**Project Name: Artist Booking System**

**Project Member:**

* **Rahul Parmar (190843120053)**
* **Justin Fernandes (190843120027)**
* **Abhishek Kadam (190843120002)**
* **Tansukh Kumar (190843120065)**

**Abstract:**

**BookMyArtist** 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 analyse 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 angular 8.In order to develop an e-commerce website, a number of Technologies must be studied and understood.These include multi-tiered architechture, server,and client side scripting techniques,implementation technolgies such as Spring boot,Angular 8 and relational database MySql.This document will discuss each of underlying technologies to create and mplement 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 BookMyArtist 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

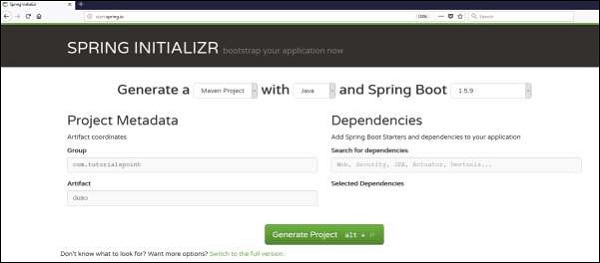
Spring Boot Auto Configuration automatically configures your Spring application based on the JAR dependencies you added in the project. For example, if MySQL database is on your class path, but you have not configured any database connection, then Spring Boot auto configures an in-memory database.

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

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.tutorialspoint</groupId>

<artifactId>demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>demo</name>

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

<parent>

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

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

<version>1.5.8.RELEASE</version>

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

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

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

</properties>

<dependencies>

<dependency>

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

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

</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.tutorialspoint.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication {

public static void main(String[] args) {

SpringApplication.run(DemoApplication.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 −

package com.tutorialspoint.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication

@RestController

public class DemoApplication {

public static void main(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=9090

## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)

spring.datasource.url = jdbc:mysql://localhost:3306/CdacProject?useSSL=false

spring.datasource.username = root

spring.datasource.password = welcome

## Hibernate Properties

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

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect

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

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

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

spring.jpa.hibernate.ddl-auto = update

**Angular 8:**

Angular 8 is a client-side TypeScript based framework which is used to create dynamic web application. Dynamic web applications are simply dynamic websites i.e. www.gmail.com, www.yahoo.com, etc. which has tendency to change data/information with respect to 3 parameter.

## Create an Angular 8 project

* **ng new BookMyArtist**

## Install Bootstrap 4 CSS framework

* **npm install bootstrap --save**

**Now, add the following code inside the angular.json file.**

"styles": [

   "src/styles.css",

   "./node\_modules/bootstrap/dist/css/bootstrap.min.css"

 ],

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

* **ng serve -o**

The server starts at the <http://localhost:4200/>

## Generate Angular Components

* **ng g c homepage**

The above component is automatically added to **app.module.ts** file. Now, we have to configure the routing of angular components inside an **app-routing.module.ts** file.

You can check the **app-routing.module.ts** file inside the src >> app folder in your project file. It is created because when we install an angular app, we permit **angular cli** to generate the routing file for us.

Now, write the following code inside an **app-routing.module.ts** file:

import { HomeComponent } from './home/home.component';

const routes: Routes = [

  {path:'',redirectTo:'/Home',pathMatch:'full'},

  {path:'Home',component:HomeComponent},

];

@NgModule({

  imports: [RouterModule.forRoot(routes)],

  exports: [RouterModule]

})

export class AppRoutingModule { }

export const rougtingComponants = [ArtistregitrationComponent,UserregitrationComponent,

UserloginComponent,ArtistloginComponent,WelcomeArtistComponent,WelcomeuserComponent,

ArtistListForUserComponent,TestComponent,OrderconfirmationComponent,AdminloginComponent,WelcomeadminComponent];

## Configure the HttpClientModule

## The Front-end applications always need HTTP protocol to communicate with the backend services. Modern browsers support two different APIs for making HTTP requests: the javaHttpRequest interface and the fetch() API.

mport { Injectable } from '@angular/core';

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

@Injectable({

  providedIn: 'root'

})

export class UserserviceService {

  constructor(private http:HttpClient) { }

  public registerUser(user)

  {

      return this.http.post("http://localhost:9090/userslist",user);

  }}

}

## Create a model file

## Inside the src >> app folder, create one file called **user.ts** and add the following code.

export class User{

    constructor(

      userEmail:string,

         userName:string,

              userPhoneNumber:number,

       userAdddress:string,

       userPassword:string,

       userDob:string,

      typeOfUser:string,

    )

     {

     }

}

## Create an Angular Service file

Type the following command to generate the service file.

* **Ng g s user**

**Tables:**

* **User**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Column Name** | **Type** |  |  |  |  |
| user\_email | varchar(255) | NO | PRI |  |  |
| type\_of\_user | varchar(255) | YES |  |  |  |
| user\_adddress | varchar(255) | YES |  |  |  |
| user\_dob | 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\_certification | varchar(255) | YES |  |  |  | | artist\_dob | varchar(255) | YES |  |  |  | | artist\_experience | varchar(255) | YES |  |  |  | | artist\_genre | varchar(255) | YES |  |  |  | | artist\_name | varchar(255) | YES |  |  |  | | artist\_password | varchar(255) | YES |  |  |  | | artist\_phone\_number | bigint(20) | NO |  |  |  | | artist\_picture | varchar(255) | YES |  |  |  | | pref\_work\_hours | varchar(255) | YES |  |  |  |  * **Orders**  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **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 |  |  |  * **Availability**  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **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 |  |  | |  |  |  |  |  |
|  |  |  |  |  |  |

User

Admin

Artist

**Figure:Use Case Daigram**

Login

Search

Artist

Admin

(Add ,view)

Artist

Login

Update availability

Order

Homepage

**Figure:Component Diagram**

User

Rew

Rew

Rew

Registration

Login

Wait for availabilility check

Book Now

Check Availability

Search Artist

Order confirmation

**Figure:State Diagram**

**Figure:State Diagram**

Rew

Rew

Rew

Logout

Logout

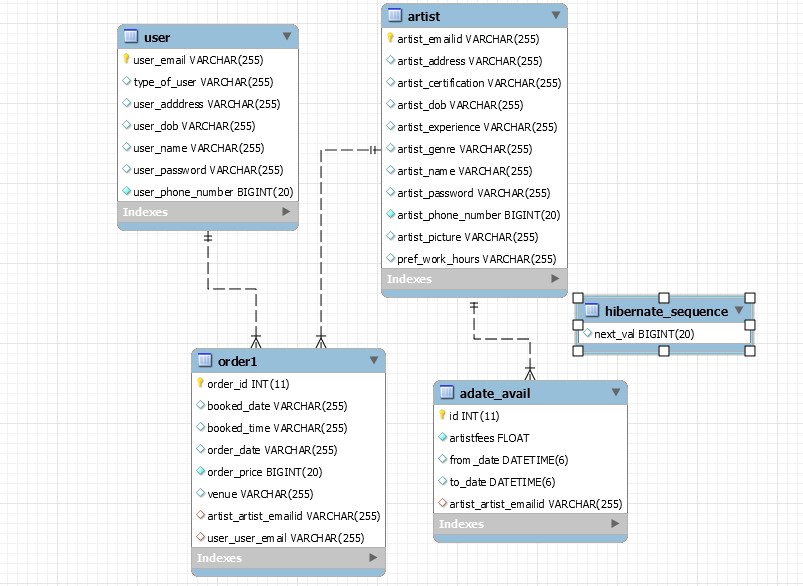
Check Orders

Provide Availability

Login

Artist

Registration



**Figure:Class Diagram**