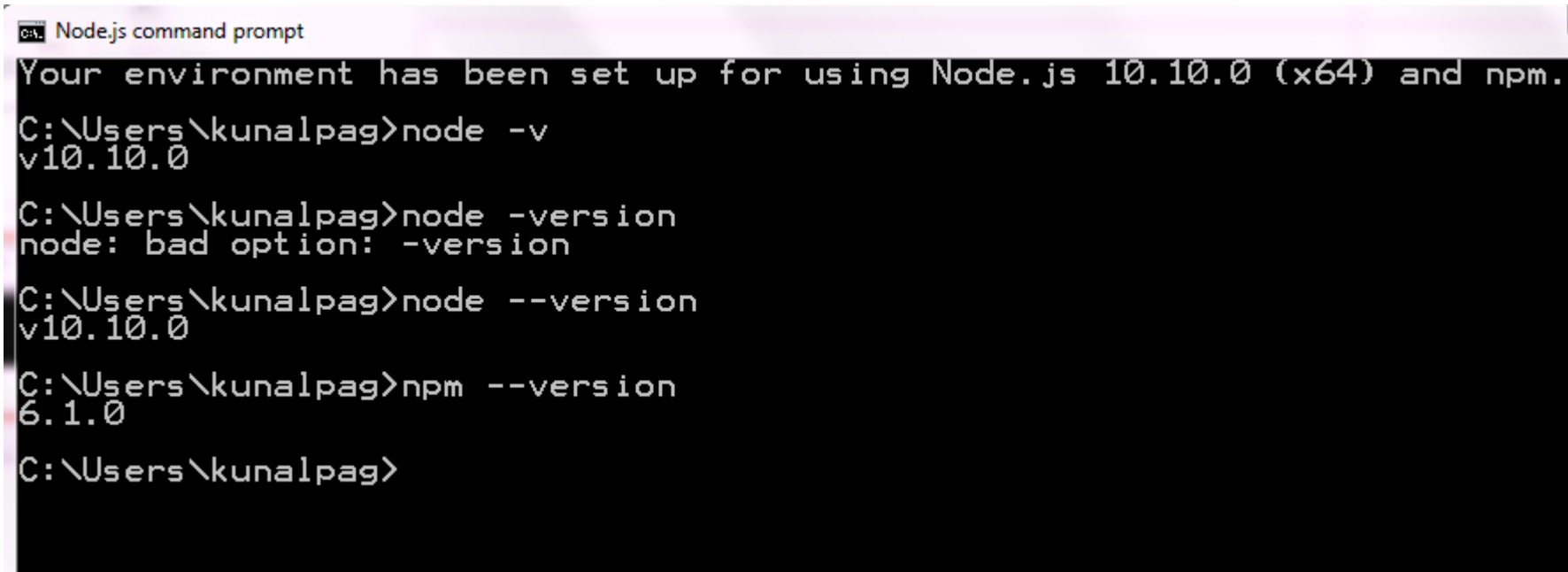


Download Node JS

<https://nodejs.org/en/download/> download LTS version

Note : When you install Node.js, make sure you select the option to add the Node.js executables to the path.

Open Node JS command prompt and Check Node Version



```
Node.js command prompt
Your environment has been set up for using Node.js 10.10.0 (x64) and npm.
C:\Users\kunalpag>node -v
v10.10.0
C:\Users\kunalpag>node -version
node: bad option: -version
C:\Users\kunalpag>node --version
v10.10.0
C:\Users\kunalpag>npm --version
6.1.0
C:\Users\kunalpag>
```

Installing the angular-cli Package

The angular-cli package has become the standard way to create and manage Angular projects during development.

Install angular/cli using npm command
`npm install --g @angular/cli`

Or
`#sudo npm install --global @angular/cli`

Create first project using ng

```
C:\WebDev\ # ng new app-name
```

What it does?

- Creates a new directory “app-name”
- Downloads and installs Angular libraries and any other dependencies
- Installs and configures TypeScript
- Installs and configures Karma & Protractor (testing libraries)

To compile and serve the application on the default port 4200

```
C:\WebDev # cd app-name
```

```
C:\WebDev\app-name # ng serve
```

To open a browser tab automatically.

```
C:\WebDev\app-name # ng serve --open or $ ng serve --port 3000 --open
```

Configure the default HTTP host and port used by the development server with two command-line options

```
ng serve --host 0.0.0.0 --port 4201
```

- **e2e** – end to end test folder. Mainly e2e is used for integration testing
- **node_modules** – The npm package installed
- **src** – source file of a project

The Angular 6 app folder has the following **file** –

- **.angular.json** – It basically holds the project name, version of cli, etc.
- **.editorconfig** – This is the config file for the editor.
- **.gitignore** – A .gitignore file should be committed into the repository, in order to share the ignore rules with any other users that clone the repository.
- **karma.conf.js** – This is used for unit testing via the protractor. All the information required for the project is provided in karma.conf.js file.(optional)
- **package.json** – The package.json file tells which libraries will be installed into node_modules when you run npm install.
- **tsconfig.json** – This basically contains the compiler options required during compilation.
- **tslint.json** – This is the config file with rules to be considered while compiling.

- It contains the files described below. These files are installed by angular-cli by default.
- **app.module.ts** – If you open the file, you will see that the code has reference to different libraries, which are imported. Angular-cli has used these default libraries for the import – angular/core, platform-browser. The names itself explain the usage of the libraries.
- They are imported and saved into variables such as **declarations**, **imports**, **providers**, and **bootstrap**.

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

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

- **declarations** – In declarations, the reference to the components is stored. The AppComponent is the default component that is created whenever a new project is initiated.
- **imports** – This will have the modules imported as shown above. At present, BrowserModule is part of the imports which is imported from @angular/platform-browser.
- **providers** – This will have reference to the services created. The service will be discussed in a subsequent chapter.
- **bootstrap** – This has reference to the default component created, i.e., AppComponent.

app.component.css – write css structure over here. we have added the background color to the div as shown below `.divdetails{ background-color: #ccc; }`

app.component.html – html code will be available in this file.

app.component.spec.ts – These are automatically generated files which contain unit tests for source component.

app.component.ts – The class for the component is defined over here. You can do the processing of the html structure in the .ts file. The processing will include activities such as connecting to the database, interacting with other components, routing, services, etc.

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent { title = 'app'; }
```

- Assets

You can save your images, js files in this folder.

- Environment

This folder has the details for the production or the dev environment. The folder contains two files.

- environment.prod.ts
- environment.ts

Both the files have details of whether the final file should be compiled in the production environment or the dev environment.

The additional file structure of Angular 4 app folder includes the following

- favicon.ico

This is a file that is usually found in the root directory of a website.

- index.html

This is the file which is displayed in the browser.

```
<!doctype html>
<html lang = "en">
  <head>
    <meta charset = "utf-8">
    <title>HTTP Search Param</title>
    <base href = "/">
    <link href = "https://fonts.googleapis.com/icon?family=Material+Icons" rel="stylesheet">
    <link href = "https://fonts.googleapis.com/css?family=Roboto|Roboto+Mono" rel="stylesheet">
    <link href = "styles.c7c7b8bf22964ff954d3.bundle.css" rel="stylesheet">
    <meta name = "viewport" content="width=device-width, initial-scale=1">
    <link rel = "icon" type="image/x-icon" href="favicon.ico">
  </head>

  <body>
    <app-root></app-root>
  </body>
</html>
```

The body has **<app-root></app-root>**. This is the selector which is used in **app.component.ts** file and will display the details from app.component.html file.

main.ts

file from where we start our project development. It starts with importing the basic module which we need. Right now if you see angular/core, angular/platform-browser-dynamic, app.module and environment is imported by default during angular-cli installation and project setup.

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

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule);
```

Add bootstrap

Download bootstrap

cd todo

`npm install bootstrap@4.1.1`

```
"styles": [
```

```
  "src/styles.css",
```

```
  "node_modules/bootstrap/dist/css/bootstrap.min.css"
```

```
],
```

To-do App

Create model for items
Data Model : add a file called model.ts to the todo/src/app folder

```
export class Model {  
  user;  
  items;  
  
  constructor() {  
  
    this.user = "Kunal";  
    this.items = [new TodoItem("Buy B'day Gifts", false),  
                  new TodoItem("Buy Cake", false),  
                  new TodoItem("Order the food online", false)  
  ]  
}
```

```
export class TodoItem {  
  action;  
  done;  
  constructor(action, done) {  
    this.action = action;  
    this.done = done;  
  }  
}
```

Update app component

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

```
import { Model } from "../model";
```

```
@Component({
```

```
  selector: "todo-app",
```

```
  templateUrl: "app.component.html"
```

```
})
```

```
export class AppComponent {
```

```
  umodel = new Model();
```

```
  getName() {
```

```
    return this.model.user;
```

```
  }
```

```
}
```


Update app component hmtl

```
<h3 class="bg-primary p-1 text-white">{{ getName() }}'s To Do List</h3>
```

Display all the items to the page

Update AppComponent (app.component.ts) with getTodoItems() functions.

```
getTodoItems() {  
    return this.model.items;  
}
```

Update app.component.html

```
<h3 class="bg-primary p-1 text-white">{{ getName() }}'s To Do List</h3>
```

```
<table class="table table-striped table-bordered">
```

```
<thead>
```

```
<tr><th></th><th>Description</th><th>Done</th></tr>
```

```
</thead>
```

```
<tbody>
```

```
<tr *ngFor="let item of getTodoItems(); let i = index">
```

```
<td>{{ i + 1 }}</td>
```

```
<td>{{ item.action }}</td>
```

```
<td [ngSwitch]="item.done">
```

```
<span *ngSwitchCase="true">Yes</span>
```

```
<span *ngSwitchDefault>No</span>
```

```
</td>
```

```
</tr>
```

```
</tbody>
```

```
</table>
```

Kunal's To Do List

Add

Description	Done
Learn bootstrap	Yes
Attend Another lecture	No
Have a lunch	No
Sleep for hand an hr and study until night	No

Two-way binding

The template contains only *one-way data bindings*, which means they are used to display a data value but are unable to change it.

Angular also supports *two-way data bindings*, which can be used to display a data value and change it, too.

Task :

Update HTML

```
<h3 class="bg-primary p-1 text-white">{{getName()}}'s To Do List</h3>
<table class="table table-striped table-bordered">
  <thead>
    <tr><th></th><th>Description</th><th>Done</th></tr>
  </thead>
  <tbody>
    <tr *ngFor="let item of getTodoItems(); let i = index">
      <td>{{i + 1}}</td>
      <td>{{item.action}}</td>
      <td><input type="checkbox" [(ngModel)]="item.done" /></td>
      <td [ngSwitch]="item.done">
        <span *ngSwitchCase="true">Yes</span>
        <span *ngSwitchDefault>No</span>
      </td>
    </tr>
  </tbody>
</table>
```

Note : To work this, you need to use angular forms

Update app.modules.ts

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

```
import { FormsModule } from "@angular/forms";  
import { AppComponent } from './app.component';
```

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

Filtering Todo Item List

Kunal's To Do List

Add

	Description	Done	
1	Buy B'day Gifts	<input type="checkbox"/>	No
2	Book the hotel	<input type="checkbox"/>	No
3	Call Pooja	<input type="checkbox"/>	No

Tick on second action. It should get removed from list

Kunal's To Do List

Add

	Description	Done	
1	Buy B'day Gifts	<input type="checkbox"/>	No
2	Call Pooja	<input type="checkbox"/>	No

Filtering To-Do Items

The checkboxes allow the data model to be updated, and the next step is to remove to-do items once they have been marked as done.

app.component.ts

```
export class AppComponent {  
  model = new Model();  
  
  getName() {  
    return this.model.user;  
  }  
  
  getTodoItems() {  
    //return this.model.items;  
  
    // return this.model.items.filter(function (item) { return !item.done });  
    //or  
    return this.model.items.filter(item => !item.done);  
  }  
}
```


Adding To-Do Items

Allow the user to create new to-do items and store them in the data model.

Handle click event on Add button

Kunal's To Do List

Add

	Description	Done	
1	Buy B'day Gifts	<input type="checkbox"/>	No
2	Call Pooja	<input type="checkbox"/>	No

Update in html

```
<h3 class="bg-primary p-1 text-white">{{getName()}}'s To Do List</h3>
```

```
<div class="my-1">
```

```
  <input class="form-control" #todoText />
```

```
  <button class="btn btn-primary mt-1" (click)="addItem(todoText.value)">
```

```
    Add
```

```
  </button>
```

```
</div>
```

.....

The input element has an attribute whose name starts with the # character, which is used to define a variable to refer to the element in the template's data bindings.

The variable is called todoText, and it is used by the binding that has been applied to the button element.

Update app.component.html

```
import { Component } from "@angular/core";
import { Model, TodoItem } from "../model";
@Component({
  selector: "todo-app",
  templateUrl: "app.component.html"
})
export class AppComponent {
  model = new Model();
  getName() {
    return this.model.user;
  }
  getTodoItems() {
    return this.model.items.filter(item => !item.done);
  }
  addItem(newItem) {
    if (newItem !== "") {
      this.model.items.push(new TodoItem(newItem, false));
    }
  }
}
```

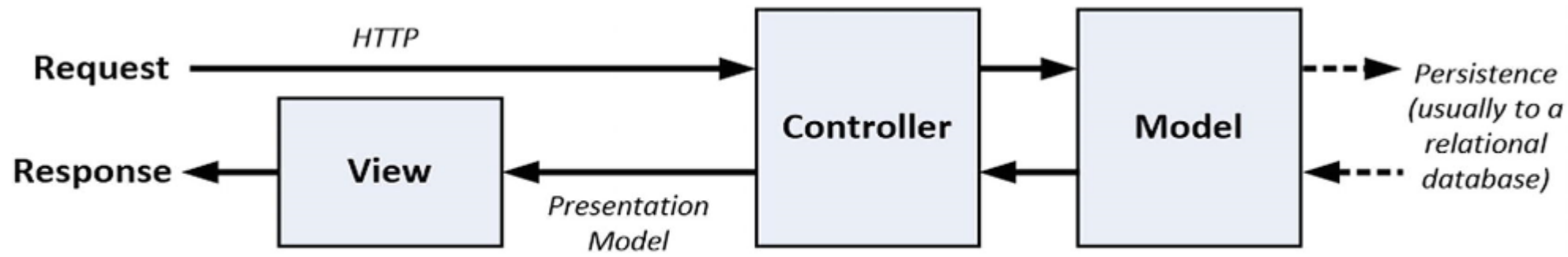
Understanding the MVC Pattern

The key to applying the MVC pattern is to implement the key premise of a *separation of concerns*, in which the data model in the application is decoupled from the business and presentation logic.

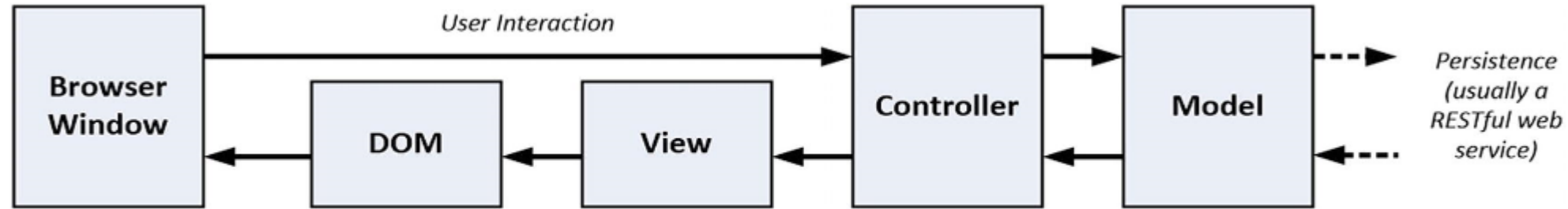
In client-side web development, this means separating the data, the logic that operates on that data, and the HTML elements used to display the data.

The result is a client-side application that is easier to develop, maintain, and test.

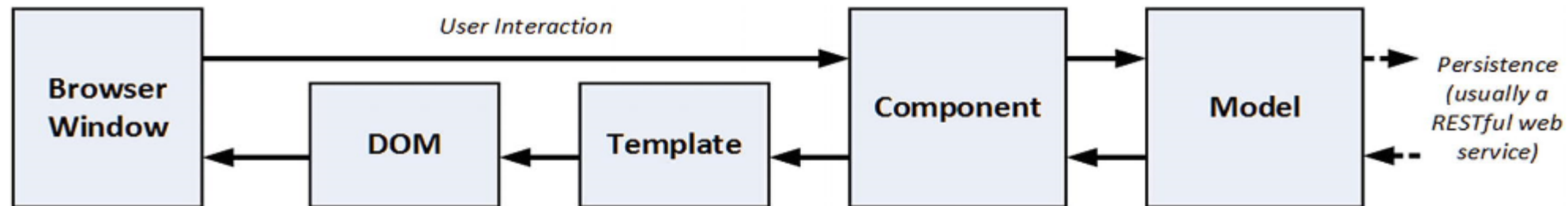
N



The server-side implementation of the MVC pattern



A client-side implementation of the MVC pattern



The Angular implementation of the MVC pattern

Services

- Why/What Service
- Creating the services
- Decorator @Injectable
- Inject Service into component

Why/What Service

Login/Registration page require code to validate the details entered by user.

You would need to call a function from various places.

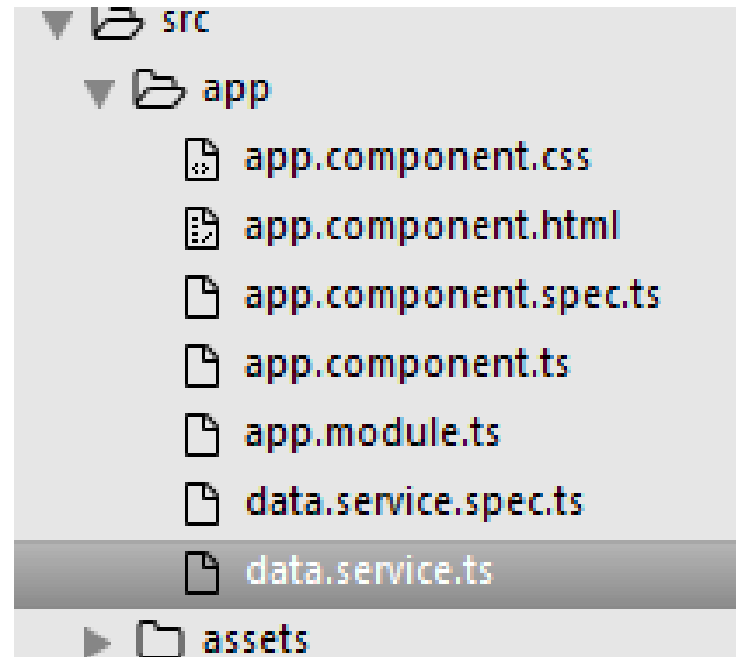
Services : Is singleton instance of a file/function which can be injected into multiple component.

Why : Component should not fetch the data, they should ask services to do those task.

Create a service

Inside project dir \$ng generate service data

Note : service does not have view



Return the record from the service

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

```
@Injectable({  
  providedIn: 'root'  
})
```

```
export class DataService {
```

```
  constructor() { }
```

```
  getServiceData(){  
    // write a code which will interact with the DB  
    return "Hello world";  
  }
```

```
  getFruitInfo(){  
    var values: Array<String> = ['Apple', 'Orange', 'Banana'];  
    return values;  
  }  
}
```

```
export class AppComponent {  
  // Make Service available  
  ngOnInit(){  
    this.record = .... ??? how  
  }  
}
```

Make service available

1) Make a Service as a common service to all component (update app.module.ts)

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

```
@NgModule({  
  declarations: [ AppComponent ],  
  imports: [ BrowserModule, ReactiveFormsModule  
],  
  providers: [DataService],  
  bootstrap: [AppComponent]  
})
```

Make service explicit to component

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

```
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
})  
export class AppComponent {  
  msg = "";  
  constructor(private mySrv:DataService){}  
  inject the service  
  ngOnInit(){  
    this.msg = this.mySrv.getServiceData();  
  }  
}
```

// dependency injection(Design pattern),
constrcutor(private mySrv:DataService)

Is same as

```
private mySrv :DataService  
Constrcutor(mySrv:DataService) {  
  this.mySrv= mySrv
```

Constructor and ngOnInit

- **The Constructor** is a default method of the class that is executed when the class is instantiated and ensures proper initialization of fields in the class and its subclasses.

```
new MyClass(someArg);
```

- **ngOnInit** is a life cycle hook called by Angular to indicate that Angular is done creating the component.

We have to import OnInit in order to use like this (actually implementing OnInit is not mandatory but considered good practice):

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

Note : ngOnInit() gets invoke after constructor.

HttpClient

- What is HttpClient
- Creating Service to make HTTP request
- Response
- RxJS

Import HttpClientModule

```
import {HttpClientModule} from '@angular/common/http'
```

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

Inject HttpClient Service into Component

```
import { HttpClient } from '@angular/common/http'
```

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

  record={}
  constructor(private mySrv:DataService, private http:HttpClient){} // dependency injection

  ngOnInit(){
    this.record = this.mySrv.getServiceData();
    let obs = this.http.get('https://api.github.com/users/kpagariya');
    obs.subscribe(()=> console.log('Response received'));
  }
}
```

Print the response received from REST API

```
export class AppComponent {
```

```
  record={}
```

```
  constructor(private mySrv:DataService,private http:HttpClient){}
```

```
  ngOnInit(){
```

```
    this.record = this.mySrv.getServiceData();
```

```
    let obs = this.http.get('https://api.github.com/users/kpagariya');
```

```
    obs.subscribe((response)=> console.log(response));
```

```
  }
```

```
}
```

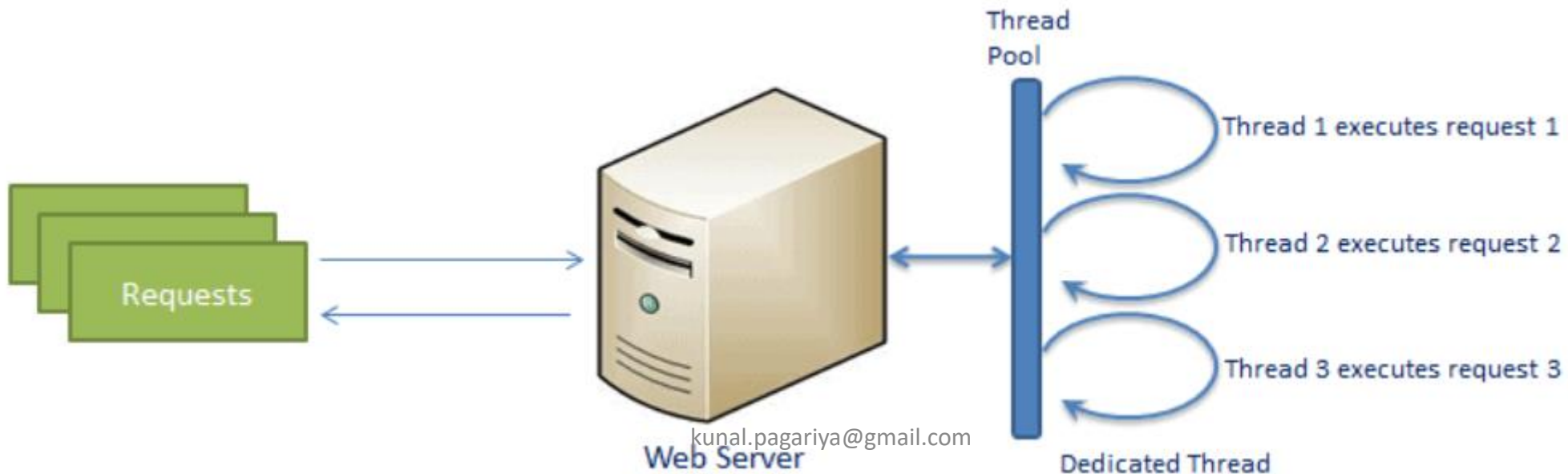

Node JS

Traditional Web Server Model

In the traditional web server model, each request is handled by a dedicated thread from the thread pool.

If no thread is available in the thread pool at any point of time then the request waits till the next available thread.

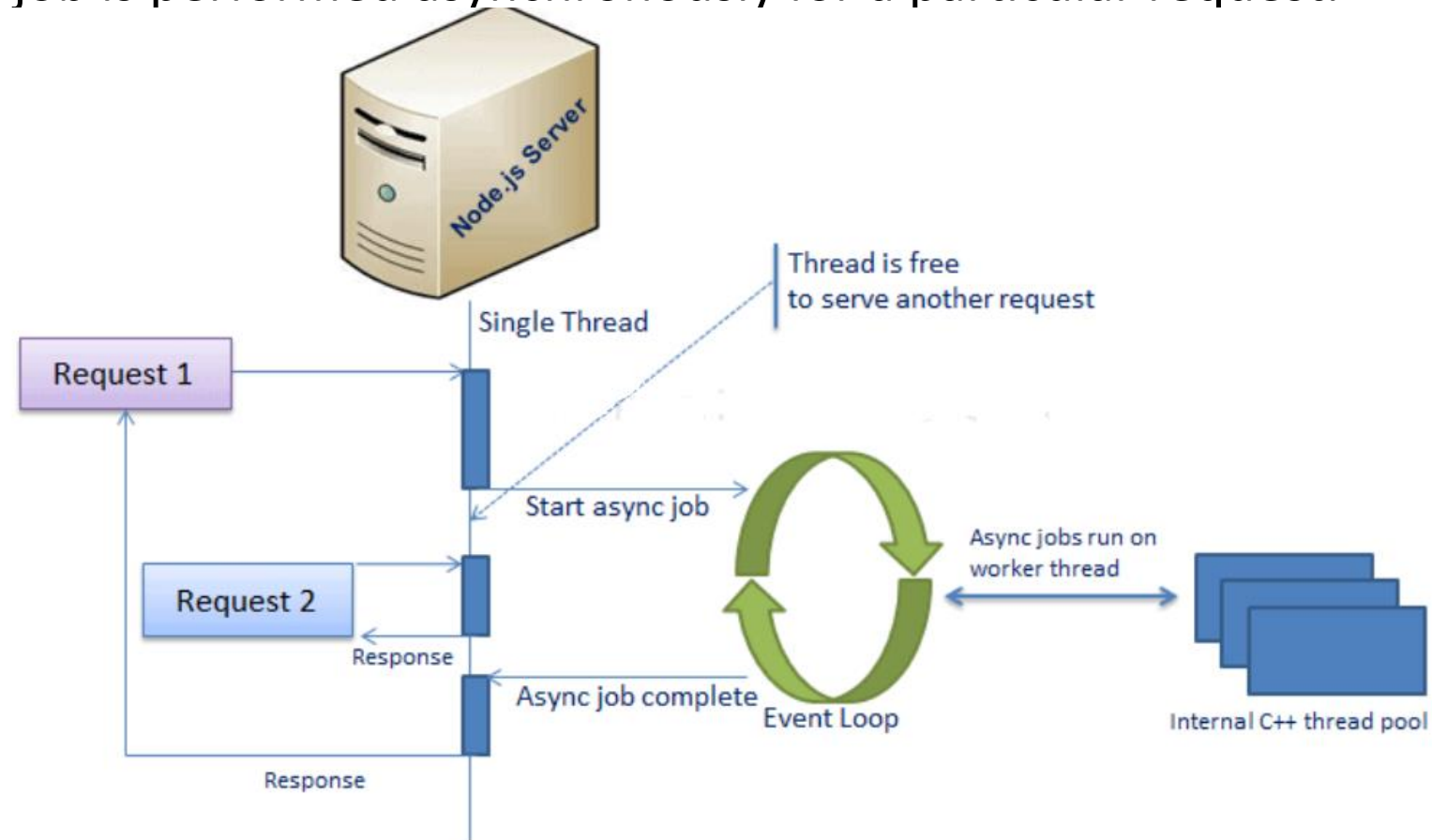
Dedicated thread executes a particular request and does not return to thread pool until it completes the execution and returns a response.



Node.js Process Model

Node.js runs in a single process and the application code runs in a single thread and thereby needs less resources than other platforms.

All the user requests to your web application will be handled by a single thread and all the I/O work or long running job is performed asynchronously for a particular request.



Node.js Process Model

Create project

- 1) Create a folder e.g rest-api and goto the folder from CMD
- 2) Create package.json using below command

C:\Users\node-js-projects\rest-api>**npm init**

And press enter enter....

```
Press ^C at any time to quit.  
package name: (rest-api)  
version: (1.0.0)  
description: First package for Development of REST API using node  
entry point: (index.js)  
test command:  
git repository:  
keywords:  
author: kunal  
license: (ISC)  
About to write to C:\Users\kunalpag\Documents\My_data\node-js-projects\rest-api\package.json:
```

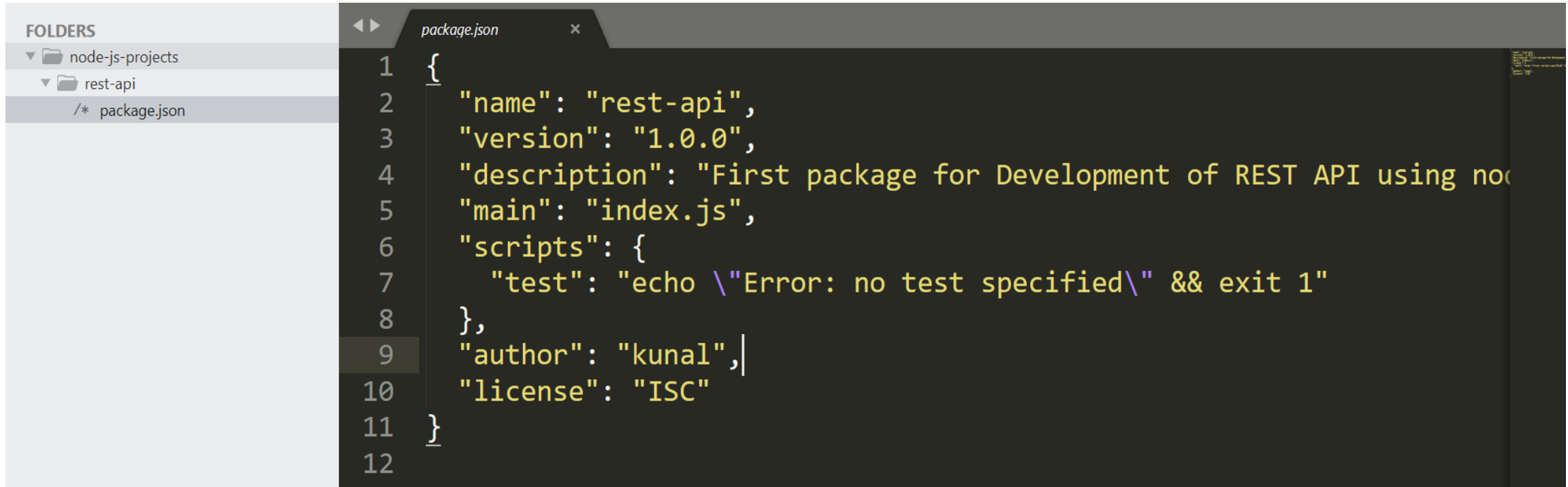
```
{  
  "name": "rest-api",  
  "version": "1.0.0",  
  "description": "First package for Development of REST API using node",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "author": "kunal",  
  "license": "ISC"  
}
```

Is this OK? (yes)

kunal.pagariya@gmail.com

C:\Users\kunalpag\Documents\My_data\node-js-projects\rest-api>

Package.



The image shows a code editor interface with a sidebar on the left and a main editor area on the right. The sidebar, titled 'FOLDERS', shows a tree structure with 'node-js-projects' expanded, containing a sub-folder 'rest-api'. Inside 'rest-api', the file 'package.json' is selected and highlighted. The main editor area shows the content of 'package.json' with line numbers 1 through 12. The JSON content is as follows:

```
1 {  
2   "name": "rest-api",  
3   "version": "1.0.0",  
4   "description": "First package for Development of REST API using node",  
5   "main": "index.js",  
6   "scripts": {  
7     "test": "echo \\\"Error: no test specified\\\" && exit 1"  
8   },  
9   "author": "kunal",  
10  "license": "ISC"  
11 }  
12
```

Install express

Express is one of the most popular lightweight web application frameworks for node. Gives us the webserver functionality.

Install express

```
C:\Users\kunalpag\Documents\My_data\node-js-projects\rest-api>npm install --save express
npm WARN registry Using stale data from https://registry.npmjs.org/ because the host is inaccessible -- are you offline?
npm WARN registry Using stale data from https://registry.npmjs.org/ due to a request error during revalidation.
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN rest-api@1.0.0 No repository field.

+ express@4.17.1
added 50 packages from 37 contributors in 4.217s

C:\Users\kunalpag\Documents\My_data\node-js-projects\rest-api>
```

```
Create a file app.js
var express = require('express');
var app = express();
```

```
app.get('/', function(req, res) {
  res.send('Hello World!');
});
```

```
app.listen(3000, function() {
  console.log('App is listening on port
3000!');
```

Go to CMD and Run : node app.js

Explanation

1) First two line : Require (import) the express module and create an [Express application](#).

app, has methods for routing HTTP requests, configuring middleware, rendering HTML views, registering a template engine, and modifying [application settings](#) that control how the application behaves.

2) The three lines starting with app.get shows a route definition. The app.get() method specifies a callback function that will be invoked whenever there is an HTTP GET request with a path ('/') relative to the site root.

3) The final block starts up the server on port '3000' and prints a log comment to the console. With the server running, you could go to localhost:3000 in your browser

Web Service-REST API Development in node js

Create index.js

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

```
const app=express();  
const port=3000;
```

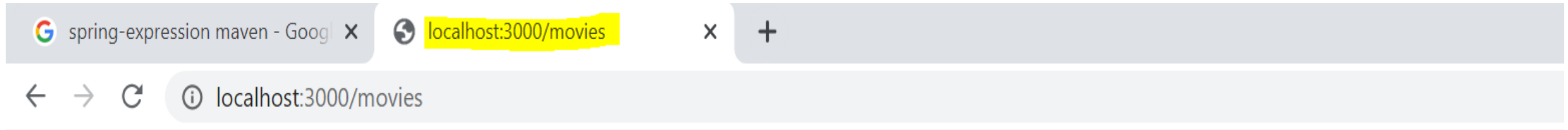
```
let movies=[  
  {  
    id:"1",  
    title:"Hera pheri",  
    hero:"Akshay"  
  },  
  {  
    id:"2",  
    title:"Sholay",  
    hero:"Amitabh"  
  }  
];
```

```
// 1st API to return list of movies as json  
app.get('/movies',(req,res) =>{  
  res.json(movies);  
});  
  
app.get('/', function(req, res) {  
  res.send('Sorry, Hit the URL with proper  
API name!');  
});  
  
app.listen(port, function() {  
  console.log('App is listening on  
port',port);  
});
```

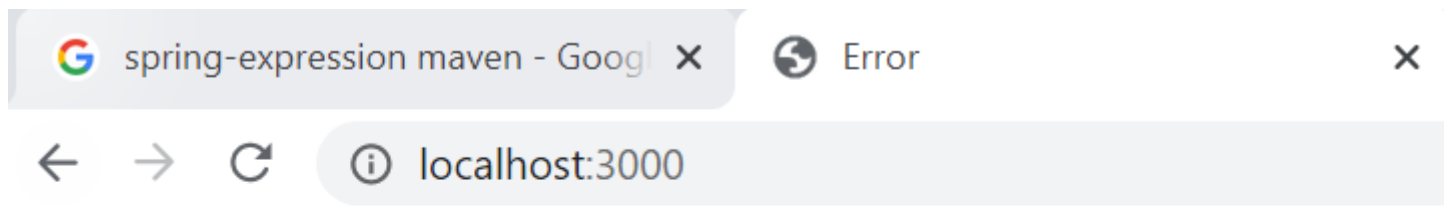
Run the application

C:\Users\node-js-projects\rest-api>node index.js

1) Hit the URL



```
[{"id": "1", "title": "Hera pheri", "hero": "Akshay"}, {"id": "2", "title": "Sholay", "hero": "Amitabh"}]
```



Cannot GET /

Add new movie

// Add new movie to the list

```
app.post("/movie",(req,res) => {  
    const new_movie=req.body  
    console.log(new_movie);  
    movies.push(new_movie);  
    res.send("Movie added....");  
});
```

Reach to kunal.Pagariya@gmail.com in case you face any issue/error during the development.