# Library system project documentation

Purpose; user friendly library system

Base design – Run the program., user is prompted to view books, borrow books, return books, exit the program

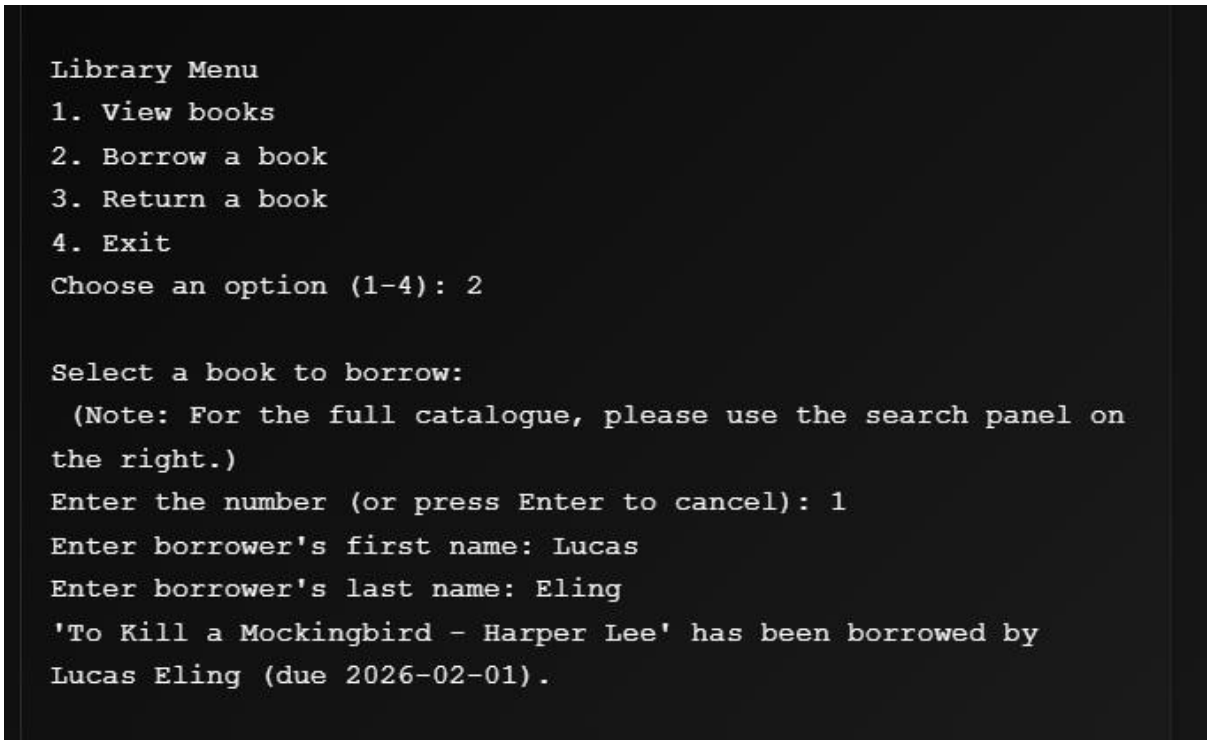
During runtime:

Press 1 to view available books

Press 2 to borrow a book, prompted for a book number (Easy to avoid typos), and then first and second name, book is then marked as borrowed and written to file

Press 3 to return a book, shows currently borrowed books, and prompts the user asking which one they would like to return.

Press 4 to end the program.




Version 1 of the code was made quickly, with a few functions forming the backbone of the program

```
available_books = [
    "To Kill a Mockingbird - Harper Lee",
    "1984 - George Orwell",
    "Pride and Prejudice - Jane Austen",
    "The Great Gatsby - F. Scott Fitzgerald",
    "Moby Dick - Herman Melville",
    "War and Peace - Leo Tolstoy",
    "Crime and Punishment - Fyodor Dostoevsky",
    "The Catcher in the Rye - J.D. Salinger",
    "Jane Eyre - Charlotte Brontë",
    "Brave New World - Aldous Huxley",
    "The Lord of the Rings - J.R.R. Tolkien",
    "Animal Farm - George Orwell",
    "Wuthering Heights - Emily Brontë",
    "The Odyssey - Homer",
    "The Iliad - Homer",
    "Les Misérables - Victor Hugo",
    "The Brothers Karamazov - Fyodor Dostoevsky",
    "Anna Karenina - Leo Tolstoy",
    "Don Quixote - Miguel de Cervantes",
    "One Hundred Years of Solitude - Gabriel García Márquez",
    "Fahrenheit 451 - Ray Bradbury",
    "The Kite Runner - Khaled Hosseini",
    "A Tale of Two Cities - Charles Dickens",
    "The Book Thief - Markus Zusak",
    "Great Expectations - Charles Dickens",
    "The Chronicles of Narnia - C.S. Lewis",
    "Dracula - Bram Stoker",
    "Frankenstein - Mary Shelley",
    "Catch-22 - Joseph Heller",
    "The Alchemist - Paulo Coelho",
    "The Little Prince - Antoine de Saint-Exupéry",
    "Lolita - Vladimir Nabokov",
    "Slaughterhouse-Five - Kurt Vonnegut",
    "The Grapes of Wrath - John Steinbeck",
    "East of Eden - John Steinbeck",
    "The Old Man and the Sea - Ernest Hemingway",
    "Of Mice and Men - John Steinbeck",
    "Lord of the Flies - William Golding",
    "A Clockwork Orange - Anthony Burgess",
    "The Picture of Dorian Gray - Oscar Wilde",
    "Dr. Jekyll and Mr. Hyde - Robert Louis Stevenson",
```

Books are stored in a list for easy access, no file management or pathing needed.

```
borrowed_books = {
    "The Hobbit - J.R.R. Tolkien": {
        "name": "Lucas Eling",
        "first_name": "Lucas",
        "last_name": "Eling",
        "email": "lucas.eling@bourne-grammar.lincs.sch.uk",
        "due": "2025-01-01",
        "emailed": False,
    }
}
```

Separate list for borrowed books, containing data about the user, formed by prompting for first and second name and then joining them together for the email

## Function 1:  menu

```
def display_menu():
    print("\nLibrary Menu")
    print("1. View books")
    print("2. Borrow a book")
    print("3. Return a book")
    print("4. Exit")
```

Very simple menu which outputs the options the user has.

## Function 2: View books

```
def view_books():
    print("\nAvailable books:")
    if available_books:
        for i, title in enumerate(available_books, 1):
            print(f" {i}. {title}")
    else:
        print(" No books available.")

    print("\nBorrowed books:")
    if borrowed_books:
        for i, (title, entry) in enumerate(borrowed_books.items(), 1):
            if isinstance(entry, dict):
                name = entry.get('name') or ((entry.get('first_name','') + ' ' + entry.get('last_name','')).strip())
                due = entry.get('due') or ''
                print(f" {i}. {title} (borrowed by {name}, due {due})")
            else:
                print(f" {i}. {title} (borrowed by {entry})")
    else:
        print(" No books are currently borrowed.")
```

If available_books returns true if the list is not empty

Enumerate function assigns an index to the books (This is useful for when they are imported from a text file later)

Borrowed books follows a similar thinking process but also bringing in the first and last names from the files that the data will be stored in later.

# Function 3: Borrow books

```
def borrow_book():
    global available_books, borrowed_books
    if not available_books:
        print("\nNo books available to borrow.")
        return
    print("\nSelect a book to borrow:")
    for i, title in enumerate(available_books, 1):
        print(f" {i}. {title}")
    choice = input("Enter the number (or press Enter to cancel): ")
    if not choice:
        print("Borrow cancelled.")
        return
    if not choice.isdigit() or not (1 <= int(choice) <= len(available_books)):
        print("Invalid selection.")
        return
    first = input("Enter borrower's first name: ") or ''
    last = input("Enter borrower's last name: ") or ''
    if not first and not last:
        print("Borrower's name cannot be empty.")
        return
    borrower = (first + ' ' + last).strip()
    idx = int(choice) - 1
    title = available_books.pop(idx)
    due = (datetime.date.today() + datetime.timedelta(days=14)).isoformat()
    email = generate_email(first, last)
    borrowed_books[title] = { 'name': borrower, 'first_name': first, 'last_name': last, 'email': email, 'due': due, 'emailed': False }
    save_data()
    print(f"'{title}' has been borrowed by {borrower} (due {due}).")
```

Brings in global variables for stored books

If the list is empty, end the loop and return to main, prompting user for input again

Displays available books choices

Prompts the user for the book they would like to select, then prompts for their first and last name, with input validation for empty inputs

Removes the selected book from the list of books and uses an imported library to generate a timestamp for current time, and when it needs to be returned by.

Then, adds to the list of borrowed books all the information on the user before saving and outputting to confirm the selection.

# Function 4: Return books

```
def return_book():
    global available_books, borrowed_books
    if not borrowed_books:
        print("\nNo borrowed books to return.")
        return
    print("\nSelect a book to return:")
    borrowed_list = list(borrowed_books.items())
    for i, (title, entry) in enumerate(borrowed_list, 1):
        borrower = entry.get('name') if isinstance(entry, dict) else entry
        print(f" {i}. {title} (borrowed by {borrower})")
    choice = input("Enter the number (or press Enter to cancel): ")
    if not choice:
        print("Return cancelled.")
        return
    if not choice.isdigit() or not (1 <= int(choice) <= len(borrowed_list)):
        print("Invalid selection.")
        return
    idx = int(choice) - 1
    title, recorded = borrowed_list[idx]
    del borrowed_books[title]
    available_books.append(title)
    save_data()
    print(f"'{title}' has been returned and is now available.")
```

Brings in global variables for lists of books

Checks if the list is empty

Prints all borrowed books with indexes

Prompts the user to select a book

Removes their choice from the borrowed pile, deleting their datastore and adding the title back to the available book list.

Then prints to confirm everything has made it through.

# Runtime loop:

```python
load_data()
running = True
while running:
    display_menu()
    choice = input("Choose an option (1-4): ")
    if not choice:
        continue
    if choice == "1":
        view_books()
    elif choice == "2":
        borrow_book()
    elif choice == "3":
        return_book()
    elif choice == "4":
        save_data()
        print("Goodbye.")
        running = False
    else:
        print("Invalid option. Please enter a number from 1 to 4.")
```

Basic while loop, ends when the user asks to exit, any other input causes the loop to repeat still.

Initial testing:

Invalid inputs;

```
 90. The Wind in the Willows - Kenneth Grahame
 91. Watership Down - Richard Adams
 92. The Hitchhiker's Guide to the Galaxy - Douglas Adams
 93. Good Omens - Terry Pratchett & Neil Gaiman
 94. American Gods - Neil Gaiman
Enter the number (or press Enter to cancel): 100
Invalid selection.
```

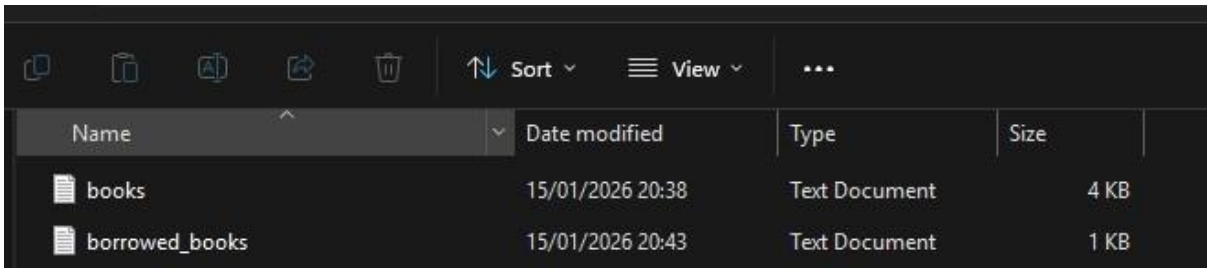
```
Library Menu
1. View books
2. Borrow a book
3. Return a book
4. Exit
Choose an option (1-4): -2
Invalid option. Please enter a number from 1 to 4.
```

```
Library Menu
1. View books
2. Borrow a book
3. Return a book
4. Exit
Choose an option (1-4): 3.5
Invalid option. Please enter a number from 1 to 4.

Library Menu
```

All input validation seems to be functioning correctly, can now choose to make a more developed program.

## Development 1; Reading and writing to a file

1) Create the files




2) Adapt program to account for file handling

```
available_books = []
borrowed_books = {}
```

Initialise lists with no data

```
# Load available books
try:
    with open("books.txt", "r", encoding="utf-8") as f:
        available_books = [line.strip() for line in f if line.strip()]
except FileNotFoundError:
    pass
```

Try to open the file books.txt, goes line by line stripping any excess data off of each line and adding it to the list of available books

```
# Load borrowed books
borrowed_books = {}
try:
    with open("borrowed_books.txt", "r", encoding="utf-8") as f:
        for line in f:
            parts = line.strip().split("|")
            if len(parts) < 2:
                continue
            title = parts[0]
            borrowed_books[title] = {
                "name": parts[1],
                "first_name": parts[2] if len(parts) > 2 else "",
                "last_name": parts[3] if len(parts) > 3 else "",
                "email": parts[4] if len(parts) > 4 else "",
                "due": parts[5] if len(parts) > 5 else "",
                "emailed": parts[6].lower() == "true" if len(parts) > 6 else False,
            }
except FileNotFoundError:
    pass
```

Attempt to load borrowed books; go line by line and find each part in sequence.

Data should be saved sequentially so opening the folder and adding to the list assumes that it was added correctly, if it doesn't find data that seems correct it leaves it blank and continues.

## New functions

Save data:

```
def save_data():
    with open("books.txt", "w", encoding="utf-8") as f:
        for book in available_books:
            f.write(book + "\n")

    with open("borrowed_books.txt", "w", encoding="utf-8") as f:
        for title, data in borrowed_books.items():
            f.write(
                "|".join([
                    title,
                    data.get("name", ""),
                    data.get("first_name", ""),
                    data.get("last_name", ""),
                    data.get("email", ""),
                    data.get("due", ""),
                    str(data.get("emailed", False))
                ]) + "\n"
            )
```

Opens the correct files and rewrites the entire file starting from the first item we have in the list.

Rewriting the file is optimal here as adding stuff onto the end may cause duplicates and does not require checking the file before writing to it.

## Email generator:

```
def generate_email(first, last):
    first = re.sub(r"[^a-zA-Z]", "", first).lower()
    last = re.sub(r"[^a-zA-Z]", "", last).lower()
    if not first and not last:
        return ""
    return f"{first}.{last}@bourne-grammar.lincs.sch.uk"
```

Takes first and last name, puts them in correct positions in the address and whacks the domain on the end.

## Next steps:

I thought the terminal on vs code was ugly and the only possible solution i could think of was building my own terminal on my web page.

So, how do you do that?

Step 1; research, is it possible – There are many sites out there such as Trinket.io which allow the user to use a python terminal in their browser

Step 2; find how to do it – the most common way people set up editors in their web pages is by using php prebuilt modules designed for this, i went with pyodide as it is simple to setup

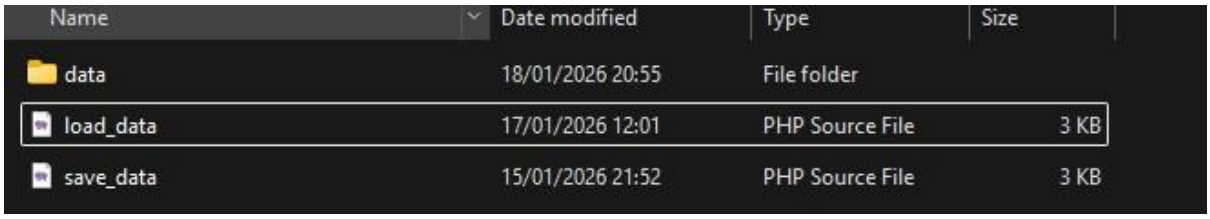
Pyodide!

Once integrated with my site, all i needed to do was paste in the code and we are done!

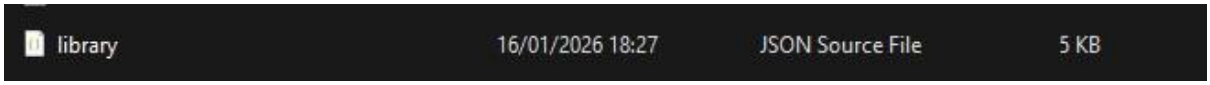
## Problems:

Now that i was hosting on my site, i could no longer simply read and write to files, i had to use php scripts to send and receive data from the servers. So;

Create php scripts:




Migrate from txt to json for more flexibility




## Load data script:

Locate file;

```
$dataDir = __DIR__ . '/data';
$storeFile = $dataDir . '/library.json';
```

If the file cannot be found, create one and add some data to it so that the program does not crash

```php
// If no store file exists, create default content
if (!file_exists($storeFile)) {
    $default = [
        'available_books' => ["1984", "The Hobbit", "Pride and Prejudice"],
        'borrowed_books' => new stdClass()
    ];
    file_put_contents($storeFile, json_encode($default, JSON_PRETTY_PRINT));
}
```

Get the data from the json, look at the headers to find ‘available_books’ which marks the begging and end of the set of data, add these to a list that can be used during runtime.

```php
$raw = file_get_contents($storeFile);
$data = json_decode($raw, true);
if (!$data) {
    echo json_encode(['success' => false, 'error' => 'Failed to parse data file']);
    exit;
}

// Normalize structure for backward compatibility
$available = isset($data['available_books']) ? array_values($data['available_books']) : [];
$borrowed = [];
$auth_required = false;
```

Let signed in users view overdue books and notify people about their tardiness

```php
// Only expose borrowed/overdue details to authenticated sessions
if (!empty($_SESSION['user'])) {
    if (isset($data['borrowed_books']) && is_array($data['borrowed_books'])) {
        foreach ($data['borrowed_books'] as $title => $entry) {
            if (is_array($entry)) {
                $borrowed[$title] = [
                    'borrower' => isset($entry['borrower']) ? $entry['borrower'] : (isset($entry[0]) ? $entry[0] : (isset($entry['name'])
                    'due' => isset($entry['due']) ? $entry['due'] : null,
                    'emailed' => isset($entry['emailed']) ? (bool)$entry['emailed'] : false
                ];
            } else {
                $borrowed[$title] = [ 'borrower' => (string)$entry, 'due' => null, 'emailed' => false ];
            }
        }
    }
} else {
```

Send all the information to the page so that it can actually be used

```php
echo json_encode([
    'success' => true,
    'available_books' => $available,
    'borrowed_books' => $borrowed,
    'auth_required' => $auth_required
]);
?>
```

# Save data script:

Find the file:

```
$dataDir = __DIR__ . '/data';
$storeFile = $dataDir . '/library.json';
```

Get the current books stored locally in the program

```
$availableBooks = isset($input['available_books']) ? $input['available_books'] : [];
$borrowedBooks = isset($input['borrowed_books']) ? $input['borrowed_books'] : [];
```

Define where the data is being sent to

```
$payload = [
    'available_books' => array_values($availableBooks),
    'borrowed_books' => new stdClass()
];
```

Define what data is being sent, ie Borrowed books should contain titles, names, dates and emails

```php
foreach ($borrowedBooks as $title => $value) {
    if (is_string($value)) {
        $payload['borrowed_books']->{$title} = [
            'borrower' => $value,
            'due' => null,
            'emailed' => false
        ];
    } elseif (is_array($value)) {
        $payload['borrowed_books']->{$title} = [
            'borrower' => isset($value['borrower']) ? $value['borrower'] : (isset($value[0]) ? $value[0] : 'Unknown'),
            'due' => isset($value['due']) ? $value['due'] : null,
            'emailed' => isset($value['emailed']) ? (bool)$value['emailed'] : false
        ];
    } elseif (is_object($value)) {
        $payload['borrowed_books']->{$title} = [
            'borrower' => isset($value->borrower) ? $value->borrower : 'Unknown',
            'due' => isset($value->due) ? $value->due : null,
            'emailed' => isset($value->emailed) ? (bool)$value->emailed : false
        ];
    }
```

Write the data to the server; error message is displayed in the terminal if it fails, success displayed if it goes through

```php
// Write JSON to disk
if (file_put_contents($storeFile, json_encode($payload, JSON_PRETTY_PRINT)) === false) {
    echo json_encode(['success' => false, 'error' => 'Failed to write data']);
    exit;
}

echo json_encode(['success' => true, 'message' => 'Data saved successfully']);
?>
```

## What other functionality can we add?

As a part of my package for purchasing a domain, I'm allowed to create an email inbox with my domain, so let's implement an email system – Sorry Arjun I stole your idea.

How do you setup an email system in html?

Step 1: Create some scripts


Configuration as listed on the domain provider, fairly simple to setup

```php
<?php


return [
    'host' => 'ox.livemail.co.uk',
    'port' => 465,               // 465 for SSL, 587 for TLS
    'username' => 'library@lucaseling.com',
    'password' => '',
    'secure' => 'ssl',           // 'ssl' for 465, 'tls' for 587

    // From address shown on emails
    'from_email' => 'library@lucaseling.com',
    'from_name' => 'Library'
];
```

Define the file that the users are stored in;

```php
try {
    $storeFile = __DIR__ . '/data/library.json';
    if (!file_exists($storeFile)) {
        throw new Exception('Data store missing');
    }
```

Create emails where possible for users without any stored emails;

```php
$email = trim($e['email'] ?? '');
if ($email === '') {
    // Prefer first/last fields if present
    $first = $first ?? trim($e['first_name'] ?? '');
    $last = $last ?? trim($e['last_name'] ?? '');
    if ($first !== '' || $last !== '') {
        $localFirst = strtolower(preg_replace('/[^a-z]/i', '', $first ?: ''));
        $localLast = strtolower(preg_replace('/[^a-z]/i', '', $last ?: '')) ?: 'unknown';
        $email = ($localFirst !== '' ? $localFirst : 'unknown') . '.' . $localLast . '@bourne-grammar.lincs.sch.uk';
    } elseif ($name !== '') {
        $parts = preg_split('/\s+/', $name);
        $localFirst = strtolower(preg_replace('/[^a-z]/i', '', $parts[0] ?? ''));
        $localLast = strtolower(preg_replace('/[^a-z]/i', '', $parts[count($parts)-1] ?? '')) ?: 'unknown';
        $email = ($localFirst !== '' ? $localFirst : 'unknown') . '.' . $localLast . '@bourne-grammar.lincs.sch.uk';
    }
}
```

Create the main body for the email;

```php
$subject = "Overdue library book: $title";
$greeting = $name !== '' ? "Hello $name,\n\n" : "Hello,\n\n";
$body = $greeting . "'$title' was due on {$due->format('Y-m-d')}. \nPlease return it to the library as soon as possible.\n\nThank you.";
```

Use the config from the other file to send the email;

```php
        $sent = false;
        if (!empty($config['host'])) {
            $sent = smtp_send(
                $config['host'],
                $config['port'] ?? 465,
                $config['secure'] ?? '',
                $config['username'] ?? '',
                $config['password'] ?? '',
                $config['from_email'] ?? 'library@lucaseling.com',
                $email, $subject, $body
            );
        } else {
            $sent = mail($email, $subject, $body);
        }

        if ($sent) {
            $results['sent'][] = ['title'=>$title,'to'=>$email];
        } else {
            $results['failed'][] = ['title'=>$title,'to'=>$email];
        }
    }
```

Relay back to the terminal what has happened;

```php
if (!$GLOBALS['__json_sent']) {
    $GLOBALS['__json_sent'] = true;
    ob_clean();
    echo $payload;
}
exit;
```

## Next:

Need to implement functionality to the actual code to allow the user to email people.

How do i do this?
Very simple setup, once the user is logged in, they may view overdue books by inputting 5;

```
Library Menu
1. View books
2. Borrow a book
3. Return a book
4. Exit
5. View overdue books
Choose an option (1-5): 5

No overdue books.
```

If any overdue books are present, prompt the user to email people.

```
Library Menu
1. View books
2. Borrow a book
3. Return a book
4. Exit
5. View overdue books
Choose an option (1-5): 5

Overdue books:
 1. To Kill a Mockingbird - Harper Lee (borrowed by Lucas
Eling, due 2025-02-01)

Send email notices to overdue borrowers? (y/N):
```

Then we simply adapt the php file to listen for any yes messages sent through, and that calls the email protocol to be run.

## Issues with this setup:

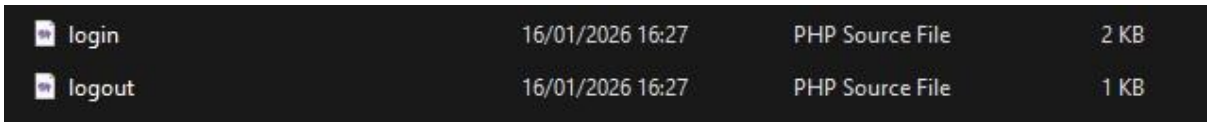
I quickly realised that my website is public, and hence I cannot really add an option to email users, people could simply borrow books and then spam email people 2 weeks later.

How do I fix this?

Create a login page, this would perfectly avoid this situation, anyone can borrow a book, anyone can return a book, but only authorised members can notify people with overdue books.

How do we do this?

Step 1: Create script files




Step 2: Create user data – Valid users that can log in


Login protocol;

```
y-system > api > login.php
<?php
session_start();
header('Content-Type: application/json');

$username = isset($_POST['username']) ? trim($_POST['username']) : '';
$password = isset($_POST['password']) ? $_POST['password'] : '';

$usersFile = __DIR__ . '/data/users.json';
if (!file_exists($usersFile)) {
    echo json_encode(['success' => false, 'message' => 'No users configured']);
    exit;
}
```

Load the json with user details

Create valid possible usernames and passwords that can be inputted

```
foreach ($users as $u) {
    if (!isset($u['username'])) continue;
    if ($u['username'] !== $username) continue;
    if (!isset($u['password'])) continue;

    $stored = $u['password'];
```

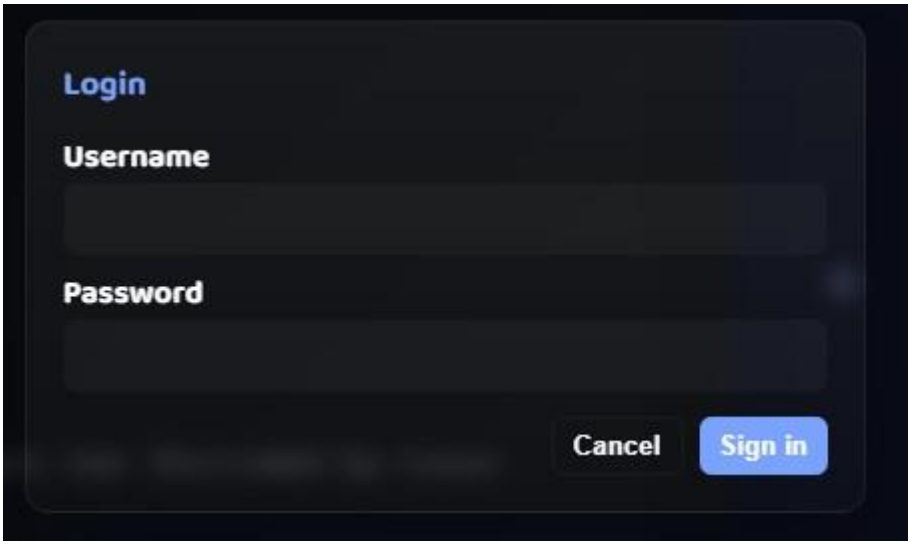
Check the login box and capture user inputs

```php
$ok = false;
if (is_string($stored) && strlen($stored) > 0 && ($stored[0] === '$')) {
    if (password_verify($password, $stored)) $ok = true;
} else {
    if ($stored === $password) $ok = true;
}
```

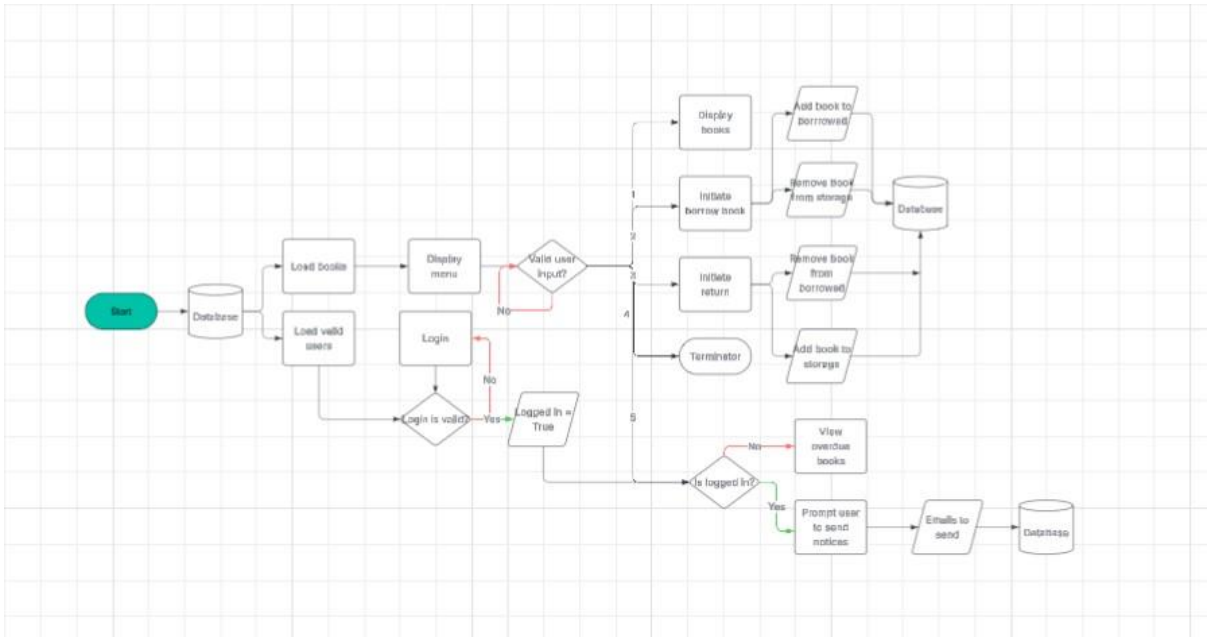
If the users input is the same as a pair in the file, then its all good.

```php
    if ($ok) {
        $_SESSION['user'] = $username;
        echo json_encode(['success' => true, 'username' => $username]);
        exit;
    }
}

echo json_encode(['success' => false, 'message' => 'Invalid credentials']);
```

If successful then display a success message to the user, and log them in. Simple login box on the web;




Flowchart of program;




Testing;

Testing done by me;

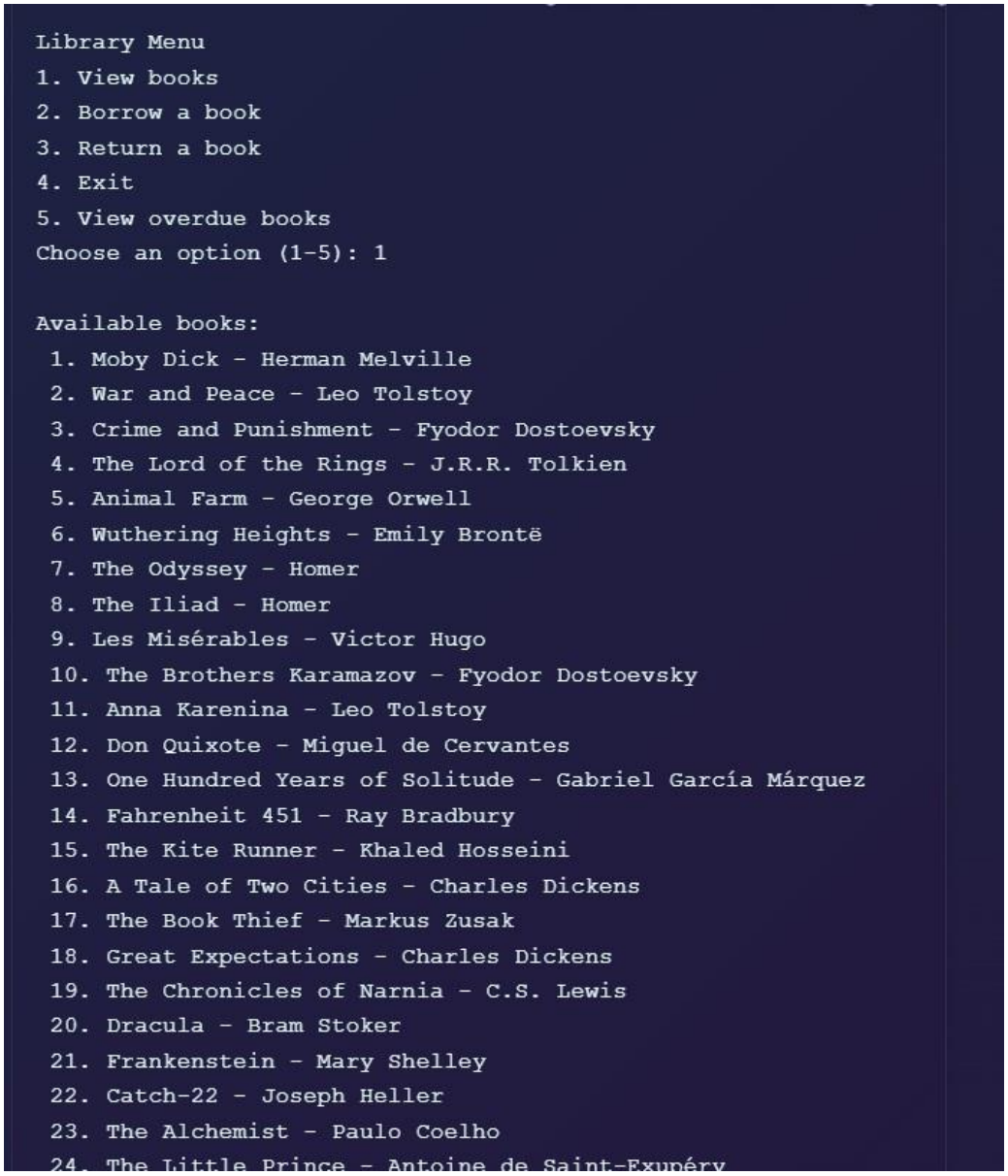
Menu testing; OOB too high, OOB too low, unexpected float.

```
Library Menu
1. View books
2. Borrow a book
3. Return a book
4. Exit
5. View overdue books
Choose an option (1-5): 6
Invalid option. Please enter a number from 1 to 5.
```

```
Library Menu
1. View books
2. Borrow a book
3. Return a book
4. Exit
5. View overdue books
Choose an option (1-5): -1
Invalid option. Please enter a number from 1 to 5.
```

```
Library Menu
1. View books
2. Borrow a book
3. Return a book
4. Exit
5. View overdue books
Choose an option (1-5): 1.4
Invalid option. Please enter a number from 1 to 5.
```

View books option;




Functions as intended.

## Borrow books menu

OOB too high, OOB too low, unexpected float, words.

```
Library Menu
1. View books
2. Borrow a book
3. Return a book
4. Exit
5. View overdue books
Choose an option (1-5): 2

Select a book to borrow:
 (Note: For the full catalogue, please use the search panel on
the right.)
Enter the number (or press Enter to cancel): 45354
Invalid selection.
```

```
Library Menu
1. View books
2. Borrow a book
3. Return a book
4. Exit
5. View overdue books
Choose an option (1-5): 2

Select a book to borrow:
 (Note: For the full catalogue, please use the search panel on
the right.)
Enter the number (or press Enter to cancel): -19
Invalid selection.
```

```
Library Menu
1. View books
2. Borrow a book
3. Return a book
4. Exit
5. View overdue books
Choose an option (1-5): 2

Select a book to borrow:
 (Note: For the full catalogue, please use the search panel on
the right.)
Enter the number (or press Enter to cancel): 1.3
Invalid selection.
```

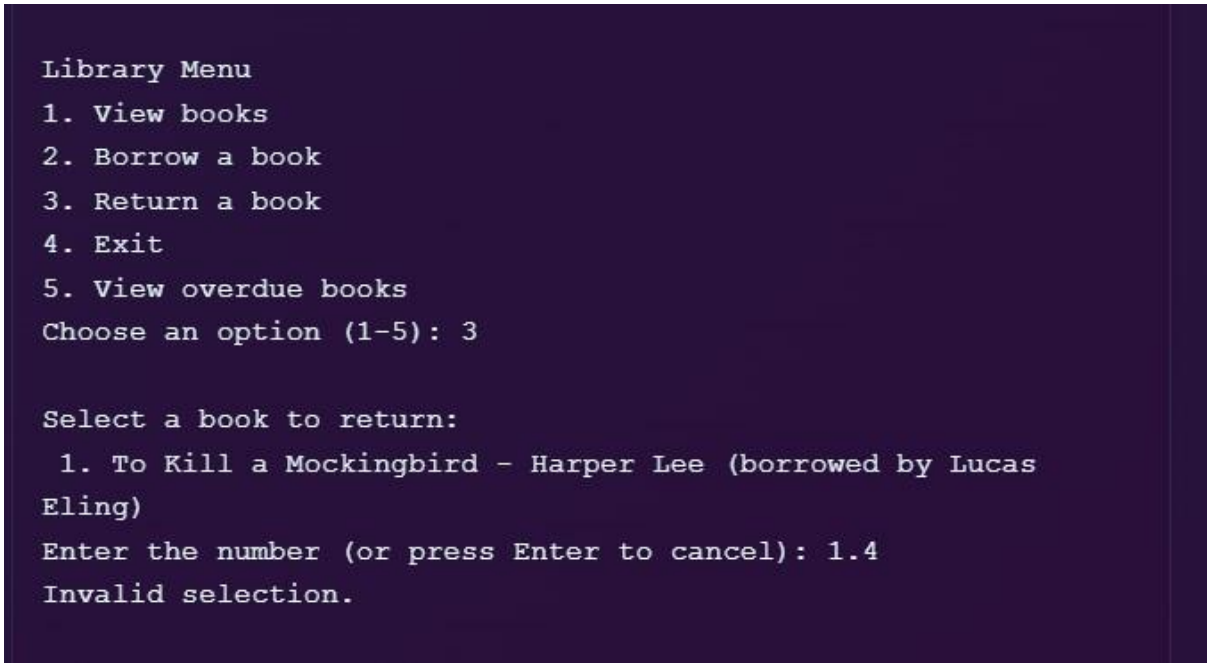
```
Library Menu
1. View books
2. Borrow a book
3. Return a book
4. Exit
5. View overdue books
Choose an option (1-5): 2

Select a book to borrow:
 (Note: For the full catalogue, please use the search panel on
the right.)
Enter the number (or press Enter to cancel): hi
Invalid selection.
```

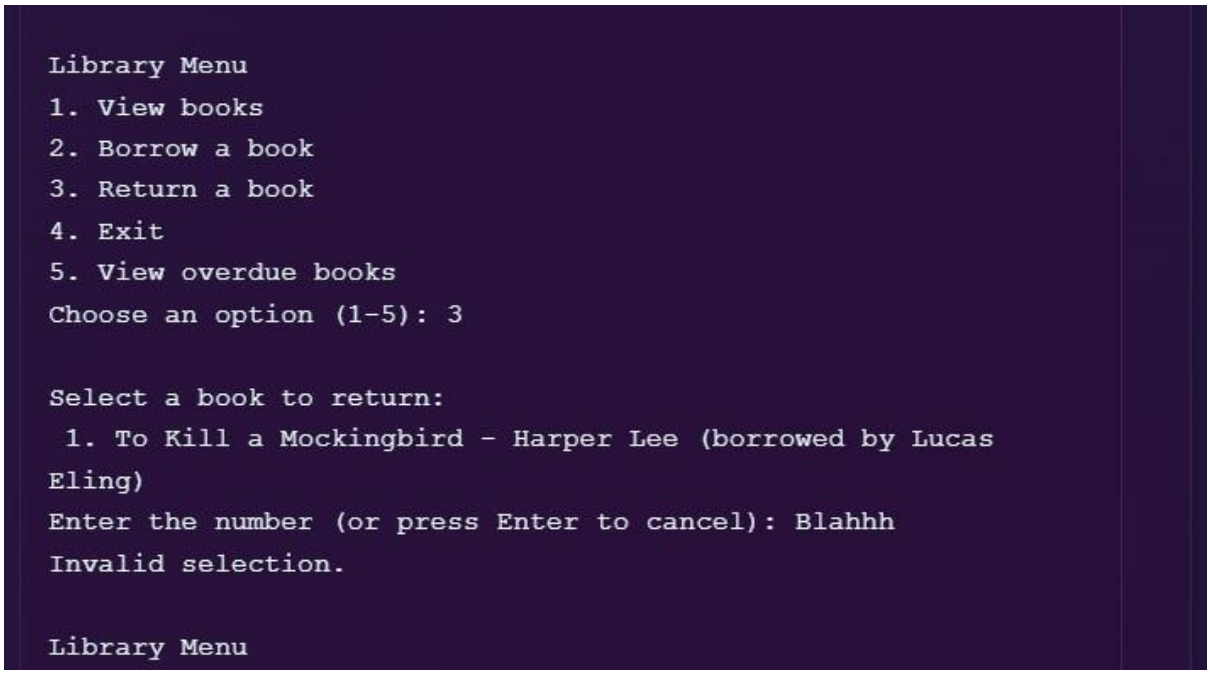
## Option 3, return a book

Data tested – OOB Too high, OOB too low, float, words.

```
Library Menu
1. View books
2. Borrow a book
3. Return a book
4. Exit
5. View overdue books
Choose an option (1-5): 3

Select a book to return:
 1. To Kill a Mockingbird - Harper Lee (borrowed by Lucas
Eling)
Enter the number (or press Enter to cancel): 4
Invalid selection.

Library Menu
```

```
Library Menu
1. View books
2. Borrow a book
3. Return a book
4. Exit
5. View overdue books
Choose an option (1-5): 3

Select a book to return:
 1. To Kill a Mockingbird - Harper Lee (borrowed by Lucas
Eling)
Enter the number (or press Enter to cancel): -2
Invalid selection.
```



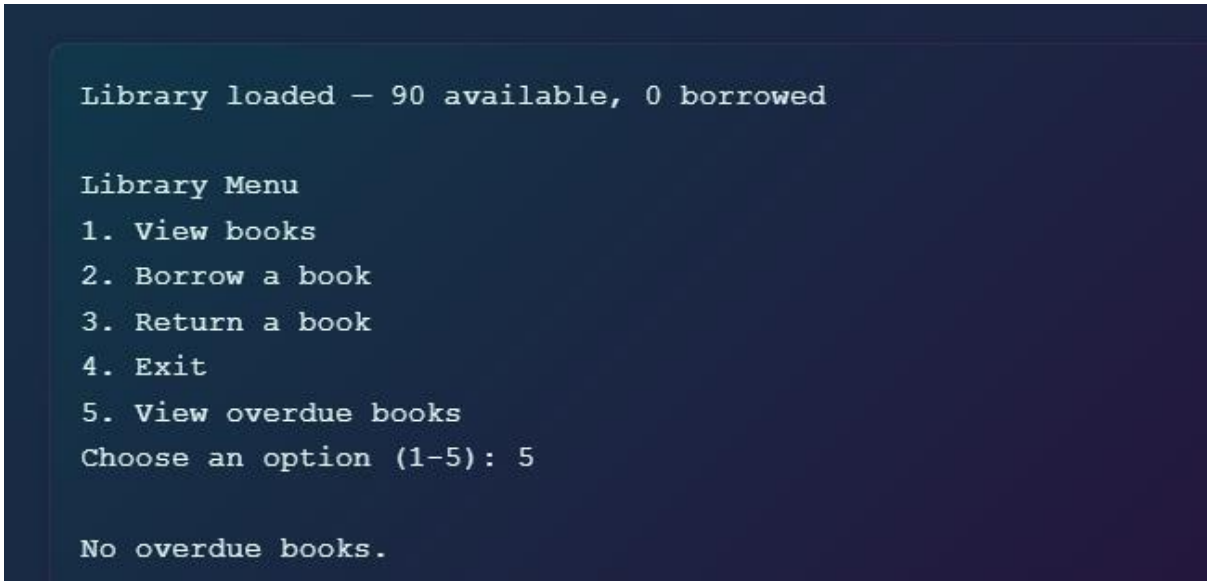




## Option 4; exit

No testing required straight forward.

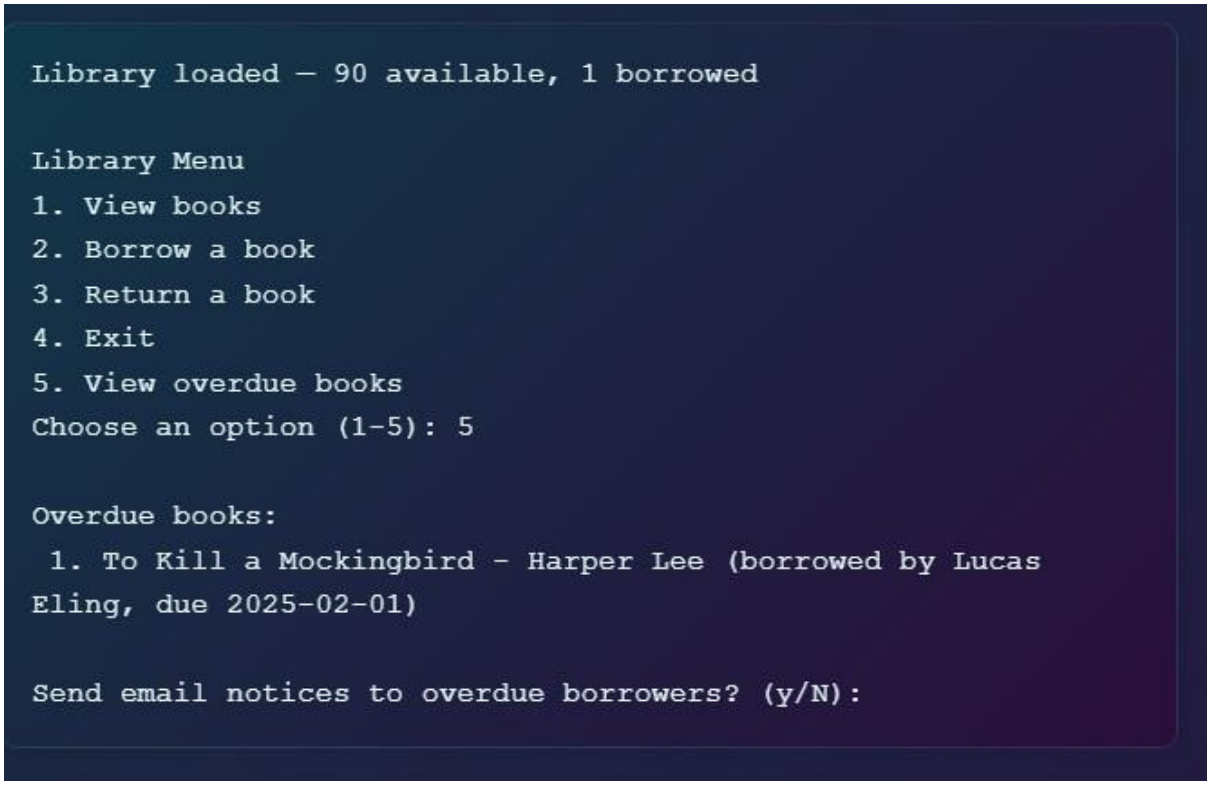
## Option 5; View overdue books

Logged out --




No overdue books shown – should still display overdue books, potentially fix before due date.

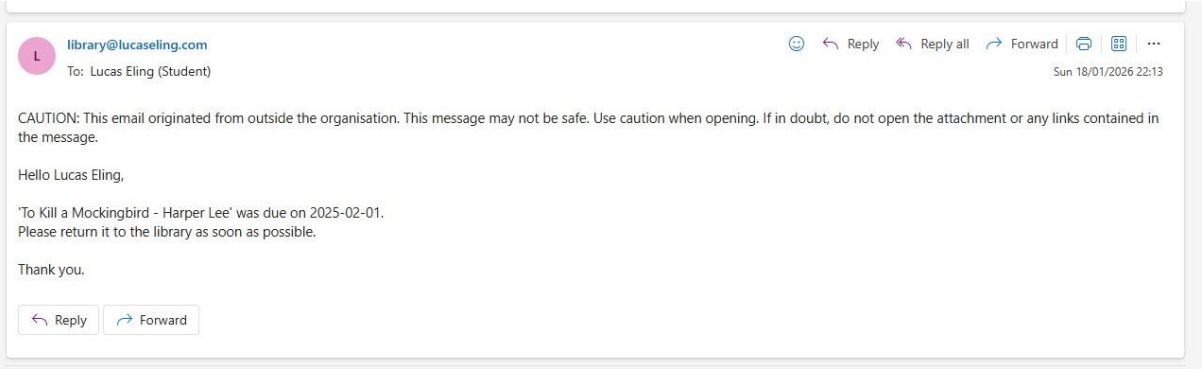
Logged in --




Books displayed properly

```
Send email notices to overdue borrowers? (y/N): y
Sending overdue email notices...
Sent: 1, Failed: 0
```

Email system functional




Next steps;

After completing the python script, I realised only coding using python would severely limit my programs potential, so I made the decision to make a “full ui” mode which gives more functionality and provides easier use.
I also realised that it would be more reliable for me to use account creation and email verification instead of asking for a users first and second name, and joining them to make an email, users may have been external and not used our domain, or lied about their name when prompted, a verification system would eliminate this and allow the user to select their preferred email.
I should also be able to do more stuff with administrator accounts, their current functionality is limited, potentially add a logs system to track each user and it would be nice to be able to write to the json with new books via a ui as editing the json takes too many clicks.

TODO list:
Allow users to create accounts
Store emails and send notifications for verification
Require the user to be logged in to use the library system
Additional functionality for administrators (Manually hand in books for users, logging system, view users and their borrowed books, add/remove copies of books via the web instead of editing the json)

Signup system;
I already had the structure for this as I had a login system, I need to adapt it to write new users, hashing/encrypting passwords for security, taking emails and storing them

[images]

After completing the login system, I realised that there is no verification for the email so people can pretend to be others and pester them, I need to now create a verification system to ensure that people are who they say they are.

[images of email verification]

Small oversight, accounts are created before email verification is completed so I need to write pending users to a separate file with their verification code, once the code is inputted, they can be transferred to the new list.

[images of new structure]

Now that accounts have emails linked to them, I can rework the mailing system and the borrowing system, no longer requiring users to input first and second name, the book can belong to their account instead.

[images of new json format]

Now that the signup system is completed, I can begin to implement newer features for administrators.

I started with a logging system so that I can view more things that I've done — creating a logging system layer on would create less logs and I wouldn't be able to properly test it.

How do I make a logging system? With every book taken/account created I should write to a logs file with the process that happens and the user it was done by, could potentially add the precise time however it would be sorted by time anyways.

Designing the logging page;
I should be able to search by user, by date, and by action.
[image of logs page]
Now that I have made the page layout I can begin to alter the scripts to write stuff and then load the logs file on the page.
[images of some new scripts]
[image of logs in the page]

Currently can filter by borrowing, returning, account creation, books added, books removed.

Now I can focus on returning books manually for users who have forgotten their login, I should probably also notify them that the book has been removed from their account, first I'll create a script to notify a user that someone has removed a book from them.

[image of script]

Now I need to add the functionality to the page, I should probably make it a general user management page incase I wish to add functionality later,
Design – pull every user from the users json and display them if they have any books borrowed.
Interacting with a user will allow me to view the books they have borrowed and return each one with a simple "return" prompt and a confirmation.
[image of design]
[image of email going through]

Now that I have this user management system, I should display information I have such as their email.
[image of now displaying email]

I should probably implement a "forgot password" menu option.