TEAM ID	NM2023TMID04391
PROJECT TITTLE	BLOCKCHAIN POWERED LIBRARY MANAGEMENT

USING FILE CONNECTOR. JS

INTRODUCTION:

- A Library Management System is basically used to manage the book record of a Library.
- where a librarian can view all the available books, add new books, delete books, issue books, and return books.

Functionalities: A librarian can do the following things with this Library Management System:

- Display Available Books
- Add New Books
- Delete Books
- Issue Books
- Return Books

Approach:

- We are going to use Body Parser by which we can capture user input values from the form such as Book Name, Author Name, Pages and Price & store them in a collection.
- Then we are going to send the data of the book to the web page using EJS. EJS is a middleware that makes it easy to send data from your server file (app.js or server.js) to a page.

- Weare also going to create the Delete Route for deleting books, an Issue Route for issuing the books, and a Return Route for returning the books.

Step 1:

Project Setup:

Initializes NPM: Create and Locate your project folder into the terminal & type the command

```
npm init -y
```

Install Dependencies: Locate your root project directory into the terminal and type the command

```
npm install express ejs body-parser
```

To install Express, EJS, and Body Parser as dependencies inside your project

Create Server File:

- Create an ‘app.js’ file, inside this file require the Express Module, and create a constant ‘app’ for creating an instance of the express module, then set the EJS as the default view engine.

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


const app = express();
```

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

Rearrange Your Directories:

- It is required to use ‘.ejs’ as an extension for the HTML file instead of ‘.html’ for using EJS inside it.

- Then you have to move every ‘.ejs’ file in the views directory inside your root directory. EJS is by default looking for ‘.ejs’ files inside the views folder.
- **Use EJS variable**: Inside your updated .ejs file, you have to use EJS Variables to receive values from your server file. You can declare variables in EJS like

```
<%= variableName %>
```

JAVASCRIPT:

```
const express = require('express')
const app = express()
app.set('view engine', 'ejs')

app.get("/", (req, res) => {
    res.render("home", { variableName: "Hello Geeks!" })
})

app.listen(3000, (req, res) => {
    console.log("App is running on port 3000")
})
```

Fetching data from form to app.js: To receive input values of a form, we have to use a node package named body-parser.

Install body-parser:

install body-parser

Require body-parser module:

```
const bodyParser = require('body-parser'
```

Step 2:

Fetch Available Books:

- We have an array of books with different properties. Let's send the array to our web page.
- In the previous step, we just sent a value to the variable, now we are sending the complete array.

```
const express = require('express')
const bodyParser = require('body-parser')
const books = [{
        bookName: "Rudest Book Ever",
        bookAuthor: "Shwetabh Gangwar",
        bookPages: 200,
        bookPrice: 240,
        bookState: "Available"
    },
    {
        bookName: "Do Epic Shit",
        bookAuthor: "Ankur Wariko",
        bookPages: 200,
        bookPrice: 240,
        bookState: "Available"
    }
]

const app = express()
```

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

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

app.get("/", function (req, res) {
        res.render("home", {
                data: books
        })
})

app.listen(3000, (req, res) => {
        console.log("App is running on port 3000")
})
```

Step 3:

Add Books to the list:

For this, we have to create a form and handle the form data inside our 'app.js' file using Body Parser.

```
<form action="/" method="post">
      <input type="text" placeholder="Book Name"
            name="bookName">
      <input type="text" placeholder="Book Author"
            name="bookAuthor">
      <input type="text" placeholder="Book Pages"
            name="bookPages">
      <input type="text" placeholder="Book Price"
            name="bookPrice">
      <button type="submit">Add</button>
</form>
```

Handle form data inside ‘app.js’: We have to fetch values from a form using req.body.valueName, and then arrange it like an object and push it inside our book’s array.

```
app.post("/", (req, res) => {
  const inputBookName = req.body.bookName
  const inputBookAuthor = req.body.bookAuthor
  const inputBookPages = req.body.bookPages
  const inputBookPrice = req.body.bookPrice

  books.push({
     bookName: inputBookName,
     bookAuthor: inputBookAuthor,
     bookPages: inputBookPages,
     bookPrice: inputBookPrice,
     bookState: "Available"
  })

  res.render("home", {
     data: books
  })
})
```

Step 4:

Issue Books:

Updating Web Page giving an issue option: We have to create a form that sends the book name which we want to issue to the server file ‘app.js’.

```
<form action="/issue" method="post">
    <input type="text" style="display: none;"
          name="bookName" value="<%= element.bookName
%>">
    <button type="submit">Issue</button>
```

```
</form>
```

For issuing a book, we have to create an issue route where we are going to fetch the requested book's name and search for the element which has the same book name, and change the element state property to Issued.

```
app.post('/issue', (req, res) => {
  var requestedBookName = req.body.bookName;
  books.forEach(book => {
    if (book.bookName == requestedBookName) {
      book.bookState = "Issued";
    }
  })
  res.render("home", {
    data: books
  })
})
```

Step 5:

Return Books:

Updating Web Page giving a return option: We have to create a form that sends the book name which we want to return to the server file 'app.js'.

```
<form action="/return" method="post">
        <input type="text" style="display: none;"
              name="bookName" value="<%= element.bookName
%>">
        <button type="submit">Return</button>
</form>
```

For returning a book, we have to create a return route where we are going to fetch the requested book's name and search for the element which has the same book name, and change the element state property to Available.

```
app.post('/return', (req, res) => {
  var requestedBookName = req.body.bookName;
  books.forEach(book => {
     if (book.bookName == requestedBookName) {
        book.bookState = "Available";
     }
  })
  res.render("home", {
     data: books
  })
})
```

Step 6:

Delete Books:

Updating Web Page giving a delete option: We have to create a form that sends the book name which we want to delete to the server file ‘app.js’.

```
<form action="/delete" method="post">
    <input type="text" style="display: none;"
          name="bookName" value="<%= element.bookName
%>">
    <button type="submit">Delete</button>
</form>
```

For deleting a book, we have to create a delete route where we are going to fetch the requested book’s name and search for the element which has the same book name, and delete the element.

```
app.post('/delete', (req, res) => {
  var requestedBookName = req.body.bookName;
  var j = 0;
```

```
    books.forEach(book => {
        j = j + 1;
        if (book.bookName == requestedBookName) {
            books.splice((j - 1), 1)
        }
    })

    res.render("home", {
        data: books
    })
})
```