**Project Name: Artist Gallery**

**(CDAC March 23 – Infoway Technologies(Pune))**

**Project Member:**

* Satish Sakar (230343120077)
* Tanmay Bolakhe (230343120096)
* Ayush Saha (230343120017)
* Rohit Sonar (230343120069)

**Abstract:**

**E-ComStar** is an web based application for user as well as artist which provides the facilities for booking artist for restaurants,pubs,lounge,bistro and private parties. And also the platform where artists can host their talent online. The hotel industry is a business venture for the owner and a solace for the artist . A user can get stranded in the quest to find the proper artist for his event if he has not made adequate plans by the existing system. Through this study, it was realized that for a user to be guaranteed a artist, he or she has to physically find the artist. The objects-oriented analysis and design methodology was therefore used to analyze the system in order to discover the various objects involved and how they interact with one another so that a new and improved system can be defined. The use of online view of artist rates was used for the new system so that the customer can view and make his choice before booking.

The system implemented using a 3-tier approach, with a backend database MySql, Spring Boot Framework and frontend React. In order to develop an e-commerce website, a number of Technologies must be studied and understood. These include multi-tiered architecture, server and client side scripting techniques, implementation technologies such as Spring boot, React and relational database MySQL.This document will discuss each of underlying technologies to create and implement and e-commerce website.

**Implementation Technologies:**

The objective of this project is to develop an online artist booking system. When the user types in the URL of the E-ComStar in the address field of the browser, a Web Server is contacted to get the requested information. Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can **just run**. You can get started with minimum configurations without the need for an entire Spring configuration setup.

**Why Spring Boot?**

* You can choose Spring Boot because of the features and benefits it offers as given here −
* It provides a flexible way to configure Java Beans, XML configurations, and Database Transactions.
* It provides a powerful batch processing and manages REST endpoints
* In Spring Boot, everything is auto configured; no manual configurations are needed.
* It offers annotation-based spring application
* Eases dependency management
* It includes Embedded Servlet Container

## How does it work?

Spring Boot automatically configures your application based on the dependencies you have added to the project by using **@EnableAutoConfiguration** annotation. For example, if MySQL database is on your classpath, but you have not configured any database connection, then Spring Boot auto-configures an in-memory database.

The entry point of the spring boot application is the class contains **@SpringBootApplication** annotation and the main method.

Spring Boot automatically scans all the components included in the project by using **@ComponentScan** annotation.

## Spring Boot Starters

Handling dependency management is a difficult task for big projects. Spring Boot resolves this problem by providing a set of dependencies for developers convenience.

For example, if you want to use Spring and JPA for database access, it is sufficient if you include **spring-boot-starter-data-jpa** dependency in your project.

Note that all Spring Boot starters follow the same naming pattern **spring-boot-starter-** \*, where \* indicates that it is a type of the application.

## Auto Configuration

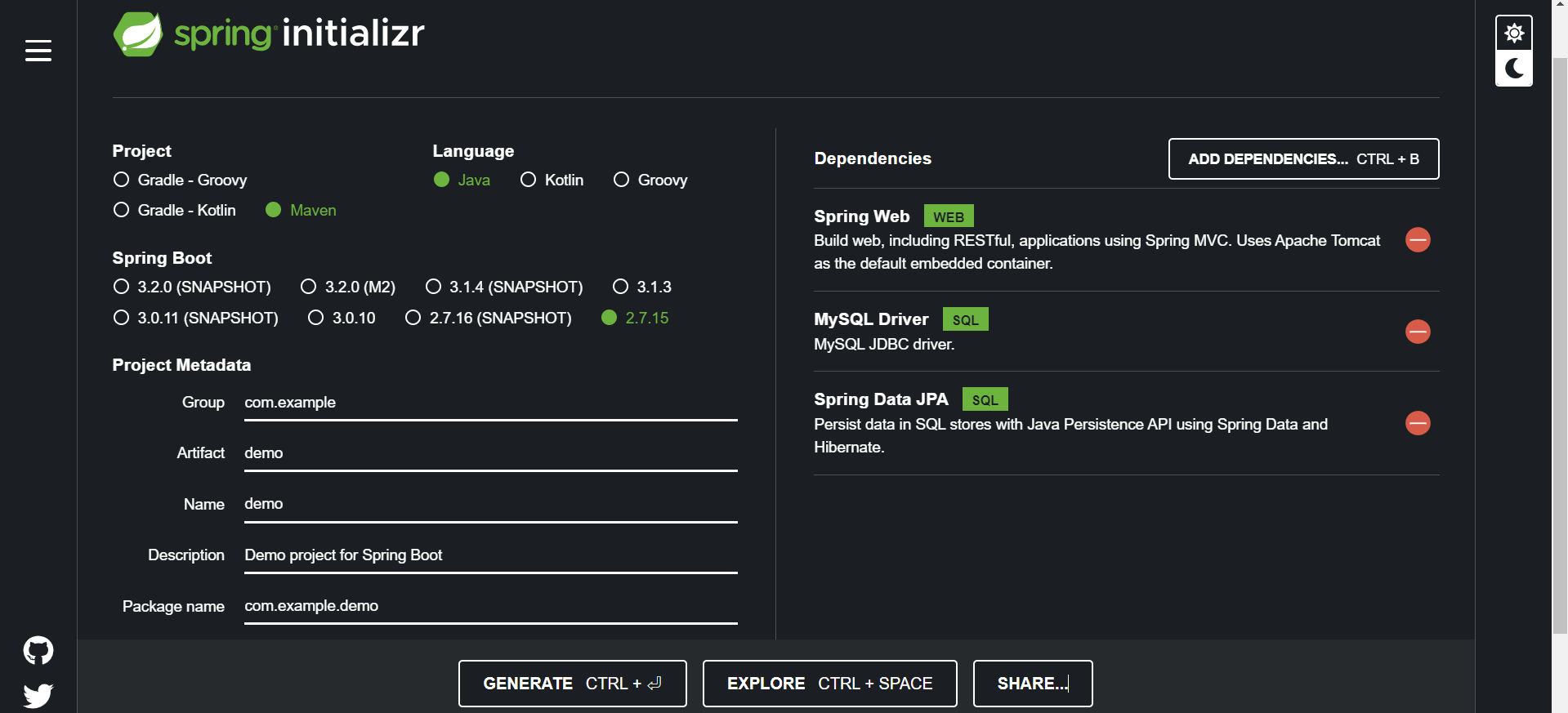
Spring Boot Auto Configuration automatically configures your Spring application based on the JAR dependencies you added in the project. For example, if MySQL database is on your class path, but you have not configured any database connection, then Spring Boot auto configures an in-memory database.

For this purpose, you need to add **@EnableAutoConfiguration** annotation or **@SpringBootApplication** annotation to your main class file. Then, your Spring Boot application will be automatically configured.

## Spring Initializer

One of the ways to Bootstrapping a Spring Boot application is by using Spring Initializer. To do this, you will have to visit the Spring Initializer web page [www.start.spring.io](https://start.spring.io/) and choose your Build, Spring Boot Version and platform. Also, you need to provide a Group, Artifact and required dependencies to run the application.

Observe the following screenshot that shows an example where we added the spring-boot-starter-web dependency to write REST Endpoints.

****

Once you provided the Group, Artifact, Dependencies, Build Project, Platform and Version, click **Generate Project** button. The zip file will download and the files will be extracted.

This section explains you the examples by using both Maven and Gradle.

## Maven

After you download the project, unzip the file. Now, your **pom.xml** file looks as shown below −

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.7.14</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.app</groupId>

<artifactId>Ecomstar</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>Ecomstar</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<!-- https://mvnrepository.com/artifact/net.sf.json-lib/json-lib -->

<dependency>

<groupId>net.sf.json-lib</groupId>

<artifactId>json-lib</artifactId>

<version>2.4</version>

<classifier>jdk15</classifier>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

## Class Path Dependencies

Spring Boot provides a number of **Starters** to add the jars in our class path. For example, for writing a Rest Endpoint, we need to add the **spring-boot-starter-web** dependency in our class path. Observe the codes shown below for a better understanding −

### **Maven dependency**

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

</dependencies>

## Main Method

The main method should be writing the Spring Boot Application class. This class should be annotated with **@SpringBootApplication**. This is the entry point of the spring boot application to start. You can find the main class file under **src/java/main** directories with the default package.

In this example, the main class file is located at the **src/java/main** directories with the default package **com.tutorialspoint.demo**. Observe the code shown here for a better understanding −

package com.app.Ecomstar;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class EcomstarApplication {

public static void main(String[] args) {

SpringApplication.run(EcomstarApplication.class, args);

}}

## Write a Rest Endpoint

To write a simple Hello World Rest Endpoint in the Spring Boot Application main class file itself, follow the steps shown below −

* Firstly, add the **@RestController** annotation at the top of the class.
* Now, write a Request URI method with **@RequestMapping** annotation.

Now, your main Spring Boot Application class file will look like as shown in the code given below −

packagecom.EcomStar.demo;

importorg.springframework.boot.SpringApplication;

importorg.springframework.boot.autoconfigure.SpringBootApplication;

importorg.springframework.web.bind.annotation.RequestMapping;

importorg.springframework.web.bind.annotation.RestController;

@SpringBootApplication

@RestController

publicclassDemoApplication{

publicstaticvoidmain(String[]args){

SpringApplication.run(DemoApplication.class,args);

}

@RequestMapping(value="/")

}

}

## Properties File

Properties files are used to keep ‘N’ number of properties in a single file to run the application in a different environment. In Spring Boot, properties are kept in the **application.properties** file under the classpath

The application.properties file is located in the **src/main/resources** directory. The code for sample **application.properties** file is given below −

#to set the server port

server.port=9999

## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)

spring.datasource.url = jdbc:mysql://localhost:3306/artist\_gallary?&useSSL=false

spring.datasource.username = root

spring.datasource.password = Rohit@123

## Hibernate Properties

# The SQL dialect makes Hibernate generate better SQL for the chosen database

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect

spring.jpa.properties.hibernate.show\_sql=true

spring.jackson.serialization.fail-on-empty-beans=false

#spring.jpa.open-in-view=true

# Hibernate ddl auto (create, create-drop, validate, update)

spring.jpa.hibernate.ddl-auto = update

**React:**

**React** (also known as **React.js** or **ReactJS**) is a free and open-source front-end JavaScript library for building user interfaces or UI components. It is maintained by Facebook and a community of individual developers and companies.React can be used as a base in the development of single-page or mobile applications.

## Create an React project

* **npx create –react –app EComStar**

**Start the React development server using the following command.**

* **npm start**

The server starts at the <http://localhost:3000/>

## Generate react Component

The component is manually imported to **app.js** file. Now, we have to configure the routing of react components using react-router-dom in which we have different module to route component.

You can check the **react-routing.dom**  module inside the package.jsonfile in your project . It is created because when we install a react app, we permit **react-router-dom** to generate the routing file for us.

Now, write the following code inside an **app.js** file:



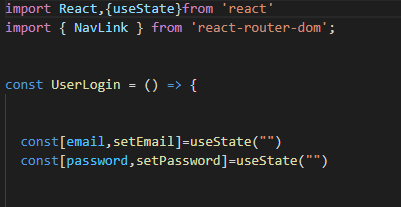
## Configure the Axios Http Module

* Axios is used to communicate with the backend and it also supports the Promise API that is native to JS ES6.
* It is a library which is used to make requests to an API, return data from the API, and then do things with that data in our React application.
* Axios is a very popular HTTP client, which allows us to make HTTP  requests from the browser.

## C:\Users\admin\Desktop\Project\Screenshot 2021-09-30 165824.png

**React hooks**

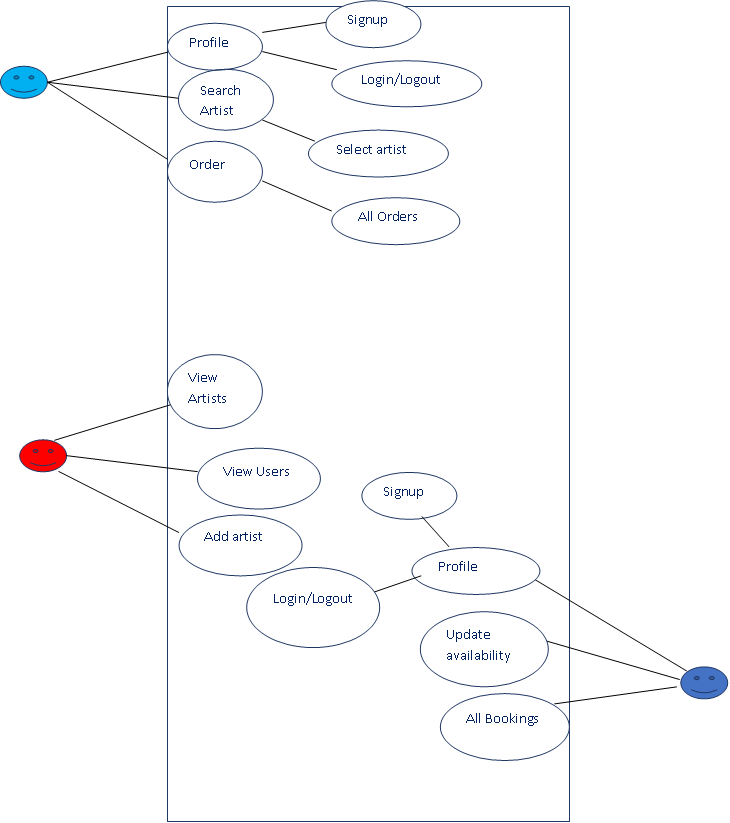
React hooks use in functional base component for managing the states

****

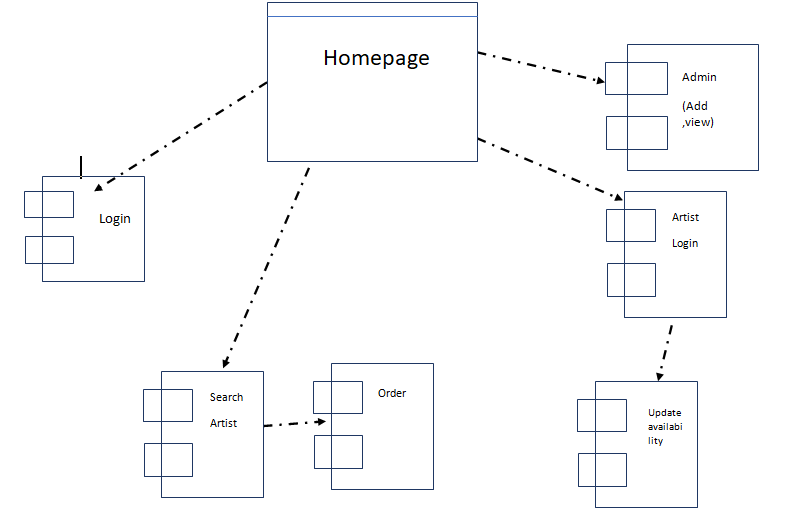
**Tables:**

* **User**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Column Name** | **Type** |  |  |  |  |
| user\_email | varchar(255) | NO | PRI |  |  |
| user\_gender | varchar(255) | YES |  |  |  |
| user\_name | varchar(255) | YES |  |  |  |
| user\_password | varchar(255) | YES |  |  |  |
| user\_phone\_number | bigint(20) | NO |  |  |  |
| * **Artist**  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Column Name** | **Type** |  |  |  |  | | artist\_emailid | varchar(255) | NO | PRI |  |  | | artist\_address | varchar(255) | YES |  |  |  | | artist\_gender | varchar(255) | YES |  |  |  | | artist\_genre | varchar(255) | YES |  |  |  | | artist\_name | varchar(255) | YES |  |  |  | | artist\_password | varchar(255) | YES |  |  |  | | artist\_phone\_number | bigint(20) | NO |  |  |  |  * **Order1**  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Column Name** | **Type** |  |  |  |  | | order\_id | int(11) | NO | PRI |  |  | | booked\_date | varchar(255) | YES |  |  |  | | booked\_time | varchar(255) | YES |  |  |  | | order\_date | varchar(255) | YES |  |  |  | | order\_price | bigint(20) | NO |  |  |  | | venue | varchar(255) | YES |  |  |  | | artist\_artist\_emailid | varchar(255) | YES | MUL |  |  | | user\_user\_email | varchar(255) | YES | MUL |  |  |  * **Adate\_avail**  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Column Name** | **Type** |  |  |  |  | | id | int(11) | NO | PRI |  |  | | artistfees | float | NO |  |  |  | | from\_date | datetime(6) | YES |  |  |  | | to\_date | datetime(6) | YES |  |  |  | | artist\_artist\_emailid | varchar(255) | YES | MUL |  |  | |  |  |  |  |  |



**Figure: Use Case Diagram**



**Figure: Component Diagram**

User

Book Now

Wait for availability check

Order confirmation

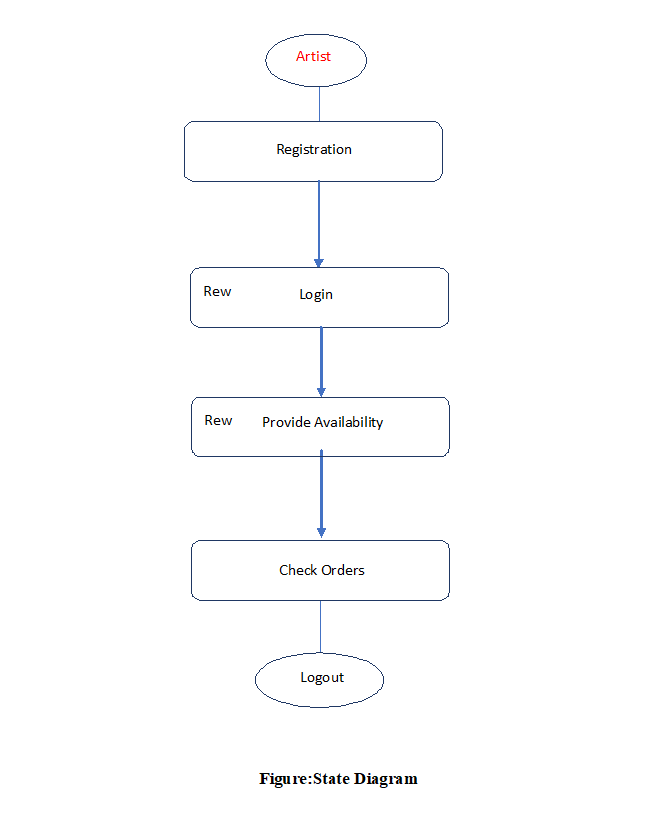
Search Artist

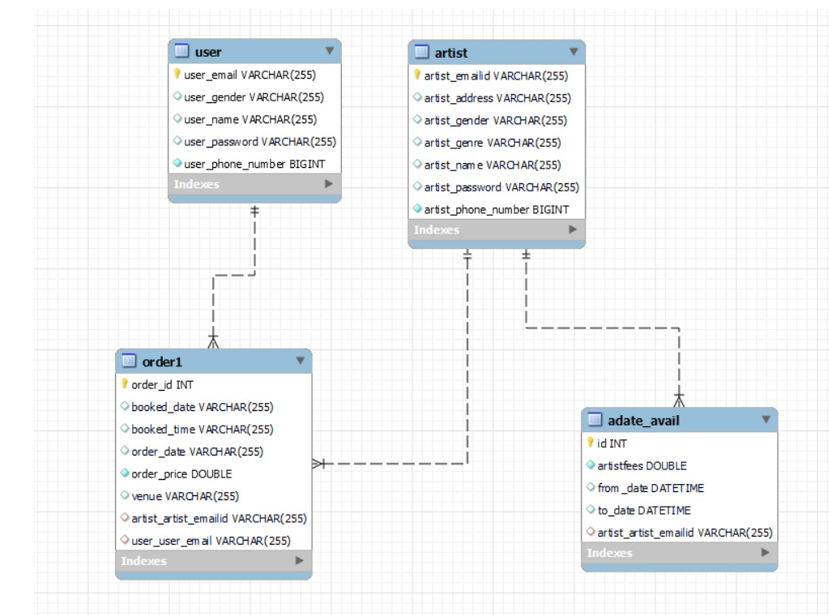
Rew Check Availability



Rew Login

**Figure: State Diagram**

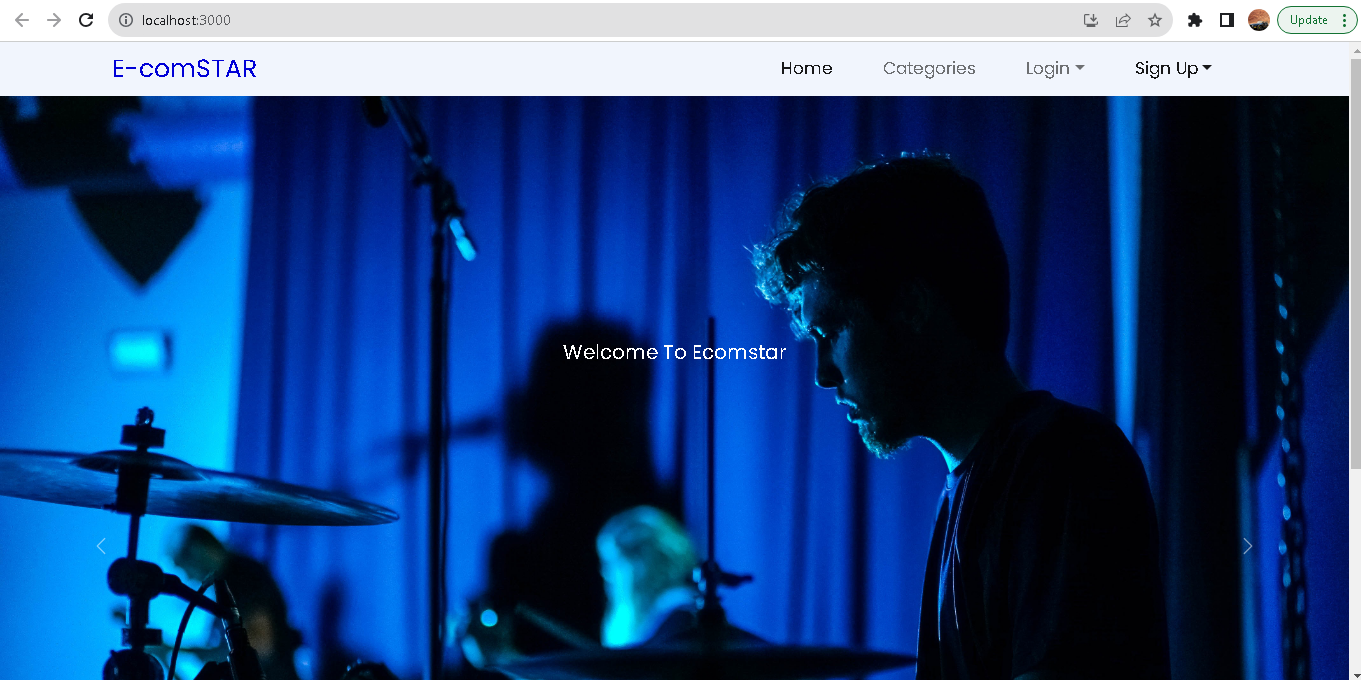




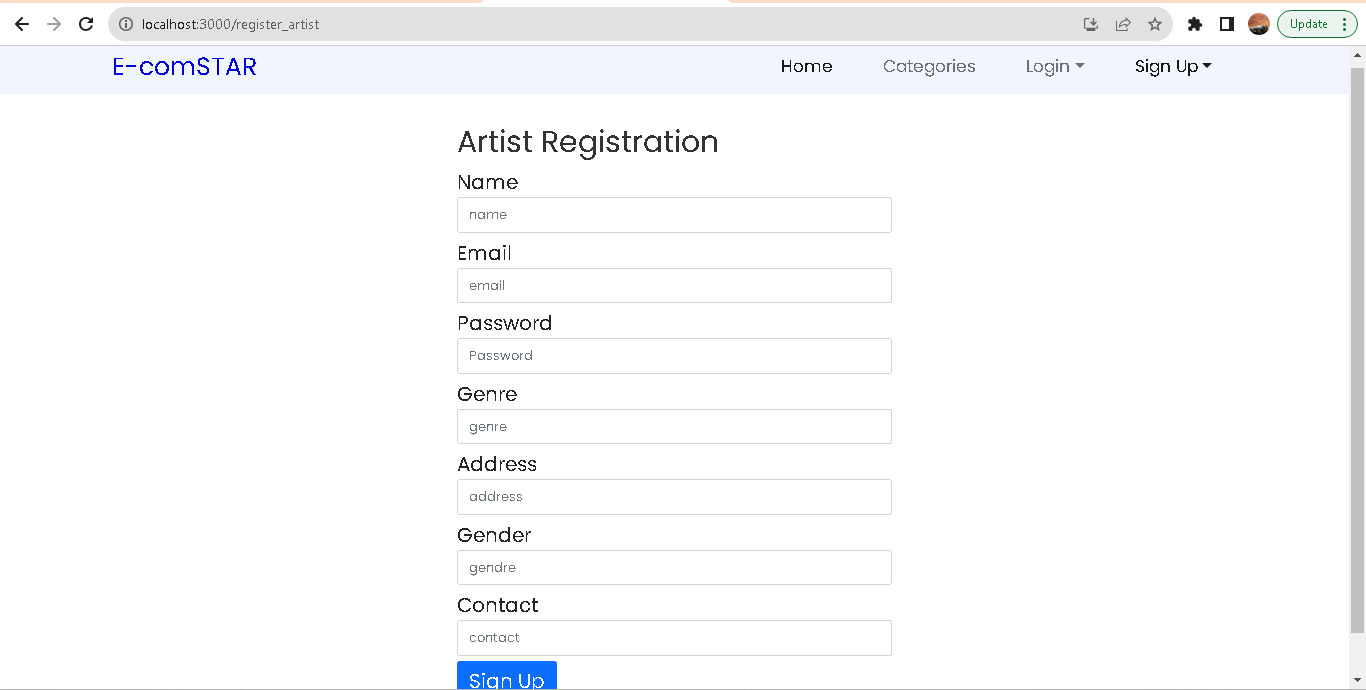
**Figure: Class Diagram**

**Glimpses of the Application**

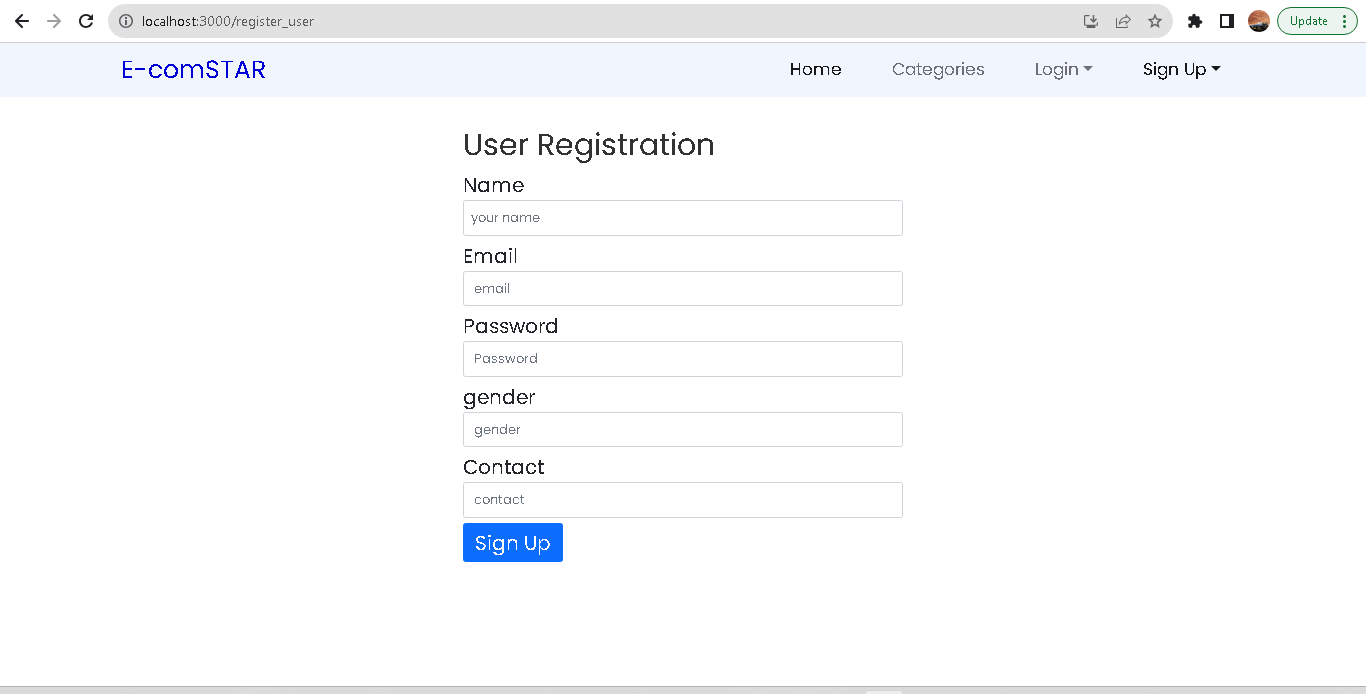
**Home Page**



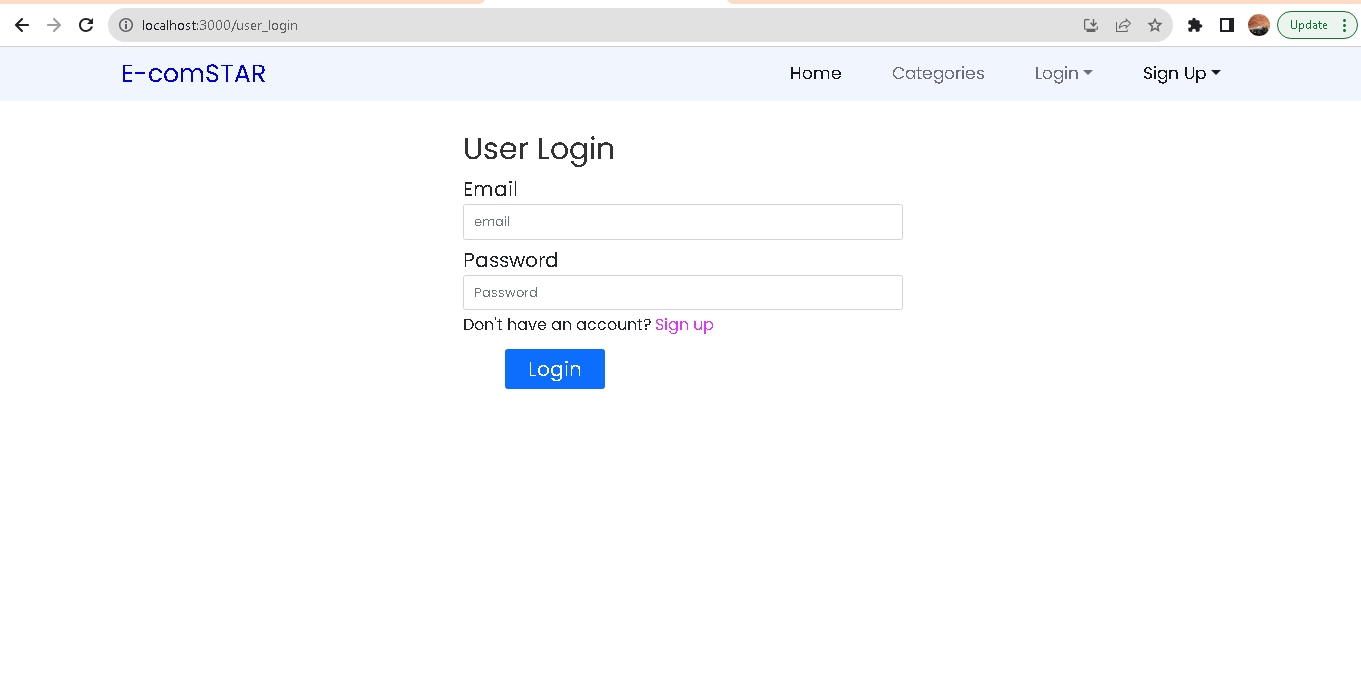
**Artist Registration**

****

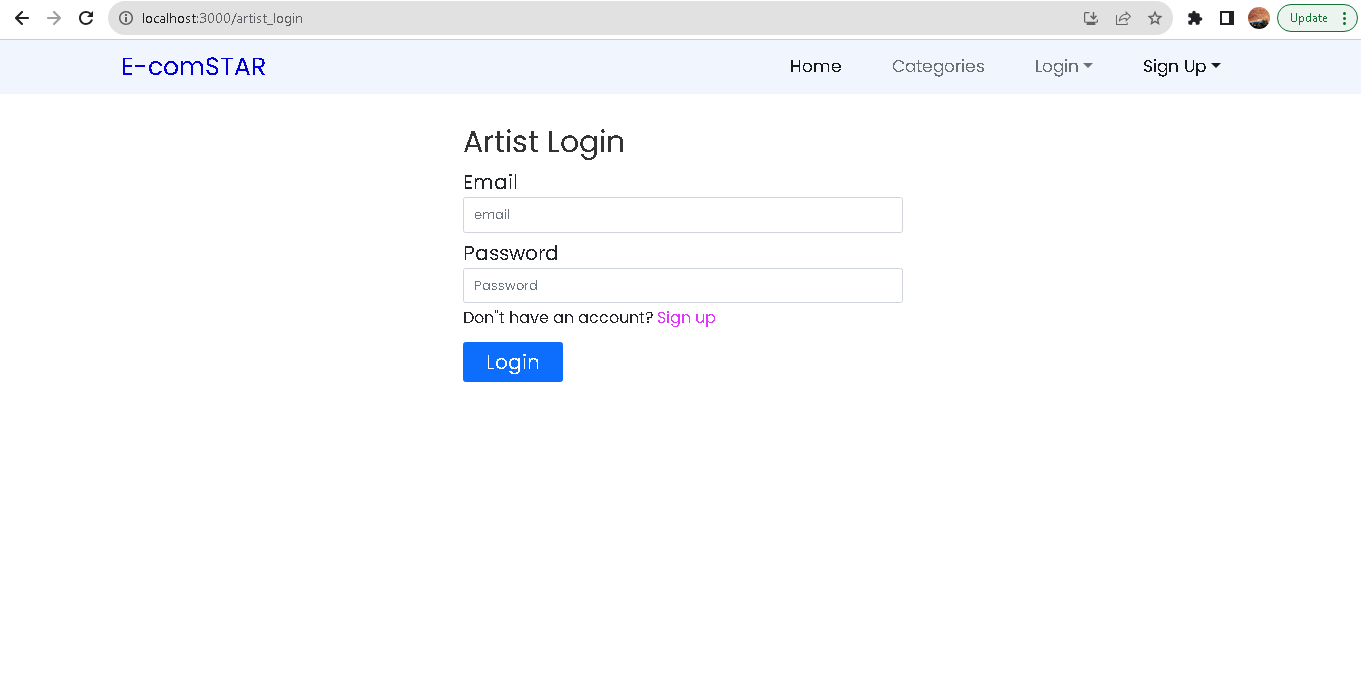
**User Registration**

****

**User Login**

****

**Artist Login**

****