1. Node Installation
2. Angular CLI installation

**npm install -g @angular/cli**

1. Create and open project

**ng new my-app**

(On CLI) **cd my-app**

**ng serve**

1. IDE installation

Visual Studio Code

1. Configure VS Code for MEAN

Angular Essentials (John Papa)

Material Icon Theme (Philipp Kief)

1. Adding first Component

Create all .ts, .html, .css, .spec.ts manually

or through Angular CLI

**ng generate component NewComponent**

1. Listen to Event

**(event)=”function()”**

1. Output Content

String Interpolation - (In html) **{{ content }}**

Property binding – **[property]=”value”**

1. Get Input

Local Reference-

**#element** (Used inside an element to create a marker)

(Pass element) **(event)=”function(element”)**

Access element anywhere in typescript

Two-Way Binding-

(In app.module.ts) **import {FormsModule} from ‘@angular/forms’**

**imports: [**

**FormsModule**

**],**

(In html element) **[(ngDirective)]=”variable”**

1. Install Angular Material

**npm install –save @angular/material**

OR

**ng add @angular/material**

1. Material on input elements

(In app.module.ts)

**import { MatInputModule } from '@angular/material';**

**imports: [**

**MatInputModule**

**],**

(In html)

**<mat-form-field>**

**<input matInput type="text">**

**</mat-form-field>**

1. Put everything in mat-card

**<mat-card>**

<!-- All Content -->

**</mat-card>**

1. Add CSS

(In .ts file)

**styleUrls: ['./css-file.css']**

(Put all css content in css-file.css

1. Add Toolbar

**import { MatToolbarModule } from '@angular/material';**

**<mat-toolbar></mat-toolbar>**

1. Add Expansion Panel

**import { MatExpansionModule } from '@angular/material';**

**<mat-accordion>**

**<mat-expansion-panel>**

**<mat-expansion-panel-header>**

**</mat-expansion-panel-header>**

**</mat-expansion-panel>**

**</mat-accordion>**

1. Add host CSS

**:host{}**

1. Use Structural Directive for outputting list

(Here list is posts)

<mat-expansion-panel **\*ngFor='let post of posts'**>

<mat-expansion-panel-header>

<p>{{ **post.title** }}</p>

</mat-expansion-panel-header>

<p>{{ **post.content** }}</p>

<mat-expansion-panel>

1. Use Structural directive for conditional outputting

<mat-accordion **\*ngIf="posts.length>0"**>

</mat-accordian>

<p **\*ngIf="posts.length==0"**>No posts added!!!</p>

1. Create a model for the data structure being used

**export interface Data{}**

1. Save posts coming from one module in another module

Using EventEmitters -

(In Output File .ts)

**import {EventEmitter, Output} from ‘@angular/core’**

**@Output() outputemitter = new EventEmitter<Data>()**

**Outputemitter.emit(output)**

(In parent component .ts)

**pushList(output) { outputlist.push(output) }**

(In parent component .html)

**(outputemitter)=”pushList($event)”** ($event gives access to data emitted)

(In Input File .ts)

**import {EventEmitter, Input} from ‘@angular/core’**

**@Input() input**

(In parent component .html)

**[input]= “outputlist”**

1. Add Forms

(In .html)

**<form (submit)=”onSubmit(newForm)” #newForm=”ngForm”>**

**<input type=”text” ngModel**

**name=”textInput” required minlength=”1”>**

<!—ngModel is removed from two-way binding -->

**</form>**

(In .ts)

**onSubmit(form: NgForm){**

**if (form.invalid) return**

**variable = form.textInput**

**}**

1. Save posts coming from one module in another module (New Method)

Using Service/Observable

(In .service.ts)

**export class NewService {**

**private output;**

**getOutput(){**

**return […this.output]}**

//All other getter and setter methods

**}**

Pass this to Angular

Using Providers directly

(In app.module.ts)

import service

**Providers: [NewService]**

Using Injectable

**import {Injectable} from ‘@angular/core’**

**@Injectable({providedIn: ‘root’})**

**export class NewService{}**

1. Use the Service

(In NewServie file)

**import {Subject} from ‘rxjs’**

**updatedOutput = new Subject<Data[]>()**

**addOutput(){**

**this.updatedOutput.next([…this.output])**

**}**

(Add UpdateListener for the output)

**getOutputUpdateListener(){**

**return this.updatedOutput.asObservable()**

**}**

(In Input file .ts)

(Make the class implement OnInit, OnDestroy from @angular/core)

**import {Subscription} from ‘rxjs’**

**newSub: Subscription**

**ngOnInit(){**

**input = newService.getOutput();**

**newSub = this.newService.getOutputUpdateListener()**

**.subscribe(funcNext, funcError, funcComplete)**

**}**

**ngOnDestroy(){**

**this.newSub.unsubscribe()**

**}**

(Get rid of all the EventEmitters and related code if you want to replace them with this Service)

(In Output file .ts)

**newService.addOutput()**

1. Generate backend

Create separate folder

1. Create separate server file

Create server.js file for backend

1. Import required packages and create server

**Const http = require(‘http’)**

**const server = http.createServer((req, res)=>{})**

**server.listen(process.env.PORT || portnum)**

1. Add express

(In cmd)

**npm install –save express**

(In express file)

**const express = require(‘express’)**

**const app = express()**

**app.use((req,res,next)={})** //next() used to shift to next app.use

//app.use will stop only when the response sends something to the server

1. Export express app and use it in server

**module.exports=app**

1. Set the port for the imported express app

**const app = require(‘app’)**

**app.set(‘port’, portnum)**

1. Update server side code with the resource material code
2. Install nodemon for better easier server development

**npm install --save-dev nodemon**

1. Change server location for some code

**app.use(‘path’, (req,res,next)=>{})**

1. Get data in json format

**res.status(statusnum).json(output)**

1. Use Angular Http for getting output

**import {HttpClientModule} from ‘@angular/common/forms’**

**http: HttpClient;**

**this.http.get<Data>(‘serverpath’)**

**.subscribe(funcNext, funcError, funcComplete);**

1. Handle Cross-Origin Resource Sharing (CORS)

**app.use((req, res, next)=>{**

**res.setHeader("Access-Control-Allow-Origin", "\*");**

**res.setHeader("Access-Control-Allow-Headers",**

**"Origin, X-Requested-With, Content-Type, Accept");**

**res.setHeader("Access-Control-Allow-Methods",**

**"GET, POST, PATCH, DELETE, OPTIONS");**

**next();**

**});**

1. Post data to backend

**app.post(‘path’, (req, res, next)=>{})**

1. Use body-parser for parsing request data

**npm install –save body-parser**

**const bodyParser = require(‘body-parser’)**

1. Read data read through the body field added by body-parser

**app.post(‘path’, (req, res, next)=>{ const data = req.body**

**})**

1. Connect angular to app.post

**this.http.post<extradata>(‘path’, data)**

**.subscribe(funcNext, funcError, funcComplete)**

1. Set up MongoDB

Create account in MongoDB Atlas and build new cluster

Add new users

Add IP whitelist (make sure you add your computer’s IP address)

1. Add Mongoose

**npm install –save mongoose**

1. Create Mongoose schema

**const mongoose = require(‘mongoose’)**

**const schema = mongoose.Schema({})**

**module.exports = mongoose.model(‘Schema’, schema)**

1. Use Schema instance

**const Schema = require(‘schemapath’)**

**const data = new Schema({})**

1. Connect Express App to MongoDB

Connect Application using instructions in the MongoDB Atlas

**const mongoose = require(‘mongoose’)**

Connect through mongoose and use promise (by then)

**mongoose.connect(‘path’)**

**.then()**

**.catch()**

1. Store data in database

**data.save()**

1. Fetch data in database

**Schema.find()**

**.then()**

1. Transforming response data

Use pipe

**this.http.get<Data>(‘serverpath’)**

**.pipe(map())**

**.subscribe(funcNext, funcError, funcComplete);**

1. Delete documents

**app.delete(‘path:id’, (req, res, next)=>{**

**Schema.deleteOn(id: req.params.id)**

**});**

1. Update frontend for deletion