**Approach Taken:**

ORM: I have used hibernate for ORM over any other tools. The purpose of using hibernate was to decrease the boiler plate code. I could have used JPA as well but it was time consuming process to configure and use the same. Another advantage of using hibernate is it makes easy to perform CRUD operations in case of complex queries. We can make use of HQL (Hibernate Query Language) to perform these complex operations.

Spring Boot: I have used Spring boot in this project as it handles all security by itself and it also eliminates the need of writing servlet. Spring boot integrates the properties file automatically. We can do the same in java but it is again a time-consuming process. Spring boot also handles Factory bean by itself using annotations and we can perform dependency injection using annotations. This again helps us to eliminate the need of boiler plate code to handle all these functionalities in pure Java.

Front end: I have used JSP along with html, CSS and bootstrap to develop web pages. I had an option to use JavaScript for the same along with html, CSS and bootstrap which could have been an advantage as the page would have not been refreshed. JavaScript calls AJAX to connect to my controller which does not give refresh effect on web page. It was time consuming process and hence, I opted for JSP for development of web pages. In case I decide to go for any other technology for front end, it will not be difficult as I have use view resolver to define the type of frontend page to be called.

**Architecture:**

I have used 2-Tier architecture where I have divided entire project in Data Access Object (now referred to as DAO) and Controller layer. All the database CRUD operations are performed in DAO layer. This layer also contains a logic to connect with database.

Second layer is Service layer where I have defined all the controllers and logics related to same. Service layer will communicate with DAO layer to perform operations on database tables.

**Design Patterns Used:**

I have used below design patterns for developing this project:

1. Dependency Injection:

I have not created objects of any classes manually. I have used spring Autowired concept for dependency injection. All the classes are annotated with Component for this purpose.

1. Factory Method:

I have used factory method by creating bean factory of all classes and using these bean as per project requirements.

1. Null Object:

I have tried to avoid null objects at all points where objects are created. This is to avoid any method throwing a null pointer exception.

**Extra stuffs I would have done if I had more time:**

1. Security:

Below are security modules I would like to implement if I had more time.

* I would have implemented 2 factor authentications for login.
* I am storing password in regular text format in database. I would have used 1 way encryption method to increase security which means I would add salt (a random text) in password at random position of password and encrypt the password. Encrypted password and salt would be saved in database table. I would then use this salt for authentication every time to login but will never decrypt at any point for security point of view.
* I would have implemented authorization feature like OAuth2 for login.

1. Separate Service layer and Controller layer:

If I have had more time, I would have separated service layer and controller layer in project.

1. Separate bean classes for DAO and Service layer:

If I have had more time, I would have separated bean classes for each layer so there would not be any dependencies on each layer.

1. Well-designed UI:

If I have had more time, I would have created a well-designed UI in either Angular or Node JS.

1. More testing:

I would have performed integration testing and have had performed more unit test for methods in all layers of my project.

1. Caching:

I would have used caching for some tables in database to decrease overall calls to database.