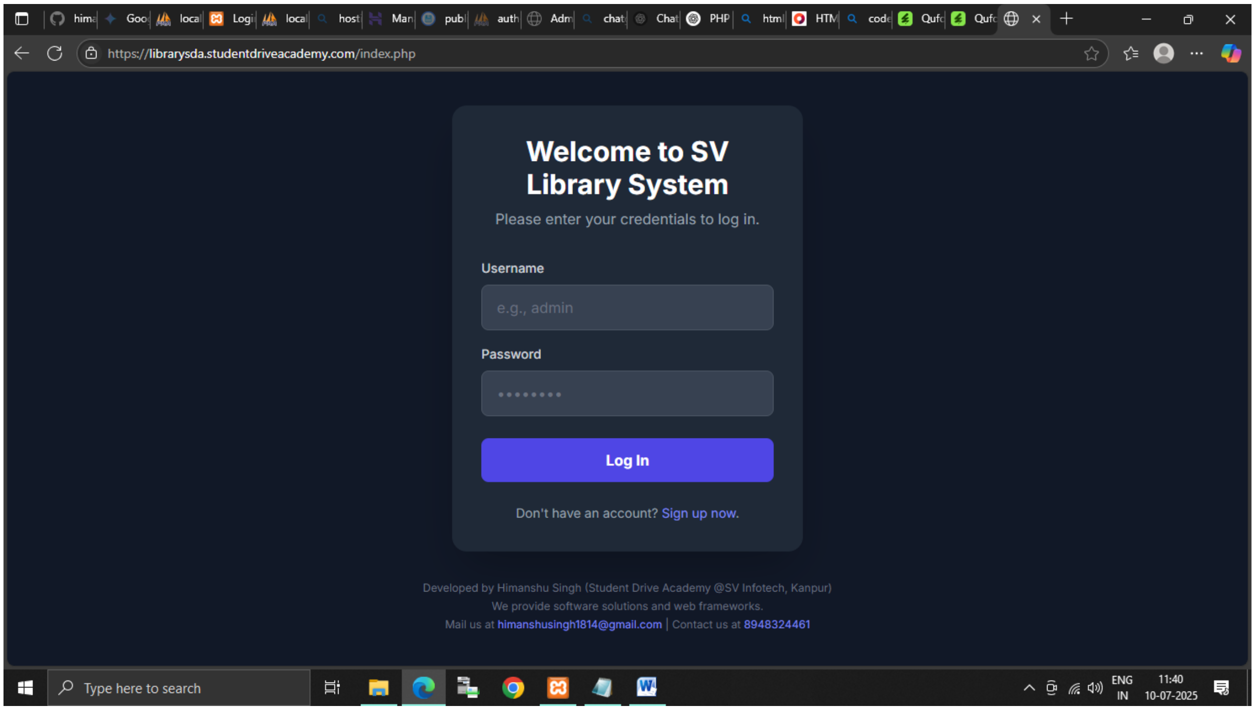
LIBRARY MANAGER WEB APPLICATION VIA PHP

PROJECT UI

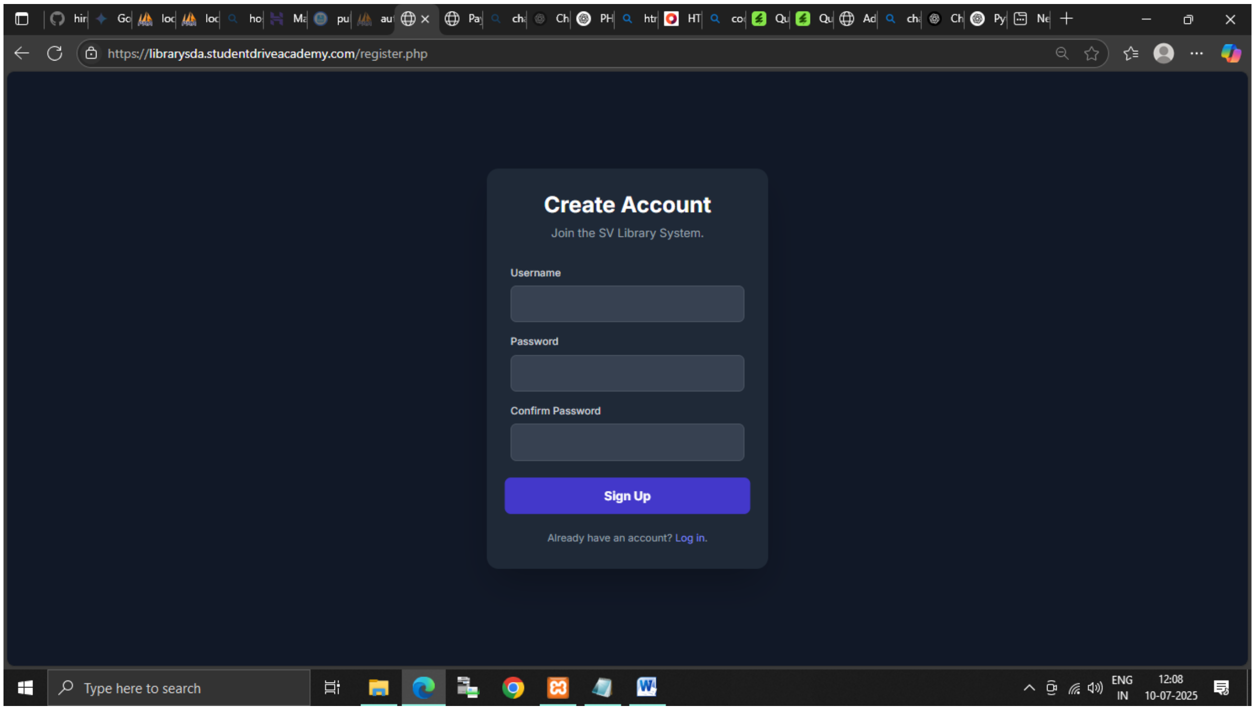


Welcome to SV Library System

Please enter your credentials to log in.

Don't have an account? Sign up now.

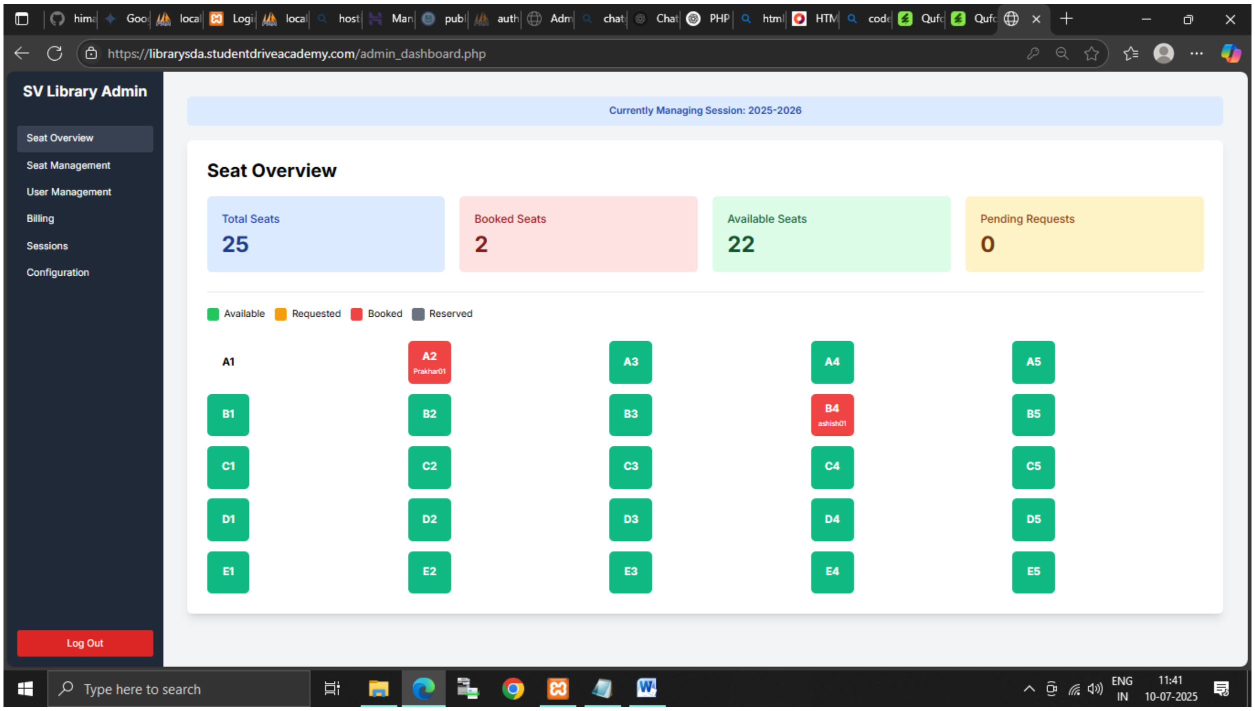
Developed by Himanshu Singh (Student Drive Academy @SV Infotech, Kanpur)
We provide software solutions and web frameworks.
Mail us at himanshusingh1814@gmail.com | Contact us at 8948324461



Create Account

Join the SV Library System.

Already have an account? Log in.



https://librarysda.studentdriveacademy.com/admin_dashboard.php

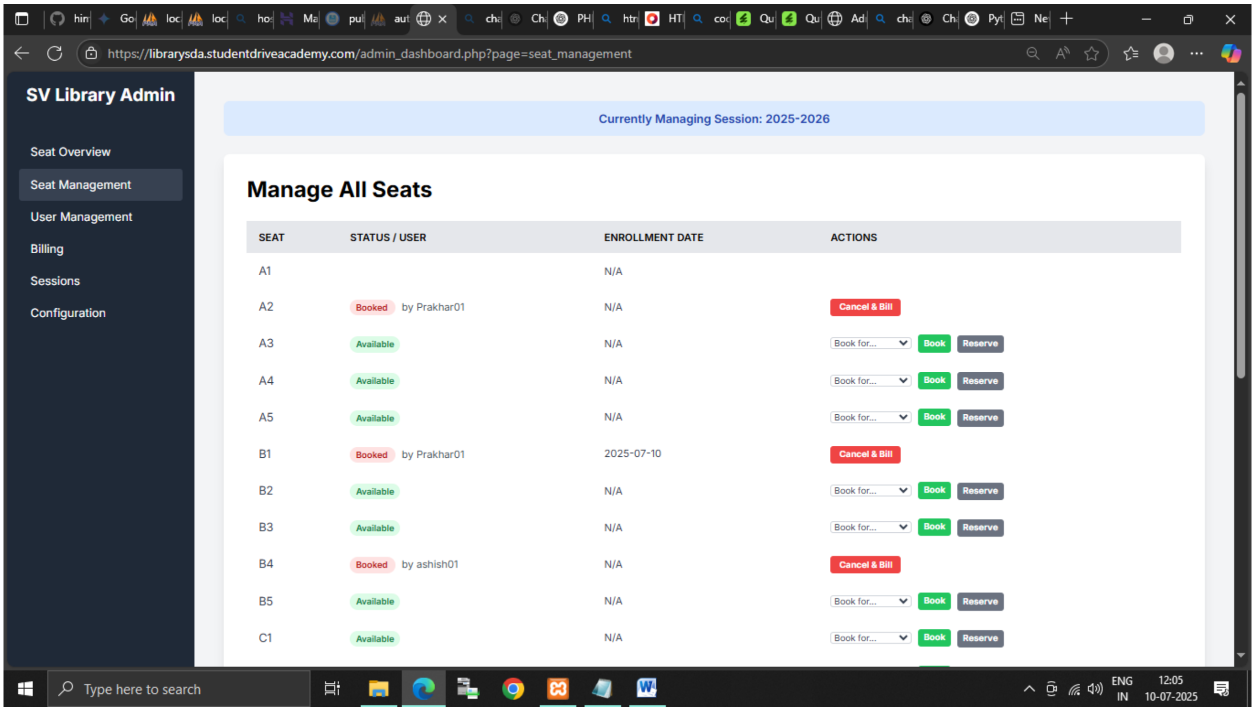
Currently Managing Session: 2025-2026

Seat Overview

Total Seats	Booked Seats	Available Seats	Pending Requests
25	2	22	0

🟩 Available 🟧 Requested 🟥 Booked ⬛ Reserved

A1	A2 Prakhar01	A3	A4	A5
B1	B2	B3	B4 ashish01	B5
C1	C2	C3	C4	C5
D1	D2	D3	D4	D5
E1	E2	E3	E4	E5

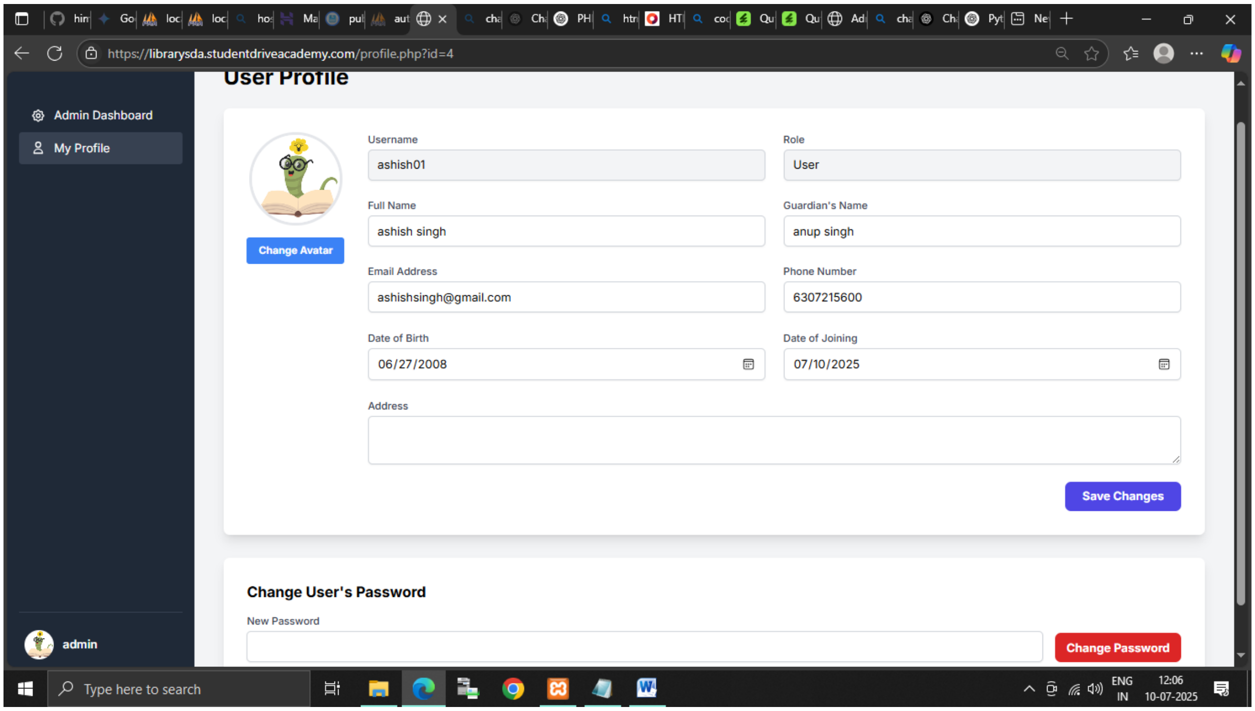


https://librarysda.studentdriveacademy.com/admin_dashboard.php?page=seat_management

Currently Managing Session: 2025-2026

Manage All Seats

SEAT	STATUS / USER	ENROLLMENT DATE	ACTIONS
A1		N/A	
A2	Booked by Prakhar01	N/A	Cancel & Bill
A3	Available	N/A	Book for... Book Reserve
A4	Available	N/A	Book for... Book Reserve
A5	Available	N/A	Book for... Book Reserve
B1	Booked by Prakhar01	2025-07-10	Cancel & Bill
B2	Available	N/A	Book for... Book Reserve
B3	Available	N/A	Book for... Book Reserve
B4	Booked by ashish01	N/A	Cancel & Bill
B5	Available	N/A	Book for... Book Reserve
C1	Available	N/A	Book for... Book Reserve



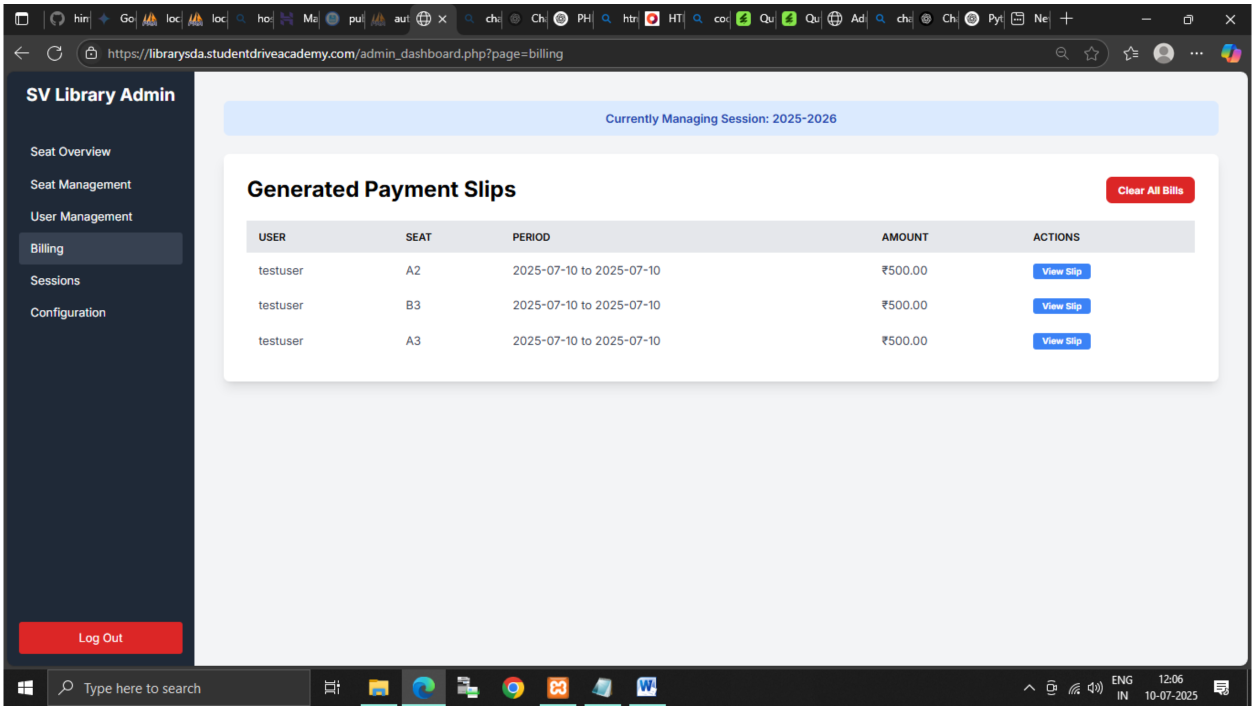
User Profile

Username: ashish01	Role: User
Full Name: ashish singh	Guardian's Name: anup singh
Email Address: ashishsingh@gmail.com	Phone Number: 6307215600
Date of Birth: 06/27/2008	Date of Joining: 07/10/2025

Address

Change User's Password

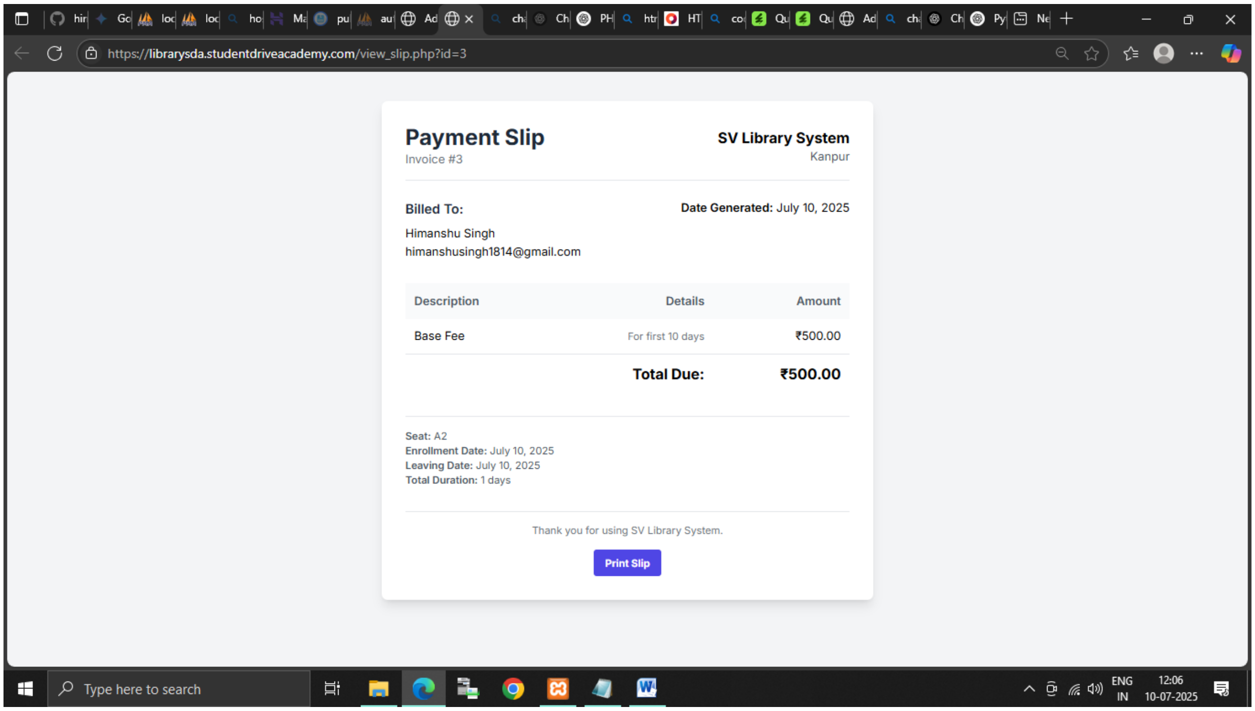
New Password



Currently Managing Session: 2025-2026

Generated Payment Slips

USER	SEAT	PERIOD	AMOUNT	ACTIONS
testuser	A2	2025-07-10 to 2025-07-10	₹500.00	View Slip
testuser	B3	2025-07-10 to 2025-07-10	₹500.00	View Slip
testuser	A3	2025-07-10 to 2025-07-10	₹500.00	View Slip


https://librarysda.studentdriveacademy.com/view_slip.php?id=3
Type here to search
ENG
IN
12:06
10-07-2025

Payment Slip

Invoice #3

SV Library System
Kanpur

Billed To:

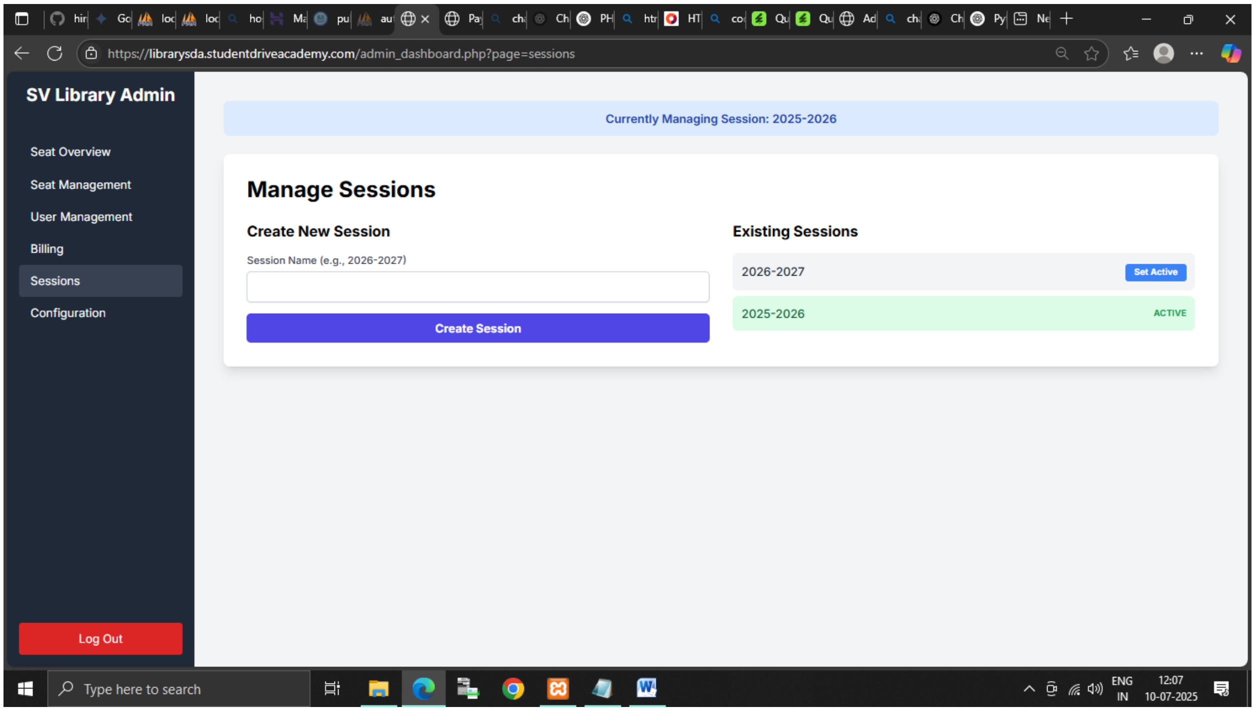
Himanshu Singh
himanshusingh1814@gmail.com

Date Generated: July 10, 2025

Description	Details	Amount
Base Fee	For first 10 days	₹500.00
	Total Due:	**₹500.00**

Seat: A2
Enrollment Date: July 10, 2025
Leaving Date: July 10, 2025
Total Duration: 1 days

Thank you for using SV Library System.



Currently Managing Session: 2025-2026

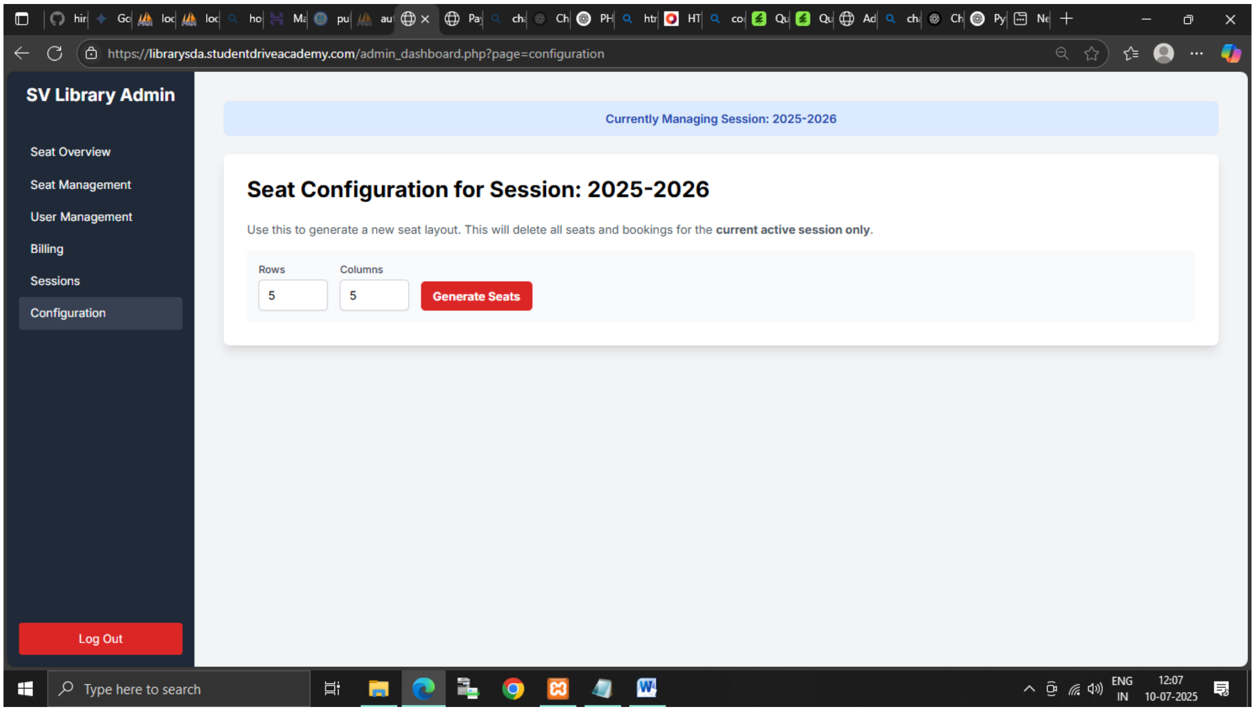
Manage Sessions

Create New Session

Session Name (e.g., 2026-2027)

Existing Sessions

2026-2027	Set Active
2025-2026	ACTIVE



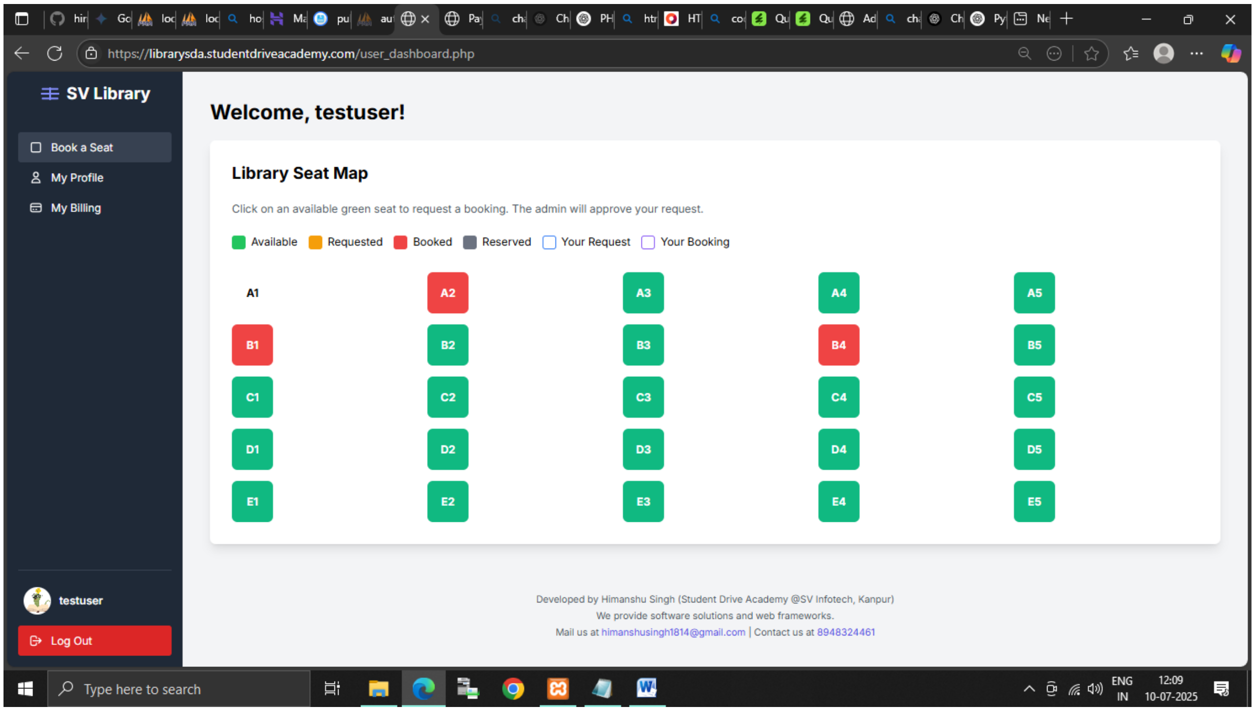
https://librarysda.studentdriveacademy.com/admin_dashboard.php?page=configuration

Currently Managing Session: 2025-2026

Seat Configuration for Session: 2025-2026

Use this to generate a new seat layout. This will delete all seats and bookings for the **current active session only**.

Rows: 5
Columns: 5



https://librarysda.studentdriveacademy.com/user_dashboard.php

Welcome, testuser!

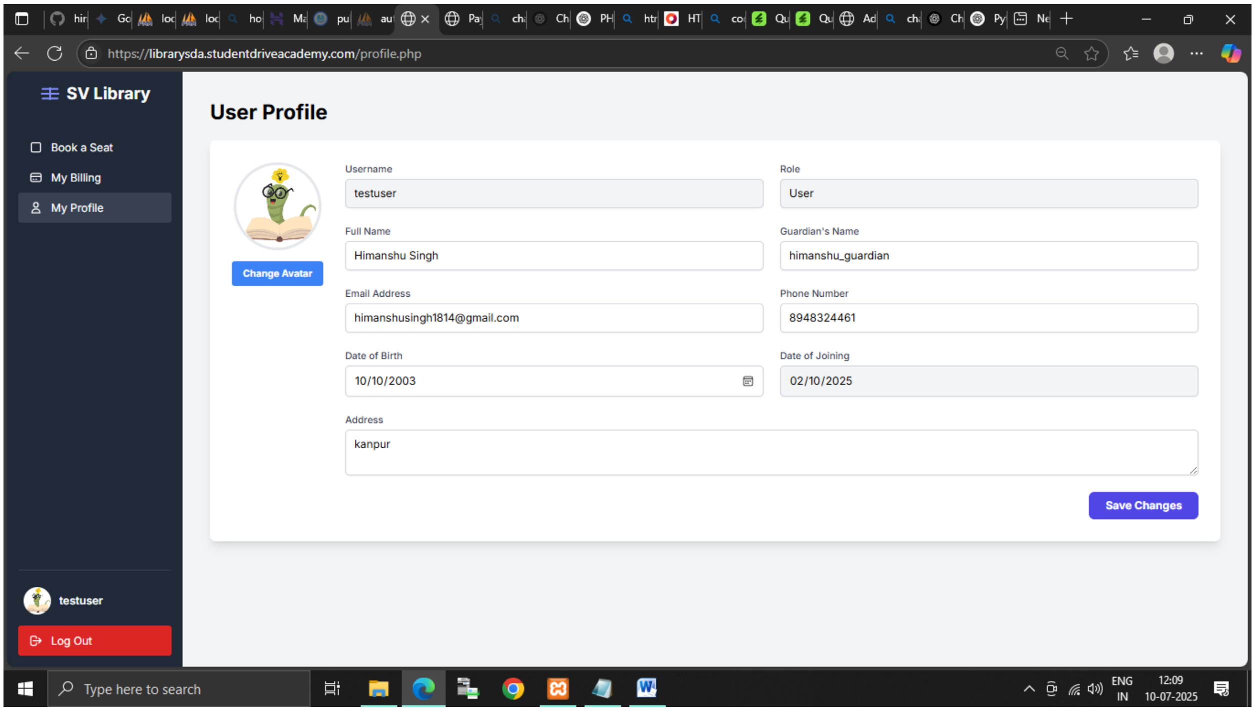
Library Seat Map

Click on an available green seat to request a booking. The admin will approve your request.

🟩 Available 🟧 Requested 🟥 Booked ⬛ Reserved □ Your Request □ Your Booking

A1	🟥 A2	🟩 A3	🟩 A4	🟩 A5
🟥 B1	🟩 B2	🟩 B3	🟥 B4	🟩 B5
🟩 C1	🟩 C2	🟩 C3	🟩 C4	🟩 C5
🟩 D1	🟩 D2	🟩 D3	🟩 D4	🟩 D5
🟩 E1	🟩 E2	🟩 E3	🟩 E4	🟩 E5

Developed by Himanshu Singh (Student Drive Academy @SV Infotech, Kanpur)
We provide software solutions and web frameworks.
Mail us at himanshusingh1814@gmail.com | Contact us at 8948324461



https://librarysda.studentdriveacademy.com/profile.php

User Profile

Username: testuser

Role: User

Full Name: Himanshu Singh

Guardian's Name: himanshu_guardian

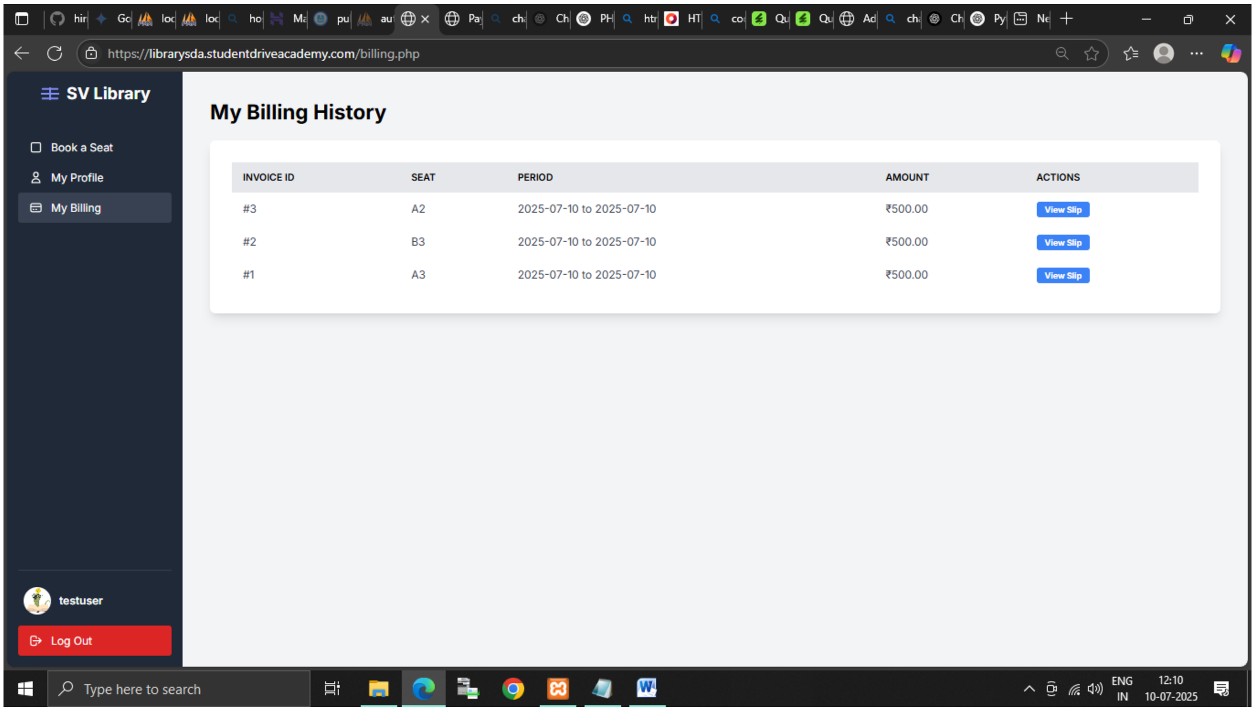
Email Address: himanshusingh1814@gmail.com

Phone Number: 8948324461

Date of Birth: 10/10/2003

Date of Joining: 02/10/2025

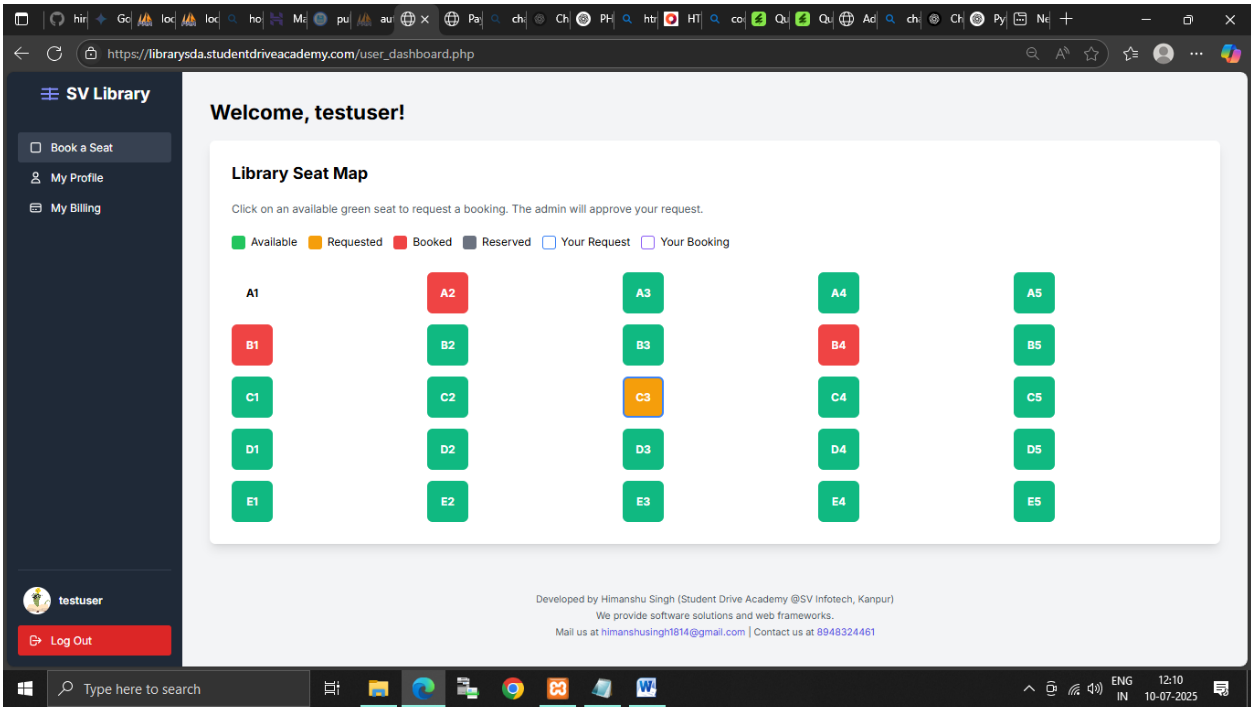
Address: kanpur



https://librarysda.studentdriveacademy.com/billing.php

My Billing History

INVOICE ID	SEAT	PERIOD	AMOUNT	ACTIONS
#3	A2	2025-07-10 to 2025-07-10	₹500.00	View Slip
#2	B3	2025-07-10 to 2025-07-10	₹500.00	View Slip
#1	A3	2025-07-10 to 2025-07-10	₹500.00	View Slip



Welcome, testuser!

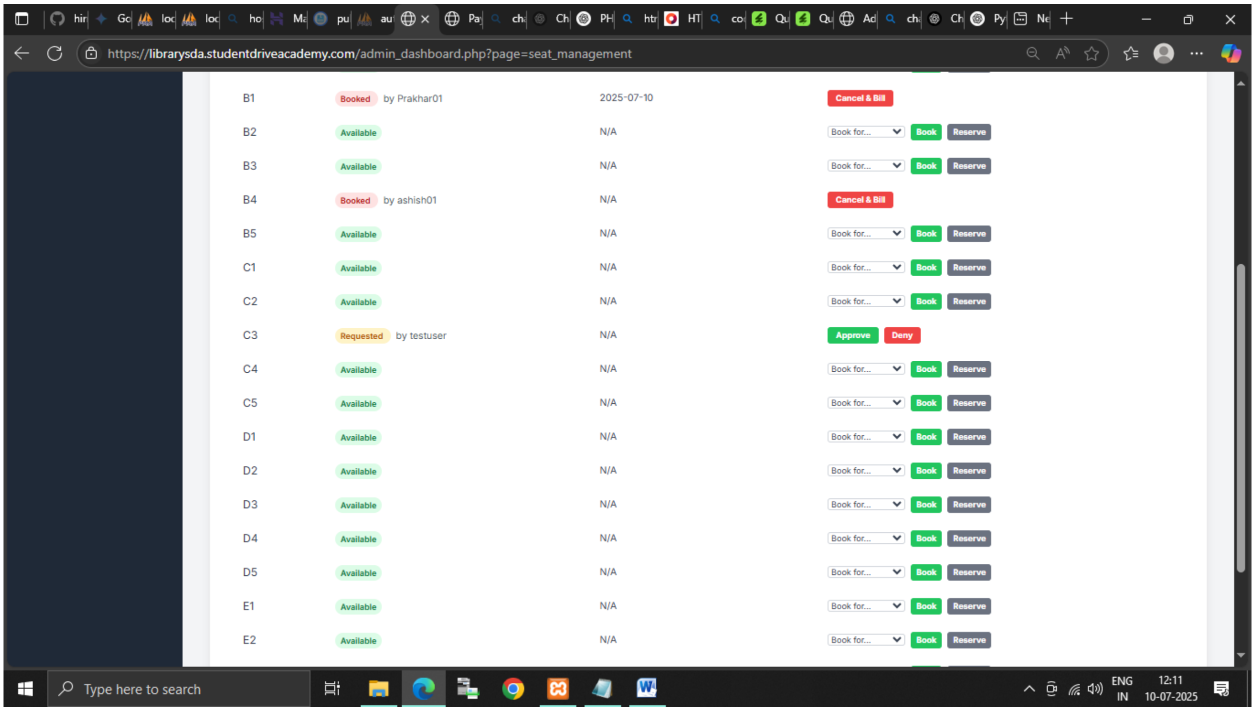
Library Seat Map

Click on an available green seat to request a booking. The admin will approve your request.

🟩 Available 🟧 Requested 🟥 Booked ⬛ Reserved □ Your Request □ Your Booking

A1	A2	A3	A4	A5
B1	B2	B3	B4	B5
C1	C2	C3	C4	C5
D1	D2	D3	D4	D5
E1	E2	E3	E4	E5

Developed by Himanshu Singh (Student Drive Academy @SV Infotech, Kanpur)
We provide software solutions and web frameworks.
Mail us at himanshusingh1814@gmail.com | Contact us at 8948324461


https://librarysda.studentdriveacademy.com/admin_dashboard.php?page=seat_management
Type here to search
ENG
IN
12:11
10-07-2025

B1	Booked by Prakhar01	2025-07-10	Cancel & Bill
B2	Available	N/A	Book for... Book Reserve
B3	Available	N/A	Book for... Book Reserve
B4	Booked by ashish01	N/A	Cancel & Bill
B5	Available	N/A	Book for... Book Reserve
C1	Available	N/A	Book for... Book Reserve
C2	Available	N/A	Book for... Book Reserve
C3	Requested by testuser	N/A	Approve Deny
C4	Available	N/A	Book for... Book Reserve
C5	Available	N/A	Book for... Book Reserve
D1	Available	N/A	Book for... Book Reserve
D2	Available	N/A	Book for... Book Reserve
D3	Available	N/A	Book for... Book Reserve
D4	Available	N/A	Book for... Book Reserve
D5	Available	N/A	Book for... Book Reserve
E1	Available	N/A	Book for... Book Reserve
E2	Available	N/A	Book for... Book Reserve

Step-by-step, from-scratch guide to get you comfortable with PHP basics and build a simple dynamic PHP page. Each section includes code snippets you can copy-paste and try in your own environment.

---

1. Setup & Environment

1. **Install a Local Server Stack**
   - **XAMPP** (Windows/macOS/Linux): https://www.apachefriends.org
   - **MAMP** (macOS): https://www.mamp.info
   - **WAMP** (Windows): http://www.wampserver.com
2. **Verify PHP Is Running**
   - Place a file named `info.php` in your server's web root (e.g., `htdocs` or `www`):

     ```
     <?php
     phpinfo();
     ```

   - Visit `http://localhost/info.php` in your browser. You should see PHP configuration details.
3. **Built-in PHP Server** (for simple testing)

   ```
   cd path/to/your/project
   php -S localhost:8000
   ```

   Then open `http://localhost:8000` in your browser.

---

2. Your First PHP Page

Create `index.php` with:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>My First PHP Page</title>
</head>
<body>
  <?php
    echo "<h1>Hello, PHP World!</h1>";
  ?>
</body>
</html>
```

- **PHP Tags**
    - Standard: `<?php ... ?>`
    - Short (only if enabled): `<?= 'Hello'; ?>`
- **Comments**

  ```
  // single-line
  # single-line
  /*
     multi-line
  */
  ```

---

3. Variables & Data Types

```
<?php
// Variable naming: starts with $
$name   = "Himanshu";     // string
$age    = 30;             // integer
$price  = 19.99;          // float
$is_admin = true;         // boolean
$items  = ["apple", "banana", "cherry"];  // indexed array
$user   = ["id" => 1, "name" => "Alice"]; // associative array
$nothing = null;          // NULL
```

- **Constants**

  ```
  define('SITE_NAME', 'MySite');
  echo SITE_NAME;
  ```

---

4. Operators

Type	Example
Arithmetic	`+ - * / %`
Assignment	`=, +=, -=, .=`
Comparison	`==, ===, !=, <, >`
Logical	`&&,
String concat	`.`

```
$a = 5;
$b = 2;
echo $a + $b;          // 7
echo "Hello, " . $name; // Hello, Himanshu
```

---

5. Control Structures

If / Else

```
if ($age >= 18) {
  echo "Adult";
} elseif ($age >= 13) {
  echo "Teen";
} else {
  echo "Child";
}
```

Switch

```
switch ($role) {
  case 'admin':
    echo "Welcome, admin";
    break;
  default:
    echo "Welcome, guest";
}
```

Loops

```
// for
for ($i = 0; $i < count($items); $i++) {
  echo $items[$i];
}

// foreach
foreach ($user as $key => $value) {
  echo "$key: $value";
}

// while
$i = 0;
while ($i < 3) {
  echo $i++;
}

// do-while
do {
  echo "At least once";
} while (false);
```

---

6. Functions

```
<?php
function greet($person) {
  return "Hello, $person!";
}

echo greet("World"); // Hello, World!
```

- **Parameter defaults**

```
function add($x, $y = 10) { return $x + $y; }
```

---

7. Working with Forms

HTML Form (`form.html`)

```
<form action="process.php" method="post">
  Name: <input name="name">
  <button type="submit">Send</button>
</form>
```

PHP Processing (`process.php`)

```
<?php
$name = $_POST['name'] ?? 'Guest';
echo "Hello, " . htmlspecialchars($name);
```

- **Security**: Always sanitize user input (e.g., `htmlspecialchars`, prepared statements).

---

8. Including Files

Break your code into parts:

- **header.php**

  ```
  <header><h1>My Site</h1></header>
  ```

- **footer.php**

  ```
  <footer>&copy; 2025</footer>
  ```

- **index.php**

  ```
  <?php include 'header.php'; ?>
  <p>Welcome to the homepage.</p>
  <?php include 'footer.php'; ?>
  ```

- **`require`** is like `include` but throws a fatal error on failure.

---

9. Sessions & Cookies

Cookies

```
setcookie('theme', 'dark', time() + 3600); // expires in 1 hour
echo $_COOKIE['theme'] ?? 'light';
```

Sessions

```
session_start();
$_SESSION['user_id'] = 123;
echo $_SESSION['user_id'];
```

- **Logout**:

  ```
  session_start();
  session_unset();
  session_destroy();
  ```

---

10. Basic Database Connectivity

Using MySQLi

```
<?php
$mysqli = new mysqli('localhost', 'root', '', 'testdb');
if ($mysqli->connect_error) {
  die('Connection error: ' . $mysqli->connect_error);
}

$result = $mysqli->query("SELECT * FROM users");
while ($row = $result->fetch_assoc()) {
  echo $row['name'] . "<br>";
}
$mysqli->close();
```

Using PDO

```
<?php
try {
  $pdo = new PDO('mysql:host=localhost;dbname=testdb', 'root', '');
  $stmt = $pdo->query("SELECT * FROM users");
  foreach ($stmt as $row) {
    echo $row['name'] . "<br>";
  }
} catch (PDOException $e) {
  echo 'Error: ' . $e->getMessage();
}
```

---

11. Next Steps

- **Error Handling**: learn `try/catch`, `error_reporting()`
- **OOP in PHP**: classes, objects, namespaces

- **Frameworks**: Laravel, Symfony for larger apps
- **APIs**: building and consuming RESTful services
- **Security**: input validation, password hashing (`password_hash`), CSRF protection

---

Try it yourself:

1. Create a simple contact form, store submissions in a database, and list them.
2. Build a tiny login/logout system using sessions.

This foundation will empower you to build full-fledged PHP applications. Happy coding!

Project Explaination

Here's the overall structure of your "librarysda" project after unzipping **librarysda.zip**:

```
librarysda/
├── admin_dashboard.php
├── billing.php
├── db_connect.php
├── generate_password_hash.php
├── index.php
├── library_system.sql
├── logout.php
├── New Text Document.txt
├── profile.php
├── register.php
├── user_dashboard.php
├── view_slip.php
└── uploads/
    └── avatars/
        └── default.png
```

Let's start with the foundational piece—the database connection file **db_connect.php**.

---

1. db_connect.php

```
<?php
// db_connect.php
// This file contains the database connection configuration.

// Database credentials
define('DB_SERVER',   'localhost');
define('DB_USERNAME', 'root');           // default XAMPP user
define('DB_PASSWORD', '');               // default XAMPP password
define('DB_NAME',     'you_DB_NAME'); // name of your database

// Attempt to connect to MySQL
$conn = new mysqli(DB_SERVER, DB_USERNAME, DB_PASSWORD, DB_NAME);

// Check the connection
if ($conn === false) {
    // Connection failed: halt execution and show error
    die("ERROR: Could not connect. " . $conn->connect_error);
}

// Note: PHP automatically closes the connection at script end,
// but you can call $conn->close(); in long-running scripts.
?>
```

Syntax & Logic Breakdown

1. **Constants for credentials**

   ```
   define('DB_SERVER',   'localhost');
   define('DB_USERNAME', 'root');
   define('DB_PASSWORD', '');
   define('DB_NAME',     'library_system');
   ```

   - `define()` creates named constants.
   - Keeps credentials in one place so other scripts simply `include 'db_connect.php';` to get `$conn`.

2. **Creating the connection**

   ```
   $conn = new mysqli(DB_SERVER, DB_USERNAME, DB_PASSWORD, DB_NAME);
   ```

   - Instantiates PHP's MySQLi object.
   - Parameters: (host, user, pass, database).

3. **Error checking**

   ```
   if ($conn === false) {
     die("ERROR: Could not connect. " . $conn->connect_error);
   }
   ```

   - If connection fails, `$conn` is `false`.
   - `die()` stops the script and outputs the error message.

4. **Closing the connection**
   - While not shown explicitly, you can later call:

     ```
     $conn->close();
     ```

   - Good practice in scripts that run many queries or persist long.

Here’s a detailed walkthrough of **index.php**—the entry point that users land on when they first visit your site.

---

1. Session Initialization & Redirect Logic

```php
<?php
// Start the session ONCE at the very top of the script.
session_start();

// If the user is already logged in, send them straight to their dashboard
if (isset($_SESSION["loggedin"]) && $_SESSION["loggedin"] === true) {
    if ($_SESSION["role"] === 'admin') {
        header("location: admin_dashboard.php");
    } else {
        header("location: user_dashboard.php");
    }
    exit;
}
```

- **`session_start()`**
  Must be called before any HTML output. It either resumes an existing session or starts a new one.
- **Redirect if already logged in**
  - Checks `$_SESSION["loggedin"]`; if `true`, we inspect `$_SESSION["role"]`.
  - Admins → `admin_dashboard.php`; regular users → `user_dashboard.php`.
  - `exit;` ensures no further code runs after the redirect.

---

2. Database Connection & Variable Setup

```php
// Bring in the $conn object from db_connect.php
require_once "db_connect.php";

// Initialize variables for form input and errors
$username = $password = "";
$username_err = $password_err = $login_err = "";
```

- **`require_once`**
  Loads your `db_connect.php` exactly once; if it fails, script halts with a fatal error.
- **Input & error variables**
  Start empty; we’ll populate these as the form is submitted and validated.

---

3. Handling the Login Form Submission

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // 1. Validate username
    if (empty(trim($_POST["username"]))) {
        $username_err = "Please enter username.";
    } else {
        $username = trim($_POST["username"]);
    }

    // 2. Validate password
    if (empty(trim($_POST["password"]))) {
        $password_err = "Please enter your password.";
    } else {
        $password = trim($_POST["password"]);
    }

    // 3. Attempt login if no errors
    if (empty($username_err) && empty($password_err)) {
        // Prepare a SELECT statement to fetch user by username
        $sql = "SELECT id, username, password, role FROM users WHERE username
= ?";
        if ($stmt = $conn->prepare($sql)) {
            $stmt->bind_param("s", $param_username);
            $param_username = $username;
            $stmt->execute();
            $stmt->store_result();

            // Check if the user exists
            if ($stmt->num_rows == 1) {
                $stmt->bind_result($id, $username, $hashed_password, $role);
                $stmt->fetch();
                // Verify the password against the hashed version
                if (password_verify($password, $hashed_password)) {
                    // Password is correct—start a new session
                    session_start();
                    $_SESSION["loggedin"] = true;
                    $_SESSION["id"]       = $id;
                    $_SESSION["username"] = $username;
                    $_SESSION["role"]     = $role;

                    // Redirect based on role
                    if ($role === 'admin') {
                        header("location: admin_dashboard.php");
                    } else {
                        header("location: user_dashboard.php");
                    }
                } else {
                    $login_err = "Invalid username or password.";
                }
            } else {
                $login_err = "Invalid username or password.";
            }
            $stmt->close();
        } else {
            echo "Oops! Something went wrong. Please try again later.";
        }
    }
    // Close DB connection
    $conn->close();
}
?>
```

Key Points

1. **Form POST check**:
   `$_SERVER["REQUEST_METHOD"] == "POST"` ensures we only process when the form is submitted.
2. **Input trimming & validation**:
   - `trim()` removes extra whitespace.
   - We set specific error messages (`$username_err`, `$password_err`) if fields are empty.
3. **Prepared statements**:
   - Avoid SQL injection by using `$conn->prepare()`, then `bind_param()` and `execute()`.
   - `store_result()` lets us call `$stmt->num_rows`.
4. **Password hashing & verification**:
   - We assume during registration you used `password_hash()`.
   - `password_verify($plain, $hashed)` safely checks credentials.
5. **Session variables**:
   - On success, we set `$_SESSION["loggedin"]`, `$_SESSION["id"]`, `$_SESSION["username"]`, and `$_SESSION["role"]`, then redirect.

---

4. HTML Login Form

After the PHP logic closes (`?>`), the file serves up a responsive HTML form (styled with Tailwind CSS):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <!-- meta tags & Tailwind CDN -->
</head>
<body class="bg-gray-100 flex items-center justify-center h-screen">
  <div class="w-full max-w-sm bg-white rounded-lg shadow-md p-6">
    <h2 class="text-2xl font-semibold text-center mb-4">Login to SV
Library</h2>

    <!-- Display login error, if any -->
    <?php
    if(!empty($login_err)){
        echo '<div class="bg-red-100 text-red-700 p-2 mb-4 rounded">'.
$login_err .'</div>';
    }
    ?>
    <form action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>"
method="post">
      <!-- Username field -->
      <div class="mb-4">
        <label class="block mb-1">Username</label>
        <input type="text" name="username"
               class="w-full border rounded px-3 py-2 <?php echo
(!empty($username_err)) ? 'border-red-500' : 'border-gray-300'; ?>"
               value="<?php echo $username; ?>">
        <span class="text-red-500 text-sm"><?php echo $username_err;
?></span>
      </div>

      <!-- Password field -->
      <div class="mb-4">
        <label class="block mb-1">Password</label>
        <input type="password" name="password"
               class="w-full border rounded px-3 py-2 <?php echo
(!empty($password_err)) ? 'border-red-500' : 'border-gray-300'; ?>">
        <span class="text-red-500 text-sm"><?php echo $password_err;
?></span>
      </div>

      <!-- Submit button -->
      <div class="flex items-center justify-between">
        <button type="submit"
                class="bg-indigo-600 text-white px-4 py-2 rounded hover:bg-
indigo-700">
          Login
        </button>
        <a href="register.php" class="text-sm text-indigo-600
hover:underline">
          Sign up
        </a>
      </div>
    </form>
  </div>
</body>
</html>
```

- **Error feedback**
  Inline `<span>`s display field-specific errors; a banner above the form shows a general login error.
- **Styling**
  Uses Tailwind's utility classes for layout, spacing, borders, and hover states.
- **Self-referencing form**
  `action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]); ?>"` keeps data safe from XSS.

1. Include Database Connection & Initialize Variables

```
<?php
// register.php

// 1. Bring in the shared $conn object
require_once "db_connect.php";

// 2. Prepare variables for form inputs and error messages
$username = $password = $confirm_password = "";
$username_err = $password_err = $confirm_password_err = "";
```

- **`require_once "db_connect.php";`** pulls in your MySQLi connection (`$conn`) so you can run queries.
- You initialize each input and its corresponding error variable to an empty string.

---

2. Handle the Form Submission

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // … validation and insertion logic goes here …
}
```

You only run the registration logic when the form submits via POST.

---

3. Validate the Username

```
// Trim whitespace and check for emptiness
if (empty(trim($_POST["username"]))) {
    $username_err = "Please enter a username.";
} else {
    // Check if username is already taken
    $sql = "SELECT id FROM users WHERE username = ?";
    if ($stmt = $conn->prepare($sql)) {
        $stmt->bind_param("s", $param_username);
        $param_username = trim($_POST["username"]);
        $stmt->execute();
        $stmt->store_result();

        if ($stmt->num_rows == 1) {
            $username_err = "This username is already taken.";
        } else {
            $username = trim($_POST["username"]);
        }
        $stmt->close();
    }
}
```

1. **Empty check**: sets `$username_err` if the field is blank.
2. **Uniqueness check**:
   - Prepares a `SELECT` to see if that username exists.
   - If `num_rows == 1`, it's already in use; otherwise, we accept it.

---

4. Validate the Password

```
if (empty(trim($_POST["password"]))) {
    $password_err = "Please enter a password.";
} elseif (strlen(trim($_POST["password"])) < 6) {
    $password_err = "Password must have at least 6 characters.";
} else {
    $password = trim($_POST["password"]);
}
```

- Ensures the password isn't empty and is at least 6 characters long, setting `$password_err` otherwise.

---

5. Validate Confirm Password

```
if (empty(trim($_POST["confirm_password"]))) {
    $confirm_password_err = "Please confirm password.";
} else {
    $confirm_password = trim($_POST["confirm_password"]);
    if (empty($password_err) && ($password !== $confirm_password)) {
        $confirm_password_err = "Passwords do not match.";
    }
}
```

- Checks for a non-empty confirm field, then ensures it matches the original password (only if no password error).

---

6. Insert the New User (When All Validation Passes)

```
if (empty($username_err) && empty($password_err) &&
empty($confirm_password_err)) {
    $sql = "INSERT INTO users (username, password, role) VALUES (?, ?, ?)";
    if ($stmt = $conn->prepare($sql)) {
        // Hash the password before storing
        $param_username = $username;
        $param_password = password_hash($password, PASSWORD_DEFAULT);
        $param_role     = 'user';
        $stmt->bind_param("sss", $param_username, $param_password,
$param_role);

        if ($stmt->execute()) {
            // Registration succeeded—redirect to login page
            header("location: index.php");
            exit;
        } else {
            echo "Oops! Something went wrong. Please try again later.";
        }
        $stmt->close();
    }
}
$conn->close();
```

1. **Collect parameters**:
   - `$param_username` ← the validated username
   - `$param_password` ← the result of `password_hash()` (securely salts & hashes)
   - `$param_role` ← defaults to `'user'`
2. **Execute the INSERT**: on success, redirect back to your login page (`index.php`).

---

7. The Registration Form (HTML + Tailwind)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <!-- Tailwind CDN, meta tags, etc. -->
</head>
<body class="bg-gray-800 flex items-center justify-center h-screen">
  <div class="bg-gray-700 p-8 rounded-lg shadow-lg w-full max-w-md">
    <h2 class="text-2xl font-bold text-white mb-6 text-center">Sign Up</h2>

    <form action="register.php" method="post" class="space-y-4">
      <!-- Username Field -->
      <div>
        <label class="block text-gray-300">Username</label>
        <input type="text" name="username"
               class="w-full mt-1 p-2 rounded <?php echo $username_err ?
'border-red-500' : 'border-gray-600'; ?>"
               value="<?php echo htmlspecialchars($username); ?>">
        <p class="text-red-400 text-sm mt-1"><?php echo $username_err; ?></p>
      </div>

      <!-- Password Field -->
      <div>
        <label class="block text-gray-300">Password</label>
        <input type="password" name="password"
               class="w-full mt-1 p-2 rounded <?php echo $password_err ?
'border-red-500' : 'border-gray-600'; ?>">
        <p class="text-red-400 text-sm mt-1"><?php echo $password_err; ?></p>
      </div>
      <!-- Confirm Password Field -->
      <div>
        <label class="block text-gray-300">Confirm Password</label>
        <input type="password" name="confirm_password"
               class="w-full mt-1 p-2 rounded <?php echo
$confirm_password_err ? 'border-red-500' : 'border-gray-600'; ?>"
               value="<?php echo htmlspecialchars($confirm_password); ?>">
        <p class="text-red-400 text-sm mt-1"><?php echo
$confirm_password_err; ?></p>
      </div>

      <!-- Submit Button -->
      <div>
        <button type="submit"
                class="w-full bg-indigo-600 hover:bg-indigo-500 text-white p-
2 rounded transition">
          Sign Up
        </button>
      </div>

      <p class="text-center text-gray-400 text-sm">
        Already have an account?
        <a href="index.php" class="text-indigo-400 hover:underline">Log
in</a>
      </p>
    </form>
  </div>
</body>
</html>
```

- Each input shows its validation error immediately beneath it.
- Uses Tailwind’s utility classes for styling and responsive spacing.
- The form posts back to itself (`action="register.php"`), letting the PHP at the top handle the submission.

That completes the walkthrough of **register.php**. You now see:

1. **How inputs are sanitized and validated.**
2. **How duplicate usernames are prevented.**
3. **How passwords are securely hashed with `password_hash()`.**
4. **How a new user row is inserted and you’re redirected back to login.**

Let’s dive into **admin_dashboard.php**, the control center for your librarians/administrators. We’ll break it down into logical sections:

---

1. Session & Security Gate

```
session_start();
require_once "db_connect.php";

// Only allow logged-in admins
if (
    !isset($_SESSION["loggedin"])
    || $_SESSION["loggedin"] !== true
    || $_SESSION["role"] !== 'admin'
) {
    header("location: index.php");
    exit;
}
```

- **`session_start()`** resumes the user’s session so you can read `$_SESSION` variables.
- Immediately after, we check:
  - Is the user logged in?
  - Is their role **exactly** `'admin'`?
- If either fails, we redirect them back to the login page (`index.php`) and `exit` to stop any further output.

---

2. Fetching the “Active” Booking Session

```
$active_session_query = $conn->query(
    "SELECT id, session_name
     FROM sessions
     WHERE is_active = 1
     LIMIT 1"
);
$active_session = $active_session_query->fetch_assoc();
$active_session_id = $active_session['id'] ?? null;
```

- We pull the single row from `sessions` where `is_active = 1`.
- Storing its `id` for use in subsequent queries (e.g., seat bookings, payments).

---

3. Auto-Generate Payments for Completed Bookings

```
// For any “requested” seat bookings that now have both joining & leaving
dates,
// calculate fee and insert into payments.
$pending = $conn->query(
    "SELECT * FROM bookings
     WHERE payment_generated = 0
       AND date_of_leaving IS NOT NULL"
);
while ($booking = $pending->fetch_assoc()) {
    $d1 = new DateTime($booking['date_of_joining']);
    $d2 = new DateTime($booking['date_of_leaving']);
    $duration = $d1->diff($d2)->days + 1;

    // Base fee + extra days
    $amount = 500.00;
    if ($duration > 10) {
        $amount += ($duration - 10) * 20.00;
    }

    // Insert into payments
    $ins = $conn->prepare(
      "INSERT INTO payments
       (user_id, seat_id, session_id, date_of_joining, date_of_leaving,
duration_days, amount_due)
       VALUES (?, ?, ?, ?, ?, ?, ?)"
    );
    $ins->bind_param(
      "iiisssd",
      $booking['user_id'],
      $booking['seat_id'],
      $booking['session_id'],
      $booking['date_of_joining'],
      $booking['date_of_leaving'],
      $duration,
      $amount
    );
    $ins->execute();
    $ins->close();

    // Mark booking as billed
    $conn->query(
      "UPDATE bookings
       SET payment_generated = 1
       WHERE id = {$booking['id']}"
    );
}
```

- **Loop** through all bookings where `payment_generated = 0` and a leaving date exists.
- **Calculate duration** via PHP's `DateTime` diff.
- **Compute amount**: ₹500 base + ₹20 per extra day beyond 10.
- **Insert** a new row in `payments` and then flag the booking as billed.

---

4. Handling Admin Actions (POST)

All POST operations end by redirecting back to the dashboard with the current tab preserved (`page` query param).

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $current_page = $_POST['current_page'] ?? 'overview';

    // A. Approve a seat request
    if (isset($_POST['approve_seat'])) {
        $stmt = $conn->prepare(
          "UPDATE seats
           SET status = 'booked',
               user_id = ?,
               requested_by_id = NULL,
               date_of_joining = ?
           WHERE id = ?"
        );
        $stmt->bind_param(
          "isi",
          $_POST['user_id'],
          date("Y-m-d"),
          $_POST['seat_id']
        );
        $stmt->execute();
        $stmt->close();
    }

    // B. Clear all bills
    if (isset($_POST['clear_all_bills'])) {
        $conn->query("TRUNCATE TABLE payments");
    }

    // C. Create a new booking session
    if (isset($_POST['create_session'])) {
        $name = trim($_POST['session_name']);
        if ($name !== "") {
            $stmt = $conn->prepare(
              "INSERT INTO sessions (session_name)
               VALUES (?)"
            );
            $stmt->bind_param("s", $name);
            $stmt->execute();
        }
    }

    // D. Activate a session
    if (isset($_POST['activate_session'])) {
        $conn->query("UPDATE sessions SET is_active = 0");
        $stmt = $conn->prepare(
          "UPDATE sessions SET is_active = 1 WHERE id = ?"
        );
        $stmt->bind_param("i", $_POST['session_id']);
        $stmt->execute();
    }

    // E. Generate seats grid
    if (isset($_POST['generate_seats'])) {
        if ($active_session_id) {
            $rows = intval($_POST['rows']);
            $cols = intval($_POST['cols']);
            // Remove old seats for this session
            $stmt = $conn->prepare(
              "DELETE FROM seats WHERE session_id = ?"
            );
            $stmt->bind_param("i", $active_session_id);
            $stmt->execute();

            // Insert new seats (e.g. A1, A2, …)
            $stmt = $conn->prepare(
              "INSERT INTO seats (seat_number, session_id)
               VALUES (?, ?)"
            );
            for ($r = 0; $r < $rows; $r++) {
                for ($c = 1; $c <= $cols; $c++) {
                    $num = chr(65 + $r) . $c;
                    $stmt->bind_param("si", $num, $active_session_id);
                    $stmt->execute();
                }
            }
        }
    }

    // F. (Optional) Manual seat booking by admin
    if (!empty($_POST['user_id_to_book'])) {
        $stmt = $conn->prepare(
          "UPDATE seats
           SET status = 'booked',
               user_id = ?,
               date_of_joining = ?
           WHERE id = ?"
        );
        $today = date("Y-m-d");
        $stmt->bind_param(
          "isi",
          $_POST['user_id_to_book'],
          $today,
          $_POST['seat_id']
        );
        $stmt->execute();
    }

    header("location: admin_dashboard.php?page=" . $current_page);
    exit;
}
```

5. Preparing Data for Each Tab (GET)

```
$page = $_GET['page'] ?? 'overview';

if ($active_session_id) {
    // 1. Seat statistics
    $stats['total']     = $conn->query("SELECT COUNT(*) FROM seats WHERE
session_id = $active_session_id")->fetch_row()[0];
    $stats['booked']    = $conn->query("SELECT COUNT(*) FROM seats WHERE
session_id = $active_session_id AND status = 'booked'")->fetch_row()[0];
    $stats['available'] = $conn->query("SELECT COUNT(*) FROM seats WHERE
session_id = $active_session_id AND status = 'available'")->fetch_row()[0];
    $stats['requested'] = $conn->query("SELECT COUNT(*) FROM seats WHERE
session_id = $active_session_id AND status = 'requested'")->fetch_row()[0];

    // 2. All seats with requester info
    $stmt = $conn->prepare(
      "SELECT s.id, s.seat_number, s.status,
              u_requested.username AS requested_by
       FROM seats s
       LEFT JOIN users u_requested ON s.requested_by_id = u_requested.id
       WHERE s.session_id = ?
       ORDER BY s.seat_number"
    );
    $stmt->bind_param("i", $active_session_id);
    $stmt->execute();
    $all_seats_result = $stmt->get_result();
}

// 3. List of users (for booking dropdown)
$users_list = $conn->query(
    "SELECT id, username FROM users WHERE role = 'user' ORDER BY username"
)->fetch_all(MYSQLI_ASSOC);

// 4. User management (view/edit all users)
$user_management_result = $conn->query(
    "SELECT id, username, full_name, email, role
     FROM users ORDER BY username"
);

// 5. All booking sessions
$all_sessions_result = $conn->query(
    "SELECT id, session_name, is_active
     FROM sessions ORDER BY session_name DESC"
);

// 6. Payment slips history
$payment_slips_result = $conn->query(
    "SELECT p.*, u.username
```

```
     FROM payments p
     JOIN users u ON p.user_id = u.id
     ORDER BY p.generated_at DESC"
);
```

- We load whatever data each tab needs: seat stats, seat lists, user list, sessions list, and past payment records.

---

6. The HTML Dashboard

Below the PHP block is a single HTML template that:

- Loads Tailwind CSS and sets up a sidebar or nav bar with links to each `?page=` (Overview, Seats, Users, Sessions, Billing, Slips).
- Displays the relevant data table or form based on the `$page` variable.
- Embeds forms/buttons for all the POST actions you saw above (Approve seat, Clear bills, Create session, etc.).

All of this lives in one file, toggled by your `$page` variable.

Why It Matters

- **Centralized Admin Logic**: Everything from seat management to billing lives here.
- **Role-Based Security**: Only admins can reach this page.
- **Single-Page “Tabs”**: Keeps your admin tools consolidated under different GET parameters rather than separate files.
- **Prepared Statements Everywhere**: Protects against SQL injection.
- **Server-Side Fee Calculation**: Written in PHP, not left to the client.

Below is the core HTML template used in **admin_dashboard.php**, immediately following the PHP logic at the top. It uses Tailwind CSS and switches content based on the `?page=` parameter.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Admin Dashboard – SV Library</title>
  <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="bg-gray-100 min-h-screen flex">

  <!-- Sidebar -->
  <aside class="w-64 bg-white shadow-md">
    <div class="p-6">
      <h1 class="text-2xl font-bold mb-6">SV Library Admin</h1>
      <nav class="space-y-2">
        <a href="admin_dashboard.php?page=overview"
           class="block px-4 py-2 rounded hover:bg-indigo-100 <?=
$page=='overview'?'bg-indigo-50 font-semibold':'' ?>">
          Overview
        </a>
        <a href="admin_dashboard.php?page=seats"
           class="block px-4 py-2 rounded hover:bg-indigo-100 <?=
$page=='seats'?'bg-indigo-50 font-semibold':'' ?>">
          Seats
        </a>
        <a href="admin_dashboard.php?page=users"
           class="block px-4 py-2 rounded hover:bg-indigo-100 <?=
$page=='users'?'bg-indigo-50 font-semibold':'' ?>">
          Users
        </a>
        <a href="admin_dashboard.php?page=sessions"
           class="block px-4 py-2 rounded hover:bg-indigo-100 <?=
$page=='sessions'?'bg-indigo-50 font-semibold':'' ?>">
          Sessions
        </a>
        <a href="admin_dashboard.php?page=billing"
           class="block px-4 py-2 rounded hover:bg-indigo-100 <?=
$page=='billing'?'bg-indigo-50 font-semibold':'' ?>">
          Billing
        </a>
        <a href="admin_dashboard.php?page=slips"
           class="block px-4 py-2 rounded hover:bg-indigo-100 <?=
$page=='slips'?'bg-indigo-50 font-semibold':'' ?>">
          Payment Slips
        </a>
        <a href="logout.php"
           class="block mt-6 px-4 py-2 text-red-600 rounded hover:bg-red-
100">
          Logout
        </a>
      </nav>
    </div>
  </aside>
  <!-- Main Content -->
  <main class="flex-1 p-8 overflow-y-auto">
    <!-- Page Title -->
    <h2 class="text-3xl font-semibold mb-6 capitalize"><?=
htmlspecialchars($page) ?></h2>

    <!-- Overview Tab -->
    <?php if ($page === 'overview'): ?>
      <div class="grid grid-cols-4 gap-6">
        <div class="bg-white p-4 rounded shadow">
          <h3 class="text-lg font-medium">Total Seats</h3>
          <p class="text-2xl"><?= $stats['total'] ?></p>
        </div>
        <div class="bg-white p-4 rounded shadow">
          <h3 class="text-lg font-medium">Booked</h3>
          <p class="text-2xl"><?= $stats['booked'] ?></p>
        </div>
        <div class="bg-white p-4 rounded shadow">
          <h3 class="text-lg font-medium">Available</h3>
          <p class="text-2xl"><?= $stats['available'] ?></p>
        </div>
        <div class="bg-white p-4 rounded shadow">
          <h3 class="text-lg font-medium">Requested</h3>
          <p class="text-2xl"><?= $stats['requested'] ?></p>
        </div>
      </div>
    <?php endif; ?>

    <!-- Seats Tab -->
    <?php if ($page === 'seats'): ?>
      <form method="post" class="mb-6">
        <input type="hidden" name="current_page" value="seats">
        <button name="generate_seats" class="bg-indigo-600 text-white px-4
py-2 rounded hover:bg-indigo-700">
          Generate Seats
        </button>
      </form>
      <div class="overflow-x-auto">
        <table class="min-w-full bg-white rounded shadow">
          <thead>
            <tr class="bg-gray-50">
              <th class="px-4 py-2">Seat</th>
              <th class="px-4 py-2">Status</th>
              <th class="px-4 py-2">Requested By</th>
              <th class="px-4 py-2">Actions</th>
            </tr>
          </thead>
          <tbody>
            <?php while($seat = $all_seats_result->fetch_assoc()): ?>
              <tr class="border-t">
                <td class="px-4 py-2"><?=
htmlspecialchars($seat['seat_number']) ?></td>
                <td class="px-4 py-2 capitalize"><?= $seat['status'] ?></td>
                <td class="px-4 py-2"><?=
htmlspecialchars($seat['requested_by'] ?? '-') ?></td>
                <td class="px-4 py-2">
```

```
                  <?php if ($seat['status']==='requested'): ?>
                    <form method="post" class="inline">
                      <input type="hidden" name="current_page" value="seats">
                      <input type="hidden" name="seat_id" value="<?=
$seat['id'] ?>">
                      <input type="hidden" name="user_id" value="<?=
$seat['requested_by_id'] ?>">
                      <button name="approve_seat"
                              class="bg-green-500 text-white px-2 py-1
rounded hover:bg-green-600">
                        Approve
                      </button>
                    </form>
                  <?php endif; ?>
                </td>
              </tr>
            <?php endwhile; ?>
          </tbody>
        </table>
      </div>
    <?php endif; ?>

    <!-- Users Tab -->
    <?php if ($page === 'users'): ?>
      <div class="overflow-x-auto">
        <table class="min-w-full bg-white rounded shadow">
          <thead>
            <tr class="bg-gray-50">
              <th class="px-4 py-2">Username</th>
              <th class="px-4 py-2">Full Name</th>
              <th class="px-4 py-2">Email</th>
              <th class="px-4 py-2">Role</th>
            </tr>
          </thead>
          <tbody>
            <?php while($u = $user_management_result->fetch_assoc()): ?>
              <tr class="border-t">
                <td class="px-4 py-2"><?= htmlspecialchars($u['username'])
?></td>
                <td class="px-4 py-2"><?= htmlspecialchars($u['full_name'])
?></td>
                <td class="px-4 py-2"><?= htmlspecialchars($u['email'])
?></td>
                <td class="px-4 py-2 capitalize"><?= $u['role'] ?></td>
              </tr>
            <?php endwhile; ?>
          </tbody>
        </table>
      </div>
    <?php endif; ?>

    <!-- Sessions Tab -->
    <?php if ($page === 'sessions'): ?>
      <form method="post" class="mb-6 space-y-4">
        <input type="hidden" name="current_page" value="sessions">
        <div>
```

```
          <label class="block">New Session Name:</label>
          <input type="text" name="session_name" class="mt-1 p-2 border
rounded w-full">
        </div>
        <button name="create_session"
                class="bg-indigo-600 text-white px-4 py-2 rounded hover:bg-
indigo-700">
          Create Session
        </button>
      </form>
      <div class="overflow-x-auto">
        <table class="min-w-full bg-white rounded shadow">
          <thead>
            <tr class="bg-gray-50">
              <th class="px-4 py-2">Name</th>
              <th class="px-4 py-2">Active</th>
              <th class="px-4 py-2">Actions</th>
            </tr>
          </thead>
          <tbody>
            <?php while($s = $all_sessions_result->fetch_assoc()): ?>
              <tr class="border-t">
                <td class="px-4 py-2"><?=
htmlspecialchars($s['session_name']) ?></td>
                <td class="px-4 py-2"><?= $s['is_active'] ? 'Yes' : 'No'
?></td>
                <td class="px-4 py-2">
                  <?php if (!$s['is_active']): ?>
                    <form method="post" class="inline">
                      <input type="hidden" name="current_page"
value="sessions">
                      <input type="hidden" name="session_id" value="<?=
$s['id'] ?>">
                      <button name="activate_session"
                              class="bg-green-500 text-white px-2 py-1
rounded hover:bg-green-600">
                        Activate
                      </button>
                    </form>
                  <?php endif; ?>
                </td>
              </tr>
            <?php endwhile; ?>
          </tbody>
        </table>
      </div>
    <?php endif; ?>

    <!-- Billing Tab -->
    <?php if ($page === 'billing'): ?>
      <form method="post" class="mb-6">
        <input type="hidden" name="current_page" value="billing">
        <button name="clear_all_bills"
                class="bg-red-600 text-white px-4 py-2 rounded hover:bg-red-
700">
          Clear All Bills
        </button>
      </form>
      <!-- You could also display a summary or chart of billing here -->
    <?php endif; ?>

    <!-- Payment Slips Tab -->
    <?php if ($page === 'slips'): ?>
      <div class="overflow-x-auto">
        <table class="min-w-full bg-white rounded shadow">
          <thead>
            <tr class="bg-gray-50">
              <th class="px-4 py-2">Slip ID</th>
              <th class="px-4 py-2">User</th>
              <th class="px-4 py-2">Amount</th>
              <th class="px-4 py-2">Duration</th>
              <th class="px-4 py-2">Dates</th>
              <th class="px-4 py-2">Generated At</th>
            </tr>
          </thead>
          <tbody>
            <?php while($p = $payment_slips_result->fetch_assoc()): ?>
              <tr class="border-t">
                <td class="px-4 py-2"><?= $p['id'] ?></td>
                <td class="px-4 py-2"><?= htmlspecialchars($p['username'])
?></td>
                <td class="px-4 py-2">₹<?= number_format($p['amount_due'],2)
?></td>
                <td class="px-4 py-2"><?= $p['duration_days'] ?> days</td>
                <td class="px-4 py-2">
                  <?= htmlspecialchars($p['date_of_joining']) ?>
                  – <?= htmlspecialchars($p['date_of_leaving']) ?>
                </td>
                <td class="px-4 py-2"><?=
htmlspecialchars($p['generated_at']) ?></td>
              </tr>
            <?php endwhile; ?>
          </tbody>
        </table>
      </div>
    <?php endif; ?>

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

How It Works

1. **Sidebar Navigation**
   - A fixed `<aside>` with Tailwind styling.
   - Highlights the active tab by checking `<?= $page ?>`.
2. **Main Content Area**
   - A single `<main>` section that conditionally includes one of six “tabs” (Overview, Seats, Users, Sessions, Billing, Payment Slips).
   - Uses PHP’s `if ($page === '…')` to show the appropriate block.

3. **Reusable Components**
    - Tables with `.min-w-full`, `.bg-white`, `.rounded`, and `.shadow` for a consistent look.
    - Forms include hidden `current_page` fields so that after a POST, the same tab will redisplay.
4. **Responsive & Accessible**
    - Utility classes for spacing (`p-4`, `mb-6`), typography (`text-xl`, `font-semibold`), and hover states (`hover:bg-indigo-100`).
    - Data cells use `htmlspecialchars()` to prevent XSS.

This HTML structure gives a clear, organized admin interface that ties directly into the PHP logic you've already seen. Let me know if you'd like to tweak any layout, add charts, or refine any component!

The **user_dashboard.php** shares the same overall "shell" as **admin_dashboard.php**—you've still got:

1. **`session_start()`** + **`require_once "db_connect.php"`**`;` to bootstrap your session and database connection.
2. A **security check** that kicks you back to `index.php` if you're not logged in.
3. A **sidebar** laid out in Tailwind, with the same base styling, but only three links:
    - **Book a Seat** (this page)
    - **My Profile** (`profile.php`)
    - **Log Out** (`logout.php`)

Beyond that, all of the heavy lifting happens in one tab, instead of six:

---

1. Fetching "Who am I?" for the Sidebar

```
$user_id_for_sidebar = $_SESSION['id'];
$sidebar_user_stmt = $conn->prepare(
  "SELECT username, avatar_path
     FROM users
    WHERE id = ?"
);
$sidebar_user_stmt->bind_param("i", $user_id_for_sidebar);
$sidebar_user_stmt->execute();
$sidebar_user = $sidebar_user_stmt->get_result()->fetch_assoc();
$sidebar_user_stmt->close();
```

This mirrors the admin's fetch-your-own-info step, but only grabs your username and avatar.

---

2. Checking Which "Session" Is Active

```
$active_session_query = $conn->query(
  "SELECT id
     FROM sessions
    WHERE is_active = 1
    LIMIT 1"
);
$active_session_id = $active_session_query
                       ->fetch_assoc()['id']
                    ?? null;
```

Same as admin, but we don’t let the user flip sessions—just read which one is live.

---

3. Handling a Seat-Request POST

```
if ($_SERVER["REQUEST_METHOD"]==="POST"
    && isset($_POST['request_seat']))
{
    $seat_id = $_POST['seat_id'];
    $user_id = $_SESSION['id'];

    // Only allow if it’s still marked “available” in DB
    $check_sql  = "SELECT status
                     FROM seats
                    WHERE id = ?
                      AND status = 'available'
                      AND session_id = ?";
    $check_stmt = $conn->prepare($check_sql);
    $check_stmt->bind_param("ii", $seat_id, $active_session_id);
    $check_stmt->execute();
    $check_stmt->store_result();

    if ($check_stmt->num_rows === 1) {
        $update_sql = "UPDATE seats
                          SET status = 'requested',
                              requested_by_id = ?
                        WHERE id = ?";
        $stmt = $conn->prepare($update_sql);
        $stmt->bind_param("ii", $user_id, $seat_id);
        $stmt->execute();
        $stmt->close();
    }
    $check_stmt->close();

    // Refresh the page so the grid updates immediately
    header("Location: user_dashboard.php");
    exit;
}
```

This is the only write-operation a user can do: request an available seat.

---

4. Pulling Down the Seat Map

```
$seats_result = null;
if ($active_session_id) {
    $stmt = $conn->prepare(
      "SELECT id, seat_number, status, requested_by_id, user_id
         FROM seats
        WHERE session_id = ?
     ORDER BY seat_number"
    );
    $stmt->bind_param("i", $active_session_id);
    $stmt->execute();
    $seats_result = $stmt->get_result();
    $stmt->close();
}
```

Much like the admin’s seat listing, but we don’t JOIN on requestors or show every column—just enough to render our grid.

---

5. The Seat-Grid UI

Below your `<head>` you have an embedded `<style>` block defining:

```
.seat-available   { background-color: #10B981; /* green */ }
.seat-requested   { background-color: #F59E0B; /* amber */ }
.seat-booked      { background-color: #EF4444; /* red */ }
.seat-reserved    { background-color: #6B7280; /* gray */ }
.seat-mine-requested { border: 3px solid #3B82F6; /* blue */ }
.seat-mine-booked    { border: 3px solid #8B5CF6; /* violet */ }
```

Then in your `<main>`:

```
<div class="grid grid-cols-5 gap-4">
  <?php if($seats_result): ?>
    <?php while($seat = $seats_result->fetch_assoc()): ?>
      <?php
        // Build the CSS class
        $seat_class = 'seat-' . $seat['status'];
        if ($seat['status']=='requested'
            && $seat['requested_by_id']==$_SESSION['id']) {
          $seat_class .= ' seat-mine-requested';
        } elseif ($seat['status']=='booked'
            && $seat['user_id']==$_SESSION['id']) {
          $seat_class .= ' seat-mine-booked';
        }
      ?>
      <?php if ($seat['status']=='available'): ?>
        <!-- Clickable form button for your request -->
        <form method="post" action="user_dashboard.php">
          <input type="hidden" name="seat_id" value="<?= $seat['id'] ?>">
          <button name="request_seat" class="seat <?= $seat_class ?>">
            <?= htmlspecialchars($seat['seat_number']) ?>
          </button>
        </form>
      <?php else: ?>
        <!-- Non-clickable: your requests/bookings carry a special border -->
        <div class="seat <?= $seat_class ?>">
          <?= htmlspecialchars($seat['seat_number']) ?>
        </div>
      <?php endif; ?>
    <?php endwhile; ?>
  <?php else: ?>
    <p class="col-span-5 text-center text-gray-500">
      No seats configured for the active session.
    </p>
  <?php endif; ?>
</div>
```

- **Available seats** render as a `<button>` inside a `<form>` so POST→this same page.
- **Other statuses** show as plain `<div>`, styled but not clickable.
- **Your own requests/bookings** get an extra colored border so you can see what you've asked for (or been assigned).

---

6. Sidebar & Footer

The sidebar is a much–slimmed-down copy of the admin's:

```
<aside class="w-64 bg-gray-800 text-white p-4 flex flex-col justify-between">
  <!-- Logo & Title -->
  <!-- "Book a Seat" link (this page) -->
  <!-- "My Profile" link -->
  <!-- "Log Out" link -->
  <!-- Your avatar & username at the bottom -->
</aside>
```

And the footer is identical to admin—just a static signature.

In Summary

- **Structure & Styling**: exactly the same Tailwind and layout approach.
- **Session & Security**: copied verbatim up front.
- **Data-Loading**: a pared-down version (only your user info + seat list).
- **Actions**: only one—request an "available" seat. No tabs, no bill-clearing, no user management.

Functionally, **user_dashboard.php** is the mirror image of **admin_dashboard.php** in look and feel, but with *one* area of interactivity instead of six—and that interactivity is locked down to “request a seat” only.

Here’s a step-by-step breakdown of **billing.php**, which lets a logged-in user view their own payment history:

---

1. Session Bootstrapping & Security

```
<?php
session_start();
require_once "db_connect.php";

// Only users (not guests or admins) may access this page:
if (
    !isset($_SESSION["loggedin"])
    || $_SESSION["loggedin"] !== true
    || $_SESSION["role"] !== 'user'
) {
    header("location: index.php");
    exit;
}
```

1. **`session_start()`** resumes the user’s session so we can read `$_SESSION['id']` and `$_SESSION['role']`.
2. We immediately check that the user is both
   - logged in, *and*
   - has the role `'user'`.
3. If not, we redirect them back to the login page.

---

2. Loading Sidebar User Info

```
$user_id = $_SESSION['id'];

$sidebar_user_stmt = $conn->prepare(
    "SELECT username, avatar_path
       FROM users
      WHERE id = ?"
);
$sidebar_user_stmt->bind_param("i", $user_id);
$sidebar_user_stmt->execute();
$sidebar_user = $sidebar_user_stmt
                    ->get_result()
                    ->fetch_assoc();
$sidebar_user_stmt->close();
```

- We fetch your **username** and **avatar file path** to display in the sidebar.
- Uses a **prepared statement** (`->prepare() + ->bind_param()`) to guard against SQL injection.

---

3. Fetching the User's Payment Slips

```
$payment_slips_sql = "
  SELECT *
    FROM payments
   WHERE user_id = ?
ORDER BY generated_at DESC";
$stmt = $conn->prepare($payment_slips_sql);
$stmt->bind_param("i", $user_id);
$stmt->execute();
$payment_slips_result = $stmt->get_result();
```

- Pulls *all* payment records for this user, sorted newest first.
- `$payment_slips_result` will drive our table in the HTML below.

---

4. HTML Head & Tailwind Setup

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My Billing - SV Library System</title>
  <script src="https://cdn.tailwindcss.com"></script>
  <!-- Optional: custom font from Google -->
  <link href="https://fonts.googleapis.com/css2?family=Inter&display=swap"
rel="stylesheet">
  <style> body { font-family: 'Inter', sans-serif; } </style>
</head>
```

- Loads Tailwind via CDN for utility-first styling.
- Sets a clean sans-serif font for the whole page.

---

5. Sidebar Navigation

```
<aside class="w-64 bg-gray-800 text-white min-h-screen p-4 flex flex-col
justify-between">
  <div>
    <!-- Logo & Title -->
    <div class="flex items-center justify-center mb-10">
      <svg class="w-8 h-8 mr-2 text-indigo-400">…</svg>
      <h2 class="text-2xl font-bold">SV Library</h2>
    </div>

    <!-- Nav Links -->
    <nav>
      <a href="user_dashboard.php" class="flex items-center p-2 rounded hover:bg-gray-700">
        <!-- icon --> Book a Seat
      </a>
      <a href="profile.php" class="flex items-center p-2 rounded hover:bg-gray-700">
        <!-- icon --> My Profile
      </a>
      <a href="billing.php" class="flex items-center p-2 rounded bg-gray-700 text-white">
        <!-- icon --> My Billing
      </a>
    </nav>
  </div>

  <!-- Bottom: Avatar, Username, Logout -->
  <div>
    <a href="profile.php" class="flex items-center p-2 rounded hover:bg-gray-700">
      <img src="<?= htmlspecialchars($sidebar_user['avatar_path']) ?>"
           alt="User Avatar" class="w-10 h-10 rounded-full mr-3">
      <span class="font-semibold"><?= htmlspecialchars($sidebar_user['username']) ?></span>
    </a>
    <a href="logout.php" class="flex items-center w-full p-2 mt-4 rounded bg-red-600 hover:bg-red-700">
      <!-- icon --> Log Out
    </a>
  </div>
</aside>
```

- Mirrors the admin sidebar but with only three links.
- Highlights **My Billing** by giving it a darker background.

---

6. Main Content: Billing Table

```
<main class="flex-1 p-10">
  <h1 class="text-3xl font-bold mb-6">My Billing History</h1>
  <div class="bg-white p-8 rounded-lg shadow-lg">
    <table class="min-w-full bg-white">
      <thead class="bg-gray-200">
        <tr>
          <th class="py-3 px-4 text-left uppercase text-xs font-semibold">Invoice ID</th>
          <th class="py-3 px-4 text-left uppercase text-xs font-
semibold">Seat</th>
          <th class="py-3 px-4 text-left uppercase text-xs font-
semibold">Period</th>
          <th class="py-3 px-4 text-left uppercase text-xs font-
semibold">Amount</th>
          <th class="py-3 px-4 text-left uppercase text-xs font-
semibold">Actions</th>
        </tr>
      </thead>
      <tbody class="text-gray-700">
        <?php if ($payment_slips_result->num_rows > 0): ?>
          <?php while ($slip = $payment_slips_result->fetch_assoc()): ?>
            <tr class="border-t">
              <td class="py-3 px-4">#<?= htmlspecialchars($slip['id'])
?></td>
              <td class="py-3 px-4"><?=
htmlspecialchars($slip['seat_number']) ?></td>
```

- Sets up a responsive table with Tailwind classes for padding, borders, and typography.
- Table headers are uppercase, small caps for clarity.

---

7. Period & Amount Logic

```
<?php
// Sometimes date_of_joining can be '0000-00-00'—we guard against that.
if ($slip['date_of_joining'] && $slip['date_of_joining'] !== '0000-00-00'):
?>
  <td class="py-3 px-4">
    <?= htmlspecialchars($slip['date_of_joining']) ?>
    – <?= htmlspecialchars($slip['date_of_leaving']) ?>
  </td>
  <td class="py-3 px-4">₹<?= number_format($slip['amount_due'], 2) ?></td>
<?php else: ?>
  <td class="py-3 px-4 text-red-500 font-medium">Invalid Period</td>
  <td class="py-3 px-4 text-red-500 font-medium">Calculation Error</td>
<?php endif; ?>
```

- **Validation check**: if the joining date is missing or zeroed out, we show a red “Invalid Period” / “Calculation Error” message instead of bad data.

---

8. View Slip Action

```
<td class="py-3 px-4">
  <a href="view_slip.php?id=<?= $slip['id'] ?>"
     class="bg-indigo-600 text-white font-bold py-1 px-3 rounded text-xs
hover:bg-indigo-700">
    View Slip
  </a>
</td>
```

- Provides a button to drill down into **view_slip.php**, passing the slip's ID via `?id=`.

---

9. No-History Fallback & Closing Tags

```
        <?php endwhile; ?>
      <?php else: ?>
        <tr>
          <td colspan="5" class="py-4 text-center text-gray-500">
            You have no payment history.
          </td>
        </tr>
      <?php endif; ?>
    </tbody>
  </table>
</div>
```

- If the user has **no** records, we render a single row saying "You have no payment history."
- Then we close out `</main>`, `</body>`, and `</html>`.

Why It Matters

- **Role Enforcement** ensures only regular users (not admins!) see their own billing data.
- **Consistent UI** uses the same utility classes as the dashboards, so the look-and-feel stays uniform.
- **Data Validation** protects against malformed dates.
- **Action Links** let users view detailed slips in a separate page.

Let me know which file you'd like to tackle next—**profile.php**, **view_slip.php**, or perhaps the password-hash generator utility (`generate_password_hash.php`)?

1. File Purpose

- **Location:** `generate_password_hash.php` in your project root.
- **Use case:** Quickly generate a `password_hash()` output for a known plaintext password (e.g. when seeding users or resetting an admin password).

2. PHP Backend Logic

```
<?php
// generate_password_hash.php
// A simple tool to generate a secure password hash.

$generated_hash = '';

// 1. Only run when the form POSTs back to this script
if ($_SERVER["REQUEST_METHOD"] == "POST") {

    // 2. Ensure the password field isn’t empty
    if (!empty($_POST['password'])) {
        // Capture the submitted plaintext
        $password_to_hash = $_POST['password'];

        // 3. Generate a bcrypt-based hash using PHP’s default settings
        $generated_hash = password_hash(
            $password_to_hash,
            PASSWORD_DEFAULT
        );
    }
}
?>
```

1. **`$generated_hash = ''`**
   Initialized so we can later check if a hash was produced.
2. **`$_SERVER["REQUEST_METHOD"] == "POST"`**
   Guards so that the hash only generates when you submit the form.
3. **`password_hash(..., PASSWORD_DEFAULT)`**
   Uses the current recommended algorithm (bcrypt by default), automatically generating a secure salt.

3. HTML Form (Frontend)

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

```
<head>
  <meta charset="UTF-8">
  <title>Generate Password Hash</title>
  <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="bg-gray-100 flex items-center justify-center h-screen">

  <div class="bg-white p-8 rounded shadow-md w-full max-w-sm">
    <h1 class="text-2xl font-bold mb-4 text-center">Password Hash
Generator</h1>

    <!-- 4. Form to accept a plaintext password -->
    <form action="generate_password_hash.php" method="post" class="space-y-
4">
      <div>
        <label for="password" class="block text-gray-700 mb-1">Password to
Hash:</label>
        <input
          type="text"
          name="password"
          id="password"
          required
          class="w-full px-3 py-2 border rounded focus:outline-none
focus:ring-2 focus:ring-indigo-500"
        >
      </div>
      <button
        type="submit"
        class="w-full bg-indigo-600 hover:bg-indigo-700 text-white py-2
rounded"
      >
        Generate Hash
      </button>
    </form>

    <!-- 5. Show the generated hash once it's created -->
    <?php if (!empty($generated_hash)): ?>
      <div class="mt-6 p-4 bg-green-50 border-l-4 border-green-500 rounded">
        <h2 class="font-semibold text-green-700">Generated Hash:</h2>
        <p class="text-sm text-gray-600 mt-2">
          Copy this value into the <code>password</code> column for your user
in phpMyAdmin.
        </p>
        <textarea
          readonly
          rows="3"
          class="w-full mt-2 p-2 border rounded bg-gray-100 font-mono break-
all"
        ><?= htmlspecialchars($generated_hash) ?></textarea>
      </div>
    <?php endif; ?>
  </div>

</body>
</html>
```

1. **Tailwind CSS** for quick, responsive styling.
2. **Form** with a single text input (`name="password"`) and a submit button.
3. **Conditional block** (`<?php if (!empty($generated_hash)): ?>`) that only appears after you generate a hash.
4. **`<textarea readonly>`** to display the hash, with `htmlspecialchars()` to prevent any HTML injection.

---

4. How to Use It

1. **Navigate** in your browser to `http://localhost/librarysda/generate_password_hash.php`.
2. **Type** the plaintext password you want to hash.
3. **Click** “Generate Hash.” The page reloads and shows your new bcrypt hash.
4. **Copy** the hash and paste it into the `password` field for a user in your `users` table (e.g., via phpMyAdmin).

---

5. Security Notes

- **Never store plaintext.** Always use `password_hash()` and verify with `password_verify()`.
- **PASSWORD_DEFAULT** may change in future PHP versions to a stronger algorithm; your code stays up-to-date.
- This utility is meant for one-off use by an administrator; it does not itself authenticate or log you in.

---

1. My Billing History (`billing.php`)

You’ve already seen most of this, but here’s a quick recap of its flow:

1. **Session & Role Check**

```
session_start();
require_once "db_connect.php";
if (!isset($_SESSION["loggedin"])
    || $_SESSION["loggedin"]!==true
    || $_SESSION["role"]!=='user') {
    header("location: index.php");
    exit;
}
```

2. **Fetch Sidebar Info** (username & avatar) for display.
3. **Query This User's Payments**

   ```
   $sql = "SELECT p.*, s.seat_number
             FROM payments p
             JOIN seats s ON p.seat_id = s.id
            WHERE p.user_id = ?
         ORDER BY p.generated_at DESC";
   $stmt = $conn->prepare($sql);
   $stmt->bind_param("i", $_SESSION['id']);
   $stmt->execute();
   $payment_slips = $stmt->get_result();
   ```

4. **Render a Tailwind-styled table** listing:
   - **Invoice ID** (`$slip['id']`)
   - **Seat Number** (`$slip['seat_number']`)
   - **Period** (`date_of_joining – date_of_leaving`)
   - **Amount Due** (`₹<?= number_format($slip['amount_due'],2) ?>`)
   - **"View Slip" button** linking to `view_slip.php?id=…`

---

2. Single Invoice View (`view_slip.php`)

This page pulls one slip by its `id` and displays a printer-friendly invoice.

a) Session & Role Guard

```
<?php
session_start();
require_once "db_connect.php";

if (!isset($_SESSION["loggedin"])
    || $_SESSION["loggedin"] !== true
    || $_SESSION["role"] !== 'user') {
    header("location: index.php");
    exit;
}
```

b) Fetch the Slip by ID

```
// 1. Get & sanitize the slip ID
$slip_id = isset($_GET['id']) ? (int)$_GET['id'] : 0;

// 2. Prepare the query: join payments, seats, sessions, and optionally users
$sql = "
  SELECT p.*,
         s.seat_number,
         ses.session_name,
         u.username,
         u.full_name,
```

```
         u.email
    FROM payments p
    JOIN seats s     ON p.seat_id    = s.id
    JOIN sessions ses ON p.session_id = ses.id
    JOIN users u     ON p.user_id    = u.id
   WHERE p.id = ?
     AND p.user_id = ?";
$stmt = $conn->prepare($sql);
$stmt->bind_param("ii", $slip_id, $_SESSION['id']);
$stmt->execute();
$slip = $stmt->get_result()->fetch_assoc();
$stmt->close();

if (!$slip) {
    // No slip found or belongs to another user
    echo "Invoice not found.";
    exit;
}
```

- We ensure the invoice belongs to the logged-in user by checking `p.user_id = ?`.

c) Invoice HTML Layout

Below the PHP you'll have something like:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Invoice #<?= $slip['id'] ?> – SV Library</title>
  <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="p-10 bg-gray-100">

  <div class="max-w-2xl mx-auto bg-white p-8 rounded shadow">
    <!-- Header -->
    <div class="flex justify-between mb-8">
      <div>
        <h1 class="text-3xl font-bold">Invoice #<?= $slip['id'] ?></h1>
        <p class="text-sm text-gray-500">
          Generated: <?= htmlspecialchars($slip['generated_at']) ?>
        </p>
      </div>
      <div class="text-right">
        <h2 class="font-semibold"><?= htmlspecialchars($slip['full_name'])
?></h2>
        <p><?= htmlspecialchars($slip['email']) ?></p>
      </div>
    </div>

    <!-- Invoice Details -->
    <table class="w-full mb-6">
      <tr class="bg-gray-50">
        <th class="p-2 text-left">Session:</th>
        <td class="p-2"><?= htmlspecialchars($slip['session_name']) ?></td>
      </tr>
      <tr>
        <th class="p-2 text-left">Seat Number:</th>
        <td class="p-2"><?= htmlspecialchars($slip['seat_number']) ?></td>
      </tr>
      <tr class="bg-gray-50">
        <th class="p-2 text-left">Joining Date:</th>
        <td class="p-2"><?= htmlspecialchars($slip['date_of_joining'])
?></td>
      </tr>
      <tr>
        <th class="p-2 text-left">Leaving Date:</th>
        <td class="p-2"><?= htmlspecialchars($slip['date_of_leaving'])
?></td>
      </tr>
      <tr class="bg-gray-50">
        <th class="p-2 text-left">Duration (days):</th>
        <td class="p-2"><?= $slip['duration_days'] ?></td>
      </tr>
      <tr>
        <th class="p-2 text-left">Amount Due:</th>
        <td class="p-2 font-semibold">₹<?=
number_format($slip['amount_due'],2) ?></td>
      </tr>
    </table>

    <!-- Footer & Print Button -->
    <div class="flex justify-between items-center">
      <p class="text-xs text-gray-500">Thank you for choosing SV Library.</p>
      <button
        onclick="window.print()"
        class="bg-indigo-600 text-white px-4 py-2 rounded hover:bg-indigo-
700">
        Print Invoice
      </button>
    </div>
  </div>

</body>
</html>
```

Key Points

- **Structured Table** shows all invoice fields clearly.
- **`window.print()`** gives a one-click print option.
- **Tailwind** ensures clean spacing, alternating row backgrounds, and responsive layout.

Overview of Billing Functionality

1. **`billing.php`** — lets the user see all their invoices and click to view any one.

2. **`view_slip.php`** — securely fetches a single invoice (ensuring it belongs to them) and renders a printer-friendly invoice page.

Together, they give full read-only billing capabilities to your end users. Let me know if you’d like to go deeper into any of these components!

---

1. Profile Management (`profile.php`)

- **What it does**: Lets users view and update their personal info (full name, email) and change their avatar.
- **Key points to review**:
  - File-upload handling (size/type checks, storing under `/uploads/avatars/`)
  - Resizing or sanitizing uploaded images
  - Updating the `users` table with new avatar path
  - Validation and error-feedback

---

2. Session Teardown (`logout.php`)

- **Simple but critical**: Calls `session_start()`, then `session_unset()` and `session_destroy()`, and finally redirects back to login.
- **Consider**:
  - Clearing cookies (if you ever set "remember me" cookies)
  - A "Goodbye" message or landing page

---

3. Database Schema & Indexes (`library_system.sql`)

- **Review your tables**:
  - `users`, `sessions`, `seats`, `bookings`, `payments`
- **Check**:
  - Are your foreign keys and cascading deletes set up?
  - Do you have indexes on columns you search/filter (e.g. `session_id`, `user_id`)?
  - Any missing NOT NULL or default values?

---

4. Security Hardening

- **CSRF protection**: Add tokens to all your POST forms.
- **Input sanitization**: Double-check every `$_GET` and `$_POST` use—even in pagination or tab switches.
- **HTTPS & Headers**:
  - Enforce HTTPS (HSTS) in production
  - Send security headers (`Content-Security-Policy`, `X-Frame-Options`, etc.)
- **Error handling & logging**:

  - Don't echo raw DB errors to users
  - Log errors to a file or monitoring service

---

5. Password Recovery & Account Management

- **Forgot-password flow**:
  - "Send me a reset link" via email
  - Temporary tokens in a `password_resets` table
- **Email integration**: Use PHPMailer or similar to send invites, notifications or receipts.

---

6. Deployment & Configuration

- **Move credentials out of code**:
  - Use a `.env` file or environment variables
  - Load them via `getenv()` or a library like phpdotenv
- **Directory permissions**:
  - Ensure `/uploads/` is writeable but not browsable
  - Lock down config files

---

7. UX/UI Enhancements

- **Responsive tweaks**: Ensure tables and grids collapse nicely on phones.
- **Charts & Dashboards**: Add a small occupancy chart (e.g. using Chart.js) on the admin overview.
- **Notifications**: Real-time alerts (via Pusher or AJAX) when a new seat request comes in.

Next Steps

Which of these would you like to dig into?

1. **Profile.php** — walk through the avatar upload and info-update logic.
2. **Logout.php** — ensure a clean, secure logout.
3. **Schema review** — optimize your SQL file and indexes.
4. **Security hardening** — CSRF, HTTPS, headers, and error logging.
5. **Password reset flow** — add "forgot password" + email.
6. **Deployment/config** — moving creds to `.env`, locking down file permissions.

FUTURE ENHANCEMENTS—GROUPED BY CATEGORY—TO TAKE YOUR LIBRARY SEAT-BOOKING SYSTEM TO THE NEXT LEVEL:

---

1. Architecture & Code Quality

- **Adopt an MVC framework** (e.g. Laravel or Symfony)
    - Enforces separation of concerns, makes routing, middleware, and testing easier.
- **Use Composer for dependencies**
    - Manage third-party libraries (PHPMailer, dotenv, PHPUnit, etc.) and autoloading.
- **PSR Standards & Static Analysis**
    - Apply PSR-12 for code style; integrate PHPStan or Psalm to catch bugs early.
- **Automated Testing**
    - Write unit tests for your core services (seat allocation, billing calculations) and feature tests for your controllers.

---

2. Security & Robustness

- **CSRF Protection**
    - Add per-form CSRF tokens (Laravel's `@csrf` or your own implementation).
- **Input Sanitization & Validation**
    - Centralize validation rules (e.g. Laravel Form Requests or a validation library).
- **Environment Configuration**
    - Move DB credentials and feature flags into a `.env` file (via phpdotenv) instead of hard-coding.
- **Role & Permission Management**
    - Build a flexible ACL (admin, librarian, super-admin) so you can add roles later without rewriting code.
- **Rate-Limiting & Brute-Force Protection**
    - Throttle login attempts per IP or per user to prevent credential-stuffing attacks.

---

3. User Experience & UI/UX

- **Responsive & Mobile-First Design**
    - Ensure tables collapse gracefully; consider a hamburger menu for the sidebar on small screens.
- **Real-Time Updates**
    - Use AJAX polling or WebSockets (e.g. Pusher, Laravel Echo) so users see seat-availability changes instantly.

- **Drag-and-Drop Seat Selection**
    - Let users click-and-drag to book multiple seats in one go.
- **Offline Support**
    - With a Service Worker & localStorage, let users draft requests even with flaky connections.

---

4. New Features & Integrations

- **Online Payments**
    - Integrate Stripe, PayPal or Razorpay so users can pay immediately rather than cash-on-departure.
- **Email / SMS Notifications**
    - Notify users when their seat request is approved, a session is created, or their invoice is due.
- **Calendar Integration**
    - Offer "Add to Google Calendar" links for session start/end dates.
- **PDF Invoice Generation**
    - Generate and email PDF invoices (e.g. using DomPDF or Snappy) instead of just HTML slips.
- **Analytics Dashboard**
    - Show admins charts (occupancy over time, revenue by session) via Chart.js or Recharts.

---

5. Data Management & DevOps

- **Automated Backups & Archiving**
    - Schedule nightly dumps of your MySQL database and rotate old backups.
- **CI/CD Pipeline**
    - Use GitHub Actions or GitLab CI to lint, test, and deploy your code automatically.
- **Containerization with Docker**
    - Dockerize your app and database for consistent dev/staging/production environments.
- **Monitoring & Logging**
    - Ship logs to a central service (Papertrail, Loggly) and set up uptime/health alerts.

---

6. Accessibility & Internationalization

- **WCAG Compliance**

    - Ensure all forms, tables, and buttons meet ARIA and keyboard-navigation standards.
- **Multi-Language Support**
    - Externalize strings and offer Hindi/English toggles (e.g. with the `gettext` extension or a localization library).

GITHUB LINK :

himanshuSinghworkPort/2k25_industrial_training_projects: c, python, java, php basic level projects

GITHUB QR: