**Project on**

**Spring**

**And**

**Angular**

**Using MySql**

Developed By:

Chaitanya Saurya

S201171400339

**Project on**

**Spring**

**And**

**Angular**

**Using MySql**

Batch Code: S201171400339

Start Date of Project: 08/08/20

End Date of Project: 29/08/20

Name of the coordinator: Lopamudra Ma’am

Name of the developer: Chaitanya Saurya

**Certificate**

This is to certify that this report titled Spring And Angular using MySQL embodies the original work done by Chaitanya Saurya, in partial fulfilment of his course requirement at NIIT.

Coordinator:

**Acknowledgment**

I cannot express enough thanks to my mentor for their continued support and encouragement: Lopamudra Ma’am. I offer my sincere appreciation for the learning opportunities provided by my mentor. Thanks to my parents as well.

**Abstract**

This project gives you a brief understanding of Springboot and Angular and MySQL database. Springboot is used for backend programming, Angular is used for front-end and MySQL is used for creating various tables and storing data to database.

**Configuration**

Operating System used: Microsoft Windows 10

Software used: 1. Eclipse

2. MySQL Workbench 8.0 CE

3. Microsoft Visual

**Table of Contents:**

1. Chapter 1: Introduction
2. Spring
3. AngularJS
4. MySQL
5. Chapter 2: Project Analysis
6. Spring Components used
7. AngularJS Components used
8. Chapter 3: Implementation Manual

**Chapter 1**

**Introduction**

**Spring**

Spring is the most popular application development framework for enterprise Java. Millions of developers around the world use Spring Framework to create high performing, easily testable, and reusable code.

Spring framework is an open source Java platform. It was initially written by Rod Johnson and was first released under the Apache 2.0 license in June 2003.

Spring is lightweight when it comes to size and transparency. The basic version of Spring framework is around 2MB.

The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building web applications on top of the Java EE platform. Spring framework targets to make J2EE development easier to use and promotes good programming practices by enabling a POJO-based programming model.

## Benefits of Using the Spring Framework

Following is the list of few of the great benefits of using Spring Framework −

* Spring enables developers to develop enterprise-class applications using POJOs. The benefit of using only POJOs is that you do not need an EJB container product such as an application server but you have the option of using only a robust servlet container such as Tomcat or some commercial product.
* Spring is organized in a modular fashion. Even though the number of packages and classes are substantial, you have to worry only about the ones you need and ignore the rest.
* Spring does not reinvent the wheel, instead it truly makes use of some of the existing technologies like several ORM frameworks, logging frameworks, JEE, Quartz and JDK timers, and other view technologies.
* Testing an application written with Spring is simple because environment-dependent code is moved into this framework. Furthermore, by using JavaBeanstyle POJOs, it becomes easier to use dependency injection for injecting test data.
* Spring's web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks such as Struts or other over-engineered or less popular web frameworks.
* Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.
* Lightweight IoC containers tend to be lightweight, especially when compared to EJB containers, for example. This is beneficial for developing and deploying applications on computers with limited memory and CPU resources.
* Spring provides a consistent transaction management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (using JTA, for example).

**Angular**

Angular is a platform and framework for building single-page client applications using HTML and TypeScript. Angular is written in TypeScript. It implements core and optional functionality as a set of TypeScript libraries that you import into your apps.

The architecture of an Angular application relies on certain fundamental concepts. The basic building blocks are NgModules, which provide a compilation context for components. NgModules collect related code into functional sets; an Angular app is defined by a set of NgModules. An app always has at least a root module that enables bootstrapping, and typically has many more feature modules.

* Components define views, which are sets of screen elements that Angular can choose among and modify according to your program logic and data.
* Components use services, which provide specific functionality not directly related to views. Service providers can be injected into components as dependencies, making your code modular, reusable, and efficient.

Modules, components and services are classes that use decorators. These decorators mark their type and provide metadata that tells Angular how to use them.

* The metadata for a component class associates it with a template that defines a view. A template combines ordinary HTML with Angular directives and binding markup that allow Angular to modify the HTML before rendering it for display.
* The metadata for a service class provides the information Angular needs to make it available to components through dependency injection (DI).

An app's components typically define many views, arranged hierarchically. Angular provides the Router service to help you define navigation paths among views. The router provides sophisticated in-browser navigational capabilities.

See the Angular Glossary for basic definitions of important Angular terms and usage.

**Modules**

Angular NgModules differ from and complement JavaScript (ES2015) modules. An NgModule declares a compilation context for a set of components that is dedicated to an application domain, a workflow, or a closely related set of capabilities. An NgModule can associate its components with related code, such as services, to form functional units.

Every Angular app has a root module, conventionally named AppModule, which provides the bootstrap mechanism that launches the application. An app typically contains many functional modules.

Like JavaScript modules, NgModules can import functionality from other NgModules, and allow their own functionality to be exported and used by other NgModules. For example, to use the router service in your app, you import the Router NgModule.

Organizing your code into distinct functional modules helps in managing development of complex applications, and in designing for reusability. In addition, this technique lets you take advantage of lazy-loading—that is, loading modules on demand—to minimize the amount of code that needs to be loaded at startup.

**Components**

Every Angular application has at least one component, the root component that connects a component hierarchy with the page document object model (DOM). Each component defines a class that contains application data and logic, and is associated with an HTML template that defines a view to be displayed in a target environment.

The @Component() decorator identifies the class immediately below it as a component, and provides the template and related component-specific metadata.

Decorators are functions that modify JavaScript classes. Angular defines a number of decorators that attach specific kinds of metadata to classes, so that the system knows what those classes mean and how they should work.

### **Templates, directives, and data binding**

A template combines HTML with Angular markup that can modify HTML elements before they are displayed. Template directives provide program logic, and binding markup connects your application data and the DOM. There are two types of data binding:

* Event binding lets your app respond to user input in the target environment by updating your application data.
* Property binding lets you interpolate values that are computed from your application data into the HTML.

Before a view is displayed, Angular evaluates the directives and resolves the binding syntax in the template to modify the HTML elements and the DOM, according to your program data and logic. Angular supports two-way data binding, meaning that changes in the DOM, such as user choices, are also reflected in your program data.

Your templates can use pipes to improve the user experience by transforming values for display. For example, use pipes to display dates and currency values that are appropriate for a user's locale. Angular provides predefined pipes for common transformations, and you can also define your own pipes.

## Services and dependency injection

For data or logic that isn't associated with a specific view, and that you want to share across components, you create a service class. A service class definition is immediately preceded by the @Injectable() decorator. The decorator provides the metadata that allows other providers to be **injected** as dependencies into your class.

Dependency injection (DI) lets you keep your component classes lean and efficient. They don't fetch data from the server, validate user input, or log directly to the console; they delegate such tasks to services.

### **Routing**

The Angular Router NgModule provides a service that lets you define a navigation path among the different application states and view hierarchies in your app. It is modeled on the familiar browser navigation conventions:

* Enter a URL in the address bar and the browser navigates to a corresponding page.
* Click links on the page and the browser navigates to a new page.
* Click the browser's back and forward buttons and the browser navigates backward and forward through the history of pages you've seen.

The router maps URL-like paths to views instead of pages. When a user performs an action, such as clicking a link, that would load a new page in the browser, the router intercepts the browser's behavior, and shows or hides view hierarchies.

If the router determines that the current application state requires particular functionality, and the module that defines it hasn't been loaded, the router can lazy-load the module on demand.

The router interprets a link URL according to your app's view navigation rules and data state. You can navigate to new views when the user clicks a button or selects from a drop box, or in response to some other stimulus from any source. The router logs activity in the browser's history, so the back and forward buttons work as well.

To define navigation rules, you associate navigation paths with your components. A path uses a URL-like syntax that integrates your program data, in much the same way that template syntax integrates your views with your program data. You can then apply program logic to choose which views to show or to hide, in response to user input and your own access rules.

**MySQL**

**MySQL** is an open-source relational database management system (RDBMS). Its name is a combination of "My", the name of co-founder Michael Widenius's daughter, and "SQL", the abbreviation for Structured Query Language. A relational database organizes data into one or more data tables in which data types may be related to each other; these relations help structure the data. SQL is a language programmers use to create, modify and extract data from the relational database, as well as control user access to the database. In addition to relational databases and SQL, an RDBMS like MySQL works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups.

MySQL is free and open-source software under the terms of the GNU General Public License, and is also available under a variety of proprietary licenses. MySQL was owned and sponsored by the Swedish company MySQL AB, which was bought by Sun Microsystems (now Oracle Corporation). In 2010, when Oracle acquired Sun, Widenius forked the open-source MySQL project to create MariaDB.

MySQL has stand-alone clients that allow users to interact directly with a MySQL database using SQL, but more often MySQL is used with other programs to implement applications that need relational database capability. MySQL is a component of the LAMP web application software stack (and others), which is an acronym for *Linux, Apache, MySQL, Perl/PHP/Python*. MySQL is used by many database-driven web applications, including Drupal, Joomla, phpBB, and WordPress. MySQL is also used by many popular websites, including Facebook, Flickr, MediaWiki, Twitter, and YouTube.

MySQL is written in C and C++. Its SQL parser is written in yacc, but it uses a home-brewed lexical analyzer.[]](https://en.wikipedia.org/wiki/MySQL#cite_note-16) MySQL works on many system platforms, including AIX, BSDi, FreeBSD, HP-UX, eComStation, i5/OS, IRIX, Linux, macOS, Microsoft Windows, NetBSD, Novell NetWare, OpenBSD, OpenSolaris, OS/2 Warp, QNX, Oracle Solaris, Symbian, SunOS, SCO OpenServer, SCO UnixWare, Sanos and Tru64. A port of MySQL to OpenVMS also exists.

The MySQL server software itself and the client libraries use dual-licensing distribution. They are offered under GPL version 2, or a proprietary license.

Support can be obtained from the official manual. Free support additionally is available in different IRC channels and forums. Oracle offers paid support via its MySQL Enterprise products. They differ in the scope of services and in price. Additionally, a number of third party organisations exist to provide support and services.

MySQL has received positive reviews, and reviewers noticed it "performs extremely well in the average case" and that the "developer interfaces are there, and the documentation (not to mention feedback in the real world via Web sites and the like) is very, very good”. It has also been tested to be a "fast, stable and true multi-user, multi-threaded SQL database server.

**Chapter 2**

**Project Analysis**

**Spring Components Used**

Dependencies Used:

1. Spring web
2. Spring Data JPA
3. SpringBoot Dev Tools
4. MySQL Server Driver packages

Annotations Used:

1. @ComponentScan- This annotation is used with @Configuration annotation to allow Spring to know the packages to scan for annotated components. @ComponentScan is also used to specify base packages using basePackageClasses or basePackage attributes to scan. If specific packages are not defined, scanning will occur from the package of the class that declares this annotation.
2. @Configuration- This annotation is used on classes which define beans. @Configuration is an analog for XML configuration file – it is configuration using Java class. Java class annotated with @Configuration is a configuration by itself and will have methods to instantiate and configure the dependencies.
3. @EnableTransactionManagement- Enables Spring's annotation-driven transaction management capability, similar to the support found in Spring's <tx:\*> XML namespace. To be used on [@Configuration](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/Configuration.html) classes to configure traditional, imperative transaction management or reactive transaction management.
4. @EnableAutoConfiguration- This annotation is usually placed on the main application class. The @EnableAutoConfiguration annotation implicitly defines a base “search package”. This annotation tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings.
5. @Bean- This annotation is used at the method level. @Bean annotation works with @Configuration to create Spring beans. As mentioned earlier, @Configuration will have methods to instantiate and configure dependencies. Such methods will be annotated with @Bean. The method annotated with this annotation works as bean ID and it creates and returns the actual bean.
6. @RestController- This annotation is used at the class level. The @RestController annotation marks the class as a controller where every method returns a domain object instead of a view. By annotating a class with this annotation you no longer need to add @ResponseBody to all the RequestMapping method. It means that you no more use view-resolvers or send html in response. You just send the domain object as HTTP response in the format that is understood by the consumers like JSON.
7. @CrossOrigin- This annotation is used both at class and method level to enable cross origin requests. In many cases the host that serves JavaScript will be different from the host that serves the data. In such a case Cross Origin Resource Sharing (CORS) enables cross-domain communication. To enable this communication you just need to add the @CrossOrigin annotation.
8. @RequestMapping- This annotation is used both at class and method level. The @RequestMapping annotation is used to map web requests onto specific handler classes and handler methods. When @RequestMapping is used on class level it creates a base URI for which the controller will be used. When this annotation is used on methods it will give you the URI on which the handler methods will be executed. From this you can infer that the class level request mapping will remain the same whereas each handler method will have their own request mapping. Sometimes you may want to perform different operations based on the HTTP method used, even though the request URI may remain the same. In such situations, you can use the method attribute of @RequestMapping with an HTTP method value to narrow down the HTTP methods in order to invoke the methods of your class.
9. @Autowired- This annotation is applied on fields, setter methods, and constructors. The @Autowired annotation injects object dependency implicitly.When you use @Autowired on fields and pass the values for the fields using the property name, Spring will automatically assign the fields with the passed values.
10. @PostMapping- This annotation is used for mapping HTTP POST requests onto specific handler methods. @PostMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.POST)
11. @GetMapping- This annotation is used for mapping HTTP GET requests onto specific handler methods. @GetMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET)
12. @DeleteMapping- This annotation is used for mapping HTTP DELETE requests onto specific handler methods. @DeleteMapping is a composed annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.DELETE)
13. @ResponseBody- This annotation is used to annotate request handler methods. The @ResponseBody annotation is similar to the @RequestBody annotation. The @ResponseBody annotation indicates that the result type should be written straight in the response body in whatever format you specify like JSON or XML. Spring converts the returned object into a response body by using the HttpMessageConveter.
14. @Repository- This annotation is used on Java classes which directly access the database. The @Repository annotation works as marker for any class that fulfills the role of repository or Data Access Object.This annotation has a automatic translation feature. For example, when an exception occurs in the @Repository there is a handler for that exception and there is no need to add a try catch block.
15. @Entity- The @Entity annotation specifies that the class is an entity and is mapped to a database table. The @Table annotation specifies the name of the database table to be used for mapping.
16. @Component- Indicates that an annotated class is a “component”. Such classes are considered as candidates for auto-detection when using annotation-based configuration and classpath scanning.
17. @Id- The @Id annotation specifies the primary key of an entity.
18. @GeneratedValue- The @GeneratedValue provides for the specification of generation strategies for the values of primary keys.
19. @Service- This annotation is used on a class. The @Service marks a Java class that performs some service, such as execute business logic, perform calculations and call external APIs. This annotation is a specialized form of the @Component annotation intended to be used in the service layer.
20. @SpringBootApplication- This annotation is used on the application class while setting up a Spring Boot project. The class that is annotated with the @SpringBootApplication must be kept in the base package. The one thing that the @SpringBootApplication does is a component scan. But it will scan only its sub-packages.

**Angular Components Used**

Commands Used:

* 1. ng new StudentAngular1- . Creates a new project.
  2. npm install bootstrap@3.3.7 --save– Installs Bootsrap.
  3. ng g c <Name> -Generates Angular Components.
  4. npm install angular-datatables@6.0.0 –save – Installs Datatables.
  5. ng serve –open – To start the application on localhost:8080.

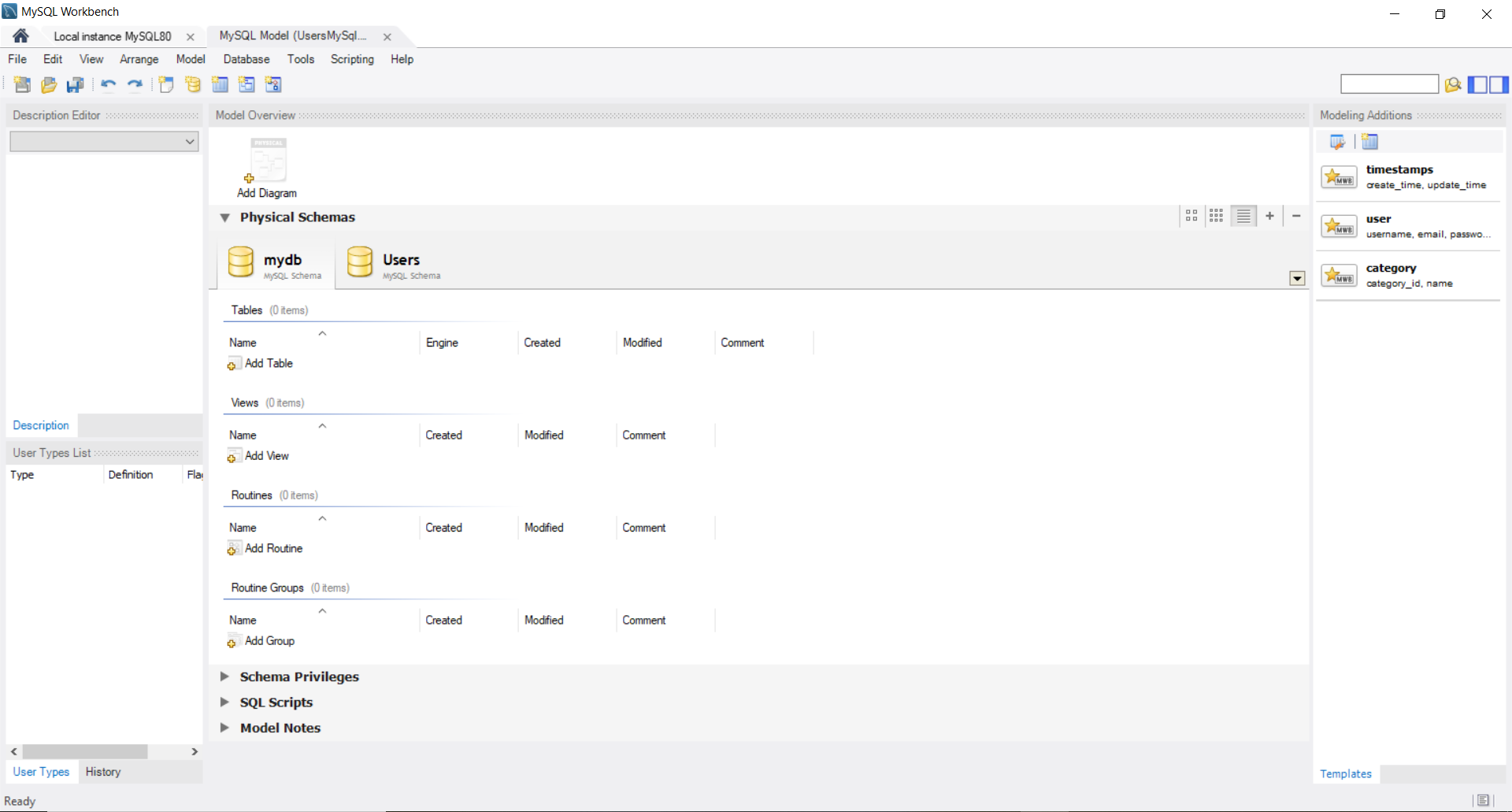
Edit the **app.module.ts** file-

* 1. **Import Routing** - Here, we are importing routing file (app-routing.module.ts) and include it in imports array.
  2. **Import ReactiveFormsModule** - Here, we are importing **ReactiveFormsModule** for reactive forms and specify it in imports array.
  3. **Import HttpModule** - Here, we are importing **HttpModule** for server requests and specifying it in imports array.
  4. **Register Service class** - Here, we are mentioning the service class in provider's array.

**Chapter 3:**

**Implementation Manual**

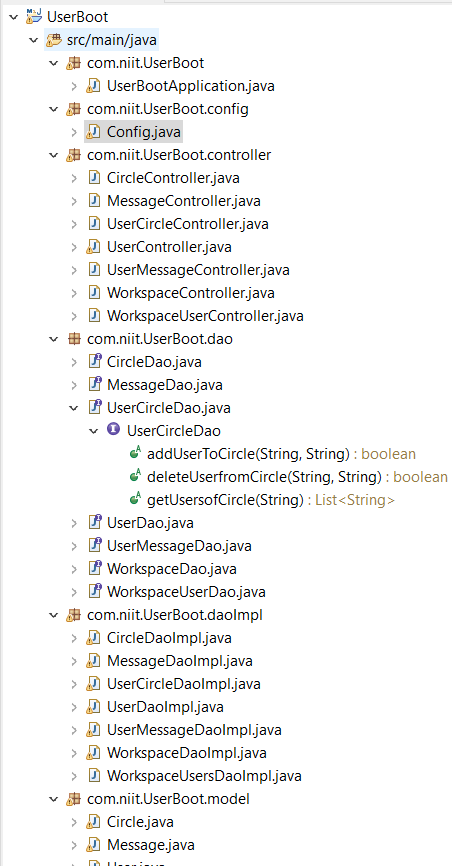
**MySQL WorkBench**

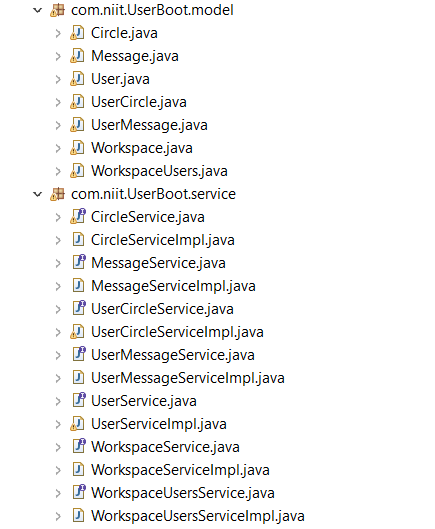


Here I have created mydb Database where all the tables will be formed and the data will be stored in it.

**Eclipse**

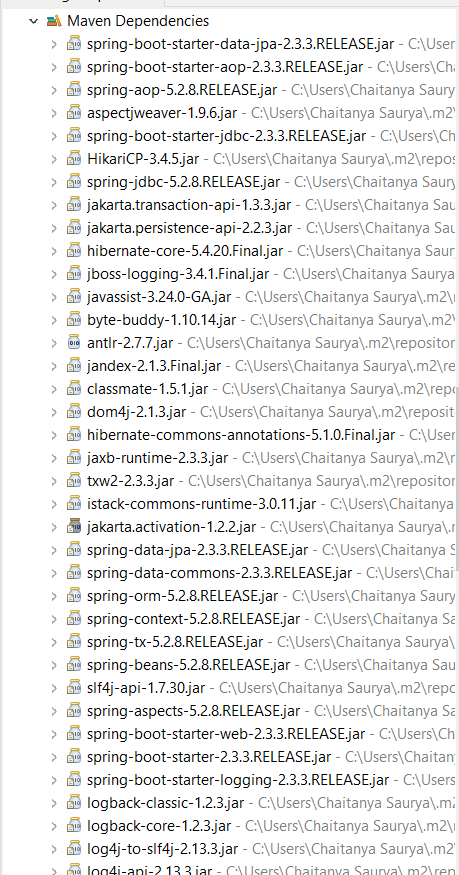
1. Project Structure-

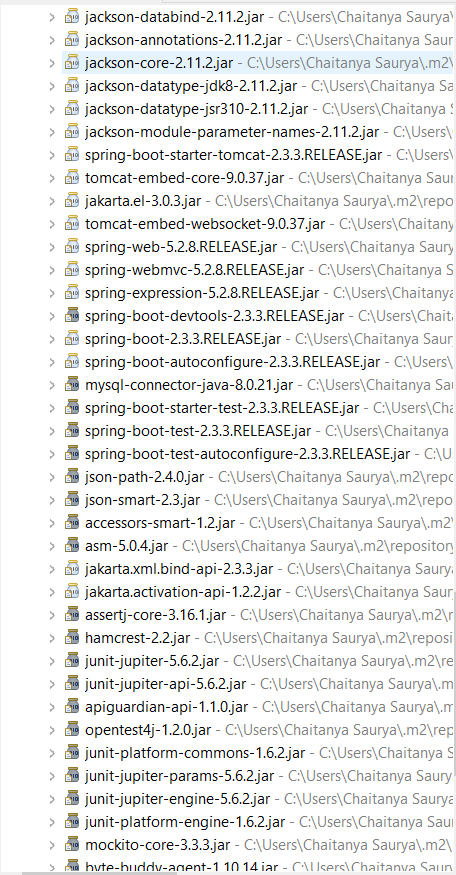




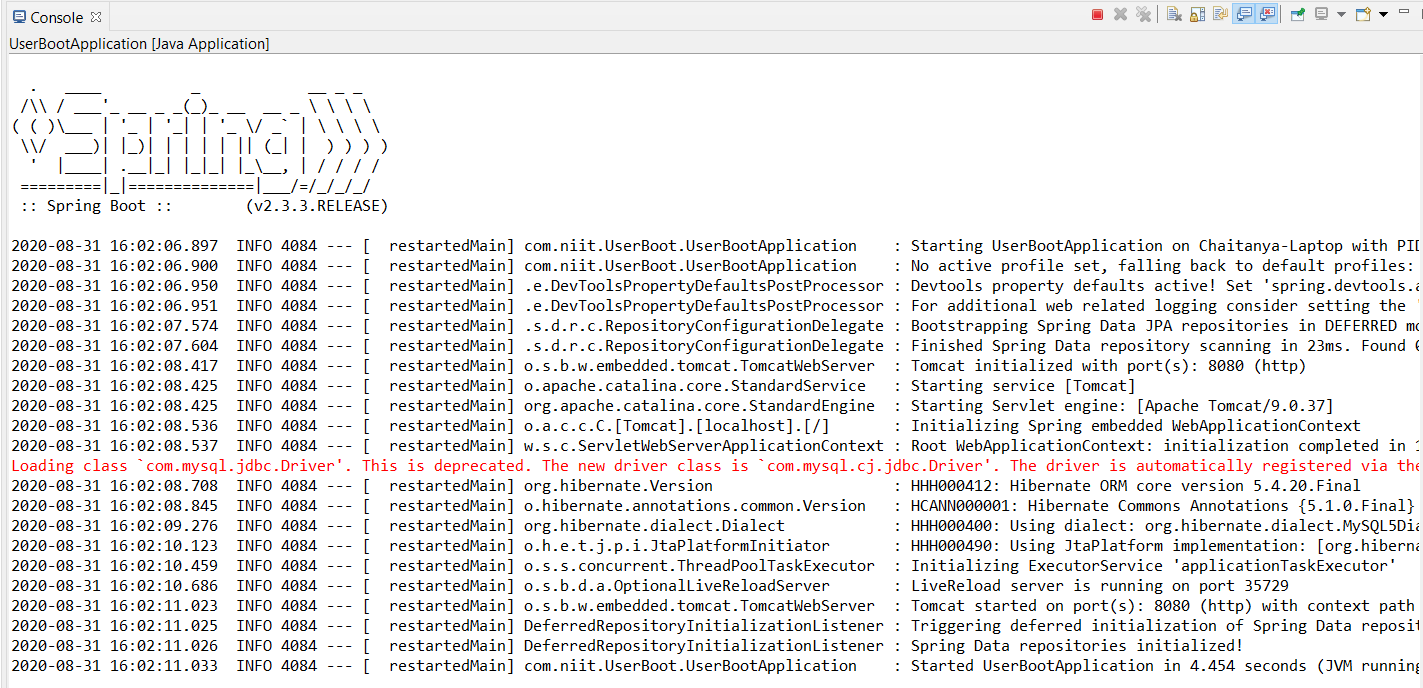
Here I have created 7 packages.

1. com.niit.UserBoot- It has properties to run the application.
2. com.niit.UserBoot.config- It has various configuration properties.
3. com.niit.UserBoot.controller- It has properties that will process the various request from client.
4. com.niit.UserBoot.dao- It has various interface Java class.
5. com.niit.UserBoot.daoImpl- Implementing DAO design pattern.
6. com.niit.UserBoot.model- It is just a plain container for user data, so it doesn't implement any other behavior worth stressing.Of course, the most relevant design choice that we need to make here is how to keep the application that uses this class isolated from any persistence mechanism that could be implemented at some point.
7. com.niit.UserBoot.service- These classes that provide some business functionalities. Spring context will autodetect these classes when annotation-based configuration and classpath scanning is used.
8. Dependencies Used:



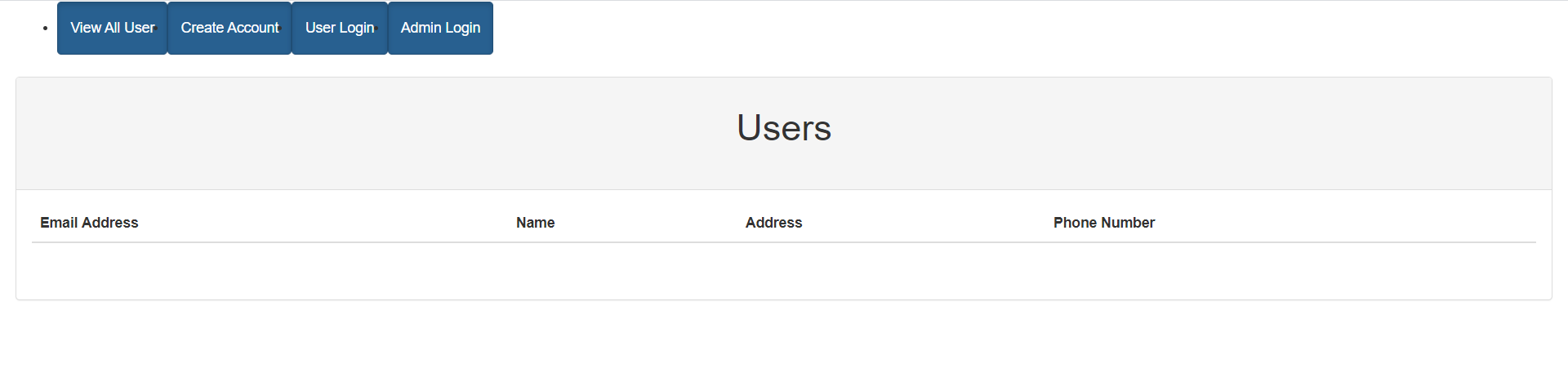


1. On Running The Spring Application-



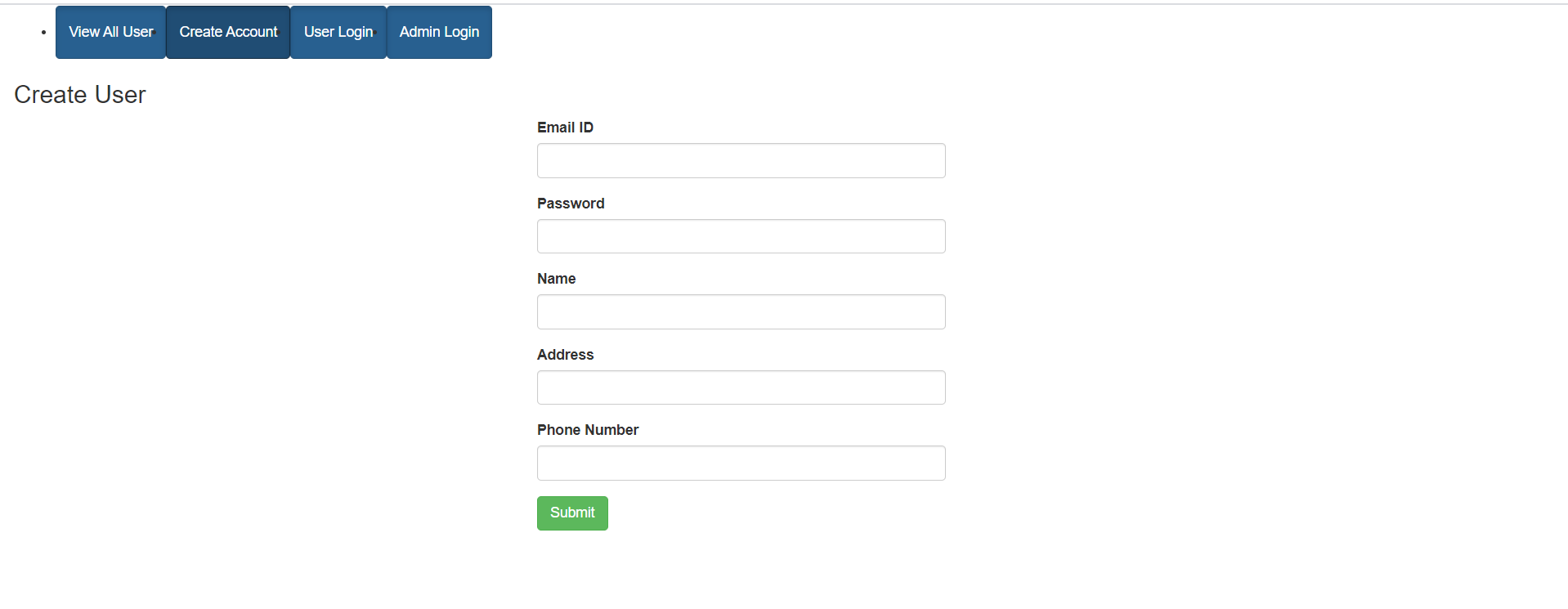
**Angular**

1. View All Users



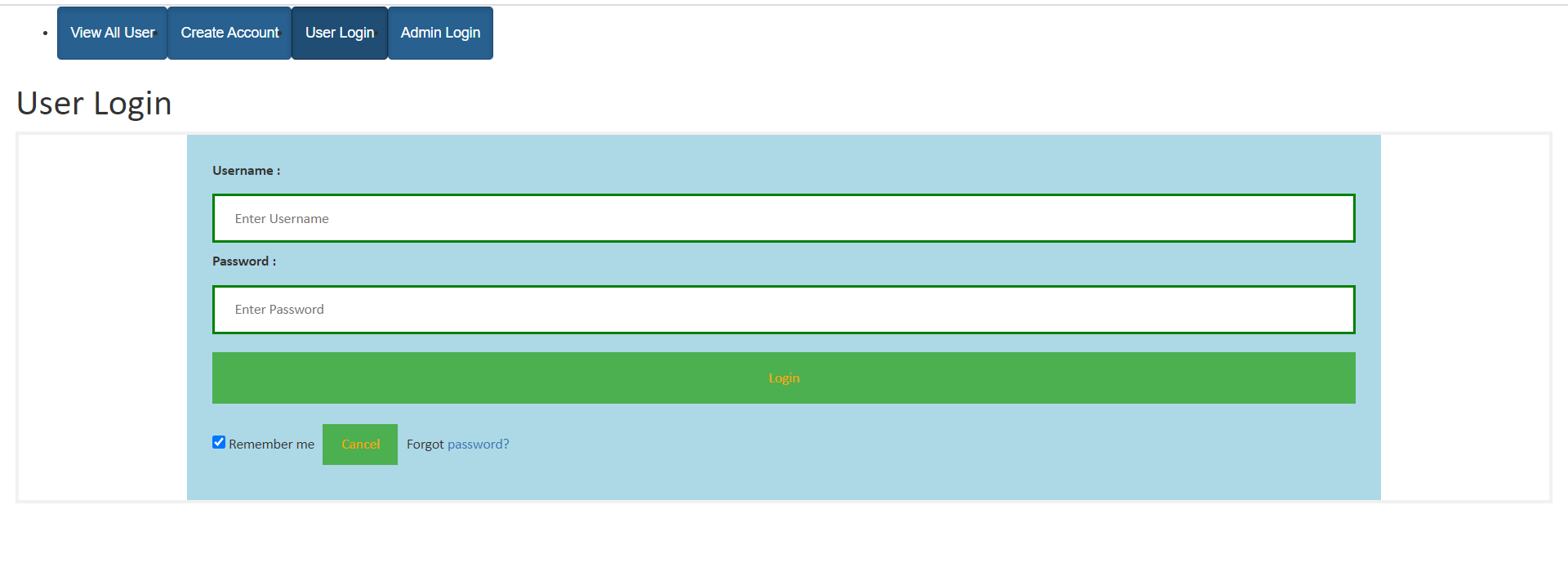
Here we can see the list of all the users and their email ID, Name, Address ,Phone Number.

1. Create Account-



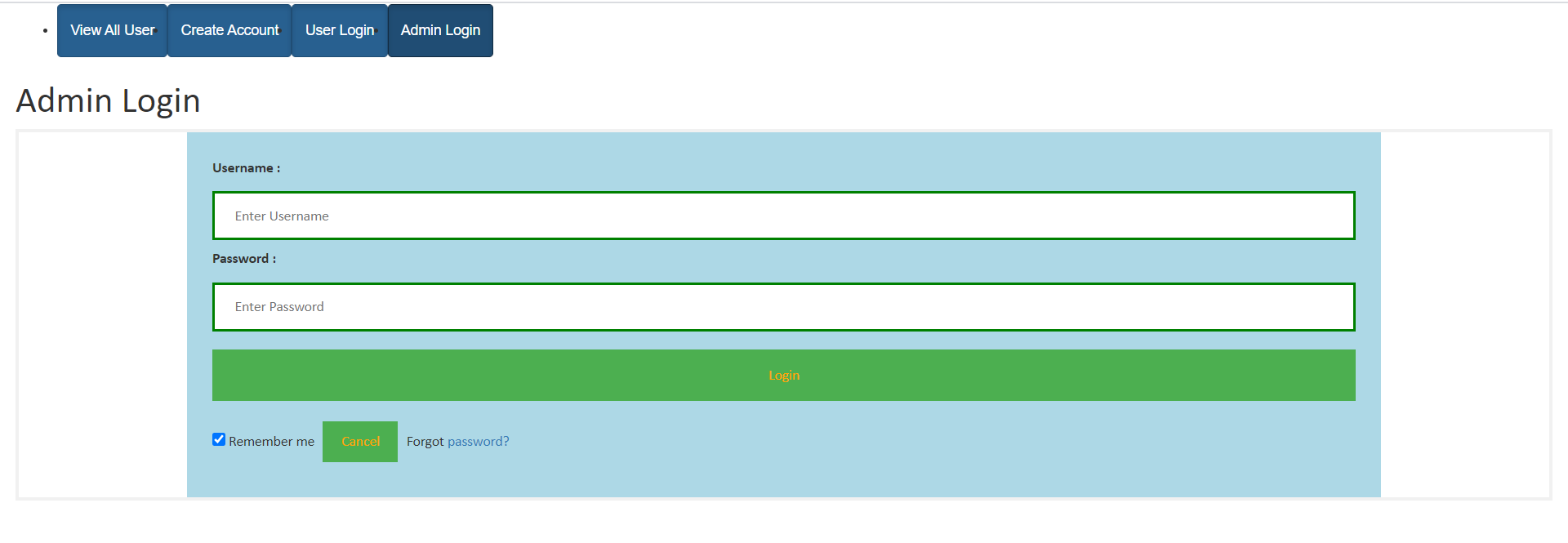
Here a User Can create their account by giving following details as shown in the picture above.

1. User Login-



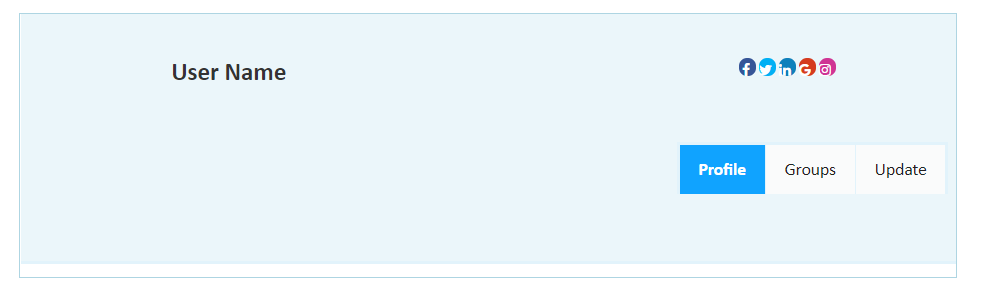
Here a Registered User can login to their account by providing correct credentials.

1. Admin Login-



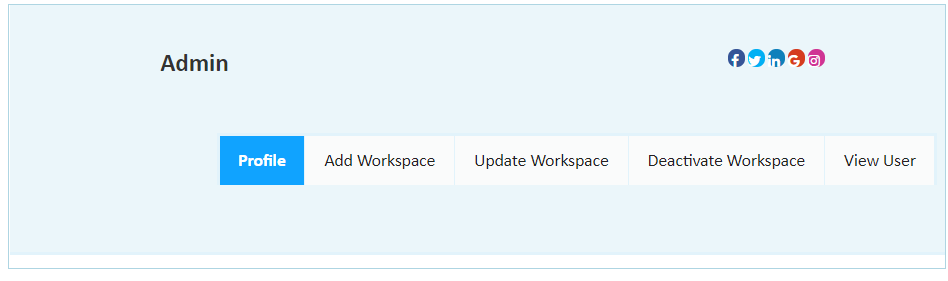
Here a Registered Admin can login to their account by providing correct credentials.

1. User Account-



Here the User Account in which User can see their profile details, Join groups by clicking on Groups tab and update their information by clicking on Update tab.

1. Admin Account-



Here the Admin can see their profile details, Add a new Workspace, Update an existing Workspace, Deactivate a Workspace and View any User account.

Thank You