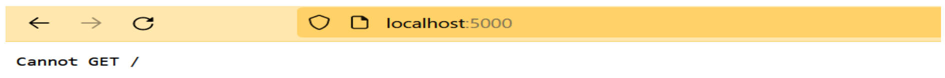
```
+ express@4.18.1
added 57 packages from 42 contributors and audited 57 packages in 20.299s

7 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

PS D:\hg_program_data\node> node app.js
```

5. Now app is running on local system on port address 5000 and the link is localhost:5000



```
found 0 vulnerabilities

PS D:\hg_program_data\node> node app.js
Node server is running..
```

6. check link on browser

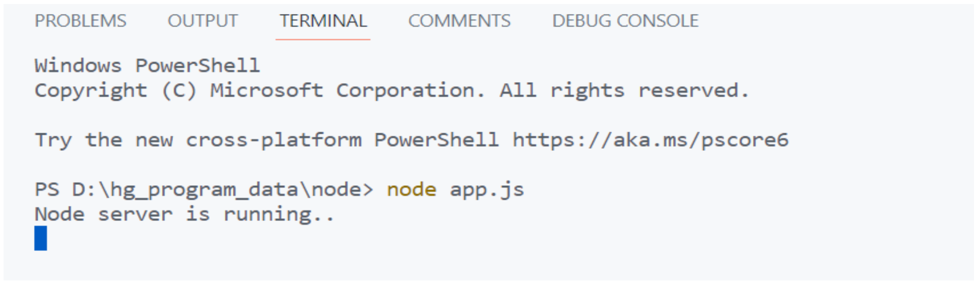


```
Cannot GET /
```

7. We are not able to see any output because routes are not defined yet. Modify the **app.js** with following code:

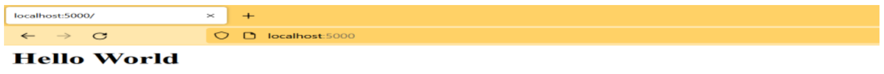
```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
res.send('<html><body><h1>Hello World</h1></body></html>');
});

app.post('/submit-data', function (req, res) {
res.send('POST Request');
});

app.put('/update-data', function (req, res) {
res.send('PUT Request');
});

app.delete('/delete-data', function (req, res) {
res.send('DELETE Request');
});

var server = app.listen(5000, function () {
console.log('Node server is running..');
});
```

8. Now run the file again using node app.js command.



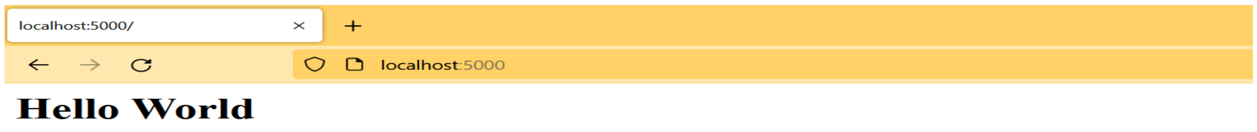
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS D:\hg_program_data\node> node app.js
Node server is running..
```

9. Server running on the same port address with the message “Hello World”



Hello World

Output/Results snippet:



Hello World

References:

- https://www.tutorialsteacher.com/nodejs/expressjs-web-application

Activity 3

Aim: Create RestAPI using express JS

Learning outcome: Able to develop the real time scenarios based on Node JS applications.

Duration: 4 Hour

List of Hardware/Software requirements:

1. Operating System – Windows 10/11 or Linux
2. Command Prompt/Power Shell
3. Text Editor – Notepad / IDE – Visual Studio Code/Any IDE
4. NodeJS
5. Express JS

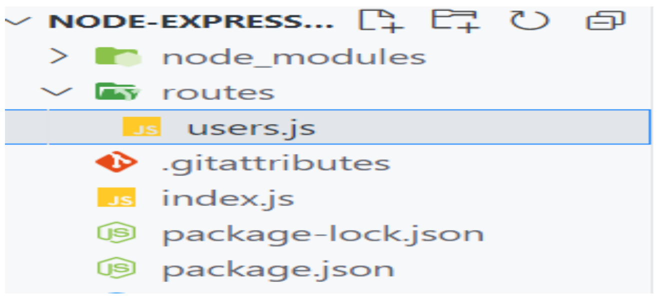
Code/Program/Procedure (with comments):

1. Make a file name **index.js** using visual studio code.

```
import  express  from 'express';
import bodyParser from 'body-parser';
import users Routes from './routes/users.js'

constapp=express();
constPORT=5000;
app.use(bodyParser.json());
app.use('/users', usersRoutes);

app.get('/', (req, res)=>res.send('This is Homepage, kindly refer json file here
http://localhost:5000/users'));
```

```
app.listen(PORT, (req, res)=>console.log(`Server is Running on port http://localhost:${PORT}`));
```

2. Install all required module with the help of **npm install express**



3. Make routes folder and create users.js in the same folder.

```
import express from'express'; // Web Framework for nodejs
import { v4 as uuidv4 } from'uuid'; //Get unique id every time
const router =express.Router(); //using express for routing

let users = [
   {
      "firstName": "John",
      "lastName": "Doe",
      "age": 25,
      "id": "fcb4efa5-608d-4d91-aed3-33af404928ae"
    },
    {
      "firstName": "Johny",
      "lastName": "Doe",
      "age": 25,
      "id": "2fb6fed6-43f3-4ee6-ab04-2bd3782c1886"
    },
    {
      "firstName": "Jason",
      "lastName": "Dew",
      "age": 30,
      "id": "a4f12aa0-8fd5-473d-9834-aa98a2bdd84c"
    },
    {
      "firstName": "BSon",
      "lastName": "DB",
      "age": 40,
      "id": "179c592c-1d4d-451c-808c-e72749e9c880"
    }
] //variable used to store data

// GET METHOD i.e. Read request can done from browser
router.get('/', (req, res)=>{
  // console.log(users);
  res.send(users);
});

// POST METHOD i.e. CREATE request can be done from POSTMAN/THUNDER CLIENT
router.post('/', (req, res)=>{
  // console.log('POST ROUTE Reached');
  // console.log(req.body);

  constuser=req.body; //when post request implemented, data stored in body and stored
in variable i.e. 'user'
  constuserID=uuidv4(); // For unique id⇨ '9b1deb4d-3b7d-4bad-9bdd-2b0d7b3dcb6d'
  constuserWithID= { ...user, id: userID } //Spread Operator
  users.push(userWithID); //With Push Function it will add data in "users" array

  // res.send('POST ROUTE Reached');
  res.send(`User with the name ${user.firstName} added to the database.`); //
Confirmation message on client side
});

// GET METHOD via passing id, Read request can done from browser
router.get('/:id', (req, res)=>{
  // console.log(req.params);
```

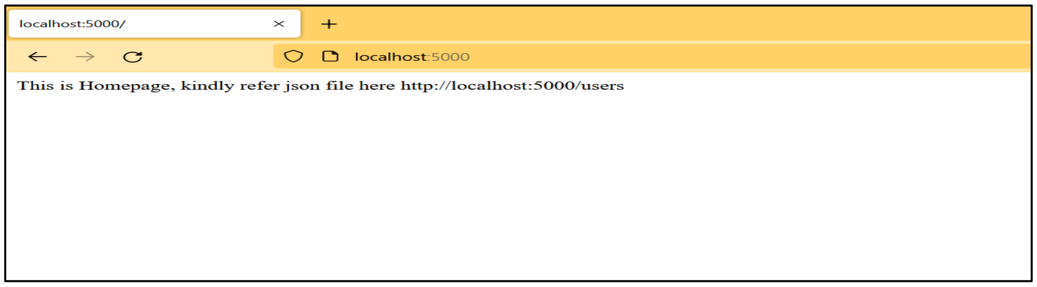
```
    // res.send('THE GET ID ROUTE');
    // res.send(req.params);
    const { id } =req.params; //id will store in params and stored in variable
    constfoundUser=users.find((user)=>user.id ==id); //using find filter to fetch exact same
id
    res.send(foundUser);
});

// DELETE METHOD via passing id i.e. DELETE request can be done from
POSTMAN/THUNDER CLIENT
router.delete('/:id', (req, res)=>{
    const { id } =req.params;
    users=users.filter((user)=>user.id !==id);
    res.send(`User with the id ${id} deleted from the database`);
});

// PATCH METHOD(PARTIAL MODIFICATION) request via passing id i.e. UPDATE
request can be done from POSTMAN/THUNDER CLIENT
router.patch('/:id', (req, res)=>{
    const { id }=req.params;
    const { firstName, lastName, age } =req.body;

    constuser=users.find((user)=>user.id ===id);

    if (firstName) {
        user.firstName=firstName;
    }
    if (lastName) {
        user.lastName=lastName;
    }
    if (age) {
        user.age=age;
    }

    res.send(`User with the id ${id} has been updated`);
```

```
});

// Note - PUT method is used to completely overwrite
exportdefaultrouter;
```

4. Now run the application using **node index.js**

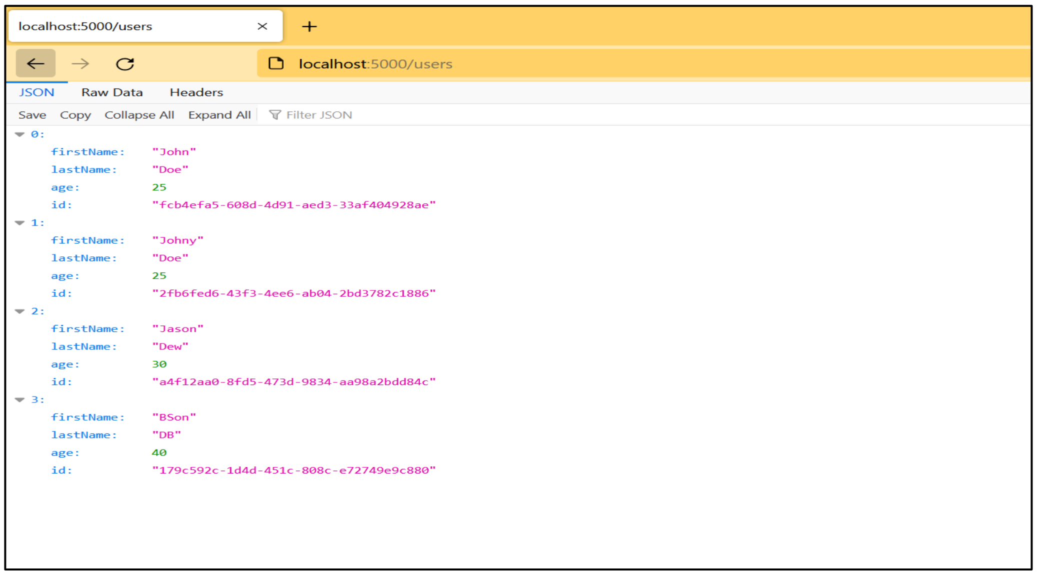
```
Try the new cross-platform PowerShell https://aka.ms/pscore6

PS D:\hg_program_data\git_clones\Node-Express-CRUD-REST-API> node index.js
```

5. Referring localhost:5000/users to see data that is mentioned in JSON format.



This is Homepage, kindly refer json file here http://localhost:5000/users

6. The data available is JSON format.



```
▾ 0:
    firstName:    "John"
    lastName:     "Doe"
    age:          25
    id:           "fcb4efa5-608d-4d91-aed3-33af404928ae"
▾ 1:
    firstName:    "Johny"
    lastName:     "Doe"
    age:          25
    id:           "2fb6fed6-43f3-4ee6-ab04-2bd3782c1886"
▾ 2:
    firstName:    "Jason"
    lastName:     "Dew"
    age:          30
    id:           "a4f12aa0-8fd5-473d-9834-aa98a2bdd84c"
▾ 3:
    firstName:    "BSon"
    lastName:     "DB"
    age:          40
    id:           "179c592c-1d4d-451c-808c-e72749e9c880"
```

References:

- https://www.tutorialsteacher.com/nodejs/expressjs

Learning Outcome

After completing this module, the student should be **Able to develop the real time scenarios based on Node JS applications.**

To meet the learning outcome, a student has to complete the following activities

1. Create Restapi & testing rest api postman

Activity 1

Aim: Create Rest api & test rest api using postman

Learning outcome: Able to develop the real time scenarios based on Node JS applications.

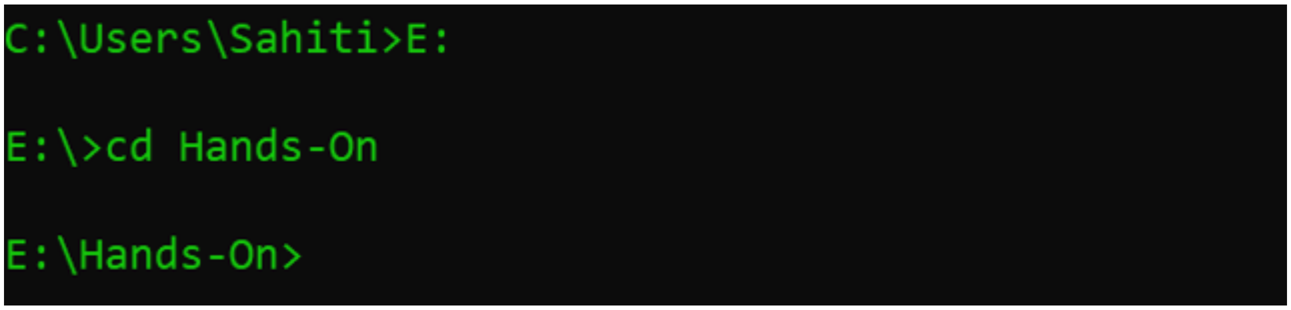
Duration: 2 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Windows OS / Linux OS - Ubuntu 18.04 LTS
2. NodeJS, ExpressJS, Joi, Nodemon, Postman software installed

Code/Program/Procedure (with comments):

Step 1: Create a project directory, which will consist of all the files present in the project. Then, open commands prompt and navigate to the project directory. Refer below.



```
C:\Users\Sahiti>E:

E:\>cd Hands-On

E:\Hands-On>
```

Step 2: Now, call npm using the below command. This will initialize the npm modules in your system.

npm init

Once you hit enter, Node.js, will ask you to enter a few details related to the project. These details will basically be the metadata for your project.

Here you can define your entry point along with several other information. For this demo, I will be using script.js as an entry point.

It will then, ask you for a confirmation for the data you must have mentioned. Just press on Y to confirm.

Step 3: Next, you have yo install Express.js using the below command:

```
npm i express
```

Express is a web framework which can be used along with Node.js. This web framework will allow you to create Restful APIs, with the help of helper methods, middle layers to configure your application.

Step 3.1: Similarly, you have to install Joi.

```
npm i joi
```

This package allows you to create blueprints for JavaScript objects which store information to ensure validation of key information.

Step 3.2: Finally, install the node monitoring package nodemon, using the below command.

```
npm i -g nodemon
```

Nodemon, keeps a watch on all the files with any type of extension present in this folder. Also, with nodemon on the watch, you don't have to restart the Node.js server each time any changes are made. Nodemon will implicitly detect the changes and restart the server for you.

package.json

```
{
"name": "restapidemo",
"version": "1.0.0",
"description": "Creation of REST API",
"main": "script.js",
"scripts": {
"test": "echo \"Error: no test specified\" && exit 1"
},
"author": "sahiti_kappagantula",
"license": "ISC",
"dependencies": {
"express": "^4.17.1",
"joi": "^14.3.1"
}
}
```

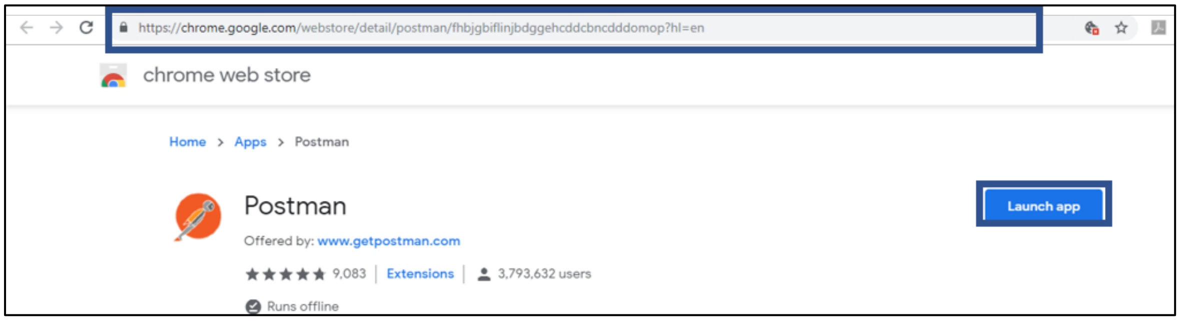
script.js

```
const express = require('express'); //Import Express
const Joi = require('joi'); //Import Joi
const app = express(); //Create Express Application on the app variable
app.use(express.json()); //used the json file

//Give data to the server
const customers = [
{title: 'George', id: 1},
{title: 'Josh', id: 2},
{title: 'Tyler', id: 3},
{title: 'Alice', id: 4},
{title: 'Candice', id: 5}
]

//Read Request Handlers
// Display the Message when the URL consist of '/'
app.get('/', (req, res) => {
res.send('Welcome to Edurekas REST API!');
});
// Display the List Of Customers when URL consists of api customers
app.get('/api/customers', (req,res)=> {
res.send(customers);
```

```
});
// Display the Information Of Specific Customer when you mention the id.
app.get('/api/customers/:id', (req, res) => {
const customer = customers.find(c => c.id === parseInt(req.params.id));
//If there is no valid customer ID, then display an error with the following message
if (!customer) res.status(404).send('<h2 style="font-family: Malgun Gothic; color: darkred;">Ooops... Cant find what you are looking for!</h2>');
res.send(customer);
});

//CREATE Request Handler
//CREATE New Customer Information
app.post('/api/customers', (req, res)=>; {

const { error } = validateCustomer(req.body);
if (error){
res.status(400).send(error.details[0].message)
return;
}
//Increment the customer id
const customer = {
id: customers.length + 1,
title: req.body.title
};
customers.push(customer);
res.send(customer);
});

//Update Request Handler
// Update Existing Customer Information
app.put('/api/customers/:id', (req, res) => {
const customer = customers.find(c=> c.id === parseInt(req.params.id));
if (!customer) res.status(404).send('<h2 style="font-family: Malgun Gothic; color: darkred;">Not Found!! </h2>');

const { error } = validateCustomer(req.body);
```

```
if (error){
res.status(400).send(error.details[0].message);
return;
}

customer.title = req.body.title;
res.send(customer);
});

//Delete Request Handler
// Delete Customer Details
app.delete('/api/customers/:id', (req, res) =>; {

const customer = customers.find( c=> c.id === parseInt(req.params.id));
if(!customer) res.status(404).send('<h2 style="font-family: Malgun Gothic; color:
darkred;">Not Found!!</h2>'</span>);

const index = customers.indexOf(customer);
customers.splice(index,1);

res.send(customer);
});
//Validate Information
function validateCustomer(customer) {
const schema = {
title: Joi.string().min(3).required()
};
return Joi.validate(customer, schema);

}

//PORT ENVIRONMENT VARIABLE
const port = process.env.PORT || 8080;
app.listen(port, () => console.log(`Listening on port ${port}..`));
```

Step 4: Now, the next step is to check whether the handlers are working properly or not. For that, we will use a Chrome extension called Postman. To install Postman you can search for postman in google web store and click on 'Add to Chrome'.

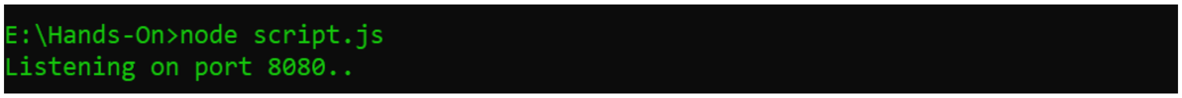

https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop?hl=en
chrome web store
Home > Apps > Postman
Postman
Offered by: www.getpostman.com
9,083 | Extensions | 3,793,632 users
Runs offline
Launch app

Step 5: Now, once you have installed Postman, open it to test your application.

Step 6: But before that you have to start your server. To start your server, type the following command.

node script.js

You would see the output as below:



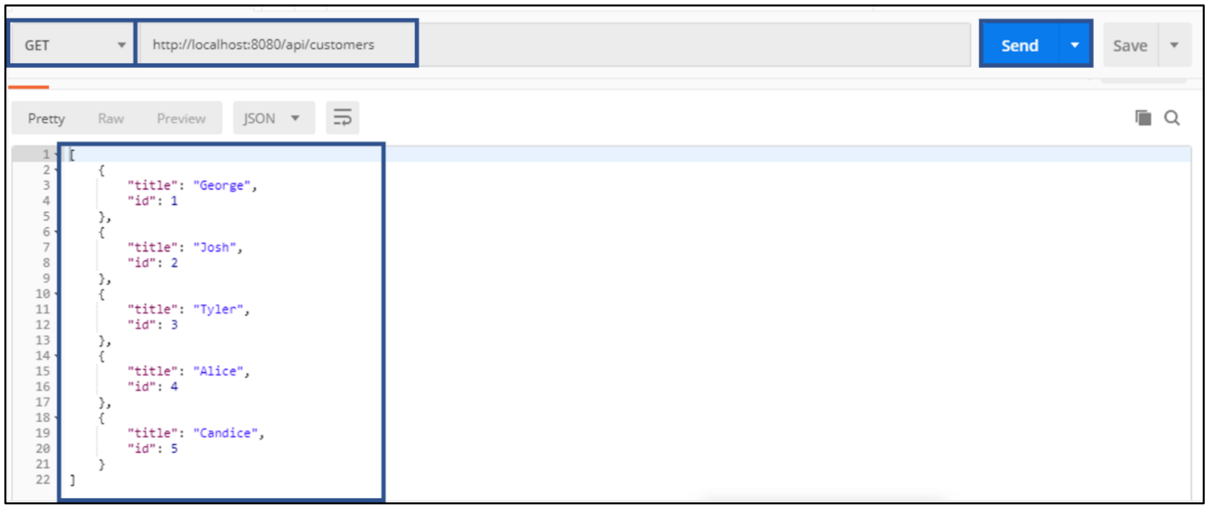
```
E:\Hands-On>node script.js
Listening on port 8080..
```

Output/Results snippet:

Let us start by testing the GET Method.

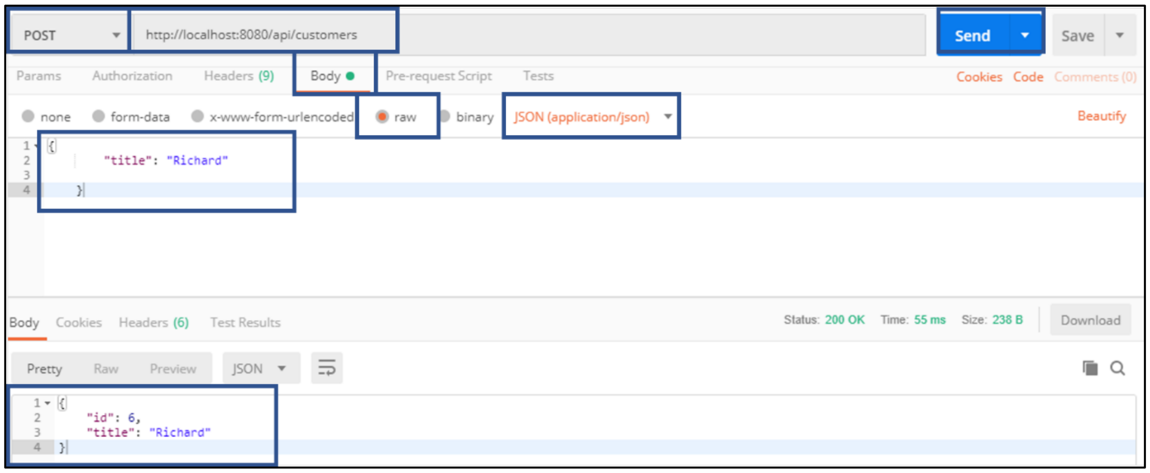
Step 7: In order to do that you need to select GET from the drop-down list, type in the defined URL and hit send.

If your code is working fine, then you will see the list of all the customers which we have added manually in our code. In the below picture, you can see how my result looks like. Here I have mentioned the URL to be localhost:8080/api/customers



```
[
    {
        "title": "George",
        "id": 1
    },
    {
        "title": "Josh",
        "id": 2
    },
    {
        "title": "Tyler",
        "id": 3
    },
    {
        "title": "Alice",
        "id": 4
    },
    {
        "title": "Candice",
        "id": 5
    }
]
```

Step 8: Now, let's try adding a new customer to our stack of customers. For that, select 'POST' from the drop-down list and type in the defined URL for the POST method. Then, click on 'Body', select 'raw' and move on to select 'JSON' from the drop-down list as depicted in the below image. Now, in the text area, type in the name of your customer as shown and hit send.



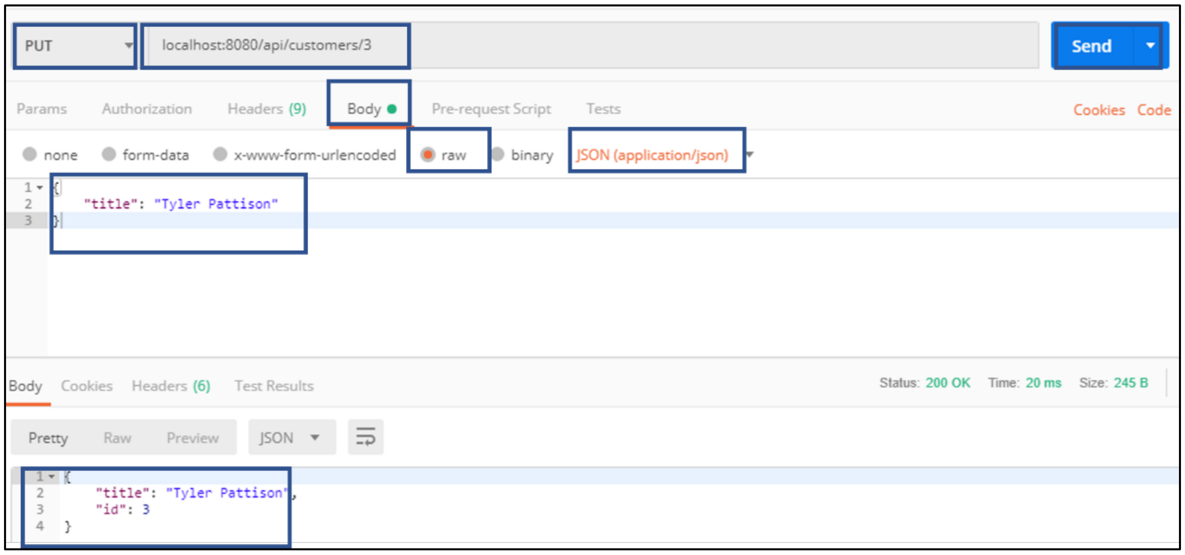
```
{
    "title": "Richard"
}
```

```
{
    "id": 6,
    "title": "Richard"
}
```

If your POST method is working fine, your response body will contain the new customer's name along with the Customer ID. Here if you observe, we have only mentioned the name but we did not give the customer ID. This implies that the Customer ID is automatically incremented

Step 9: Now, let’s try to update a Customer name. Let us say we ant to update the name of the Customer ID = 3. So, to update the data, you need to first select ‘PUT’ from the drop-down table and enter the PUT request’s URL along with the customer id you wish to update. Next in the ‘Body’, type in the new customer name and hit enter.

This will give you a response with the customer id and updated customer name.



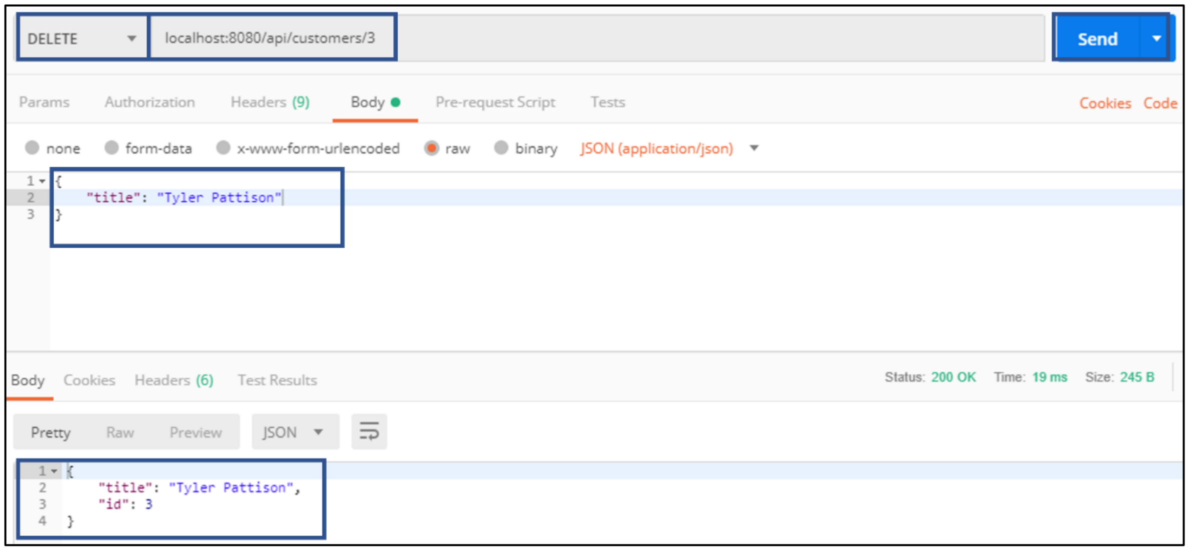
localhost:8080/api/customers/3

```
{
    "title": "Tyler Pattison"
}
```

Status: 200 OK   Time: 20 ms   Size: 245 B

```
{
    "title": "Tyler Pattison",
    "id": 3
}
```

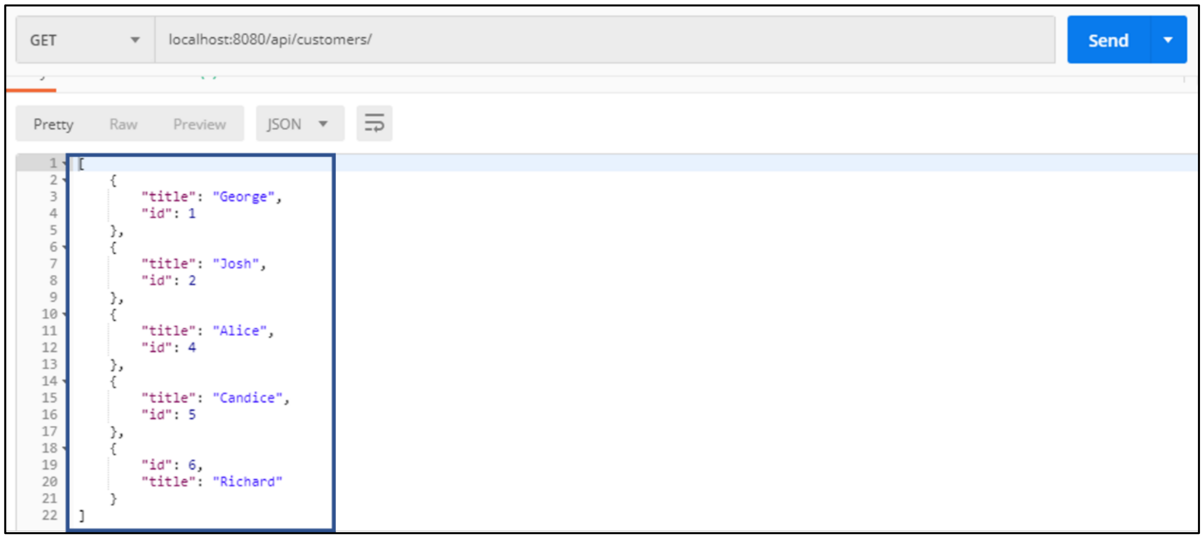
Step 10: Finally, let’s send a ‘DELETE’ request to delete an existing record. For that select DELETE from the drop-down list and type in the URL of the delete request handler along with the customer’s details, you want to remove and hit enter. Let’s say, I want to delete the details of a customer with id = 3. If your transaction is successful, you will see the complete details of the entry you have removed in the response body.



```
{
    "title": "Tyler Pattison"
}
```

```
{
    "title": "Tyler Pattison",
    "id": 3
}
```

Now, let’s send a GET request for our final list of customers.



```
[
    {
        "title": "George",
        "id": 1
    },
    {
        "title": "Josh",
        "id": 2
    },
    {
        "title": "Alice",
        "id": 4
    },
    {
        "title": "Candice",
        "id": 5
    },
    {
        "id": 6,
        "title": "Richard"
    }
]
```

As you can see from the above screenshot, the response body contains a total of five customers with the customer id 3 missing as we have already deleted that entry.

References:

- https://www.edureka.co/blog/what-is-rest-api/

Learning Outcome

After completing this module, the student should be able to configure an embedded Mongo DB and Create application with CRUD operation

To meet the learning outcome, a student has to complete the following activities

1. Create application with CRUD operations using Mongo DB (2Hr)
2. Web Application Integration Using Express+Angular+Mongo

Activity 1

Aim: Create application with CRUD operations using Mongo DB

Learning outcome: Able to configure embedded Mongo DB application with Node JS

Duration: 2 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Windows 10/11
2. Mongo DB
3. Node JS, Curl

Code/Program/Procedure (with comments):

Node JS and MongoDB CRUD Operations

Method 1

1. Node JS with MongoDB CRUD Operations: MongoDB and Node.js Setup

Your computer should be installed with MongoDB and Node.js and a command-line with *curl* command. Download MongoDB MSI package from its official website and run it to install MongoDB. After the installation, run the *mongo* command to ensure that MongoDB is running.

Next, ensure that both MongoDB and npm are installed on your computer. You can download Node.js from its official website and install it on your computer. Curl will help you to run HTTP requests on the command line.

2. Node JS with MongoDB CRUD Operations: Create a New Node.js Project

Open the folder where you will create a project and type the following command:

```
npm init
```

Give the project the name *node-mongo* and accept defaults for the other settings. Add the necessary dependencies to the project. Run the following command within the project directory:

npm install mongodb polka --save

The above command will install the Node.js driver for MongoDB and the Polka HTTP Server to handle HTTP requests. Add the following start script to the *package.json* file:

Create a ***/node-mongo/src/index.js*** file and add the following contents to it:

```
const polk= require('polka');
polk()
  .get('/create', (req, res) => {
    res.end(`It works`);
  })
  .listen(3000, err => {
    if (err) throw err;
    console.log(`> localhost:3000`);
  });
```

Now, run the following command to start the server:

npm run start

Test the server by running the following command:

curl http://localhost:3000/create

3. Node JS with MongoDB CRUD Operations: Inserting a Record into GridDB

We need to perform an insert, which is the C in CRUD. Change your *index.js* file to the following:

```
const polk = require('polka');
const { MongoClient } = require("mongodb");
```

```
polk()
  .get('/create', (req, res) => {
    const cl = new MongoClient("mongodb://localhost:27017");
    async function run() {
      try {
        await cl.connect();
        const dbs = client.db("intro");
        const coll = dbs.collection("quotes");

        const rest = await coll.insertOne({"quote":"This is my quote."});
        res.end(JSON.stringify(rest));
      } catch (ex) {
        console.log("Error: " + ex);
      } finally {
        await cl.close();
      }
    }
    run().catch(console.dir);
  })
  .listen(3000, err => {
    if (err) throw err;
    console.log(`> localhost:3000`);
  });
```

The above script is an example of a node js with MongoDB crud operation. It establishes a connection to a MongoDB instance, selects a database named *intro* and a collection named *quotes.* It then inserts a document into the collection and returns the results as an HTTP response.

To run the insert, press Ctrl+C to stop and restart the node. Next, run the following command on the terminal:

```
npm run startcurl http://localhost:3000/create
```

4. Node JS with MongoDB CRUD Operations: Retrieving from MongoDB

We can use the *.get*() method to retrieve a document from MongoDB. The following script demonstrates this:

```
.get('/retrieve', (req, res) => {
  const cl = new MongoClient("mongodb://localhost:27017");
  async function run() {

    try {
      await cl.connect();
      const dbs= client.db("intro");
      const coll = dbs.collection("quotes");

      const cur = coll.find({}, {});

      let items = [];
      await cur.forEach(function(doc){
        items.push(doc);
      });
      res.end(JSON.stringify(items));
    } catch (err){
      console.warn("ERROR: " + err);
      if (errCallback) errCallback(err);
    } finally {
      await cl.close();
    }
  }
  run().catch(console.dir);
 })
```

We have used the *find* command and passed to it an empty query. This means that it will match all documents in the collection. The response is then returned to the client in the form of an array.

5. Node JS with MongoDB CRUD Operations: Updating MongoDB Documents

In this section, we will demonstrate how to **update** MongoDB documents, which is the “U” in CRUD:

```
.get('/update', (req, res) => {
    const cl = new MongoClient("mongodb://localhost:27017");
    async function run() {
      try {
        await cl.connect();
        const dbs = client.db("intro");
        const coll = dbs.collection("quotes");

        const updateDocument = {
          $set: {
            author:
              "Martin Kings",
          },
        };

        const rst = await coll.updateOne({}, updateDocument, {});
        res.end("Updated: " + rst.modifiedCount);
      } catch (ex) {
        errCallback(ex);
      } finally {
        await cl.close();
      }
    }
    run().catch(console.dir);
  })
```

In the above node js with MongoDB crud operation, we have connected to the database and created an updated document. We have then changed the *author* field to *Martin Kings.* The *updateOne()* function helped us to execute the update. Since we wanted to match all documents, the filter is empty.

6. Node JS and MongoDB CRUD Operations: Deleting MongoDB Documents

The following script demonstrates how to delete a MongoDB document from Node.js:

```
.get('/delete', (req, res) => {
    const cl = new MongoClient("mongodb://localhost:27017");
    async function run() {
      try {
        await cl.connect();
        const dbs = cl.db("intro");
        const coll = dbs.collection("quotes");
        const qry = { };
        const rst = await coll.deleteOne(qry);
        if (rst.deletedCount === 1) {
          res.end("One document deleted.");
        } else {
          res.end("No document was deleted.");
        }
      } finally {
        await cl.close();
      }
    }
```

We have used an empty query so as to match all the documents within the collection. The *deleteOne()* function tells us the number of documents that have been deleted.

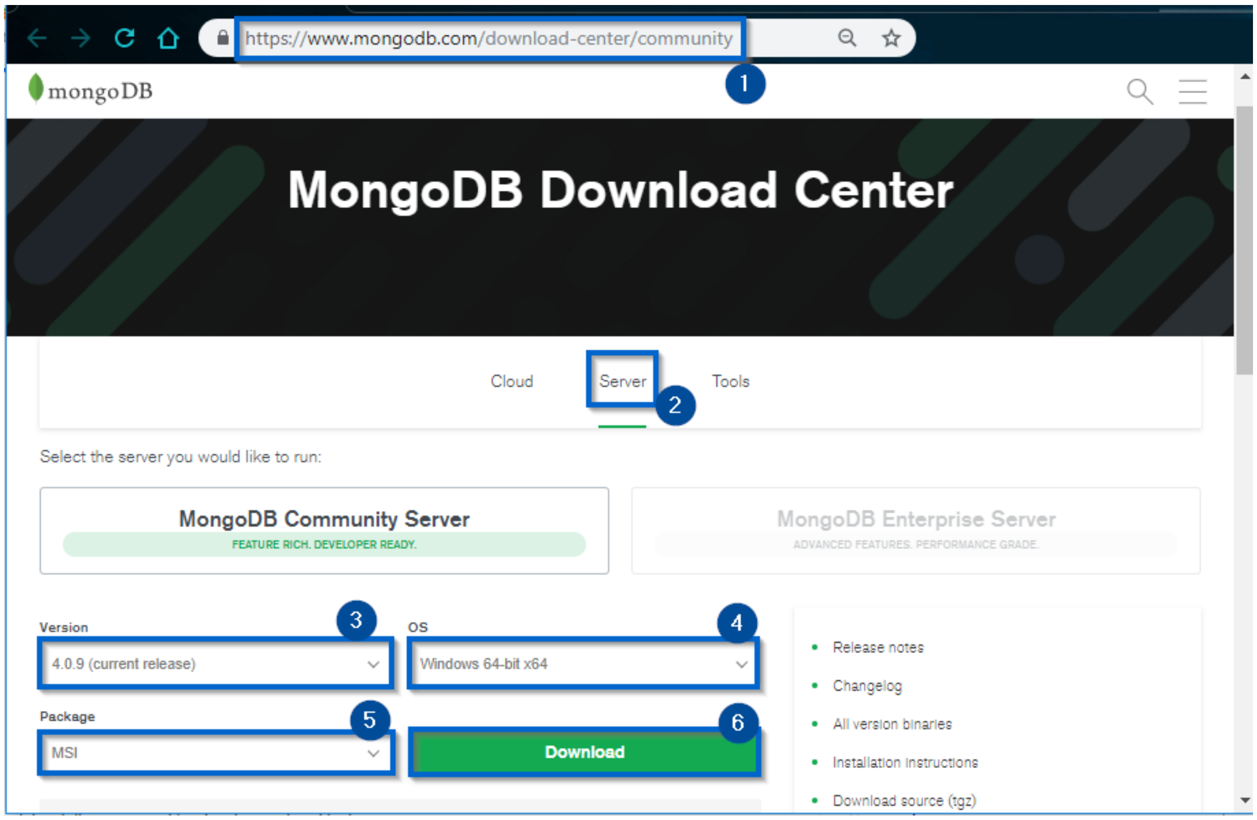
That is how to run node js with MongoDB crud operations.

Method 2

MongoDB Installation

Step I: Download the latest version MongoDB server from its official site: https://www.mongodb.com/download-center/community

Step II: Next click on the 'Server' tab as shown in the below screenshot.



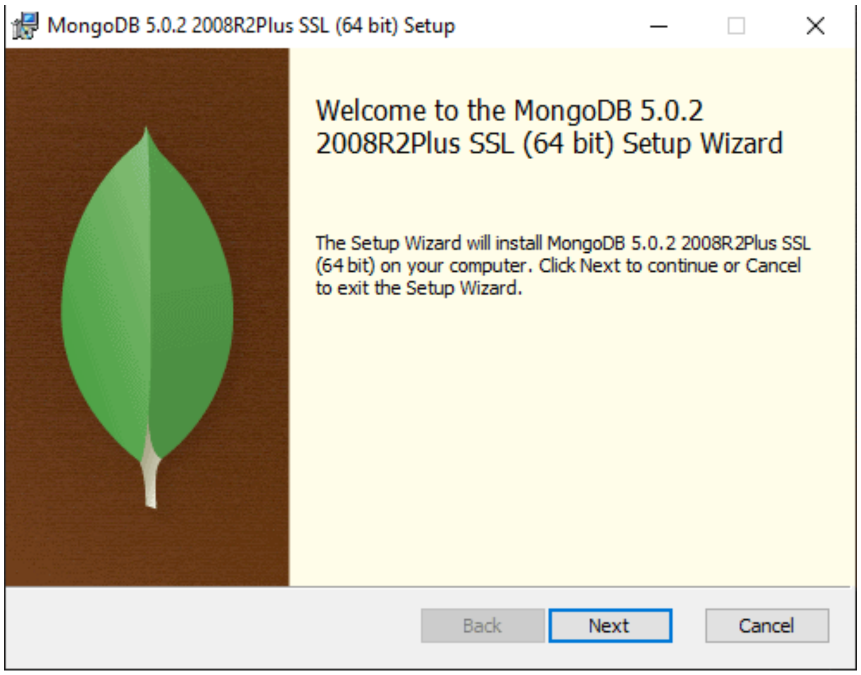
Step III: If you are looking for any specific version, you can select it from the drop-down list or you can just download the latest version.

Step IV: Select your OS from the drop down. Since I am working on Windows I will go for Windows 64 bit.

Step V: Now, select the package as MSI.

Step VI: Finally, click on 'Download' to begin the download process.

Step VII: Once downloaded, double click on the MSI file to open it and proceed with the installation wizard.


MongoDB 5.0.2 2008R2Plus SSL (64 bit) Setup

Welcome to the MongoDB 5.0.2 2008R2Plus SSL (64 bit) Setup Wizard

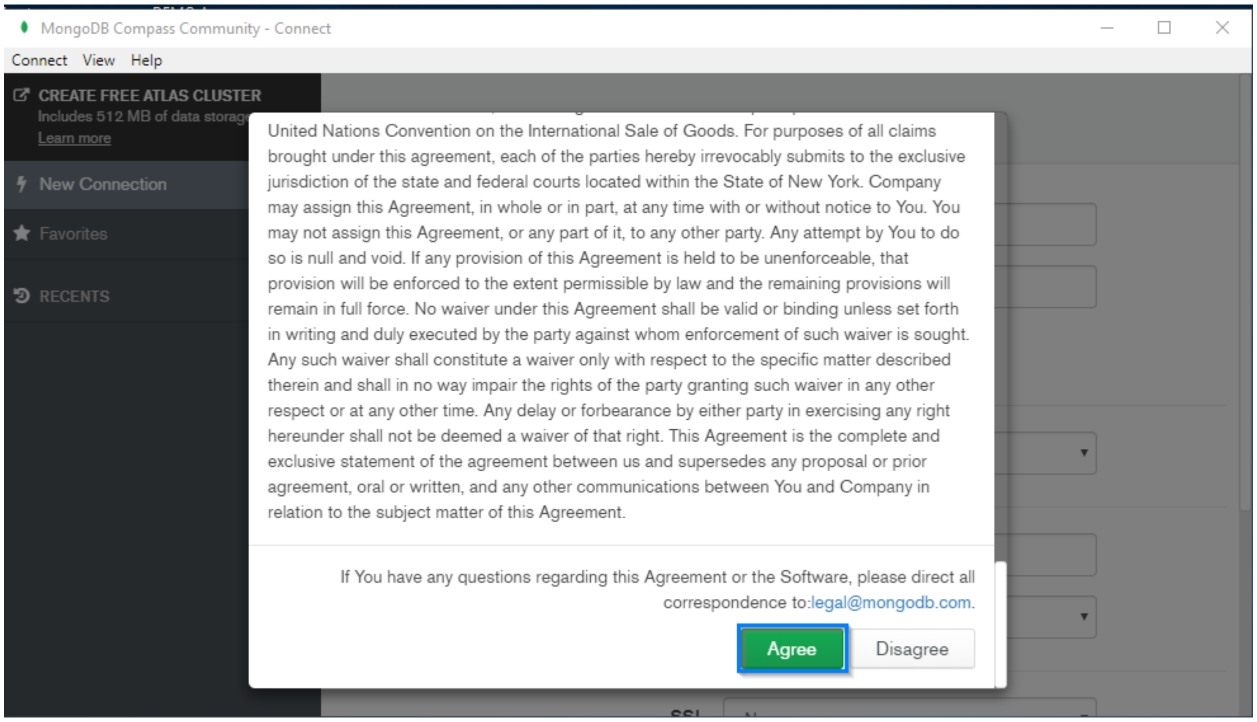
The Setup Wizard will install MongoDB 5.0.2 2008R2Plus SSL (64 bit) on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Step VIII: Now, in order to start the MongoDB server you have to run the .exe file and assign the database folder. To make the work easier, all you need to do is write down a few lines of code in a notepad file and save it with the .bat extension. In other words, you just need to create a batch file, which will start the MongoDB server for you without any hassle. To create the batch file type in the below code:

```
1. cd C:Program FilesMongoDBServer.0in (MongoDB path)
2. mongod.exe --dbpath F:MongoDBdata (database dump destination)
```

Now, whenever you want to launch the MongoDB server, all you need to do is double click this batch file and open the MongoDB Compass application.

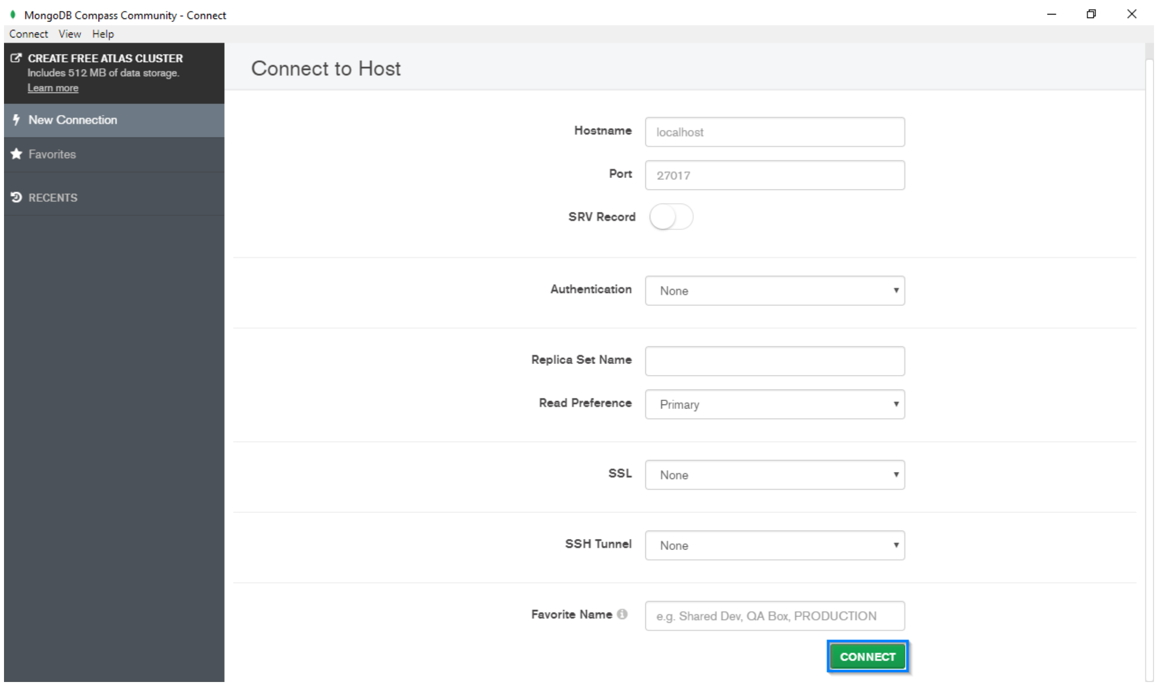
Step IX: Next, you need to launch the 'MongoDB Compass' and agree to its terms of use.


MongoDB Compass Community - Connect
Connect View Help
CREATE FREE ATLAS CLUSTER
Includes 512 MB of data storag
Learn more
New Connection
Favorites
RECENTS
Agree
Disagree

United Nations Convention on the International Sale of Goods. For purposes of all claims brought under this agreement, each of the parties hereby irrevocably submits to the exclusive jurisdiction of the state and federal courts located within the State of New York. Company may assign this Agreement, in whole or in part, at any time with or without notice to You. You may not assign this Agreement, or any part of it, to any other party. Any attempt by You to do so is null and void. If any provision of this Agreement is held to be unenforceable, that provision will be enforced to the extent permissible by law and the remaining provisions will remain in full force. No waiver under this Agreement shall be valid or binding unless set forth in writing and duly executed by the party against whom enforcement of such waiver is sought. Any such waiver shall constitute a waiver only with respect to the specific matter described therein and shall in no way impair the rights of the party granting such waiver in any other respect or at any other time. Any delay or forbearance by either party in exercising any right hereunder shall not be deemed a waiver of that right. This Agreement is the complete and exclusive statement of the agreement between us and supersedes any proposal or prior agreement, oral or written, and any other communications between You and Company in relation to the subject matter of this Agreement.

If You have any questions regarding this Agreement or the Software, please direct all correspondence to:legal@mongodb.com.

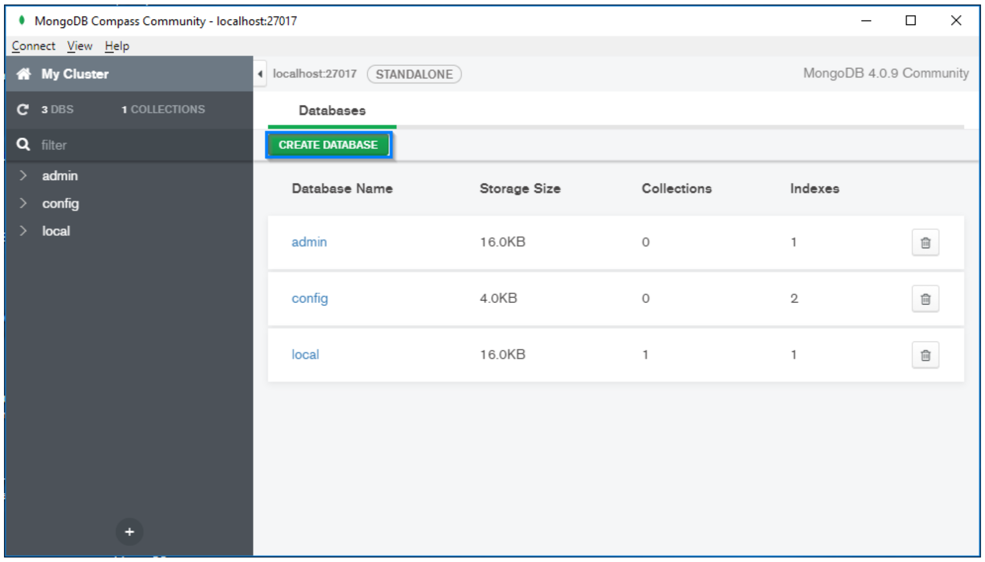
Step X: Now you need to provide the server configurations and hit ‘Connect’.


MongoDB Compass Community - Connect
Connect View Help
CREATE FREE ATLAS CLUSTER
Includes 512 MB of data storage.
Learn more
New Connection
Favorites
RECENTS
CONNECT

Connect to Host

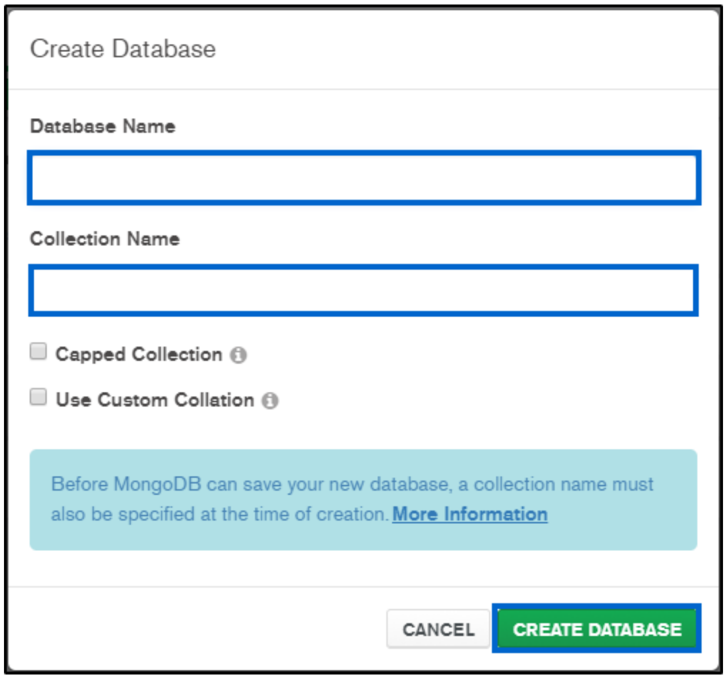
Hostname	localhost
Port	27017
SRV Record	
Authentication	None
Replica Set Name	
Read Preference	Primary
SSL	None
SSH Tunnel	None
Favorite Name	e.g. Shared Dev, QA Box, PRODUCTION

Step XI: Next, click on ‘Create Database’.



Database Name	Storage Size	Collections	Indexes
admin	16.0KB	0	1
config	4.0KB	0	2
local	16.0KB	1	1

Step XII: Now, provide a relevant name for your database and collection and hit ‘Create Database’.



Before MongoDB can save your new database, a collection name must also be specified at the time of creation. More Information

I guess, now you are all set to get started with the practical part, so without any more delay let's dive into the code.

Node.js MongoDB Demo

Here I will be creating a CRUD application for Course Management with the help of Node.js and Express.js and use MongoDB to store the data. In this application, I will be taking course details like name, id, duration, and fee as inputs. For that, I will be creating a few view files which will act as an interface. Then in order to handle the data, I will be needing a controller as well which will help in manipulating the data. Finally, I will be needing a few model files to store the data. So basically, I will be following an MVC pattern for this application development. So, lets now jump into development.

Our application will be having the following hierarchy:

- **NodejsMongoDbDemo**
  - package.json
  - script.js
  - **controllers**
    - courseController.js
  - **img**
    - logo.jpg
  - **models**
    - course.model.js
    - mongodb.js
  - **views**
    - **course**
      - courseAddEdit.hbs
      - list.hbs
    - **layouts**
      - mainLayout.hbs

So, let's begin the application development by creating a directory for the project. Once you are done, open the command prompt and navigate to your project directory. Now you need to set up the project configurations for that, type in the below command and provide the necessary details:

```
npm init
```

Now, you need to install the required packages. So, in this project, I am using the below packages:

- **express.js:** It is a web framework.
- **express-handlebars:** It is a template engine and helps in creating client-side applications.
- **mongoose:** Helps in communicating with MongoDB.
- **body-parser:** Helps in converting the POST data into the request body.
- **nodemon:** Helps in automatically restarting the server whenever the code changes.

In order to install these packages, type in the following command:

```
npm i --s express express-handlebars mongoose body-parser
```

Since I want to install nodemon such that it can access any file in the directory, I will be installing it with the global command:

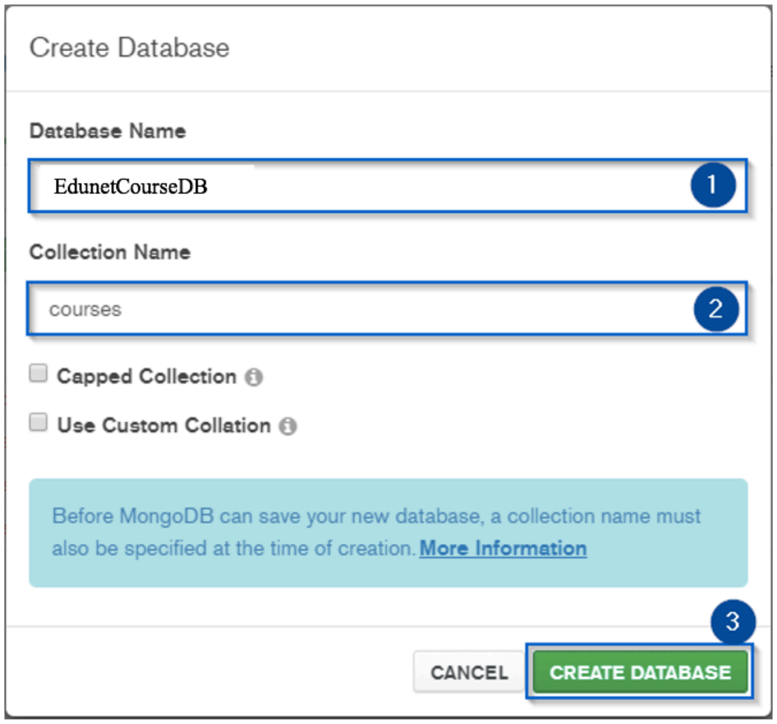
```
npm i -g nodemon
```

Once you are done installing with the packages, your final JSON file should look like the below file:

package.json

```
{

"name": "samplenodemongo",

"version": "1.0.0",

"description": "Edunet demo on how to build a Node.js application with MongoDB",

"main": "script.js",
"scripts": {
"test": "echo "Error: no test specified" && exit 1"
},
"author": "Edunet",
"license": "ISC",
"dependencies": {
"body-parser": "^1.19.0",
"express": "^4.16.4",
"express-handlebars": "^3.0.2",
"mongoose": "^5.5.6",
"nodemon": "^1.19.0"
}
}
```

As you can see, in the dependencies section all the installed packages have been successfully listed. So, lets now create the database we will be using in this demo. For that start the batch file and open MongoDB application. Now, create a new database and provide a collection name. In my application, I will be using ‘EdunetCoursesDB’ as the database name and ‘courses’ as the collection.


Create Database

Database Name

EdunetCourseDB

Collection Name

courses

Capped Collection

Use Custom Collation

Before MongoDB can save your new database, a collection name must also be specified at the time of creation. More Information

Now, switch back to your code editor where we will be creating the files to establish connectivity between Node.js and MongoDB. For that, first, you need to create a folder inside the project directory and name it 'model'. Inside this folder, create a javascript file with the name '**mongodb.js**' and type in the below code:

mongodb.js

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

mongoose.connect('mongodb://localhost:27017/EdunetCoursesDB', {useNewUrlParser:
true}, (err) => {

if (!err) {

console.log('Successfully Established Connection with MongoDB')

}

else {
```

```
console.log('Failed to Establish Connection with MongoDB with Error: '+ err)

}

});

 //Connecting Node and MongoDB

require('./course.model');
```

Now, you need to define the schema of your course database. For that, create a new JS file within the model folder and name it '**course.model.js**'. So, I am using four fields in my course object I am using four fields which are name, id, duration, and fee. To create this file, type in the below-given code.

course.model.js

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


//Attributes of the Course object

var courseSchema = new mongoose.Schema({

courseName: {

type: String,

required: 'This field is required!'

},

courseId: {

type: String

},

courseDuration: {

type: String
},

courseFee: {

type: String

}

});


mongoose.model('Course', courseSchema);
```

Now, you need to create the root file called ‘**script.js**‘. This file is the entry point of this application and will contain all the connection paths in it. You need to be really careful while providing the paths in this file as it might result in an error or application failure. Along with this, it is also responsible for invoking the server and establish the connection. In order to create this file, type in the below code:

script.js

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


//Import the necessary packages

const express = require('express');

var app = express();

const path = require('path');

const exphb = require('express-handlebars');

const bodyparser = require('body-parser');
const courseController = require('./controllers/courseController');


app.use(bodyparser.urlencoded({

extended: true

}));


//Create a welcome message and direct them to the main page

app.get('/', (req, res) => {

res.send('


<h2 style="font-family: Malgun Gothic; color: midnightblue ">Welcome to Edunet
Node.js MongoDB Tutorial!!</h2>


 Click Here to go to <b> <a href="/course">Course Page</a> </b>');

});

app.use(bodyparser.json());


//Configuring Express middleware for the handlebars

app.set('views', path.join(__dirname, '/views/'));

app.engine('hbs', exphb({ extname: 'hbs', defaultLayout: 'mainLayout', layoutDir:
__dirname + 'views/layouts/' }));

app.set('view engine', 'hbs');
```

```
//Establish the server connection

//PORT ENVIRONMENT VARIABLE

const port = process.env.PORT || 8080;

app.listen(port, () => console.log(`Listening on port ${port}..`));

 //Set the Controller path which will be responding the user actions

app.use('/course', courseController);
```

Next, in order to handle the user requests, you need to create the router file. For that first, create a folder and name it ‘controller’ and within this folder create a file with the name ‘**courseController.js**‘. In this file, we will be dealing with the CRUD operations related to the employee. Below is the code for creating this file:

courseController.js

```
//Import the dependencies

const express = require('express');

const mongoose = require('mongoose');

//Creating a Router

var router = express.Router();

//Link

const Course = mongoose.model('Course');


//Router Controller for READ request

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

res.render("course/courseAddEdit", {

viewTitle: "Insert a New Course for Edunet"
});
});

//Router Controller for UPDATE request
router.post('/', (req,res) => {
if (req.body._id == '')
insertIntoMongoDB(req, res);
else
updateIntoMongoDB(req, res);
});

//Creating function to insert data into MongoDB
function insertIntoMongoDB(req,res) {
var course = new Course();
course.courseName = req.body.courseName;
course.courseId = req.body.courseId;
course.courseDuration = req.body.courseDuration;
course.courseFee = req.body.courseFee;
course.save((err, doc) => {
if (!err)
res.redirect('course/list');
else
```

```
console.log('Error during record insertion : ' + err);
});
}
 //Creating a function to update data in MongoDB
function updateIntoMongoDB(req, res) {
Course.findOneAndUpdate({ _id: req.body._id }, req.body, { new: true }, (err, doc) => {
if (!err) { res.redirect('course/list'); }
else {
if (err.name == 'ValidationError') {
handleValidationError(err, req.body);
res.render("course/courseAddEdit", {
//Retaining value to be displayed in the child view
viewTitle: 'Update Course Details',
employee: req.body
});
}
else
console.log('Error during updating the record: ' + err);
}
});
}
```

```
//Router to retrieve the complete list of available courses

router.get('/list', (req,res) => {

Course.find((err, docs) => {

if(!err){

res.render("course/list", {

list: docs

});

}

else {

console.log('Failed to retrieve the Course List: '+ err);

}

});

});


//Creating a function to implement input validations

function handleValidationError(err, body) {

for (field in err.errors) {

switch (err.errors[field].path) {

case 'courseName':

body['courseNameError'] = err.errors[field].message;

break;

default:
```

```
break;
}
}
}

//Router to update a course using it's ID
router.get('/:id', (req, res) => {
Course.findById(req.params.id, (err, doc) => {
if (!err) {
res.render("course/courseAddEdit", {
viewTitle: "Update Course Details",
course: doc
});
}
});
});

//Router Controller for DELETE request
router.get('/delete/:id', (req, res) => {
Course.findByIdAndRemove(req.params.id, (err, doc) => {
if (!err) {
res.redirect('/course/list');
}

else { console.log('Failed to Delete Course Details: ' + err); }

});

});



module.exports = router;
```

Now, that we are done with backend files, the next step is to create the Views. For that first, you need to create a wrapper for the child views. But before that, create a folder with name '**views**'. Inside this folder create two more folders with names '**course**' and '**layouts**' respectively with .hbs extension. Now, navigate inside the 'layouts' folder and create the wrapper with the name '**mainLayout.hbs**'. This file will contain the basic skeleton of the application which will be reflected in the child views as well. In this file, I am inserting an image as well, so for that I will create a local folder with name **img** and save my image inside.

To create this file, type in the below codes:

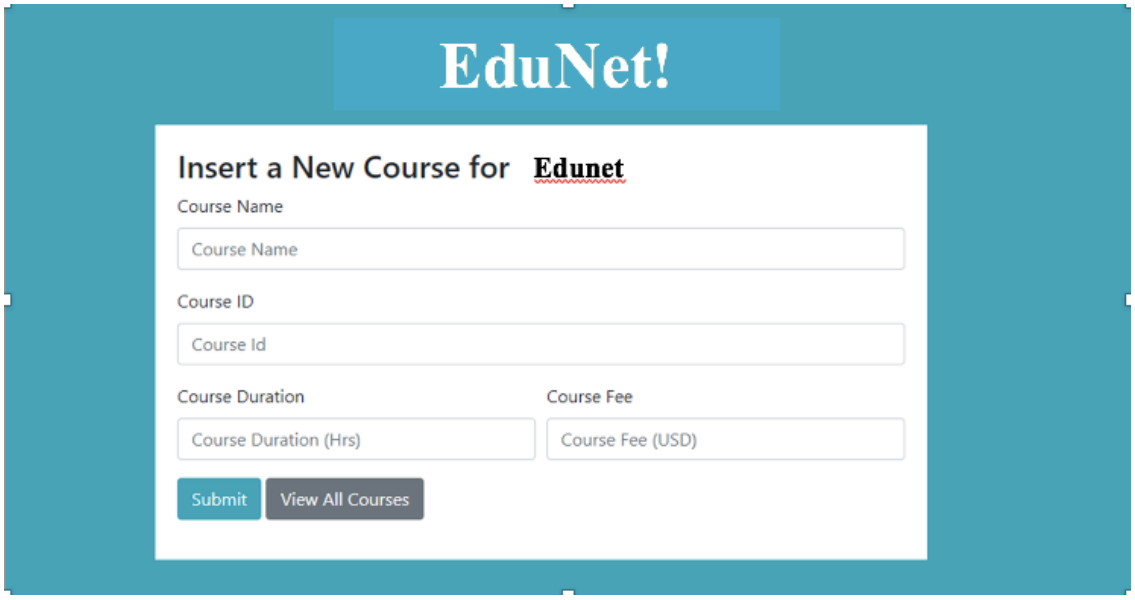
mainLayout.hbs

```
<!DOCTYPE html>

<html>

<head>

<title>Edunet Node.js MongoDB Demo</title>

<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">
```

```
<img
src="data:image/gif;base64,R0lGODlhAQABAIAAAAAAAP///yH5BAEAAAAALAAAAAA
BAAEAAAIBRAA7" data-wp-
preserve="%3Cscript%20src%3D%22https%3A%2F%2Fajax.googleapis.com%2Fajax
%2Flibs%2Fjquery%2F3.2.1%2Fjquery.min.js%22%3E%3C%2Fscript%3E" data-mce-
resize="false" data-mce-placeholder="1" class="mce-object" width="20" height="20"
alt="&lt;script&gt;" title="&lt;script&gt;" />

<img
src="data:image/gif;base64,R0lGODlhAQABAIAAAAAAAP///yH5BAEAAAAALAAAAAA
BAAEAAAIBRAA7" data-wp-
preserve="%3Cscript%20src%3D%22https%3A%2F%2Fmaxcdn.bootstrapcdn.com%2
Fbootstrap%2F3.3.7%2Fjs%2Fbootstrap.min.js%22%3E%3C%2Fscript%3E" data-mce-
resize="false" data-mce-placeholder="1" class="mce-object" width="20" height="20"
alt="&lt;script&gt;" title="&lt;script&gt;" />

</head>

 <body class = "bg-info">




<div align="center">

<!-- Inserting the image -->

<img src="/static/EdunetLogo.png" alt="Edunet Logo">

</div>

<div class="row">

 <div class="col-md-6 offset-md-3" style="background-color: #fff; margin-top: 40px;
padding:20px;">
```

```
<!-- retrieving HTML String from the child Views -->

{{{body}}}

</div>

 </div>

 </body>

</html>
```

Finally, inside the course folder, we will be creating two child views, one of which will be used for course addition or update and the second view will display the complete list of the available courses. Let's first focus on the first view i.e **courseAddEdit.hbs** which will look like the below screenshot.



So, let me now show, how to build this view.

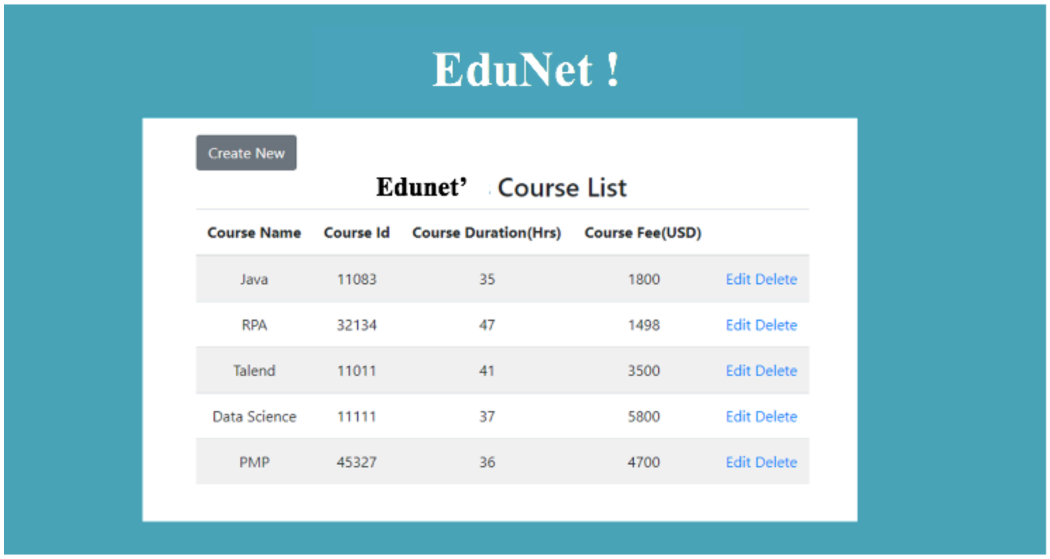
As you can see in the screenshot, the page will be containing four input fields and two buttons. One button will be submitting details filled in by the user into the database and the second button will display the complete list of courses available in the database. In order to make sure that the inputs are correct, you need to add some validations as well. Once done, you will be able to see this view using ‘/course’ URL. Below is the code, you will be needing to create **courseAddEdit.hbs** file:

courseAddEdit.hbs

```
<!-- Obtaining value from the course controller -->

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

<form action="/course" method="POST">

<input type="hidden" name="_id" value="{{course._id}}">

<div class = "form-group">

<label>Course Name</label>

<input type="text" maxlength="100" class="form-control" name="courseName"
placeholder="Course Name" value="{{course.courseName}}">

<div class="text-danger">

{{course.courseNameError}}</div>

</div>

<div class = "form-group">

<label>Course ID</label>

<input type="number" min='10000' max='99999' class="form-control" name="courseId"
placeholder="Course Id" value="{{course.courseId}} " required>

</div>

<div class = "form-row">

<div class = "form-group col-md-6">

<label>Course Duration</label>
```

```
<input type="number" min='10' max='99' class="form-control" name="courseDuration"
placeholder="Course Duration (Hrs)" "{{course.courseDuration}} " required>

</div>

<div class = "form-group col-md-6">

<label>Course Fee</label>

<input type="number" min='100' max="100000" class="form-control" name="courseFee"
placeholder="Course Fee (USD)" "{{course.courseFee}} " required>

</div>

</div>

<div class="form-group">

<button type="submit" class="btn btn-info"><i class="fa fa-database"></i>
Submit</button>

<a class="btn btn-secondary" href="/course/list"><i class="fa fa-list-alt"></i> View All
Courses</a>

</div>

</form>
```

Now, let me show you the next view i.e **list.hbs**, which will retrieve the complete list of available courses from the database and display them on your screen:



Course Name	Course Id	Course Duration(Hrs)	Course Fee(USD)	
Java	11083	35	1800	Edit Delete
RPA	32134	47	1498	Edit Delete
Talend	11011	41	3500	Edit Delete
Data Science	11111	37	5800	Edit Delete
PMP	45327	36	4700	Edit Delete

In this view, I am using a table to display the list of courses. This table will have five columns where the first four will display the course details while the last column will enable you to edit/delete a record directly from the application interface. The controllers of these functions have been already created in the **script.js** file. So, the only thing left is to add the view and in order to do so create a **list.hbs** file and type in the below-written code.

list.hbs

```
<div>

<a class="btn btn-secondary" href="/course"><i class="fa fa-plus"></i> Create New</a>

<h3 align="center">Edunet's Course List</h3>

</div>

<table class="table table-striped">

<thead>

<tr>

<th>Course Name</th>

<th>Course Id</th>
```

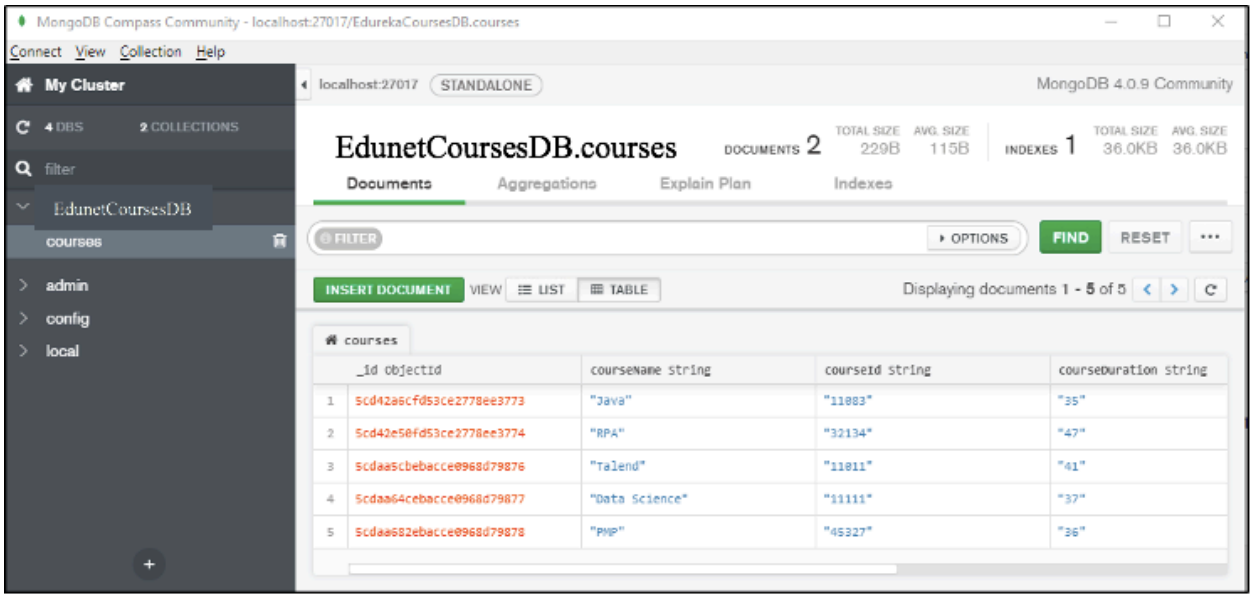
```
<th>Course Duration(Hrs)</th>

<th>Course Fee(USD)</th>

<th></th>

</tr>

</thead>

<tbody>

{{#each list}}

<tr align="center">

<td>{{this.courseName}}</td>

<td>{{this.courseId}}</td>

<td>{{this.courseDuration}}</td>

<td>{{this.courseFee}}</td>

<td>

<a href="/course/{{this._id}}"> Edit </a>

<a href="/course/delete/{{this._id}}" onclick="return confirm('Are you sure to delete this
record ?');"> Delete </a>

</td>

</tr>

{{/each}}

</tbody>

</table>
```

This concludes are the coding part, now it's time to test our application. For that, open the command prompt and navigate to the project folder or if you are using an IDE open the terminal and type in the below command to start the server.

nodemon script.js

Now you can launch your application in any browser at **http://localhost:8080**.

Once you have added your own data, you can go back in MongoDB and check whether the data has been added there or not. If you refer the below screenshot, you will see all my data has been successfully added. Which means my MongoDB is connected and working perfectly with my Node.js API.


MongoDB Compass Community - localhost:27017/EdurekaCoursesDB.courses
Connect View Collection Help
My Cluster
4 DBS 2 COLLECTIONS
filter
EdunetCoursesDB
courses
admin
config
local
localhost:27017 STANDALONE
MongoDB 4.0.9 Community
EdunetCoursesDB.courses
DOCUMENTS 2 TOTAL SIZE 229B AVG. SIZE 115B
INDEXES 1 TOTAL SIZE 36.0KB AVG. SIZE 36.0KB
Documents Aggregations Explain Plan Indexes
FILTER
OPTIONS
FIND
RESET
INSERT DOCUMENT
VIEW LIST TABLE
Displaying documents 1 - 5 of 5
courses

	_id ObjectId	courseName string	courseId string	courseDuration string
1	5cd42a6cfd53ce2778ee3773	"Java"	"11083"	"35"
2	5cd42e50fd53ce2778ee3774	"RPA"	"32134"	"47"
3	5cdaa5cbebacce0968d79876	"Talend"	"11011"	"41"
4	5cdaa64cebacce0968d79877	"Data Science"	"11111"	"37"
5	5cdaa682ebacce0968d79878	"PMP"	"45327"	"36"

Reference:

- https://hevodata.com/learn/node-js-with-mongodb-crud/
- https://www.edureka.co/blog/node-js-mongodb-tutorial/

Activity 2

Aim: Web Application Integration Using Express+Angular+Mongo

Learning outcome: Able to develop the real time scenarios based on Node JS applications.

Duration: 2 hours

List of Hardware/Software requirements:

1. Laptop/Computer with Windows OS / Linux OS - Ubuntu 18.04 LTS
2. NodeJS, ExpressJS, AngularJS, MongoDB software installed

Code/Program/Procedure (with comments):

Create Node.js App

First, we create a folder:

```
$ mkdir nodejs-express-mongodb
$ cd nodejs-express-mongodb
```

Next, we initialize the Node.js App with a **package.json** file:

```
npm init
name: (nodejs-express-mongodb)
version: (1.0.0)
description: Node.js Restful CRUD API with Node.js, Express and MongoDB
entry point: (index.js) server.js
test command:
git repository:
keywords: nodejs, express, mongodb, rest, api
author: bezkoder
license: (ISC)
Is this ok? (yes) yes
```

We need to install necessary modules: express, mongoose and cors.
Run the command:

npm install express mongoose cors –save

Setup Express web server
In the root folder, let’s create a new server.js file:

```
const express = require("express");
const cors = require("cors");
const app = express();
var corsOptions = {
  origin: "http://localhost:8081"
};
app.use(cors(corsOptions));
// parse requests of content-type - application/json
app.use(bodyParser.json());
// parse requests of content-type - application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: true }));
// simple route
app.get("/", (req, res) => {
  res.json({ message: "Welcome to bezkoder application." });
});
// set port, listen for requests
const PORT = process.env.PORT || 8080;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}.`);
});
```

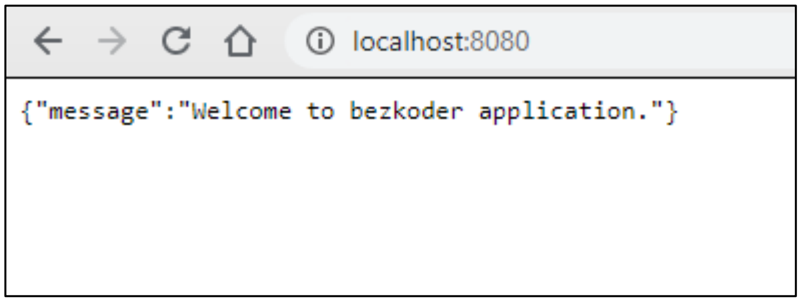
What we do are:
– import express and cors modules:

- Express is for building the Rest apis
- cors provides Express middleware to enable CORS with various options.

– create an Express app, then add body-parser (json and urlencoded) and cors

middlewares using app.use() method. Notice that we set origin: http://localhost:8081.
– define a GET route which is simple for test.
– listen on port 8080 for incoming requests.

Now let's run the app with command: node server.js.
Open your browser with url http://localhost:8080/, you will see:



```
{"message":"Welcome to bezkoder application."}
```

Configure MongoDB database & Mongoose

In the app folder, we create a separate config folder for configuration with db.config.js file like this:

```
module.exports = {
  url: "mongodb://localhost:27017/bezkoder_db"
};
```

Define Mongoose

We're gonna define Mongoose model (tutorial.model.js) also in app/models folder in the next step.

Now create app/models/index.js with the following code:

```
const dbConfig = require("../config/db.config.js");
const mongoose = require("mongoose");
mongoose.Promise = global.Promise;
const db = {};
```

```
db.mongoose = mongoose;
db.url = dbConfig.url;
db.tutorials = require("./tutorial.model.js")(mongoose);
module.exports = db;
```

Call connect() method in server.js:

...

```
const app = express();
app.use(...);
const db = require("./app/models");
db.mongoose
  .connect(db.url, {
    useNewUrlParser: true,
    useUnifiedTopology: true
  })
  .then(() => {
    console.log("Connected to the database!");
  })
  .catch(err => {
    console.log("Cannot connect to the database!", err);
    process.exit();
  });
```

Define the Mongoose Model

In models folder, create tutorial.model.js file like this:

```
module.exports = mongoose => {
  const Tutorial = mongoose.model(
    "tutorial",
    mongoose.Schema(
      {
        title: String,
        description: String,
      published: Boolean
    },
    { timestamps: true }
   )
  );
  return Tutorial;
};
```

This Mongoose Model represents tutorials collection in MongoDB database. These fields will be generated automatically for each Tutorial document: _id, title, description, published, createdAt, updatedAt, __v.

```
{
  "_id": "5e363b135036a835ac1a7da8",
  "title": "Js Tut#",
  "description": "Description for Tut#",
  "published": true,
  "createdAt": "2020-02-02T02:59:31.198Z",
  "updatedAt": "2020-02-02T02:59:31.198Z",
  "__v": 0
}
```

If you use this app with a front-end that needs id field instead of _id, you have to override toJSON method that map default object to a custom object. So the Mongoose model could be modified as following code:

```
module.exports = mongoose => {
  var schema = mongoose.Schema(
    {
      title: String,
      description: String,
      published: Boolean
    },
    { timestamps: true }
  );
  schema.method("toJSON", function() {
    const { __v, _id, ...object } = this.toObject();
    object.id = _id;
    return object;
  });
  const Tutorial = mongoose.model("tutorial", schema);
  return Tutorial;
};
And the result will look like this-

{
  "title": "Js Tut#",
  "description": "Description for Tut#",
  "published": true,
  "createdAt": "2020-02-02T02:59:31.198Z",
  "updatedAt": "2020-02-02T02:59:31.198Z",
  "id": "5e363b135036a835ac1a7da8"
}
```

After finishing the steps above, we don't need to write CRUD functions, Mongoose Model supports all of them:

create a new Tutorial: object.save()
find a Tutorial by id: findById(id)
retrieve all Tutorials: find()
update a Tutorial by id: findByIdAndUpdate(id, data)
remove a Tutorial: findByIdAndRemove(id)
remove all Tutorials: deleteMany()
find all Tutorials by title: find({ title: { $regex: new RegExp(title), $options: "i" } })
These functions will be used in our Controller.

Create the Controller

Inside app/controllers folder, let's create tutorial.controller.js with these CRUD functions:

- create

- findAll
- findOne
- update
- delete
- deleteAll
- findAllPublished

```
const db = require("../models");
const Tutorial = db.tutorials;
// Create and Save a new Tutorial
exports.create = (req, res) => {

};
// Retrieve all Tutorials from the database.
exports.findAll = (req, res) => {

};
// Find a single Tutorial with an id
exports.findOne = (req, res) => {

};
// Update a Tutorial by the id in the request
exports.update = (req, res) => {

};
// Delete a Tutorial with the specified id in the request
exports.delete = (req, res) => {

};
// Delete all Tutorials from the database.
exports.deleteAll = (req, res) => {

};
// Find all published Tutorials
exports.findAllPublished = (req, res) => {
 };
```

Run the Node.js Express Server

Run our Node.js application with command: **node server.js**

Angular 10 Front-end

- The App component is a container with router-outlet. It has navbar that links to routes paths via routerLink.
- TutorialsList component gets and displays Tutorials.
- Tutorial component has form for editing Tutorial's details based on :id.
- AddTutorial component has form for submission new Tutorial.
- These Components call TutorialService methods which use Angular HTTPClient to make HTTP requests and receive responses.

There are 3 components: tutorials-list, tutorial-details, add-tutorial.

- tutorial.service has methods for sending HTTP requests to the Apis.
- app-routing.module.ts defines routes for each component.
- app component contains router view and navigation bar.
- app.module.ts declares Angular components and import necessary modules.

Implementation

Setup Angular 10 Project

Let's open cmd and use Angular CLI to create a new Angular Project as following command:

```
ng new Angular10Crud
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? CSS
```

We also need to generate some Components and Services:

```
ng g s services/tutorial
ng g c components/add-tutorial
ng g c components/tutorial-details
ng g c components/tutorials-list
```

Set up App Module

Open app.module.ts and import FormsModule, HttpClientModule:

```
...
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/common/http';
@NgModule({
  declarations: [ ... ],
  imports: [
    ...
    FormsModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Define Routes for Angular AppRoutingModule

There are 3 main routes:
– /tutorials for tutorials-list component
– /tutorials/:id for tutorial-details component
– /add for add-tutorial component

app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { TutorialsListComponent } from './components/tutorials-list/tutorials-
list.component';
```

```
import { TutorialDetailsComponent } from './components/tutorial-details/tutorial-
details.component';
import { AddTutorialComponent } from './components/add-tutorial/add-
tutorial.component';
const routes: Routes = [
  { path: '', redirectTo: 'tutorials', pathMatch: 'full' },
  { path: 'tutorials', component: TutorialsListComponent },
  { path: 'tutorials/:id', component: TutorialDetailsComponent },
  { path: 'add', component: AddTutorialComponent }
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Create Data Service

This service will use Angular HTTPClient to send HTTP requests.
You can see that its functions includes CRUD operations and finder method.

services/tutorial.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
const baseUrl = 'http://localhost:8080/api/tutorials';
@Injectable({
  providedIn: 'root'
})
export class TutorialService {
  constructor(private http: HttpClient) { }
  getAll(): Observable<any> {
    return this.http.get(baseUrl);
  }
  get(id): Observable<any> {
    return this.http.get(`${baseUrl}/${id}`);
  }
  create(data): Observable<any> {
    return this.http.post(baseUrl, data);
  }
  update(id, data): Observable<any> {
    return this.http.put(`${baseUrl}/${id}`, data);
  }
  delete(id): Observable<any> {
    return this.http.delete(`${baseUrl}/${id}`);
  }
  deleteAll(): Observable<any> {
    return this.http.delete(baseUrl);
  }
  findByTitle(title): Observable<any> {
    return this.http.get(`${baseUrl}?title=${title}`);
  }
}
```

Create Angular Components

As you've known before, there are 3 components corresponding to 3 routes defined in ***AppRoutingModule***.

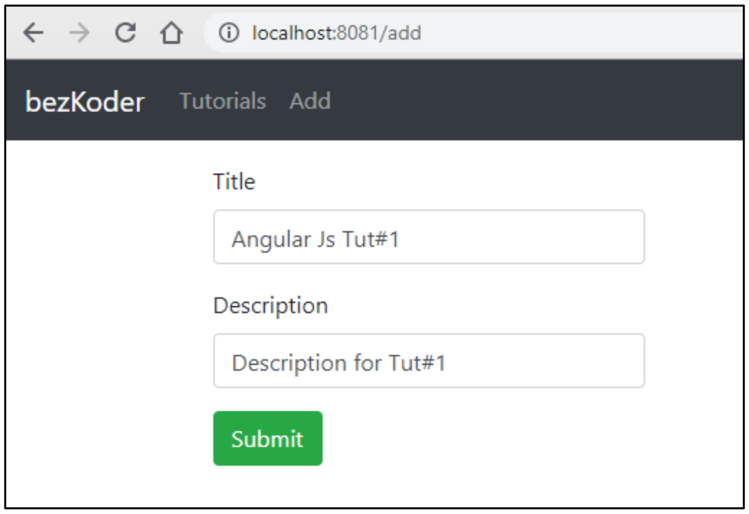
- Add new Item Component
- List of items Component
- Item details Component

Run the Angular App

You can run this App with command: ng serve --port 8081.
If the process is successful, open Browser with Url: http://localhost:8081/ and check it.

Output/Results snippet:

- ✓ Add an object:



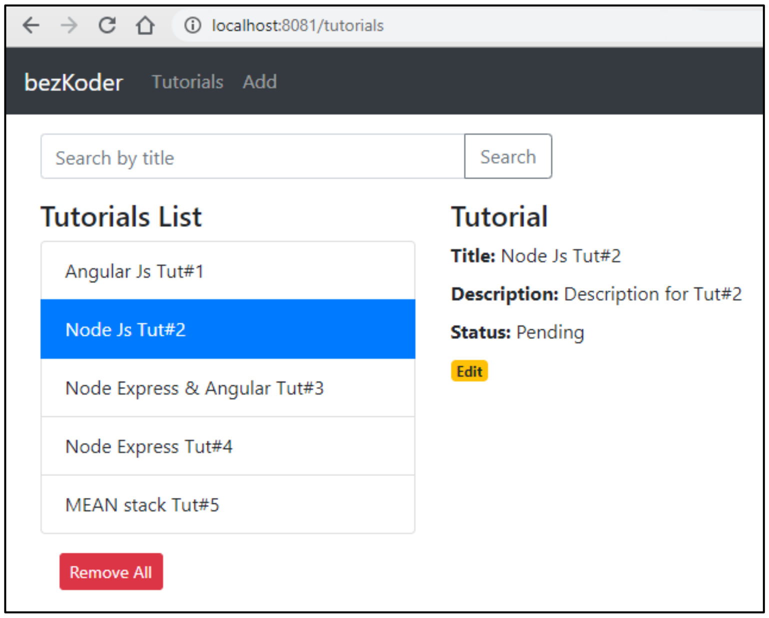
Title

Angular Js Tut#1

Description

Description for Tut#1

- ✓ Retrieve all objects:


localhost:8081/tutorials

Tutorials List

- Angular Js Tut#1
- Node Js Tut#2
- Node Express & Angular Tut#3
- Node Express Tut#4
- MEAN stack Tut#5

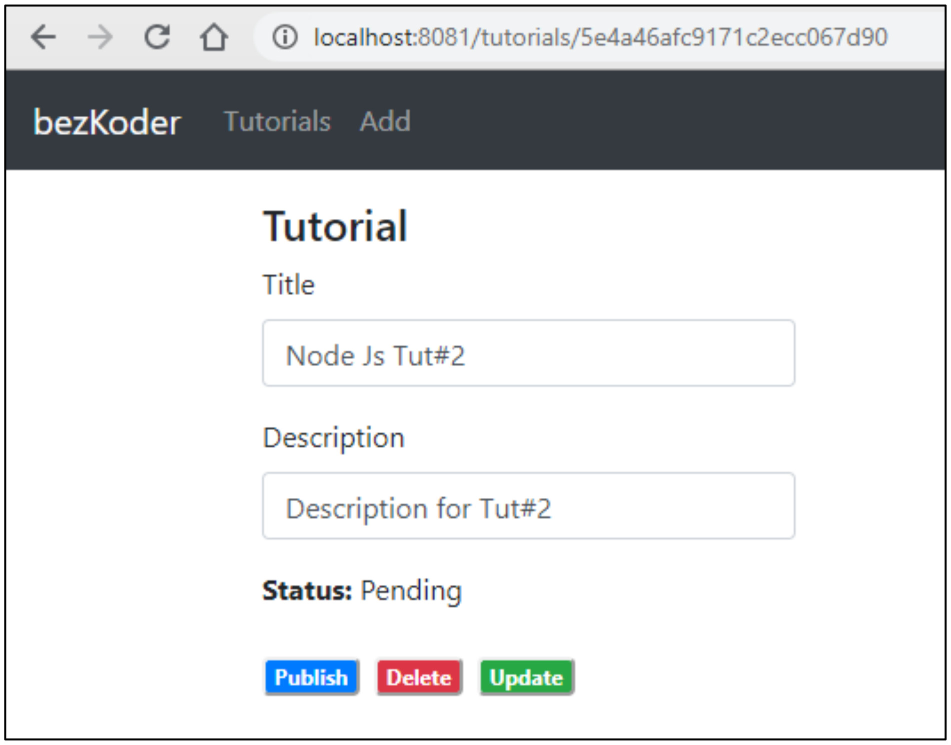
Tutorial

Title: Node Js Tut#2

Description: Description for Tut#2

Status: Pending

✓ Click on Edit button to update an object:


localhost:8081/tutorials/5e4a46afc9171c2ecc067d90

Tutorial

Title: Node Js Tut#2

Description: Description for Tut#2

Status: Pending

References:

1. https://www.bezkoder.com/angular-mongodb-node-express/