

PROGRESSIVE PROJECT REPORT
PLAYERS DATA MANAGEMENT

Submitted by

ARVIND (Reg. No. 07)
ANCHAL SHRIVASTAV (Reg. No. 06)
PARAS KUMAR (Reg. No. 05)

Under the Guidance of
National Skill Training Institute
Mumbai

ABSTRACT

Players data management is a project which aims to develop a computerized. It is an application that lets you add players to a database and also display their details from the database. you can also edit, delete and update players details. Besides the advantage of time saving.it also use data effectively.

Another plus point of data management is this data automatically saved in database.

ABSTRACT

1 INTRODUCTION

- 1.1 Project Aims and Objectives
- 1.2 Background of Project
- 1.3 Operation Environment

2 SYSTEM ANALYSIS

- 2.1 Software Requirement Specification
- 2.2 Software and Hardware Requirements
- 2.3 Software Used Tools

3 PROJECT SCHEDULE

- 3.1 Gantt Chart
- 3.2 Project Diary

4 DATA FLOW DIAGRAM

- 4.1 DFD
- 4.2 DFD Symbols

5 UML DIAGRAM

- 5.1 ER Diagram

6 WHOLE TABLE DESIGN(DATABASE)/BLOCK DIAGRAM

- 6.1 Block Diagram

7 SYSTEM IMPLEMENTATION

- 7.1 Screenshots

8 SYSTEM TESTING

- 8.1 Testing Methodologies
- 8.2 Levels of Testing
- 8.3 Testing Methodologies

9 CONCLUSION & FUTURE SCOPE

Conclusion/Code/Reference

INTRODUCTION

Data Players Management are using this project we can save our team data and information. So we can use this project.

1.1 Project Aims and Objectives

The Project aims and objectives that will be achieved after completion of this project are discussed in this sub chapter. The aims and objectives are as follows.

- User-friendly System
- Add data, edit and update
- Responsive design
- Offering several types of options

1.2 Background of Project

Players data management is an application which lets you add players data in efficient manner and automatically save data in database. The main objectives of this application to save time and cost. all the function of this players data management is easy to use and hand able by the users.

1.3 Operation Environment

Processor:

Intel core processor with better performance

Operating system:

Windows 7, windows 10, Ubuntu

Memory:

1GB Ram or more

Hard disk space:

Minimum 3GB for Database usage for future

Database:

MYSQL

CHAPTER 2

SYSTEM ANALYSIS

2.1 Software Requirement Specification

Product Description:

As Players data management is computerized system which help user to give Access to put data any time from anywhere as it is a web based application system.It reduces the risk of paper work such as file lost, file damaged and time consuming.

Problem Statement:

The problem occurred before having computerized system includes:

1. Lot of Paper works
2. Difficult to search record
3. Space consuming
4. Time consuming
5. Cost consuming

Problem Solution:

After implementing computerized system:

1. Easy to use
2. Save time
3. No more paper work
4. Reliable and accurate

2.2 Software and Hardware Requirements

This section describes the software and hardware requirements of the system

1. Software Requirements

- Operating system- windows 7/windows 10
- Database -My SQL
- Development tools- notepad/notepad ++ or Sublime text.
- Programming language- html , css, and java script
- Xampp as databases

2. Hardware Requirements

- Operating system of 32/64 bits
- Intel core i3/i5/i7 processors
- 1GB Ram

2.3 Software tools Used

The whole projects are divided in two parts

1. Front end
2. Back end

1. Front end:

The front end is designed using of html, css, and java script

➤ HTML

HTML stands for HYPER TEXT MARKUP LANGUAGE, which is most widely used language on web to develop web pages. HTML refers to the way in which Web pages (HTML documents) are linked together. Thus, the link available on a web page is called Hypertext.

HTML was created by Berners -Lee in late 1991 but “HTML 2.0” was the first standard HTML specification which was published in 1995. HTML 4.01 was a major version of HTML and it was published in late 1999. Though HTML4.01

version is widely used but currently we are having HTML-5 version which is an extension to HTML 4.01, and this version was published in 2012.

As its name suggests, HTML is a Mark-up Language which means you use HTML to simply “mark-up” a text document with tags that tells a web browser how to structure it to display.

Originally, HTML was develop with the intent of defining the structure of documents like heading, paragraph, lists, and so forth to facilitate the sharing of scientific information between researchers. Now, HTML is being widely used to format web pages with the help of different tags available in HTML.

➤ **CSS:**

Cascading Style Sheet is a style sheet language used for describing the presentation of a document written in a markup language Although most often used to set the visual style of web page and user interfaces written in HTML and XHTML, the language can be applied to any XML document, including plain XML, SVG and XUL, and is applicable to rendering in speech, or on other media. Along with HTML and JavaScript, CSS is a cornerstone technology used by most websites to create visually engaging webpages, user interfaces for web applications, and user interfaces for many mobile applications.

CSS is designed primarily to enable the separation of document content from document presentation, including aspects such as the layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple HTML pages to share formatting by specifying the relevant CSS in a separate .CSS file, and reduce complexity and repetition in the structural content.

The CSS specifications are maintained by the World Wide Web Consortium (W3C). Internet media type (MIME type) text/CSS is registered for use with CSS by RFC 2318 (March 1998). The W3C operates a free CSS validation service for CSS documents.

CSS has a simple syntax and uses a number of English keywords to specify the names of various style properties. A style sheet consists of a list of rules. Each rule or rule-set consists of one or more selectors, and a declaration block.

➤ **Java script:**

JavaScript is a lightweight, interpreted programming language. It is designed for creating network-centric applications. It is complimentary to and integrated with Java. JavaScript is very easy to implement because it is integrated with HTML. It is open and cross-platform.

JavaScript is a high-level, dynamic, untapped, and interpreted programming language. It has been standardized in the ECMA Script language specification. Alongside HTML and CSS, it is one of the three core technologies of World Wide Web content production; the majority of websites employ it and it is supported by all modern Web browsers without plug-ins. JavaScript is prototype-based with first-class functions, making it a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles. It has an API for working with text, arrays, dates and regular expressions, but does not include any I/O, such as networking, storage, or graphics facilities, relying for these upon the host environment in which it is embedded.

Despite some naming, syntactic, and standard library similarities, JavaScript and Java are otherwise unrelated and have very different semantics. The syntax of JavaScript is actually derived from C, while the semantics and design are influenced by the self and Scheme programming languages.

JavaScript is also used in environments that are not Web-based, such as PDF documents, site-specific browsers, and desktop widgets. Newer and faster JavaScript virtual machines (VMs) and platforms built upon them have also increased the popularity of JavaScript for server-side Web applications. On the client side, JavaScript has been traditionally implemented as an interpreted language, but more browsers that are recent perform just-in-time compilation. It is

also used in game development, the creation of desktop and mobile applications, and server-side network programming with runtime environments such as Node.js.

1. Back end:

The back end is designed using MySQL which is used to design the. Databases MySQL is an open source RDBMS that relies on SQL for processing the data in database. MySQL provides APIs for the languages like C, C++, Eiffel, JAVA, Perl, PHP and Python. MySQL is most commonly used for web applications and for embedded applications and has become a popular alternative to proprietary database system because of its speed and reliability. MySQL can run on UNIX, Windows and Mac OS

MySQL is an essential part of almost every open source PHP application. Good examples for PHP/MySQL-based scripts are PHP, OS. One of the most important things about using MySQL is to have a MySQL specialized host.

MySQL is the most popular Open Source Relational SQL database management system. MySQL is one of the best RDBMS being used for developing web based software applications.

MySQL is an open source relational database management system (RDBMS) based on Structured Query Language (SQL). MySQL runs on virtually all platforms, including Linux, UNIX, and Windows. Although it can be used in a wide range of applications, MySQL is most often associated with web-based applications and online publishing and is an important component of an enterprise stack called LAMP. LAMP is a Web development platform that uses Linux as the operating system, Apache as the Web server, MySQL as the relational database management system and PHP as the object oriented scripting language. (Sometimes Perl or Python is used instead of PHP.)

CHAPTER 3

PROJECT SEHEDULE

3.1.1 Gantt chart

A Gantt chart is a type of bar chart, adapted by Karol Adamiecki in 1896 and independently by Henry Gantt in the 1910s, that illustrates a project schedule. Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project. Terminal elements and summary elements comprise the work breakdown structure of the project. Modern Gantt charts also show the dependency (i.e., precedence network) relationships between activities. Gantt charts can be used to show current schedule status using percent-complete shadings and a vertical "TODAY" line as shown here.

Although now regarded as a common charting technique, Gantt charts were considered revolutionary when first introduced. This chart is also used in information technology to represent data that has been collected.

A Gantt chart is a horizontal bar chart developed as a production control tool in 1917 by Henry L. Gantt, an American engineer and social scientist. Frequently used in project management, a Gantt chart provides a graphical illustration of a schedule that helps to plan, coordinate, and track specific tasks in a project.

A Gantt chart, commonly used in project management, is one of the most popular and useful ways of showing activities (tasks or events) displayed against time. On the left of the chart is a list of the activities and along the top is a suitable time scale. Each activity is represented by a bar; the position and length of the bar reflects the start date, duration and end date of the activity. This allows you to see at a glance:

- What the various activities are.
- When each activity begins and ends.
- How long each activity is scheduled to last.

- Where activities overlap with other activities, and by how much.
- The start and end date of the whole projects

3.1.2 Project Diary

Duration	Tasks Accomplished
12 th September to 18 th September	Synopsis
19 th September to 28 th September	Introduction of project
28 th September to 8 th October	Project requirement
8 th October to 17 th October	System Requirement Specification
18 th October to 26 th October	Back End table Design
26 th October to 2 nd November	Front End Home Page
2 nd November to 8 th November	Front End Sign Up Page
9 th November to 15 th November	Front End Login Page
16 th November to 22 nd November	Front End customer Page
22 th November to 28 th November	Front End profilePage
28 th November to 20 th December	Front booksPage
21 st December to 31 st December	Layout
01 st January to 15 th January	Testing
16 th January to 24 th January	First Running Module Of Project
25 th January to 31 st January	Activity diagram, DFD, E-R Diagram

CHAPTER 4

DATA FLOW DIAGRAM

4.1 DFD:

A data flow diagram is graphical tool used to describe and analyze movement of data through a system. These are the central tool and the basis from which the other components are developed.

The transformation of data from input to output, through processed, may be described logically and independently of physical components associated with the system.

These are known as the logical data flow diagrams. The physical data flow diagrams show the actual implements and movement of data between people, departments and workstations.

A full description of a system actually consists of a set of data flow diagrams. Using two familiar notations Yourdon, Gane and Sarson notation develops the data flow diagrams. Each component in a DFD is labeled with a descriptive name. Process is further identified with a number that will be used for identification purpose.

The development of DFD'S is done in several levels. Each process in lower level diagrams can be broken down into a more detailed DFD in the next level. The top-level diagram is often called context diagram.

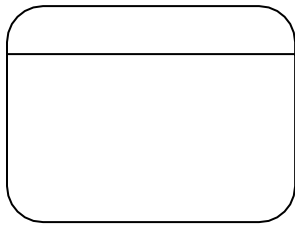
It consist single process bit, which plays vital role in studying the current system. The process in the context level diagram is exploded into other process at the first level DFD.

The idea behind the explosion of a process into more process is that understanding at one level of detail is exploded into greater detail at the next level. This is done until further explosion is necessary and an adequate amount of detail is described for analyst to understand the process.

4.2.1 DFD SYMBOLS:

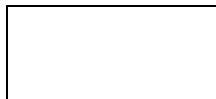
In the DFD, there are four symbols:

- A square defines a source(originator) or destination of system data
- An arrow identifies data flow. It is the pipeline through which the information flows
- A circle or a bubble represents a process that transforms incoming data flow into outgoing data flows.
- An open rectangle is a data store, data at rest or a temporary repository of data



Process that transforms data flow.

Fig. 4.2 Process that Transforms data Flow



Source or Destination of data

Fig. 4.3 Source or Destination of Data

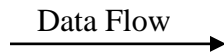


Fig. 4.4 Data Flow



Fig. 4.5 Data Store

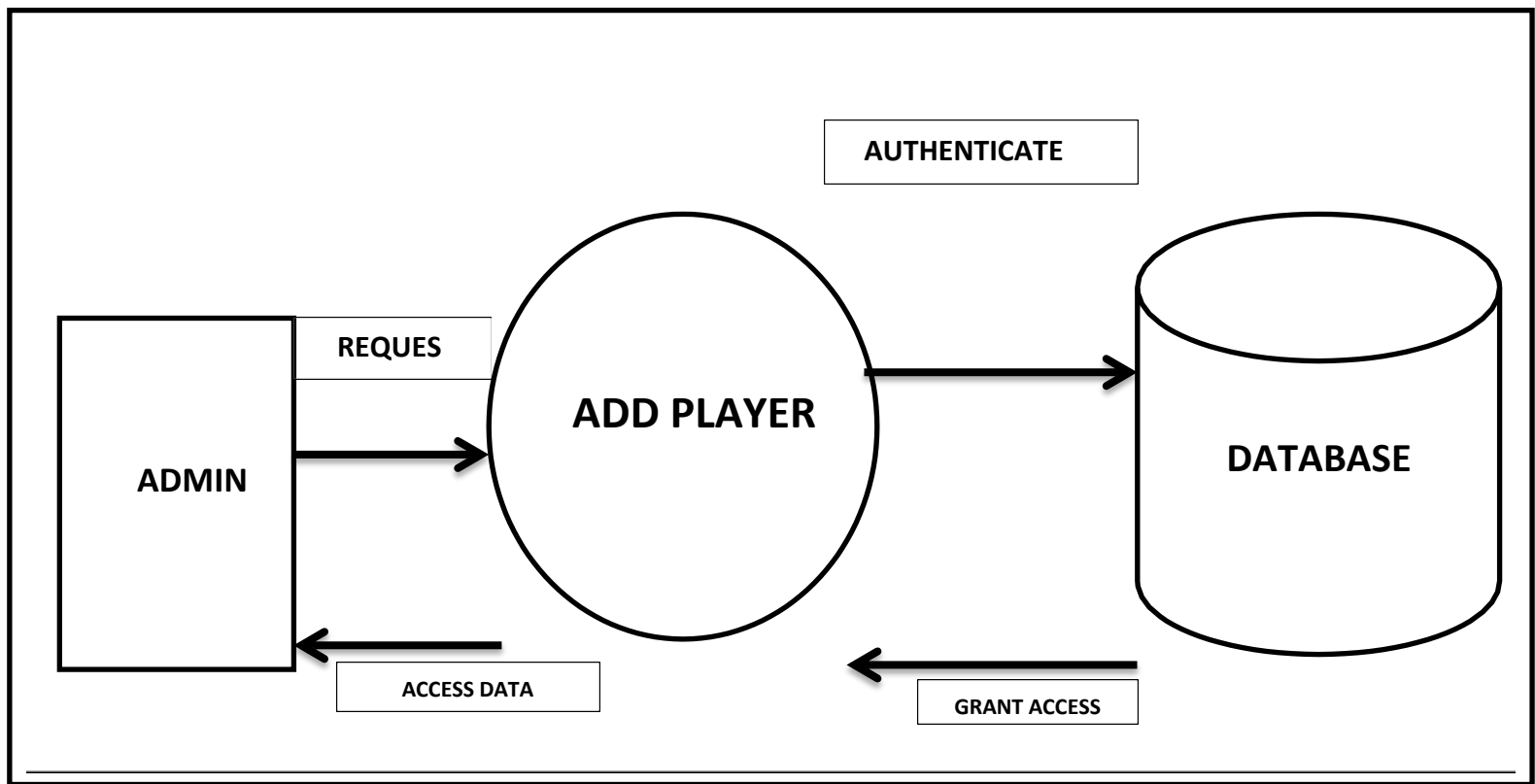


Fig. 4.6 DFD

CHAPTER 5

UML Diagrams

5.1 ER Diagram

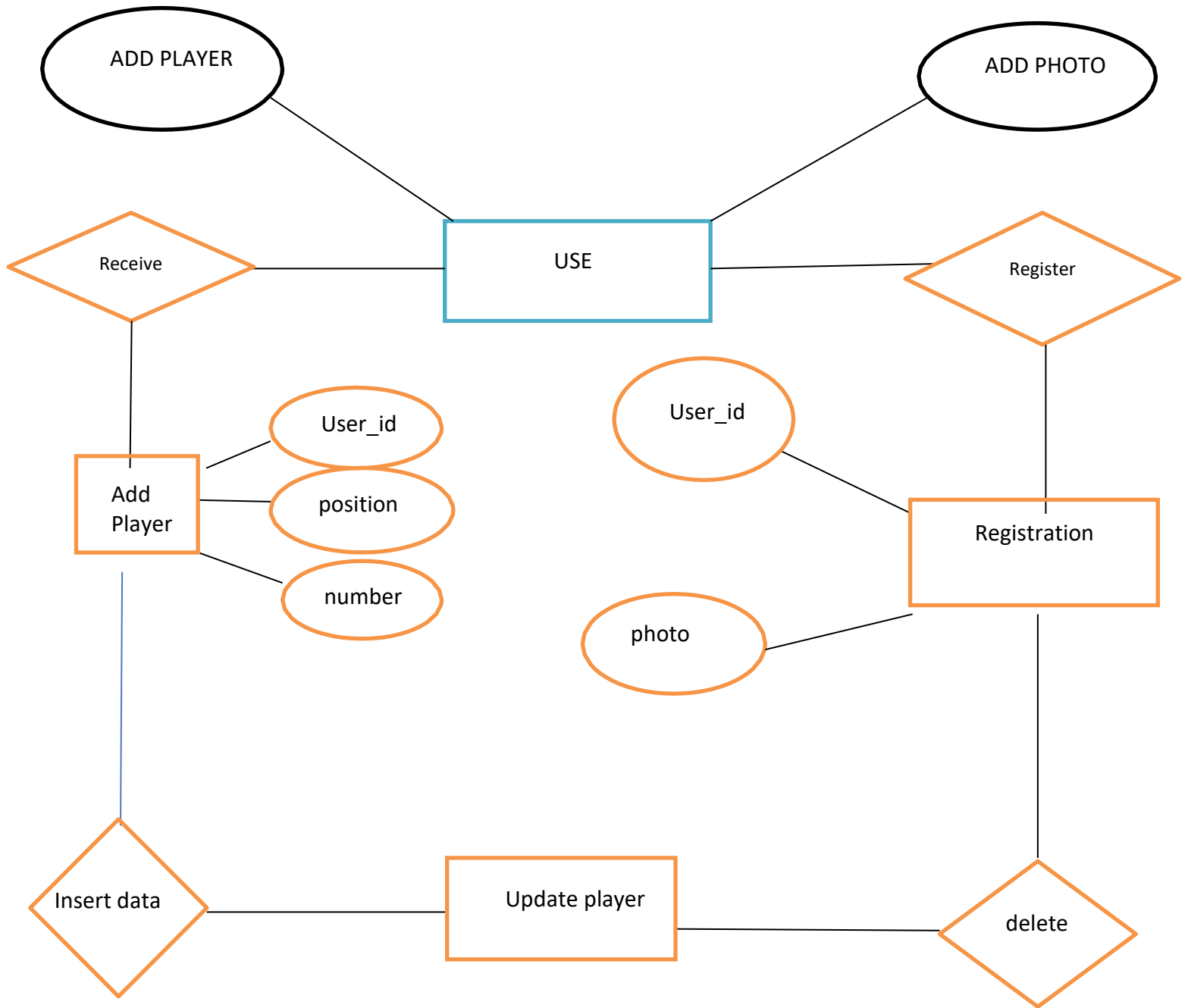
The Entity-Relationship Data Model (ERD) perceives the real world as consisting of basic objects, called entity & relationship among these objects.

It was developed to facilitate database design by allowing specification of an enterprise schema, which represents overall logical structure of a database. The ERD model is very useful in mapping the meaning & interactions of the outside world enterprises onto a conceptual schema.

The ERD model consists of the following major components

- ELLIPSE which represents attributes.
- RECTANGLES which represents entity-sets.
- DIAMONDS which represents the relationship sets.
- LINES which link attributes to entity sets to relationship sets.

E-R DIAGRAM



CHAPTER-6

SYSTEM DESIGN

1. Whole table design (DATABASE)/Block Diagram

The screenshot shows the phpMyAdmin interface for a database named 'socka'. The table 'players' is selected, and its structure is displayed. The table has 7 columns: 'id' (int(5), primary key, auto-increment), 'first_name' (varchar(255)), 'last_name' (varchar(255)), 'position' (varchar(255)), 'number' (int(11)), 'image' (varchar(255)), and 'user_name' (varchar(20)). All columns are using the 'latin1_swedish_ci' collation and have no default values. The 'id' column is the primary key.

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int(5)			No	None		AUTO_INCREMENT	Change Drop More
2	first_name	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
3	last_name	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
4	position	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
5	number	int(11)			No	None			Change Drop More
6	image	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
7	user_name	varchar(20)	latin1_swedish_ci		No	None			Change Drop More

Indexes

Action	Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
Edit Drop	PRIMARY	BTREE	Yes	No	id	0	A	No	

Create an index on 1 columns Go

Partitions

Console

6.1 Block diagram:

└─ players data management(main folder)

└─ node_modules

└─ public

└─ assets

└─ img

└─ routes

└─ index.js

└─ player.js

└─ views

└─ partials

└─ header.ejs

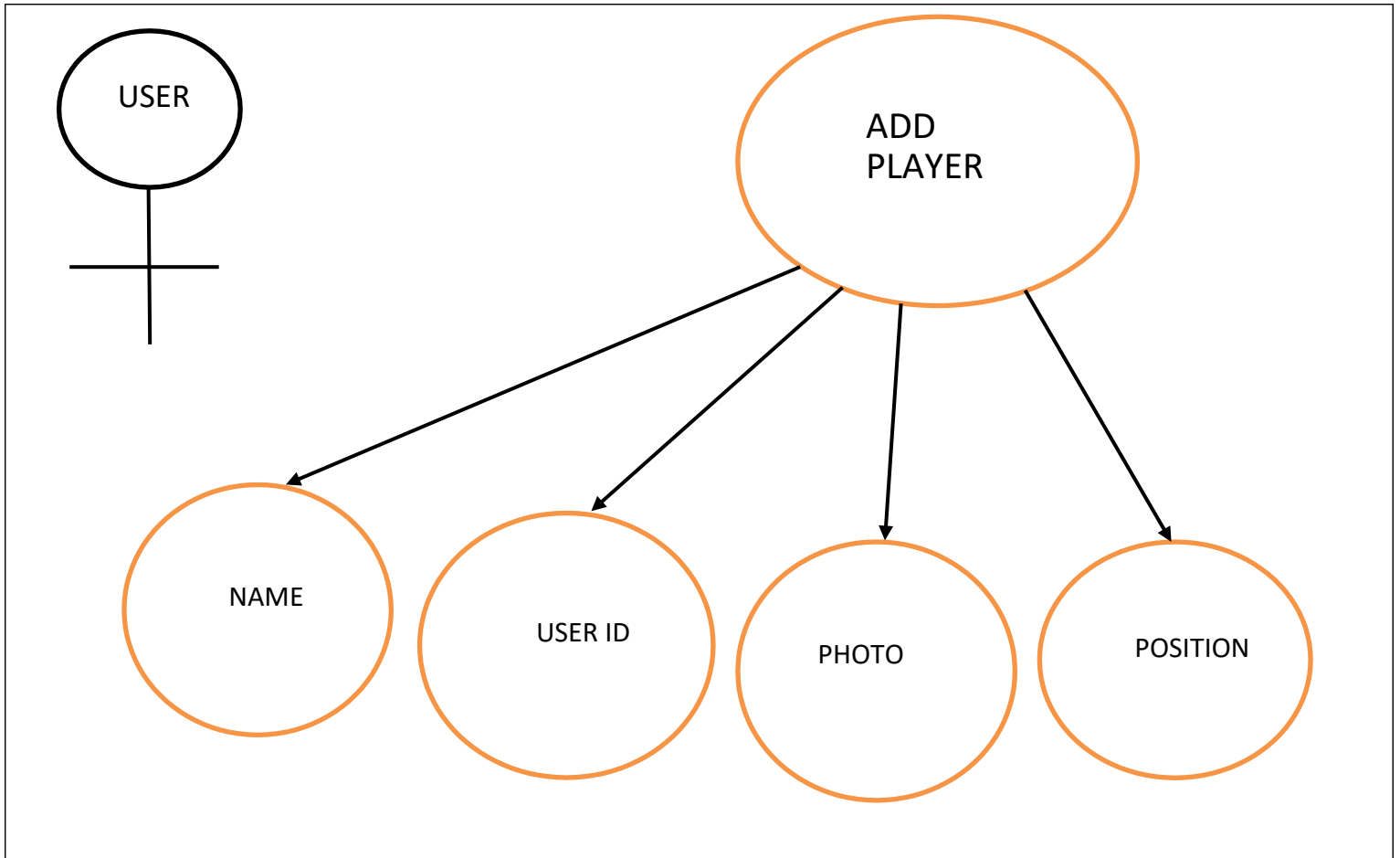
└─ index.ejs

└─ add-player.ejs

└─ edit-player.ejs

└─ app.js

User modules:



CHAPTER -7

SYSTEM IMPLEMENTATION

7.1 Screenshots

1. Add Player Page




No players found. Go [here](#) to add players.

2. Insert Data page:

A form titled 'socka games' with a light blue header bar containing the text 'socka games' on the left and 'Add a player' on the right. The form contains several input fields: a text field with 'Priya', a text field with 'hina', a text field with '@Priya', a text field with '1', and a dropdown menu with 'Defender'. Below these fields is a section labeled 'Trainee Image' with a 'Choose File' button and the filename 'goal.jpg'. At the bottom right of the form is a blue button labeled 'Add Player'.

3. Edit Page

[socka games](#)[Add a player](#)

ID	Image	First Name	Last Name	Position	Number	Username	Action
3		Priya	hina	Defender	1	@@Priya	<div>EditDelete</div>

4. Update data:

[socka games](#)[Add a player](#)

First Name

Priya

Last Name

hina

Username

@@Priya

Number

1

Position


Centre Back

Update Player

5 .Delete data

socka games

[Add a player](#)

ID	Image	First Name	Last Name	Position	Number	Username	Action
4		Riya	shrivastav	Defender	1	@anchalshrivastavmbd@	Edit Delete

localhost:3000/delete/4

CHAPTER-8

SYSTEM TESTING

8.1. TESTING METHODOLOGIES

1. Black box Testing
2. White box Testing

8.2. LEVELS OF TESTING

1. Unit Testing
2. Integration Testing

1. TESTING METHODOLOGIES

1. Black box Testing:

It is the testing process in which tester can perform testing on an application without having any internal structural knowledge of application. Usually Test Engineers are involved in the black box testing.

2. White box Testing:

It is the testing process in which tester can perform testing on an application with having internal structural knowledge. Usually The Developers are involved in white box testing.

8.2 LEVELS OF TESTING

1. Unit Testing:

Unit Testing concentrates on the verification of the smallest element of the program i.e. Module. In this testing all control paths are tested to identify errors within the bounds of the module. The important goal of unit testing is to isolate each part of the program and

show individual parts are correct. It is very easy to perform and requires less amount of time because the modules are smaller in size.

In unit testing it is possible that the outputs produced by one unit become input for another unit hence, if incorrect output produced by one unit is provided as input to the second unit then it also produces wrong output. If this process is not corrected, the entire software may produce unexpected outputs. To avoid this, all the units in the software are tested independently using unit –testing. In unit testing, the units are tested to ensure that they operate correctly. In software engineering the unit testing is not just performed once during software development, but repeated whenever the software is modified.

2. Integration Testing:

When unit testing is complete integration testing begins. In integration testing the tested units are combined together to form system as whole. The aim of this testing is to ensure that all modules are working properly according to user's requirements when they are combined. The integration test takes all tested individual modules, integrate them, test them again and develop the software. It ensures that all modules work together properly and transfer accurate data across their interfaces. Integration testing contains: - Non – Incremental integration: The entire program is tested as a whole and all errors are identified. Incremental integration: The program is constructed and tested in small segments, to find out errors.

CHAPTER-9

CONCLUSION & FUTURE SCOPE

Conclusion:

This application provides a computerized version of players data managements which will benefit the user to put data easily and changes into data without having any problem.

It makes entire process online where user can just input detail of player and it automatically saved in database.

Future scope:

The main aim of our project is create a good interaction between the admin and user. We are trying to do the project at best level to satisfy all the end users.

CHAPTER-10

CODE

All programming Coding

Database code:

```
CREATE DATABASE socka;

CREATE TABLE IF NOT EXISTS `players` (

`id` int(5) NOT NULL AUTO_INCREMENT,

`first_name` varchar(255) NOT NULL,

`last_name` varchar(255) NOT NULL,

`position` varchar(255) NOT NULL,

`number` int(11) NOT NULL,

`image` varchar(255) NOT NULL,

`user_name` varchar(20) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1;
```

App.js

```
const express = require('express');

const fileUpload = require('express-fileupload');

const bodyParser = require('body-parser');

const mysql = require('mysql');
```

```
const path = require('path');

const app = express();


// const {getHomePage} = require('./routes/index');

// const {addPlayerPage, addPlayer, deletePlayer, editPlayer, editPlayerPage} =
require('./routes/player');

const port = 5000;


// create connection to database

// the mysql.createConnection function takes in a configuration object which contains
host, user, password and the database name.

const db = mysql.createConnection ({

host: 'localhost',

user: 'root',

password: "",

database: 'socka'

});

db.connect((err) => {

if (err) {

throw err;

}

console.log('Connected to database');
```

```
});

global.db = db;


// configure middleware

app.set('port', process.env.port || port); // set express to use this port

app.set('views', __dirname + '/views'); // set express to look in this folder to render our view

app.set('view engine', 'ejs'); // configure template engine

app.use(bodyParser.urlencoded({ extended: false }));

app.use(bodyParser.json()); // parse form data client

app.use(express.static(path.join(__dirname, 'public'))); // configure express to use public folder

app.use(fileUpload()); // configure fileupload


// routes for the app


app.listen(port, () => {

  console.log(`Server running on port: ${port}`);

});
```

Index.js

```
module.exports = {

  getHomePage: (req, res) => {
```

```
    let query = "SELECT * FROM `players` ORDER BY id ASC"; // query database to get all  
the players
```

```
    // execute query  
  
    db.query(query, (err, result) => {  
  
        if (err) {  
  
            res.redirect('/');  
  
        }  
  
        res.render('index.ejs', {  
  
            title: Welcome to Socka | View Players  
  
            ,players: result  
  
        });  
  
    });  
  
},  
  
};
```

Player.js

```
const fs = require('fs');  
  
module.exports = {  
  
    addPlayerPage: (req, res) => {
```

```
res.render('add-player.ejs', {  
  
  title: "Welcome to socka| Add a new Player"  
  
  ,message: "  
  
});  
  
},  
  
addPlayer: (req, res) => {  
  
  if (!req.files) {  
  
    return res.status(400).send("No files were uploaded.");  
  
  }  
  
  
  
  
  let message = "  
  
  let first_name = req.body.first_name;  
  
  let last_name = req.body.last_name;  
  
  let position = req.body.position;  
  
  let number = req.body.number;  
  
  let username = req.body.username;  
  
  let uploadedFile = req.files.image;  
  
  let image_name = uploadedFile.name;  
  
  let fileExtension = uploadedFile.mimetype.split('/')[1];  
  
  image_name = username + '.' + fileExtension;
```



```
let usernameQuery = "SELECT * FROM `players` WHERE user_name = '" + username +
""";

db.query(usernameQuery, (err, result) => {

  if (err) {

    return res.status(500).send(err);

  }

  if (result.length > 0) {

    message = 'Username already exists';

    res.render('add-player.ejs', {

      message,

      title: "Welcome to Socka | Add a new player"

    });

  } else {

    if (uploadedFile.mimetype === 'image/png' || uploadedFile.mimetype === 'image/jpeg'
|| uploadedFile.mimetype === 'image/gif') {

      uploadedFile.mv(`public/assets/img/${image_name}`, (err) => {

        if (err) {

          return res.status(500).send(err);

        }

      });

    }

  }

});
```

```

        let query = "INSERT INTO `players` (first_name, last_name, position, number,
image, user_name) VALUES (" +

            first_name + ", " + last_name + ", " + position + ", " + number + ", " +
image_name + ", " + username + ")";

        db.query(query, (err, result) => {

            if (err) {

                return res.status(500).send(err);

            }

            res.redirect('/');

        });

    });

} else {

    message = "Invalid File format. Only 'gif', 'jpeg' and 'png' images are allowed.";

    res.render('add-player.ejs', {

        message,

        title: "Welcome to Socka | Add a new player"

    });

}

});

},

```

```
editPlayerPage: (req, res) => {  
  
  let playerId = req.params.id;  
  
  let query = "SELECT * FROM `players` WHERE id = '" + playerId + "'";  
  
  db.query(query, (err, result) => {  
  
    if (err) {  
  
      return res.status(500).send(err);  
  
    }  
  
    res.render('edit-player.ejs', {  
  
      title: "Edit Player"  
  
      ,player: result[0]  
  
      ,message: "  
  
    });  
  
  });  
  
},
```

```
editPlayer: (req, res) => {  
  
  let playerId = req.params.id;  
  
  let first_name = req.body.first_name;  
  
  let last_name = req.body.last_name;  
  
  let position = req.body.position;  
  
  let number = req.body.number;
```

```
    let query = "UPDATE `players` SET `first_name` = '" + first_name + "', `last_name` = '" + last_name + "', `position` = '" + position + "', `number` = '" + number + "' WHERE `players`.`id` = '" + playerId + "'";
```

```
    db.query(query, (err, result) => {
```

```
        if (err) {
```

```
            return res.status(500).send(err);
```

```
        }
```

```
        res.redirect('/');
```

```
    });
```

```
},
```

```
deletePlayer: (req, res) => {
```

```
    let playerId = req.params.id;
```

```
    let getImageQuery = 'SELECT image from `players` WHERE id = '" + playerId + "'";
```

```
    let deleteUserQuery = 'DELETE FROM players WHERE id = '" + playerId + "'";
```

```
    db.query(getImageQuery, (err, result) => {
```

```
        if (err) {
```

```
            return res.status(500).send(err);
```

```
        }
```

```
        let image = result[0].image;
```

```
fs.unlink(`public/assets/img/${image}`, (err) => {

  if (err) {

    return res.status(500).send(err);

  }

  db.query(deleteUserQuery, (err, result) => {

    if (err) {

      return res.status(500).send(err);

    }

    res.redirect('/');

  });

});

});

}
```

Header.ejs

<!doctype html>

<html lang="en">

<head>

<meta charset="UTF-8">

```
<meta name="viewport"
      content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0,
minimum-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">

<!--<link rel="stylesheet" href="/assets/css/custom.css">-->

<title><%= title %></title>

</head>

<style>

.table-wrapper {

    margin-top: 50px;

}

.player-img {

    width: 40px;

    height: 40px;

}

.add-player-form {
```

```

        margin-top: 50px;

    }

</style>

<body>

<div class="page-wrapper">

    <nav class="navbar navbar-light bg-light">

        <span class="navbar-brand mb-0 h1"><a href="/">socka games</a></span>

        <a class="float-right" href="/add" title="Add a New students">Add a students</a>

    </nav>

```

Add-player.ejs

```

<% include partials/header.ejs %>

<div class="container">

    <% if (message != "") { %>

        <p class="text-center text-danger"><%= message %></p>

    <% } %>

    <form class="add-player-form" action="" method="post" enctype="multipart/form-data">

        <div class="form-row">

            <div class="form-group col-md-4">

                <input type="text" class="form-control" name="first_name" id="first-name"
placeholder="First Name" required>

```

</div>

<div class="form-group col-md-4">

<input type="text" class="form-control" name="last_name" id="last-name"
placeholder="Last Name" required>

</div>

<div class="form-group col-md-4">

<input type="text" class="form-control" name="username" id="username"
placeholder="Username" required>

</div>

</div>

<div class="form-row">

<div class="form-group col-md-6">

<input type="number" class="form-control" name="number" id="number"
placeholder="Number" required>

</div>

<div class="form-group col-md-6">

<select id="position" name="position" class="form-control" required>

<option selected disabled>Choose Trades</option>

<option>Turner</option>

<option>Fitter</option>

<option>MMV</option>

<option>DMM</option>


```
<option>CSA</option>

<option>Electrician</option>

<option>Electronics</option>

<option>Welder</option>

<option>Machinist</option>

<option>Diploma</option>

</select>

</div>

<div class="col-md-12">

  <label for="player-img"><b>Trainee Image</b></label><br>

  <input type="file" name="image" id="player-img" class="" required>

</div>

</div>

<button type="submit" class="btn btn-primary float-right">Add Students</button>

</form>

</div>

</div>

</body>

</html>
```

Edit-player.ejs

```
<% include partials/header.ejs %>
```

```
<div class="container">
```

```
<% if (message) { %>
```

```
<p class="text-center text-danger"><%= message %></p>
```

```
<% } %>
```

```
<% if (player) { %>
```

```
<form class="add-player-form" action="" method="post" enctype="multipart/form-data">
```

```
<div class="form-row">
```

```
<div class="form-group col-md-4">
```

```
<label for="first-name">First Name</label>
```

```
<input type="text" class="form-control" name="first_name" id="first-name"
value="<%= player.first_name %>" required>
```

```
</div>
```

```
<div class="form-group col-md-4">
```

```
<label for="last-name">Last Name</label>
```

```
<input type="text" class="form-control" name="last_name" id="last-name"
value="<%= player.last_name %>" required>
```

```
</div>
```

```
<div class="form-group col-md-4">
```

```
<label for="username">Username</label>
```

```
<input type="text" class="form-control" name="username" id="username"
value="<%= player.user_name %>" required disabled title="Username can't be changed">
```

```
</div>
```

```
</div>
```

```
<div class="form-row">
```

```
<div class="form-group col-md-6">
```

```
<label for="number">Number</label>
```

```
<input type="number" class="form-control" name="number" id="number"
placeholder="Number" value="<%= player.number %>" required>
```

```
</div>
```

```
<div class="form-group col-md-6">
```

```
<label for="position">Position</label>
```

```
<select id="position" name="position" class="form-control" required>
```

```
<option selected><%= player.position %></option>
```

```
<option>Goalkeeper</option>
```

```
<option>Centre Back</option>
```

```
<option>Right Back</option>
```

```
<option>Left Back</option>
```

```
<option>Defensive Midfielder</option>
```

```
<option>Central Midfielder</option>
```

```
<option>Attacking Midfielder</option>
```

```
<option>Right Wing Forward</option>
```

```

        <option>Left Wing Forward</option>

        <option>Striker</option>

    </select>

</div>

</div>

<button type="submit" class="btn btn-success float-right">Update Player</button>

</form>

<% } else { %>

    <p class="text-center">Player Not Found. Go <a href="/add">here</a> to add players.</p>

<% } %>

</div>

</div>

</body>

</html>

```

Index.ejs

```

<% include partials/header.ejs %>

<div class="table-wrapper">

    <% if (players.length > 0) { %>

        <table class="table table-hovered">

```

```

<thead class="thead-dark">

  <tr>

    <th scope="col">ID</th>

    <th scope="col">Image</th>

    <th scope="col">First Name</th>

    <th scope="col">Last Name</th>

    <th scope="col">Position</th>

    <th scope="col">Number</th>

    <th scope="col">Username</th>

    <th scope="col">Action</th>

  </tr>

</thead>

<tbody>

  <% players.forEach((player, index) => { %>

    <tr>

      <th scope="row"><%= player.id %></th>

      <td></td>

      <td><%= player.first_name %></td>

      <td><%= player.last_name %></td>

      <td><%= player.position %></td>

```

<td><%= player.number %></td>

<td>@<%= player.user_name %></td>

<td>

<a href="/edit/<%= player.id %>" target="_blank" rel="noopener"
class="btn btn-sm btn-success">Edit

<a href="/delete/<%= player.id %>" class="btn btn-sm btn-
danger">Delete

</td>

</tr>

<% }) %>

</tbody>

</table>

<% } else { %>

<p class="text-center">No players found. Go here to add
players.</p>

<% } %>

</div>

</div>

</body>

</html>

References

I. HTML, CSS, JavaScript

- www.w3school.com

II. HTML, CSS, JavaScript

- www.tutorialpoints.com

III. Apache Sever

- <https://www.apachefriends.org/download.html>