COLLEGE CODE : 9103

COLLEGE NAME : chendhuran college of engineering and

technology

DEPARTMENT :Computer science

PROJECT NAME :

E-Library with secure Pdf access

SUBMITTED BY,
NAME: GURU.R (Team Leader)
STUDENT NM ID: 7F42E92A638F36097B14325442645DEA
NAME : JEEVA K
STUDENT NM ID:1AC297F51649B47A8E44A5FCBF26FDFB
NAME : GOKUL.M
STUDENT NM ID :06E156CB9A4E3628EDC4AD4B75CD3867
NAME : MUTHUKUMARAN.M
STUDENT NM ID: 1AEFB58ADC071FFAD9141F534C8E8A39
NAME : ARUN.M
STUDENT NM ID:E47A8AC69C957B196DC4576A2B3A6A06

E-Library with Secure PDF Access

Project Overview & Objectives:

The E-Library with Secure PDF Access is a modern, web-based platform designed to provide users with a seamless and secure digital reading experience. The core problem it addresses is the challenge of distributing and accessing digital books (primarily PDFs) in a controlled manner, preventing unauthorized copying, redistribution, and ensuring that content is only available to verified users.

Key features will include:

User Authentication & Authorization:Secure user registration and login.

* Role-Based Access Control (RBAC):Differentiating between Admin users (who can upload and manage books) and Members (who can read books).

* Digital Library Catalog: A searchable and categorized collection of books with details like title, author, description, and cover

image.

* Secure PDF Viewer:Integrated PDF viewer that prevents easy downloading and printing. PDFs will be served dynamically rather than via direct, static links.

* Reading Progress Tracking:Allows users to bookmark their last read page and resume from there.

* Admin Dashboard: A dedicated interface for admins to upload new books, manage existing ones, and view user activity.

The expected outcome is a fully functional, secure, and user- friendly web application that can be deployed by educational institutions, small libraries, or organizations to manage and share their digital document collections confidently.

Phase 1: Planning & Foundation

1. Technology Stack & Environment Setup

* Backend:
    * Runtime:Node.js

    * Framework: Express.js for building robust and scalable RESTful APIs.
    * Authentication:JSON Web Tokens (JWT) for stateless and secure user sessions.
    * Password Hashing:bcryptjs to securely hash user passwords before storing them.
* Database:
    * System: MongoDB with Mongoose ODM. Its flexible schema is ideal for storing book metadata and user profiles. Collections will include `Users`, `Books`, and `ReadingSessions`.
* Frontend:
    * Framework:React.js (with Vite for fast setup) to create a dynamic and responsive single-page application (SPA).
    * State Management: React Context API or Redux Toolkit for managing global state like user authentication and the list of books.
    * UI Library:Tailwind CSS for rapid and modern UI development.
    * PDF Viewer: Mozilla's `PDF.js` integrated into a React component to display PDFs securely without exposing the original file URL.
* Tools & Services:
    * Version Control:Git with a repository on GitHub.
    * Package Manager:npm or yarn.

* File Upload:Multer` middleware for handling PDF and cover image uploads.

* Deployment: Backend on Render or Railway, Frontend on Netlify or Vercel, and MongoDB Atlas for the database.

2. API Design & Data Model

Data Models (Schema):

* User: `{ _id, email, passwordHash, role ("admin" / "member"), createdAt }`

* Book: `{ _id, title, author, description, genre, coverImageUrl, pdfUrl, uploadDate, uploadedBy (ref to User) }`

* *ReadingSession: `{ _id, userId (ref to User), bookId (ref to Book), lastPage, updatedAt }`

Planned REST Endpoints:

* Authentication Routes:
    * `POST /api/auth/register` - Create a new member account.
    * `POST /api/auth/login` - Authenticate user and return a JWT.
* Book Routes (Protected):

    * `GET /api/books` - Get a paginated list of all books (search/filterable).
    * `GET /api/books/:id` - Get details of a single book.

* Admin Routes (Protected & Admin-only):
    * `POST /api/admin/books` - Upload a new book (cover image + PDF).
    * `PUT /api/admin/books/:id` - Update book metadata.
    * `DELETE /api/admin/books/:id` - Remove a book from the library.

* Reading Routes (Protected):
    * `GET /api/read/:bookId` - Securely serve the PDF file for the integrated viewer. This endpoint will verify the user's JWT before streaming the file.
    * `POST /api/read/progress` - Save or update the user's current page in a book.

3. Front-End UI/UX Plan

*Wireframes & Navigation Flow:

1. Public Landing Page: Showcases the library, with calls-to- action for login and register.
2. *Login/Registration Pages:Simple forms for user access.

3. Member Dashboard:Upon login, users see a searchable grid of book cards. A navigation bar provides links to the catalog and a logout button.

4. *Book Detail Page:Clicking a book card opens a page with the book's description, author, and a "Start Reading" button.

5. Reading Page:A clean, focused interface with the PDF.js viewer. The navigation is minimal, and the browser's right-click "Save As" functionality will be disabled for the PDF iframe.

6. *Admin Dashboard:A separate view for admins with a form to upload books and a table to manage existing ones.

* State Management Approach:The React Context API will be used to create an `AuthContext` to manage the global user state (user info, JWT token, login status). A `BookContext` may be used to cache the library catalog for faster navigation.

4. Development & Deployment Plan

* Team Roles:

  * Backend Developer:Focuses on Node.js/Express API, database models, authentication, and secure file delivery.

  * Frontend Developer: Implements the React UI, integrates with the API, and builds the secure PDF viewer component.

  * UI/UX Designer (can be shared role):** Creates wireframes and ensures a consistent design system with Tailwind CSS.

* Git Workflow:
    * Use a feature-branch workflow. `main` branch will always hold the production-ready code.
    * New features will be developed in branches named `feature/description` (e.g., `feature/user-authentication`).
    * Pull Requests (PRs) are required to merge a feature branch into `main`, ensuring code review.
* Testing Approach:
    * Backend: Use Jest and Supertest for unit and integration testing of API endpoints.
    * Frontend:Use React Testing Library for component testing.
    * Manual Testing:Thoroughly test user flows like registration, login, book upload, and the secure reading experience.
* Hosting & Deployment Strategy:
    1. Database: Set up a production cluster on MongoDB Atlas.
    2. Backend:Deploy the Node.js/Express API to Render or Railway, connecting it to the MongoDB Atlas database.
    3. Frontend: Build the React app into static files and deploy them on Netlify or Vercel. The frontend will be configured to communicate with the deployed backend API URL.

This comprehensive Phase 1 plan provides a solid blueprint for developing a robust and secure E-Library, ensuring the team is

aligned on the vision, technology, and execution path before a single line of code is written.

1. Technology Stack & Data Model Examples

A. Database Schema Examples (MongoDB with Mongoose)

```
// User Model (models/User.js)

const userSchema = new mongoose.Schema({ email: { type: String, required: true,
  unique: true }, passwordHash: { type: String, required: true },
  role: { type: String, enum: ['member', 'admin'], default: 'member' }, createdAt: { type: Date, default: Date.now }
});


// Book Model (models/Book.js)

const bookSchema = new mongoose.Schema({ title: { type: String, required: true },
  author: { type: String, required: true }, description: String,
  genre: [String], // e.g., ["Fiction", "Science"]

  coverImageUrl: String, // e.g., "/uploads/covers/the-hobbit.jpg" pdfUrl: { type: String, required: true }, // e.g.,
  "/uploads/pdfs/the-
```

```
hobbit.pdf"
  uploadedBy: { type: mongoose.Schema.Types.ObjectId, ref: 'User'
 },
  uploadDate: { type: Date, default: Date.now }
});

// ReadingSession Model (models/ReadingSession.js) const readingSessionSchema = new
mongoose.Schema({
  userId: { type: mongoose.Schema.Types.ObjectId, ref: 'User', required: true },
  bookId: { type: mongoose.Schema.Types.ObjectId, ref: 'Book', required: true },
  lastPage: { type: Number, default: 1 }, updatedAt: { type: Date, default:
  Date.now }
});
```

B. Example JWT Token Payload

When a user logs in, the server creates a JWT token with a payload like this:

```
{
  "userId": "507f1f77bcf86cd799439011", "email": "user@example.com",
```

"role": "member",

"iat": 1719500000, // Issued at timestamp "exp": 1719503600 // Expires in 1 hour

}

2. API Endpoint Examples

A. User Registration Request & Response

Request:

POST /api/auth/register Content-Type: application/json

{

"email": "newuser@example.com", "password": "mySecurePassword123"

}

Success Response (201 Created):

json

{

"message": "User registered successfully",

```
  "user": {
    "id": "507f1f77bcf86cd799439011",
    "email": "newuser@example.com", "role": "member"
  }
}
```


C. Secure PDF Serving Endpoint

Backend Logic (pseudo-code for the route handler):// routes/read.js

```
app.get('/api/read/:bookId', authenticateUser, async (req, res) => { try {
    const book = await Book.findById(req.params.bookId);
    if (!book) return res.status(404).json({ message: 'Book not found' });

    // Check if user has permission (is logged in)
    // The `authenticateUser` middleware already verified the JWT
```

```
    // Set headers to prevent caching and discourage saving
    res.setHeader('Content-Type', 'application/pdf'); res.setHeader('Cache-Control', 'no-cache, no-
    store, must-
revalidate');

    res.setHeader('Pragma', 'no-cache');


    // Stream the PDF file to the client

    const filePath = path.join(_____dirname, '..', book.pdfUrl); const fileStream

    = fs.createReadStream(filePath); fileStream.pipe(res);


  } catch (error) {

    res.status(500).json({ message: 'Error serving book' });

  }

});
```

3. Front-End UI/UX Examples

A. Book Card Component (React)

```
// components/BookCard.jsx import { Link } from 'react-router-dom';


const BookCard = ({ book }) => {
```

```
 return (
<div className="bg-white rounded-lg shadow-md overflow- hidden hover:shadow-lg transition-shadow">
  <img src={book.coverImageUrl} alt={`Cover of
    ${book.title}`}
    className="w-full h-48 object-cover"
   />
  <div className="p-4">
    <h3 className="text-xl font-semibold mb- 2">{book.title}</h3>
    <p className="text-gray-600 mb-2">by {book.author}</p>
    <div className="flex flex-wrap gap-1 mb-3">
      {book.genre.map((tag, index) => (
        <span key={index} className="bg-blue-100 text-blue-800 text-xs px-2 py-1 rounded">
          {tag}
        </span>
      ))}
    </div>
    <Link
```

```
            to={`/book/${book._id}`}
            className="bg-blue-500 hover:bg-blue-600 text-white px-4 py-2 rounded block text-center"
          >
            View Details
          </Link>
        </div>
      </div>
    );
};
```

B. Secure PDF Viewer Component

```
// components/PDFReader.jsx

import { useState, useEffect } from 'react'; import { Document, Page, pdfjs } from
'react-pdf';
import 'react-pdf/dist/Page/AnnotationLayer.css';


// Configure PDF.js worker pdfjs.GlobalWorkerOptions.workerSrc =
`//cdnjs.cloudflare.com/ajax/libs/pdf.js/${pdfjs.version}/pdf.work er.min.js`;
```

```
const PDFReader = ({ bookId }) => {
  const [numPages, setNumPages] = useState(null); const [pageNumber, setPageNumber] =
  useState(1);
  const [lastSavedPage, setLastSavedPage] = useState(1);

  // Secure PDF URL - points to our backend endpoint, not the direct file
  const pdfUrl = `/api/read/${bookId}`;

  function onDocumentLoadSuccess({ numPages }) { setNumPages(numPages);
    // Here you would fetch the user's last saved page from the API
    // and setPageNumber to that value
  }

  // Auto-save progress when page changes useEffect(() => {
    const saveProgress = setTimeout(() => { if (pageNumber !==
      lastSavedPage) {
```

```
        fetch('/api/read/progress', { method: 'POST',
          headers: {
            'Content-Type': 'application/json',
            'Authorization': `Bearer ${localStorage.getItem('token')}`
          },
          body: JSON.stringify({ bookId: bookId,
            lastPage: pageNumber
          })
        });
        setLastSavedPage(pageNumber);
      }
    }, 1000);

    return () => clearTimeout(saveProgress);
  }, [pageNumber, bookId, lastSavedPage]);

  return (
    <div className="pdf-viewer">
```

```
<div className="pdf-controls bg-gray-100 p-2 flex justify- between items-center">
  <button
    onClick={() => setPageNumber(prev => Math.max(prev - 1,
1))}
                                                    disabled={pageNumber <= 1}
                                                    className="bg-blue-500 text-white px-4 py-2 rounded
disabled:bg-gray-300"
  >
    Previous
  </button>

  <span className="text-gray-700"> Page {pageNumber} of
    {numPages}
  </span>

  <button
    onClick={() => setPageNumber(prev => Math.min(prev + 1, numPages))}
    disabled={pageNumber >= numPages} className="bg-blue-500 text-white px-4 py-2
    rounded
```

```
disabled:bg-gray-300"
          >
            Next
          </button>
        </div>

        <div className="pdf-container border">
          <Document
            file={pdfUrl}
            onLoadSuccess={onDocumentLoadSuccess} options={{
              httpHeaders: {
                'Authorization': `Bearer ${localStorage.getItem('token')}`
              }
            }}
          >
            <Page pageNumber={pageNumber}
              renderTextLayer={false} // Makes text selection harder
            />
```

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

C. Admin Book Upload Form

```
// components/AdminUploadForm.jsx const AdminUploadForm = () => {
  const [formData, setFormData] = useState({ title: '',
    author: '', description: '',
    genre: '', coverImage: null,
    pdfFile: null
  });

  const handleSubmit = async (e) => { e.preventDefault();
    const submitData = new FormData();
```

```
submitData.append('title', formData.title); submitData.append('author', formData.author);
submitData.append('description', formData.description); submitData.append('genre', formData.genre);
submitData.append('coverImage', formData.coverImage); submitData.append('pdfFile', formData.pdfFile);

try {
  const response = await fetch('/api/admin/books', { method: 'POST',
    headers: {
      'Authorization': `Bearer ${localStorage.getItem('token')}`
    },
    body: submitData
  });

  if (response.ok) {
    alert('Book uploaded successfully!');
    // Reset form
    setFormData({ title: '', author: '', description: '', genre: '',
```

```
coverImage: null, pdfFile: null });
        }
      } catch (error) {
        alert('Error uploading book');
      }
  };

  return (
    <form onSubmit={handleSubmit} className="space-y-4 max-w
-2xl mx-auto">
        <input type="text"
          placeholder="Book Title" value={formData.title}
          onChange={(e) => setFormData({...formData, title: e.target.value})}
          className="w-full p-2 border rounded" required
        />

        <input
```

```
   type="text" placeholder="Author"
   value={formData.author}
   onChange={(e) => setFormData({...formData, author: e.target.value})}
   className="w-full p-2 border rounded" required
/>

<textarea placeholder="Description"
   value={formData.description}
   onChange={(e) => setFormData({...formData, description: e.target.value})}
   className="w-full p-2 border rounded" rows="3"
/>

<input type="file"
   accept="image/*"
   onChange={(e) => setFormData({...formData, coverImage: e.target.files[0]})}
```

```
        className="w-full p-2 border rounded" required
      />

      <input type="file"
        accept=".pdf"
        onChange={(e) => setFormData({...formData, pdfFile: e.target.files[0]})}
        className="w-full p-2 border rounded" required
      />

      <button type="submit"
        className="bg-green-500 text-white px-6 py-2 rounded hover:bg-green-600"
      >
        Upload Book
      </button>
    </form>
  );
```

```
};
```