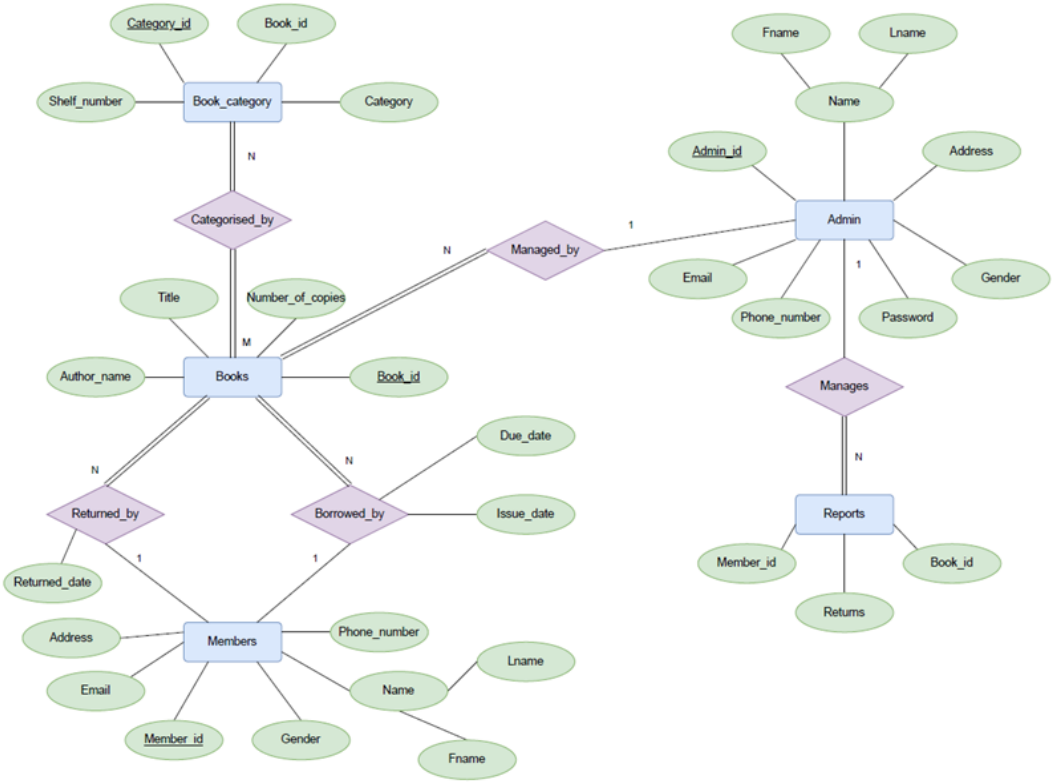
Library Management System

Description

- The Library Database keeps track of books, admin and library members.
- The database is organized into books, each book has a unique ID, a title, authors name, book category, number of copies and shelf number.
- Book category stores the category id, the book id and the category the books belong to and the shelf number where the book is present.
- The system keeps track of the borrowed books, its borrow id, along with the date it was issued and the due date. It stores the date it was returned and also calculates the fine if it exceeds the due date
- If a member has to pay fine, system stores the member id of the member along with its fine status and the fine amount to be paid.
- The member details such as member id, name, gender, address, phone number and email will be stored by the system.
- Admin stores its admin id, password, name, gender, address, phone number and email.
- The admin manages the reports such as book ids, book returns and giving membership to new users.
- The admin can update the number of available books, check if a member has paid fine or not, see the books that have been borrowed the most or the member who has borrowed the most books.
- The system has an authentication system the staff and members to log into the system using a user id and a password.

ER Diagram


Category_id
Book_id
Shelf_number
Book_category
Category
N
Categorised_by
Title
Number_of_copies
M
Author_name
Books
Book_id
N
N
Returned_by
Borrowed_by
Due_date
Issue_date
1
1
Returned_date
Address
Members
Phone_number
Email
Name
Lname
Member_id
Gender
Fname
Fname
Lname
Name
Admin_id
Address
Admin
Email
Phone_number
Password
Gender
1
N
Managed_by
1
Manages
N
Reports
Member_id
Book_id
Returns

Database Schema

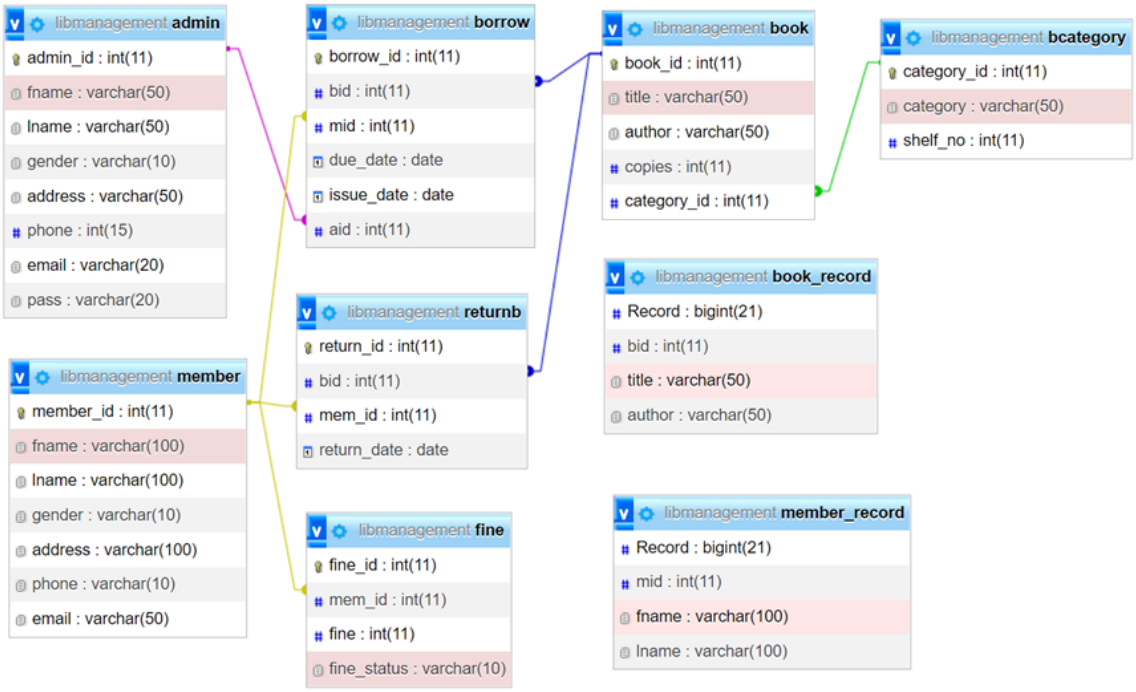


Figure 1: Admin Login

In the admin login page, admin has to enter the admin ID and password, and once validated that the id and password are correct, the admin logs in to the admin portal. The validation is

done using javascript for client side and php for server side validation

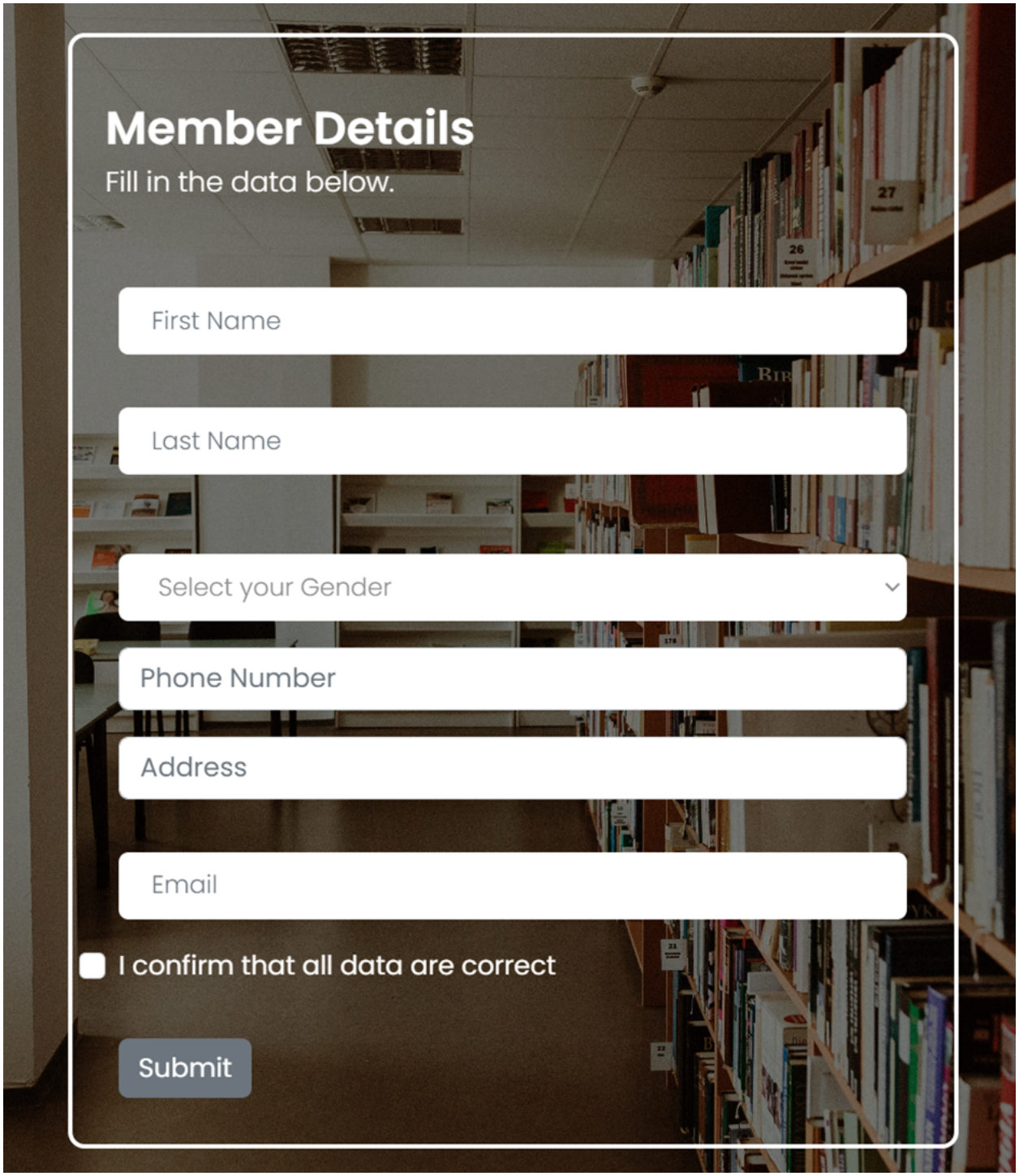


Figure 2: Add new member

This form is used to create new Members. It stores the first name , last name , gender ,phone number, email and address of the member.

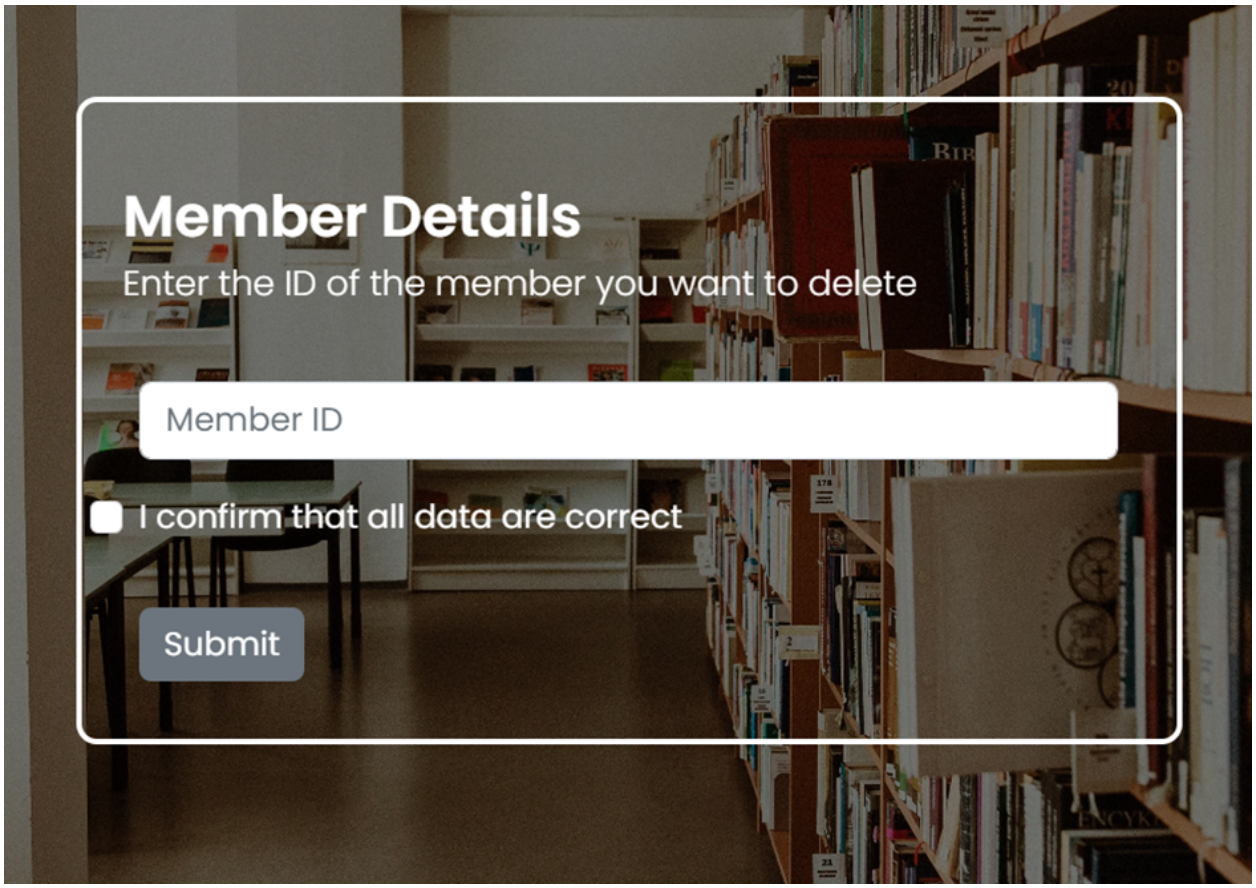


Figure 3: Delete member

If the admin needs to delete membership , he/she must enter the member Id of the member that will be deleted

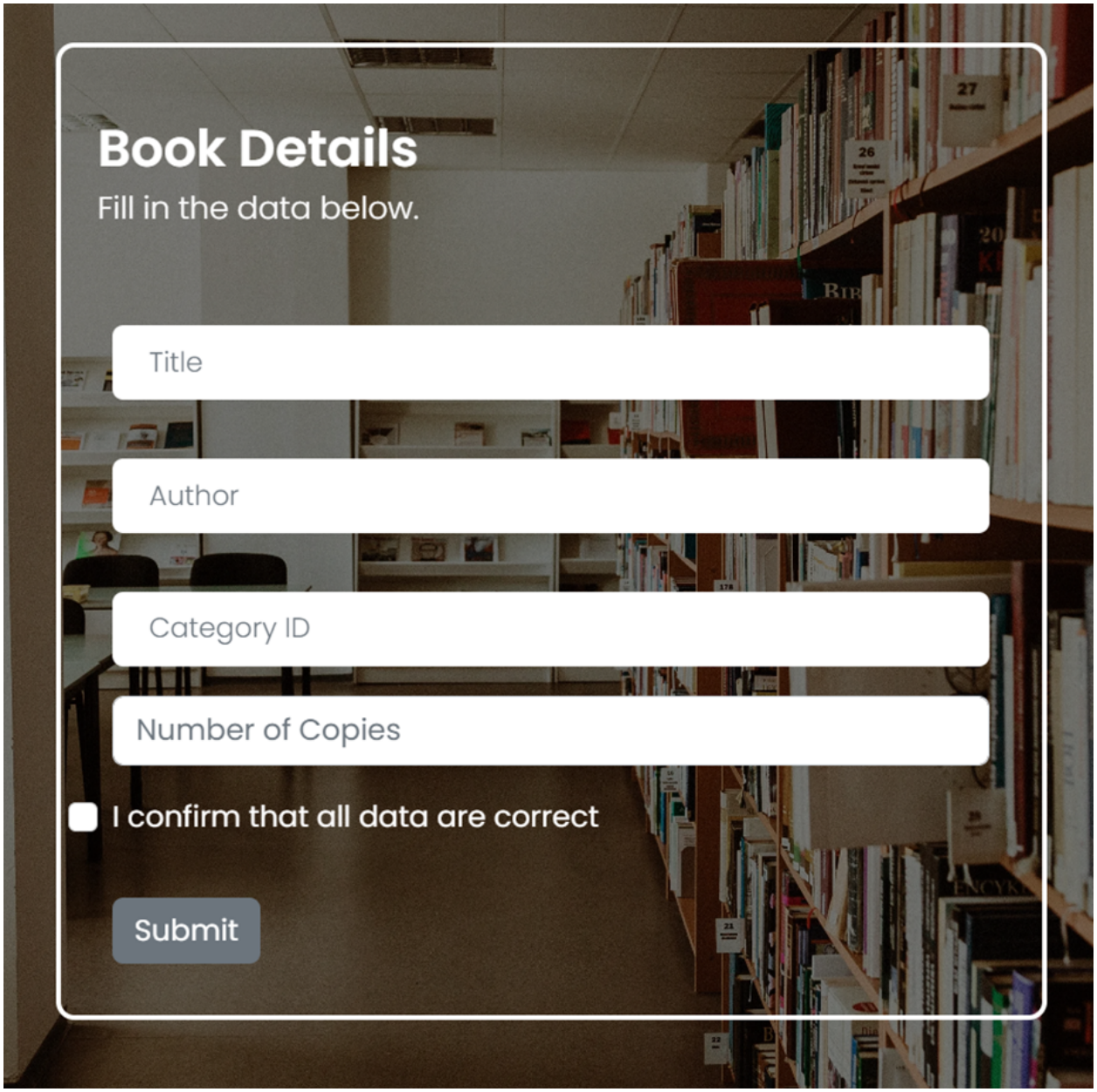


Figure 4: Enter new book

To add a new book into the system , the admin must enter the title , author, category id and number of copies available

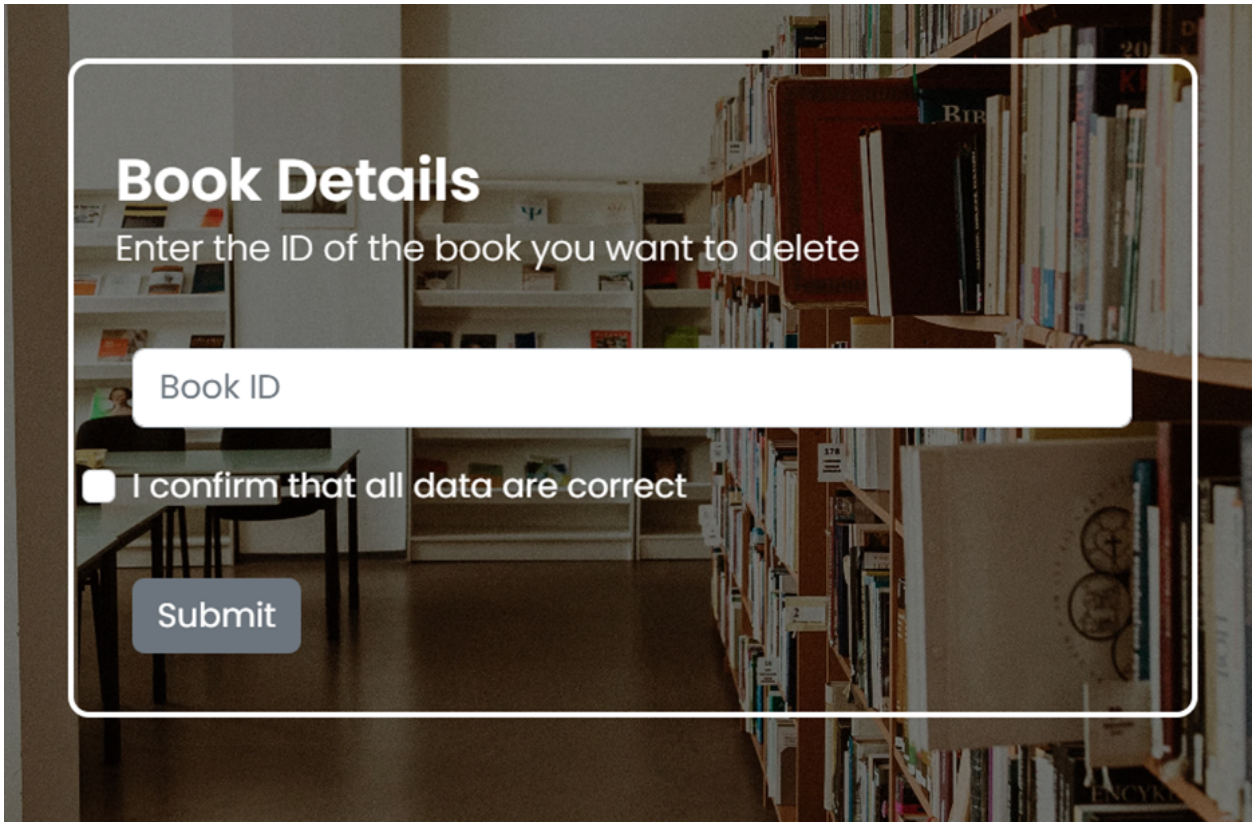


Figure 5: Delete book

If a book must be deleted from the system , the admin should enter the book id of that particular book

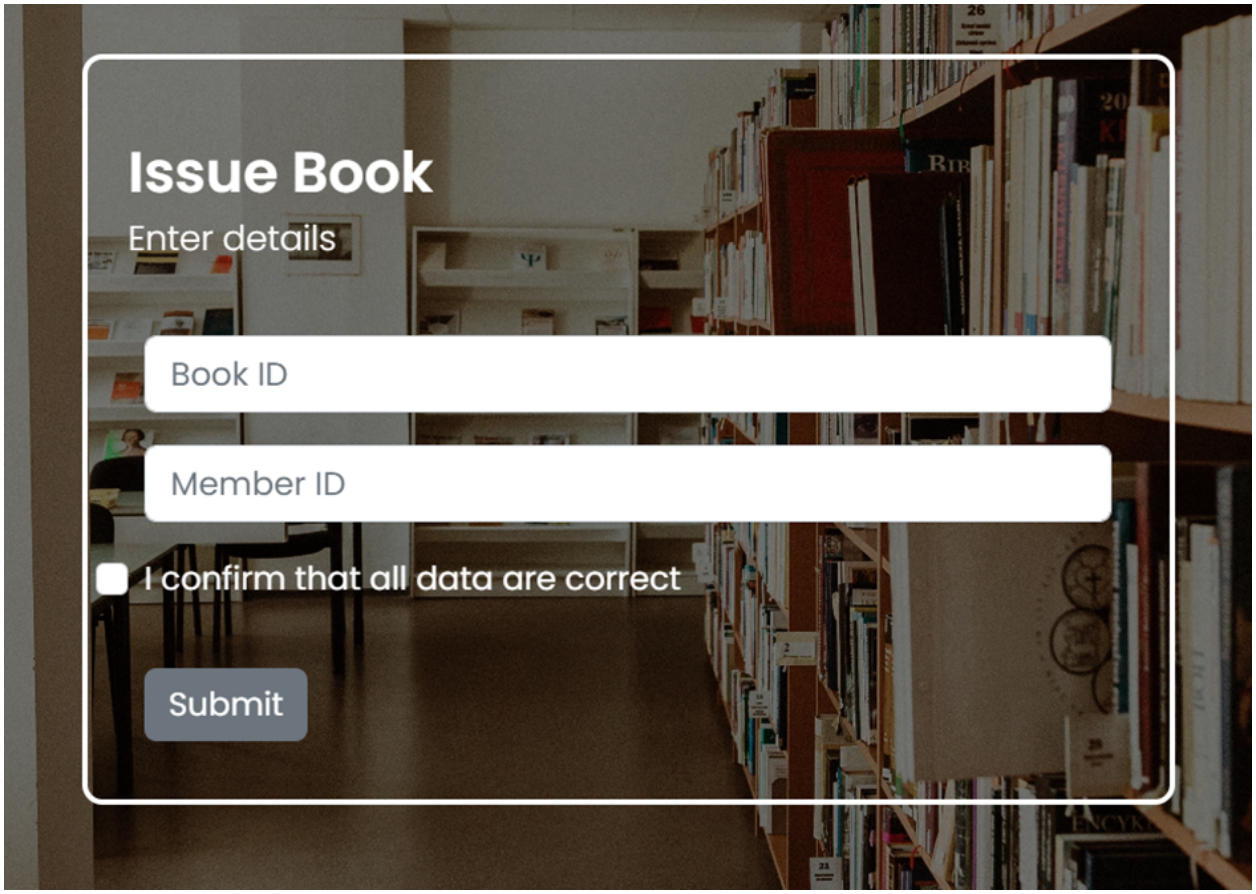


Figure 6: Issue book

To issue a book requested by member, the admin must enter the book id and the member Id of the member requesting the book.

Figure 7: Return book

If the member has returned the book , the admin must update the system by entering the book id and the member Id of the member returning the book

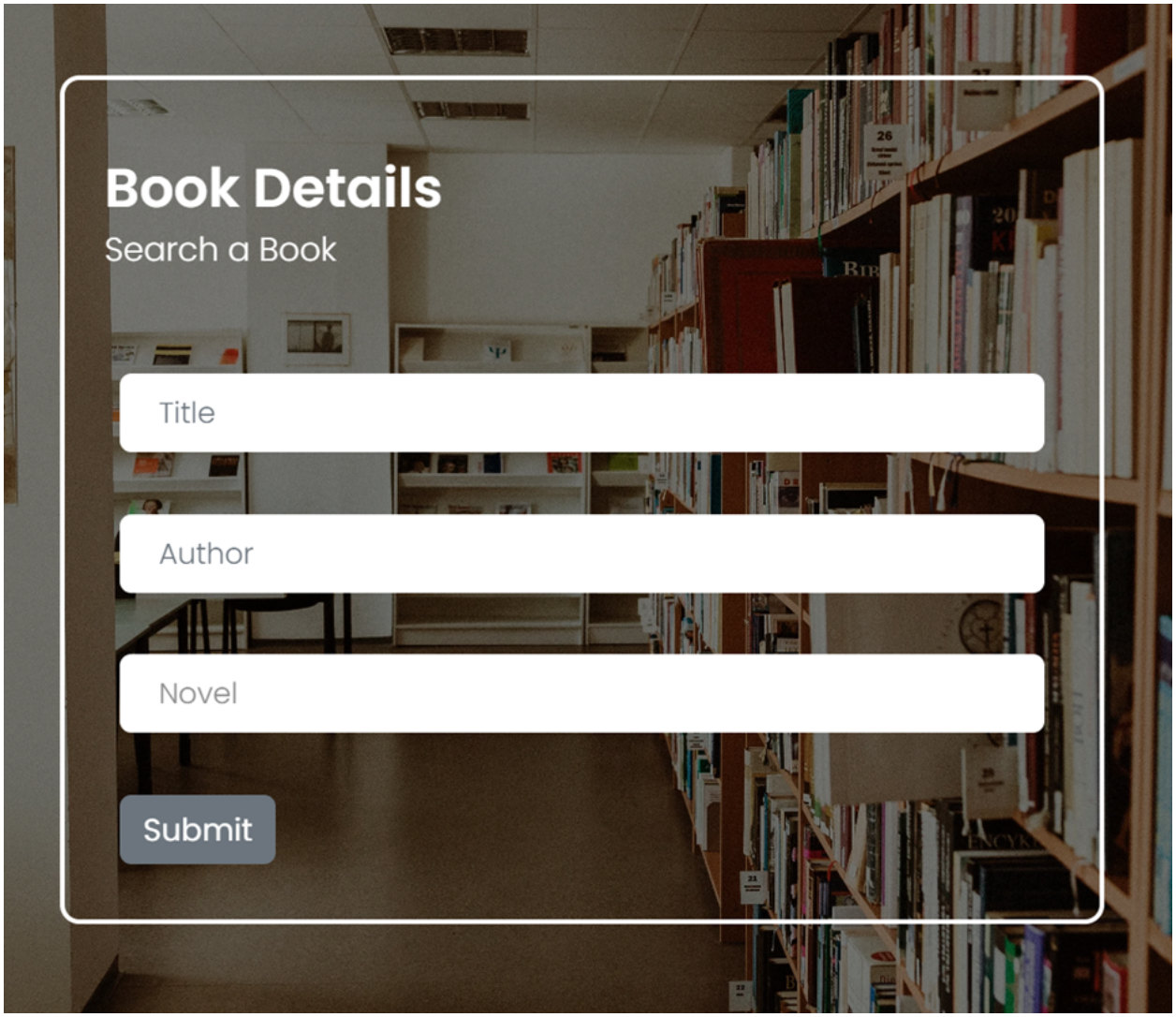


Figure 8: Search book

This form is used by the member to search for a book present in the library. The member can search a book either by title, author or category. Once submitted this form displays the details of the book

All the above forms are coded using HTML, CSS, Javascript and PHP.

Form validation is done on both server side and client side. Client side validation is done using Javascript and server side validation is done using PHP

The book id will be present on the book itself
And the member id will be present on the membership card issued to the member

DDL:

```
CREATE DATABASE `libmanagement`

CREATE TABLE `admin` (
  `admin_id` int(11) NOT NULL,
  `fname` varchar(50) NOT NULL,
  `lname` varchar(50) NOT NULL,
  `gender` varchar(10) NOT NULL,
  `address` varchar(50) NOT NULL,
  `phone` int(15) NOT NULL,
  `email` varchar(20) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE `adminlogin` (
  `username` varchar(50) NOT NULL,
  `password` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE `book` (
  `book_id` int(11) NOT NULL,
  `title` varchar(50) NOT NULL,
  `author` varchar(50) NOT NULL,
  `copies` int(11) DEFAULT NULL,
  `category_id` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE `member` (
  `member_id` int(11) NOT NULL,
  `fname` varchar(100) NOT NULL,
  `lname` varchar(100) NOT NULL,
  `gender` varchar(10) NOT NULL,
  `address` varchar(100) NOT NULL,
```

```
  `phone` varchar(10) DEFAULT NULL,
  `email` varchar(50) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE `bcategory` (
  `category_id` int(11) NOT NULL,
  `category` varchar(50) NOT NULL,
  `shelf_no` int(11) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE `borrow` (
  `borrow_id` int(11) NOT NULL,
  `bid` int(11) NOT NULL,
  `mid` int(11) NOT NULL,
  `due_date` date DEFAULT NULL,
  `issue_date` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE `returnb` (
  `return_id` int(11) NOT NULL,
  `bid` int(11) NOT NULL,
  `mem_id` int(11) NOT NULL,
  `return_date` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

// view `member_record`

CREATE VIEW book_record AS SELECT COUNT('bid') Record,
`bid`,`title`,`author` FROM borrow INNER JOIN book ON
borrow.bid=book.book_id GROUP BY bid ORDER BY Record DESC;

CREATE TABLE `member_record` (
`Record` bigint(21)
,`mid` int(11)
,`fname` varchar(100)
,`lname` varchar(100)
);
```

```
// view `book_record`

CREATE VIEW member_record AS SELECT COUNT('mid') Record,
`mid`,`fname`,`lname` FROM borrow INNER JOIN member ON
borrow.mid=member.member_id GROUP BY mid ORDER BY
Record DESC;

CREATE TABLE `book_record` (
`Record` bigint(21)
,`bid` int(11)
,`title` varchar(50)
,`author` varchar(50)
);

DELIMITER $$
CREATE TRIGGER `due_date` BEFORE INSERT ON `borrow` FOR
EACH ROW SET new.due_date = DATE_ADD(CURRENT_DATE,
INTERVAL 20 DAY)
$$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER `issue_book` AFTER INSERT ON `borrow` FOR
EACH ROW UPDATE book SET copies = copies - 1 WHERE book_id
= new.bid
$$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER `createfine` AFTER INSERT ON `returnb` FOR
EACH ROW BEGIN
 IF (DATEDIFF(now(),new.return_date) > 0)THEN
INSERT INTO fine
SET `fine_id`=new.`return_id`,
`mem_id`=new.`mem_id`,
`fine`=DATEDIFF(now(),new.return_date)*5 ,
`fine_status`="fine";
 END IF;
```

```
END
$$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER `return_book` AFTER INSERT ON `returnb`
FOR EACH ROW UPDATE book SET copies = copies + 1 WHERE
book_id = new.bid
$$
DELIMITER ;

DELIMITER $$
CREATE TRIGGER `returned` AFTER INSERT ON `returnb` FOR
EACH ROW DELETE FROM borrow WHERE bid = new.bid AND mid
= new.mem_id
$$
DELIMITER ;

ALTER TABLE `admin`
  ADD PRIMARY KEY (`admin_id`);

ALTER TABLE `bcategory`
  ADD PRIMARY KEY (`category_id`);

ALTER TABLE `book`
  ADD PRIMARY KEY (`book_id`),
  ADD KEY `fk_category_id` (`category_id`);

ALTER TABLE `borrow`
  ADD PRIMARY KEY (`borrow_id`),
  ADD KEY `fk_borrow_bid` (`bid`),
  ADD KEY `fk_borrow_mid` (`mid`);

ALTER TABLE `fine`
  ADD PRIMARY KEY (`fine_id`),
  ADD KEY `fk_fine_mid` (`mem_id`);

ALTER TABLE `member`
```

```
  ADD PRIMARY KEY (`member_id`);

ALTER TABLE `returnb`
  ADD PRIMARY KEY (`return_id`),
  ADD KEY `fk_return_mid` (`mem_id`),
  ADD KEY `fk_return_bid` (`bid`);

ALTER TABLE `book`
  ADD CONSTRAINT `fk_category_id` FOREIGN KEY
(`category_id`) REFERENCES `bcategory` (`category_id`) ON
DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE `borrow`
  ADD CONSTRAINT `fk_borrow_bid` FOREIGN KEY (`bid`)
REFERENCES `book` (`book_id`) ON DELETE CASCADE ON
UPDATE CASCADE,
  ADD CONSTRAINT `fk_borrow_mid` FOREIGN KEY (`mid`)
REFERENCES `member` (`member_id`) ON DELETE CASCADE
ON UPDATE CASCADE;

ALTER TABLE `fine`
  ADD CONSTRAINT `fk_fine_mid` FOREIGN KEY (`mem_id`)
REFERENCES `member` (`member_id`) ON DELETE CASCADE
ON UPDATE CASCADE;

ALTER TABLE `returnb`
  ADD CONSTRAINT `fk_return_bid` FOREIGN KEY (`bid`)
REFERENCES `book` (`book_id`) ON DELETE CASCADE ON
UPDATE CASCADE,
  ADD CONSTRAINT `fk_return_mid` FOREIGN KEY
(`mem_id`) REFERENCES `member` (`member_id`) ON
DELETE CASCADE ON UPDATE CASCADE;
COMMIT;
```

Project By:
Varad Kelkar 2014061
Swizel Antao 2014056