**CHAPTER 1**

**INTRODUCTION**

The main objective of this project is sports club management. It is organizing and managing all the activities in sport clubs. This is a web based application that makes it easier to access all activities in the browser itself. It is developed to automate the tasks of entering all the ground bookings in a book, batch timings records and also maintaining the users and timings. The system will help the users to request the ground access to administrator and according to his approval ground bookings are done. There are two separate users one is Admin and the others are regular Users.

User will be able to see all the updates and news of the sports club, users need to register in order to access the sport club’s facilities. After registering and logging in they will be able to request to book the grounds for any events and also track the status of the request. Users can also register for the available training batches in batch registration.

Admin should login with their credentials in the login page to go to the admin portal, they will be able to see the users list in order the maintain the records, admin will be able to see the requests for the ground, he can also approve or decline the request made by the user based on the ground availability, he will also be able to see the batches that users have registered.

**CHAPTER 2**

**SYSTEM ANALYSIS**

Analysis may be defined as the process of dividing the project into parts, identifying each part and establishing relationship between them.

Analysis is a detailed study of various operations performs by a system and their relationship within and outside of the system.

One aspect of analysis is identifying the boundaries of system and determining whether or not a candidate system should consider other related system. In the specific content of users, system analysis comprises of taking known facts concerning a system breaking them into elements and establishing logical relationship between the elements with the objective of producing the specification requirements.

During analysis, data are collected on available files, decision points and transaction handled by the present system. This can be done only in a disciplined way, using appropriate tools. Data flow diagram interviews, On-site observation and questionnaires are examples: Training, experience and common sense are required for the collection of information needed to do analysis.

**2.1 EXISTING SYSTEM**

File is always lost because of human environment. Sometimes due to some human error there may be a loss of records. File is always lost due to some accident like spilling of water by some member on file accidentally. Besides some natural disaster like floods or fires may also damage the files. Difficult to search record when there is no computerized system there is always a difficulty in searching of records if the records are large in number.

**Disadvantages:**

* File got damaged**.**
* Space consuming.
* Cost consuming.

**2.2 PROPOSED SYSTEM**

Sports club management system is a web based application that helps in managing the sports club based activities. It is used by the club’s administrator to manage the daily activities of the club like track the users, plan the batches accordingly, taking actions on the ground requests to manage the ground bookings and etc. Batches and Ground maintenance modules are also in this system to manage the batches with the respective time accordingly and also the grounds event management. This all activities are computerized and there will be no loss of records or documents and cause problems which usually occurs in non-computerized systems.

**Advantages: -**

* Avoid the problem arises we need an automated system that keeps a track of all the records & related information
* All details of sports management club can be viewed at any point of time.
* Admin will able to manage the approval of ground request in click.
* It is capable of registering of batches without the help of admin.
* It provides better and efficient services to user and admin.
* Reduce the work of the employee in many ways.

**CHAPTER 3**

**SYSTEM SPECIFICATION**

**3.1 HARDWARE SPECIFICATION**

Processor: Intel(R) Core(TM)

Hard Disk: 250 GB

Keyboard: Standard PS/2

RAM: 3 GB

Speed: 3.20 GHz

**3.2 SOFTWARE SPECIFICATION**

Operating System: Windows 10.

Language: JAVA

Back end: MYSQL

**CHAPTER 4**

**SOFTWARE DESCRIPTION**

**4.1 FRONT END**

**SPRING MVC: -**

A Spring MVC is a Java framework which is used to build web applications. It follows the Model-View-Controller design pattern. It implements all the basic features of a core spring framework like Inversion of Control, Dependency Injection.

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.

A Spring MVC provides an elegant solution to use MVC in spring framework by the help of DispatcherServlet. Here, DispatcherServlet is a class that receives the incoming request and maps it to the right resource such as controllers, models, and views.



* **Model** - A model contains the data of the application. A data can be a single object or a collection of objects.
* **Controller** - A controller contains the business logic of an application. Here, the @Controller annotation is used to mark the class as the controller.
* **View** - A view represents the provided information in a particular format. Generally, JSP+JSTL is used to create a view page. Although spring also supports other view technologies such as Apache Velocity, Thymeleaf and FreeMarker.
* **Front Controller** - In Spring Web MVC, the DispatcherServlet class works as the front controller. It is responsible to manage the flow of the Spring MVC application.

## **Understanding the flow of Spring Web MVC: -**



1. As displayed in the figure, all the incoming request is intercepted by the DispatcherServlet that works as the front controller.
2. The DispatcherServlet gets an entry of handler mapping from the XML file and forwards the request to the controller.
3. The controller returns an object of ModelAndView.
4. The DispatcherServlet checks the entry of view resolver in the XML file and invokes the specified view component.

## **DispatcherServlet as the Heart of Spring MVC**

What we really want to do as developers of a web application is to abstract away the following tedious and boilerplate tasks and focus on useful business logic:

* Mapping an HTTP request to a certain processing method
* Parsing of HTTP request data and headers into data transfer objects (DTOs) or domain objects
* Model-view-controller interaction
* Generation of responses from DTOs, domain objects, etc.

The Dispatcher servlet provides exactly that. It is the heart of the Spring Web MVC framework; this core component receives all requests to your application.

As you’ll see, DispatcherServlet is very extensible. For example, it allows you to plug in different existing or new adapters for a lot of tasks:

* Map a request to a class or method that should handle it (implementations of the HandlerMappinginterface)
* Handle a request using a specific pattern, like a regular servlet
* Resolve views by name, allowing you to use different templating engines, XML, XSLT or any other view technology (implementations of the ViewResolver interface)
* Parse multipart requests by using the default Apache Commons file uploading implementation or writing your own MultipartResolver
* Resolve locale with any LocaleResolver implementation, including cookie, session, Accept HTTP header, or any other way of determining the locale expected by the user

## **Processing of an HTTP Request**

First, let’s trace the processing of simple HTTP requests to a method in your controller layer and back to the browser/client.

The DispatcherServlet has a long inheritance hierarchy; it’s worth understanding these individual aspects one by one, top-down. The request processing methods will interest us the most.

##### 

Understanding the HTTP request, both locally during standard development, [as well as remotely](https://stackify.com/prefix-remote-http-calls/), is a critical part of understanding the MVC architecture.

## **Advantages of Spring MVC Framework: -**

* **Separate roles** - The Spring MVC separates each role, where the model object, controller, command object, view resolver, DispatcherServlet, validator, etc. can be fulfilled by a specialized object.
* **Light-weight** - It uses light-weight servlet container to develop and deploy your application.
* **Powerful Configuration** - It provides a robust configuration for both framework and application classes that includes easy referencing across contexts, such as from web controllers to business objects and validators.
* **Rapid development** - The Spring MVC facilitates fast and parallel development.
* **Reusable business code** - Instead of creating new objects, it allows us to use the existing business objects.
* **Easy to test** - In Spring, generally we create JavaBeans classes that enable you to inject test data using the setter methods.
* **Flexible Mapping** - It provides the specific annotations that easily redirect the page.

**MAVEN: -**

Maven is a project management and comprehension tool that provides developers a complete build lifecycle framework. Development team can automate the project's build infrastructure in almost no time as Maven uses a standard directory layout and a default build lifecycle.

In case of multiple development teams environment, Maven can set-up the way to work as per standards in a very short time. As most of the project setups are simple and reusable, Maven makes life of developer easy while creating reports, checks, build and testing automation setups.

Maven provides developers ways to manage the following −

* Builds
* Documentation
* Reporting
* Dependencies
* SCMs
* Releases
* Distribution
* Mailing list

To summarize, Maven simplifies and standardizes the project build process. It handles compilation, distribution, documentation, team collaboration and other tasks seamlessly. Maven increases reusability and takes care of most of the build related tasks.

## **Maven Evolution**

Maven was originally designed to simplify building processes in Jakarta Turbine project. There were several projects and each project contained slightly different ANT build files. JARs were checked into CVS.

Apache group then developed **Maven** which can build multiple projects together, publish projects information, deploy projects, share JARs across several projects and help in collaboration of teams.

## **Objective**

The primary goal of Maven is to provide developer with the following −

* A comprehensive model for projects, which is reusable, maintainable, and easier to comprehend.
* Plugins or tools that interact with this declarative model.

Maven project structure and contents are declared in an xml file, pom.xml, referred as Project Object Model (POM), which is the fundamental unit of the entire Maven system. In later chapters, we will explain POM in detail.

## **Convention over Configuration**

Maven uses **Convention** over **Configuration**, which means developers are not required to create build process themselves.

Developers do not have to mention each and every configuration detail. Maven provides sensible default behavior for projects. When a Maven project is created, Maven creates default project structure. Developer is only required to place files accordingly and he/she need not to define any configuration in pom.xml.

As an example, following table shows the default values for project source code files, resource files and other configurations. Assuming, **${basedir}**denotes the project location −

|  |  |
| --- | --- |
| Item | Default |
| source code | ${basedir}/src/main/java |
| Resources | ${basedir}/src/main/resources |
| Tests | ${basedir}/src/test |
| Complied byte code | ${basedir}/target |
| distributable JAR | ${basedir}/target/classes |

In order to build the project, Maven provides developers with options to mention life-cycle goals and project dependencies (that rely on Maven plugin capabilities and on its default conventions). Much of the project management and build related tasks are maintained by Maven plugins.

Developers can build any given Maven project without the need to understand how the individual plugins work. We will discuss Maven Plugins in detail in the later chapters.

**Features of Maven**

* Simple project setup that follows best practices.
* Consistent usage across all projects.
* Dependency management including automatic updating.
* A large and growing repository of libraries.
* Extensible, with the ability to easily write plugins in Java or scripting languages.
* Instant access to new features with little or no extra configuration.
* Model-based builds − Maven is able to build any number of projects into predefined output types such as jar, war, metadata.
* Coherent site of project information − Using the same metadata as per the build process, maven is able to generate a website and a PDF including complete documentation.
* Release management and distribution publication − Without additional configuration, maven will integrate with your source control system such as CVS and manages the release of a project.
* Backward Compatibility − You can easily port the multiple modules of a project into Maven 3 from older versions of Maven. It can support the older versions also.
* Automatic parent versioning − No need to specify the parent in the sub module for maintenance.
* Parallel builds − It analyzes the project dependency graph and enables you to build schedule modules in parallel. Using this, you can achieve the performance improvements of 20-50%.
* Better Error and Integrity Reporting − Maven improved error reporting, and it provides you with a link to the Maven wiki page where you will get full description of the error.

**JSP: -**

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc

### **Advantages of JSP over Servlet: -**

There are many advantages of JSP over the Servlet. They are as follows:

#### 1) **Extension to Servlet**

JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

#### 2) **Easy to maintain**

JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

#### 3) **Fast Development: No need to recompile and redeploy**

If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

#### 4) **Less code than Servlet**

In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

## **JSP Processing**

## The following steps explain how the web server creates the Webpage using JSP −

## All the above mentioned steps can be seen in the following diagram −

## As with a normal page, your browser sends an HTTP request to the web server.

## The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with .jsp instead of .html.

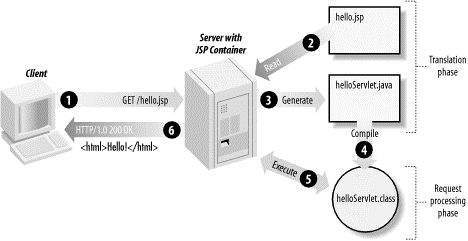
## The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code. This code implements the corresponding dynamic behavior of the page.

## The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.

## A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format. The output is furthur passed on to the web server by the servlet engine inside an HTTP response.

## The web server forwards the HTTP response to your browser in terms of static HTML content.

## Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.



Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated servlet still matches the JSP's contents. This makes the process more efficient than with the other scripting languages (such as PHP) and therefore faster.

So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular servlet.

**HIBERNATE: -**

Hibernate is a Java framework that simplifies the development of Java application to interact with the database. It is an open source, lightweight, ORM (Object Relational Mapping) tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence.

## **ORM Tool:**

An ORM tool simplifies the data creation, data manipulation and data access. It is a programming technique that maps the object to the data stored in the database.



The ORM tool internally uses the JDBC API to interact with the database.

## **What is JPA?**

he Java Persistence API (JPA) is a specification of Java. It is used to persist data between Java object and relational database. JPA acts as a bridge between object-oriented domain models and relational database systems.

As JPA is just a specification. It requires an implementation. So, ORM tools like Hibernate, TopLink and iBatis implements JPA specifications for data persistence.

## **Advantages of Hibernate Framework:**

### 1) Open Source and Lightweight

### 2) Fast Performance

### 3) Database Independent Query

### 4) Automatic Table Creation

### 5) Simplifies Complex Join

### 6) Provides Query Statistics and Database Status

# **Hibernate Architecture**

The Hibernate architecture includes many objects such as persistent object, session factory, transaction factory, connection factory, session, transaction etc.

* Java application layer
* Hibernate framework layer
* Backhand api layer
* Database layer
* 

This is the high level architecture of Hibernate with mapping file and configuration file.



Hibernate framework uses many objects such as session factory, session, transaction etc. alongwith existing Java API such as JDBC (Java Database Connectivity), JTA (Java Transaction API) and JNDI (Java Naming Directory Interface).

**4.2 BACKEND**

**MYSQL: -**

MySQL is the most popular Open Source **Relational SQL Database Management System**. MySQL is one of the best RDBMS being used for developing various web-based software applications. MySQL is developed, marketed and supported by MySQL AB, which is a Swedish company.

# **MySQL Features:**

* **It is secure**: MySQL consist of a solid data security layer that protects sensitive data from intruders. Passwords are encrypted in MySQL.
* **Client/ Server Architecture**: MySQL follows a client /server architecture. There is a database server (MySQL) and arbitrarily many clients (application programs), which communicate with the server; that is, they query data, save changes, etc.
* **It is scalable**: MySQL can handle almost any amount of data, up to as much as 50 million rows or more. The default file size limit is about 4 GB. However, you can increase this number to a theoretical limit of 8 TB of data.
* **Compatibale on many operating systems:** MySQL is compatible to run on many operating systems, like Novell NetWare, Windows\* Linux\*, many varieties of UNIX\* (such as Sun\* Solaris\*, AIX, and DEC\* UNIX), OS/2, FreeBSD\*, and others. MySQL also provides a facility that the clients can run on the same computer as the server or on another computer (communication via a local network or the Internet).
* **Allows roll-back:** MySQL allows transactions to be rolled back, commit and crash recovery.
* **High Performance:** MySQL is faster, more reliable and cheaper because of its unique storage engine architecture.
* **High Flexibility:** MySQL supports a large number of embedded applications which makes MySQL very flexible.
* **High Productivity:** MySQL uses Triggers, Stored procedures and views which allows the developer to give a higher productivity.

## **Disadvantages / Drawback of MySQL: -**

* MySQL version less than 5.0 doesn't support ROLE, COMMIT and stored procedure.
* MySQL does not support a very large database size as efficiently.
* MySQL doesn't handle transactions very efficiently and it is prone to data corruption.
* MySQL is accused that it doesn't have a good developing and debugging tool compared to paid databases.
* MySQL doesn't support SQL check constraints.

**Compatibility with other services: -**

MySQL was designed to be compatible with other systems. It supports deployment in

virtualized environments, such as Amazon RDS for MySQL, Amazon RDS for MariaDB and

Amazon Aurora for MySQL. Users can transfer their data to a SQL Server database by using

database migration tools like AWS Schema Conversion Tool and the AWS Database

Migration Service

**CHAPTER 5**

**PROJECT DESCRIPTION**

**5.1 PROBLEM DEFINITION**

The purpose of Sports Club Management is organizing and managing all the activities in sport clubs. It is developed to automate the tasks of entering all the ground bookings in a book, batch timings records and also maintaining the users and timings.

**DARWBACKS OF EXISTING SYSTEM**

* More man power.
* Time consuming.
* Consumes large volume of paper work.
* Needs manual calculations.
* No direct role for the higher officials.
* To avoid all these limitations and make the working more accurately the system needs

to be computerized in a better way.

**5.2 OVERVIEW OF THE PROJECT**

The concept of providing online services for the sports club management is to make the work of the sports club more easier for managing the crowd during the summer and focusing on the planning more than executing and collecting details that can be automated using this process in a simple website that users will be able to view and fulfill their necessities.

**ADVANTAGES OF PROPOSED SYSTEM**

The system is very simple in design and to implement. The system requires very low

system resources and the system will work in almost all configurations. It has got following

features;

* Minimize manual data entry.
* Minimum time needed for the various processing.
* Greater efficiency.
* Better service.

**5.3 MODULE DESCRIPTION**

**MODULES: -**

**User Modules:**

* User Registration
* Batch Registration
* Ground Registration
* Request Status

**Admin Modules:**

* Login Page
* User List
* Ground Request Details
* Batches List

**USER MODULES:**

**User Registration:**

Members will be able to register to the clubs website to get access to the Batches and Ground Registration. They will have their own Username, Password that helps while logging in.

**Batch Registration:**

Users will be allowed to register for the upcoming batches that are planned by the club and can be registered using this page in this module.

**Ground Registration:**

It is used by the members in order to request for the ground access at a particular date that is sent to the admin portal for approval

**Request Status:**

The requested ground access status can be viewed with the help of this module. Users can track the request status here.

**ADMIN MODULES:**

**Login Page:**

This page allows the access of the features of the club based onthe roles i.e Users or Admin. Admin credentials access the admin portal. Users to the member access pages.

**User List:**

Admin will be able to view the list of Users registered in the sports club and manage the facilites and plan according to the crowd.

**Ground Request Details:**

This page provides the details of the ground request and allows the admin to provide approval of the request or decline the request in a click of button.

**Batches List:**

This module helps the admin to view the list of the batches that the user s registered to manage the crowd and close the batch timings accordingly.

**5.4 DATAFLOW DIAGRAM**

**Data flow diagram (DFD)**

Data flow diagram models a system by using external entries from which data flows to

a process which transforms the data and create output data flow which go to other process or

external entries or files. Data in a file may also flow to process as inputs. Like flowcharts,

dataflow diagram can be used at any desired level of abstraction.

The main merits of DFD is that it can provide of what data a system wood process, what

transformation of date are done, what files are used, and where result flow .The graphical

representation of the system makes it a good communication tool between a user and an

analyst.

The four symbols used in DFDs are:

**--** A rectangle represents the data source or data destination

-- The circle represents the process

-- An arrow represents the flow of data

-- An open ended rectangle the data storage

**5.4.1 Level 0:**

AdminController

Data Store

Main Controller

User Interface

User Controller

**5.4.2 Level 1:**

Batch or Ground

Registration

Database

**5.4.3 Level 2:**

Ground Request List

Database

Database

**s**

**5.5 DATABASE DESIGN**

This database used in this project is similar in structure to that of the relational database,

while considering the database architecture there are two main categories such as relational

database and document database. In the document database the data values are maintained as

a file in which all the information are stored in document.But the relational database is like

tables, which are represented, in physical database as stored files. They have their own

independent existence. A table consists of rows and columns. Each column corresponds to a

piece of information called fields. A set of fields constitutes a record. The record contains all

the information, specific to the particular item.

**5.5.1: Table Name : USERS**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **DATATYPE** | **DESCRIPTION** |
| fullname | VARCHAR | Full Name |
| mobile | VARCHAR | Mobile Number |
| email | VARCHAR | Email Address |
| address | VARCHAR | Address |
| role | INT | Role of the User |
| username | VARCHAR | User Name |
| password | VARCHAR | Password |
| userid | INT | User Id |

**5.5.2: Table Name : BATCHES**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **DATATYPE** | **DESCRIPTION** |
| id | INT | Id |
| sport\_name | VARCHAR | Sport Name |
| sport\_time | VARCHAR | Sport Batch timing |
| desc | VARCHAR | Description |

**5.5.3: Table Name: GROUNDS**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **DATATYPE** | **DESCRIPTION** |
| id | INT | Id |
| event | VARCHAR | Event Name |
| date | DATE | Date of the event |
| desc | VARCHAR | Description |
| status | VARCHAR | Status |
| userid | INT | Foreign Key:: To the USERS table ID |

**CHAPTER 6**

**APPENDICES**

**6.1 SAMPLE SOURCE CODE:**

**Sport-Servlet.XML (Dispatcher Servlet)::**

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

<**beans** xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:p=*"http://www.springframework.org/schema/p"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-3.0.xsd*

*http://www.springframework.org/schema/tx*

*http://www.springframework.org/schema/tx/spring-tx.xsd*

*http://www.springframework.org/schema/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc-3.2.xsd"*>

<**context:component-scan** base-package=*"com.sport2"* />

<**mvc:annotation-driven**/>

<**bean** class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<**property** name=*"prefix"* value=*"/WEB-INF/views/"* />

<**property** name=*"suffix"* value=*".jsp"* />

</**bean**>

<**bean** id=*"dataSource"* class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*>

<**property** name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"* />

<**property** name=*"url"* value=*"jdbc:mysql://localhost:3306/sports\_club\_man"* />

<**property** name=*"username"* value=*"alpha"* />

<**property** name=*"password"* value=*"Alpha\_123"* />

</**bean**>

<**bean** id=*"jdbcTemplate"* class=*"org.springframework.jdbc.core.JdbcTemplate"*>

<**property** name=*"dataSource"* ref=*"dataSource"* />

</**bean**>

<**bean** id=*"entityManagerFactoryBean"* class=*"org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"*>

<**property** name=*"dataSource"* ref=*"dataSource"* />

<**property** name=*"packagesToScan"* value=*"com.sport2.bean"* />

<**property** name=*"jpaVendorAdapter"*>

<**bean** class=*"org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter"* />

</**property**>

<**property** name=*"jpaProperties"*>

<**props**>

<**prop** key=*"hibernate.hbm2ddl.auto"*>update</**prop**>

<**prop** key=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</**prop**>

<**prop** key=*"hibernate.show\_sql"*>true</**prop**>

</**props**>

</**property**>

</**bean**>

<**bean** id=*"transactionManager"* class=*"org.springframework.orm.jpa.JpaTransactionManager"*>

<**property** name=*"entityManagerFactory"* ref=*"entityManagerFactoryBean"* />

</**bean**>

<**mvc:resources** mapping=*"/resources/\*\*"* location=*"/WEB-INF/resources/"* />

<**tx:annotation-driven**/> **</beans>**

**WEB.xml::**

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

<**web-app** xmlns=*"http://java.sun.com/xml/ns/javaee"*

version=*"2.5"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*>

<**display-name**>Archetype Created Web Application</**display-name**>

<**servlet**>

<**servlet-name**>dispatcher</**servlet-name**>

<**servlet-class**>org.springframework.web.servlet.DispatcherServlet </**servlet-class**>

<**init-param**>

<**param-name**>contextConfigLocation</**param-name**>

<**param-value**>/WEB-INF/sport-servlet.xml</**param-value**>

</**init-param**>

<**load-on-startup**>1</**load-on-startup**>

</**servlet**>

<**servlet-mapping**>

<**servlet-name**>dispatcher</**servlet-name**>

<**url-pattern**>/</**url-pattern**>

</**servlet-mapping**>

</**web-app**>

**User Controller::**

@Controller

public class UserController {

@Autowired

UserServiceI service;

@RequestMapping(value="/register", method=RequestMethod.POST)

public String User\_Register(@Valid @ModelAttribute("users") Users users,BindingResult result, ModelMap model){

if(result.hasErrors()){

return "user\_reg";

}

Users user = new Users();

System.out.println(users);

user.setFullname(users.getFullname());

user.setMobile(users.getMobile());

user.setEmail(users.getEmail());

user.setAddress(users.getAddress());

user.setUsername(users.getUsername());

user.setPassword(users.getPassword());

System.out.println(user);

Users reg\_user = service.save(user);

model.put("id", reg\_user.getId());

return "thankyou";}

@RequestMapping(value = "/ground", method = RequestMethod.POST)

public String User\_Ground(@Valid @ModelAttribute("ground") Ground\_Reg ground,BindingResult result, ModelMap model) {

if(result.hasErrors()){

return "ground\_reg";

}

Ground\_Reg g = new Ground\_Reg();

g.setEvent(ground.getEvent());

g.setDate(ground.getDate());

g.setDesc(ground.getDesc());

System.out.println(g);

Ground\_Reg gr = service.save\_ground(g);

model.put("id", gr.getId());

return "thankyou";}

@RequestMapping(value = "/batch", method = RequestMethod.POST)

public String User\_Batch(@Valid @ModelAttribute("batch") Batch\_Reg batch,BindingResult result, ModelMap model){

if(result.hasErrors()){

return "batch\_reg";

}

Batch\_Reg b = new Batch\_Reg();

b.setSport\_name(batch.getSport\_name());

b.setSport\_time(batch.getSport\_time());

b.setDesc(batch.getDesc());

System.out.println(b);

Batch\_Reg ba = service.save\_batch(b);

System.out.println(ba);

model.put("id", ba.getId());

return "thankyou";}

@RequestMapping("/status/{id}")

public String req\_status(ModelMap model, @PathVariable Integer id){ System.out.println(id);

List<Ground\_Reg> list = service.req\_status(id);

model.put("ground", list);

return "req\_status";

}

}

**User DAO::**

*@Repository*

public class UserDaoImpl implements UserDaoI {

*@PersistenceContext*

private EntityManager em;

public Users save(Users user) {

System.*out*.println(user);

em.persist(user);

return user;

}

public void update(Users user) {

}

public Batch\_Reg save\_batch(Batch\_Reg batch) {

em.createNativeQuery("INSERT INTO BATCHES (`sport\_name`, `sport\_time`, `desc`) VALUES (?,?,?)")

.setParameter(1, batch.getSport\_name())

.setParameter(2, batch.getSport\_time())

.setParameter(3, batch.getDesc())

.executeUpdate();

return batch;

}

public Ground\_Reg save\_ground(Ground\_Reg ground) {

em.persist(ground);

return ground;

}

*@SuppressWarnings*("unchecked")

public List<Ground\_Reg> req\_status(Integer id) {

Query sql = em.createQuery("select g from Ground\_Reg g where userid = :id");

sql.setParameter("id", id);

List<Ground\_Reg> ground = sql.getResultList();

return ground;

}

User Bean:

*@Entity*

*@Table*(name = "USERS")

public class Users {

*@Id*

*@GeneratedValue*(strategy=*GenerationType*.*IDENTITY*)

private Integer userid;

*@Size*(min = 3, message = "Enter a valid name..!")

*@NotNull*(message="Cannot be Null..!")

private String fullname;

*@Size*(min = 10, message = "Enter a valid Mobile Number..!")

*@NotNull*(message="Cannot be Null..!")

private String mobile;

*@Email*(message="Enter Valid Email..!")

*@NotNull*(message="Cannot be Null..!")

private String email;

*@Size*(min = 3, message = "Enter a valid Address..!")

*@NotNull*(message="Cannot be Null..!")

private String address;

private Integer role = 2;

*@Size*(min = 5, message = "Enter a valid Username..!")

*@NotNull*(message="Cannot be Null..!")

private String username;

*@Size*(min = 5, message = "Enter a valid Password..!")

*@NotNull*(message="Cannot be Null..!")

private String password;

public Users() {

super();

}

public Users(Integer id, String fullname, String mobile, String email, String address, Integer role, String username,

String password) {

super();

this.userid = id;

this.fullname = fullname;

this.mobile = mobile;

this.email = email;

this.address = address;

this.role = role;

this.username = username;

this.password = password;

}

public Integer getId() {

return userid;

}

public void setId(Integer userid) {

this.userid = userid;

}

public String getFullname() {

return fullname;

}

public void setFullname(String fullname) {

this.fullname = fullname;

}

public String getMobile() {

return mobile;

}

public void setMobile(String mobile) {

this.mobile = mobile;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public Integer getRole() {

return role;

}

public void setRole(Integer role) {

this.role = role;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

*@Override*

public String toString() {

return "Users [id=" + userid + ", fullname=" + fullname + ", mobile=" + mobile + ", email=" + email + ", address="

+ address + ", role=" + role + ", username=" + username + ", password=" + password + "]";

}

}}

**Welcome Page:**

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<%@ page isELIgnored="false" %>

<%@include file="header.jsp" %>

<%@include file="navigation.jsp"%>

<div class=*"main"*>

<div class=*"login"* >

<h3>User Login</h3><br>

<c:if test="${err != null }">

<p>${err}</p>

</c:if>

<c:if test="${logout != null }">

<p>${logout}</p>

</c:if>

<form action=*"usr\_login"* method=*"POST"*>

<div class=*"row"*>

<div class=*"form-group col-xs-5"*>

<label>Username</label>

<input type=*"text"* class=*"form-control"* name=*"username"*/>

</div>

</div>

<div class=*"row"*>

<div class=*"form-group col-xs-5"*>

<label>Password</label>

<input type=*"password"* class=*"form-control"* name=*"password"*/>

</div>

</div>

<input type=*"submit"* class=*"btn btn-light"*value=*"LOGIN"*/>

<a href=*"usr\_register"* class=*"btn btn-light"*>REGISTER</a>

</form>

</div>

<div class=*"news"*>

<h2>News Update</h2>

<ul>

<li>Ground Bookings are open..!</li>

<li>Batches for training are open for cricket</li>

<li>Batches for training are open for cricket</li>

</ul>

</div>

<div class=*"ground"*>

<p></p>

</div>

</div>

<%@include file="footer.jsp" %>

**Registration:**

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>

<%@include file="header.jsp" %>

<%@include file="navigation.jsp" %>

<div class=*"main-2"*>

<div class=*"form"*>

<h3>User Registration Form</h3><br>

<form:form method="POST" action="register" modelAttribute="users">

<div class=*"row"*>

<div class=*"form-group col-md-12"*>

<form:label class="col-md-3 control-lable" for="fullname" path="fullname">Full Name</form:label>

<div class=*"col-md-3"*>

<form:input type="text" class="form-control input-sm" name="fullname" path="fullname"></form:input>

<form:errors class="error" path="fullname"/>

</div>

</div>

</div>

<div class=*"row"*>

<div class=*"form-group col-md-12"*>

<form:label class="col-md-3 control-lable" for="mobile" path="mobile">Mobile Number</form:label>

<div class=*"col-md-3"*>

<form:input type="tel" class="form-control input-sm" name="mobile" path="mobile"></form:input>

<form:errors class="error" path="mobile"/>

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

<div class=*"row"*>

<div class=*"form-group col-md-12"*>

<form:label class="col-md-3 control-lable" for="email" path="email">Email</form:label>

<div class=*"col-md-3"* class=*"form-control input-sm"*>

<form:input type="email" class="form-control input-sm" name="email" path="email"></form:input>

<form:errors class="error" path="email"/>

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

<div class=*"row"*>

<div class=*"form-group col-md-12"*>

<form:label class="col-md-3 control-lable" for="address" path="address">Address</form:label>

<div class=*"col-md-3"* class=*"form-control input-sm"*>

<form:input type="text" class="form-control input-sm" name="address" path="address"></form:input>

<form:errors class="error" path="address"/>

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

<div class=*"row"*>

<div class=*"form-group col-md-12"*>

<form:label class="col-md-3 control-lable" for="username" path="username">Username</form:label>

<div class=*"col-md-3"* class=*"form-control input-sm"*>

<form:input type="text" class="form-control input-sm" name="username" path="username"></form:input>

<form:errors class="error" path="username"/>

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

<div class=*"row"*>

<div class=*"form-group col-md-12"*>

<form:label class="col-md-3 control-lable" for="password" path="password">Password</form:label>

<div class=*"col-md-3"* class=*"form-control input-sm"*>

<form:input type="password" class="form-control input-sm" name="password" path="password"></form:input>

<form:errors class="error" path="password"/>

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

<div class=*"row buttons"*>

<input type=*"submit"* value=*"REGISTER"* class=*"btn btn-primary btn-sm"*>

<button type=*"reset"* class=*"btn btn-sm btn-light"*>CLEAR</button>

</div>

</form:form><br>

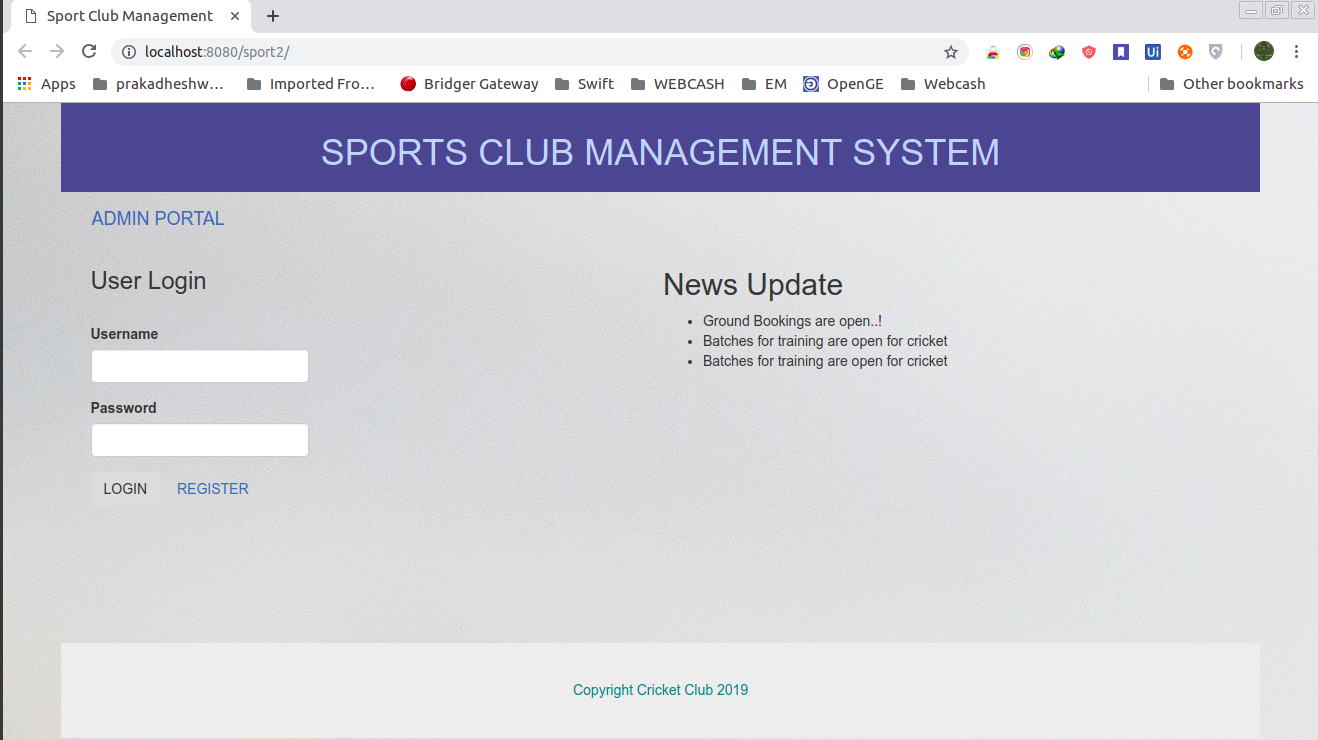
</div>

</div>

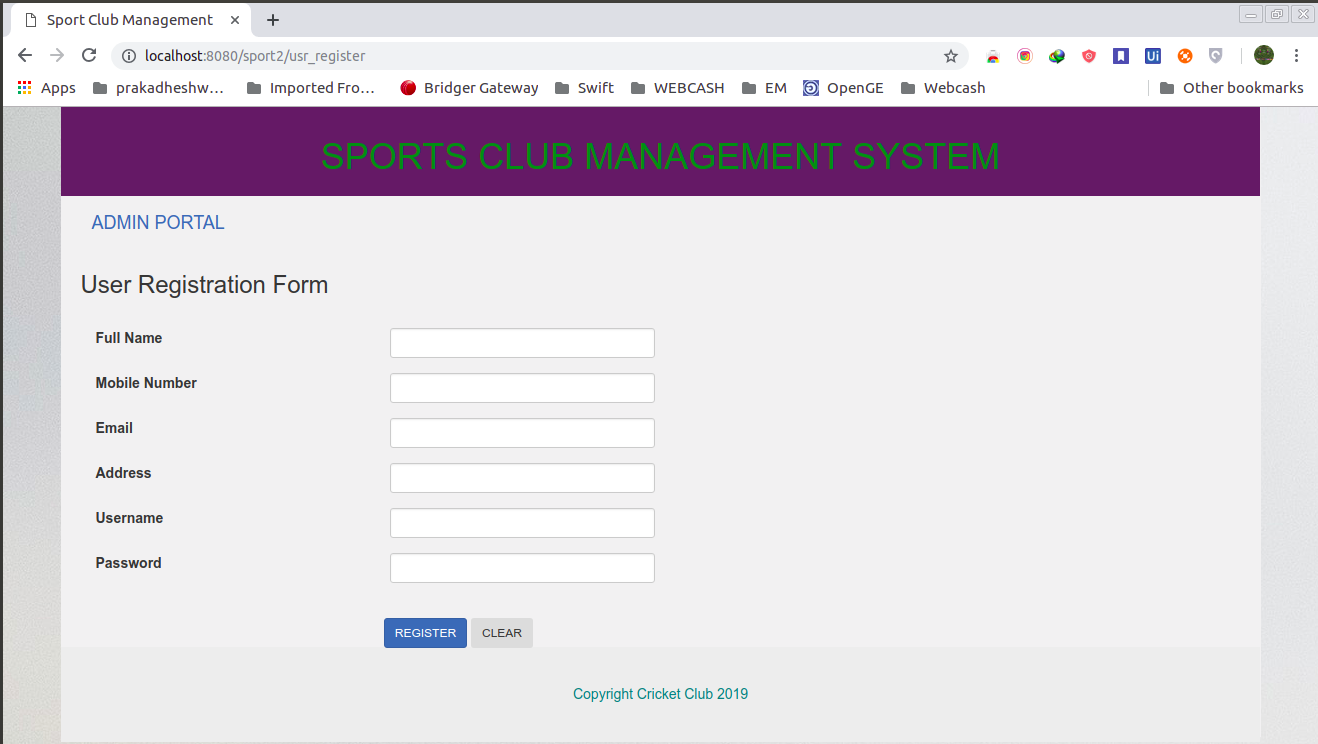
<%@include file="footer.jsp" %>

**6.2 SAMPLE DESIGN**

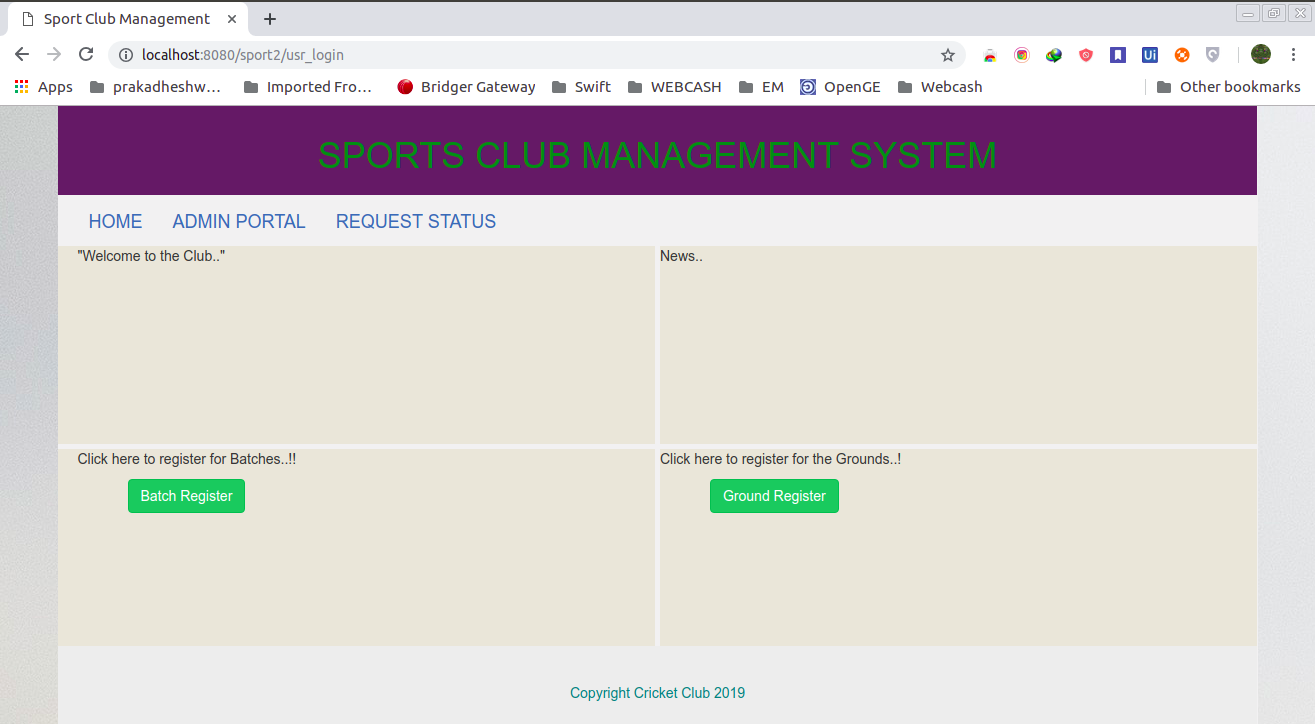
**LOGIN PAGE:**



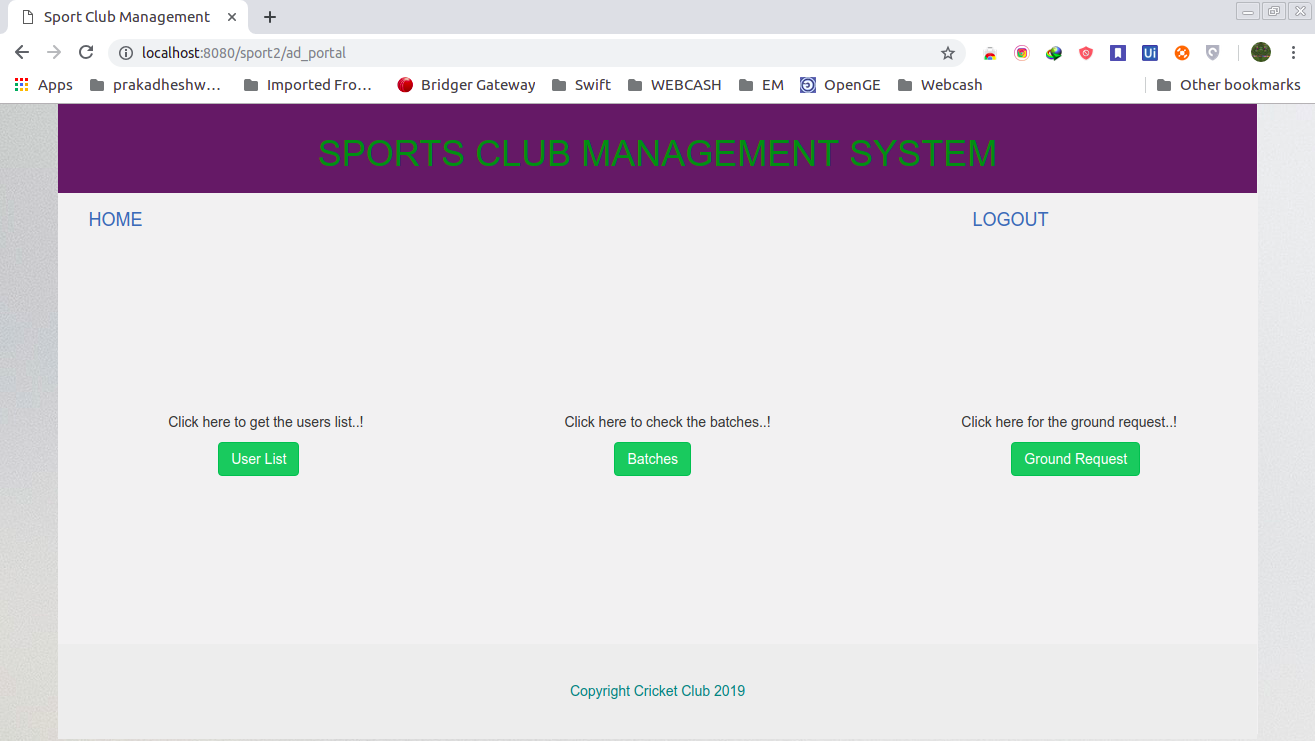
**REGISTRATION PAGE:**



**USER WELCOME:**



**ADMIN WELCOME:**



**CHAPTER 7**

**REFERENCES**

**Referred books:**

* Java the complete reference – seventh edition, Herbert-Schildt
* SQL the complete reference – Third edition, James R. Groff

**Referred Web sites:**

* www.w3schools.com
* www.tutorialspoint.com
* www.javatpoint.com