

# Web Application Integration Angular Express NodeJS & MongoDB



## Introduction of MEAN Stack

The MEAN stack is a popular web development stack that consists of four key technologies: MongoDB, Express.js, Angular, and Node.js.

### MongoDB [Database System]

A NoSQL database that stores data in flexible, JSON-like documents. It offers scalability and flexibility, making it ideal for handling large volumes of data and accommodating changes in data structure over time.

### Express.js [Back-end Framework]

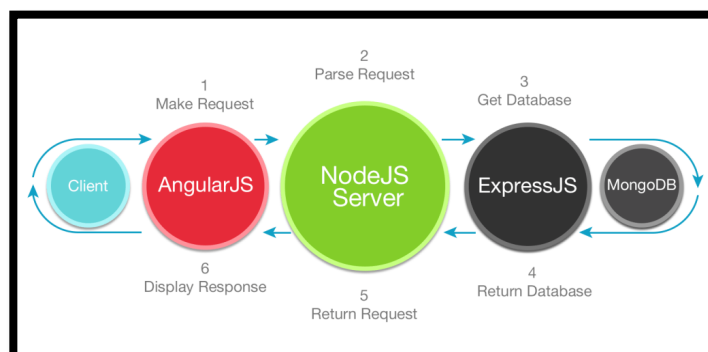
A lightweight web application framework for Node.js, Express.js simplifies the process of building server-side web applications and APIs.

### Angular [ Front-end Framework]

A front-end framework maintained by Google, Angular enables the creation of dynamic, single-page web applications.

### Node.js [Back-end Runtime Environment]

A server-side JavaScript runtime environment, Node.js allows developers to build scalable and high-performance applications.



Aim: Web Application integration using Angular Express MongoDB

### Setting Up the Environment:

We need to install Node.js, Angular CLI and MongoDB.

[How to Install Node JS & Angular CLI](#)

[How to Install MongoDB](#)

## Step 1: Create a new Angular project

Use the Angular CLI to create a new Angular project:

```
C:\Users\anshi\OneDrive\Desktop\Frontend angular>ng new main --no-standalone
? Which stylesheet format would you like to use? CSS
? Do you want to enable Server-Side Rendering (SSR) and Static Site Generation (SSG/Prerendering)? Yes
CREATE main/angular.json (3091 bytes)
CREATE main/package.json (1257 bytes)
CREATE main/README.md (1085 bytes)
CREATE main/tsconfig.json (936 bytes)
CREATE main/.editorconfig (290 bytes)
CREATE main/.gitignore (590 bytes)
CREATE main/tsconfig.app.json (342 bytes)
CREATE main/tsconfig.spec.json (287 bytes)
CREATE main/server.ts (1782 bytes)
CREATE main/.vscode/extensions.json (134 bytes)
```

If you use command:

```
ng new Folder_name --no-standalone --minimal --style=css
```

Then you create a short Angular project.

## Step 2: Create a new Express server

Create a new directory for the Express server:

```
mkdir my-app-server
```

then Navigate to this directory

```
cd my-app-server
```

### Initialize a new Node.js project:

The npm init command is used to initialize a new Node.js project, creating a package.json file that contains metadata about the project and its dependencies.

```
npm init -y
```

```
C:\Windows\System32\cmd.e X + v
Microsoft Windows [Version 10.0.22631.3155]
(c) Microsoft Corporation. All rights reserved.

C:\Users\anshi\OneDrive\Desktop\MEAN_STACK>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See 'npm help init' for definitive documentation on these fields
and exactly what they do.

Use 'npm install <pkg>' afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (mean_stack)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\Users\anshi\OneDrive\Desktop\MEAN_STACK\package.json:
{
  "name": "mean_stack",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}

Is this OK? (yes) y
C:\Users\anshi\OneDrive\Desktop\MEAN_STACK>
```

## Install Express and other necessary dependencies:

**npm install express mongoose cors body-parser**

**express:**        **npm install express**

**mongoose:**     **npm install mongoose**

**cors:**         **npm install cors**

**Body-Parser:**   **npm install body-parser**

- \* **Express:** Express is a minimal and flexible Node.js web application framework that provides a robust set of features for building web and mobile applications.
- \* **Mongoose:** Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js. It provides a straightforward schema-based solution to model application data.
- \* **CORS (Cross-Origin Resource Sharing):** CORS is a mechanism that allows web servers to specify which origins are permitted to access the resources on a server, thus enabling cross-origin HTTP requests.
- \* **Body-parser:** Body-parser is a middleware for Express.js that extracts the entire body portion of an incoming request stream and exposes it on req.body as something easier to interface with.

```
C:\Users\anshi\OneDrive\Desktop\Activity3\Activity3\server>npm install express mongoose body-parser cors
added 86 packages, and audited 87 packages in 2s

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

found 0 vulnerabilities
```

## Now Let's create a Project

STEP 1: Open your text editor then open your express Project here you need create a file **server.js**.

In this file we can create our own server, connection to MongoDB, Add our angular application using domain.

### Server.js

```
const express = require('express');
const mongoose = require('mongoose');
const bodyParser = require('body-parser');
const cors = require('cors'); // Import the cors package

const app = express();
const PORT = process.env.PORT || 3000;

// Connect to MongoDB database
mongoose.connect('mongodb://localhost:27017/abc', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

// Define a schema for your data
const formDataSchema = new mongoose.Schema({
  name: String,
  email: String,
  message: String,
});

const FormData = mongoose.model('FormData', formDataSchema);

// Middleware for parsing JSON data
app.use(bodyParser.json());

// Middleware for enabling CORS
app.use(cors({ origin: "http://localhost:4200" }));
// Allow requests from all origins

// Route to handle form submission
app.post('/', async (req, res) => {
  try {
    const { name, email, message } = req.body;

    // Check if email already exists
    const existingFormData = await FormData.findOne({ email });
```

```

    if (existingFormData) {
      return res.status(400).json({ message: 'Email already exists' });
    }

    // Create a new document using the FormData model
    const formData = new FormData({
      name,
      email,
      message,
    });

    // Save the form data to the database
    await formData.save();

    res.status(201).json({ message: 'Form data saved successfully!' });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Internal server error' });
  }
});

// Start the server
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});

```

## STEP 2: Create an angular App using Angular CLI

**Command:** `ng new App_name --no-standalone --minimal --style=css`

When we create this app we don't allow SSR (Server side rendering).

After creating the application, we need to create a component

**Command:** `ng g c form`

After that we need to create service

**Command:** `ng g s data`

Open your angular application on VS Code

Open index.html in root folder then add bootstrap CDN link

```

<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet"
  integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTWfSpd3yD65VohhpuuCOMLASjC"
crossorigin="anonymous">

```

```
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min
.js"
integrity="sha384-
MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
crossorigin="anonymous"></script>
```

## Open data.service.ts to Implement the service

Here we connect our express api and collection

Use of observable module store our data into the collection

**RxJS** is a library for reactive programming that uses Observables to make it easier to compose asynchronous or callback-based code.

## data.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class DataService {
  private readonly apiUrl = 'http://localhost:3000/'; // Assuming your server
  is running on the same host as your Angular app

  constructor(private http: HttpClient) { }

  submitForm(formData: any): Observable<any> {
    return this.http.post<any>(this.apiUrl, formData);
  }
}
```

## Open form.component.ts

Here we need to create a form and handle form submission and the POST request.

## form.component.ts

```
import { Component, EventEmitter, Input, OnInit, Output } from
 '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { BehaviorSubject } from 'rxjs';
```

```

import { DataService } from '../data.service';

@Component({
  selector: 'app-form',
  template: `
    <form class="custom-form" autocomplete="off" [formGroup]="myForm"
    (ngSubmit)="submitForm()">
      <div class="form-group">
        <input class="form-control" type="text" id="name"
        formControlName="name" placeholder="Name" required>
        <div *ngIf="name?.invalid && (name?.dirty || name?.touched)"
        class="alert alert-danger">
          <div *ngIf="name?.errors?.['required']">Name is required</div>
          <div *ngIf="name?.errors?.['minlength']">Name must be at least 3
characters long</div>
        </div>
      </div>

      <div class="form-group">
        <input class="form-control" type="email" id="email"
        formControlName="email" placeholder="Email" required>
        <div *ngIf="email?.invalid && (email?.dirty || email?.touched)"
        class="alert alert-danger">
          <div *ngIf="email?.errors?.['required']">Email is required</div>
          <div *ngIf="email?.errors?.['email']">Please enter a valid email
address</div>
        </div>
      </div>

      <div class="form-group">
        <textarea class="form-control" id="message"
        formControlName="message" placeholder="Message" required></textarea>
        <div *ngIf="message?.invalid && (message?.dirty ||
message?.touched)" class="alert alert-danger">
          <div *ngIf="message?.errors?.['required']">Message is
required</div>
        </div>
      </div>

      <button class="btn btn-primary" type="submit"
[disabled]="myForm.invalid">Submit</button>

      <!-- Success message -->
      <div *ngIf="successMessage" class="alert alert-success mt-3">
        {{ successMessage }}
      </div>
    </form>
  `,
  providers: [DataService],
})
export class AppFormComponent {
  myForm: FormGroup;
  successMessage: string;

  constructor(private dataService: DataService) {}

  ngOnInit(): void {
    this.myForm = new FormGroup({
      name: new FormControl(''),
      email: new FormControl(''),
      message: new FormControl(''),
    });
  }

  submitForm(): void {
    if (this.myForm.valid) {
      this.dataService.sendMessage(this.myForm.value).subscribe(
        (response) => {
          this.successMessage = response.message;
        },
        (error) => {
          console.error('Error submitting form:', error);
        }
      );
    }
  }
}

```

```

styles: [
  `
    .custom-form {
      max-width: 400px;
      margin: auto;
    }

    .form-group {
      margin-bottom: 1rem;
    }
  `
]
}))

export class FormComponent implements OnInit {
  @Input() initialState: BehaviorSubject<any> = new BehaviorSubject({});
  @Output() formValuesChanged = new EventEmitter<any>();
  @Output() formSubmitted = new EventEmitter<any>();

  myForm!: FormGroup;
  successMessage: string = '';

  constructor(private fb: FormBuilder, private dataService: DataService) {}

  get name() { return this.myForm?.get('name'); }
  get email() { return this.myForm?.get('email'); }
  get message() { return this.myForm?.get('message'); }

  ngOnInit(): void {
    this.myForm = this.fb.group({
      name: ['', [Validators.required, Validators.minLength(3)]],
      email: ['', [Validators.required, Validators.email]],
      message: ['', Validators.required]
    });

    this.initialState.subscribe(data => {
      this.myForm.patchValue(data);
    });

    this.myForm.valueChanges.subscribe(val => {
      this.formValuesChanged.emit(val);
    });
  }

  submitForm(): void {
    if (this.myForm?.valid) {
      this.dataService.submitForm(this.myForm.value).subscribe(
        (response) => {
          console.log('Form submitted successfully:', response);
        }
      );
    }
  }
}

```



```

        this.successMessage = 'Form submitted successfully!';
        this.formSubmitted.emit(response);
    },
    (error) => {
        console.error('Error submitting form:', error);
    }
  );
}
}
}
}

```

## Open module.ts

Here we need to import some modules HttpClient, ReactiveForms, and Services

## app.module.ts

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http'; // Import
HttpClientModule
import { ReactiveFormsModule } from '@angular/forms'; // Import
ReactiveFormsModule

import { AppComponent } from './app.component';
import { FormComponent } from './form/form.component';
import { DataService } from './data.service'; // Import your service

@NgModule({
  declarations: [
    AppComponent,
    FormComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule, // Add HttpClientModule to imports
    ReactiveFormsModule // Add ReactiveFormsModule to imports
  ],
  providers: [
    DataService // Provide your DataService
  ],
  bootstrap: [AppComponent]
})

```

```
})  
export class AppModule { }
```

## Open routing.ts

here we need to define our routes form access a form

## app.routing.module.ts

```
import { NgModule } from '@angular/core';  
import { RouterModule, Routes } from '@angular/router';  
import { FormComponent } from '../form/form.component'; // Import your form  
component  
  
const routes: Routes = [  
  { path: 'form', component: FormComponent }, // Route to display the form  
  // Add more routes as needed  
  { path: '', redirectTo: '/form', pathMatch: 'full' }, // Redirect to the  
form component by default  
];  
  
@NgModule({  
  imports: [RouterModule.forRoot(routes)],  
  exports: [RouterModule]  
})  
export class AppRoutingModule { }
```

## Run the server

Firstly, run the backend server

**Command: nodemon server.js**

```
C:\Users\anshi\OneDrive\Desktop\Activity3\Activity3\server>nodemon server.js  
[nodemon] 3.0.3  
[nodemon] to restart at any time, enter 'rs'  
[nodemon] watching path(s): *.*  
[nodemon] watching extensions: js,mjs,cjs,json  
[nodemon] starting 'node server.js'  
(node:10624) [MONGODB DRIVER] Warning: useUrlParser is a deprecated option: useUrlParser has no effect since Node.  
js Driver version 4.0.0 and will be removed in the next major version  
(Use 'node --trace-warnings ...' to show where the warning was created)  
(node:10624) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since  
Node.js Driver version 4.0.0 and will be removed in the next major version  
Server is running on http://localhost:3000  
[nodemon] restarting due to changes...  
[nodemon] starting 'node server.js'  
(node:11636) [MONGODB DRIVER] Warning: useUrlParser is a deprecated option: useUrlParser has no effect since Node.  
js Driver version 4.0.0 and will be removed in the next major version  
(Use 'node --trace-warnings ...' to show where the warning was created)  
(node:11636) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since  
Node.js Driver version 4.0.0 and will be removed in the next major version  
Server is running on http://localhost:3000
```

Then run the Angular application

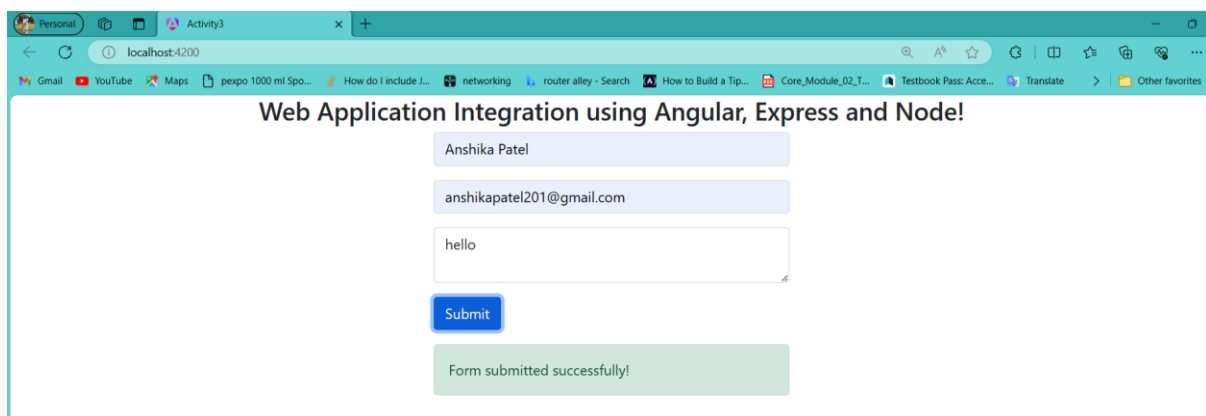
**Command: ng serve**

```
npm
Microsoft Windows [Version 10.0.22631.3155]
(c) Microsoft Corporation. All rights reserved.

C:\Users\anshi\OneDrive\Desktop\Activity3\Activity3>ng serve
- Building...
Initial chunk files | Names | Raw size
polyfills.js | polyfills | 83.60 kB |
main.js | main | 11.40 kB |
styles.css | styles | 95 bytes |

| Initial total | 95.09 kB
Application bundle generation complete. [3.162 seconds]
Watch mode enabled. Watching for file changes...
→ Local: http://localhost:4200/
→ press h + enter to show help
```

Go to this link then you will redirected to your form then enter the values then click on submit



Web Application Integration using Angular, Express and Node!

Anshika Patel

anshikapatel201@gmail.com

hello

Submit

Form submitted successfully!

Then go to MongoDB Compass your data will be stored successfully

abc.formdatas

Documents

Aggregations

Schema

Indexes

Validation

Filter

Type a query: { field: 'value' } or [Generate query](#)

ADD DATA

EXPORT DATA

UPDATE

DELETE

```
_id: ObjectId('65e1602076708c699ecd8216')
name: "Anshika Patel"
email: "anshikapatel201@gmail.com"
message: "hello"
__v: 0
```