LIBRARY MANAGEMENT SYSTEM

A project report submitted to

Rajiv Gandhi University of Knowledge Technologies

SRIKAKULAM

In partial fulfilment of the requirements for the Award of the degree of

BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING

Submitted By

3rd year B.Tech 2nd Semester

S.PAVANKUMAR (S180614)

G.HARITHA (S180510)

CH.RAMADEVI (S180253)

Under the Esteemed Guidance of

Mr. N. SESHA KUMAR

Assistant Professor

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that this project work titled “LIBRARY MANAGEMENT SYSTEM” was carried out by S.Pavankumar (S180614),G.Haritha (S180510),CH.Ramadevi (S180253).In partial fulfillment of requirements for the award of Bachelor of Technology in Computer Science and Engineering is a bona fide work carried out by them under my supervision and guidance.

The report has not been submitted previously in part or in full to any other University or Institution for the award of any degree or diploma.

Mr. N. SESHA KUMAR	Mr. N. SESHA KUMAR
Head of the Department	Head of the Department

DECLARATION

We, S. Pavan kumar ,G. Haritha, CH.Ramadevi hereby declare that this report entitled “LIBRARY MANAGEMENT SYSTEM” submitted by us under the guidance and supervision of Mr. N. Sesha Kumar is a bona fide work carried out by us during the year 2022-2023 in partial fulfillment of the requirements for the mini project in Computer Science and Engineering. I also declare that it has not been submitted previously in part or in full to this University or Institution for the award of any degree or diploma. The matter embodied in this dissertation report has not been submitted elsewhere for any other degree. Furthermore, the technical details furnished in various chapters of this thesis are purely relevant to the above project and there is no deviation from the theoretical point of view for design, development and implementation.

S.PAVANKUMAR (S180614)

G.HARITHA (S180510)

CH.RAMADEVI (S180253)

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to Mr.N.Sesha Kumar , Head of the department of the Computer Science and Engineering, our project Guide, for valuable suggestions and keen interest throughout the progress of our course of research and for providing excellent computing facilities and a congenial atmosphere for progressing with our project. At the outset, we would like to thank Rajiv Gandhi University of Knowledge and Technologies, Srikakulam for providing all the necessary resources for the successful completion of our course work. At last, but not the least, we thank our classmates and other students for their physical and moral support.

Project members:

S.PAVANKUMAR (S180614)

G.HARITHA (S180510)

CH.RAMADEVI (S180253)

ABSTRACT

The functions and resources of a library are managed by the library management system, a piece of software. This technology automates a variety of library functions, such as circulation, procurement, and stock maintenance. The librarian may effortlessly complete a variety of tasks, including adding, modifying, searching for books, maintaining borrower information, and generating reports, thanks to the library management system's effective and user-friendly interface. The system is designed to provide library staff and users with prompt and accurate information while ensuring efficient and effective management of the library's resources. The system uses a single database to keep track of all information about books, borrowers, and library transactions in order to maintain data accuracy and consistency. Additionally, the library management system offers security capabilities to guarantee the protection of the library's resources and prevent unauthorized access. With the aid of this system, librarians can efficiently run the library and give customers a better experience.

KEYWORDS:

HTML,CSS(cascading style sheets),MERN(MONGO DB,EXPRESS JS,REACT JS,NODE JS).

INDEX

Chapter 1

Introduction

1.1 Introduction

A complete software program called the Library Management System was created to streamline and automate library operations. It offers a central platform for controlling different library operations such book cataloguing, circulation, member management, and reporting. The objective of this project report is to present a thorough and expert analysis of the Library Management System, highlighting its advantages, characteristics, and importance with regard to modern libraries.

1.2 Problem statement

Librarians manage library resources and services by hand. It emphasizes the necessity of a modern library management system to streamline operations and automate tasks. Features including book acquisition, borrower data management, and report generating should be included of the system. It should have an easy-to-use user interface so that librarians may complete tasks quickly. To safeguard library resources and guarantee authorized access, security measures are crucial. The system should be scalable and flexible enough to accommodate the particular needs of various libraries. The usage of a library management system can improve user experience, increase productivity, and save time and money.

1.3 Scope of the project

A library management system automates and streamlines library operations. It covers functionalities like book acquisition, borrower information management, reservation and holds, reporting and analytics, and security measures. The system aims to improve efficiency, enhance user experience, and safeguard library resources. It should be scalable, adaptable, and capable of integration with other systems or services.

1.4 Acronyms, Definitions & Keywords

- **MERN STACK:** Mongo DB, Express JS, react JS and node JS**.**
- **HTML:** HTML is the standard markup language for creating web pages.
- **CSS:** CSS stands for Cascading Style Sheets is a style sheet language used to describe the presentation of a document written in HTML.CSS describes how elements should be rendered on screen, on paper, in speech, or on other media**.**

Chapter 2

Literature Survey

2.1 Study:

Developing a Web-Based Library Management System: Design and Implementation Sheikh Saiful Islam, Md. Mostafizur Rahman, and Md. Iqbal Hasan are the authors.

Xplore IEEE link: Web-Based Library Management System Design and Implementation.

Title: Design and Implementation of a Web Services-Based Digital Library Management System. Tao Hong, Jing Zeng, and Zibin Wang are the authors. Design and Implementation of a Digital Library Management System Based on Web Services (IEEE Xplore Link).

Reading these papers can help us develop a better understanding of the topic and may even inspire ideas for our own project by giving us insights into the design, implementation, and technologies utilized in various library management systems.

2.2 Benefits

- **Improved Resource Discovery**: The LMS has capabilities that make it simple for users to search and find library resources. By offering appropriate search outcomes, availability data, and recommendations, it improves the user experience.

- **Accurate Inventory Management:** Libraries can keep accurate and current inventory data by using an LMS. It helps keep track of the materials' status, and availability, reducing mistakes and decreasing the possibility of losing objects.

- **Time and cost Savings**: Using an LMS to automate and streamline library processes results in significant time and money savings. By minimizing laborious tasks, removing of redundant jobs, and improving the use of resources, it allows the librarians to concentrate on providing better services.

Chapter 3
INSTALLATIONS

3.1 Installing Process:

Installing Node.js and npm (node package manager) for the library management system is as follows:

- Visit https://nodejs.org to access the official Node.js website.
- Acquire the LTS (Long-Term Support) version Run the installer and follow to the instructions. • As installing, accept the license's conditions.
- Run node -v and npm -v in a command prompt or terminal to check the installation.
- Utilize npm install in the project directory to install project dependencies. Use Node.js to begin developing your library management system**.**

Installing GitHub Desktop for the Library Management System is as follows:

- Go to https://desktop.github.com to access the GitHub Desktop page.
- Download the installer for Windows.
- Start the installer and adhere to the instructions.
- Sign in with your GitHub credentials or make a new account.
- Finish installing files.
- Create a repository and Start using GitHub Desktop version.

Chapter 4

System analysis

4.1 **Existing system**:

Existing library management systems depend on Relational Database Management Systems such as MySQL, Oracle, and others. In comparison to the proposed system, the current system is not more scalable. It is made with outdated front-end technologies, making it less dynamic and responsive. It uses an RDBMS, which limits performance and flexibility when managing huge amounts of data. Customizing and expanding it further might not be much simpler.

4.2 **Proposed system**:

MERN stack could make use of a more modern technological stack. MERN stack is known for its scalability, which means it may be better able to handle large amounts of traffic and data. It may have a more responsive, perceptive, and engaged user because it is developed with the MERN stack. It makes use of Mongo DB, a No SQL database with greater performance and flexibility for managing massive amounts of data. It might be simpler to modify and add to it.

4.3 System requirements:

4.3.1 Software components:

- OPERATING SYSTEM : Windows
- FRONT END TECHNOLOGIES: HTML, CSS, REACT JS.
- BACK END TECHNOLOGIES: EXPRESS JS, NODE JS, MONGO DB.

4.3.2 Hardware Requirements:

- PROCESSOR: CORE i3 and above
- RAM: 3GB RAM or More
- MEMORY: 8GB or More

Chapter-5

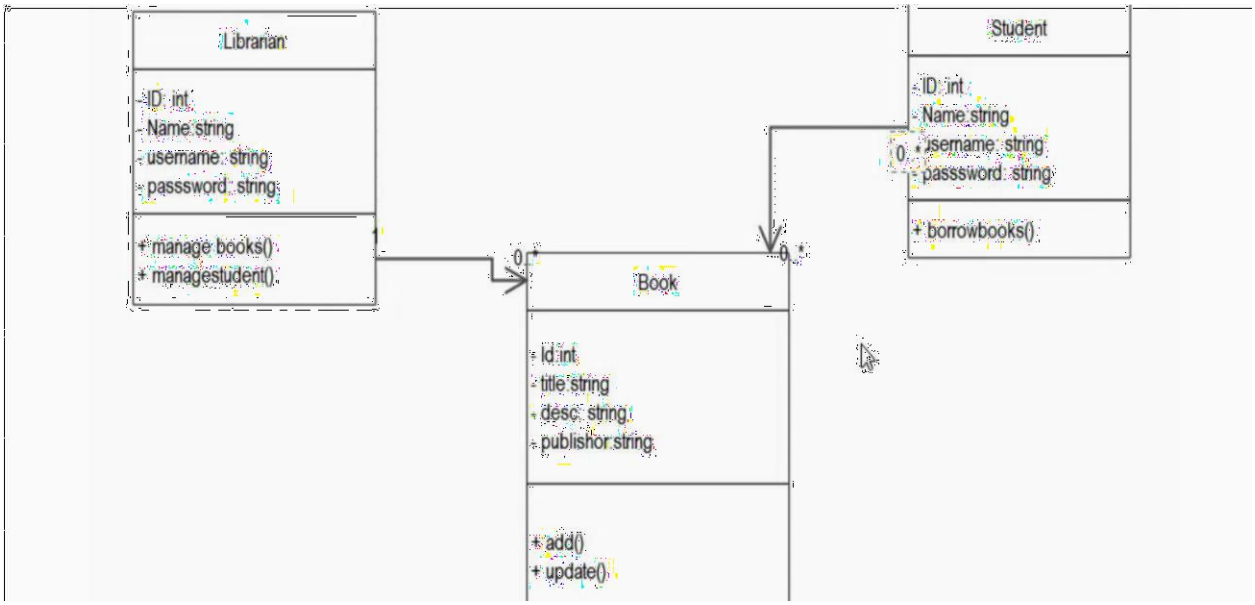
System Design

5.1 Design of the system:

The library management system is created as a client-server application that runs on the web. The system makes use of the MERN stack, which includes Node.js as the runtime environment, Express.js as the server-side framework, React as the clientside user interface, and Mongo DB as the database. The system's backend is constructed with Node.js and Express.js, and it offers a API to manage HTTP requests and responses. Data storage and retrieval are made possible by the API's use of the Mongoose library to communicate with the Mongo DB database. The database's collections for books, users, transactions, and other pertinent entities ensure that the various data sets are properly structured and related to one another. Safe password management and storage. Common security risks are prevented by the implementation of input validation and sanitization. The user interface was created with adaptability to various devices and screen sizes in mind. Users may quickly navigate and interact with the system because to the usage of an intuitive and user-friendly design. Utilizing UI frameworks or component libraries, as well as using consistent and visually pleasing UI components, can improve design efficiency and consistency. A dependable, userfriendly, and effective application to manage library resources, user accounts, and transactions can be built on a solid foundation provided by this system design for the library management system utilizing the MERN stack.

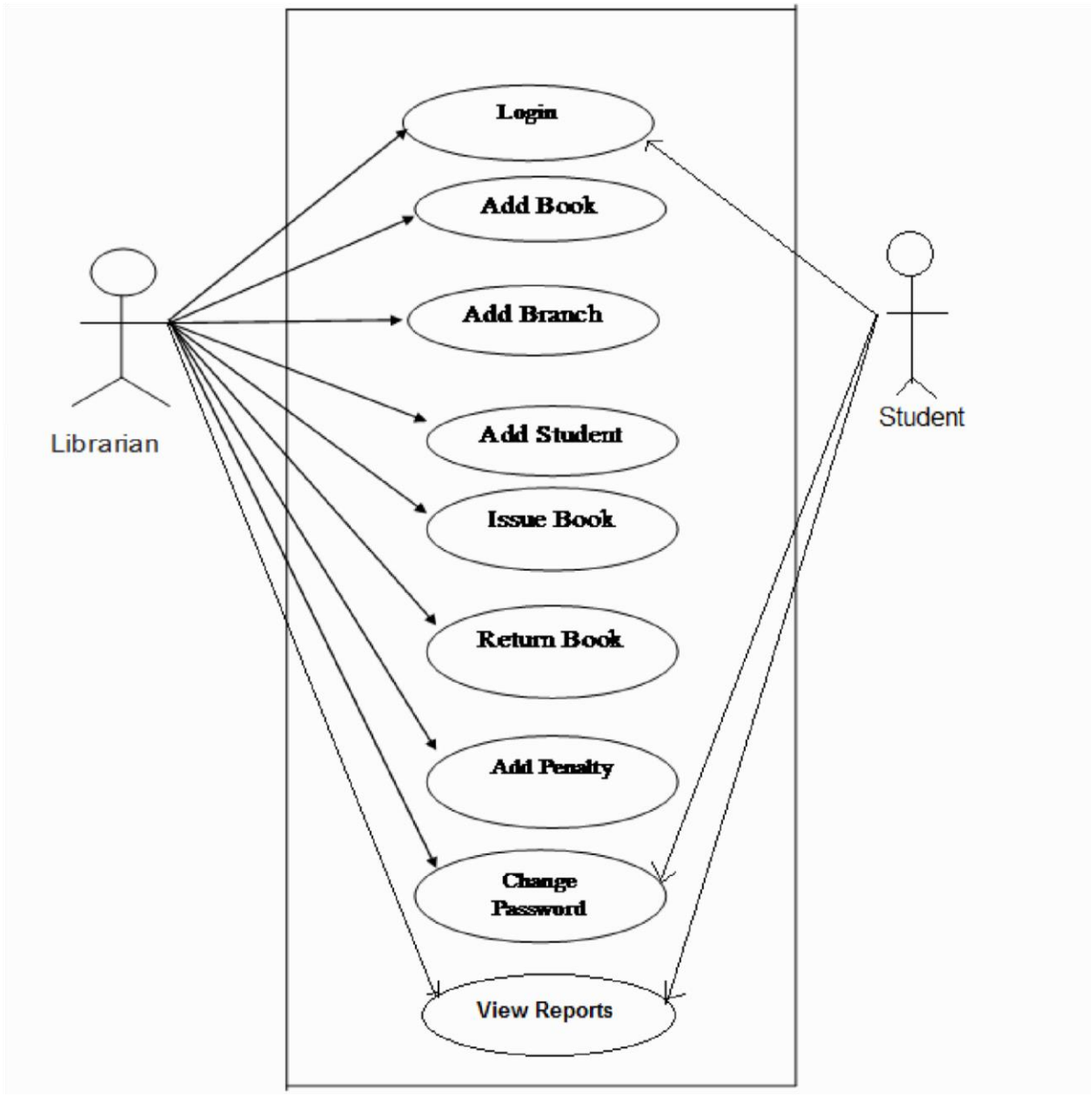
5.1.1 Class Diagram:

The Class Diagram represents classes, their attributes, and their relationships to show how the system is arranged. Book, student, Librarian, Administrator are some examples of important classes in a library management system. Relationships like a Member borrowing a book are defined by associations between classes.



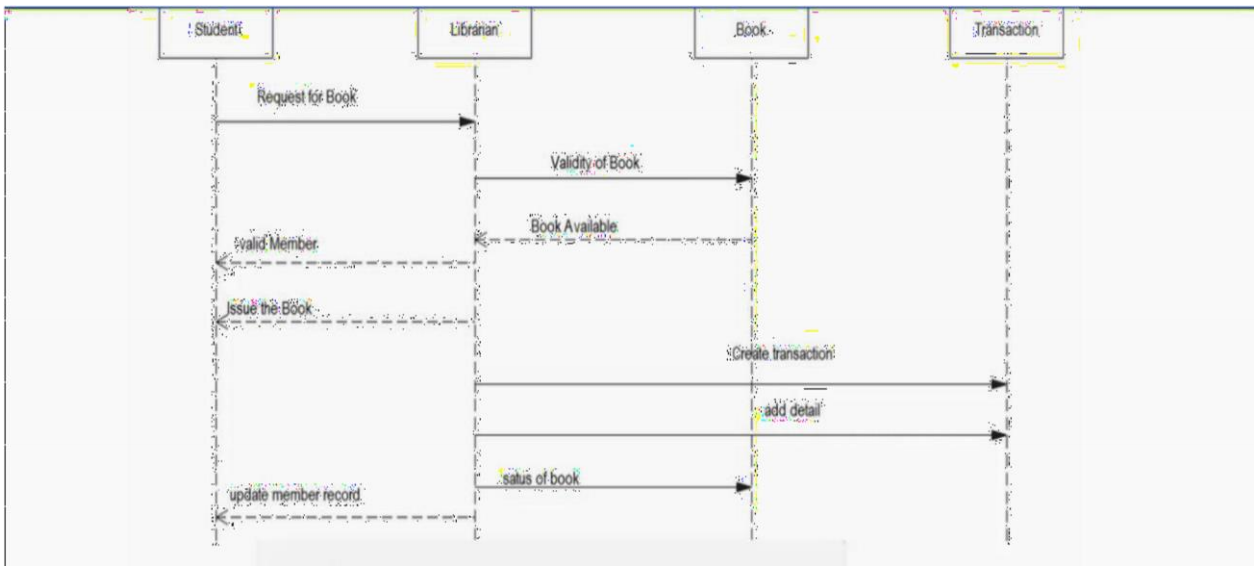
5.1.2 Use Case Diagram:

The Use Case Diagram shows how the system and its actors interact. The system administrator, library employees, and readers are frequently the performers. Use cases consist of procedures like creating a new user account, borrowing and receiving books, looking for resources, creating reports, and maintaining user accounts.



5.1.3 Sequence Diagram:

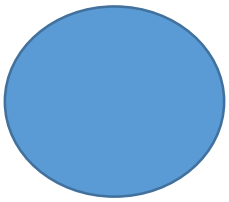
The flow of interactions between items or components through time is shown in a sequence diagram. It shows how various system parts work together to carry out particular tasks. A sequence diagram, for instance, could show how a Member looks for a book, makes a reservation for it, and then borrows it from the library.



5.1.4 DFD Diagrams:

A data flow diagram or bubble chart (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated. DFD's can also be used for the visualization of data processing (structured design). A DFD shows what kinds of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel (which is shown on a flowchart).The primitive symbols used for constructing DFD's are: Symbols used in DFD.

A circle represents a process.

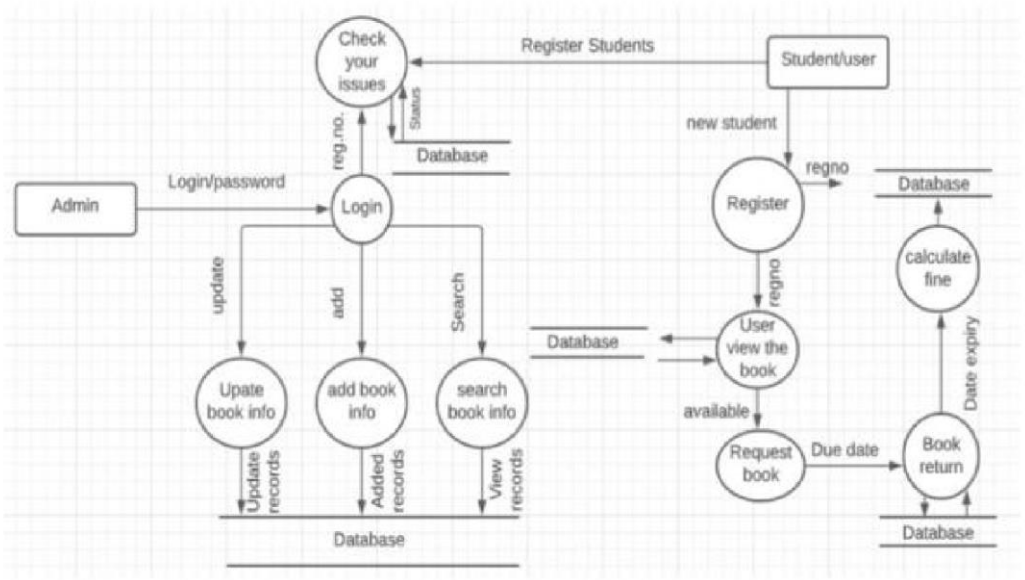


A rectangle represents external entities.

A square defines a source or destination of the system data.

An arrow identifies data flow.

LEVEL 2 DFD


Check your issues
Register Students
Student/user
Status
reg.no.
Database
new student
regno
Database
Admin
Login/password
Login
Register
calculate fine
update
add
Search
regno
Database
User view the book
Upate book info
add book info
search book info
available
Date expiry
Request book
Due date
Book return
Update records
Added records
View records
Database
Database

Chapter 6

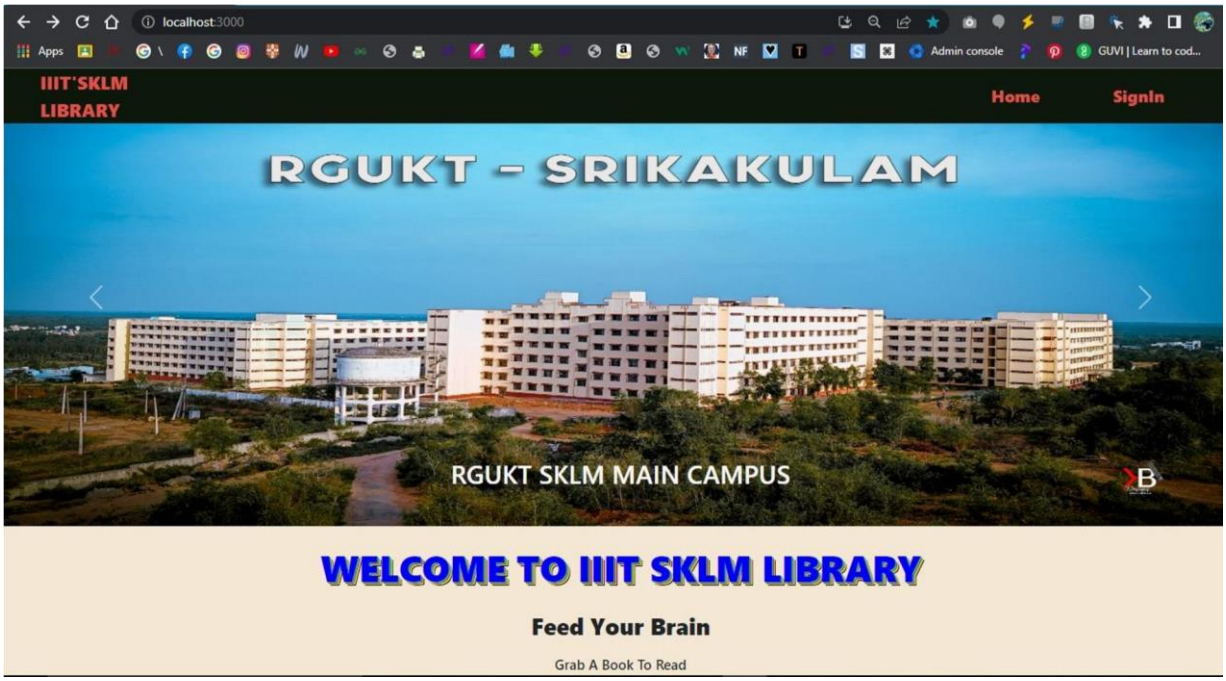
Working and design concepts

6.1 Procedure

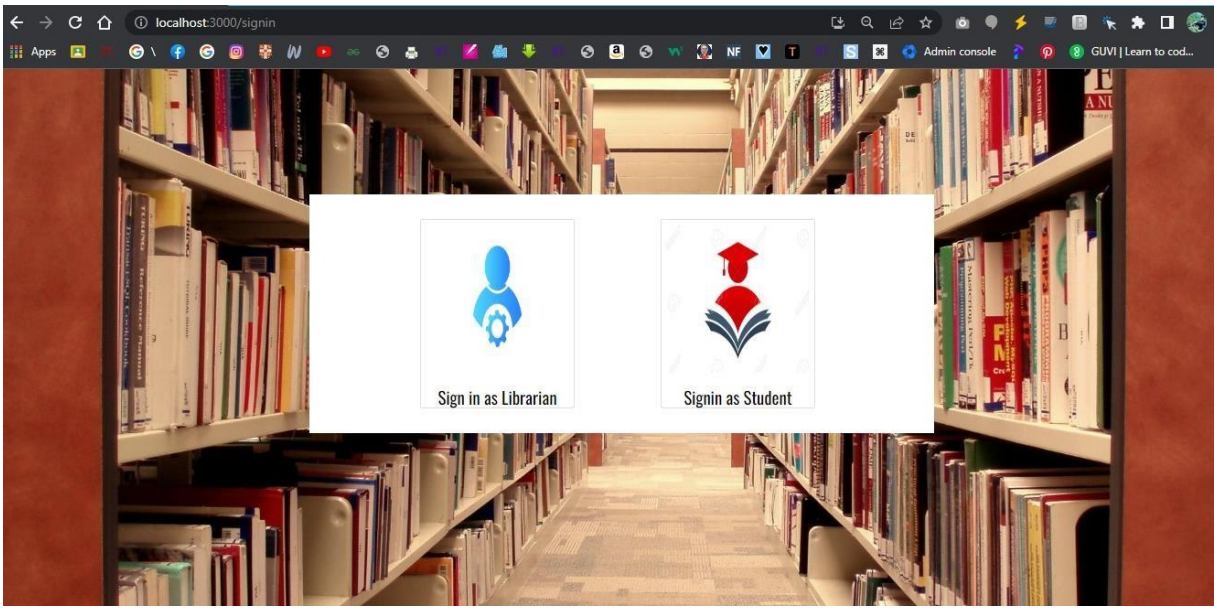
Project planning and requirements collecting, where the project scope and goals are specified and requirements are gathered from clients, are the first steps in the process of creating a library management system utilizing the MERN stack. The next phase is database design, which includes setting up relationships and indexes as well as developing a schema to represent books, users, and transactions. Backend development entails establishing a Node.js project with Express.js, adding routes, establishing a connection to Mongo DB using Mongoose, and adding validation, authentication, and authorization controls. Throughout the development process, best practices are followed, including the use of tools for tracking progress, version control systems, and stakeholder communication. A strong and maintainable library management system is created by following best practices for software development. Reusable components are made, React Router is used for navigation, and user interfaces for book listing, search, user profiles, and transactions are designed in frontend development. Features like registration, login, and logout are implemented together with user authentication and authorization. A strong and maintainable library management system is created by following best practices for software development.

CHAPTER 7

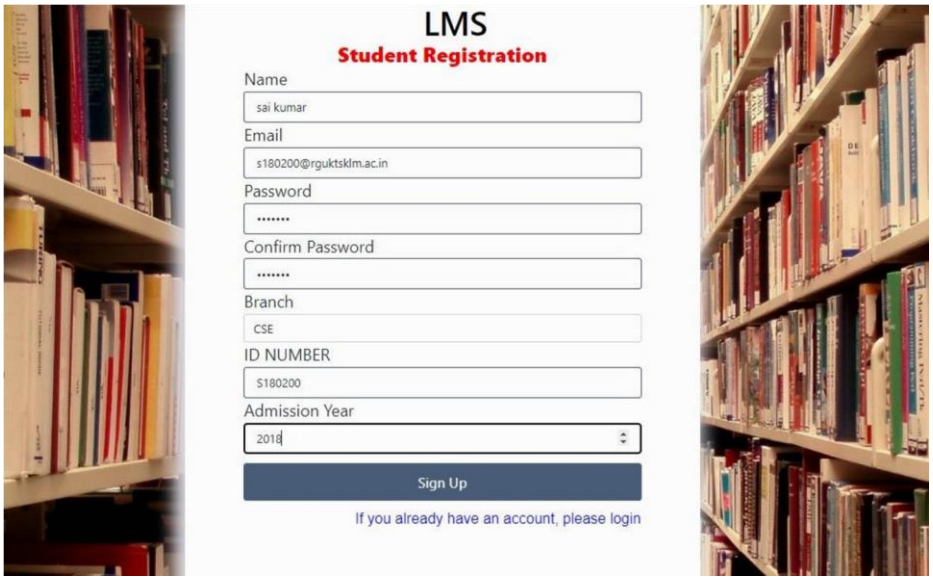
SYSTEM IMPLEMANTATION



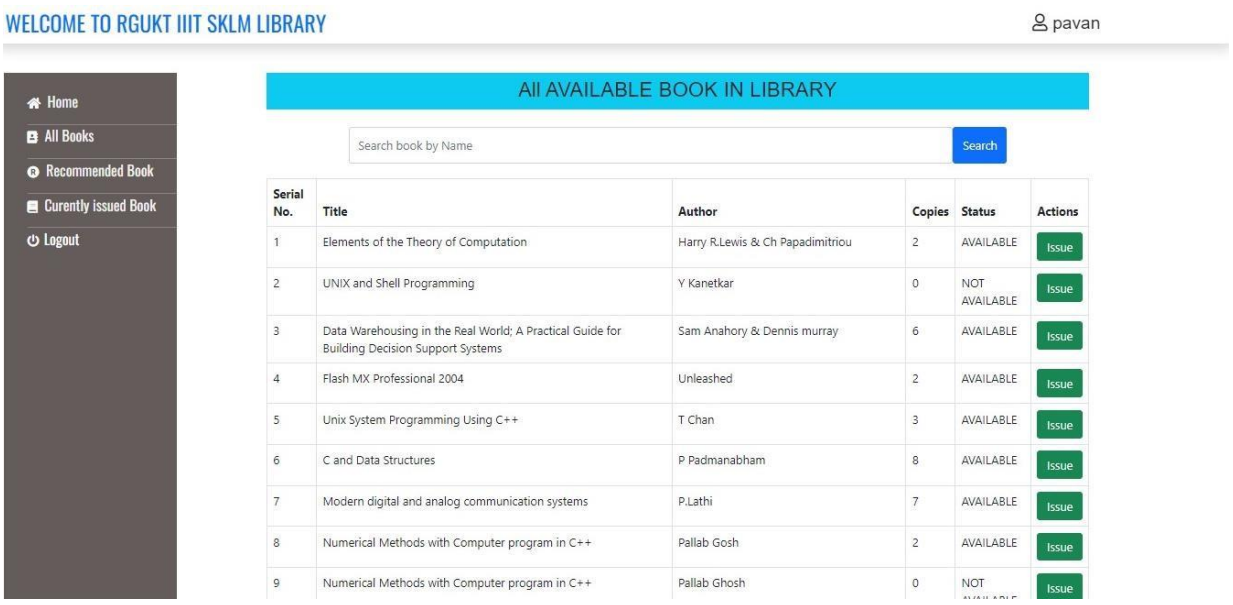
1. OUTPUT OF SIGN IN PAGE:



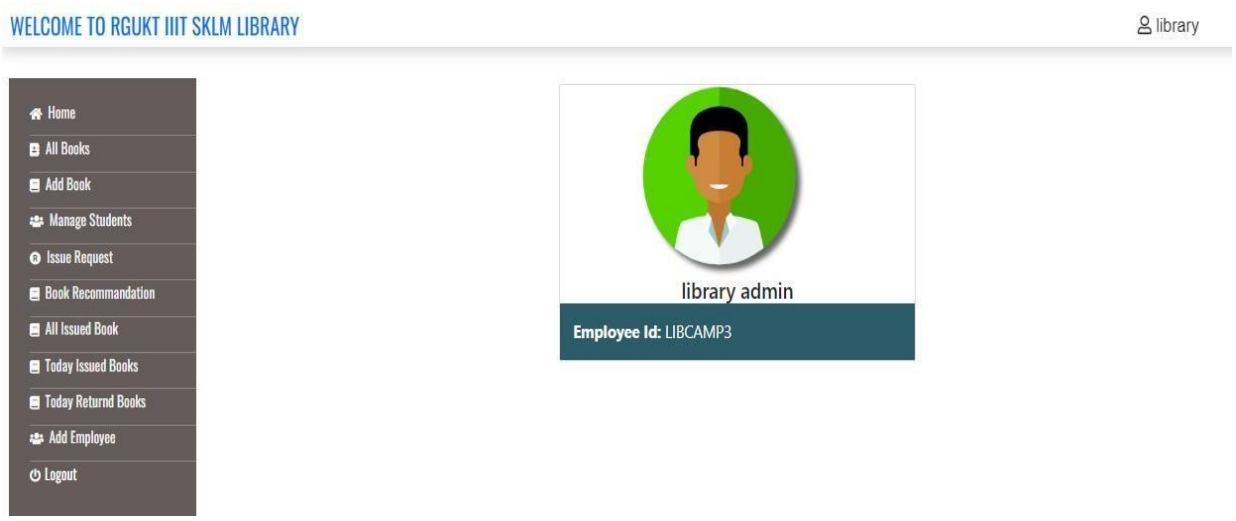
2. OUTPUT OF STUDENT REGISTRATION PAGE:

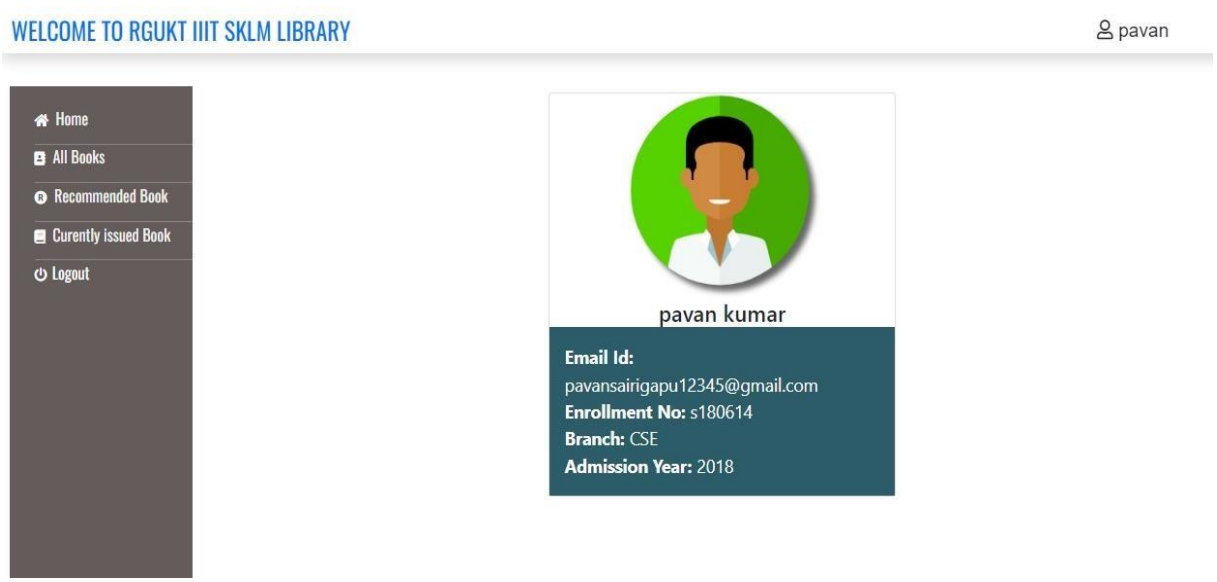


3. OUTPUT OF ALL BOOKS:



4. OUTPUT OF STUDENT AND ADMIN DASHBOARDS:





Chapter-7

Source Code

```
// AdminLogin.js import React, { useState } from
"react"; import { Link } from "react-router-dom";
import { loginUser } from "../actions/user_action";
import { useDispatch } from 'react-redux'; import
AdminIMage from "../Images/admin2.jpg"; const
AdminLogin = () => {     const [password,
setPassword] = useState("");     const [show,
setShow] = useState("password");     const [roll_no,
setRoll_no] = useState("");     const dispatch =
useDispatch();     const PostData = () => {
const user = { password, roll_no };
dispatch(loginUser(user));
   };
   const handleShow = () => {
if (show === "password") {
setShow("text");
      } else {
         setShow("password");
      }     };
return (
<div>
         <div className="LoginPage"></div>
         <div className="login_container">
            <div className="col-md-6 m-auto" style={{ opacity: 1 }}>
               <div style={{ marginLeft: "37%" }}>
                  {/* <div id="circle"></div> */}
                  {/* <h3 className="LMS" style={{ fontFamily: "sans-serif" }}> LOGIN </h3>
*/}
               </div>
               <p style={{ color: "white", fontWeight: "800", textAlign: "center" }}>Welcome to
Library Management System</p>

               <img src={AdminIMage} alt="AdminImage" style={{ height: "220px", width:
"220px", borderRadius: "50%" }} />
               <br />
               <div style={{ marginTop: "20px" }}
                  <input
                    type="text"
```

```
                    className="form-control"
                    style={{ height: "60px", borderRadius: "20px" }}
                    placeholder="Employee Id"
                    value={roll_no}
                    onChange={(e) => setRoll_no(e.target.value)}
                  />
                </div>
                <br />
                <div>
<input
type={show}
                    className="form-control"                    style={{ height:
"60px", borderRadius: "20px" }}
placeholder="Password"                    value={password}
                    onChange={(e) => setPassword(e.target.value)}
                  />
                  <i className="fas fa-eye" onClick={handleShow}></i>
                </div>
                <br />
                <button                    style={{ width: "100%", height: "60px", color: "white",
borderRadius: "20px",
backgroundColor: "red" }}
                  onClick={PostData}
                >
                  Login
                </button>
              </div>
<br />
              <Link to="/" style={{ fontFamily: "sans-serif", color: "white", textDecoration:
"none" }}>
                Go To Home Page
              </Link>
            </div>
          </div>
    );
};
 export default AdminLogin;
```

// student login.js :

```
import React, { useState } from "react"; import {
Link } from "react-router-dom"; import { loginUser
} from "../actions/user_action"; import {
useDispatch } from 'react-redux'; import
StudentIMage from "../Images/student4.jpg";
```

```
const Login = () => {   const [password, setPassword]
= useState("123456");   const [show, setShow] =
useState("password");   const [roll_no, setRoll_no] =
useState("CS3150");   const dispatch = useDispatch();

  const PostData = () => {     const
user = { password, roll_no };
dispatch(loginUser(user));
  };

  const handleShow = () => {
if (show === "password") {
setShow("text");     } else {
     setShow("password");
   }
  };

  return (
    <div>
      <div className="LoginPage"></div>
      <div className="login_container">
        <div className="col-md-6 m-auto" style={{ opacity: 1 }}>
          <div style={{ marginLeft: "37%" }}>
            {/* <div id="circle"></div> */}
            {/* <h6 className="LMS" style={{ fontFamily: "sans-serif" }}>
             RGUKT SRIKAKULAM LIBRARY
            </h6> */}
          </div>          <h3
style={{              color:
"white",
fontWeight: "800",
textAlign: "center",
            }}
          >
            Welcome to Library Management System
          </h3>
<img
            src={StudentIMage}
alt="StudentIMage"
style={{              height:
"220px",              width:
"220px",
             borderRadius: "50%",
```

```
          }}
        />
        <div style={{ marginTop: "20px" }}>
          <input
type="text"
classNam="form-
control"
            style={{ height: "60px", borderRadius: "20px" }}
            placeholder="roll_no"
value={roll_no}
            onChange={(e) => setRoll_no(e.target.value)}
          />
        </div>
        <br />
        <div>
          <input
type={show}
className="form-control"
            style={{ height: "60px", borderRadius: "20px" }}
placeholder="password"              value={password}
            onChange={(e) => setPassword(e.target.value)}
          />
          <i className="fas fa-eye" onClick={() => handleShow()}></i>
        </div>
        <br />          <button
style={{              width:
"100%",              height:
"60px",             color: "white",
borderRadius: "20px",
            backgroundColor: "green",
          }}
          onClick={() => PostData()}
        >
          Login
        </button>
      </div>
      <br />
<Link
to="/register"
style={{
          fontFamily: "sans-serif",
color: "white",
          textDecoration: "none",
        }}
```

```
      >
        If you don't have an account, please register
      </Link>
    </div>
  </div>
 );
};
export default Login;
```

// SIGN IN.js:

```
import React from 'react'
import { Link } from 'react-router-dom/cjs/react-router-dom.min'
// import AdminIMage from '../Images' import
StudentIMage from "../Images/student4.jpg" import
AdminIMage from "../Images/admin2.jpg"
export default function Signin1() {
return (
        <div className="HomePage"  >

         <div className="col-md-6 m-auto" style={{display:"flex"
,backgroundColor:"white",padding:"2%"}}>
            <div className="card" style={{marginLeft:"15%"}}>
<img src={AdminIMage} alt="StudentIMage"
style={{height:"250px",width:"250px"}} />
              <br />
              <Link className="link_class" to="/adminLogin"> <h3
style={{fontFamily:"Oswald"}}>Sign in as  Librarian</h3></Link>
            </div>
            <div className="card" style={{marginLeft:"15%"}}>
<img src={StudentIMage} alt="StudentIMage"
style={{height:"250px",width:"250px"}} />
              <br />
              <Link className="link_class" to="/login"> <h3
style={{fontFamily:"Oswald"}}>Signin as  Student</h3></Link>
            </div>
          </div>
          {/* <div>
            <Link to="/login">signin as a admin</Link>
          </div>
          <div>

            <Link to="/login">signin as a student</Link>
          </div>
```

```
          <div>
             <Link to="/addBook">Add Book</Link>
          </div>
          <div>

             <Link to="/allBook">All Book</Link>
          </div>
          <div>

<Link to="/stuReqIssue">Issue Request</Link>
</div>
          <div>

<Link to="/manageStudent">Manage Student</Link>
</div> */}

       </div>
  )
}
```

//REGISTER.js:

```
import React, { useState, useEffect } from "react"; import {
Formik, Form, Field, ErrorMessage } from "formik"; import {
Link } from "react-router-dom"; import * as Yup from "yup";
import "yup-phone"; import "./Register.css"; import {
useDispatch } from "react-redux"; import { registerUser }
from "../../actions/user_action";

const signInSchema = Yup.object().shape({
  name: Yup.string()
.required("Name is required")
    .min(3, "Name is too short - should be 3 chars minimum")
.max(15, "Name is too long - should be 15 chars maximum"),
email: Yup.string().email().required("Email is required"),
password: Yup.string()     .required("Password is required")
    .min(6, "Password is too short - should be 6 chars minimum"),
confirmPassword: Yup.string()
    .oneOf([Yup.ref("password"), null], "Passwords must match")
    .required("Confirm Password is required"),
year: Yup.number()
    .required("Admission Year is required")
    .min(2016, "Enter the correct Admission Year ")
    .integer("Admission Year must be a whole number"), });
```

```
const initialValues = {
name: "",   email: "",
password: "",
confirmPassword: "",
branch: "CSE",
roll_no: "",   year: "",
};
const SignInForm = () => {
  const [registrationSuccess, setRegistrationSuccess] = useState(false);
const dispatch = useDispatch();

  const handleRegistration = async (values, { resetForm }) => {
await dispatch(registerUser(values));
setRegistrationSuccess(true);     resetForm();
  };

  useEffect(() => {     if
(registrationSuccess) {
     alert("You have successfully registered! Please Login!!");
    }
  }, [registrationSuccess]);


  return (
    <div className="col-md-12 register-container">
      <div className="col-md-5 m-auto  ">
        <Formik
initialValues={initialValues}
validationSchema={signInSchema}
         onSubmit={handleRegistration}
        >
         {(formik) => {
          const { errors, touched, isValid, dirty } = formik;
return (
            <div className="container">
              <div style={{ marginLeft: "37%" }}>
                <h1
style={{                    color:
"black",                   marginLeft:
"44px",                    marginTop:
"0px",                    fontSize:
"45px",
                  }}
                >
```

```
            LMS
          </h1>
        </div>
<h3             style={{
color: "red",
            fontWeight: "800",
textAlign: "center",
          }}
        >
          Student Registration
        </h3>

          {registrationSuccess && (
            <div className="success-message">
You have successfully registered!
            </div>
          )}

          <Form>
            <div className="form-row">
              <label htmlFor="name">Name</label>
              <Field
type="text"
name="name"
id="name"
                className={errors.name && touched.name ? "input-error" : null}
              />
              <ErrorMessage name="name" component="span" className="error" />
            </div>

            <div className="form-row">
              <label htmlFor="email">Email</label>
              <Field
type="email"
name="email"
id="email"
                className={errors.email && touched.email ? "input-error" : null}
              />
              <ErrorMessage name="email" component="span" className="error" />
            </div>

            <div className="form-row">
              <label htmlFor="password">Password</label>
```

```
                <Field
type="password"
name="password"
id="password"
                  className={errors.password && touched.password ? "input-error" : null}
                />
                <ErrorMessage name="password" component="span" className="error" />
              </div>

              <div className="form-row">
                <label htmlFor="confirmPassword">Confirm Password</label>
                <Field
              type="password"
name="confirmPassword"
id="confirmPassword"                    className={
                    errors.confirmPassword && touched.confirmPassword ? "input-error" : null
                  }
                />
                <ErrorMessage
name="confirmPassword"
component="span"                  className="error"
                />
              </div>

              <div className="form-row">
                <label htmlFor="branch">Branch</label>
                <Field
as="select"
name="branch"
id="branch"
className={`${
                    errors.branch && touched.branch ? "input-error" : null
                  } form-control`}
                >
                  <option value="CSE">CSE</option>
                  <option value="ECE">ECE</option>
                  <option value="CHEMI">CHEMI</option>
                  <option value="EEE">EEE</option>
                  <option value="MECH">MECH</option>
                  <option value="CIVIL">CIVIL</option>
                  <option value="PUC1">PUC1</option>
                  <option value="PUC2">PUC2</option>
                </Field>
                <ErrorMessage name="branch" component="span" className="error" />
```

```
            </div>

            <div className="form-row">
              <label htmlFor="roll_no">ID NUMBER</label>
              <Field
type="text"
name="roll_no"
id="roll_no"
                 className={errors.roll_no && touched.roll_no ? "input-error" : null}
              />
              <ErrorMessage name="roll_no" component="span" className="error" />
            </div>
          <div className="form-row">
            <label htmlFor="year">Admission Year</label>
            <Field
type="number"
name="year"
id="year"
                 className={errors.year && touched.year ? "input-error" : null}
              />
              <ErrorMessage name="year" component="span" className="error" />
            </div>

            <button
type="submit"
              className={!(dirty && isValid) ? "disabled-btn" : ""}
disabled={!(dirty && isValid)}
            >
              Sign Up
            </button>
          </Form>

          <div style={{ marginTop: "10px", marginLeft: "28%" }}>
            <Link
to="/login"
style={{
                fontFamily: "sans-serif",
color: "blue",                  textDecoration:
"none",                 fontSize: "20px",
              }}
            >
              If you already have an account, please login
            </Link>
          </div>
```

```
            </div>
          );
        }}
      </Formik>
    </div>
  </div>
 );
};

export default SignInForm; // BOOKS .js const mongoose = require("mongoose");

const bookSchema = new mongoose.Schema({
    title: {
      type: String,

  },

  author: {
type: String,

  },

  publisher: {
      type: String,

},

  year: {
type: String,
      default:2021
  },

  copies: {
type: Number,
default:15
  }


}, { timestamps: true });

module.exports = mongoose.model("Book", bookSchema);
```

```
//Issue.js

const mongoose = require("mongoose");

const IssueSchema = new mongoose.Schema({

   userId:{
type: String,
   },
   userName:{
      type: String,
   },
   userBranch:{
     type: String,
   },
   bookId:{
      type: String,
   },
title: {
      type: String,

   },

   author: {
type: String,

   },

   publisher: {
      type: String,

},

   year: {
type: String,
   },

   noOfday:{
      type:Number
   },    isIssue:{
type:Boolean,
default:false
   },
```

```
   isRecom:{
type:Boolean,
default:false
   }


}, { timestamps: true });

module.exports = mongoose.model("Issue", IssueSchema); const
mongoose = require("mongoose");

const returnSchema = new mongoose.Schema({

   userId:{
type: String,
   },
   userName:{
    type: String,
   },
   userBranch:{
type: String,
   },
   bookId:{
type: String,
   },
title: {
       type: String,

   },

   author: {
       type: String,

   },

   publisher: {
type: String,
          },
lateFees:{
type: String,
   },

   year: {
type: String,
```

```
    },

    noOfday:{
        type:Number
    },
    isReturn:{
type:Boolean,
default:false
    },


}, { timestamps: true });

module.exports = mongoose.model("ReturnBook", returnSchema);

const mongoose = require("mongoose");

const studentSchema = new mongoose.Schema({

    name: {
        type: String,

   },

   email: {
type: String,

    },

    password: {
type: String,
required: true,
    },

    roll_no: {
type: String,
required: true,
    },     branch: {
type: String,
    },
    addmission_year: {
type: String,
    },
    phone_no: {
```

```
        type: Number,
    },
    isAdmin:{
type:Boolean,
default:false
    }


}, { timestamps: true });

module.exports = mongoose.model("Student", studentSchema);
```

//ROUTES.js

```
const express = require("express"); const router =
express.Router(); const mongoose =
require("mongoose"); const Student =
require("../models/student"); const requireLogin =
require("../middleware/auth") const bcrypt =
require("bcryptjs");
const jwt = require("jsonwebtoken");

const { JWT_SECRET } = require("../keys"); router.post("/signup", (req, res) => {
    const {
name,
        password, email, roll_no, branch, year, phone_no,isAdmin

    } = req.body;     if
(!roll_no || !password) {
        return res.status(422).json({ error: "please add all the fields" });
    }
    Student.findOne({ roll_no: roll_no })
        .then((savedUser) => {
if (savedUser) {
return res
.status(422)
                    .json({ error: "student already exists with that roll no" });
            }
            bcrypt.hash(password, 12).then((hashedpassword) => {
                const user = new Student({
                    name,
                    email, roll_no, branch, addmission_year: year, phone_no,
password: hashedpassword,isAdmin

                });
```

```
            user
.save()
              .then((user) => {
                res.json({ message: "saved successfully" });
              })
              .catch((err) => {
console.log(err);
              });
          });
      })
      .catch((err) => {
console.log(err);
      });
});

router.post("/signin", (req, res) => {
console.log(req.body)
   const { roll_no, password } = req.body;
   if (!roll_no || !password) {        return res.status(422).json({ error: "please
add roll_no amd password" });
  }
  Student.findOne({ roll_no: roll_no }).then((savedUser) => {
if (!savedUser) {
        return res.status(422).json({ error: "Invalid roll_no or password" });
      }
      bcrypt
        .compare(password, savedUser.password)
        .then((doMatch) => {
          if (doMatch) {

            const token = jwt.sign({ _id: savedUser._id }, JWT_SECRET);

            res.json({
token,
              user: savedUser
            });
          } else {
            return res.status(422).json({ error: "Invalid roll_no or password" });
          }
        })
        .catch((err) => {
console.log(err);
        });
   });
```

```
});

router.get("/profile", requireLogin, (req, res) => {

   Student.find({ _id: req.user._id })
   .select("-password")
.then((admins) => {
     res.json(admins);
   })
   .catch((err) => {
console.log(err);
   });
   });

router.get("/allStudent", (req, res) => {
console.log("okk")
   Student.find().sort({ createdAt: -1 }).then(data => {
      res.status(200).json(
        data
      );
   });
});

router.post("/removeStudent" , async(req,res)=>{
       try
{
      await Student.findOneAndDelete({ _id: req.body.postId }) ;

      res.send("you successfully remove the student")

   } catch (error) {
     console.log(error);
   }


})


module.exports = router;



//issue_routes
```

```
const express = require("express"); const
router = express.Router();
const requireLogin = require("../middleware/auth")
const Issue = require("../models/issue"); const
Book = require("../models/book");
const ReturnBook= require("../models/Return")




router.post("/issueRequest", async (req, res) => {


  const { title,author,publisher,year,userId,bookId,userBranch,userName,isRecom,copies } =
req.body ;

  if(req.body.bookId != undefined){
   const Modbook = await Book.findOne({_id : bookId})
Modbook.copies -= 1 ;
   await Modbook.save();
  }

  const book = await new Issue({
     title,author,publisher,year,userId,bookId,userBranch,userName,isRecom,copies
  })
  await book.save();


})

router.post("/returnReq",async(req,res)=>{
    const {title,author,publisher,userName,userBranch,userId,bookId} = req.body

    const returnBook = await new
ReturnBook({title,author,publisher,userName,userBranch,userId,bookId});
await returnBook.save()

})

router.get("/allreturnedBook" ,(req,res)=>{
```

```
   ReturnBook.find()
.then((admins) => {
     res.json(admins);
   })
   .catch((err) => {
console.log(err);
   });
})

router.get("/issuedBook", requireLogin ,(req,res)=>{
   Issue.find({ userId: req.user._id })
   .then((admins) => {
     res.json(admins);
   })
   .catch((err) => {
console.log(err);
   });
})

router.get("/allIssuedBook" ,(req,res)=>{

   Issue.find()
.then((admins) => {
     res.json(admins);
   })
   .catch((err) => {
   console.log(err);
   });
})

router.get("/allIssueRequest" ,(req,res)=>{

   Issue.find()
.then(admins => {
     res.json(admins);
   })
   .catch((err) => {
console.log(err);
   });
})

router.post("/issuedBookDelete" , async(req,res)=>{
```

```
    const {postId} = req.body ;

try {
       await Issue.findOneAndDelete({ bookId: req.body.postId }) ;
const book = await Book.findOne({_id : postId}) ;

       book.copies += 1 ;
await book.save();
       res.send("you successfully return the book")

    } catch (error) {
       console.log(error);
    }


})


router.post("/issuedReqAccept", async(req, res) => {

   const {bookId,postId} = req.body ;

try {
       const issue = await Issue.findOne({_id : bookId})
const book = await Book.findOne({_id : postId})
         book.copies -= 1 ;
await book.save();
issue.isIssue = true     await
issue.save()
       res.send('issue Delivered Successfully')
  } catch (error) {

      return res.status(400).json({ message: error});

   }

});

router.post("/issueReqDelete" , async(req,res)=>{
      try
{
      await Issue.findOneAndDelete({ _id: req.body.postId }) ;
```

```
      res.send("you successfully return the book")

    } catch (error) {
      console.log(error);
    }


})



router.post("/issuedBook", async(req, res) => {

   const postId = req.body.postId
try {
       const book = await Book.findOne({_id : postId})
       console.log(book)
book.isIssue = true         await
book.save()
       res.send('book issued Successfully')
    } catch (error) {

       return res.status(400).json({ message: error});

    }

});

router.post("/singleIssuedBook", async(req, res) => {

    const postId = req.body.postId

    try {
      const book = await Book.findOne({_id : postId});       res.json(book)
   } catch (error) {

       return res.status(400).json({ message: error});

    }

});


module.exports = router;
```

```
///Book_routtes

const express = require("express");
const router = express.Router();

const Book = require("../models/book");
const Issue = require("../models/issue");



router.post("/addBook", async (req, res) => {

  const { title,author,publisher,year,copies } = req.body ;
console.log("req.body",req.body)
   if(req.body._id){
      const obj = await Issue.find({_id:req.body._id})
obj[0].isRecom = false
      await obj[0].save()
   }
   const book = await new Book({ title,author,publisher,year,copies})
await book.save()

   // const book = new Book({
   //    title,author,publisher,year,copies
   // })
   // book.save().then(result => {
   //    res.status(201).json({
   //       message: "Done upload!",

   //    })
   // }).catch(err => {
  //  console.log(err),
   //       res.status(500).json({
  //          error: err
  //       });
   // })

})
router.get("/allBook", (req, res) => {
   Book.find().sort({ createdAt: -1 }).then(data => {
res.status(200).json(
```

```
          data
        );
    }); });
  module.exports = router;

// ALLBOOKS.js:
import React, { useState, useEffect } from 'react';
import { getAllBook, filterBook } from "../actions/book_action"; import {
issueABook, getAllIssuedBook } from "../actions/Issue_action"; import {
useDispatch, useSelector } from 'react-redux'; import { Toast, Spinner }
from "react-bootstrap";

const AllBook = () => {   const dispatch =
useDispatch();   const [searchKey, setSearchKey]
= useState("");   const [show, setShow] =
useState(false);   const [bootTitle, setBookTitle] =
useState(null);   const [error, setError] =
useState(false);

  useEffect(() => {
dispatch(getAllBook());
dispatch(getAllIssuedBook());
  }, [dispatch, show]);

  const { books } = useSelector(state => state.getAllBookReducer);   const {
all_IssuedBook } = useSelector(state => state.allIssuedBookReducer);   const {
currentUser } = useSelector(state => state.userLoginReducer);   const userId =
currentUser.user._id;   const userBranch = currentUser.user.branch;
  const userName = currentUser.user.name;

  let filterBook22 = all_IssuedBook && all_IssuedBook.filter(book => book.userId === userId);
let newBooksId = filterBook22 && filterBook22.map(book => book.bookId);

  const postData = (book) => {
    if (newBooksId && newBooksId.includes(book._id)) {
    console.log("You cannot add one more book");
setError(true);     setTimeout(() => {
setError(false);       }, 3000);       return;
    }

    if (book.copies < 1) {
      console.log("No copies available");
return;
    }
```

```
    const { title, author, publisher, year, _id, copies } = book;
  const issueUser = {
      title,
  author,
  publisher,
  year,      userId,
  bookId: _id,
  userBranch,
  userName,
  copies
    };

    if (book.copies) {
      dispatch(issueABook(issueUser));
      setBookTitle(title);
  setShow(true);
  dispatch(getAllBook());
    } else {
      alert("Book not available");
    }

    setTimeout(() => {
  setShow(false);
    }, 5000);
   };

   return (
    <div>
     <div className="col-md-10 m-auto">
      <h3 className='text-center bg-info p-2' style={{ fontFamily: "sans-serif" }}>All
  AVAILABLE BOOK IN LIBRARY</h3>
       <br />
 <Toast onClose={() => setShow(false)} show={show} delay={3000} autohide>
<Toast.Body style={{ backgroundColor: "green", color: "white", fontSize: "18px" }}>You
                          successfully sent an issue request for {bootTitle}</Toast.Body>
     </Toast>
    </div>

    {error && <div className="alert alert-danger">You have already requested this book</div>}

    {!books.length ? (
     <div style={{ marginLeft: "40%", marginTop: "5%" }}>
      <Spinner animation="border" variant="danger" />
```

```
      </div>
    ) : (
      <>
        <div className="col-md-8 m-auto" style={{ display: "flex" }}>
          <input
type="text"
            className="form-control"
placeholder="Search book by Name"
            style={{ height: "50px" }}
            onChange={(e) => setSearchKey(e.target.value)}
value={searchKey}
          />
          <button onClick={() => dispatch(filterBook(searchKey))} className="btn
btnprimary">Search</button>
        </div>

        <div className="col-md-10 m-auto">
          <table className='table table-bordered table-responsive-sm' style={{ marginTop: "20px"
}}>
            <thead className='thead-dark'>
              <tr>
                <th>Serial No.</th>
<th>Title</th>
                <th>Author</th>
                <th>Copies</th>
<th>Status</th>
                <th>Actions</th>
              </tr>
            </thead>
            <tbody>
              {books && books.map((book, index) => (
                <tr key={book._id}>
                  <td>{index + 1}</td>
                  <td>{book.title}</td>
                  <td>{book.author}</td>
                 <td>{book.copies}</td>
                      <td>{book.copies > 0 ? "AVAILABLE" : "NOT AVAILABLE"}</td>
                <td>
                  {currentUser.user.isAdmin && (
                     <i className='fa fa-trash m-1' onClick={() => console.log("okk")}></i>
                   )}
                   {!currentUser.user.isAdmin && (
                     <button onClick={() => postData(book)} className={`btn
btnsuccess`}>Issue</button>
```

```
                )}
              </td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  </>
 )}
</div>
);
};

export default AllBook; //
ISSUE REQUEST.js
import React,{useEffect} from 'react';
import {getAllBookIssueReq ,issuedReq,issuedReqDeletedByAdmin} from
"../actions/Issue_action"
import { useDispatch, useSelector } from 'react-redux'

const IssueRequest = () => {

  const dispatch = useDispatch() ;

  useEffect(()=> {
    dispatch(getAllBookIssueReq())
  }, [dispatch])



  const {issuebooks} = useSelector(state => state.getAllIssueBookReqReducer)
const newIssuedBook = issuebooks && issuebooks.filter(item => !item.isIssue &&
!item.isRecom)

  return (
    <div className="col-md-10 m-auto">
<p style={{fontFamily:"sans-
serif",fontSize:"30px",textAlign:"center",padding:"10px"}}>Student Requested to Admin to issue
these Book</p>
          <table  className='table table-bordered table-responsive-sm'>

<thead className='thead-dark'>
  <tr >
    <th>Book Name</th>
```

```
        <th>Author</th>
        <th>Student Name</th>
        <th>Student Branch</th>
        <th>Actions</th>
    </tr>
</thead>
<tbody>

{newIssuedBook && newIssuedBook.map(book=>{

    return <tr key={book._id} >
        <td>{book.title}</td>
        <td>
            {book.author}
        </td>
        <td>
            {book.userName}
        </td>
        <td>
            {book.userBranch}
        </td>

        <td>

            <button onClick={() => dispatch(issuedReq(book._id,book.bookId))} className="btn
btn-success">Accepted</button> { "  "}
            <button onClick={() => dispatch(issuedReqDeletedByAdmin(book._id))} className="btn
btn-danger">Rejected</button>
        </td>

    </tr>

})}
</tbody>

</table>
        </div>
    );
};

export default IssueRequest; //
ADD BOOK .js:
import React, { useState } from "react"; import
{addOneBook} from "../actions/book_action"
```

```
import { useDispatch } from 'react-redux'
import {Toast} from "react-bootstrap"

const AddBook = () => {
const[title,setTitle] = useState("")
const[author,setAuthor] = useState("")
const[publisher,setPublisher] = useState("")
const[year,setYear] = useState("")
const[copies,setCopies] = useState("")
   const [show, setShow] = useState(false);

   const dispatch = useDispatch()

     const PostData = () => {
       const book = { title,author,publisher,year,copies };

       dispatch(addOneBook(book));
       setShow(true)
setTitle("")        setAuthor("")
setPublisher("")
setCopies("")        setYear("")
   };
   return (
      <div  className=" mt-5">


          <div className="card col-md-6 m-auto p-3" >
          <Toast onClose={() => setShow(false)} show={show} delay={3000} autohide>
             <Toast.Body style={{backgroundColor:"green",color:"white",fontSize:"18px"}}>One
Book Added</Toast.Body>
           </Toast>
               <h2 style={{textAlign:"center",marginBottom:"20px"}}>Add a New Book</h2>
<div className="mb-3">
                   <input type="text" placeholder="Book title" style={{height:"60px"}}
onChange={(e) => setTitle(e.target.value)} value={title} className="formcontrol" />
                </div>
                <div className="mb-2">
                   <input type="text" placeholder="Author name"
value={author}  style={{height:"60px"}}
                   onChange={(e) => setAuthor(e.target.value)} className="form-control" />
                </div>
                <div className="mb-2">
                <input type="text" placeholder="Publisher"
value={publisher}  style={{height:"60px"}}
```

```
                    onChange={(e) => setPublisher(e.target.value)} className="form-control" />
                </div>
                <div className="mb-2">
                    <input type="text" placeholder="Year" value={year} style={{height:"60px"}}
onChange={(e) => setYear(e.target.value)} className="form-control" />
                </div>
                <div className="mb-2">
<input type="text"
                     placeholder=" Number of Copies"  style={{height:"60px"}}
                     value={copies} onChange={(e) => setCopies(e.target.value)}
className="formcontrol" />
                </div>



                <button className="btn btn-primary " onClick={() => PostData()}>
Add Book
                </button>

            </div>
        </div>
    );
};

export default AddBook;
```

CHAPTER 8
SYSTEM TESTING

INTRODUCTION:

The cause of testing is to detect mistakes. Making an attempt out is the technique of looking for to realize each viable fault or weakness in a piece product. It presents a method to determine the performance of add-ons, sub-assemblies, assemblies and/or a completed product. It is the method of ex excising g program with the intent of constructing certain that the application procedure meets its necessities and client expectations and does no longer fail in an unacceptable process. There are rather plenty of forms of scan. Each experiment sort addresses a special trying out requirement.

TYPES OF TESTS:

Unit testing:

Unit testing is undertaken when a module has been created and successfully reviewed .In order to test a single module we need to provide a complete environment i.e.
besides the module we would require.

- The procedures belonging to other modules that the module under test calls.
- Non local data structures that module accesses.
- A procedure to call the functions of the module under test with appropriate parameters

Integration testing:

Testing admin login form-This form is used for log in of administrator of the system. In this we enter the username and password if both are correct administration page will open otherwise if any of data is wrong it will get redirected back to the login page and again ask for username and password.

Functional testing:

Testing on the functionality of the website is done by using Functional Testing process. By using functional testing the website validation process completed successful.

System testing:

Testing of the website as a whole is also done after the completion of all the modules and previous tests. System testing gave the places where system giving wrong outputs.

White Box Testing:

This testing is a trying out wherein where the application tester has competencies of the interior workings, constitution and software language, or at

least its cause. It's rationale. It's used to test areas that can't be reached from a black box stage.

Black Box Testing:

This is testing the software with none advantage of the inside workings, establishment or words of the unit life form veteran.

LEVELS OF TESTING

Unit testing strategy:

Unit checking out is most commonly performed as a part of a mixed code and unit experiment part of the software lifecycle, though it be not exceptional for coding and unit checking out to be performed as two targeted phases.

Test strategy and approach:

Field testing out can be carried out manually and sensible assessments shall be written in element.

Test objectives:

- Each field must be work correctly.
- Each page must be activated through the specified link.
- Features to be tested Verify that the entries are of the correct format No duplicate entries should be allowed.

Integration testing strategy:

Software integration testing is the incremental integration checking out of two otherwise further included software gears on top of a solo stage to fabricate failure induced with the aid of interface defects. The project of the mixing scan is to check that components or program applications,

Example: Components in a program approach or œ one step up software purposes at the company degree œ interact without error.

Test Results:

All of the scan circumstances recounted above passed efficiently. No defects encountered.

Acceptance Testing

User Acceptance testing trying out is a crucial section of any mission and requires enormous participation by the tip user. It additionally ensures that the procedure meets the functional specifications.

Test Results:

The entire test cases recounted above passed effectually. No defects Encountered.

Conclusion

This project “ONLINE LIBRARY MANAGEMENT SYSTEM, a college/University library website” was successfully executed by our team to check the end results using own database and it is found that our proposed system provides efficient and effective result when compared to existing system. Our System provides the facilities as mentioned like job opportunities and for sales etc,. It can be accessed through smart phones app and as well as web application to get an opportunity and saves time, resources as well. It provides useful job opportunities and news in the form of posts that will be helpful for the people.

Future Enhancement

This website provides a computerized version of library management system which will benefit the students as well as staff of the library. It makes entire process online where student can search books, staff can generate reports and do book transactions. It also has a facility for student login where student can login and can see status of books issued as well request for book. There is a future scope of this facility that many more features such as online lectures video tutorials can be added by teachers as well as online assignments submission facility, a feature of group chat where students can discuss various issues of engineering can be added to this project thus making it more interactive more user friendly and project which fulfils each user’s need in the way possible.

REFERENCES

- “Library Management System using MERN Stack" by A. S. Kale and V. S. Joshi: This paper discusses the development of a library management system using the MERN stack, emphasizing the role of Mongo DB as the database, Express.js as the server framework, and React.js for building the user interface.
- https://www.youtube.com/watch?v=7CqJlxBYj