

PROFESSIONAL TRAINING 1 REPORT
at
Sathyabama Institute of Science and Technology
(Deemed to be University)

Submitted in partial fulfillment of the requirements for the award of Bachelor
of Engineering Degree in Computer Science and Engineering

By

AMAN GUPTA
REG. NO. 39120007



DEPARTMENT OF INFORMATION TECHNOLOGY

SCHOOL OF COMPUTING

SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600119, TAMILNADU

NOVEMBER 2021



SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

**Accredited with Grade “A” by
NAAC**

(Established under Section 3 of UGC Act, 1956)
JEPPIAAR NAGAR, RAJIV GANDHI SALAI
CHENNAI– 600119
www.sathyabama.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **AMAN GUPTA** (Reg. No: 39120007) who carried out the project entitled “**Digital Music Shopping System Website**” under my supervision from June 2021 to November 2021.

Internal Guide

Dr.S Gowri

Head of the Department

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I, **AMAN GUPTA** hereby declare that the project report entitled **Digital Music Shopping System Website** done by me under the guidance of **Dr. S GOWRI** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering Degree in Computer Science and Engineering.

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE

Acknowledgement

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E. , Ph.D, Dean, School of Computing, Dr. R. Subhashini, M.E., Ph.D. Head of the Department of Information Technology** for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. S GOWRI** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Information Technology** who were helpful in many ways for the completion of the project

TRAINING CERTIFICATE



CERTIFICATE OF EXCELLENCE

THIS IS TO CERTIFY THAT

Aman Gupta

has successfully completed
ANGULAR CERTIFIED TRAINING
Course Conducted During September 2021 - November 2021

Date: 07-11-2021



Authorised Signatory

#30, 3rd Main, Rajalakshmi Nagar, Velachery, Chennai - 42. Ph : +91-44-2245 5904 / 98844 12301, Web : www.credosystemz.com

ABSTRACT

This document presents about the Angular JS project entitled “**musiCart**” . Its aim is to formally create a music store Website Application.

This is a web page made using HTML, CSS, Java script, and Framework like Angular.

Everyone like music. Music is like a sound that impress everyone’s heart. it may feel and melt the heart easily It can lessen the stress, pain, struggle, distraction and bring positivity and calmness in our daily life.

This is web portal based automation project, which provides music items such as music album, singles. User can purchase their needed music songs anywhere, anytime just with few steps.

This project manages a music store. A user can add music albums to a shopping cart, edit the shopping cart, and filtering which music albums are in the shopping cart. This utilizes JavaScript framework AngularJS, incorporating components and pipe concepts.

This is an alternate and more efficient method of providing Music to all the Music lovers without waiting in queue in Music stores.

TABLE OF CONTENTS

CHAPTER No.	TITLE	PAGE No
	ABSTRACT	6
1	INTRODUCTION	
	1.1 Purpose and Motivation	9
2.	AIM AND SCOPE OF THE PRESENT INVESTIGATION	11
	2.1 AIM	10
	2.2 SCOPE	10
	2.3 ACCESSIBILITY	11
	2.4 ADVANTAGE	11
	2.5 FEASIBILITY STUDY	12
3.	EXPERIMENTAL OR MATERIALS AND METHODS, ALGORITHMS USED	13
	3.1 Software Required	13
	3.1.1 Windows 10	
	3.1.2 Node JS	
	3.1.3 Angular CLI	
	3.2 Hardware Required	17
	3.2.1 Hard Disk	
	3.2.2 RAM	
	3.2.3 Processor	
	3.2.4 Mouse	
	3.2.5 Keyboard	
	3.2.6 Speaker	
4.	TESTING	20
	4.1 System Testing	21
	4.2 Unit Testing	
	4.3 Integration testing	
5.	RESULTS AND DISCUSSION, PERFORMANCE Recommendation	22
	5.1 Result and discussion	22
	5.2 Recommendation	
	5.3 Screen shots	23
	5.3.1 Input	

	5.3.2 Output	
6.	SUMMARY AND CONCLUSIONS	24
7.	REFERENCES	25
	APPENDIX	26
	SOURCE CODE	26
	SAMPLE SCREEN	56

CHAPTER - 1

INTRODUCTION

1.1 ABOUT THE CONCEPT

The objective of this project is to implement a Music CD store web application With user interface. The motivation of this project comes from my desire to learn the increasingly growing field of web development including database designing, website designing and their growing popularity.

The word “design” in the context of a Web Application can mean many things. Its Most popular usage probably refers to the visual and user interface (UI) design of a website. This aspect is crucial because, the visitor is often more impressed with how a website looks and how easy it is to use than about which technologies and techniques are used behind the scenes, or what operating system the web server is running.

If the site is hard to use and easy to forget, it just doesn't matter what technologies was used to create it. Unfortunately, this truth makes many inexperienced programmers underestimate the importance of the way the invisible part of the site is implemented—the code, the database, and so on. The visual part of a site gets visitors interested to begin with, but its functionality makes them come back. A web site can sometimes be implemented very quickly based on certain initial requirements, but if not properly architected, it can become difficult, if not impossible, to change.

Thus, performance is also a major thrust area in the Web application which is one

Of the main reasons why users get attracted to it. Growing user needs should be taken in to concern with new features to be included. Effective performance can be achieved by making proper Database design strategy. Also, easy navigation also needs to be accomplished while executing this project. These are the main motivations for the project.

CHAPTER - 2

AIM AND SCOPE OF PRESENT INVESTIGATION

2.1 AIM

Music plays an integral and essential role in our life. Music and music items has become an inevitable part of our life. Customers obtain these music items from music stores. This existing system is nothing but a manual Music store. A manually run Music store has so many drawbacks. In such store the customer will go to such music shops and search for a specific music CD/Cassette. If the music CD/Cassette is available then only they will purchase the same.

The main problem faced was unnecessary delay. The previous system in use was also expensive and time consuming. In order to avoid unnecessary delay and minimize the flaws that existed in the previous system a follow up Web application (module) for the existing system has been designed called the 'musiCart' .

The main intention of our 'musiCart' is automate the music shopping channel between the company and the clients, and easy the user work. Through this an end user can easily purchase and enjoy music of his own choice from his home, or anywhere by just clicking the few options. And it increases the mentality of shopping.

2.2 SCOPE

Online Music Store has been introduced to save the time of customer and by giving copyright permission to the user to explore and enjoy different music according to their taste. Without actually spending our priceless time waiting in queue attending big lines in offline music stores.

For music creators and artist, it is a great source of earning, wherein they can earn some money by showing their talent from anywhere-anytime via the module.

2.3 ACCESSIBILITY

As technology enables us to move towards creating more interactive and user friendly web application, we risk widening the gap between the users who really needed access to sites like music store and those artist who have talents but unable to showcase their talents due to lack of platform.

Accessibility in Music Store doesn't mean just providing a Word or PDF version of a course. We can do so much more to create innovative e-learning material with accessibility at the heart of it and one format of content that will cater to all requirements.

Innovation and accessibility are not mutually exclusive. We can make this music store more enjoyable for everyone - regardless of whether or not they have time to be on big queue.

Creating accessible digital music store content relies on three key things: Universal Design, Universal Access and management buy-in.

2.4 ADVANTAGES

- ❖ Automate the communication between the users and the company.
- ❖ Save the time of customer.
- ❖ It provides 24 hours of facility.
- ❖ Everyone can access the system those who are living in any place.
- ❖ Easy the work of shopping.
- ❖ Increase the shopping interest.
- ❖ Decrease the time wastage.

2.5 FEASIBILITY STUDY

All projects are feasible when given unlimited resources and infinite time. It's both necessary and prudent to evaluate the feasibility of a project at the earliest possible time. The efforts and resources spent in developing the system will be a waste if the end solution does not offer timely and satisfactory solution to its users.

Feasibility study is a test of system proposed regarding workability, impact on the organization ability to meet user needs, and effective use of resources. Thus when a new application is proposed, it normally goes through a feasibility study before it is approved for development.

Feasibility and risk analysis are related in many ways. If project risk is great, the possibility of producing quality software is reduced.

CHAPTER 3

EXPERIMENTAL OR MATERIALS AND METHODS

3.1 SOFTWARE REQUIREMENT

- Windows 10 or above
- Node JS
- Angular CLI

3.1.1 Windows 10 or above

Windows 10 is a Microsoft operating system for personal computers, tablets, embedded devices and internet of things devices.

Microsoft released Windows 10 in July 2015 as a follow-up to Windows 8. The company has said it will update Windows 10 in perpetuity rather than release a new, full-fledged operating system as a successor.

3.1.2 Node JS

As an asynchronous event-driven JavaScript runtime, Node.js is designed to build scalable network applications. In the following "hello world" example, many connections can be handled concurrently. Upon each connection, the callback is fired, but if there is no work to be done, Node.js will sleep.

This is in contrast to today's more common concurrency model, in which OS threads are employed. Thread-based networking is relatively inefficient and very difficult to use. Furthermore, users of Node.js are free from worries of dead-locking the process, since there are no locks. Almost no function in Node.js directly performs I/O, so the process never blocks except when the I/O is performed using synchronous methods of Node.js standard library. Because nothing blocks, scalable systems are very reasonable to develop in Node.js.

If some of this language is unfamiliar, there is a full article on Blocking vs. Non-Blocking.

Node.js is similar in design to, and influenced by, systems like Ruby's Event Machine and Python's Twisted. Node.js takes the event model a bit further. It presents an event loop as a runtime construct instead of as a library. In other systems, there is always a blocking call to start the event-loop. Typically, behavior is defined through callbacks at the beginning of a script, and at the end a server is started through a blocking call like `EventMachine::run ()`. In Node.js, there is no such start-the-event-loop call. Node.js simply enters the event loop after executing the input script. Node.js exits the event loop when there are no more callbacks to perform. This behavior is like browser JavaScript — the event loop is hidden from the user.

HTTP is a first-class citizen in Node.js, designed with streaming and low latency in mind. This makes Node.js well suited for the foundation of a web library or framework.

Node.js being designed without threads doesn't mean you can't take advantage of multiple cores in your environment. Child processes can be spawned by using our `child_process.fork ()` API, and are designed to be easy to communicate with. Built upon that same interface is the `cluster` module, which allows you to share sockets between processes to enable load balancing over your cores.



3.1.3 Angular CLI

This topic can help you understand Angular: what Angular is, what advantages it provides, and what you might expect as you start to build your applications.

Angular is a development platform, built on TypeScript . As a platform, Angular includes:

A component-based framework for building scalable web applications

A collection of well-integrated libraries that cover a wide variety of features, including routing, forms management, client-server communication, and more

A suite of developer tools to help you develop, build, test, and update your code

With Angular, you're taking advantage of a platform that can scale from single-developer projects to enterprise-level applications. Angular is designed to make updating as straightforward as possible, so take advantage of the latest developments with a minimum of effort. Best of all, the Angular ecosystem consists of a diverse group of over 1.7 million developers, library authors, and content creators.

This section explains the core ideas behind Angular. Understanding these ideas can help you design and build your applications more effectively.

Components

Components are the building blocks that compose an application. A component includes a Typescript class with a `@Component()` decorator, an HTML template, and styles. The `@Component()` decorator specifies the following Angular-specific information:

A CSS selector that defines how the component is used in a template. HTML elements in your template that match this selector become instances of the component.

An HTML template that instructs Angular how to render the component.

An optional set of CSS styles that define the appearance of the template's HTML elements.

Templates

Every component has an HTML template that declares how that component renders. You define this template either inline or by file path.

Angular extends HTML with additional syntax that lets you insert dynamic values from your component. Angular automatically updates the rendered DOM when your component's state changes. One application of this feature is inserting dynamic text, as shown in the following example.

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.19043.1288]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS>node -v
v14.18.0

C:\Users\ASUS>npm -v
6.14.15

C:\Users\ASUS>ng --version

Angular CLI 12.2.12
Node: 14.18.0
Package Manager: npm 6.14.15
OS: win32 x64

Angular:
...

Package      Version
-----
@angular-devkit/architect 0.1202.12
@angular-devkit/core      12.2.12
@angular-devkit/schematics 12.2.12 (cli-only)
@schematics/angular       12.2.12 (cli-only)
```

3.2 HARDWARE REQUIREMENT

- Hard Disk
- RAM – 1 GB
- Processor – Dual Core or Above
- Mouse
- Keyboard
- Monitor
- Speaker

3.2.1 Hard Disk

A hard disk or fixed disk, is an electro-mechanical data storage device that uses magnetic storage to store and retrieve digital information using one or more rigid rapidly rotating disks (platters) coated with magnetic material.

During the mid-1990s the typical hard disk drive for a PC had a capacity of about 1 gigabyte. As of January 2019, desktop hard disk drives typically had a capacity of 1 to 6 terabytes, with the largest-capacity drives reaching 15 terabytes

3.2.2 RAM

Random access memory (RAM) is a type of data storage used in computers that is generally located on the motherboard. This type of memory is volatile and all information that was stored in RAM is lost when the computer is turned off.

The RAM chip may be individually mounted on the motherboard or in sets of several chips on a small board connected to the motherboard. Older memory types were in the form of chips called dual in-line package (DIP). Although DIP chips are still used today, the majority of memory is in the form of a module, a narrow printed circuit board attached to a connector on the motherboard. The three main memory circuit boards types containing chips are: RIMMs (Rambus in-line memory modules), DIMMs (dual in-line memory modules) and SIMMs (single in-line memory modules). Most motherboards

today use DIMMs. Sometimes, people get confused about the difference between memory and storage, in part because both can be measured in megabytes (MB), gigabytes (GB) and terabytes (TB).

By contrast, storage is slower, but it can retain data when the device is powered down. So, for example, if a document has been saved to a hard drive prior to a power outage or system crash, the user will still be able to retrieve it when the system is back up and running.

3.2.3 Processor

A processor is an integrated electronic circuit that performs the calculations that run a computer. A processor performs arithmetical, logical, input/output (I/O) and other basic instructions that are passed from an operating system (OS). Most other processes are dependent on the operations of a processor.

Dual core is a CPU that has two distinct processors that work simultaneously in the same integrated circuit. This type of processor can function as efficiently as a single processor but can perform operations up to twice as quickly.

3.2.4 Mouse

A mouse is a small handheld input device that controls a computer screen's cursor or pointer in conjunction with the way it is moved on a flat surface. The mouse term name originates from its likeness to a small, corded and elliptical shaped device that looks like a mouse tail.

Some mouse devices have integrated features, such as extra buttons that may be programmed and assigned with different commands.

Because the mouse reduces the use of a keyboard, its invention and continuous innovation is considered one of the most important breakthroughs in computer ergonomics.

3.2.5 Keyboard

A computer keyboard is an input device used to enter characters and functions into the computer system by pressing buttons, or keys. It is the primary device used to enter text. A keyboard typically contains keys for individual letters, numbers and special characters, as well as keys for specific functions. A keyboard is connected to a computer system using a cable or a wireless connection.

A computer keyboard is an input device used to enter characters and functions into the computer system by pressing buttons, or keys. It is the primary device used to enter text. A keyboard typically contains keys for individual letters, numbers and special characters, as well as keys for specific functions.

3.2.6 Monitor

A computer monitor is an electronic device that shows pictures for computers. Monitors often look similar to televisions. The main difference between a monitor and a television is that a monitor does not have a television tuner to change channels. Monitors often have higher display resolution than televisions.

The advent of display technology has paved the way for the continuous evolution of the monitor, whether it's for computers, television, mobile devices or any device that has a display. The current contenders for top-tier technology being used for display devices includes Super LCD 3 (SLCD3) and Super AMOLED. It should be noted that LED displays are actually just a kind of LCD display that use LED lights as backlight illumination.

3.2.7 Speaker

Most Important Speaker is required for this project because when we enter any alphabetical word or paragraph it convert it into sound which will come through the speaker of the system.

CHAPTER 4

TESTING

4.1 System Testing

Quality assurance is an important step in software engineering. This overlaps with all the phases of development right from the requirement analysis. This quality requirement of the software system must be clearly extracted during the requirement analysis and all the subsequent phases should be made biased to that, the final testing will become trivial and less expensive. There are number of quality parameters like correctness, accuracy, reliability, robustness, efficiency, effectiveness, reusability, maintainability etc.. The state of requirement of each of these parameters will vary depending upon the name and domain of the application. The testing should be done at the end of all development steps. Even though the final testing and verification are inevitable for better life and functionality of the software. The different software testing approaches and methods like white box testing and black box testing. The major phases in testing are design of test plan, setting up test case and test candidate and test procedure, testing and correction. This is a cycle process and the software will circulate through all the steps till it attends the required quality. The testing is carried in the following steps.

4.2 Unit testing

Unit testing focuses verification effort on the smallest unit of software design the module. Using the details design description as a guide, important control paths are tested to uncover errors within boundary of the boundary of the module. The relative complexity of tests and the errors detected as a result is limited by the constrained scope established for unit testing.

Unit testing is normally considered an adjacent to coding steps. After source level code has been developed, reviewed, and verified for correct syntax, unit test case design begins. A review of design information provides guidance for establishing test cases

that are likely to uncover error in each case of the categories discussed above. Each test case should be coupled with a set of expected results.

4.3 Integration testing

Integration testing is systematic technique for constructing the program structure while at the same time conducting test to uncover error associated with interfacing .The objective is to take unit tested modules and build a program structure that has been dictated by design.

There is often a tendency to attempt no incremental integration; that is to construct the program using a “big bang “approach. The entire modules are combined in advance. The entire program is tested as whole and chaoses usually result! A set of error is encountered. Correction is difficult because the isolation of cause is complicated by the vast expanse of entire program. Once errors are corrected, new ones appear and process continues in a seemingly endless loop.

CHAPTER 5

RESULT AND DISCUSSION, PERFORMANCE ANALYSIS

5.1 Result & Discussion

Music Shop is not just a change of technology. It is part of a redefinition of how we utilize our modern technology. Lovers of Music will have access to millions or billions of Music according to their taste. Some will easily can create some popular playlist and later can share with friend. Others may suggest few improvement they required in the music site.

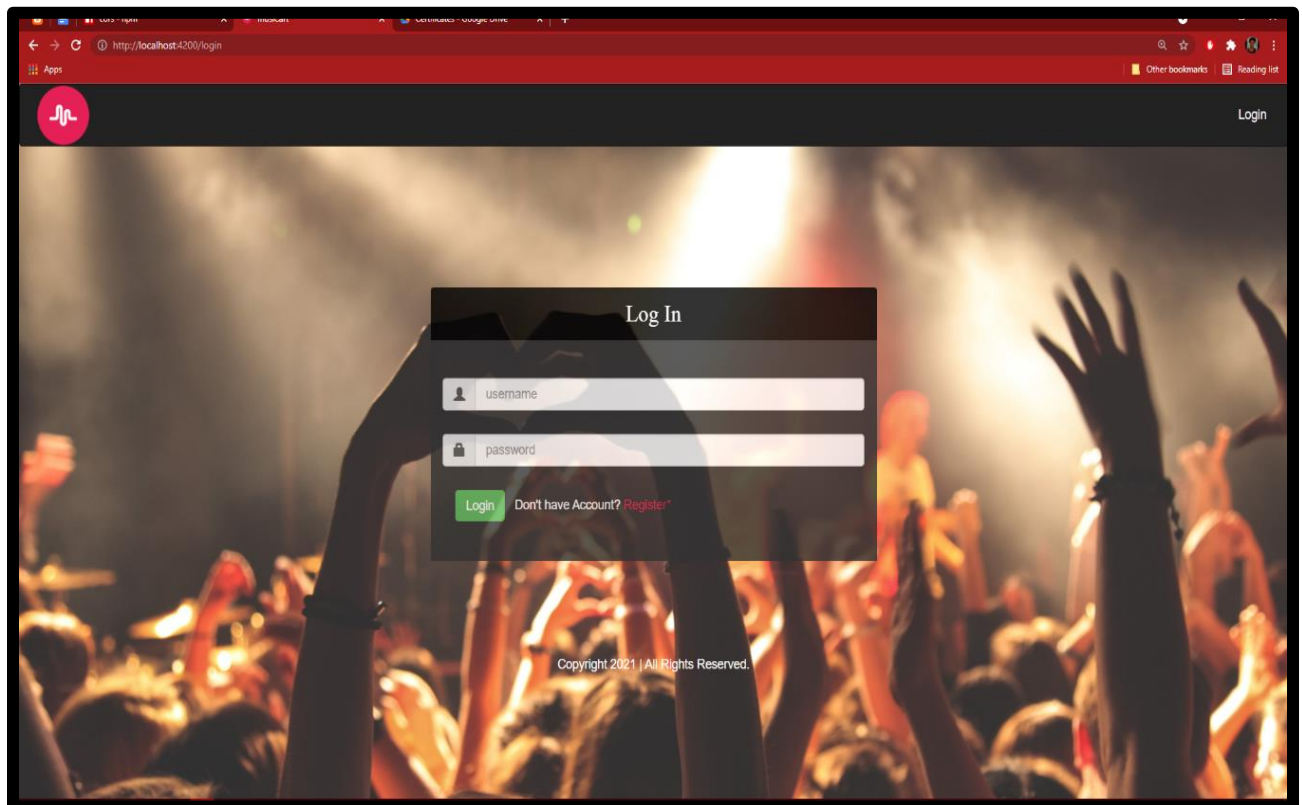
5.2 RECOMMENDATION

Recommendations The Music Store meets a lot of expectations but would perform better if the following recommendations and suggestions are considered:

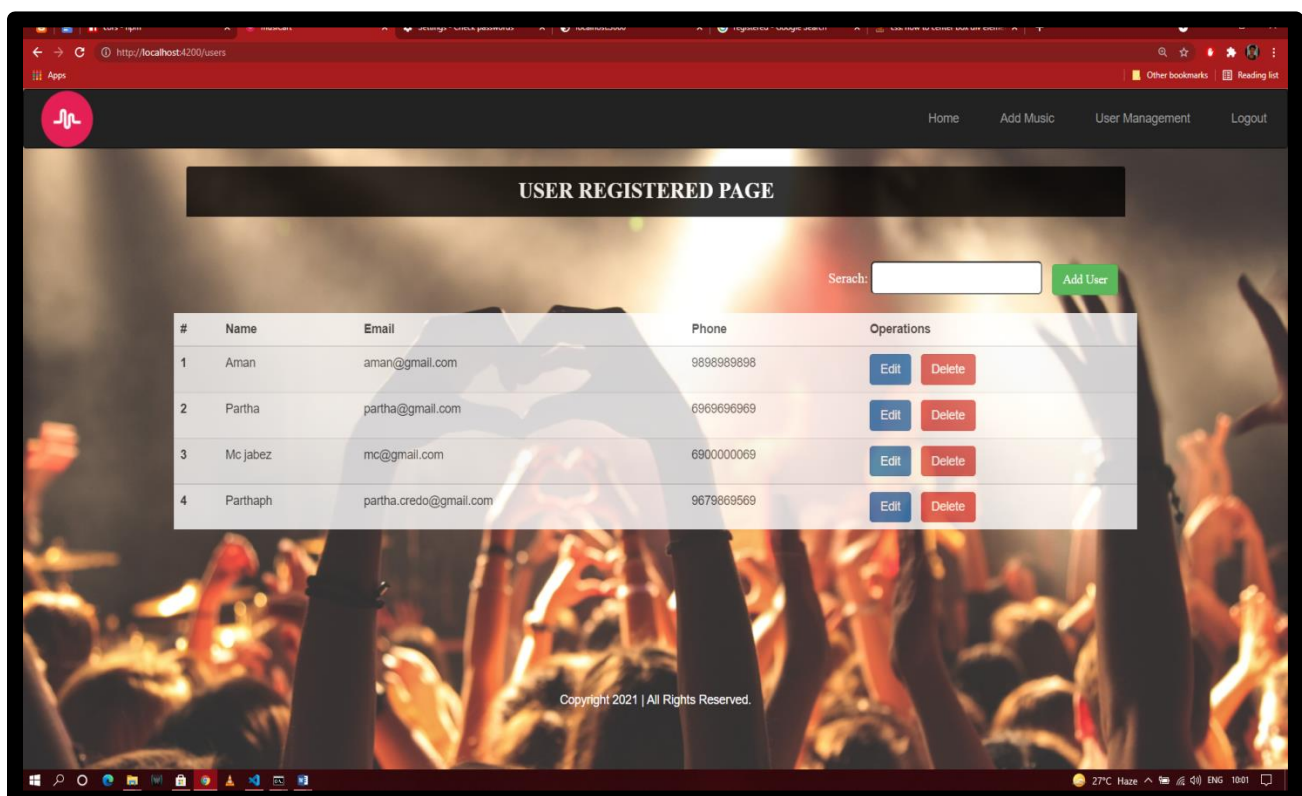
- i. System testing and maintenance should be performed regularly to avoid sudden system failures.
- ii. Updates and other modifications should be introduced with prior notice to users.
- iii. A course administrator should be employed that can manage the interactions between artist , end users and administrators on one side and technicians on the other. This will aid ensure that proper support is provided for the system.

5.3 SCREENSHOTS

5.3.1 INPUT



4.3.2 OUTPUT



CHAPTER 6

SUMMARY AND CONCLUSIONS

The software 'Online music store' reduces the considerable drawbacks like burden of human labor, portable defect and errors. This software saves time and provides 24 hour accessibility even from a remote place. Programs are menu driven which help even a newcomer to use the system with little training. Testing has been done with actual data and system is much better than the existing one. CSS makes the interface very much user friendly.

The system is highly user friendly and is well efficient to ease interactions with the users of the system. Reports generate with live data are proved to be informative and also helpful in making important decisions. The system is tested and implemented with high degree of accuracy.

The system is done with an insight into the necessary modification that may require in the future. Hence the system can be maintained successfully, without much rework.

REFERENCES

- 1) www.google.com
- 2) www.geeksforgeeks.org
- 3) www.wikipedia.org/html
- 4) www.wikipedia.org/css
- 5) www.wikipedia.org/javascript
- 6) www.wikipedia.org/angular
- 7) www.tutorialspointsg.com
- 8) www.javaworld.com
- 9) www.angularjs.com
- 10) www.nodejs.com
- 11) www.github.com

APPENDIX

BACKEND

```
const express = require("express");
const cors = require("cors");
const bodyparser = require("body-parser");
const mongodb = require("mongodb").MongoClient;

const app = express();

app.use(cors());

app.use(bodyparser.json());

var db;

mongodb.connect("mongodb+srv://aman:1234@digitalmusic.bc8qg.mongodb.net/collegebatch1?retryWrites=true&w=majority",(error, result)=>{

  if(error)
  {
    console.log("DB Not Connect");
  }
  else
  {
    db = result.db("collegebatch1");
    console.log("DB Connected");
  }
});

app.use((req, res, next)=>{                                //middleware for all paths

  console.log("Middleware 1");

  next();
});
```

```

app.use("/home", (req, res, next) => { // middleware for /home
  incoming request path

  console.log("Middleware 2");

  next();
});

function verifyUser(req, res, next)
{
  console.log("User Verified");

  next();
}

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

  console.log("Index Page");

  //res.send("<h1>Welcome to Express JS</h1>");

  var data = { a: "hi", b: "hello" };

  res.json(data);

});

app.get("/home", verifyUser, (req, res) => {

  console.log("Home Page");

  res.json("Welcome to our homepage");

});

app.post("/register", (req, res) => {

  req.body._id = new Date().getTime();

  console.log(req.body);

  db.collection("users").save(req.body, (error, data) => {

    if (error)
    {
      res.status(403).json("Error in insert Query");
    }
  });
});

```

```

        }
        else
        {
            res.json("User Registered Successfully");
        }
    });
});

app.post("/login", (req, res)=>{

    console.log(req.body);

    db.collection("users").find(req.body, {projection:{_id : 1, uname
: 1}}).toArray((error, data)=>{

        if(error)
        {
            res.status(403).json("Error in finding the login
document");
        }
        else
        {
            res.json(data);
        }
    });
});

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

    db.collection("users").find(null, {projection: {upassword :
0}}).toArray((error, data)=>{

        if(error)
        {
            res.status(403).json("Error in finding the login
document");
        }
        else
        {
            res.json(data);
        }
    });
});
});

```

```

app.get("/usernamecheck/:username", (req, res)=>{

    console.log(req.params.username);

    db.collection("users").find({uname: req.params.username},
{projection:{_id : 1}}).toArray((error, data)=>{

        if(error)
        {
            res.status(403).json("Error in finding Username
Availability");
        }
        else
        {
            res.json(data);
        }
    });
});

app.get("/getuser/:userid", (req, res)=>{

    console.log(req.params);

    db.collection("users").find({_id :
Number(req.params.userid)}).toArray((error, data)=>{

        if(error)
        {
            res.status(403).json("Error in finding the Document");
        }
        else
        {
            res.json(data);
        }
    });
});

app.put("/updateuser", (req, res)=>{

    console.log(req.body);

    var condition = {_id : req.body._id};

```

```

    var newValues = {$set: {uname: req.body.uname, uemail:
req.body.uemail, upassword: req.body.upassword, uphone:
req.body.uphone}};

    db.collection("users").update(condition, newValues, (error,
data)=>{

        if(error)
        {
            res.status(403).json("Error in Updating the Document");
        }
        else
        {
            res.json("User Updated Successfully");
        }
    })
});

app.delete("/deleteuser/:userid",(req, res)=>{

    console.log(req.params);

    db.collection("users").deleteOne({_id :
Number(req.params.userid)}),(error, data)=>{

        res.json("User Deleted Successfully");

    });
});

app.get("/search/:searchtxt?", (req, res)=>{

    console.log(req.params);

    if(req.params.searchtxt!=undefined)
    {
        var search = new RegExp(req.params.searchtxt, 'i');
        var searchCond = {uname : search};
    }
    else
    {
        var searchCond = null;
    }

```

```

        db.collection("users").find({searchCond}).toArray((error, data)=>{
            res.json(data);
        });
    });
app.post("/addmusic", (req, res)=>{
    req.body._id = new Date().getTime();
    console.log(req.body);
    db.collection("music").save(req.body, (error, data)=>{
        if(error)
        {
            res.status(403).json("Error in insert Query");
        }
        else
        {
            res.json("Music Added Successfully");
        }
    });
});

app.get("/allmusic", (req, res)=>{
    db.collection("music").find().toArray((error, data)=>{
        if(error)
        {
            res.status(403).json("Error in finding the document");
        }
        else
        {
            res.json(data);
        }
    });
});

module.exports = app;

```

angular.json

```
{
  "$schema": "./node_modules/@angular/cli/lib/config/schema.json",
  "cli": {
    "analytics": "d89cb836-b026-4d37-b07f-b76e98a74a39"
  },
  "version": 1,
  "newProjectRoot": "projects",
  "projects": {
    "firstProject": {
      "projectType": "application",
      "schematics": {
        "@schematics/angular:application": {
          "strict": true
        }
      },
      "root": "",
      "sourceRoot": "src",
      "prefix": "app",
      "architect": {
        "build": {
          "builder": "@angular-devkit/build-angular:browser",
          "options": {
            "outputPath": "dist/firstProject",
            "index": "src/index.html",
            "main": "src/main.ts",
            "polyfills": "src/polyfills.ts",
            "tsConfig": "tsconfig.app.json",
            "assets": [
              "src/favicon.ico",
              "src/assets"
            ],
            "styles": [
              "src/assets/css/bootstrap.css",
              "src/styles.css"
            ],
            "scripts": []
          },
          "configurations": {
            "production": {
              "budgets": [
                {
                  "type": "initial",
```

```

        "maximumWarning": "500kb",
        "maximumError": "1mb"
    },
    {
        "type": "anyComponentStyle",
        "maximumWarning": "2kb",
        "maximumError": "4kb"
    }
],
"fileReplacements": [
    {
        "replace": "src/environments/environment.ts",
        "with": "src/environments/environment.prod.ts"
    }
],
"outputHashing": "all"
},
"development": {
    "buildOptimizer": false,
    "optimization": false,
    "vendorChunk": true,
    "extractLicenses": false,
    "sourceMap": true,
    "namedChunks": true
}
},
"defaultConfiguration": "production"
},
"serve": {
    "builder": "@angular-devkit/build-angular:dev-server",
    "configurations": {
        "production": {
            "browserTarget": "firstProject:build:production"
        },
        "development": {
            "browserTarget": "firstProject:build:development"
        }
    }
},
"defaultConfiguration": "development"
},
"extract-i18n": {
    "builder": "@angular-devkit/build-angular:extract-i18n",
    "options": {
        "browserTarget": "firstProject:build"
    }
}

```

```

    },
    "test": {
      "builder": "@angular-devkit/build-angular:karma",
      "options": {
        "main": "src/test.ts",
        "polyfills": "src/polyfills.ts",
        "tsConfig": "tsconfig.spec.json",
        "karmaConfig": "karma.conf.js",
        "assets": [
          "src/favicon.ico",
          "src/assets"
        ],
        "styles": [
          "src/styles.css"
        ],
        "scripts": []
      }
    }
  }
},
"defaultProject": "musicCart"
}

```

package.json

```

{
  "name": "first-project",
  "version": "0.0.0",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "watch": "ng build --watch --configuration development",
    "test": "ng test"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "~12.2.0",
    "@angular/common": "~12.2.0",
    "@angular/compiler": "~12.2.0",
    "@angular/core": "~12.2.0",
    "@angular/forms": "~12.2.0",
    "@angular/platform-browser": "~12.2.0",
    "@angular/platform-browser-dynamic": "~12.2.0",

```

```

    "@angular/router": "~12.2.0",
    "body-parser": "^1.19.0",
    "bootstrap": "^5.1.3",
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "mongodb": "^3.7.3",
    "nodemon": "^2.0.15",
    "rxjs": "~6.6.0",
    "tslib": "^2.3.0",
    "zone.js": "~0.11.4"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "~12.2.8",
    "@angular/cli": "~12.2.8",
    "@angular/compiler-cli": "~12.2.0",
    "@types/jasmine": "~3.8.0",
    "@types/node": "^12.11.1",
    "jasmine-core": "~3.8.0",
    "karma": "~6.3.0",
    "karma-chrome-launcher": "~3.1.0",
    "karma-coverage": "~2.0.3",
    "karma-jasmine": "~4.0.0",
    "karma-jasmine-html-reporter": "~1.7.0",
    "typescript": "~4.3.5"
  }
}

```

server.js

```

const http = require("http");
const app = require("./backend/app");

//const server = http.createServer((req, res)=>{
//
//    res.end("Welcome to my 1st Node JS");
//});

const server = http.createServer(app);

server.listen(3000, ()=>{
    console.log("Please Cheack Port No. 3000");
})

```

addMusic.component.html

```
<br><br><br><br>
<div class="container" style="opacity: 0.8;">
  <div style="margin-top:50px;" class="mainbox col-md-6 col-md-
offset-3 col-sm-8 col-sm-offset-2">
    <div class="panel panel-info" >
      <div class="panel-heading">
        <div class="panel-title text-center"
style="color:white; font-family:'time new roman'; font-size:
25px;">Add Music</div>
      </div>
      <div style="padding-top:30px; background-color:#333;"
class="panel-body" >
        <p style="color:white;">
          {{msg}}
        </p>
        <form (submit)="doAddmusic(f)"
#f="ngForm">
          <div style="margin-bottom: 25px" class="input-
group">
            <span class="input-group-addon"><i
class="glyphicon glyphicon-file"></i></span>
            <input type="file" class="form-
control" name="mfile" ngModel
            >
          </div>
          <div style="margin-bottom: 25px" class="input-
group">
            <span class="input-group-addon"><i
class="glyphicon glyphicon-film"></i></span>
            <input type="text" class="form-
control" name="malbum" ngModel placeholder="Album Name">
          </div>
          <div style="margin-bottom: 25px" class="input-
group">
```

```

        <span class="input-group-addon"><i
class="glyphicon glyphicon-user"></i></span>
        <input type="text" class="form-
control" name="mname" ngModel placeholder="Singer Name"
>
    </div>

    <div style="margin-bottom: 25px" class="input-
group">
        <span class="input-group-addon"><i
class="glyphicon glyphicon-usd"></i></span>
        <input type="number" class="form-
control" name="mprice" ngModel placeholder="Price"
>
    </div>

    <div style="margin-top:10px" class="form-
group">
        <!-- Button -->
        <div class="col-sm-12"
style="color:white;">
            <input type="submit" class="btn btn-
success" value="Add"/> &nbsp;
        </div>
    </div>
</form>
<br> <br><br>

</div>
</div>
</div>

```

addMusic.component.ts

```

import { Component, OnInit } from '@angular/core';
import { NgForm } from '@angular/forms';
import { UsersService } from '../users.service';

@Component({
  selector: 'app-addmusic',
  templateUrl: './addmusic.component.html',

```

```

    styleUrls: ['./addmusic.component.css']
  })
  export class AddmusicComponent implements OnInit {

    msg: string;

    constructor(public userSer : UsersService) { }

    ngOnInit(): void {
    }

    doAddmusic(form : NgForm)
    {
      console.log("Music Added");

      console.log(form.value);

      this.userSer.addMusic(form.value).subscribe((data:string)=>{

        console.log(data);

        this.msg = data;

        form.reset();

      }, (error: any)=>{

        console.log(error);

        this.msg ="Something Went Wrong";

      });

    }

  }
}

```

addUser.component.html

```
<br><br><br><br>
<div class="container" style="opacity: 0.8;">
  <div style="margin-top:50px;" class="mainbox col-md-6 col-md-
offset-3 col-sm-8 col-sm-offset-2">
    <div class="panel panel-info" >
      <div class="panel-heading">
        <div class="panel-title text-center"
style="color:white; font-family:'time new roman'; font-size:
25px;">Add Users</div>
      </div>
      <div style="padding-top:30px; background-color:#333;"
class="panel-body" >
        <p style="color:white;">
          {{msg}}
        </p>
        <form (submit)="addUsers(f)"
#f="ngForm">
          <div style="margin-bottom: 25px" class="input-
group">
            <span class="input-group-addon"><i
class="glyphicon glyphicon-user"></i></span>
            <input type="text" class="form-
control" name="uname" #username="ngModel"
(blur)="usernameCheck(username.value)" ngModel placeholder="Username"
required minlength="4">
          </div>
          <p class="alert alert-danger" *ngIf =
"username.errors?.required && username.touched">
            Please Enter Username
          </p>
          <p class="alert alert-warning" *ngIf =
"username.errors?.minlength && username.touched">
            Please Enter Username minimum 6
characters
          </p>
          <div style="margin-bottom: 25px" class="input-
group">
            <span class="input-group-addon"><i
class="glyphicon glyphicon-envelope"></i></span>
```

```

<input type="text" class="form-
control" name="uemail" ngModel #useremail="ngModel"
placeholder="Email" required
email>

</div>

<p class="alert alert-danger" *ngIf =
"useremail.errors?.required && useremail.touched">
    Please Enter Email Id
</p>

<p class="alert alert-danger"
*ngIf="useremail.errors?.email && useremail.touched">
    Please Enter Valid Email Id
</p>
<div style="margin-bottom: 25px" class="input-
group">
    <span class="input-group-addon"><i
class="glyphicon glyphicon-lock"></i></span>
    <input type="password"
class="form-control" name="upassword" ngModel placeholder="Password">
</div>

<div style="margin-bottom: 25px" class="input-
group">
    <span class="input-group-addon"><i
class="glyphicon glyphicon-phone"></i></span>
    <input type="text" class="form-
control" name="uphone" ngModel placeholder="Phone Number"
pattern="^(0|[1-9][0-9]*)$" #userphone="ngModel" maxlength="10"
minlength="10">
</div>

<p class="alert alert-danger"
*ngIf="userphone.errors?.pattern">
    Please Enter Valid Phone Number
</p>

<p class="alert alert-warning" *ngIf =
"userphone.errors?.minlength && userphone.touched">
    Invalid Phone Number!!
</p>

<div style="margin-top:10px" class="form-
group">

```

```

                                <!-- Button -->
                                <div class="col-sm-12"
style="color:white;">
                                <input type="submit" class="btn btn-
success" value="Add User" [disabled]="f.invalid || !usernameAvail"/>
                                &nbsp;
                                </div>
                                </div>
                                </form>
                                <br> <br><br>
                                </div>
                                </div>
                                </div>
</div>

```

addUser.component.ts

```

import { Component, OnInit } from '@angular/core';
import { NgForm } from '@angular/forms';
import { UsersService } from '../users.service';

@Component({
  selector: 'app-adduser',
  templateUrl: './adduser.component.html',
  styleUrls: ['./adduser.component.css']
})
export class AdduserComponent implements OnInit {

  msg : string;

  usernameAvail = false;

  constructor(public userSer : UsersService) { }

  ngOnInit(): void {
  }

  addUsers(form:NgForm)

```

```

{
  console.log(form.value);

  this.userSer.userRegistration(form.value).subscribe((data:string)=
>{

    console.log(data);

    this.msg = data;

    form.reset();

  },(error:any)=>{

    console.log(error);

    this.msg = "Something Went Wrong";

  }));
}

usernameCheck(uname : string)
{
  this.userSer.usernameAvailability(uname).subscribe((data:
any[])=>{

    console.log(data);

    if(data.length==0)
    {
      this.msg = "Congrats! Username is available for you!!";
      this.usernameAvail = true;
    }
    else
    {
      this.msg = "Username is not available";
      this.usernameAvail = false;
    }

  }, (error:any)=>{

    console.log(error);

  }));
}

```

```
}
```

editUser.component.html

```
<div class="container">
  <div style="margin-top:50px;" class="mainbox col-md-6 col-md-
offset-3 col-sm-8 col-sm-offset-2">
    <div class="panel panel-info" >
      <div class="panel-heading">
        <div class="panel-title">Edit User</div>
      </div>
      <div style="padding-top:30px" class="panel-body" >
        <p>
          {{msg}}
        </p>
        <form #f="ngForm" (submit)="editUser(f)"
*ngIf="userData">
          <div style="margin-bottom: 25px" class="input-
group">
            <span class="input-group-addon"><i
class="glyphicon glyphicon-user"></i></span>
            <input type="text" class="form-control" name="uname"
[ngModel]="userData.uname" #username="ngModel" placeholder="Username"
required minlength="4">
          </div>
          <p class="alert alert-danger"
*ngIf="username.errors?.required &&
username.touched">
            Please Enter Username
          </p>
          <p class="alert alert-danger"
*ngIf="username.errors?.minlength &&
username.touched">
            Please Enter Minimum 6 Characters
          </p>
          <div style="margin-bottom: 25px" class="input-
group">
            <span class="input-group-addon"><i
class="glyphicon glyphicon-envelope"></i></span>
```

```

        <input type="text" class="form-control" name="uemail"
[ngModel]="userData.uemail" #useremail="ngModel" placeholder="Email"
required email>

        </div>

        <p class="alert alert-danger"
*ngIf="useremail.errors?.required &&
useremail.touched">
            Please Enter Your Email
        </p>
        <p class="alert alert-danger"
*ngIf="useremail.errors?.email &&
useremail.touched">
            Please Enter Valid Email
        </p>
        <div style="margin-bottom: 25px" class="input-
group">
            <span class="input-group-addon"><i
class="glyphicon glyphicon-lock"></i></span>
            <input type="password" class="form-
control" name="upassword" [ngModel]="userData.upassword"
#userpassword="ngModel" placeholder="Password" required>
        </div>

        <div style="margin-bottom: 25px" class="input-
group">
            <span class="input-group-addon"><i
class="glyphicon glyphicon-phone"></i></span>
            <input type="text" class="form-control" name="uphone"
[ngModel]="userData.uphone" placeholder="Phone Number"
#userphone="ngModel" pattern="^(0|[1-9])[0-
9]*)$" >
        </div>

        <p class="alert alert-danger"
*ngIf="userphone.errors?.pattern">
            Please Enter Nos Only
        </p>
        <div style="margin-top:10px" class="form-
group">
            <!-- Button -->
            <div class="col-sm-12">
                <input type="submit" class="btn btn-success"
value="Update" [disabled]="f.invalid" /> &nbsp;
            </div>

```

```

        </div>
      </form>
      <br> <br><br>
    </div>
  </div>
</div>

```

editUser.component.ts

```

import { Component, OnInit } from '@angular/core';
import { NgForm } from '@angular/forms';
import { ActivatedRoute, Params } from '@angular/router';
import { UsersService } from '../users.service';

@Component({
  selector: 'app-edituser',
  templateUrl: './edituser.component.html',
  styleUrls: ['./edituser.component.css']
})
export class EdituserComponent implements OnInit {

  userData : { _id : number, uname: string, upassword : string, uemail
: string, uphone : string};

  msg : string;

  constructor(public activeRoute : ActivatedRoute, public userSer :
UsersService) { }

  ngOnInit(): void {

    this.activeRoute.params.subscribe((param:Params)=>{

      console.log(param);

      if(param.userid)
      {

```

```

        this.userSer.getSingleUserData(param.userid).subscribe((data:any[])=>{

            console.log(data);

            this.userData = data[0];

        }, (error:any)=>{

            console.log(error);

        });

    });

}

editUser(form:NgForm)
{
    form.value._id = this.userData._id;

    this.userSer.editSingleUserData(form.value).subscribe((data:string)=>{

        console.log(data);

        this.msg = data;

    }, (error:any)=>{

        console.log(error);

        this.msg = "Something Went Wrong";

    });

}

}

```

Login.component.html

```

<br><br><br><br>
<div class="container" style="opacity: 0.8;">

```

```

    <div style="margin-top:50px;" class="mainbox col-md-6 col-md-
offset-3 col-sm-8 col-sm-offset-2">
        <div class="panel panel-info" >
            <div class="panel-heading">
                <div class="panel-title text-center"
style="color:white; font-family:'time new roman'; font-size:
25px;">Log In</div>
            </div>
            <div style="padding-top:30px; background-color:#333;"
class="panel-body" >
                <p style="color:#FF2451;">
                    {{msg}}
                </p>
                <form (submit)="doLogin(f)"
#f="ngForm">
                    <div style="margin-bottom: 25px" class="input-
group">
                        <span class="input-group-addon"><i
class="glyphicon glyphicon-user"></i></span>
                        <input type="text" class="form-
control" name="uname" ngModel placeholder="username"
>
                    </div>

                    <div style="margin-bottom: 25px" class="input-
group">
                        <span class="input-group-addon"><i
class="glyphicon glyphicon-lock"></i></span>
                        <input type="password"
class="form-control" name="upassword" ngModel placeholder="password">
                    </div>
                    <div style="margin-top:10px" class="form-
group">
                        <!-- Button -->
                        <div class="col-sm-12"
style="color:white;">
                            <input type="submit" class="btn btn-
success" value="Login"/> &nbsp;
                            Don't have Account? <a
routerLink="/register" style="color:#FF2451;">Register*</a>
                        </div>
                    </div>
                </form>
                <br> <br><br>

```

```

        </div>
    </div>
</div>

```

login.component.ts

```

import { Component, OnInit } from '@angular/core';
import { NgForm } from '@angular/forms';
import { Router } from '@angular/router';
import { UsersService } from '../users.service';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {

  msg : string;

  constructor(public userSer : UsersService, public myRouter : Router)
  { }

  ngOnInit(): void {
  }

  doLogin(form:NgForm)
  {

    console.log(form.value);

    this.userSer.userLogin(form.value).subscribe((data:any[])=>{

      //this.msg = data;

      console.log(data);

      if(data.length==0)
      {

```

```

        this.msg = "Invalid Login**";
    }
    else
    {
        localStorage.setItem("loggeduser",data[0]._id);
        this.myRouter.navigateByUrl("/music");
    }
},(error: any)=>{

    console.log(error);
    this.msg = "Something Went Wrong";
});
}
}

```

register.component.ts

```

<br><br><br><br>
<div class="container" style="opacity: 0.8;">
    <div style="margin-top:50px;" class="mainbox col-md-6 col-md-
offset-3 col-sm-8 col-sm-offset-2">
        <div class="panel panel-info" >
            <div class="panel-heading">
                <div class="panel-title text-center"
style="color:white; font-family:'time new roman'; font-size:
25px;">Register</div>
            </div>
            <div style="padding-top:30px; background-color:#333;"
class="panel-body" >
                <p style="color:white;">
                    {{msg}}
                </p>
                <form (submit)="doRegistration(f)"
#f="ngForm">
                    <div style="margin-bottom: 25px" class="input-
group">
                        <span class="input-group-addon"><i
class="glyphicon glyphicon-user"></i></span>

```

```

        <input type="text" class="form-
control" name="uname" #username="ngModel" ngModel
placeholder="Username" required
minlength="4">

    </div>

    <p class="alert alert-danger" *ngIf =
"username.errors?.required && username.touched">
        Please Enter Username
    </p>

    <p class="alert alert-warning" *ngIf =
"username.errors?.minlength && username.touched">
        Please Enter Username minimum 6
characters
    </p>

    <div style="margin-bottom: 25px" class="input-
group">
        <span class="input-group-addon"><i
class="glyphicon glyphicon-envelope"></i></span>
        <input type="text" class="form-
control" name="uemail" ngModel #useremail="ngModel"
placeholder="Email" required
email>
    </div>

    <p class="alert alert-danger" *ngIf =
"useremail.errors?.required && useremail.touched">
        Please Enter Email Id
    </p>

    <p class="alert alert-danger"
*ngIf="useremail.errors?.email && useremail.touched">
        Please Enter Valid Email Id
    </p>

    <div style="margin-bottom: 25px" class="input-
group">
        <span class="input-group-addon"><i
class="glyphicon glyphicon-lock"></i></span>
        <input type="password"
class="form-control" name="upassword" ngModel placeholder="Password">
    </div>

```

```

        <div style="margin-bottom: 25px" class="input-
group">
            <span class="input-group-addon"><i
class="glyphicon glyphicon-phone"></i></span>
            <input type="text" class="form-
control" name="uphone" ngModel placeholder="Phone Number"
pattern="^(0|[1-9][0-9]*)$" #userphone="ngModel" maxlength="10"
minlength="10">
        </div>

        <p class="alert alert-danger"
*ngIf="userphone.errors?.pattern">
            Please Enter Valid Phone Number
        </p>

        <p class="alert alert-warning" *ngIf =
"userphone.errors?.minlength && userphone.touched">
            Invalid Phone Number!!
        </p>

        <div style="margin-top:10px" class="form-
group">
            <!-- Button -->
            <div class="col-sm-12"
style="color:white;">
                <input type="submit" class="btn btn-
success" value="Register" [disabled]="f.invalid"/> &nbsp;
                Already have Account? <a
routerLink="/login" style="color:#FF2451;">Login</a>
            </div>
        </div>
    </form>
    <br> <br><br>

    </div>
</div>
</div>

```

Register.component.ts

```
import { Component, OnInit } from '@angular/core';
import { NgForm } from '@angular/forms';
import { UsersService } from '../users.service';

@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrls: ['./register.component.css']
})
export class RegisterComponent implements OnInit {

  msg : string;

  constructor(public userSer : UsersService) { }

  ngOnInit(): void {
  }

  doRegistration(form: NgForm)
  {
    console.log("User Registered");

    console.log(form.value);

    this.userSer.userRegistration(form.value).subscribe((data:string)=
>{

      console.log(data);

      this.msg = data;

      form.reset();

    },(error:any)=>{

      console.log(error);

      this.msg ="Something Went Wrong";

    });

  }

}
```

```
}
```

User.component.html

```
<div class="container">
  <div class="row" style="padding-top: 30px;">
    <div class="col-sm-4"></div>

    <div class="col-sm-4"></div>

    <div class="col-sm-4" style="color:white; font-family:'Times New
Roman'; font-size: 16px;">
      Serach: <input type="text" #search
(keyup)="doSearch(search.value)" style="border-radius:5px; padding:
5px;"> &nbsp;
      <a routerLink="/adduser"><button class="btn btn-success">Add
User</button></a>
    </div>
  </div>
<br><br>
  <div class="row">
<div class="col"></div>

<div class="col">
  {{msg}}
  <table class="table table-striped">
    <thead>
      <tr>
        <th scope="col">#</th>
        <th scope="col">Name</th>
        <th scope="col">Email</th>
        <th scope="col">Phone</th>
        <th scope="col">Operations</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let user of users; index as i">
        <th scope="row">{{i+1}}</th>
        <td>{{user.username}}</td>
        <td>{{user.uemail}}</td>
        <td>{{user.uphone}}</td>
        <td>
          <a routerLink="/edituser/{{user._id}}"> <button
class="btn btn-primary">Edit</button></a> &nbsp;
```

```

                <button class="btn btn-danger"
(click)="deleteUser(user._id)">Delete</button>
            </td>
        </tr>
    </tbody>
</table>
</div>

<div class="col"></div>
</div>
</div>
<br><br><br><br>

```

user.component.ts

```

import { Component, OnInit } from '@angular/core';
import { UsersService } from '../users.service';

@Component({
  selector: 'app-users',
  templateUrl: './users.component.html',
  styleUrls: ['./users.component.css']
})
export class UsersComponent implements OnInit {

  users : any[] = [];

  msg : string;

  constructor(public userSer : UsersService) { }

  ngOnInit(): void {

    this.userSer.getAllUsers().subscribe((data: any[])=>{

      console.log(data);
      this.users = data;

    },(error:any)=>{

      console.log(error);
      this.msg = "Something Went Wrong";
    })
  }
}

```

```

}

deleteUser(userId:number)
{
  if(confirm("Are you sure want to Delete this Record"))
  {
    //console.log("User Deleted with user id "+userId);

    this.userSer.deleteUserData(userId).subscribe((data:string)=>{

      this.msg=data;

      var index = this.users.findIndex((obj)=>{

        return obj._id == userId;

      });

      this.users.splice(index,1);

    }, (error:any)=>{

      console.log(error);
      this.msg = "Something Went Wrong";

    });
  }
}

doSearch(search:string)
{
  this.userSer.searchUsers(search).subscribe((data:any[])=>{

    console.log(data);

    this.users = data;

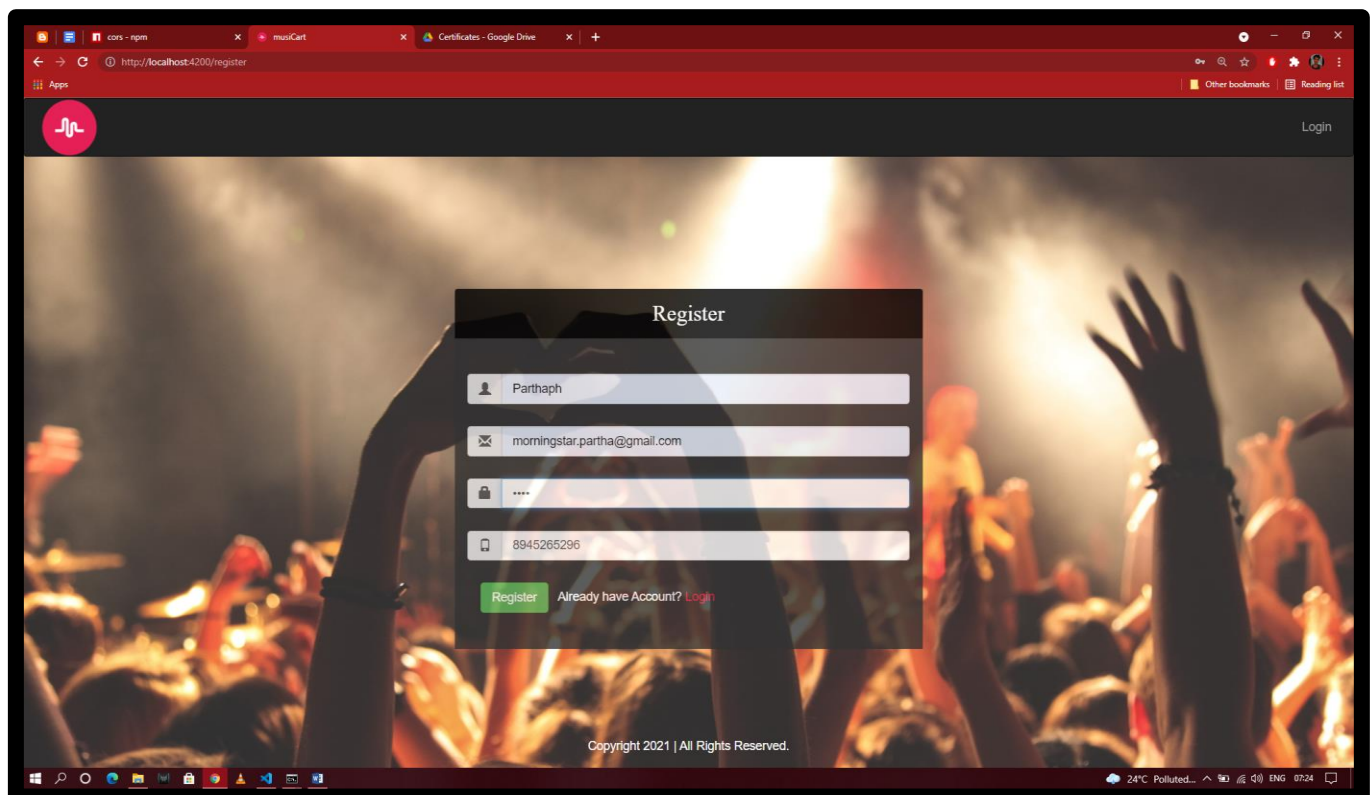
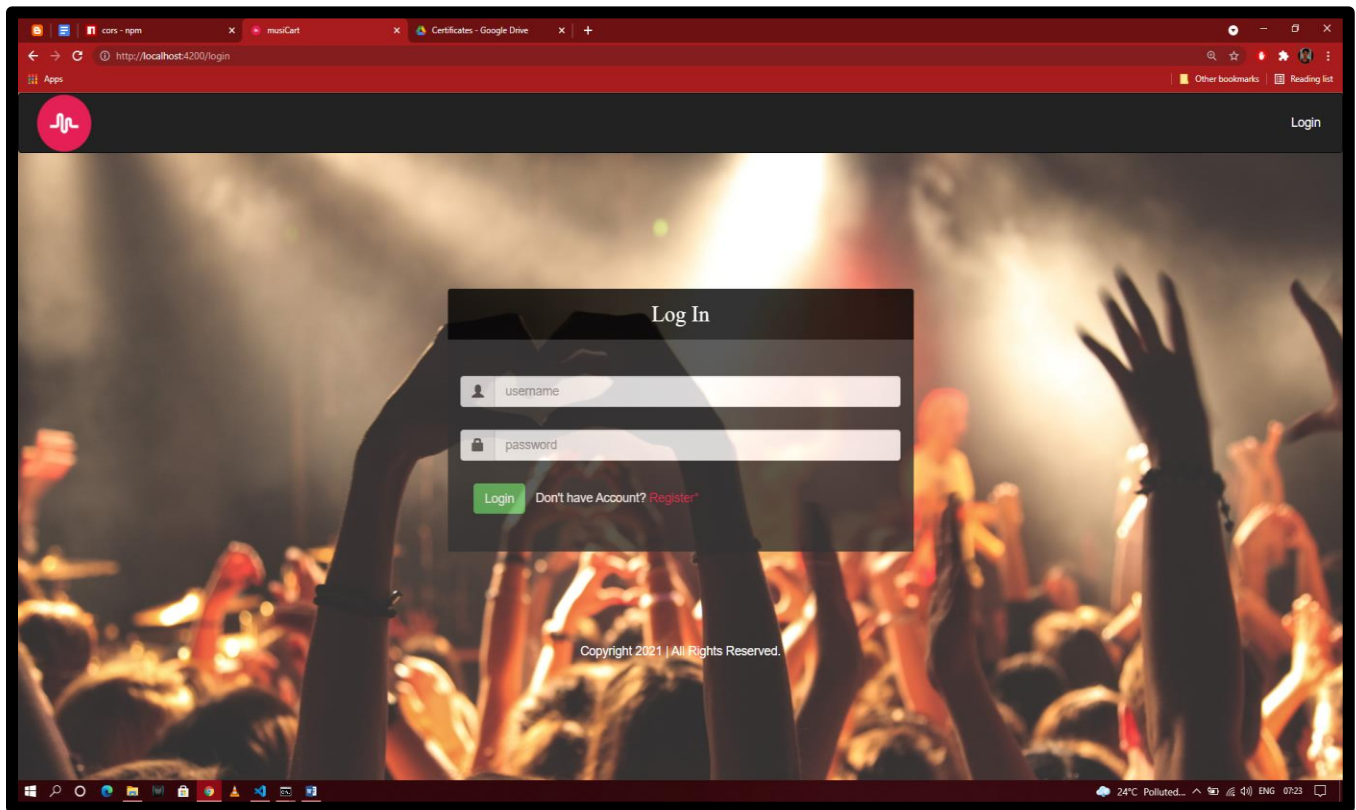
  }, (error:any)=>{

    console.log(error);

  });
}}

```

SAMPLE SCREENS



[Home](#)[Add Music](#)[User Management](#)[Logout](#)

Ik Mulaqaat

Singer: Palak Muchhal

\$ 2.8

Buy

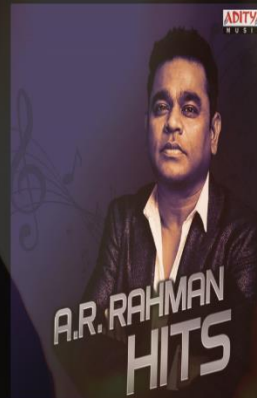


Tu kitni achhi hai

Singer: Lata Mangeshkar

\$ 3.5

Buy



Kun faya kun

Singer: A R Rahman

\$ 3

Buy



Radha

Singer: Shreya Ghoshal

\$ 2.5

Buy



Aadat

Singer: Atif Aslam

\$ 4.2

Buy



Aazma

Singer: Shruti Haasan

\$ 1.9

Buy

Copyright 2021 | All Rights Reserved.

