## Back End Engineering-II

Project Report

Semester-V (Batch-2022)

Library Management System

A red and white sign

Description automatically generated with low confidence

**Supervised By: Submitted by:**

Mr. Raveesh Samkaria Sai madhav bhalla: 2210990760

Saksham dogra:2210990763

G-12

**Department of Computer Science and Engineering**

## Chitkara University Institute of Engineering & Technology,

## Chitkara University, Punjab

**ABSTRACT**

This project presents a comprehensive Library Management System (LMS) developed using Node.js and MongoDB, designed to efficiently manage the core functions of a library. The primary objective is to facilitate seamless operations for library administrators by automating tasks related to book inventory management, student records, and the borrowing and returning of books. The system is built on a modern technology stack, utilizing Node.js for the backend to leverage its non-blocking, event-driven architecture, which allows for efficient handling of multiple concurrent requests, ensuring a responsive user experience. MongoDB serves as the database, providing a scalable and flexible solution for storing diverse data related to books, students, and transactions.

The LMS includes a secure user authentication system, implemented with Express Sessions, to ensure that only authorized users can access the administrative functionalities. Role-based access control further restricts user actions based on their roles, enhancing the system's security. The LMS supports full CRUD operations (Create, Read, Update, Delete) for managing book and student records. Administrators can easily add new books to the inventory, update book details, categorize books into different genres or subjects, and remove books when necessary. Similarly, student management is simplified, allowing the addition, updating, and deletion of student records as required.

A significant feature of the LMS is its ability to track the borrowing and returning of books. The system records the issuance of books to students, monitors due dates, and updates the availability status of books upon return. It also automatically calculates fines for overdue returns, ensuring timely compliance and return of books. The categorization feature helps organize books systematically, making it easier for users to search and retrieve books based on their categories.

In conclusion, the Library Management System developed with Node.js and MongoDB offers a robust, scalable, and efficient solution for modern library management. By automating critical library functions and providing a user-friendly interface, the LMS significantly enhances the efficiency and effectiveness of library operations, making it an ideal choice for libraries seeking to modernize their systems.

**ABLE OF CONTENTS**

|  |  |
| --- | --- |
| S. No. | SECTION |
| 1. | Introduction |
| 2. | Problem statement |
| 3. | Technical details |
| 4. | Code |
| 5. | LIST OF REFERENCES |

**Introduction**

### Introduction

Libraries have always been pivotal to the dissemination of knowledge and education, acting as repositories of information and resources for learning, research, and entertainment. Traditionally, library operations have relied on manual processes to manage their collections, track book loans, and handle user interactions. However, with the growth in library collections and the increasing number of users, these manual processes have become inefficient, time-consuming, and error-prone. The advent of digital technology offers a promising solution to these challenges, allowing libraries to automate their operations and improve their services significantly.

In recent years, the need for automated Library Management Systems (LMS) has become apparent. Such systems help streamline and optimize various library functions, including cataloging books, managing user records, tracking borrowing and returning of books, and generating reports. By automating these tasks, libraries can operate more efficiently, reduce errors, and provide better service to their patrons.

This project introduces a Library Management System developed using Node.js, a popular JavaScript runtime, and MongoDB, a leading NoSQL database. Node.js is known for its non-blocking, event-driven architecture, which makes it suitable for building scalable and high-performance web applications. MongoDB, with its flexible schema and scalability, is an ideal choice for managing the diverse and growing data of a library. Together, these technologies provide a robust platform for building a modern LMS that can handle the dynamic needs of today's libraries.

The proposed LMS will cover essential aspects of library management, including user authentication, book management, student record management, borrowing and returning books, and report generation. It will provide a secure, user-friendly interface for library staff to manage their operations efficiently and allow patrons to interact with the library more seamlessly. By leveraging the power of Node.js and MongoDB, this system aims to bring significant improvements in the way libraries operate, ensuring they can meet the demands of modern users while maintaining high standards of service and efficiency.

* **Background**

Libraries have long been essential resources for educational institutions, research centers, and communities, providing access to vast collections of books and other informational resources. Traditionally, managing these resources has involved manual processes, which are often time-consuming and prone to errors. As libraries have grown in size and complexity, there has been a growing need for more efficient and automated systems to handle the cataloging, lending, returning, and organizing of books. Modern technology, particularly the advent of database management systems and web technologies, offers a solution to these challenges. By leveraging technologies like Node.js and MongoDB, libraries can streamline their operations, reduce manual workload, and improve the user experience for both library staff and patrons.

* **Objective**

The primary objective of this project is to develop a robust and efficient Library Management System (LMS) using Node.js and MongoDB. This system aims to automate and streamline library operations, including:

* **User Authentication and Authorization**: To ensure secure access to the system, allowing only authorized personnel to perform specific actions based on their roles.
* **Book Management**: To provide a user-friendly interface for adding, updating, deleting, and categorizing books, ensuring that the library's inventory is always accurate and up-to-date.
* **Student Management**: To manage student records efficiently, enabling easy tracking of who has borrowed which books.
* **Borrowing and Returning Books**: To track the issuance and return of books, monitor due dates, and automatically calculate fines for overdue books.
* **Reporting**: To generate various reports that provide insights into the library's operations, such as lists of borrowed books, overdue books, and inventory status.
* **Significance**

The implementation of this Library Management System is significant for several reasons:

1. **Efficiency**: By automating routine tasks, the system reduces the administrative burden on library staff, allowing them to focus on more critical functions. This leads to faster processing of book issues and returns, more accurate inventory management, and a more organized library environment.
2. **Scalability**: Using MongoDB as the database allows the system to handle large volumes of data efficiently. This is particularly important for large libraries with vast collections of books and numerous patrons.
3. **Improved User Experience**: A well-designed LMS provides a better experience for both library staff and users. Staff can easily manage records and track inventory, while users benefit from quicker checkouts and returns, and easier access to information about book availability.
4. **Data Accuracy and Security**: The system minimizes human errors that are common with manual record-keeping. Additionally, secure authentication and role-based access control protect sensitive data and ensure that only authorized users can access or modify the records.
5. **Cost-Effectiveness**: Automating library operations reduces the need for paper-based record-keeping and can minimize the costs associated with managing a library. It also reduces the time spent on manual processes, which can translate into labor cost savings.
6. **Support for Decision-Making**: By providing comprehensive reporting features, the LMS helps library administrators make informed decisions about inventory management, procurement, and other strategic areas, leading to better resource utilization.

**Problem Statement:**

Traditional libraries face numerous challenges in managing their day-to-day operations due to the reliance on manual processes. These challenges include the difficulty of maintaining accurate records of books, students, and transactions, the inefficiency of handling book loans and returns, and the increased likelihood of human errors in tracking overdue books and fines. As library collections grow and user bases expand, these issues become more pronounced, leading to disorganized inventories, delayed services, and a poor experience for both library staff and patrons.

Specifically, the problems that libraries encounter include:

1. \*\*Inefficient Record-Keeping\*\*: Manual tracking of book inventory, student information, and loan records is time-consuming and prone to errors. Misplaced or incorrect records can lead to difficulties in locating books and ensuring their availability to users.

2. \*\*Poor User Experience\*\*: Long waiting times for book checkouts and returns, along with the inability to easily search and find books, negatively impact the user experience. Patrons may find it challenging to know the availability of a book or its due date for return.

3. \*\*Difficulty in Managing Borrowing and Returns\*\*: Without automated tracking, it is hard to monitor which books are borrowed, who has borrowed them, and when they are due. This can lead to books being lost or not returned on time, affecting the library's collection.

4. \*\*Challenges in Fine Calculation\*\*: Calculating fines for overdue books manually is error-prone and cumbersome, leading to inconsistencies and disputes between library staff and users.

5. \*\*Lack of Comprehensive Reporting\*\*: Manual systems make it difficult to generate reports that provide insights into library operations, such as the most borrowed books, overdue items, or inventory shortages, which are crucial for effective management and decision-making.

6. \*\*Security Concerns\*\*: Ensuring that only authorized personnel can access sensitive information and perform administrative tasks is challenging without a secure authentication and authorization system.

These problems highlight the need for a robust, automated Library Management System that can efficiently handle the various aspects of library operations. By implementing a digital solution using modern technologies like Node.js and MongoDB, libraries can overcome these challenges, leading to improved efficiency, better user experiences, and more effective management of resources. The proposed system aims to address these issues by providing a comprehensive, scalable, and user-friendly platform for managing library operations seamlessly.

**Technical Details:**

The Library Management System (LMS) is designed to be a robust, scalable, and efficient platform for managing library operations. The technical implementation utilizes modern web development technologies, with Node.js as the backend framework and MongoDB as the database solution. Below are the key technical details of the LMS:

#### 1. \*\*Architecture and Design\*\*

- \*\*Client-Server Architecture\*\*: The LMS follows a client-server architecture where the client (web browser) interacts with the server (Node.js) through HTTP requests. The server processes these requests, performs the necessary operations (e.g., accessing the database), and sends back the response.

- \*\*RESTful API\*\*: The system's functionalities are exposed via RESTful API endpoints. Each endpoint corresponds to a specific resource (e.g., books, students) and supports CRUD (Create, Read, Update, Delete) operations.

- \*\*Modular Design\*\*: The application is structured in a modular fashion, with separate modules for handling different functionalities such as user authentication, book management, and borrowing/returning of books. This modularity facilitates maintainability and scalability.

#### 2. \*\*Technologies Used\*\*

- \*\*Node.js\*\*: A JavaScript runtime built on Chrome's V8 engine, used for building the server-side application. Node.js is chosen for its non-blocking, event-driven architecture, which is well-suited for handling multiple concurrent requests efficiently.

- \*\*Express.js\*\*: A minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications. Express is used to handle routing, middleware, and server-side logic.

- \*\*MongoDB\*\*: A NoSQL database known for its flexibility, scalability, and ease of use. MongoDB stores data in JSON-like documents, making it an ideal choice for the LMS, where different types of data need to be managed efficiently (e.g., books, students, transactions).

- \*\*EJS (Embedded JavaScript)\*\*: A templating engine used to generate HTML pages dynamically. EJS allows the incorporation of JavaScript code within HTML, enabling the creation of dynamic web pages based on server-side data.

#### 3. \*\*Database Design\*\*

- \*\*Collections\*\*: MongoDB collections are used to store different types of data. Key collections include:

- \*\*Books\*\*: Stores information about each book, such as `book\_id`, `title`, `author`, `category`, `price`, `total\_copies`, `available\_copies`, and `status`.

- \*\*Students\*\*: Contains records for each student, including `student\_id`, `name`, `email`, `phone\_no`, `department`, and `status`.

- \*\*Borrow\*\*: Tracks borrowing transactions, including `borrow\_id`, `student\_id`, `book\_id`, `issue\_date`, `due\_date`, and `status`.

- \*\*Return\*\*: Manages returning transactions with details like `borrow\_id`, `return\_date`, `fine`, and `status`.

- \*\*Categories\*\*: Stores different book categories identified by `cat\_id` and `name`.

- \*\*Relationships\*\*: While MongoDB is a non-relational database, relationships between data entities (e.g., books and students) are managed through referencing. For example, a borrowing transaction document references a `student\_id` and a `book\_id` to establish a link between the student and the borrowed book.

#### 4. \*\*Security and Authentication\*\*

- \*\*Express Sessions\*\*: Used for handling user sessions. Session data is stored on the server, and a session ID is sent to the client as a cookie. This ensures that user state is maintained across different requests.

- \*\*Role-Based Access Control\*\*: Different user roles (e.g., admin, librarian, student) are defined to restrict access to certain features based on user roles. This ensures that only authorized personnel can perform specific actions (e.g., adding or deleting books).

#### 5. \*\*Error Handling and Validation\*\*

- \*\*Middleware\*\*: Express middleware is used to handle errors and perform validation. Custom middleware functions validate input data (e.g., checking if a book's details are correctly formatted before adding it to the database) and handle exceptions to prevent the application from crashing.

- \*\*Error Handling\*\*: Global error handling middleware catches and processes errors, providing meaningful error messages to the client and logging errors for debugging purposes.

#### 6. \*\*Static File Handling and Templating\*\*

- \*\*Serving Static Files\*\*: Express is configured to serve static files (e.g., CSS, JavaScript, images) from a designated directory. This allows the front-end interface to be styled and behave interactively.

- \*\*EJS Templates\*\*: HTML templates are rendered using EJS, which allows embedding JavaScript logic within HTML. This approach helps in generating dynamic content, such as displaying lists of books, students, and transactions based on real-time data from the server.

#### 7. \*\*Deployment and Scalability\*\*

- \*\*Environment Setup\*\*: The application is designed to run on a Node.js environment, with MongoDB as the backend database. It can be deployed on cloud platforms like AWS, Heroku, or DigitalOcean for scalability and reliability.

- \*\*Scalability\*\*: The use of Node.js and MongoDB ensures that the LMS can handle increased loads by scaling horizontally. Additional instances of the Node.js application can be deployed to manage more users and transactions, while MongoDB's sharding capabilities can distribute the database load.

Top of Form

**CODE**

**Index.js**

const express = require('express');

const app = express();

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

const session = require('express-session');

let { ObjectId } = require('mongodb');

let db = require('./database.js');

const path = require('path');

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

app.use(express.static('public'));

app.set('view engine', 'ejs');

app.use(session({secret: "node\_mongo123!@#", resave:true, saveUninitialized: true}));

app.set('view engine', 'ejs');

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

// Login

app.get('/', (req, res) => {

res.render('login');

});

app.post('/login', (req, res) => {

const { username, password } = req.body;

db.collection('admin').findOne({ username, password })

.then(user => {

if (user) {

req.session.user = user;

res.redirect('/dashboard');

} else {

res.render('login', { error: 'Invalid username or password, please try again' });

}

})

.catch(err => {

console.error(err);

res.render('login', { error: 'An error occurred, please try again' });

});

});

//Dashboard

app.get('/dashboard', (req, res) => {

if (req.session.user) {

res.render('dashboard');

} else {

res.redirect('/');

}

});

//Students

app.get('/students', (req, res) => {

db.collection('Student').find().toArray()

.then(results => res.render('students', { students: results }));

});

app.post('/students/add', (req, res) => {

db.collection('Student').insertOne(req.body)

.then(() => res.redirect('/students'));

});

app.post('/students/delete/:id', (req, res) => {

const { id } = req.params;

db.collection('Student').deleteOne({ student\_id: id })

.then(() => res.redirect('/students'));

});

// Edit form for a student

app.get('/students/edit/:id', (req, res) => {

const { id } = req.params;

db.collection('Student').findOne({ student\_id: id })

.then(student => {

if (student) {

res.render('edit\_student', { student });

} else {

res.redirect('/students');

}

});

});

//Update Student

app.post('/students/edit/:id', (req, res) => {

const { id } = req.params;

const updatedStudent = {

student\_id: req.body.student\_id,

name: req.body.name,

email: req.body.email,

phone\_no: req.body.phone\_no,

department: req.body.department,

session: req.body.session,

student\_status: req.body.student\_status

};

db.collection('Student').updateOne({ student\_id: id }, { $set: updatedStudent })

.then(() => res.redirect('/students'));

});

// Books

app.get('/books', (req, res) => {

db.collection('Books').find().toArray()

.then(results => res.render('books', { books: results }));

});

app.post('/books/add', (req, res) => {

db.collection('Books').insertOne(req.body)

.then(() => res.redirect('/books'));

});

app.post('/books/delete/:id', (req, res) => {

const { id } = req.params;

db.collection('Books').deleteOne({ book\_id: id })

.then(() => res.redirect('/books'));

});

// Edit form for a book

app.get('/books/edit/:id', (req, res) => {

const { id } = req.params;

db.collection('Books').findOne({ book\_id: id })

.then(book => {

if (book) {

res.render('edit\_book', { book });

} else {

res.redirect('/books');

}

});

});

// Update Book

app.post('/books/edit/:id', (req, res) => {

const { id } = req.params;

const updatedBook = {

book\_id: req.body.book\_id,

title: req.body.title,

author: req.body.author,

language: req.body.language,

cat\_id: req.body.cat\_id,

price: req.body.price,

total\_copies: req.body.total\_copies,

available\_copies: req.body.available\_copies,

book\_status: req.body.book\_status

};

db.collection('Books').updateOne({ book\_id: id }, { $set: updatedBook })

.then(() => res.redirect('/books'));

});

// Borrowing

app.get('/borrow', (req, res) => {

db.collection('Borrow').find().toArray()

.then(results => res.render('borrow', { borrow: results }));

});

app.post('/borrow/add', (req, res) => {

db.collection('Borrow').insertOne(req.body)

.then(() => res.redirect('/borrow'));

});

app.post('/borrow/delete/:id', (req, res) => {

const { id } = req.params;

db.collection('Borrow').deleteOne({ borrow\_id: id })

.then(() => res.redirect('/borrow'));

});

// Edit form for a borrowed book

app.get('/borrow/edit/:id', (req, res) => {

const { id } = req.params;

db.collection('Borrow').findOne({ borrow\_id: id })

.then(borrow => {

if (borrow) {

res.render('edit\_borrow', { borrow });

} else {

res.redirect('/borrow');

}

});

});

// Update Borrowing

app.post('/borrow/edit/:id', (req, res) => {

const { id } = req.params;

const updatedBorrow = {

borrow\_id: req.body.borrow\_id,

student\_id: req.body.student\_id,

book\_id: req.body.book\_id,

issue\_date: req.body.issue\_date,

due\_date: req.body.due\_date

};

db.collection('Borrow').updateOne({ borrow\_id: id }, { $set: updatedBorrow })

.then(() => res.redirect('/borrow'));

});

// Returning

app.get('/return', (req, res) => {

db.collection('Return').find().toArray()

.then(results => res.render('return', { returns: results }));

});

app.post('/return/add', (req, res) => {

db.collection('Return').insertOne(req.body)

.then(() => res.redirect('/return'));

});

app.post('/return/delete/:id', (req, res) => {

const { id } = req.params;

db.collection('Return').deleteOne({ borrow\_id: id })

.then(() => res.redirect('/return'));

});

// Edit form for a returned book

app.get('/return/edit/:id', (req, res) => {

const { id } = req.params;

db.collection('Return').findOne({ borrow\_id: id })

.then(returnedBook => {

if (returnedBook) {

res.render('edit\_return', { returnedBook });

} else {

res.redirect('/return');

}

});

});

// Update Returned book

app.post('/return/edit/:id', (req, res) => {

const { id } = req.params;

const updatedReturn = {

borrow\_id: req.body.borrow\_id,

student\_id: req.body.student\_id,

book\_id: req.body.book\_id,

return\_date: req.body.return\_date,

due\_date: req.body.due\_date,

fine: req.body.fine,

return\_status: req.body.return\_status,

fine\_status: req.body.fine\_status

};

db.collection('Return').updateOne({ borrow\_id: id }, { $set: updatedReturn })

.then(() => res.redirect('/return'));

});

// Category

app.get('/categories', (req, res) => {

db.collection('Category').find().toArray()

.then(results => res.render('categories', { categories: results }));

});

app.post('/categories/add', (req, res) => {

db.collection('Category').insertOne(req.body)

.then(() => res.redirect('/categories'));

});

app.post('/categories/delete/:id', (req, res) => {

const { id } = req.params;

db.collection('Category').deleteOne({ cat\_id: id })

.then(() => res.redirect('/categories'));

});

//Edit form for a category

app.get('/categories/edit/:id', (req, res) => {

const { id } = req.params;

db.collection('Category').findOne({ cat\_id: id })

.then(category => {

if (category) {

res.render('edit\_category', { category });

} else {

res.redirect('/categories');

}

});

});

// Update Category

app.post('/categories/edit/:id', (req, res) => {

const { id } = req.params;

const updatedCategory = {

cat\_id: req.body.cat\_id,

name: req.body.name

};

db.collection('Category').updateOne({ cat\_id: id }, { $set: updatedCategory })

.then(() => res.redirect('/categories'));

});

app.get('/logout', (req, res) => {

req.session.destroy();

res.redirect('/');

});

app.listen(8080, () => {

console.log('Server is running on port 8080');

});

**LIST OF REFERENCES:**

1.  **MongoDB Documentation**
   * The official documentation provides comprehensive information on MongoDB, including installation, queries, aggregation, and more.
   * [MongoDB Documentation](https://www.mongodb.com/docs/)
2. **MongoDB University**
   * Free online courses offered by MongoDB, ranging from beginner to advanced levels.
   * [MongoDB University](https://university.mongodb.com/)
3. **MongoDB Manual**
   * Detailed manual covering all aspects of MongoDB, including tutorials and reference materials.
   * [MongoDB Manual](https://www.mongodb.com/docs/manual/)
4. **The Little MongoDB Book**
   * A concise and free book that offers an introduction to MongoDB.
   * [The Little MongoDB Book](https://github.com/karlseguin/the-little-mongodb-book)
5. **Mongoose Documentation**
   * If you’re using Mongoose, this is the go-to resource for schemas, models, and more.
   * Mongoose Documentation

**Node.js**

1. **Node.js Documentation**
   * The official Node.js documentation is a comprehensive resource for learning about Node.js APIs and features.
   * Node.js Documentation
2. **Express.js Documentation**
   * The official guide for Express.js, a popular Node.js web framework.
   * Express.js Documentation
3. **Node.js Design Patterns Book**
   * A book that dives deep into Node.js design patterns and best practices.
   * [Node.js Design Patterns](https://www.nodejsdesignpatterns.com/)
4. **You Don’t Know Node.js Yet**
   * A free online book series that provides an in-depth understanding of Node.js.
   * [You Don’t Know Node.js Yet](https://github.com/getify/You-Dont-Know-JS)
5. **The Node Beginner Book**
   * A beginner-friendly book that introduces Node.js and how to use it effectively.
   * [The Node Beginner Book](https://www.nodebeginner.org/)