

INDIRA GANDHI NATIONAL OPEN UNIVERSITY

BCSP-064

Project Report on

LEARN SPHERE (LEARNING MANAGEMENT SYSTEM)

By

Name: Suraj Kumar Goswami

Enrolment No: 2200172438

Under Guidance

of

Mr. Deeptanshu Kumar

BACHELOR OF COMPUTER APPLICATION(BCA)

2024



Indira Gandhi National Open University

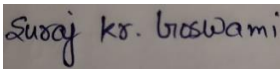
Maidan Garhi

New Delhi – 110068.

CERTIFICATE OF ORIGINALITY

This is to certify that the project report entitled LEARN SPHERE (LEARNING MANAGEMENT SYSTEM) Submitted to Indira Gandhi National Open University in partial fulfillment of the requirement for the award of the degree of BACHELOR OF COMPUTER APPLICATIONS (BCA) , is an original work carried out by Mr./ Ms. SURAJ KUMAR GOSWAMI Enrolment No.: 2200172438 under the guidance of Mr./ Ms. Deeptanshu Kumar

The matter embodied in this project is a genuine work done by the student and has not been submitted whether to this University or to any other University/Institute for the fulfilment of the requirement of any course of study.



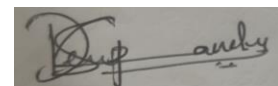
Signature of the Student

Date : 15-11-2024

Name and Address
of the student :

Suraj Kumar Goswami, Jai Guru ITI,
Baramuri,Dhanbad

Enrolment No.: 2200172438



Signature of the Guide

Date : 15-11-2024

Name, Designation and
Address of the Guide:

Deeptanshu Kumar(Kodnest
Trainer), BTM Layout,Bangalore

CURRICULUM VITAE

Deeptanshu Kumar

2nd Stage BTM Layout

Bangalore 560076

Karnataka

E-mail: deeptanshu473@gmail.com

Mobile No : +91 92348 32150

CAREER OBJECTIVE :

To be an efficient learner and to contribute in the field of technology to the best of my ability and to provide a direction to my skills and knowledge for the advancement of the company.

EDUCATIONAL QUALIFICATIONS :

- B.TECH in 2017 from Aryabhatta Knowledge University.
- Intermediate in 2013 from Patna University.
- Matriculation in 2011 from Trinity Global Best CBSE School.

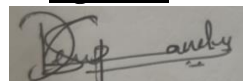
TECHNICAL SKILLS :

- Languages: Html, CSS, JavaScript, Java, Hibernate, Spring, Spring Boot, Java Servlet, SQL and Manual Testing.
- Concepts Known: Data Structure , Operating System and DBMS.
- Operating System: Microsoft Windows / Linux.
- Web Designing: HTML, CSS and Java Script.
- Database: ORACLE 10g and MYSQL.

WORK EXPERIENCE:

- Currently Working as Lead Technical Trainer in Kodnest in Bangalore.
- Worked as a Technical Trainer in ABC for Technology Training in Bangalore.

Signature



CONTENTS

| | |
|--|----------------|
| 1. INTRODUCTION | 5 |
| 1.1 PROJECT OVERVIEW | 5 |
| 1.2 PROJECT DESCRIPTION | 6-11 |
| 2. 2. SYSTEM CONFIGURATION | 12 |
| 2.1 HARDWARE SPECIFICATION | 12 |
| 2.2 SOFTWARE SPECIFICATION | 12 |
| 3. SYSTEM ANALYSIS... | 13 |
| 3.1 PRELIMINARY INVESTIGATION | 13 |
| 3.2 EXISTING SYSTEM | 13-14 |
| 3.3 PROPOSED SYSTEM | 14 |
| 3.4 FEASIBILITY ANALYSIS... | 14-15 |
| 3.5 ADVANTAGES OF PROPOSED SYSTEM | 15 |
| 3.6 REQUIREMENT SPECIFICATION | 15-38 |
| 4. SYSTEM DESIGN | 39 |
| 4.1 INTRODUCTION | 39 |
| 4.2 DATA FLOW DIAGRAM | 40-47 |
| 4.3 INPUT DESIGN | 47-48 |
| 4.4 OUTPUT DESIGN | 48 |
| 4.5 E-R DIAGRAM... | 48-53 |
| 4.6 DATABASE DESIGN | 53-55 |
| 4.7 CODING | 56-108 |
| 5. SYSTEM TESTING | 109 |
| 5.1 TESTING METHODS | 109 |
| 5.2 TEST PLAN ACTIVITIES | 109-113 |
| 5.3 TESTING OBJECTIVE... | 113 |
| 5.4 SCREEN LAYOUTS | 114-120 |
| 6. SYSTEM IMPLEMENTATION | 120-121 |
| 7. CONCLUSION AND SCOPE FOR FUTURE ENHANCEMENT... | 121-122 |
| 8. BIBLIOGRAPHY... | 122 |

1. INTRODUCTION:

1.1 PROJECT OVERVIEW

The project entitled “Learn Sphere” a Learning Management System has to be developed using front-end tool HTML5, CSS3, JavaScript, and back-end tool JAVA and its framework Spring Boot. It also involves databases like MYSQL. It is a window-based application. The main aim of the project is to create a computerized system maintaining various works of LMS.

This system was designed with a singular focus — to enrich the educational journey for both educators and learners. Our platform boasts an intuitive interface that transforms the management of courses, access to educational materials, and progress tracking into a fluid and user-friendly experience. At Learning Management System, we recognize the pivotal role technology plays in education. With this understanding, we've crafted a platform that empowers educators to navigate the intricacies of course management effortlessly. Creating, organizing, and delivering content has never been more intuitive, allowing educators to focus on what they do best – teaching.

For learners, the Learning Management System opens the door to a digital realm where educational materials are just a click away. Our user-friendly interface ensures seamless access to a wealth of resources, from interactive content to in-depth learning materials, fostering an environment where knowledge acquisition is not only effective but enjoyable. What sets the Learning Management System apart is our unwavering commitment to progress. We understand that monitoring and tracking progress are essential elements of the learning journey. With the Learning Management System, educators and learners alike have at their disposal robust tools for tracking achievements, assessing performance, and navigating the path to success. Embark on a journey of educational excellence with LMS— where online learning transcends the ordinary, and every click brings you closer to a world of knowledge. Welcome to a future of seamless online learning experiences.

1.2 PROJECT DESCRIPTION

This project will serve the following objectives:-

The main objective behind this project is to provide a user-friendly environment to provide knowledge and give everyone a chance to learn, irrespective of where they are, provided they register themselves with the system.

The main features that the system provides can be made use of, once the registered people select their interested Courses. This helps to establish an incremental learning process. The project on the Learning Management System is to manage the details of Students, Trainers and Courses.

Benefits to the user:-

- Learn anytime, anywhere with internet access, offering unparalleled flexibility.
- Customize your learning experience by progressing at your own speed.
- Access all learning materials in one place for easy navigation.
- Choose learning paths aligned with your goals and interests.
- Acquire new skills online for professional growth and career advancement.
- Save on commuting and accommodation costs with online learning.
- Ensure consistent and standardized content delivery for all learners.

The different modules are:

REGISTER

It shows the details about the Username, Email, Password and Role.

In this first, the interested Students and trainers get registered by providing the necessary details. Each person will and can register only one time. Details of each person along with their username and password are saved permanently in the database.

The user can register themselves as

- Trainer
- Student

LOGIN

After providing the correct username and password, the user log's in to the Learning system homepage. There the user can select the available modules as per their registry.

➤ If Login as Trainer:

- **ADD COURSE**

Trainers can add the on-demand course with Course ID, Course Name, and Course Price.

- **ADD LESSON**

Trainers can add multiple lessons inside each course with details like Course ID, Lesson ID, Lesson Name, Lesson Topics, and Lesson Link.

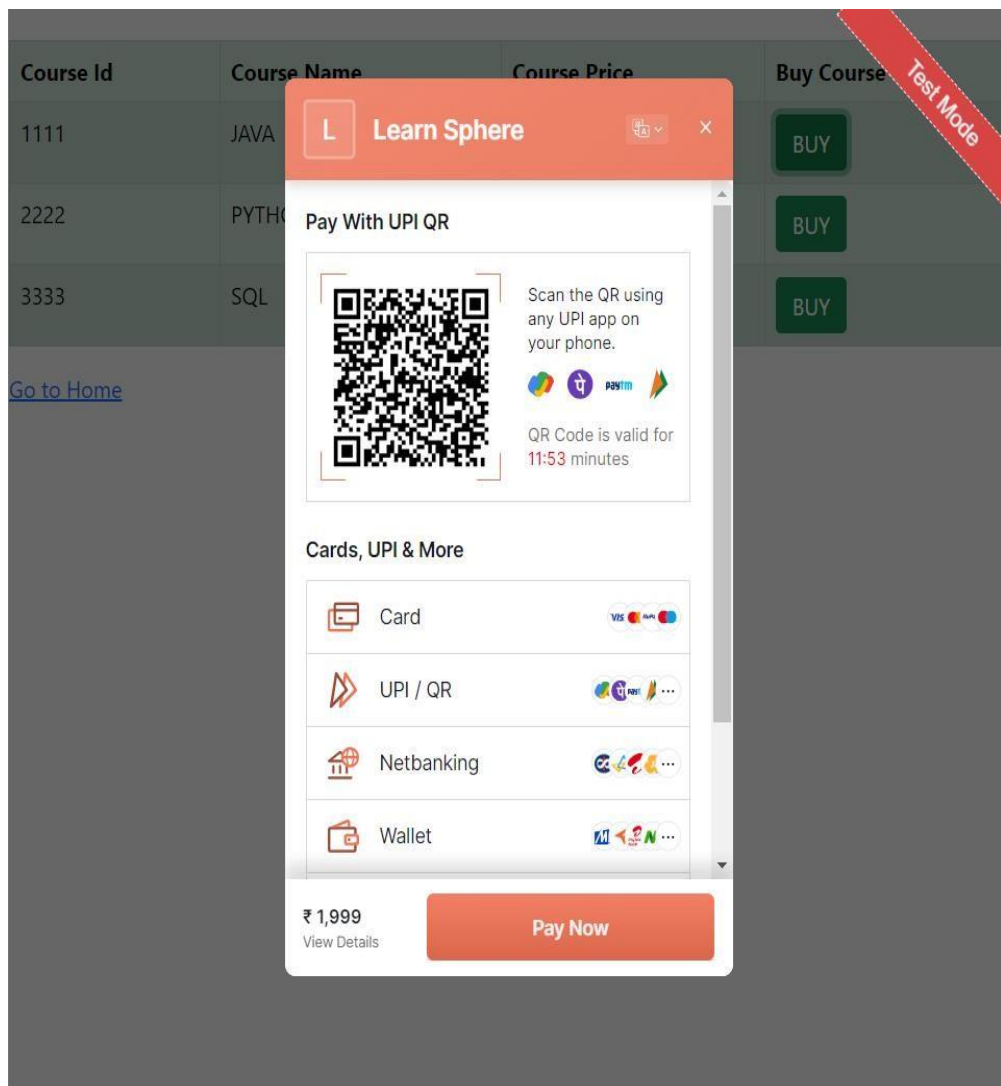
- **VIEW COURSE**

The Trainer can view the courses added in the form table with column headings Course ID, Course Name, Course Price, and Lessons.

➤ If Login as a Student:

- ◆ **PURCHASE COURSE**

The Students on logging in can have the choice of purchasing a course. The course added by the Trainer can be seen in this module with the Course ID, Course Name, Course Price, and Buy Course in the form of a table.



◆ MY COURSE

In this module, the student can view the courses bought by them after making the successful payment.

The details include Course ID, Course Name, Course Price, and Lessons.

Learning Management System has the following modules:-

Register

| Function | Requirement | Description | User |
|----------------------|---|--|-------|
| Create Admin profile | <ul style="list-style-type: none">• User Name• Email• Password• Role | First-time user (admin) should create his/her profile. | Admin |

Log in

| Function | Requirement | Description | User |
|----------|--|---|-----------------|
| Login | <ul style="list-style-type: none">• Email• Password | Student/Trainer can Log In to the system. | Student/Trainer |

ADD COURSE

| Function | Requirements | Description | User |
|-------------------|---|--|----------------|
| Add Course | <ul style="list-style-type: none">• Course ID• Course Name• Course Price | Trainers can add courses to the system. | Trainer |

ADD LESSON

| Function | Requirements | Description | User |
|-------------------|--|---|----------------|
| Add Lesson | <ul style="list-style-type: none">• Course ID• Lesson ID• Lesson Name• Lesson Topics• Lesson Link | Trainers can add multiple lessons to the System. | Trainer |

VIEW COURSE

| Function | Requirements | Description | Users |
|--------------------|--|--|----------------|
| View Course | <ul style="list-style-type: none">• Course ID• Course Name• Course Price | Trainer can view the created courses by them in the System. | Trainer |

PURCHASE COURSE

| Function | Requirements | Description | Users |
|------------------------|---|--|----------------|
| Purchase Course | <ul style="list-style-type: none">• Course ID• Course Name• Course Price• Buy Course | Students can Log In to the System and purchase their desired courses. | Student |

MY COURSE

| Function | Requirements | Description | Users |
|------------------|---|---|----------------|
| My Course | <ul style="list-style-type: none">• Course ID• Course Name• Course Price• Lesson | Students can view the courses purchased. | Student |

2. SYSTEM CONFIGURATION

2.1 HARDWARE SPECIFICATION

- Processor : Core 2 Duo Processor
- RAM : 2 GB
- Hard disk : 20 GB
- Monitor : 14 inch
- Mouse : 3 Button scroll
- CD Drive : 52 X
- Keyboard : 108 keys

2.2 SOFTWARE SPECIFICATION

- Operating System : Windows 10, OS and Linux
- Languages : JAVA and SQL
- Front End : HTML, CSS and JavaScript
- Tools : Spring Boot
- Backend : MYSQL

3 SYSTEM ANALYSIS

3.1 PRELIMINARY INVESTIGATION

Preliminary investigation is a problem-solving activity that requires intensive communication between the system users and system developers. It does various feasibility studies. In these studies, a rough figure of the system activities can be obtained, from which the decisions about the strategies to be followed for effective system study and analysis can be taken.

During the preliminary investigation, an initial picture of the system working is obtained from the information obtained from this study, and the data collection methods were identified. Right from the investigation of the system, many existing drawbacks of the system could be identified, which helped a lot in the later stages of more rigorous study and analysis of the manual system.

The most critical phase of managing system projects is planning. To launch a system investigation, we need a master plan detailing the steps to be taken, the people to be questioned, and the outcome expected.

3.2 EXISTING SYSTEM

At present all the administration in our state municipalities office works are done manually. Only hard copies of the documents will be available. So data cannot be retrieved if lost once. The data may be missed and searching of any data will be very slow. The updates of data are inefficient. Hurdles were encountered during the initial implementation

- Non-availability of useful/vital data
- Poorly teaching quality at remote areas
- Business logic not clearly defined
- Limited resources(power problems, space, staff) at remote locations.

DRAWBACKS OF EXISTING SYSTEM:

- **More manpower.**
- **Time consuming.**
- **Consumer large volume of paperwork.**
- **Needs manual calculations.**
- **Non-availability of useful/vital data**
- **Poorly maintained physical records**
- **Books of accounts not up-to-date**
- **Business logic not clearly defined**
- **Limited resources (power problems, space, staff) at remote locations.**

To avoid all these limitations and make the work more accurate the system needs to be

computerized.

3.3 PROPOSED SYSTEM

The proposed system aims to develop a system of improved facilities. The proposed system can overcome all the limitations of the existing system. The system provides proper security and reduces manual work. The Learning Management System is automated information updates deployed and secure. The system is mainly concerned with performing learning operations. The Users can register themselves as Students or Trainers. The Trainer can create a course, create a lesson, and add a lesson name and lesson link where Students can purchase the course and view the purchased courses.

The key points of the proposed system are

- **Centralized Learning Platform:** An LMS serves as a centralized platform for managing, delivering, and tracking learning activities and content.
- **Course Management:** It allows instructors to create, organize, and deliver courses.
- **User Management:** Administrators can manage user roles, permissions, and access levels within the system, including students and instructors.
- **Content Management:** LMS platforms provide tools for creating courses, lessons, and multimedia content such as videos.

EXPECTED ADVANTAGES OF PROPOSED SYSTEM:

The system is very simple to design and implement. The system requires very low system resources and the system will work in almost all configurations.

3.4 FEASIBILITY ANALYSIS

The feasibility study is a test of the system proposal according to its workability, impact on the organization, ability to meet the needs, and effective use of resources. During the study, the problem definition is crystallized, and common problems to be included in this system are determined. The result of the feasibility study is a formal proposal. If the proposal is accepted, we continue with the project.

FEASIBILITY CONSTRAINTS

→ ECONOMIC FEASIBILITY:

Economic analysis is the most frequently used method for evaluating the effectiveness of a candidate system. More commonly known as cost/benefit analysis, the procedure is to

determine the benefits and savings that are expected from a candidate system and compare them with costs. The proposed system is an economically feasible one. We do not want to keep a lot of books for storing the data. Manipulating data using a computer reduces cost. We do not want a lot of employees; we simply want one to operate it, Administrator.

— TECHNICAL FEASIBILITY:

Technical feasibility centers on the existing computer system and to what extent it can support the proposed system. It involves financial considerations to accommodate technical enhancements. If the budget is a serious constraint, then the project is judged not feasible. Here we need only a computer working at low speed to accomplish the task.

— BEHAVIOUR FEASIBILITY:

People inherently resist change and computers have been known to facilitate change. An estimate should be made of how strong a reaction the user staff is likely to have toward the development computerized system. The computer installations have something to do with turnover, retraining, and changes to employee status. In the proposed system, it behaves very feasibly. It is very easy to train the people in the proposed system. We simply want to tell the purpose of each button and about a little data to enter.

3.5 ADVANTAGES OF PROPOSED SYSTEM

- Users will receive better and quicker service.**
- Security is ensured by protecting the system with passwords.**
- Normalized database tables eliminate data redundancy.**
- Provision for minimizing errors in data entry.**
- Efficient data storage.**
- Real-time response and user-friendliness.**
- Time saving.**

3.6 REQUIREMENT SPECIFICATION

Introduction to Java, Swing , NetBeans & Database :

⌘ JAVA

Java is a general-purpose object-oriented programming language developed by the Sun microsystem of the USA in 1991. Originally called Oak by James Gosling, one of the inventors of the language, java was designed for the development of software for consumer electronic devices like TVs, VCRs, Toasters, and other devices.

The Java team which included Patrick Naught discovered that the existing languages like c and C++ had limitations in terms of reliability, portability, and security. However, they modeled their new language java on C and C++ but removed several features of C and C++ that were considered the source of problems and thus made Java a simple, reliable, portable, and powerful language.

Features of Java:

- **Compiled and Interpreted**
- **Platform- independent and Portable**
- **Object Oriented**
- **Robust and Secure**
- **Distributed**
- **Familiar, Simple and Small**
- **Multithreaded**
- **High performance**
- **Dynamic and Extensible**

Compiled & Interpreted

A programming Language may be either a compiled or interpreted language but Java uses both approaches that means Java source code is first compiled using the Java compiler “javac” to generate “java byte code” or intermediate code. It is not a machine code but a code that can only be understood by JVM(Java Virtual Machine). This byte code is again interpreted into the machine code by using a JVM interpreter known as “java”. Examples of interpreted languages are: python, PHP, JavaScript, Perl, etc., and examples of compiled languages are: C, C++, COBOL, Fortran, etc.

Platform Independent & Portable

The most important feature of Java is its portability that is Java programs can be easily moved from one computer system to another anywhere, at any time. Whatever the changes in the operating system, processors, and system resources will not force any changes in the java program. We can download a Java applet from a remote computer onto our local system via the internet and execute it locally.

Object Oriented

Java is a purely object oriented language. Almost everything in java is an object. Java comes with an extensive set of classes, arranged in packages.

Robust & Secure

Java is a robust language. Java uses exception handling which captures serious errors

and eliminates any risk of crashing the system. Java is secure because Java doesn't use pointers which prevent unauthorized access to memory.

Distributed

Java is designed as a distributed language for creating application on networks. This enables multiple programmers at multiple remote locations to collaborate and work together on a single project.

Simple, Small & Familiar

Java is a small and simple language. For example, java doesn't use pointers, preprocessor header files, operator overloading, and multiple inheritance. It is familiar because it is modeled on C and C++ language therefore java looks like C++ codes.

Multithreaded

Multithreaded means handling multiple tasks simultaneously. Java supports multi-threaded programs. This means that we need not wait for the application to finish one task before beginning another. For example, we can listen to an audio clip while scrolling the page at the same time download an applet from a distant computer. High-performance Java performance is impressive due to intermediate byte code and support of multithreading. Dynamic and Extensible Java is a dynamic language. Java is capable of dynamically linking new class libraries, methods, and objects. Java supports functions written in other languages such as C and C++. These functions are also known as the native method.

Java versions

| | |
|------------------------------|----------------------------------|
| JDK 1.0 | - RELEASED IN 23-JAN-1996 |
| JDK 1.1 | - 1997 |
| J2SE 1.2 | -1998 |
| J2SE 1.3 | - 2000 |
| J2SE 1.4 | - 2002 |
| J2SE 1.5 OR J2SE 5.0 | - 2004 |
| J2SE 1.6 OR java SE 6 | - 2006 |
| JAVA SE 7 | - 2011 |
| JAVA SE 8 | - 2014 |

Basic Java Architecture

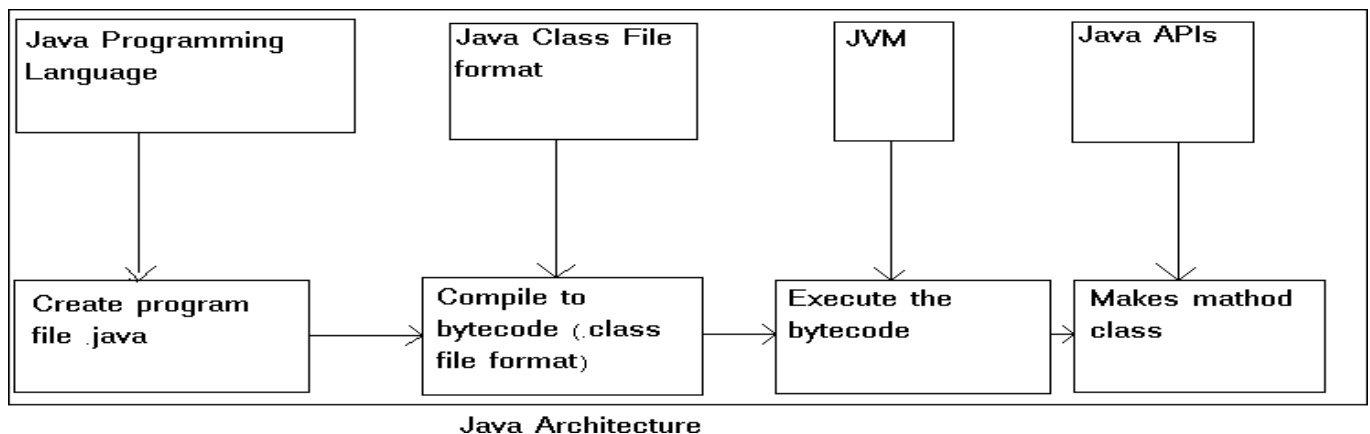
- 1. java programming language**
- 2. java class file format**

3. jvm

4. java APIs.

The JVM and Java APIs form a platform for executing Java programs. Together they are also called Java Runtime System or Java Runtime Environment (JRE). The Java compiler (javac) compiles the programmer's created source code to instructions understood by the JVM. The interpreter, which is the part of the JVM then converts these instructions to the machine code that is understood by the hardware for which the interpreter has been created. It is the JVM that converts the Java byte code into the machine language and then executes it. One of the tasks of the JVM is loading of .class file from the permanent storage media. The JVM has class loaders that load the .class files into memory and then execute the byte code contained within them. The .class files of the Java APIs are also loaded into the JVM. The JVM can control multiple class loaders at run time. Each Java application runs inside a JVM. The JVM starts when a Java program executes and terminates when the program ends. The JVM calls the main method to start an application.

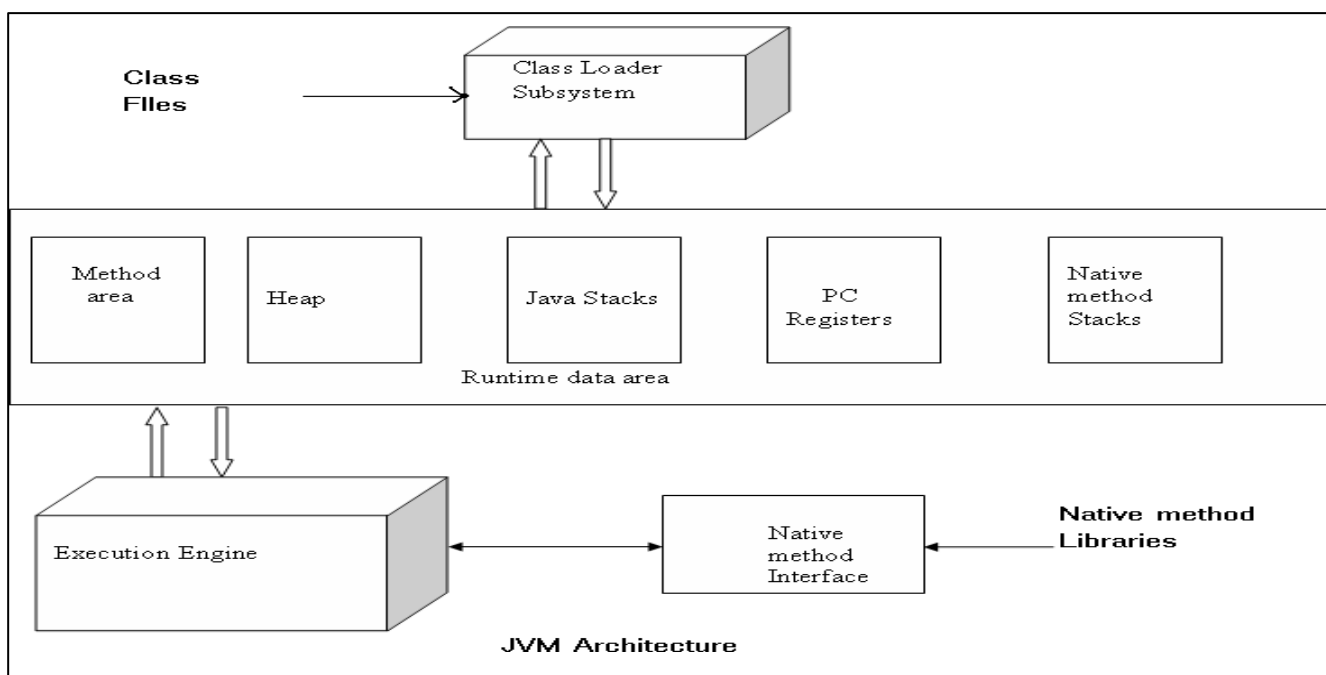
JVM Architecture



Class loader subsystem

The class loader is the subsystem of JVM, after compilation of the Java source code, the class file is generated in the current working directory before the JVM executes the Java byte codes, JVM implicitly calls the class loader subsystem to load the class files into JVM allocated memory space. At this time the class loader subsystem performs the following steps:

- It loads a class file.



- It checks the correctness of the class file, if anyone has manipulated the class file, then that class file cannot be executed.
- It allocates memory to the static variable, block, and methods.
- It sets default values to all static variables.

Method Area

It is a logical memory component of JVM. This logical section of memory holds information about classes and interfaces. The static variables, blocks, and methods allocate memory from the method area. The size of the method area is not fixed. It shrinks and expands according to the size of the application.

Heap Area

In Java when an object or array is created, memory is allocated to them from heap. The JVM through the use of a new operator allocates memory from the heap for an

object. The JVM has a daemon thread known as a garbage collector whose task is to free those objects from the heap whose reference is not alive in the stack.

Java stacks

At execution time when a method is called the memory is allocated from the stack for its local variable, arguments, and references.

Program counter

It keeps track of the sequence of execution of the program. PC register holds the addresses of the instructions to be executed next.

Native method stacks

When a Java application invokes a native method, then the native method stack is used to allocate memory to the local variables and arguments. The library required for the execution of native methods is available to the JVM through Java Native Interface (JNI).

Execution Engine

This component of the JVM is responsible for executing the java byte code. Java platform consists of a JVM and implementation of standard libraries (APIs).

Java Platform

There are three types of Java platform.

1. java platform, micro edition [java ME]
 2. java platform, standard edition [java SE]
 3. java platform, Enterprise edition [java EE]
-
1. **Java ME:** It is a collection of Java APIs for the development of software for devices like PDAs, cell phones, and consumer products such as Air conditioners, Fridges, and microwave ovens. Java ME has become popular for creating games for cell phones.
 2. **Java SE:** It is a collection of Java APIs useful for developing general-purpose software applications for desktop PCs, Servers, and similar devices.
 3. **Java EE:** Useful in developing a software component running one application server.

Java Environment

Java includes a large number of development tools and hundreds of classes and methods. The development tools are part of the system known as Java Development Kit (JDK) where classes and methods are part of the Java Standard Library (JSL), also known as Application Programming Interface (API).

Java Development Kit

It is a collection of Tools that are required at the time of Java Application Development. These Tools are

- **applet viewer:** This tool is required to run Java applets.

```
import java.awt.*;  
  
import java.applet.*;  
  
/*  
  
<applet code="test.class" height=100 width=100>  
  
</applet>
```
- **javah:** It produces header files for use with native methods.
- **javadoc:** Javadoc is a tool that comes with JDK and it is used for generating Java code documentation in HTML format from Java source code which has required documentation in a predefined format.

```
import java.io.*;  
  
/**  
  
* <h1>Add Two Numbers!</h1>  
  
* The AddNum program implements an application that  
  
* simply adds two given integer numbers and Prints  
  
* the output on the screen.  
  
*/  
  
public class test extends Applet  
{  
  
    public void paint(Graphics g)
```

```

{
    g.drawString("hello applet",100,100);
}
}

```

- **javac:** It is a Java compiler which translates java source code to byte code files so that the interpreter can understand.
- **java:** It is a java interpreter which runs java applications by reading and interpreting byte code files.
- **javap:** It is a Java dis-assembler, which enables us to convert byte code files into the program description.

Example:

Javap <classFilename>

Javap AddNum

* <p>

* Note: Giving proper comments in your program makes it more

* user friendly and it is assumed as a high quality code.

*

* @author Gunjan Jha

* @version 1.0

* @since 2015-03-31

*/

public class AddNum {

/**

* This method is used to add two integers. This is

* a the simplest form of a class method, just to

* show the usage of various javadoc Tags.

```

* @param numA This is the first paramter to addNum method
* @param numB This is the second parameter to addNum method
* @return int This returns sum of numA and numB.
*/

public int addNum(int numA, int numB) {
    return numA + numB;
}
/**
* This is the main method which makes use of addNum method.
* @param args Unused.
* @return Nothing.
* @exception IOException On input error.

* @see IOException
*/

public static void main(String args[]) throws IOException
{
    AddNum obj = new AddNum();
    int sum = obj.addNum(10, 20);

    System.out.println("Sum of 10 and 20 is : " + sum);
}
}

```

javadoc AddNum.java

- **jar** : create and maintain java archive files.
- **apt**: annotation processing tool.
- **extcheck**: utility to detect jar conflicts.
- **jdb**: It's a java debugger, which help us to find errors in our programs. it is a debugging tool that comes along with the sun's JDK. It is available in bin directory of java known as jdb.exe.

Let us consider the below program:

```
class testfirst  
{  
    public static void main(String arg[])  
    {  
        int $x$y=11,y=12;  
        float z;  
  
        z=$x$y+y;  
  
        System.out.println(z);  
    }  
}
```

In a command prompt type the following commands

`javac -g testfirst.java`

`jdb testfirst`

(initializing jdb...)

`stop in testfirst.main(java.lang.String[])` `//to give break point in this method`

`run` `//now JVM has been started.`

After this statements the prompt will be changed to

`Main[1]`

Main[1] list //to see the current line to be executed next

Main[1] next //to execute next line

Main[1] locals //it displays all the values of local variables for this above program must be compiled using javac -g testfirst.java

Main[1] print obj.val1 //it will display the object member value

Main[1] dump obj //it will display all members value of an object.

There are mainly four types of applications that can be created using Java.

Standalone application: It is also known as desktop application or window-based application. An application that needs to be installed on every machine such as a media player, antivirus, etc.

Web application: An application that runs on the web server and creates a dynamic page, called a web application. Currently, servlet, jsp, struts etc technologies are used for creating web applications in Java.

Enterprise application: An application that is distributed in nature, such as banking applications, etc. has the advantages of high-level security, load balancing, etc. In Java, EJB is used for creating these types of applications.

Mobile application

An application that is created for mobile devices. Currently, Android and Java ME are used for creating mobile applications.

Application programming interface (APIs)

The Java standard library(or API) includes hundreds of classes and methods grouped into several functional packages. The most commonly used packages are:

1. Language Support package (java. lang): Contains a no. of classes and methods that are used by the Java compiler itself. They include classes for primitive types, Strings, math functions, threads, and exceptions.
2. Utilities packages (java. util): A collection of classes to provide utility functions such as date and time functions.
3. Input/Output packages (java.io): A collection of classes required for input/output manipulation. Networking package (java.net): A collection of classes for communicating with the local computer and internet servers.

4. **AWT package (java.awt):** The abstract window tool kit package contains classes for implementing graphical user interface. They include classes for windows, buttons, list, menus and so on.
5. **Applet package (java.applet):** This includes a set of classes that allows us to create java applets.
6. **Sql package:** This includes a set of classes those are required for creating applications based on database connectivity.

❖ JAVA SWING

Swing did not exist in the early days of Java. Rather, it was a response to deficiencies present in Java's original GUI subsystem: the Abstract Window Toolkit. The AWT defines a basic set of controls, windows, and dialog boxes that support a usable, but limited graphical interface. One reason for the limited nature of the AWT is that it translates its various visual components into their corresponding, platform-specific equivalents. This means that the look and feel of a component is defined by the platform, not by Java. Because the AWT components use native code resources, they are referred to as heavyweight.

The use of native peers led to several problems. First, because of variations between operating systems, a component might look, or even act, differently on different platforms. Second, the look and feel of each component was fixed (because it is defined by the platform) and could not be (easily) changed. Third, the use of heavyweight components caused some frustrating restrictions. For example, a heavyweight component is always rectangular and opaque.

Not long after Java's original release, it became apparent that the limitations and restrictions present in the AWT were sufficiently serious that a better approach was needed. The solution was Swing. Introduced in 1997, Swing was included as part of the Java Foundation Classes (JFC). Swing was initially available for use with Java 1.1 as a separate library. However, beginning with Java 1.2, Swing (and the rest of the JFC) was fully integrated into Java.

Swing was created to address the limitations present in the AWT. It does this through two key features: lightweight components and a pluggable look and feel.

Together they provide an elegant, yet easy-to-use solution to the problems of the AWT.

Swing Features

- **Light Weight** - Swing component are independent of native Operating System's API as Swing API controls are rendered mostly using pure JAVA code instead of underlying operating system calls.
- **Rich controls** - Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, colorpicker, table controls
- **Highly Customizable** - Swing controls can be customized in very easy way as visual apperance is independent of internal representation.
- **Pluggable look-and-feel**- SWING based GUI Application look and feel can be changed at run time based on available values.

Java Swing is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java. Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

| No. | Java AWT | Java Swing |
|-----|--|---|
| 1) | AWT components are platform-dependent. | Java swing components are platform-independent. |
| 2) | AWT components are heavyweight. | Swing components are lightweight. |

| | | |
|----|---|---|
| 3) | AWT doesn't support pluggable look and feel. | Swing supports pluggable look and feel. |
| 4) | AWT provides less components than Swing. | Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc. |
| 5) | AWT doesn't follows MVC (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view. | Swing follows MVC (Model View Controller). |

Components & Containers

A Swing GUI consists of two key items: components and containers. However, this distinction is mostly conceptual because all containers are also components. The difference between the two is found in their intended purpose: As the term is commonly used, a component is an independent visual control, such as a push button or slider. A container holds a group of components. Thus, a container is a special type of component that is designed to hold other components.

Furthermore, in order for a component to be displayed, it must be held within a container. Thus, all Swing GUIs will have at least one container. Because containers are components, a container can also hold other containers. This enables Swing to define what is called a containment hierarchy, at the top of which must be a top-level container.

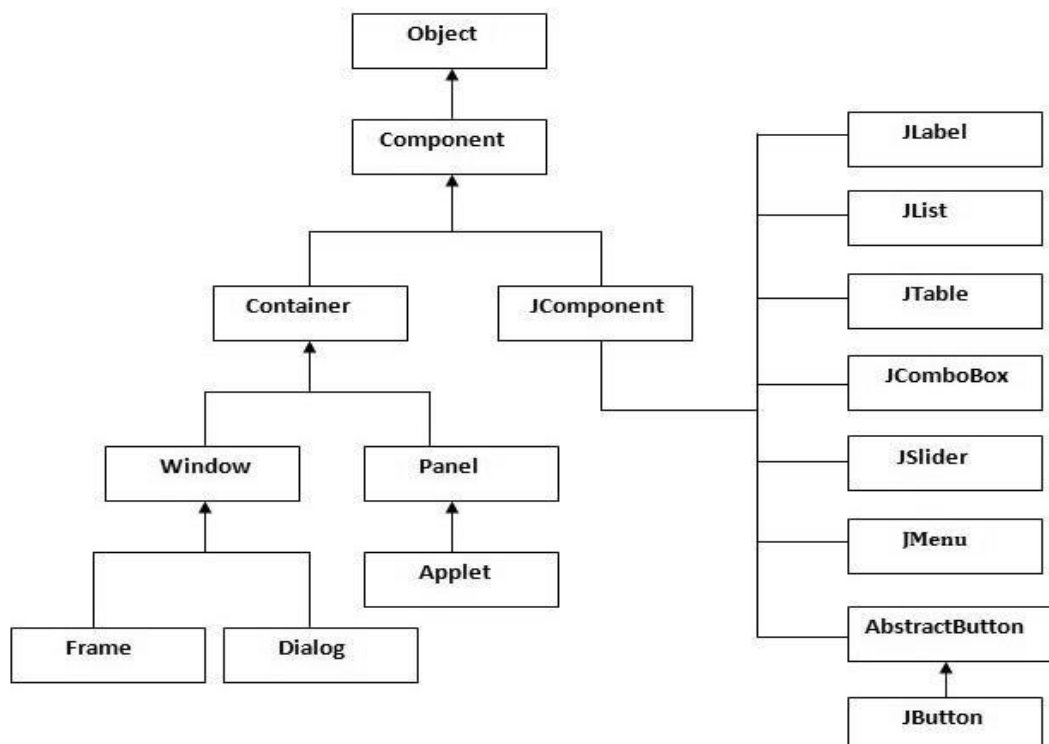
In general, Swing components are derived from the JComponent class. JComponent provides the functionality that is common to all components. For example, JComponent supports the pluggable look and feel. JComponent inherits the AWT classes Container and Component. All of Swing's components are represented by classes defined within the package javax.swing.

Examples of components and containers.

| | | | |
|---------------|----------------|----------------------|---------------------|
| JApplet | JButton | JCheckBox | JCheckBoxMenuItem |
| JColorChooser | JComboBox | JComponent | JDesktopPane |
| JDialog | JEditorPane | JFileChooser | JFormattedTextField |
| JFrame | JInternalFrame | JLabel | JLayeredPane |
| JList | JMenu | JMenuBar | JMenuItem |
| JOptionPane | JPanel | JPasswordField | JPopupMenu |
| JProgressBar | JRadioButton | JRadioButtonMenuItem | JRootPane |
| JScrollBar | JScrollPane | JSeparator | JSlider |
| JSpinner | JSplitPane | JTabbedPane | JTable |
| JTextArea | TextField | JTextPane | JToggleButton |
| JToolBar | JToolTip | JTree | JViewport |
| JWindow | | | |

Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



Introduction to Oracle

Database

The Oracle database has a logical layer and a physical layer. The physical layer consists of the files that reside on the disk; the components of the logical layer map the data to these physical components.

The Physical Layer

The physical layer of the database consists of three types of files:

One or more data files: Data files store the information contained in the database. You can have as few as one data file or as many as hundreds of data files. The information for a single table can span many data files or many tables can share a set of data files. Spreading table spaces over many data files can have a significant positive effect on performance. The number of data files that can be configured is limited by the Oracle parameter MAXDATAFILES.

Two or more redo log files: Redo log files hold information used for recovery in the event of a system failure. Redo log files, known as the redo log, store a log of all changes made to the database. This information is used in the event of a system failure to reapply changes that have been made and committed but that might not have been made to the data files. The redo log files must perform well and be protected against hardware failures (through software or hardware fault tolerance). If redo log information is lost, you cannot recover the system.

One or more control files: Control files contain information used to start an instance, such as the location of data files and redo log files; Oracle needs this.

Cluster: A cluster is a set of tables physically stored together as one table that shares a common column. If data in two or more tables is frequently retrieved together based on data in the common column, using a clustered table can be quite efficient. Tables can be accessed separately even though they are part of a clustered table. Because of the structure of the cluster, related data requires much less I/O overhead if accessed simultaneously.

Index: An index is a structure created to help retrieve data more quickly and efficiently (just as the index in this book allows you to find a particular section more quickly). An index is declared on a column or set of columns. Access to the table based on the value of the indexed column(s) (as in a WHERE clause) will use the index to locate the table data.

View: A view is a window into one or more tables. A view does not store any data; it presents table data. A view can be queried, updated, and deleted as a table

without restriction. Views are typically used to simplify the user's perception of data access by providing limited information from one table, or a set of information from several tables transparently. Views can also be used to prevent some data from being accessed by the user or to create a join from multiple tables.

information to start the database instance. Control files must be protected. Oracle provides a mechanism for storing multiple copies of control files.

The Logical Layer

The logical layer of the database consists of the following elements:

One or more table spaces: The database schema, which consists of items such as tables, clusters, indexes, views, stored procedures, database triggers, sequences, and so on.

Table spaces and Data files

New Term: The database is divided into one or more logical pieces known as table spaces. A table space is used to logically group data together.

Table: A table, which consists of a table name and rows and columns of data, is the basic logical storage unit in the Oracle database. Columns are defined by name and data type. A table is stored within a table space; often, many tables share a table space.

Stored procedure: A stored procedure is a predefined SQL query that is stored in the data dictionary. Stored procedures are designed to allow more efficient queries. Using stored procedures, you can reduce the amount of information that must be passed to the RDBMS and thus reduce network traffic and improve performance.

Database trigger: A database trigger is a procedure that is run automatically when an event occurs. This procedure, which is defined by the administrator or developer, triggers, or is run whenever this event occurs. This procedure could be an insert, a deletion, or even a selection of data from a table.

Sequence: The Oracle sequence generator is used to automatically generate a unique sequence of numbers in cache. By using the sequence generator you can avoid the steps necessary to create this sequence on your own such as locking the record that has the last value of the sequence, generating a new value, and then unlocking the record.

Segments, Extents, and Data Blocks

Within Oracle, the space used to store data is controlled by the use of logical

structures. These structures consist of the following:

Data blocks: A block is the smallest unit of storage in an Oracle database. The database block contains header information concerning the block itself as well as the data.

Extents: Extents consist of data blocks.

Segments: A segment is a set of extents used to store a particular type of data

An Oracle database can use four types of segments:

Data segment: Stores user data within the database.

Index segment: Stores indexes.

Rollback segment: Stores rollback information used when data must be rolled back.

Temporary segment: Created when a SQL statement needs a temporary work area; these segments are destroyed when the SQL statement is finished. These segments are used during various database operations, such as sorts.

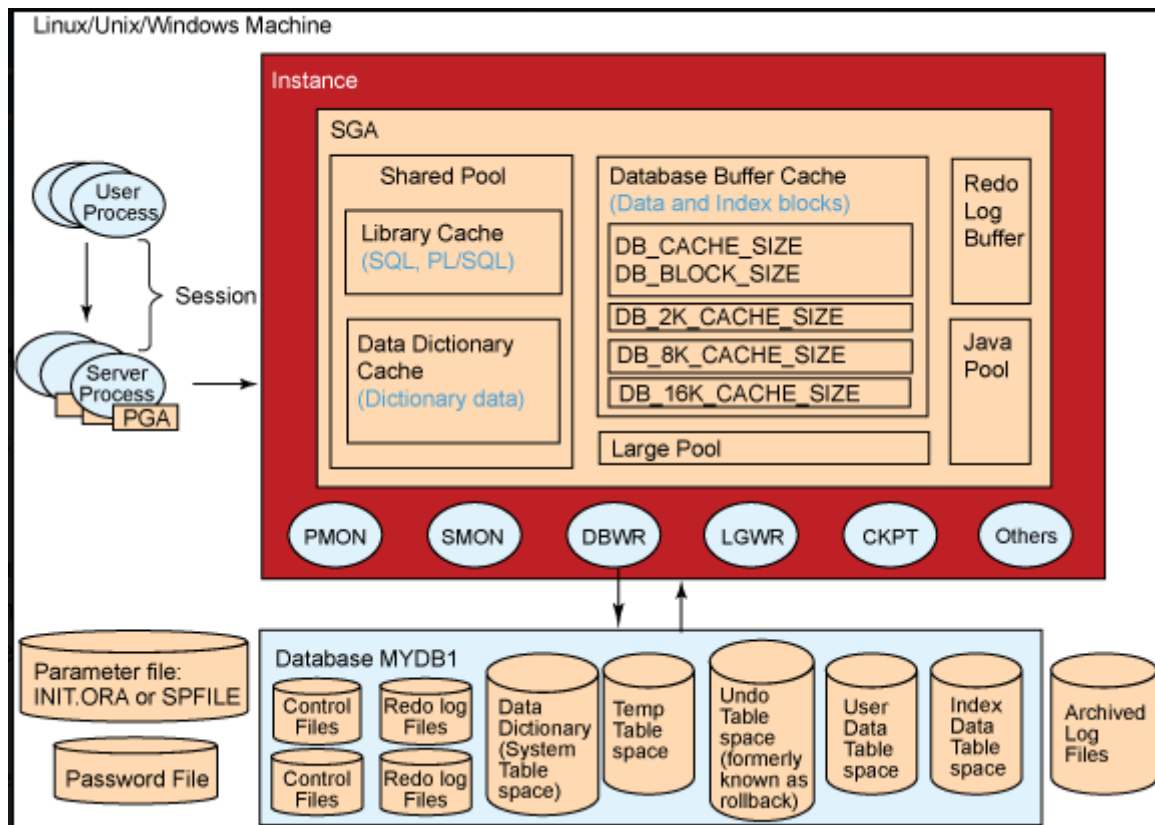
Extents

Extents are the building blocks of segments; in turn, they consist of data blocks. An extent is used to minimize the amount of wasted (empty) storage. As more and more data is entered into table spaces in your database, the extents used to store that data can grow or shrink as necessary. In this manner, many table spaces can share the same storage space without pre allocating the divisions between those table spaces.

At table space: creation time, you can specify the minimum number of extents to allocate as well as the number of extents to add at a time when that allocation has been used. This arrangement gives you efficient control over the space used in your database.

Data Blocks

Data blocks are the smallest pieces of an Oracle database; they are physically stored on disk. Although the data block in most systems is 2KB (2,048 bytes), you can change this size for efficiency depending on your application or operating system.



The Oracle Instance

The Oracle instance consists of the Oracle processes and shared memory necessary to access information in the database. The instance is made up of the user processes, the Oracle background processes, and the shared memory used by these processes.

The Oracle Memory Structure

New Term: Oracle uses shared memory for several purposes, including caching of data and indexes as well as storing shared program code. This shared memory is broken into various pieces, or memory structures. The basic memory structures associated with Oracle are the System Global Area (SGA) and the Program Global Area (PGA).

The System Global Area (SGA)

The SGA is a shared memory region that Oracle uses to store data and control information for one Oracle instance. The SGA is allocated when the Oracle instance starts and de-allocated when the Oracle instance shuts down. Each Oracle instance that starts has its own SGA. The information in the SGA consists of the following elements, each of which has a fixed size and is created at instance

startup:

The database buffer cache

This stores the most recently used data blocks. These blocks can contain modified data that has not yet been written to disk (sometimes known as dirty blocks), blocks that have not been modified, or blocks that have been written to disk since modification (sometimes known as clean blocks). Because the buffer cache keeps blocks based on a most recently used algorithm, the most active buffers stay in memory to reduce I/O and improve performance.

The redo log buffer

This stores redo entries, or a log of changes made to the database. The redo log buffers are written to the redo log as quickly and efficiently as possible.

Remember that the redo log is used for instance recovery in the event of a system failure.

The shared pool

This is the area of the SGA that stores shared memory structures such as shared SQL areas in the library cache and internal information in the data dictionary. The shared pool is important because an insufficient amount of memory allocated to the shared pool can cause performance degradation. The shared pool consists of the library cache and the data dictionary cache.

The Library Cache

The library cache is used to store shared SQL. Here the parse tree and the execution plan for every unique SQL statement are cached. If multiple applications issue the same SQL statement, the shared SQL area can be accessed by each to reduce the amount of memory needed and to reduce the processing time used for parsing and execution planning.

The Data-Dictionary Cache

The data dictionary contains a set of tables and views that Oracle uses as a reference to the database. Oracle stores information here about the logical and physical structure of the database. The data dictionary contains information such as the following:

User information, such as user privileges

Integrity constraints defined for tables in the database

Names and data types of all columns in database tables

Information on space allocated and used for schema objects

The data dictionary is frequently accessed by Oracle for the parsing of SQL statements. This access is essential to the operation of Oracle; performance bottlenecks in the data dictionary affect all Oracle users. Because of this, you should make sure that the data-dictionary cache is large enough to cache this data. If you do not have enough memory for the data-dictionary cache, you see a severe performance degradation. If you ensure that you have allocated sufficient memory to the shared pool where the data-dictionary cache resides, you should see no performance problems.

The Program Global Area (PGA)

The PGA is a memory area that contains data and control information for the Oracle server processes. The size and content of the PGA depends on the Oracle server options you have installed. This area consists of the following components:

Stack space: This is the memory that holds the session's variables, arrays, and so on.

Session information: If you are not running the multithreaded server, the session information is stored in the PGA. If you are running the multithreaded server, the session information is stored in the SGA.

Private SQL area: This is an area in the PGA where information such as binding variables and runtime buffers is kept.

Processes

New Term: In many operating systems, traditional processes have been replaced by threads or lightweight processes. The term process is used in this book to describe a thread of execution or a mechanism that can execute a set of code; process refers to the mechanism of execution and can refer to a traditional process or a thread.

The Oracle RDBMS uses two types of processes: user processes and Oracle processes (also known as background processes). In some operating systems (such as Windows NT), these processes are threads; for the sake of consistency, I will refer to them as processes.

User Processes

User, or client, processes are the user's connections to the RDBMS system. The user process manipulates the user's input and communicates with the Oracle

server process through the Oracle program interface. The user process is also used to display the information requested by the user and, if necessary, can process this information into a more useful form.

Oracle Processes

Oracle processes perform functions for users. Oracle processes can be split into two groups: server processes (which perform functions for the invoking process) and background processes (which perform functions on behalf of the entire RDBMS).

Server Processes (Shadow Processes)

Server processes, also known as shadow processes communicate with the user and interact with Oracle to carry out the user's requests. For example, if the user

process requests a piece of data not already in the SGA, the shadow process is responsible for reading the data blocks from the data files into the SGA. There can be a one-to-one correlation between user processes and shadow processes (as in a dedicated server configuration); although one shadow process can connect to multiple user processes (as in a multithreaded server configuration), doing so reduces the utilization of system resources.

Background Processes

Background processes are used to perform various tasks within the RDBMS system. These tasks vary from communicating with other Oracle instances and performing system maintenance and cleanup to writing dirty blocks to disk. Following are brief descriptions of the nine

Oracle background processes:

DBWR (Database Writer) DBWR is responsible for writing dirty data blocks from the database block buffers to disk. When a transaction changes data in a data block, that data block need not be immediately written to disk. Therefore, the DBWR can write this data to disk in a manner that is more efficient than writing when each transaction completes. The DBWR usually writes only when the database block buffers are needed for data to be read. Data is written in a least recently used fashion. For systems in which asynchronous I/O (AIO) is available, there should be only one DBWR process. For systems in which AIO is not available, performance can be greatly enhanced by adding more DBWR processes.

LGWR (Log Writer)- The LGWR process is responsible for writing data from the log buffer to the redo log.

CKPT (Checkpoint)- The CKPT process is responsible for signaling the DBWR process to perform a checkpoint and to update all the data files and control files for the database to indicate the most recent checkpoint. A checkpoint is an event in which all modified database buffers are written to the data files by the DBWR. The CKPT process is optional. If the CKPT process is not present, the LGWR assumes these responsibilities.

PMON (Process Monitor)-PMON is responsible for keeping track of database processes and cleaning up if a process prematurely dies (PMON cleans up the cache and frees resources that might still be allocated). PMON is also responsible for restarting any dispatcher processes that might have failed.

SMON (System Monitor)- SMON performs instance recovery at instance startup. This includes cleaning temporary segments and recovering transactions that have died because of a system crash. The SMON also defragments the database by coalescing free extents within the database.

RECO (Recovery)- RECO is used to clean transactions that are pending in a distributed database. RECO is responsible for committing or rolling back the local portion of the disputed transactions.

ARCH (Archiver)-ARCH is responsible for copying the online redo log files to archival storage when they become full. ARCH is active only when the RDBMS is operated in ARCHIVELOG mode. When a system is not operated in ARCHIVELOG mode, it might not be possible to recover after a system failure. It is possible to run in NOARCHIVELOG mode under certain circumstances but typically should operate in ARCHIVELOG mode.

LCKn (Parallel Server Lock)-Up to 10 LCK processes are used for inter-instance locking when the Oracle Parallel Server option is used.

Dnnn (Dispatcher)- When the Multithreaded Server option is used, at least one Dispatcher process is used for every communications protocol in use. The Dispatcher process is responsible for routing requests from the user processes to available shared server processes and back.

Dr. E.F. Codd proposed a relational model for database systems in 1970

A relational database is a collection of relations or two-dimensional tables.

A Relational database can be accessed & modified by executing SQL statements. which is the American National Standard Institute(ANSI) standard language for operating relational databases.

Using SQL we can communicate with the Oracle server.

An Oracle server provides an open, comprehensive (including all & everything) and integrated approach to information management.

An Oracle server consists of two things:

-oracle database

-oracle server instance

Every time a database is started an SGA (System Global Area) is allocated and the Oracle background process is started the SGA is an area used for database information shared by the database user the combination of the background processes and memory buffer is called an Oracle instance.

Sql Was developed by IBM 1970s also called SEQUEL.

Oracle SQL complies with industry-accepted standards. Industry_accepted committees are ANSI and ISO both have accepted SQL as the standard language for relational databases

SQL statements:

- | | |
|-------------------------------|---|
| 1. Data Retrieval | -SELECT |
| 2. DML | -INSERT, UPDATE, DELETE and MERGE |
| 3. DDL | -CREATE, ALTER,DROP, RENAME and TRUNCATE |
| 4. Transaction Control | -COMMIT, ROLLBACK and SAVEPOINT |
| 5. DCL | -GRANT and REVOKE |

4 SYSTEM DESIGN

4.1 INTRODUCTION

System design is a transaction from user-oriented documents to document-oriented programmers or database personnel, it emphasizes on translating performance specification into design specification and involves conceiving, planning, and then carrying out the plan by generating the necessary reports and outputs. The design phase acts as a bridge between the software requirement specification and the implementation phase, which satisfies the requirements.

In the design phase, the detailed design of the system selected in the study phase is accomplished. Major steps in design are:

- Method of data captures and data input.
- Modification to be done to convert the existing system to the proposed system
- Operations to be performed to produce output and maintain the file
- Design input and output forms
- Output to be produced

A modular approach has been adapted in the development of the proposed system. Each module is designed for a specific application and they are operated independently.

4.2 DATA FLOW DIAGRAM

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 consists a 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.

Larry Constantine first developed the DFD as a way of expressing system requirements in a graphical form, this lead to the modular design.

A DFD is also known as a "bubble Chart" has the purpose of clarifying system requirements and identifying major transformations that will become programs in system design. So it is the starting point of the design to the lowest level of detail. A DFD consists of a series of bubbles joined by data flows in the system.

DFD SYMBOLS:

In the DFD, there are four symbols

1. A square defines a source(originator) or destination of system data
2. An arrow identifies data flow. It is the pipeline through which the information flows

3. A circle or a bubble represents a process that transforms incoming data flow into outgoing data flows.
4. An open rectangle is a data store, data at rest or a temporary repository of data



Process that transforms data flow.



Source or Destination of data



Data flow



Data Store

CONSTRUCTING A DFD:

Several rules of thumb are used in drawing DFD's:

1. Process should be named and numbered for an easy reference. Each name should be representative of the process.
2. The direction of flow is from top to bottom and from left to right. Data traditionally flow from source to the destination although they may flow back to the source. One way to indicate this is to draw long flow line back to a source. An alternative way is to repeat the source symbol as a destination. Since it is used more than once in the DFD it is marked with a short diagonal.
3. When a process is exploded into lower level details, they are numbered.
4. The names of data stores and destinations are written in capital letters. Process and dataflow names have the first letter of each word capitalized

A DFD typically shows the minimum contents of data store. Each data store should contain all the data elements that flow in and out.

Questionnaires should contain all the data elements that flow in and out. Missing interfaces redundancies and like is then accounted for often through interviews.

FEATURES OF DFD's

1. The DFD shows flow of data, not of control loops and decision are controlled considerations do not appear on a DFD.
2. The DFD does not indicate the time factor involved in any process whether the data flows take place daily, weekly, monthly or yearly.
3. The sequence of events is not brought out on the DFD.

TYPES OF DATA FLOW DIAGRAMS

1. Current Physical
2. Current Logical
3. New Logical
4. New Physical

1. CURRENT PHYSICAL:

In Current Physical DFD process label include the name of people or their positions or the names of computer systems that might provide some of the overall system-processing label includes an identification of the technology used to process the data. Similarly data flows and data stores are often labels with the names of the actual physical media on which data are stored such as file folders, computer files, business forms or computer tapes.

2. CURRENT LOGICAL:

The physical VBects at the system are removed as much as possible so that the current system is reduced to its essence to the data and the processors that transforms them regardless of actual physical form.

3. NEW LOGICAL:

This is exactly like a current logical model if the user were completely happy with the user were completely happy with the functionality of the current system but had problems with how it was implemented typically through the new logical

model will differ from current logical model while having additional functions, absolute function removal and inefficient flows recognized.

4. NEW PHYSICAL:

The new physical represents only the physical implementation of the new system.

RULES GOVERNING THE DFD'S

➤ PROCESS:

- No process can have only outputs.
- No process can have only inputs. If an object has only inputs than it must be a sink.
- A process has a verb phrase label.

➤ DATA STORE:

- Data cannot move directly from one data store to another data store, a process must move data.
- Data cannot move directly from an outside source to a data store, a process, which receives, must move data from the source and place the data into data store
- A data store has a noun phrase label.

➤ SOURCE OR SINK:

- Data cannot move directly from a source to sink it must be moved by a process
- A source and /or sink has a noun phrase label

➤ DATA FLOW:

A Data Flow has only one direction of flow between symbol. It may flow in both directions between a process and a data store to show a read before an update. The later is usually indicated however by two separate arrows since these happen at different type.

A join in DFD means that exactly the same data comes from any of two or more different processes data store or sink to a common location.

A data flow cannot go directly back to the same process it leads. There must be at least one other process that handles the data flow produce some other data flow returns the original data into the beginning process.

A Data flow to a data store means update (delete or change).

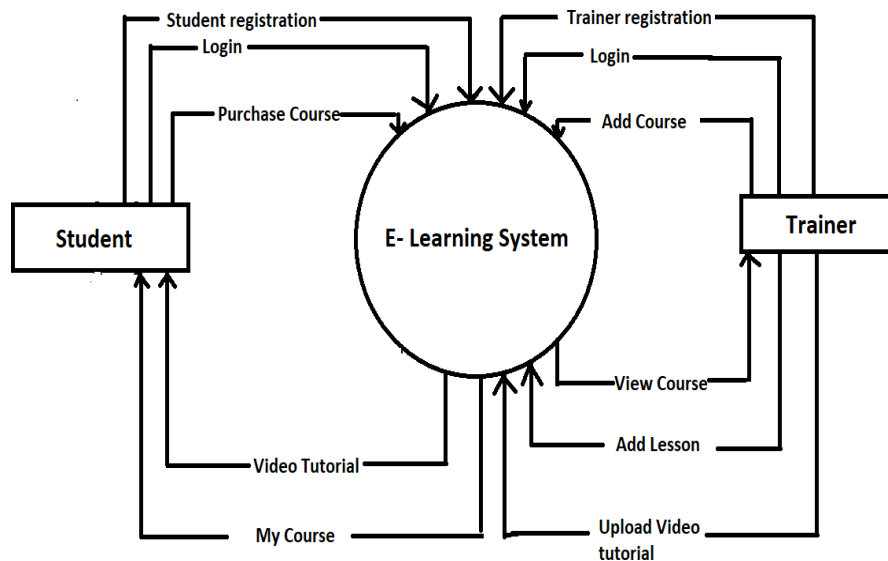
A data Flow from a data store means retrieve or use.

A data flow has a noun phrase label more than one data flow noun phrase can

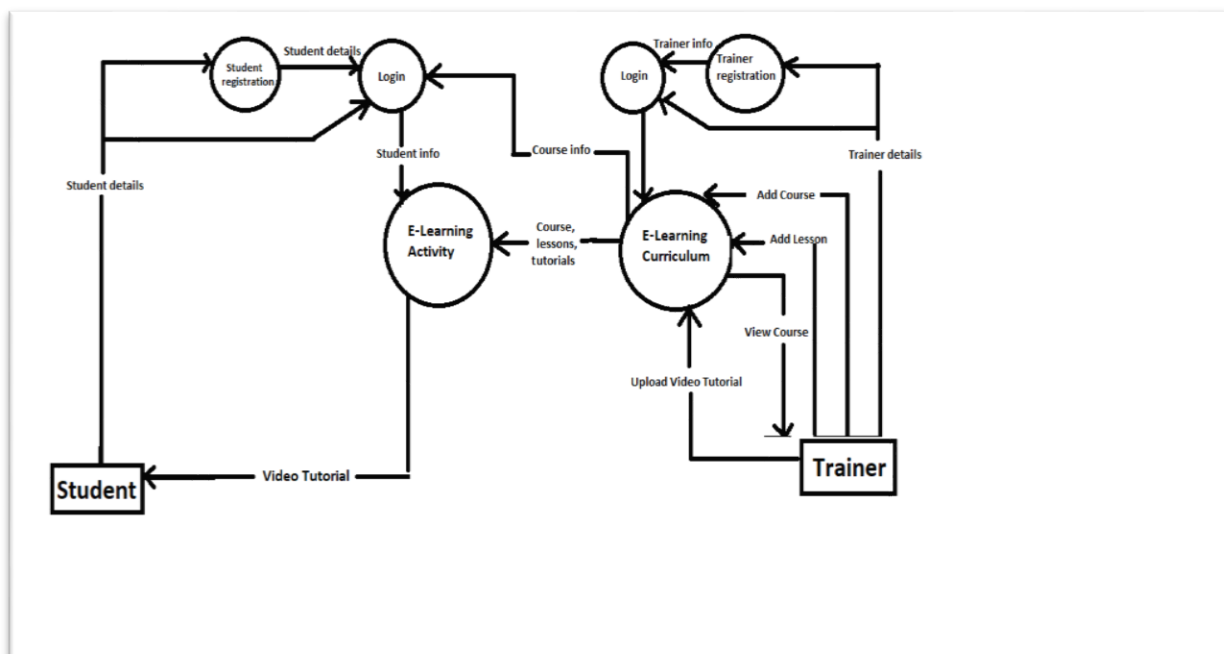
appear on a single arrow as long as all of the flows on the same arrow move together as one package.

4.2 DATA FLOW DIAGRAM

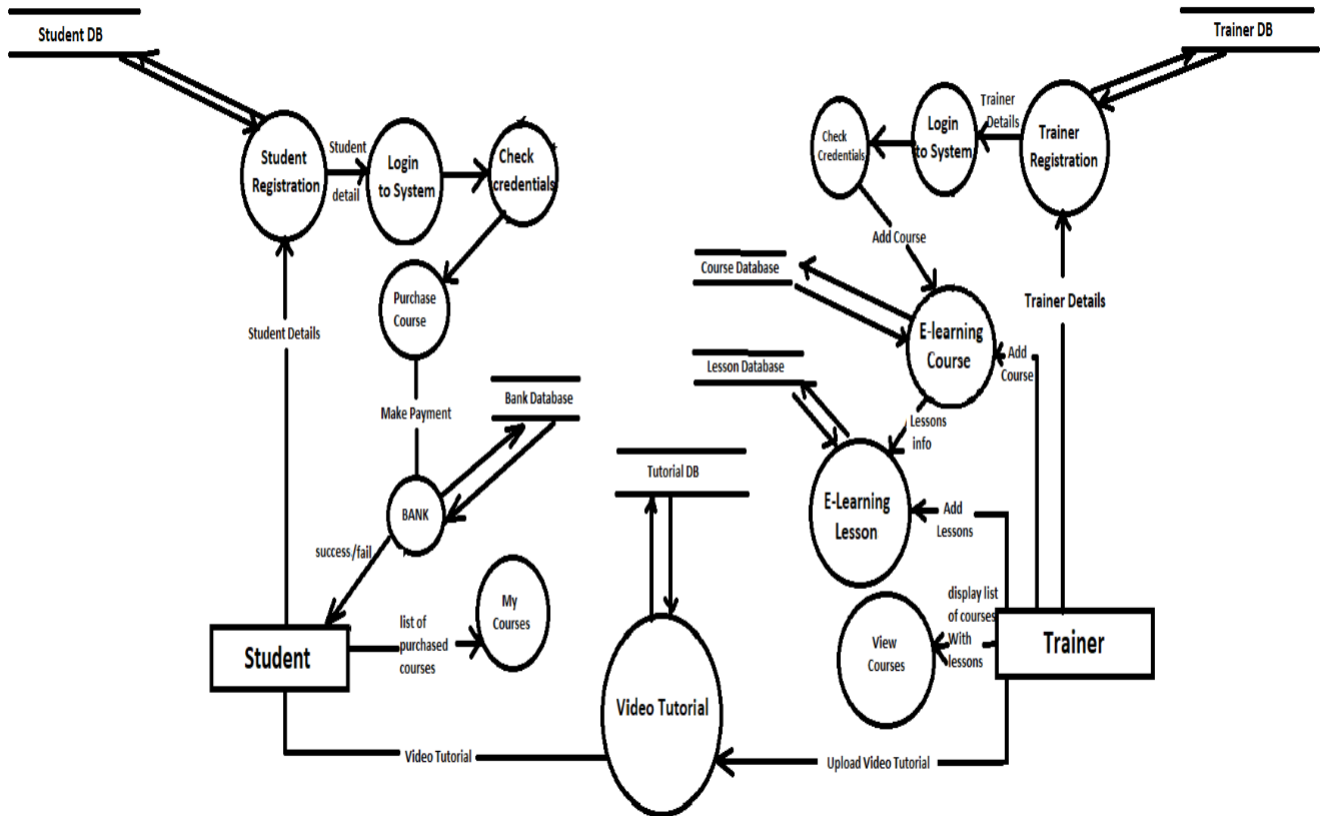
DFD OF THE PROJECT:



0LEVEL DFD



1level DFD



2 LEVEL DFD

Flow chart of LMS

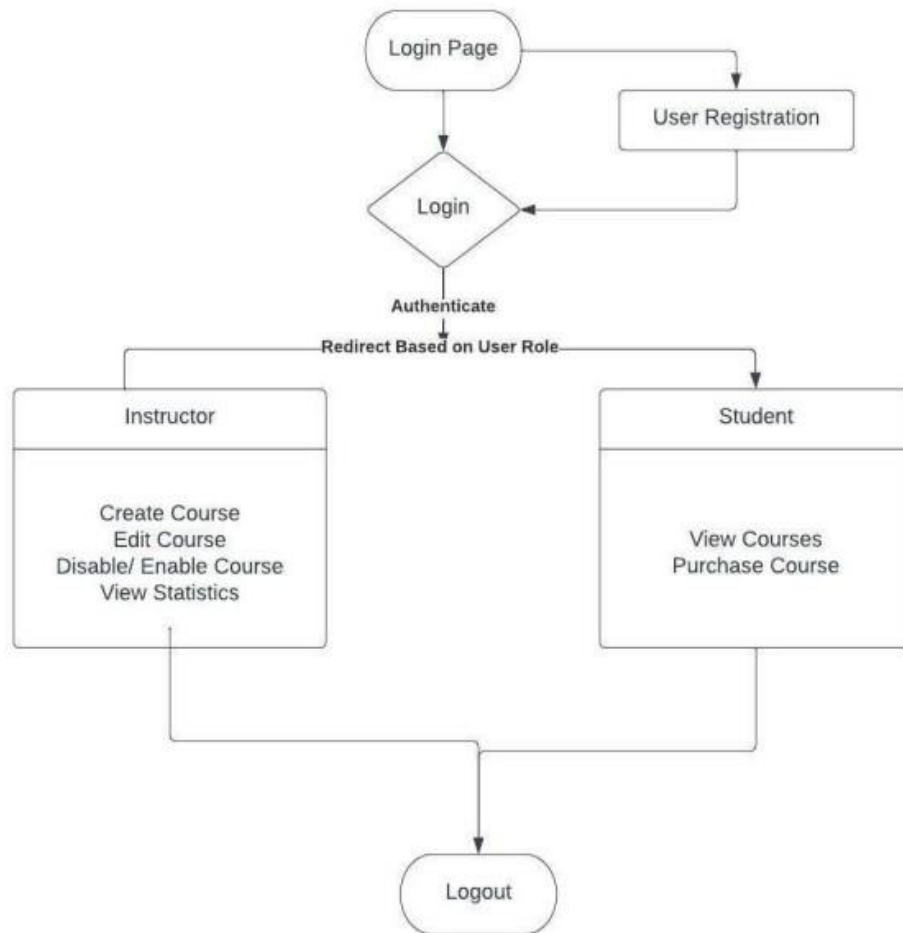


Fig 1: Flow Chart

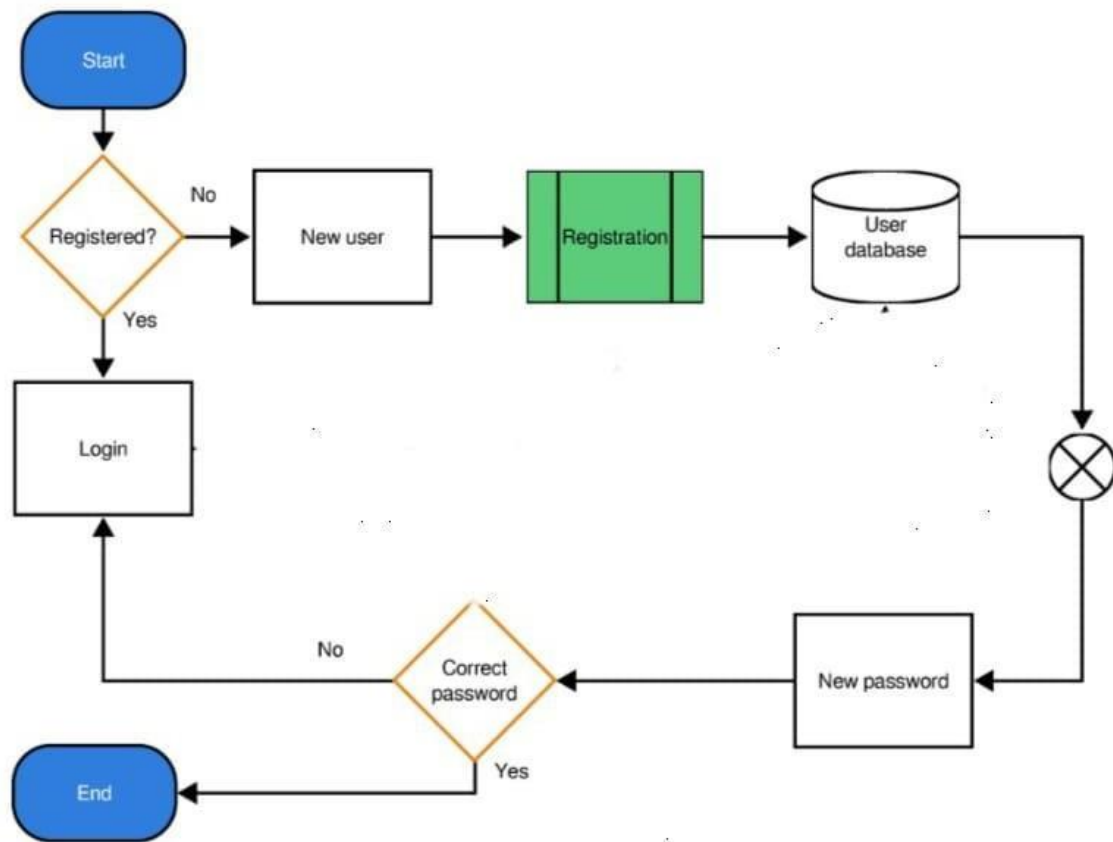


Fig:- Flowchart 2

4.3 INPUT DESIGN

The input data are collected and organized to make data entry easy, logical, and error free. Each area in the input form should be identified should be specified for the user what to write and where to write.

A screen is an actually a display station that has a buffer for storing data. The main objective of screen design is for simplicity, accurate and quick data capture or entry.

The objective in the input design is to ensure that the data which will be processed by the system is collected and inserted into the system efficiently according to the specified requirements, and with the minimum errors. The basic design consideration that would satisfy the user requirements were as follows.

Our guidelines are:

4.3.1 Use the same format throughout the project.

4.3.2 Allow ample space to avoid data overcrowding because it causes eyestrain and may reduce the interest of the user.

4.3.3 Use easy tolerant and consistent terms such as “add”, “Delete” and “close”.

4.4 OUTPUT DESIGN

Report design is a very important concept in the computerized system, without reliable output the user may feel the entire system is unnecessary and avoid using it. The proper output design is important in any system and facilitates effective decision-making.

The basic output considerations were as follows:-

4.4.1 Simple and legible methods were used for output using standard display controls.

4.4.2 All the output screens were informative and integrative in such a way the user could fulfill his requirements.

4.4.3 Quality reports were made available to the user.

4.5 E-R DIAGRM

The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases.

Entity

An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

Attributes

Entities are represented by means of their properties, called attributes. All attributes have values. For example, a student entity may have name, class, and age as attributes.

There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Types of Attributes

Simple attribute – Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.

Composite attribute – Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.

Derived attribute – Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.

Single-value attribute – Single-value attributes contain single value. For example – Social_Security_Number

Multi-value attribute – Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

These attribute types can come together in a way like –

simple single-valued attributes

simple multi-valued attributes

composite single-valued

attributes composite multi-valued

attributes Entity-Set and Keys

Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.

For example, the roll number of a student makes him/her identifiable among students.

Super Key – A set of attributes (one or more) that collectively identifies an entity in an entity set.

Candidate Key – A minimal super key is called a candidate key. An entity set may have more than one candidate key.

Primary Key – A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.

Relationship

The association among entities is called a relationship. For example, an employee works at a department, a student enrolls in a course. Here, Works_at and Enrolls are called relationships.

Relationship Set

A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called descriptive attributes.

Degree of Relationship

The number of participating entities in a relationship defines the degree of the relationship.

Binary = degree 2

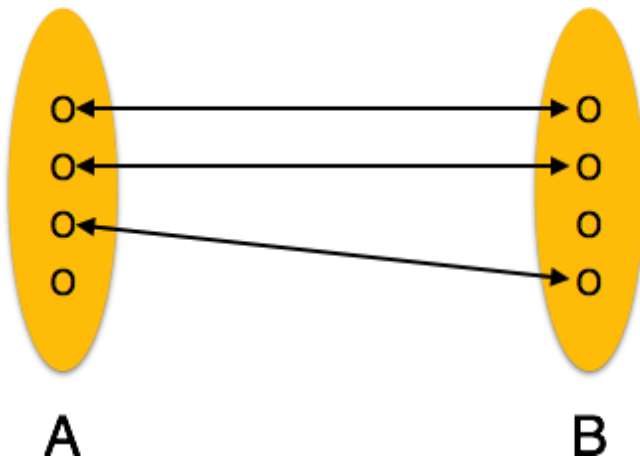
Ternary = degree 3

n-ary = degree n

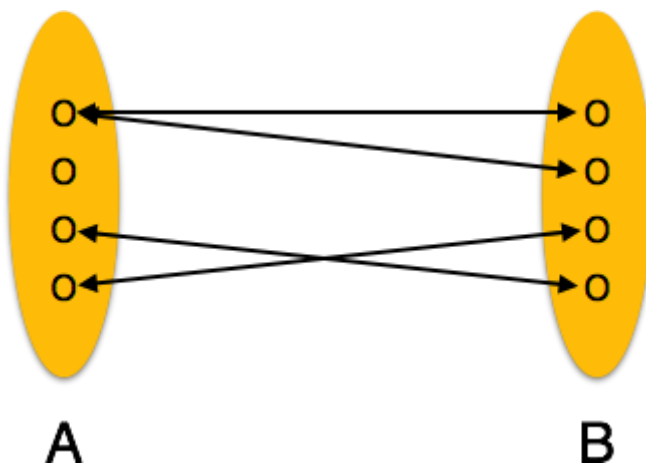
Mapping Cardinalities

Cardinality defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set.

One-to-one – One entity from entity set A can be associated with at most one entity of entity set B and vice versa.



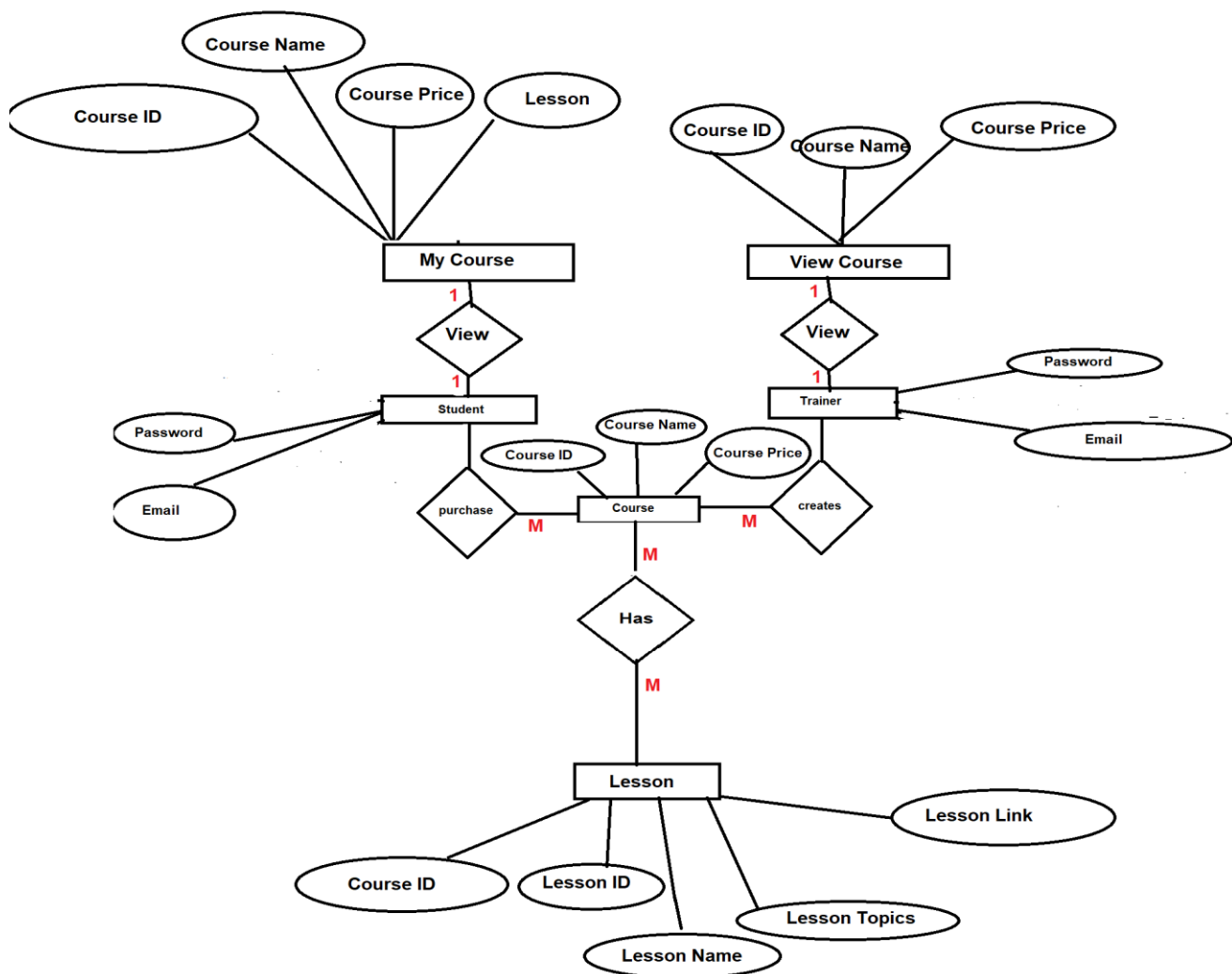
One-to-many – One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.



Many-to-one – More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set

Many-to-many – One entity from A can be associated with more than one entity from B and vice versa.

ER-DIAGRAM OF LMS



4.6 **DATABASE DESIGN** It is a process of designing the database file, which is the key source of the information in the system. The objective of the database is to design is to provide storage and it contributes to the overall efficiency of the system. The file should be properly designed and planned for collection, accumulation, editing, and retrieving of the required information.

The primary objectives of a database design are fast response time to inquiries for more information at low cost, control of redundancy, clarity and ease of use, accuracy, and integrity of the system, fast recovery, and availability of powerful end-user languages. The theme behind a database is to handle information as an integrated whole thus the main objective is to make information access easy, quick, inexpensive, and flexible for the users.

DATABASE LAYOUT

The database tables that have been identified for storing data are:

TABLE 1

Table name: Course

Description: Contains Course Details

| TERMS USED | DATATYPE | WIDTH | CONSTRAINTS |
|--------------|----------|-------|-------------|
| COURSE_ID | Int | 50 | Primary Key |
| COURSE_NAME | VARCHAR | 255 | - |
| COURSE_PRICE | Int | 50 | - |

TABLE 2

Table name: Lesson

Description:
Contains details
about Lessons.

| TERMS USED | DATATYPE | WIDTH | CONSTRAINTS |
|---------------|----------|-------|-------------|
| LESSON_ID | Int | 50 | Primary Key |
| LESSON_LINK | VARCHAR | 255 | - |
| LESSON_NAME | VARCHAR | 255 | - |
| LESSON_TOPICS | VARCHAR | 255 | - |
| COURSE_ID | Int | 50 | - |

TABLE 3

Table name: Course_Lesson_List

Description: Maps the lessons through

| TERMS USED | DATATYPE | WIDTH | CONSTRAINTS |
|-----------------------|----------|-------|-------------|
| COURSE_COURSE_ID | Int | 50 | - |
| LESSON_LIST_LESSON_ID | Int | 50 | Primary Key |

TABLE 4

Table name: USERS

Description: Contains details about all the users registering themselves.

| TERMS USED | DATATYPE | WIDTH | CONSTRAINT |
|------------|----------|-------|-------------|
| ID | NUMBER | 50 | PRIMARY KEY |
| EMAIL | VARCHAR | 255 | NOT NULL |
| PASSWORD | VARCHAR | 255 | - |
| ROLE | VARCHAR | 255 | - |
| USERNAME | VARCHAR | 255 | - |

TABLE 5

Table name: USERS_COURSE_LIST

Description: Maps the user with the course they purchased through course_id.

| TERMS USED | DATATYPE | WIDTH | CONSTRAINTS |
|-----------------------|----------|-------|-------------|
| USER_ID | Int | 50 | - |
| COURSE_LIST_COURSE_ID | Int | 50 | - |

4.7 CODING

index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Index Page</title>
  <style>

    body {
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
      padding: 0;
      margin: 0;
      background: linear-gradient(180deg, #FAF0D7, #ADC4CE);
    }

    nav,
    .nav-links {
      display: flex;
    }

    nav {
      justify-content: space-between;
      margin: 20px 5%;
      align-items: center;
      height: 10vh;
      background-color: rgba(255, 255, 255, 0.8);
      border-radius: 10px;
      padding: 10px;
      box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
    }

    .nav-links {
      gap: 2rem;
      list-style: none;
      font-size: 1.5rem;
      font-weight: 600;
      margin: 0;
    }

    a {
```



```

        color: black;
        text-decoration: none;
    }

    a:hover {
        color: indigo;
    }

    .logo {
        font-size: 2rem;
        font-weight: 900;
    }

    .main-page {
        width: 90%;
        margin: 20px 5%;
        padding: 2%;
        font-size: 18px;
        display: flex;
        flex-direction: column;
        gap: 30px;
        background-color: rgba(255, 255, 255, 0.8);
        border-radius: 10px;
        box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
    }

    .heading {
        width: 100%;
        border-radius: 5px;
        padding: 20px;
        font-size: 18px;
        background-color: #FAF0D7;
        border-bottom: 2px solid #ADC4CE;
    }

    .container {
        width: 100%;
        display: flex;
        flex-direction: column;
        gap: 15px;
    }

    .content {
        border-radius: 10px;
        width: 100%;
        padding: 20px;
        background-color: #FAF0D7;
        box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
    }

    .content h2 {

```

```

        color: indigo;
    }

    .content.details {
        padding-right: 20px;
        text-align: justify;
    }

    /* ... (previous styles) */

    /* Navigation link hover effects */
    .nav-links a:hover {
        color: #ffffff;
        /* Change text color on hover */
        background-color: darkorange;
        /* Add a background color on hover */
        transition: color 0.3s ease-in-out, background-color 0.3s ease-in-out;
        /* Add a smooth transition effect */
    }

    /* ... (rest of the styles) */

    /* Media Query for tablets and smaller screens */
    @media only screen and (max-width: 768px) {
        .nav-links {
            flex-direction: column;
            align-items: center;
        }

        .logo {
            text-align: center;
        }

        .main-page {
            padding: 5%;
        }
    }
</style>
</head>

<body>
    <nav id="desktop-nav">
        <div class="logo">LearnSphere</div>
        <div>
            <ul class="nav-links">
                <li><a href="/register">Register</a></li>
                <li><a href="/login">Login</a></li>
                <li><a href="#aboutus">About Us</a></li>
                <li><a href="#contact">Contact</a></li>
            </ul>
        </div>
    </nav>

```

```

        </ul>
    </div>
</nav>
<div class="main-page">
    <div class="heading">
        <h3>Embark on a transformative learning journey with LearnSphere,
where education meets opportunity,
        and empower yourself for a brighter future filled with knowledge
and thriving job opportunities.</h3>
    </div>
    <div class="container">
        <div id="full-stack" class="content">
            <h2>Full-Stack Development</h2>
            <div class="details">
                <ul>
                    <li>Our comprehensive full stack development
                        training program equips you with the skills needed
                        to create end-to-end web applications.</li>

                    <li>Join our intensive full stack development training
                        and master both front-end and back technologies.</li>

                    <li>Our full stack development training covers a wide
                        range of programming languages,
                        frameworks, and databases to make you a well-
                        rounded developer.</li>

                    <li>Learn to seamlessly integrate user interfaces and
                        server-side logic through hands-on full
                        stack development training.</li>

                    <li>Our experienced instructors guide you through
                        real-world projects, enabling you to apply
                        your full stack development skills in practical scenarios.</li>
                </ul>
            </div>
        </div>
        <div id="testing" class="content">
            <h2>Software Automation Testing</h2>
            <div class="details">
                <ul>
                    <li>Our software automation testing training
                        program equips you with the expertise to automate
                        testing processes and ensuring software quality.</li>

                    <li>Join our hands-on software automation testing
                        training to learn how to develop and execute
                        automated test scripts efficiently.</li>

                    <li>From test planning to script development, our
                        software automation testing training covers

```

```

        the entire automation lifecycle.</li>

        <li>Enroll in our software automation testing training
            to master popular automation tools and
            frameworks used in the industry.</li>

        <li>Our experienced trainers guide you through
            creating robust automated test suites that
            enhance the reliability of software applications.</li>
    </ul>
</div>
</div>
</div>

```

```
</body>
```

```
</html>
```

register.html

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
    <meta charset="ISO-8859-1">
```

```
    <title>Register Page</title>
```

```
    <style>
```

```

        body {
            font-family: Georgia, 'Times New Roman', Times, serif;
            background: linear-gradient(180deg, #FAF0D7, #ADC4CE);
            height: 98.3vh;
            font-size: 20px;
            display: flex;
            flex-direction: column;
            justify-content: space-evenly;
            align-items: center;
            padding: 0;
        }
    
```

```

        .main-page {
            height: 90vh;
            width: 100%;
            font-size: 20px;
            display: flex;
            flex-direction: row;
            justify-content: space-around;
        }
    
```

```

        align-items: center;
        padding: 0;
    }

    form {
        border: 2px solid black;
        box-shadow: 0 0 10px 2px;
        padding: 50px;
        width: 600px;
        height: 70%;
        display: flex;
        flex-direction: column;
        justify-content: center;
        align-items: center;
        gap: 30px;
    }

    label {
        width: 150px;
        font-weight: 700;
    }

    .input-data {
        font-size: 20px;
        width: 500px;
        display: flex;
        justify-content: space-between;
        flex-direction: row;
        gap: 5px;
    }

    .text-fields {
        border-radius: 5px;
        font-size: 17px;
        height: 22px;
        width: 350px;
        margin-right: 150px;
    }

    .radio {
        width: 40px;
        cursor: pointer;
    }

    button {
        font-family: fantasy;
        font-size: 18px;
        width: 150px;
        height: 50px;
    }

```

```

        border-radius: 10px;
        background-color: #ADC4CE;
        cursor: pointer;
        transition: 0.3s
    }

    .content-holder {
        width: 60%;
        height: 90%;
        margin-left: 25px;
        padding: 10px;
        display: flex;
        flex-direction: column;
        justify-content: flex-start;
        align-items: flex-start;
    }

    form span {
        font-weight: 600;
        width: 260px;
    }

    .role {
        width: 150px;
    }

    .form-holder {
        display: flex;
        justify-content: center;
        align-items: center;
        width: 50%;
        height: 80%;
    }

    button:hover {
        box-shadow: 0 0 10px 2px #000000;
    }

    nav,
    .nav-links {
        display: flex;
    }

    nav {
        justify-content: space-between;
        margin: 0px 45px 0px;
        align-items: center;
        height: 4vh;
    }

    .nav-links {

```

```

        gap: 2rem;
        list-style: none;
        font-size: 1.5rem;
        font-weight: 600;
    }

    a {
        color: black;
        text-decoration: none;
        transition: 0.5s;
    }

    a:hover {
        color: black;
        text-decoration: underline;
        text-underline-offset: 0.5rem;
        text-decoration-color: indigo;
    }

    .logo {
        float: left;
        display: block;
        font-size: 2rem;
        font-weight: 900;
        width: 1200px;
    }

    .logo:hover {
        cursor: default;
    }

    form .instructions {
        font-size: 16px;
    }

    /* CSS for error message */
    .error-message {
        color: red; /* Set text color to red */
        font-weight: bold; /* Make the error message bold */
        font-size: 0.8em; /* Initial font size */
        position: absolute; /* Set position to absolute */
        top: 300px; /* Align with the top of the input box */
        right: 400px; /* Adjust position to align with the end of the input */
        margin-top: 2px; /* Adjust the vertical margin as needed */
    }

    @media screen and (max-width: 768px) {
        .main-page {
            flex-direction: column;
        }
    }

```

```

    </style>
</head>

<body>
  <nav id="desktop-nav">
    <div class="logo">LearnSphere</div>
    <div>
      <ul class="nav-links">
        <li><a href="/index">Home</a></li>
        <li><a href="#aboutus">About Us</a></li>
        <li><a href="#contact">Contact</a></li>
        <li><a href="/login">Login</a></li>
      </ul>
    </div>
  </nav>
  <div class="main-page">
    <div class="content-holder">
      <h3>Benefits you receive:</h3>
      <ul>
        <li>
          <h4>Comprehensive Tech Education:</h4> Discover a one-
stop online platform that offers in-depth
Automation Testing.
          </li>
          <li>
            <h4>Hands-On Learning:</h4> Immerse yourself in
practical, hands-on exercises and real-world
projects that empower you to apply your skills effectively.
            </li>
            <li>
              <h4>Industry-Relevant Skills:</h4> Gain proficiency in
three high-demand tech domains, ensuring
you're well-equipped for success in today's competitive job market.
              </li>
              <li>
                <h4>Expert Instructors:</h4> Learn from experienced
professionals who guide you through every step
of your journey to mastery.
                </li>
                <li>
                  <h4>Cutting-Edge Tools:</h4> Get hands-on experience
with the latest tools, frameworks, and
technologies used in the industry.
                  </li>

```



```

        <li>
            <h4>Certification:</h4> Earn certificates of completion to
            validate your expertise and showcase your
            accomplishments to potential employers.
        </li>
    </ul>
</div>
<div class="form-holder">

    <form action="/addUser" method="post">
        <div class="input-data">
            <label>Username</label>
            <input type="text" name="username" class="text-fields"
required>

        </div>
        <div class="input-data">
            <label>Email</label>
            <input type="text" name="email" class="text-fields"
required>

        <!-- Place this code wherever you want to display the error
message -->
            <div th:if="${errorMessage}" class="error-message">
                <p th:text="${errorMessage}"></p>
            </div>
        </div>
        <div class="input-data">
            <label>Password</label>
            <input type="password" name="password" class="text-
fields" required>

        </div>
        <div class="input-data">
            <label class="role">Role</label>
            <input type="radio" name="role" class="radio"
value="trainer" required>

            <span>Trainer</span>
            <input type="radio" name="role" class="radio"
value="student" required>

            <span>Student</span>
        </div>
        <button type="submit">Register</button>
        <div class="instructions">
            <span>Instructions:</span>
            <ul>
                <li>Username should be between 8-25
characters</li>
                <li>Password should contain atleast one number and
special characters</li>
            </ul>
        </div>
        <a href="/login">Already an User? Login here</a>
    </form>

```

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

```
</body>
```

```
</html>
```

login.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Login</title>
<style>
```

```
    body{
        font-family: Georgia, 'Times New Roman', Times, serif;
        height: 98vh;
        font-size: 20px;
        display: flex;
        flex-direction: column;
        justify-content: flex-start;
        align-items: center;
        background: linear-gradient(180deg, #FAF0D7 , #ADC4CE);
    }
    form{
        border: 2px solid black;
        box-shadow: 0 0 10px 2px;
        padding: 20px;
        height: 300px;
        width: 400px;
        display: flex;
        flex-direction: column;
        justify-content: space-around;
        align-items: center;
        gap: 30px;
    }

    label{
        width: 150px;
    }
    .input-data{
        display: flex;
        justify-content: space-between;
```

```

        flex-direction: row;
        gap: 5px;
    }

    .text-fields{
        border-radius: 5px;
        height: 22px;
        width: 220px;
    }

    button{
        font-size: 20px;
        width: 150px;
        height: 50px;
        font-family: fantasy;
        font-size: 18px;
        border-radius: 10px;
        background-color: #ADC4CE;
        cursor: pointer;
    }

    button:hover{
        box-shadow: 0 0 10px 2px #000000;
        transform: translateY(-3px);
        transition: 0.3s;
    }

    nav,
    .nav-links{
        display: flex;
    }

    nav{
        justify-content: space-around;
        align-items: center;
        height: 10vh;
    }

    .nav-links{
        gap: 2rem;
        list-style: none;
        font-size: 1.5rem;
        font-weight: 600;
    }

    a{
        color: black;
        text-decoration: none;
        text-decoration-color: white;
    }

    a:hover{

```

```

    color: black ;
    text-decoration: underline;
    text-underline-offset: 0.5rem;
    text-decoration-color: indigo;
}

.logo{
    font-size: 2rem;
    font-weight: 900;
    width: 1200px;
}

.logo:hover{
    cursor: default;
}

.main-page{
    height: 90vh;
    display: flex;
    justify-content: center;
    align-items: center;
}

.message label{
    color: red;
}

.text-fields {
    border-radius: 5px;
    font-size: 17px;
    height: 22px;
    width: 250px;
}

}

</style>
</head>
<body>
    <nav id="desktop-nav">
    <div class="logo">LearnSphere</div>
    <div>
    <ul class="nav-links">
        <li><a href="/index">Home</a></li>
        <li><a href="#services">Services</a></li>
        <li><a href="#aboutus">About Us</a></li>
        <li><a href="#contact">Contact</a></li>
        <li><a href="/register">Register</a></li>
    </ul>

```

```

</div>
</nav>
<div class="main-page">
  <form action="/validateUser" method="post">
    <div class="input-data">
      <label>Email</label>
      <input type="text" class="text-fields" name="email" required>
    </div>
    <div class="input-data">
      <label>Password</label>
      <input type="password" class="text-fields" name="password" required>
    </div>

    <button type="submit">Login</button>
    <div class="message">
      <label th:text = "${errorMessage}"></label>
    </div>
    <a href="/register">Not an User? Register here</a>
  </form>
</div>

</body>
</html>

```

trainer.home

```

<!DOCTYPE html>
<html>

```

```

<head>
  <meta charset="ISO-8859-1">
  <title>Trainer Page</title>
  <style>

```

```

body {
  font-family: Georgia, 'Times New Roman', Times, serif;
  height: 98vh;
  font-size: 20px;
  display: flex;
  flex-direction: column;
  justify-content: flex-start;
  align-items: center;

```

```

        background: linear-gradient(65deg, #FAF0D7, #ADC4CE);
        gap: 50px;
    }

    .options {
        height: 50vh;
        display: flex;
        flex-direction: column;
        justify-content: center;
        align-items: center;
        gap: 35px;
    }

    .options button {
        height: 60px;
        width: 400px;
        border-radius: 10px;
        cursor: pointer;
        background-color: #FAF0D7;
        font-weight: 800;
        font-size: 20px;
        transition: 0.3s;
    }

    button:hover {
        box-shadow: 0 0 10px 2px black;
        transform: translateY(-3px);
    }

    .welcome {
        display: flex;
        justify-content: center;
        align-items: center;
        gap: 15px;
    }

    nav,
    .nav-links {
        display: flex;
    }

    nav {
        justify-content: space-around;
        margin: 0px 45px 0px;
        align-items: flex-start;
        height: 2px;
        gap: 45rem;
    }

    .nav-links {
        gap: 2rem;
    }

```

```

        list-style: none;
        font-size: 1.5rem;
        width: 300px;
        font-weight: 600;
    }

    a {
        color: black;
        text-decoration: none;
        text-decoration-color: white;
    }

    a:hover {
        color: black;
        text-decoration: underline;
        text-decoration-color: indigo;
    }

    .logo {
        font-size: 2rem;
        font-weight: 900;
        width: 800px;
    }

    .logo:hover {
        cursor: default;
    }

    .alert-success {
color: #155724; /* Text color */
background-color: #d4edda; /* Background color */
border-color: #c3e6cb; /* Border color */
padding: 15px; /* Padding */
border-radius: 5px; /* Border radius */
margin-bottom: 20px; /* Margin bottom */
}

.success-message {
    color: green;
    /* Set text color to green */
    font-weight: bold;
    /* Make the success message bold */
    font-size: 1em;
    /* Adjust font size as needed */
    /* Add any additional styling as needed */
}

</style>
</head>

```

```

<body>

    <nav id="desktop-nav">
        <div class="logo">LearnSphere</div>
        <div>
            <ul class="nav-links">
                <li><a href="/index">Logout</a></li>
            </ul>
        </div>
    </nav>

    <div class="welcome">
        <h2>Welcome Trainer</h2>
        <!--<h2 th:text="{user.username}"></h2>-->
    </div>
    <div th:if="{successMessage}" class="success-message">
<span th:text="{successMessage}"></span>
</div>

    <div class="options">
        <a href="/addCourse"><button>Add Course</button></a>
        <a href="/addLesson"><button>Add Lesson</button></a>
        <a href="/viewCourse"><button>View Courses</button></a>
    </div>
</body>

</html>

```

addCourse.html

```

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Add Course</title>
<style>

    body{
        font-family: Georgia, 'Times New Roman', Times, serif;
        height: 98vh;
        font-size: 20px;
        display: flex;
        flex-direction: column;
        justify-content: flex-start;
        align-items: center;
    }

```



```
background: linear-gradient(65deg, #FAF0D7 , #ADC4CE);
gap: 200px;
}
```

```
nav,.nav-links{
display: flex;
}
```

```
nav{
justify-content: space-around;
margin: 0px 45px 0px;
align-items: flex-start;
height: 10vh;
gap: 45rem;
}
```

```
.nav-links{
gap: 2rem;
list-style: none;
font-size: 1.5rem;
font-weight: 600;
}
```

```
a{
color: black;
text-decoration: none;
text-decoration-color: white;
}
```

```
a:hover{
color: black ;
text-decoration: underline;
text-underline-offset: 0.5rem;
text-decoration-color: indigo;
}
```

```
.logo{
font-size: 2rem;
font-weight: 900;
width: 800px;
}
```

```
.logo:hover{
cursor: default;
}
```

```
form{
position: relative;
height: 400px;
display: flex;
flex-direction: column;
```



```

        <li><a href="/index">Logout</a></li>
    </ul>
</div>
</nav>

    <form action="/addCourse" method="post">
        <div class="input-box">
            <label>Course ID:</label>
            <input type="text" name="courseId" pattern="[A-Za-z0-9]+" required>
        </div>
        <div class="input-box">
            <label>Course Name:</label>
            <input type="text" name="courseName" required>
        </div>
        <div class="input-box">
            <label>Course Price:</label>
            <input type="text" name="coursePrice" required>
        </div>
        <input type="submit" class="button" value="ADD COURSE">
    </form>
</body>
</html>

```

addLesson.html

```

<html>
<meta charset="ISO-8859-1">
<title>Add Lesson</title>
<style>

```

```

body{
    font-family: Georgia, 'Times New Roman', Times, serif;
    height: 98vh;
    font-size: 20px;
    display: flex;
    flex-direction: column;
    justify-content: flex-start;
    align-items: center;
    background: linear-gradient(65deg, #FAF0D7 , #ADC4CE);
    gap: 150px;
}

form{
    position: relative;
    height: 500px;
    display: flex;
    flex-direction: column;

```

```

        gap: 40px;
        justify-content: space-evenly;
        font-size: 30px;
    }

    .input-box{
        display: flex;
        flex-direction: row;
        justify-content: space-between;
        gap: 50px;
    }

    .input-box input{
        font-size: 30px;
        height: 35px;
        width: 400px;
        border: 1px solid black;
        border-radius: 3px;
        box-shadow: 0 0 5px 1px black;
    }

    .input-box label{
        font-weight: 700;
        text-decoration: underline;
        text-underline-offset: 5px;
    }

    .button{
        background: #FAF0D7;
        height: 45px;
        font-size: 20px;
        font-weight: 700;
        transition: 0.3s;
        cursor: pointer;
        border-radius: 5px;
    }

    .button:hover{
        box-shadow: 0 0 10px 2px black;
        transform: translateY(5px);
    }

    nav,
    .nav-links{
        display: flex;
    }

    nav{
        justify-content: space-around;
        margin: 0px 45px 0px;
    }

```

```

    align-items: flex-start;
    height: 10vh;
    gap: 45rem;
}

.nav-links{
    gap: 2rem;
    list-style: none;
    font-size: 1.5rem;
    font-weight: 600;
}
a{
    color: black;
    text-decoration: none;
    text-decoration-color: white;
}

a:hover{
    color: black ;
    text-decoration: underline;
    text-underline-offset: 0.5rem;
    text-decoration-color: indigo;
}

.logo{
    font-size: 2rem;
    font-weight: 900;
    width: 800px;
}

.logo:hover{
    cursor: default;
}

```

```

</style>
</head>
<body>

```

```

    <nav id="desktop-nav">
    <div class="logo">LearnSphere</div>
    <div>
        <ul class="nav-links">

            <li><a href="/index">Logout</a></li>
        </ul>
    </div>
    </nav>

    <form action="/addLesson" method="post">
        <div class="input-box">
            <label>Course ID:</label>

```

```

        <input type="text" name="course" pattern="[A-Za-z0-9]+" required>
    </div>
    <div class="input-box">
        <label>Lesson ID:</label>
        <input type="text" name="lessonId" pattern="[A-Za-z0-9]+" required>
    </div>
    <div class="input-box">
        <label>Lesson Name:</label>
        <input type="text" name="lessonName" required>
    </div>
    <div class="input-box">
        <label>Lesson Topics:</label>
        <input type="text" name="lessonTopics" required>
    </div>
    <div class="input-box">
        <label>Lesson Link:</label>
        <input type="text" name="lessonLink" required>
    </div>
    <input type="submit" class="button" value="ADD LESSON">
</form>
</body>
</html>

```

course.html

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Course List</title>
    <style>

        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f4f4f4;
        }

        table {
            width: 80%;
            margin: 20px auto;
            border-collapse: collapse;
            border: 1px solid #ddd;
            background-color: #fff;
        }
    </style>

```

```

th, td {
    padding: 12px 15px;
    text-align: left;
    border-bottom: 1px solid #ddd;
}

th {
    background-color: #f2f2f2;
}

tr:hover {
    background-color: #f2f2f2;
}
</style>
</head>
<body>
<table>
  <thead>
    <tr>
      <th scope="col">Course Id</th>
      <th scope="col">Course Name</th>
      <th scope="col">Course Price</th>

    </tr>
  </thead>
  <tbody>
    <tr th:each="course : ${courseList}">
      <td th:text="${course.courseId}"></td>
      <td th:text="${course.courseName}"></td>
      <td th:text="${course.coursePrice}"></td>
      <!-- Add lessons here if needed -->
    </tr>
  </tbody>
</table>
</body>
</html>

```

studentHome.html

```

<html>
<head>
<meta charset="ISO-8859-1">
<title>Student Home</title>
<style>

```

```

    body{
        font-family: Georgia, 'Times New Roman', Times, serif;

```

```

height: 98vh;
font-size: 20px;
display: flex;
flex-direction: column;
justify-content: flex-start;
align-items: center;
background: linear-gradient(65deg, #FAF0D7 , #ADC4CE);
gap: 50px;
}
.options{
    height: 50vh;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: center;
    gap: 35px;
}

.options button{
    height: 60px;
    width: 400px;
    border-radius: 10px;
    cursor: pointer;
    background-color: #FAF0D7;
    font-weight: 800;
    font-size: 20px;
    transition: 0.3s;
}

button:hover{
    box-shadow: 0 0 10px 2px black;
    transform: translateY(-3px);
}

.welcome{
    display: flex;
    justify-content: center;
    align-items: center;
    gap: 15px;
}

nav, .nav-links{
display: flex;
}

nav{
    justify-content: space-around;
    margin: 0px 45px 0px;
    align-items: flex-start;
    height: 2px;
    gap: 45rem;
}

```



```

}

.nav-links{
  gap: 2rem;
  list-style: none;
  font-size: 1.5rem;
  width: 300px;
  font-weight: 600;
}
a{
  color: black;
  text-decoration: none;
  text-decoration-color: white;
}

a:hover{
  color: black ;
  text-decoration: underline;
  text-underline-offset: 0.5rem;
  text-decoration-color: indigo;
}

.logo{
  font-size: 2rem;
  font-weight: 900;
  width: 800px;
}

.logo:hover{
  cursor: default;
}
</style>
</head>
<body>

    <nav id="desktop-nav">
    <div class="logo">LearnSphere</div>
    <div>
      <ul class="nav-links">
        <li><a href="/index">Logout</a></li>
      </ul>
    </div>
    </nav>

    <div class="welcome">
    <h2>Welcome Student</h2>
    <!--<h2 th:text="${user.username}"></h2>-->
    </div>
    <div class="options">
    <a href="/purchase"><button>Purchase Course</button></a>
    <a href="/mycourse"><button>My Courses</button></a>

```

```

    </div>
</body>
</html>

```

purchaseCourse.html

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1">
    <title>View Courses</title>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTWfSpd3yD65Vohhpuc0mLASjC"
crossorigin="anonymous">
    <script src="https://checkout.razorpay.com/v1/checkout.js"></script>
    <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
  </head>
  <body>

    <div>
      <!-- Your table -->
      <table class="table table-success table-striped table-bordered">
        <thead>
          <tr>
            <th scope="col">Course Id</th>
            <th scope="col">Course Name</th>
            <th scope="col">Course Price</th>
            <th scope="col">Buy Course</th> <!-- New column header -->
          </tr>
        </thead>
        <tbody>
          <tr th:each="course : ${courseList}">
            <td th:text="${course.courseId}"></td>
            <td th:text="${course.courseName}"></td>
            <td th:text="${course.coursePrice}"></td>

            <td>
              <form id="payment-form">
                <input type="hidden" class="email" th:value="${session.loggedInUser.email}"/>
                <input type="hidden" class="course-id" th:value="${course.courseId}"/>
                <button type="submit" id="pay-button" class="btn btn-success buy-button"
th:data-amount="${course.coursePrice}">BUY</button>
              </form>
            </td>
          </tr>
        </tbody>
      </table>
    </div>
  </body>
</html>

```

```

        </form>
    </td> <!-- New column with Buy button -->
</tr>
</tbody>

```

```

<br>

```

```

</table>
</div>

```

```

<a href="/studentHome">Go to Home</a>

```

```

<script>
$(document).ready(function() {
    $(".buy-button").click(function(e) {
        e.preventDefault();
        var form = $(this).closest('form');
        var amount = $(this).data("amount");
        var email = form.find('.email').val();
        var courseId = form.find('.course-id').val();
        createOrder(amount, email, courseId);
    });
});

function createOrder(amount, email, courseId) {
    alert(amount+email+courseId)
    $.post("/takeOrder", { amount: amount, email: email, courseId: courseId })
    .done(function(order) {
        order = JSON.parse(order);
        var options = {
            "key": "rzp_test_kTMXY8IGNvKH0",
            "amount": order.amount_due.toString(),
            "currency": "INR",
            "name": "Learn Sphere",
            "description": "Test Transaction",
            "order_id": order.id,
            "handler": function (response) {
                verifyPayment(response.razorpay_order_id, response.razorpay_payment_id,
response.razorpay_signature);
            },
            "prefill": {
                "name": "Your Name",
                "email": "test@example.com",
                "contact": "9999999999"
            },
            "notes": {
                "address": "Your Address"
            },
            "theme": {

```

```

        "color": "#F37254"
    }
};
var rzp1 = new Razorpay(options);
rzp1.open();
})
.fail(function(error) {
    console.error("Error:", error);
});
}

function verifyPayment(orderId, paymentId, signature) {
    $.post("/verify", { orderId: orderId, paymentId: paymentId, signature: signature })
        .done(function(isValid) {
            if (isValid) {
                console.log("Payment successful");
            } else {
                console.log("Payment failed");
            }
        })
        .fail(function(error) {
            console.error("Error:", error);
        });
}
</script>
</html>

```

viewCourse.html

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Courses</title>
    <style>

    body {
        font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
        background: linear-gradient(180deg, #FAF0D7, #ADC4CE);
        height: 100vh;
        font-size: 20px;
        display: flex;
        flex-direction: column;
        justify-content: flex-start;
        align-items: center;
        margin: 0;
    }

```

```
table {  
  width: 80%;  
  margin-top: 30px;  
  border-collapse: collapse;  
  background-color: #fff;  
  box-shadow: 0 0 10px 2px rgba(0, 0, 0, 0.2);  
  border-radius: 10px;  
  overflow: hidden;  
}
```

```
th, td {  
  padding: 15px;  
  border-bottom: 1px solid #ddd;  
  text-align: center;  
}
```

```
th {  
  background-color: #ADC4CE;  
  color: white;  
}
```

```
td {  
  background-color: #FAF0D7;  
}
```

```
nav, .nav-links {  
  display: flex;  
  justify-content: space-between;  
  align-items: center;  
  margin: 20px;  
}
```

```
.nav-links {  
  gap: 2rem;  
  list-style: none;  
  font-size: 1.5rem;  
}
```

```
a {  
  color: black;  
  text-decoration: none;  
  font-weight: 600;  
  transition: color 0.3s ease-in-out;  
}
```

```
a:hover {  
  color: indigo;  
}
```

```
.logo {
```

```

    font-size: 2rem;
    font-weight: 900;
  }

  .logo:hover {
    cursor: default;
  }
</style>
</head>
<body>

<nav id="desktop-nav">
  <div class="logo">LearnSphere</div>
  <div>
    <ul class="nav-links">
      <li><a href="/index">Logout</a></li>
    </ul>
  </div>
</nav>

<table border="3">
  <thead>
    <tr>
      <th>Course ID</th>
      <th>Course Name</th>
      <th>Course Price</th>
    </tr>
  </thead>
  <tbody>
    <tr th:each="course:${cList}">
      <td th:text="${course.courseId}"></td>
      <td th:text="${course.courseName}"></td>
      <td th:text="${course.coursePrice}"></td>
    </tr>
  </tbody>
</table>
</body>
</html>

```

myCourses.html

```

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>My Courses</title>
<style>

```

```

body{
  font-family: Georgia, 'Times New Roman', Times, serif;
  height: 98vh;
  font-size: 20px;
  display: flex;
  flex-direction: column;
  justify-content: flex-start;
  align-items: center;
  background: linear-gradient(65deg, #FAF0D7 , #ADC4CE);
  gap: 300px;
}
table{
  width: 700px;
  background-color: antiquewhite;
  font-size: 20px;
  text-align: center;
  box-shadow: 0 0 10px 2px black;
}
table tr{
  height: 40px;
}

```

```

nav,
.nav-links{
  display: flex;
}

```

```

nav{
  justify-content: space-around;
  margin: 0px 45px 0px;
  align-items: flex-start;
  height: 2px;
  gap: 45rem;
}

```

```

.nav-links{
  gap: 2rem;
  list-style: none;
  font-size: 1.5rem;
  width: 300px;
  font-weight: 600;
}

```

```

a{
  color: black;
  text-decoration: none;
  text-decoration-color: white;
}

```

```

a:hover{

```

```

color: black ;
text-decoration: underline;
text-underline-offset: 0.5rem;
text-decoration-color: indigo;
}

```

```

.logo{
font-size: 2rem;
font-weight: 900;
width: 800px;
}

```

```

.logo:hover{
cursor: default;
}

```

```

</style>
</head>
<body>

```

```

    <nav id="desktop-nav">
    <div class="logo">LearnSphere</div>
    <div>
    <ul class="nav-links">
    <li><a href="/purchase">Purchase Courses</a></li>
    <li><a href="/index">Logout</a></li>
    </ul>
    </div>
    </nav>

```

```

<table class="table table-success table-striped table-bordered">
<thead>
<tr>
<th scope="col">Course Id</th>
<th scope="col">Course Name</th>
<th scope="col">Course Price</th>
<th scope="col">Lessons</th>
</tr>
</thead>
<tbody>
<tr th:each="course:${courseList}">
<td th:text="${course.courseId}"></td>
<td th:text="${course.courseName}"></td>
<td th:text="${course.coursePrice}"></td>
<td>
<span th:each="lesson, lessonStat : ${course.lessonList}">
    <a th:href="@{/viewLesson(lessonId=${lesson.lessonId})}"
th:text="${lesson.lessonName}"></a><span th:if="${!lessonStat.last}">, </span>
</span>

```



```

    </td>
  </tr>
</tbody>
</table>

```

```

</body>
</html>

```

myLesson.html

```

<!DOCTYPE html>
<html xmlns:th="https://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Lesson</title>
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

<style>
  body{
    font-family: Georgia, 'Times New Roman', Times, serif;
    height: 98vh;
    font-size: 20px;
    display: flex;
    flex-direction: column;
    justify-content: flex-start;
    align-items: center;
    background: linear-gradient(65deg, #FAF0D7 , #ADC4CE);
    gap: 100px;
  }

  .lesson-table{
    width: 1500px;
    background-color: antiquewhite;
    font-size: 20px;
    text-align: center;
  }

  .lesson-table td, .lesson-table th{
    border-bottom: 2px solid black;
  }

  .lesson-table tr{
    height: 50px;
  }

```

```

        .lesson-video{
            height: 550px;
            display: flex;
            justify-content: center;
        }

        .lesson-link{
            width: 900px;
        }

        nav,
        .nav-links{
            display: flex;
        }

        nav{
            justify-content: space-around;
            margin: 0px 45px 0px;
            align-items: flex-start;
            height: 1px;
            gap: 45rem;
        }

        .nav-links{
            gap: 2rem;
            list-style: none;
            font-size: 1.5rem;
            font-weight: 600;
        }
        a{
            color: black;
            text-decoration: none;
            text-decoration-color: white;
        }

        a:hover{
            color: black;
            text-decoration: underline;
            text-decoration-color: indigo;
        }

        .logo{
            font-size: 2rem;
            font-weight: 900;
            width: 800px;
        }

        .logo:hover{

```

```

    cursor: default;
}

</style>

</head>
<body>

    <nav id="desktop-nav">
    <div class="logo">Learn Sphere</div>
    <div>
        <ul class="nav-links">
            <li><a href="/mycourse">My Courses</a></li>
            <li><a href="/index">Logout</a></li>
        </ul>
    </div>
    </nav>

    <div class="container">
    <h2 class="mb-4">Lesson Details:</h2>
    <table class="lesson-table">
        <thead>
            <tr>
                <th scope="col">Lesson Id</th>
                <th scope="col">Lesson Name</th>
                <th scope="col">Lesson Topics</th>

            </tr>
        </thead>
        <tbody>
            <tr>
                <td th:text="{lesson.lessonId}"></td>
                <td th:text="{lesson.lessonName}"></td>
                <td th:text="{lesson.lessonTopics}"></td>

            </tr>
        </tbody>
    </table>

    <h2 class="mt-4 mb-4">Lesson Video:</h2>
    <div class="embed-responsive embed-responsive-16by9">
        <iframe class="embed-responsive-item"
            th:src="'https://www.youtube.com/embed/' + {lesson.lessonLink}"
            title="YouTube video player" frameborder="0"
            allow="accelerometer; autoplay; clipboard-write;
            encrypted-media; gyroscope; picture-in-picture"
            allowfullscreen>
        </iframe>
    </div>

```

```
</body>
</html>
```

NavController.java

```
package com.learnSphere.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class NavController {

    @GetMapping("/index")
    public String index() {
        return "index";
    }

    @GetMapping("/register")
    public String register() {
        return "register";
    }

    @GetMapping("/login")
    public String login() {
        return "login";
    }

    @GetMapping("/addCourse")
    public String addCourse() {
        return "addCourse";
    }

    @GetMapping("/addLesson")
    public String addLesson()
    {
        return "addLesson";
    }
    @GetMapping("/studentHome")
    public String studentHome()
    {
        return "studentHome";
    }
}
```

UsersController.java

```
package com.learnSphere.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import com.learnSphere.entities.Users;
import com.learnSphere.services.UserService;

import jakarta.servlet.http.HttpSession;

@Controller
public class UsersController {

    @Autowired
    UserService uservice;
    @PostMapping("/addUser")
    public String addUser(@ModelAttribute Users user, Model model) {
        String email=user.getEmail();
        boolean isPresent=uservice.checkEmail(email);

        if(isPresent==false) {
            uservice.addUser(user);
            return "login";
        }
        else {
            model.addAttribute("errorMessage", "Email already exists");
            return "register";
        }
    }

    @PostMapping("/validateUser")
    public String validateUser(@RequestParam("email") String email,
                              @RequestParam("password") String password,HttpSession session,
Model model)
    {

        Users user=uservice.findUserByEmail(email);
        String dbPassword=user.getPassword();
        String role=user.getRole();

        if(password.equals(dbPassword)) {
            session.setAttribute("loggedInUser", user);
            model.addAttribute("user", user);
        }
    }
}
```

```

        if(role.equals("trainer")) {
            return "trainerHome";
        }
        else {
            return "studentHome";
        }
    }
    else {
        String errorMessage = "Invalid username or password";
        model.addAttribute("errorMessage", errorMessage);
        return "login";
    }
}

}
}
}

```

TrainerController.java

```

package com.learnSphere.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import com.learnSphere.entities.Course;
import com.learnSphere.entities.Lesson;
import com.learnSphere.services.TrainerService;

@Controller
public class TrainerController {

    @Autowired
    TrainerService tservice;

    @PostMapping("/addCourse")
    public String addCourse(@ModelAttribute Course course) {
        tservice.addCourse(course);
        return "trainerHome";
    }
}

```

```
}
```

```
@GetMapping("/viewCourse")
public String viewCourse(Model model) {
    List<Course> courseList=tservice.fetchAllCourse();
    model.addAttribute("courseList",courseList);
    return "course";
}
```

```
@PostMapping("/addLesson")
public String addLesson(@ModelAttribute Lesson lesson)
{
    System.out.println(lesson);
    Course course=lesson.getCourse();
    tservice.addLesson(lesson);
    course.getLessonList().add(lesson);
    tservice.saveCourse(course);
```

```
    return "trainerHome";
}
```

```
}
```

StudentController.java

```
package com.learnSphere.controller;
```

```
import java.util.List;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
```

```
import com.learnSphere.entities.Course;
import com.learnSphere.entities.Lesson;
import com.learnSphere.entities.Users;
import com.learnSphere.services.StudentService;
import com.learnSphere.services.UsersService;
```

```
import jakarta.servlet.http.HttpSession;
```

```
@Controller
public class StudentController {
    @Autowired
    StudentService sservice;
```

```

@Autowired
UserService uservice;
@GetMapping("/purchase")
public String purchase(Model model) {
    //fetch course list
    List<Course> courseList=sservice.fetchCourseList();
    //add course list to model object
    model.addAttribute("courseList",courseList);
    return "purchaseCourse";
}
@GetMapping("/mycourse")
public String myCourses(Model model, HttpSession session) {
    Users loggedUser=(Users)session.getAttribute("loggedInUser");

    String email=loggedUser.getEmail();

    Users user=uservice.findUserByEmail(email);

    List<Course>courseList=user.getCourseList();
    model.addAttribute("courseList",courseList);
    return "myCourses";
}
@GetMapping("/viewLesson")
    public String viewLesson(@RequestParam("lessonId") int lessonId,
Model model,HttpSession session) {

        Lesson lesson =sservice.getLesson(lessonId);
        //Extract the youtube video id from url
        String youtubeUrl = lesson.getLessonLink();

        String videoId = youtubeUrl.substring(youtubeUrl.indexOf("/") + 1);
        lesson.setLessonLink(videoId);

        model.addAttribute("lesson",lesson);

        return "myLesson";
    }
}

```

orderCreation.java

```

package com.learnSphere.controller;

import org.json.JSONObject;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

```



```

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

import com.learnSphere.entities.Course;
import com.learnSphere.entities.Users;
import com.learnSphere.services.StudentService;
import com.learnSphere.services.TrainerService;
import com.learnSphere.services.UsersService;
import com.razorpay.Order;
import com.razorpay.RazorpayClient;
import com.razorpay.RazorpayException;
@Controller
public class orderCreation {
    @Autowired
    UsersService userservice;
    @Autowired
    StudentService sservice;
    @Autowired
    TrainerService tservice;

    @PostMapping("/takeOrder")
    @ResponseBody
    public String takeorder(@RequestParam int amount,String email,int courseId) {
        mapCourseAndUser(email,courseId); //modified
        Order order =null;

        try {
            RazorpayClient razorpay=new
RazorpayClient("rzp_test_kTMXY8IGNvKHZ0","F5XoKNe4KIDkp9SWanBqzb7o");
            JSONObject orderRequest = new JSONObject();
            orderRequest.put("amount", amount*100); // amount in the smallest
currency unit

            orderRequest.put("currency", "INR");
            orderRequest.put("receipt", "order_rcptid_11");

            order = razorpay.orders.create(orderRequest);
        } catch (RazorpayException e) {
            // Handle Exception
            System.out.println(e.getMessage());
        }
        return order.toString();
    }
    public void mapCourseAndUser(String email,int courseId) {
        Users user=userservice.findUserByEmail(email);
        Course course=sservice.fetchCourse(courseId);

        user.getCourseList().add(course);
        course.getUserList().add(user);

        tservice.saveCourse(course);
        userservice.saveUsers(user);
    }
}

```

```
}  
}
```

Users.java

```
package com.learnSphere.entities;
```

```
import java.util.List;
```

```
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
import jakarta.persistence.ManyToMany;
```

```
@Entity
```

```
public class Users {
```

```
    @Id
```

```
    @GeneratedValue(strategy=GenerationType.AUTO)
```

```
    int id;
```

```
    String username;
```

```
    String email;
```

```
    String password;
```

```
    String role;
```

```
    @ManyToMany
```

```
    List<Course> courseList;
```

```
    public Users() {
```

```
        super();
```

```
        // TODO Auto-generated constructor stub
```

```
    }
```

```
    public Users(int id, String username, String email, String password, String role,  
List<Course> courseList) {
```

```
        super();
```

```
        this.id = id;
```

```
        this.username = username;
```

```
        this.email = email;
```

```
        this.password = password;
```

```
        this.role = role;
```

```
        this.courseList = courseList;
```

```
    }
```

```
    public int getId() {
```

```
        return id;
```

```
    }
```

```
    public void setId(int id) {
```

```

        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
        this.role = role;
    }

    public List<Course> getCourseList() {
        return courseList;
    }

    public void setCourseList(List<Course> courseList) {
        this.courseList = courseList;
    }

    @Override
    public String toString() {
        return "Users [id=" + id + ", username=" + username + ", email=" + email + ",
password=" + password + ", role="
            + role + ", courseList=" + courseList + "]";
    }

```

```
}
```

Course.java

```
package com.learnSphere.entities;

import java.util.List;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.ManyToMany;
import jakarta.persistence.OneToMany;

@Entity
public class Course {

    @Id
    int courseId;
    String courseName;
    int coursePrice;

    @OneToMany
    List<Lesson> lessonList;

    @ManyToMany
    List<Users> userList;

    public Course() {
        super();
        // TODO Auto-generated constructor stub
    }

    public Course(int courseId, String courseName, int coursePrice, List<Lesson>
lessonList, List<Users> userList) {
        super();
        this.courseId = courseId;
        this.courseName = courseName;
        this.coursePrice = coursePrice;
        this.lessonList = lessonList;
        this.userList = userList;
    }

    public int getCourseId() {
        return courseId;
    }

    public void setCourseId(int courseId) {
        this.courseId = courseId;
    }

    public String getCourseName() {
        return courseName;
    }
}
```

```

    }

    public void setCourseName(String courseName) {
        this.courseName = courseName;
    }

    public int getCoursePrice() {
        return coursePrice;
    }

    public void setCoursePrice(int coursePrice) {
        this.coursePrice = coursePrice;
    }

    public List<Lesson> getLessonList() {
        return lessonList;
    }

    public void setLessonList(List<Lesson> lessonList) {
        this.lessonList = lessonList;
    }

    public List<Users> getUserList() {
        return userList;
    }

    public void setUserList(List<Users> userList) {
        this.userList = userList;
    }

    @Override
    public String toString() {
        return "Course [courseId=" + courseId + ", courseName=" + courseName + ",
coursePrice=" + coursePrice
                + ", lessonList=" + lessonList + ", userList=" + userList + "];"
    }

    public void addLesson(Lesson lesson)
    {
        lessonList.add(lesson);
    }
}

```

Lesson.java

```

package com.learnSphere.entities;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;

```

```
import jakarta.persistence.ManyToOne;
```

```
@Entity
```

```
public class Lesson {
```

```
    @Id
```

```
    int lessonId;
```

```
    String lessonName;
```

```
    String lessonTopics;
```

```
    String lessonLink;
```

```
    @ManyToOne
```

```
    Course course;
```

```
    public Lesson() {
```

```
        super();
```

```
        // TODO Auto-generated constructor stub
```

```
    }
```

```
    public Lesson(int lessonId, String lessonName, String lessonTopics, String lessonLink,  
Course course) {
```

```
        super();
```

```
        this.lessonId = lessonId;
```

```
        this.lessonName = lessonName;
```

```
        this.lessonTopics = lessonTopics;
```

```
        this.lessonLink = lessonLink;
```

```
        this.course = course;
```

```
    }
```

```
    public int getLessonId() {
```

```
        return lessonId;
```

```
    }
```

```
    public void setLessonId(int lessonId) {
```

```
        this.lessonId = lessonId;
```

```
    }
```

```
    public String getLessonName() {
```

```
        return lessonName;
```

```
    }
```

```
    public void setLessonName(String lessonName) {
```

```
        this.lessonName = lessonName;
```

```
    }
```

```
    public String getLessonTopics() {
```

```
        return lessonTopics;
```

```
    }
```

```
    public void setLessonTopics(String lessonTopics) {
```

```
        this.lessonTopics = lessonTopics;
```

```

    }

    public String getLessonLink() {
        return lessonLink;
    }

    public void setLessonLink(String lessonLink) {
        this.lessonLink = lessonLink;
    }

    public Course getCourse() {
        return course;
    }

    public void setCourse(Course course) {
        this.course = course;
    }

    @Override
    public String toString() {
        return "Lesson [lessonId=" + lessonId + ", lessonName=" + lessonName + ",
lessonTopics=" + lessonTopics
                + ", lessonLink=" + lessonLink + ", course=" + course + "]";
    }
}

```

UsersRepository.java

```

package com.learnSphere.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.learnSphere.entities.Users;

public interface UsersRepository extends JpaRepository<Users, Integer>{
    Users findByEmail(String email); //providing signature because system knows only
    findById.
    boolean existsByEmail(String email);
}

```

CourseRepository.java

```

package com.learnSphere.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.learnSphere.entities.Course;

public interface CourseRepository extends JpaRepository<Course, Integer>{
    Course findById(int courseId);
}

```

```
}
```

LessonRepository.java

```
package com.learnSphere.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.learnSphere.entities.Lesson;

public interface LessonRepository extends JpaRepository<Lesson, Integer>{

    Lesson findByLessonId(int lessonId);

}
```

UsersService.java

```
package com.learnSphere.services;

import com.learnSphere.entities.Users;

public interface UsersService {
    String addUser(Users user);
    Users findUserByEmail(String email);
    boolean checkEmail(String email);
    String saveUsers(Users user);
}
```

UsersServiceImplementation.java

```
package com.learnSphere.services;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.learnSphere.entities.Users;
import com.learnSphere.repository.UsersRepository;

@Service
public class UsersServiceImplementation implements UsersService{

    @Autowired
    UsersRepository repo;

    @Override
    public String addUser(Users user) {
        repo.save(user);
        return "User added !";
    }

    @Override
```



```

        public Users findUserByEmail(String email) {
            return repo.findByEmail(email); // you have to provide signatures
        }

        @Override
        public boolean checkEmail(String email) {

            return repo.existsByEmail(email);
        }
        @Override
        public String saveUsers(Users user) {
            repo.save(user);
            return "user updated";
        }
    }
}

```

StudentService.java

```

package com.learnSphere.services;

import java.util.List;

import com.learnSphere.entities.Course;
import com.learnSphere.entities.Lesson;

public interface StudentService {
    List<Course> fetchCourseList();
    Course fetchCourse(int courseId);

    Lesson getLesson(int lessonId);

}

```

StudentServiceImplementation.java

```

package com.learnSphere.services;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.learnSphere.entities.Course;
import com.learnSphere.entities.Lesson;
import com.learnSphere.repository.CourseRepository;
import com.learnSphere.repository.LessonRepository;

@Service
public class StudentServiceImplementation implements StudentService{
    @Autowired
    CourseRepository repo;
}

```

```

    @Autowired
    LessonRepository lrepo;
    @Override

    public List<Course> fetchCourseList(){
        return repo.findAll();
    }

    public Course fetchCourse(int courseId) {
        return repo.findById(courseId);
    }

    @Override
    public Lesson getLesson(int lessonId) {
        // TODO Auto-generated method stub
        return lrepo.findById(lessonId);
    }
}

TrainerService.java

```

```

package com.learnSphere.services;

import java.util.List;

import com.learnSphere.entities.Course;
import com.learnSphere.entities.Lesson;

public interface TrainerService {

    String addCourse(Course course);
    List<Course> fetchAllCourse();
    String addLesson(Lesson lesson);
    String saveCourse(Course course);
}

```

```

TrainerServiceImplementation.java

package com.learnSphere.services;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.learnSphere.entities.Course;

```

```

import com.learnSphere.entities.Lesson;
import com.learnSphere.repository.CourseRepository;
import com.learnSphere.repository.LessonRepository;

@Service
public class TrainerServiceImpl implements TrainerService{

    @Autowired
    CourseRepository repo;
    @Autowired
    LessonRepository lrepo;

    @Override
    public String addCourse(Course course) {
        repo.save(course);
        return "course added";
    }

    @Override
    public List<Course> fetchAllCourse()
    {
        return repo.findAll();
    }

    @Override
    public String addLesson(Lesson lesson) {
        lrepo.save(lesson);
        return "lesson added";
    }

    @Override
    public String saveCourse(Course course) {
        repo.save(course);
        return "course updated";
    }
}

```

Application.java

```

package com.learnSphere;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

```

```
    public static void main(String[] args) {  
        SpringApplication.run(Application.class, args);  
    }  
}
```

application.properties

server.port=9967

spring.datasource.url:jdbc:mysql://localhost/learnSphereProject

spring.datasource.username:root

spring.datasource.password:root

spring.jpa.hibernate.ddl-auto:update

5 SYSTEM TESTING

5.1 TESTING METHODS

In a software development project, errors can be injected at any stage during the development. Testing performs a very critical role for quality and for ensuring the reliability of software. During testing, the program to be tested is executed with set of test cases, and the output of the program for the test cases is evaluated to determine if the program is performing as it is expected to. Due to its approach, dynamic testing can only ascertain the presence of error in the program; the exact nature of the error is not usually decided by testing. Testing forms the first step in determining the errors in the program. Clearly the success of testing in revealing errors in programs depends critically on the test cases.

Testing is usually relied upon to detect the faults that occur during any phase of the software development cycle, in addition to the faults that introduced during the coding phase itself. For this, different levels of testing are used which perform different tasks and aim to test different VBects of the system. the basic

levels of testing are unit testing, integration testing, system and acceptance testing. the different levels of testing attempt to detect different types of faults.

The main objective of testing is to uncover errors from the system. For the uncovering process we have to give proper input data to the system. So we should have more conscious to give input data. It is important to give correct inputs to efficient testing.

5.2 Test Plan:

Testing is the process of exercising software with the intent of finding errors. The Web-app testing is a collection of related activities with a single goal: to uncover errors in web application content, function, usability, navigability, performance, capacity and security.

There are several areas of testing involved in web applications. For the current web application, I used some of them as follows.

CONTENT TESTING

Content testing attempts to uncover errors in content of the web application. In addition to examining static content for errors, this testing step also considers dynamic content derived from data maintained as a part of database system that has been integrated with the web application.

Content testing of all web pages is evaluated for syntactic and semantic errors.

At the syntactic level, I have verified the content for spelling, punctuation, and any grammar mistakes on all pages which contain the content of the website.

At semantic level I have verified for the following aspects.

Whether the content is valid or not.

Whether the format of the content is good and readable or not.

Whether all the web pages are showing consistent content or not.

The content includes the dynamic information about the companies, stock values and flowchart details which is fetched from the database. The consistency of this information is thoroughly tested.

DATABASE TESTING

Database testing is done to uncover the errors which occur as a consequence of fetching large equities of data from the database, extracting relevant data from the database, accessing the database using several queries etc,

In this project, I have tested the application for database errors in following areas.

While converting the user request into a database query

While fetching dynamic content to the web pages.

While opening and closing the active connections to the database

While presenting the raw data fetched from database in a formatted HTML output.

Communication between the web application and the remote database.

USER INTERFACE TESTING

All the interfaces that have been designed are reviewed whether they meet the customer requirement or not. While testing all interfaces I have verified for errors as follows.

Errors related to specific interface mechanisms for example proper execution of all menu links that are provided in each web page

Errors related to all semantics of navigation and web application functionally that is provided in each web page.

Errors in consistency related to different aspects of the interfaces like font style, color, size, screen background color etc.,

Errors in viewing the interfaces in different web browsers like Microsoft internet explorer, Mozilla firefox etc.,

INTERFACE MECHANISM TESTING

When a user interacts with a web application, the interaction occurs through one or more mechanisms which are called interface mechanisms. Testing done within these mechanisms is the interface mechanism testing. This testing is done in following areas.

Links:

Each navigation link is tested to ensure that appropriate web page is linked or not. I have listed all the links in each form to test whether each link is connecting the appropriate page or not.

Forms:

Testing forms has been done at two different levels i.e. at minimum level and at more targeted level

Whether labels been correctly defined for fields or not.

Whether server is receiving all the information contained in the form and no data are lost in the transmission between client and server.

Whether appropriate default values are available when the user does not select any item in the selection box.

Whether scripts that perform data validation from the client-side are working properly or not.

At more targeted level I have tested for:

Whether text fields have proper width to enter data.

Whether text fields are allowing string length more than specified length.

Whether tab order among different controls is in required order or not.

Client Side Scripting:

Each and every function written in scripting has been tested by Black Box Testing.

I have combined the forms testing with this client-side script testing, because input for scripting is provided from forms. Some methods of scripting will be performed in some particular browsers and in others not. So I have also performed compatibility testing to ensure that the scripting functions will work properly in all browsers.

USABILITY TESTS

In this testing I have verified up to, which level that, users can interact with the system effectively. Tests are designed to determine the degree to which the web application interface makes users easy to work with. I have designed test case so that usability testing can be verified at different levels:

Usability test has been performed on each and every individual interface i.e. forms.

Usability test has been performed on total web page with related client side scripting functions.

Usability test has been performed on total web application.

COMPATIBILITY TESTS

As this is a web application, it should run on different environments like different computer architectures, operating systems, browsers, and network connection speeds.

As different computing configurations can result in differences in client-side scripting speeds and display resolution, operating system variance may cause web application processing issues.

Different browsers produce slightly different results than we expected, in some cases these results may not be a problem but in some cases, there will be serious errors.

To perform these testing strategies first we have prepared what are all the client-side functions that encounter problems with different compatibilities. In essence of those we have tested by identifying different computing platform, typical display devices, the operating systems supported on the platform, the browsers that are available with me.

NAVIGATION TESTING

Navigability is tested to ensure that all navigation syntax and semantics are exercised to uncover any navigation errors. (ex: dead links, improper links, erroneous links). The job of navigation testing is to ensure that the navigation mechanisms are functional, and to validate that each Navigation Semantic Unit can be achieved by the appropriate user category.

We have done the navigation testing in following areas.

Navigation links are thoroughly tested.

Redirects are properly checked.

Is the target page to a navigation link is correct or not.

Is the link caption meaningful or not.

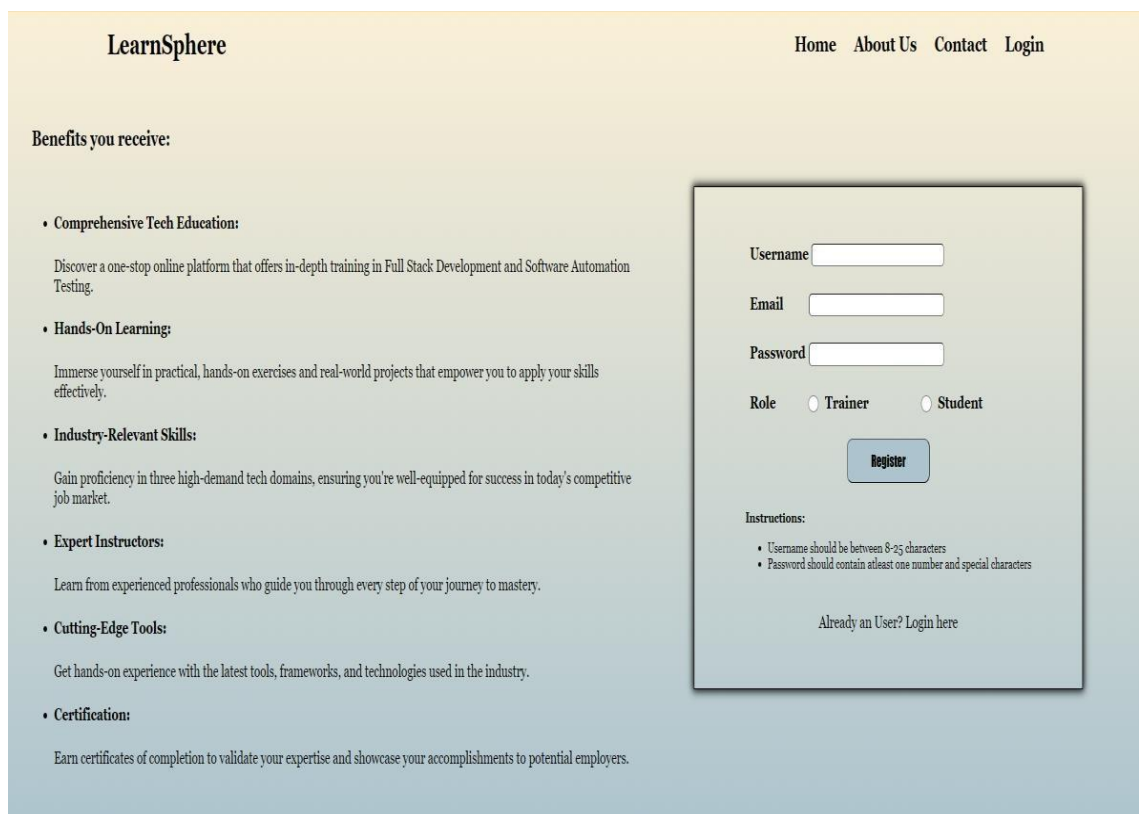
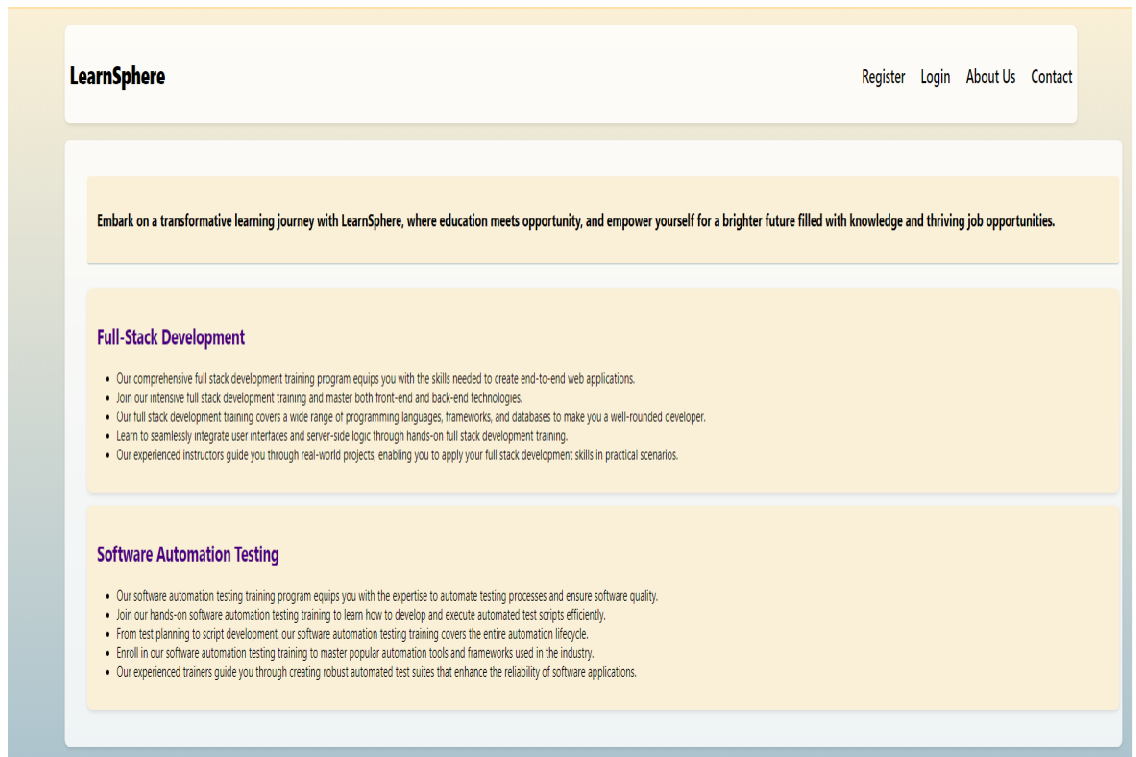
DEPLOYMENT:

The error-free project, which passed all the tests, is now deployed at the client environment in this phase.

5.3 TESTING OBJECTIVE:

Testing is the process of executing a program with the intention of finding an error. A good test case is one that has a high probability of finding an as-yet undiscovered error. A successful test is that in which no error are found. The objective is to design tests that systematically **uncover different classes of error and do with a minimum amount of time and effort.**

5.4 SCREEN LAYOUTS



Benefits you receive:**• Comprehensive Tech Education:**

Discover a one-stop online platform that offers in-depth training in Full Stack Development and Software Automation Testing.

• Hands-On Learning:

Immerse yourself in practical, hands-on exercises and real-world projects that empower you to apply your skills effectively.

• Industry-Relevant Skills:

Gain proficiency in three high-demand tech domains, ensuring you're well-equipped for success in today's competitive job market.

• Expert Instructors:

Learn from experienced professionals who guide you through every step of your journey to mastery.

• Cutting-Edge Tools:

Get hands-on experience with the latest tools, frameworks, and technologies used in the industry.

• Certification:

Earn certificates of completion to validate your expertise and showcase your accomplishments to potential employers.

Username Email Password Role ☐ Trainer ☐ Student**Instructions:**

- Username should be between 8-25 characters
- Password should contain atleast one number and special characters

[Already an User? Login here](#)Email Password [Not an User? Register here](#)

LearnSphere

[Home](#) [Services](#) [About Us](#) [Contact](#) [Register](#)

Email

Password

Login

Invalid username or password

Not an User? Register here

LearnSphere

Logout

Welcome Trainer

Add Course

Add Lesson

View Courses

LearnSphere

Logout

Course ID:

Course Name:

Course Price:

ADD COURSE

LearnSphere

Logout

Course ID:

104

Course Name:

Python

Course Price:

15999

ADD COURSE

LearnSphere

Logout

Course ID:

Lesson ID:

Lesson Name:

Lesson Topics:

Lesson Link:

ADD LESSON

Course ID:

Lesson ID:

Lesson Name:

Lesson Topics:

Lesson Link:

ADD LESSON

| Course Id | Course Name | Course Price |
|-----------|-------------|--------------|
| 101 | Java | 25000 |
| 102 | SQL | 1999 |
| 103 | BCom | 16000 |
| 104 | Python | 15999 |

Welcome Student

Purchase Course

My Courses


| Course Id | Course Name | Course Price | Buy Course |
|-----------|-------------|--------------|---------------------|
| 101 | Java | 25999 | BUY |
| 102 | SQL | 1999 | BUY |
| 103 | BCom | 15999 | BUY |
| 104 | Python | 15999 | BUY |

[Go to Home](#)

| Course Id | Course Name | Course Price | Buy Course |
|-----------|-------------|--------------|---------------------|
| 101 | Java | 25999 | BUY |
| 102 | SQL | | BUY |
| 103 | BCom | | BUY |
| 104 | Python | | BUY |

Learn Sphere

Pay With UPI QR



Scan the QR using any UPI app on your phone.

UPI Code is valid for 11:39 minutes

Pay With UPI ID/ Mobile Number

UPI ID/ Mobile Number

Enter UPI ID/ Mobile Number

999999999@yb

₹ 25,999

View Details

Pay Now

| Course Id | Course Name | Course Price | Buy Course |
|-----------|-------------|--------------|---------------------|
| 101 | Java | 25999 | BUY |
| 102 | SQL | | BUY |
| 103 | BCom | | BUY |
| 104 | Python | | BUY |

✓

Payment successful

₹ 1,999

Learn Sphere

Mar 18, 2024 | 01:21 AM

UPI | pay_NnXnKJ2nMkaRlo

© Visit razorpay.com/support for queries

Redirecting in 5 seconds


LearnSphere
Purchase Courses
Logout

| Course Id | Course Name | Course Price | Lessons |
|-----------|-------------|--------------|-------------|
| 102 | SQL | 1999 | Constraints |
| 104 | Python | 15000 | Strings |

Learn Sphere
My Courses
Logout

Lesson Details:

| Lesson Id | Lesson Name | Lesson Topics |
|-----------|-------------|-------------------------------|
| 1041 | Strings | Strings Slicing and Operation |

Lesson Video:


6. SYSTEM IMPLEMENTATION

Implementation is the process of bringing a developed system into operational use and turn it over to the user. Implementation activities extend from planning through conversion from the old system to the new.

6.1 The Implementation Plan:

To implement the Learning Management System successfully, the process begins with a comprehensive needs, understanding the specific requirements and challenges of educators

and learners in the online learning landscape. Stakeholder engagement is crucial, involving educators, administrators, and potential users to gather feedback and ensure collaboration. The platform is then customized to align with the unique needs of the institution, emphasizing user-friendly interfaces and tailored design. Training and onboarding sessions are developed for educators and learners, with a focus on efficient course management and progress tracking. Content migration is facilitated, and rigorous testing and quality assurance measures are implemented to guarantee a stable platform. The rollout strategy involves a phased approach, possibly starting with a pilot phase to gather feedback and make adjustments. A robust communication plan keeps stakeholders informed throughout the process. Ongoing monitoring, evaluation, and scalability planning are essential, anticipating future growth and ensuring the platform's security. Integration with existing systems is carefully addressed to optimize interoperability. This holistic approach ensures a smooth implementation of the Learning Management System, maximizing its effectiveness in facilitating seamless online learning experiences for educators and learners alike.

7. CONCLUSION AND SCOPE FOR FUTURE ENHANCEMENT

7.1 CONCLUSION:

Learning Management System emerges as an exemplary solution in the realm of online learning, offering a robust Learning Management System (LMS) that prioritizes user experience and efficient course management. The platform's user-friendly interface stands out as a beacon, ensuring accessibility for both educators and learners and fostering a positive and engaging online learning environment. The efficiency in course management provided by the Learning Management System allows educators to channel their efforts more substantively into teaching, enhancing the overall educational experience. However, it is crucial to acknowledge potential challenges, particularly in regions with issues related to internet connectivity. To ensure the widespread success of the Learning Management System, a proactive approach to addressing these challenges and implementing comprehensive user training becomes paramount. By navigating these considerations effectively, the Learning Management System has the potential to not only facilitate seamless online learning experiences but also contribute significantly to the evolution of education in diverse settings. Its strengths in accessibility, efficiency, and progress tracking position the Learning Management System as a valuable asset in the ever-evolving landscape of digital education.

5.2 FUTURE ENHANCEMENT

Learning Management System (LMS), has a promising future with potential enhancements across various fronts. The platform can evolve by integrating emerging technologies like virtual and augmented reality, incorporating artificial intelligence for personalized learning, and optimizing for mobile devices to enable on-the-go education. Gamification elements, collaborative tools, and robust data analytics can further enhance user engagement and provide valuable insights into learner progress. Expansion of content variety, multilingual support, and a focus on accessibility contribute to a more inclusive learning environment. Exploring blockchain for credential verification, forging strategic partnerships, and continuous user feedback loops will ensure that the Learning Management System stays at the forefront of the evolving online education landscape. With a commitment to security, compliance, and adapting to global educational standards, Learning Management System can continue to provide educators and learners with an innovative and seamless online learning experience.

6 BIBLIOGRAPHY

- www.Scribd.com
- www.java.sun.com
- www.Roseindia.com
- Java The Complete References (By Phil Hana)
- www.FreeProjectz.com

