E-Library

An Online Library Management System

By

STUDENT1306302	AMAAN AL MIR
STUDENT1119343	AZAH ARIF ALI TAHA
STUDENT1400519	ALA HAMID ABUBAKAR MOHAMED ALHASAN

Table of Contents

Certificate of Completion

This is to certify that the E-Library Team has satisfactorily completed the e-Project organized by Aptech. The program, conducted from November 8, 2023, to November 27, 2023, encompassed Web Application Development. Throughout the duration of the program, the E-Library Team demonstrated dedication, proficiency, and commitment to learning, successfully acquiring the essential knowledge and skills. This certificate attests to their diligence and successful fulfillment of all requirements as stipulated in the e-Project.

Issued this November 26, 2023.

Mr. Ashish Jha

Aptech Qatar

Acknowledgement

We would like to express our heartfelt gratitude to the individuals and organizations who played a crucial role in the successful completion of this project. Each team member's dedication and collaborative spirit have contributed to the project's overall success.

Our gratitude goes to **Mr. Ashish Jha** for his guidance, support, and mentorship. His expertise and encouragement have been instrumental in steering the team in the right direction.

We are also thankful to **Aptech Qatar** for providing the necessary resources, facilities, and a conducive environment for teamwork and project development.

We express our thanks to the authors and researchers whose work served as a foundation for our understanding of Web Application Development

Finally, we extend our thanks to anyone else who has supported the team in various ways. Your collective efforts have made a significant impact on the project's success.

Thank you all for your dedication, hard work, and collaborative spirit. This project would not have been possible without each and every one of you.

- E-Library Team

Project Synopsis

The E-Library project aims to streamline library management through an Online Library Management System. The system automates maintenance of book information, author details, member records, and library staff. This computerized solution reduces manual workload for librarians, ensuring efficient maintenance and preventing data loss. The system encompasses modules for Guest Users to view available titles, Register Users to access features like online reading and downloads, and an Admin module for comprehensive management. Admin capabilities include overseeing book details, user registration, categories, and generating reports. With simplified access and robust features, E-Library enhances library management, providing a user-friendly experience for both administrators and library members.

Project Analysis

The online library management system is designed to streamline and automate various operations related to book management, user registration, and administrative tasks. The manual complexities of managing a library are addressed through computerization, reducing the workload and enhancing overall efficiency.

1. Objectives

- Automate book management, user registration, and library operations.
- Reduce manual workload for librarians.
- Ensure accuracy and security of book and user records.
- Provide convenient access to information for both guests and registered users.
- Enable online reading and downloading of books (if permitted by admin).
- Facilitate efficient management through an admin dashboard.

2. System Modules

Guest Users

- View available book titles.
- Make inquiries through the contact us page.
- Access general information about the library, including timings and book categories.

Registered Users

- One-time registration for E-Library access.
- Login to access various features.
- Search for books based on different criteria.
- Read books online.
- Download books (if permitted by admin).
- Access video tutorials under different categories.
- Change their passwords for security.

Administrator

- Superuser with comprehensive control over the system.
- Dashboard for an overview of books, registered users, and categories.
- Manage categories: Create, update and delete.
- Manage books and videos: Add, edit, and set download permissions.
- Manage users: update and delete

3. Key Features

- **Efficiency:** Automation reduces manual work for librarians, making library management more efficient.
- **Accuracy:** Computerization ensures accurate maintenance of book and user records.
- **Convenience:** Users can access and manage library resources online, improving convenience.
- **Security:** User data and book records are secure and less prone to loss.
- **User Engagement:** Features like online reading, downloads, and video tutorials enhance user engagement.

4. Challenges and Considerations

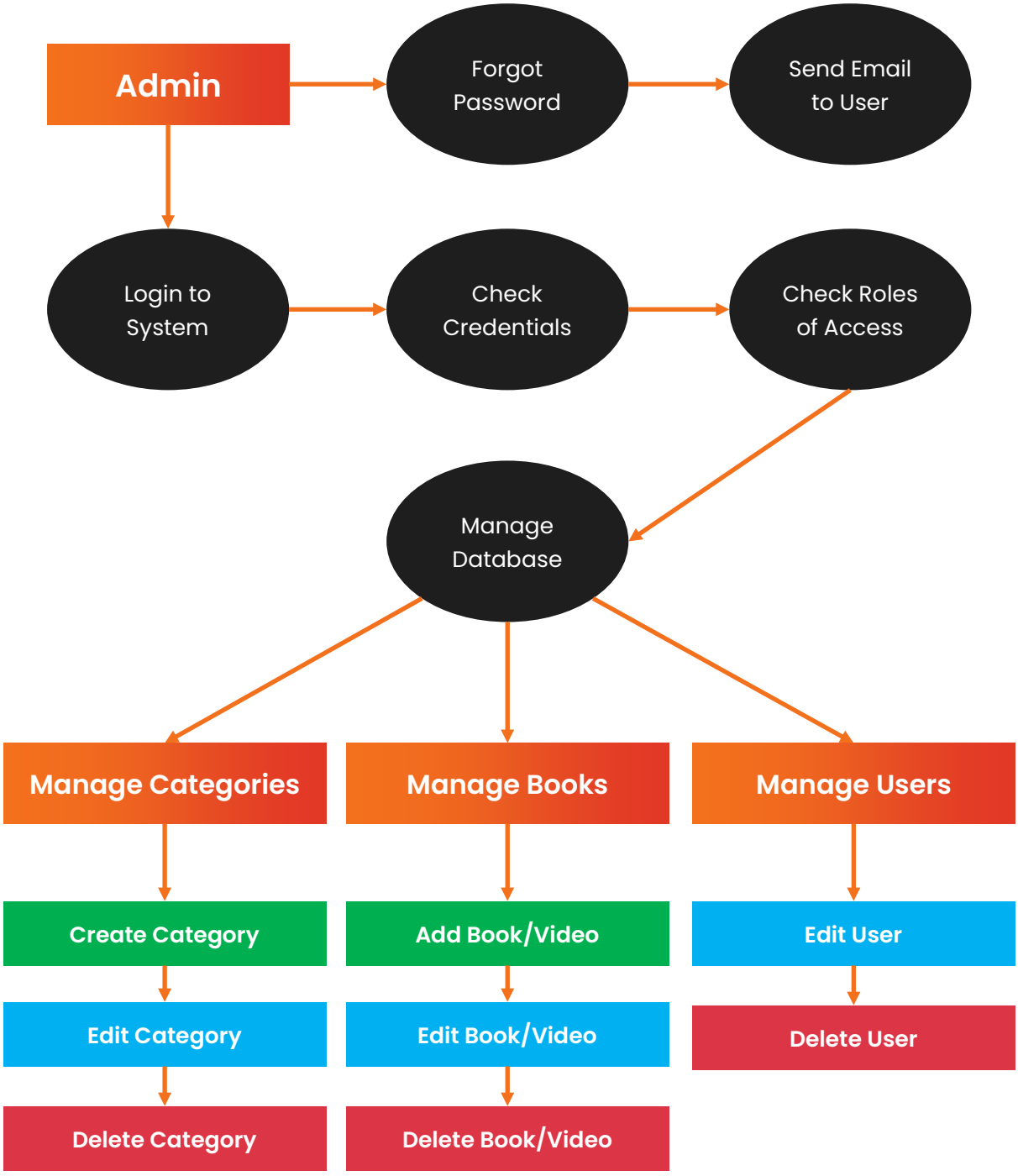
- Ensure data security and privacy.
- Implement effective search functionality for users.
- Regularly update and maintain the system for optimal performance.
- Provide adequate documentation and user support.
- Test the system thoroughly to identify and address any bugs or issues.

5. Conclusion

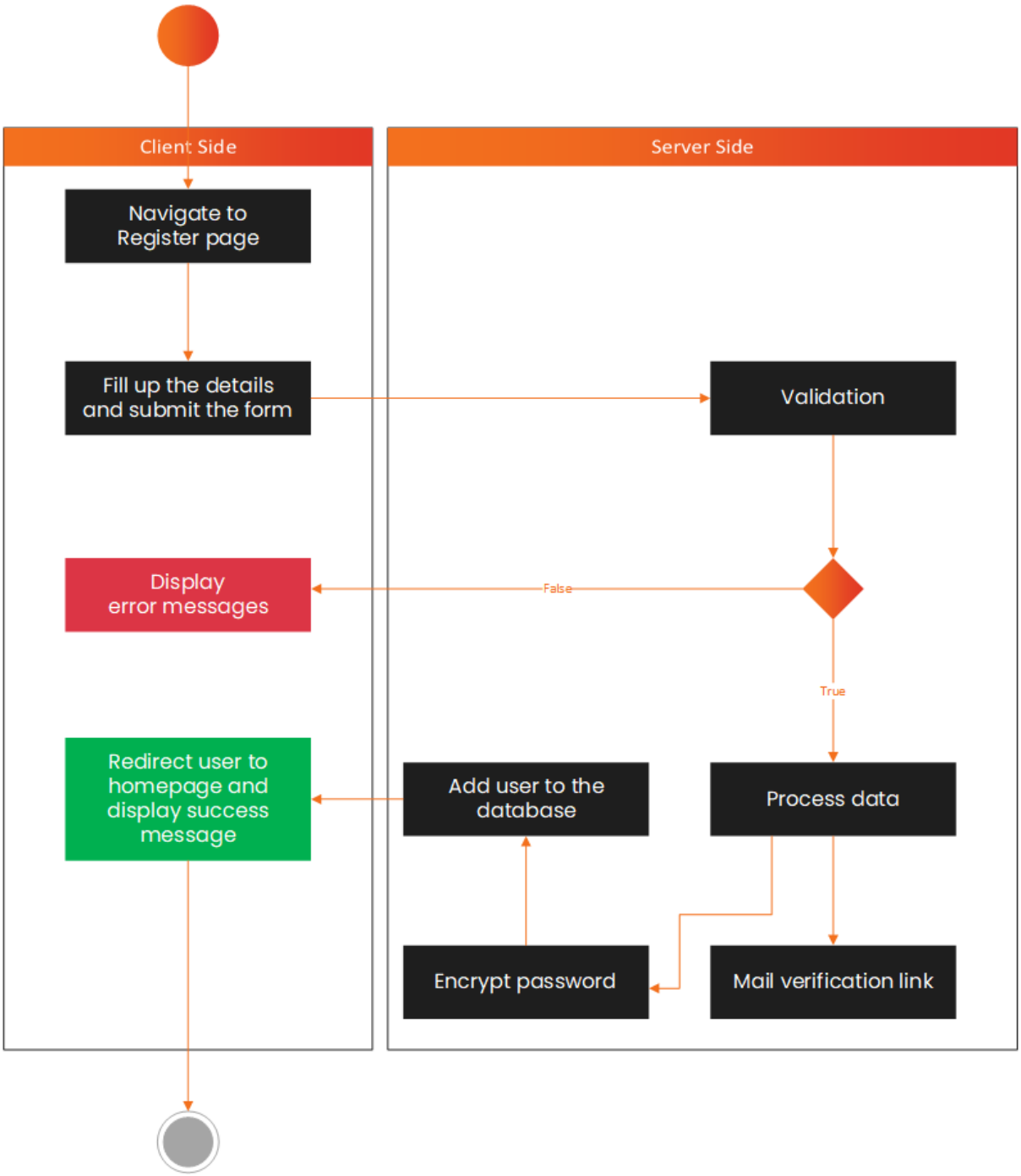
E-Library aims to modernize and simplify library operations, benefiting both librarians and users. By incorporating user-friendly interfaces and robust administrative features, the system is poised to enhance the overall library experience in the digital age. Regular updates and user feedback will be essential to ensuring the ongoing success and relevance of the system.

Application Design

Data Flow Diagram – Admin User

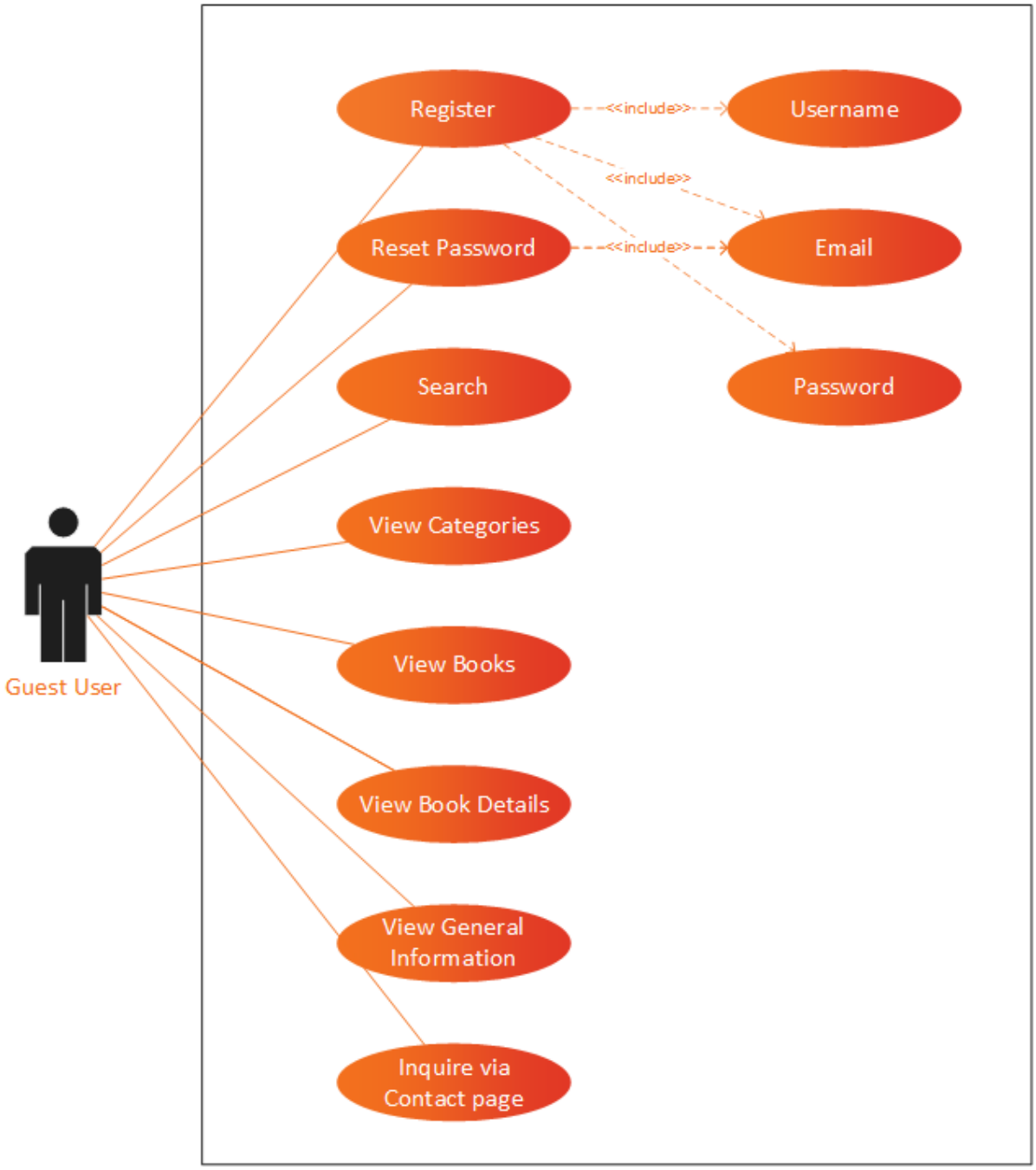


Activity Diagram – Account Registration

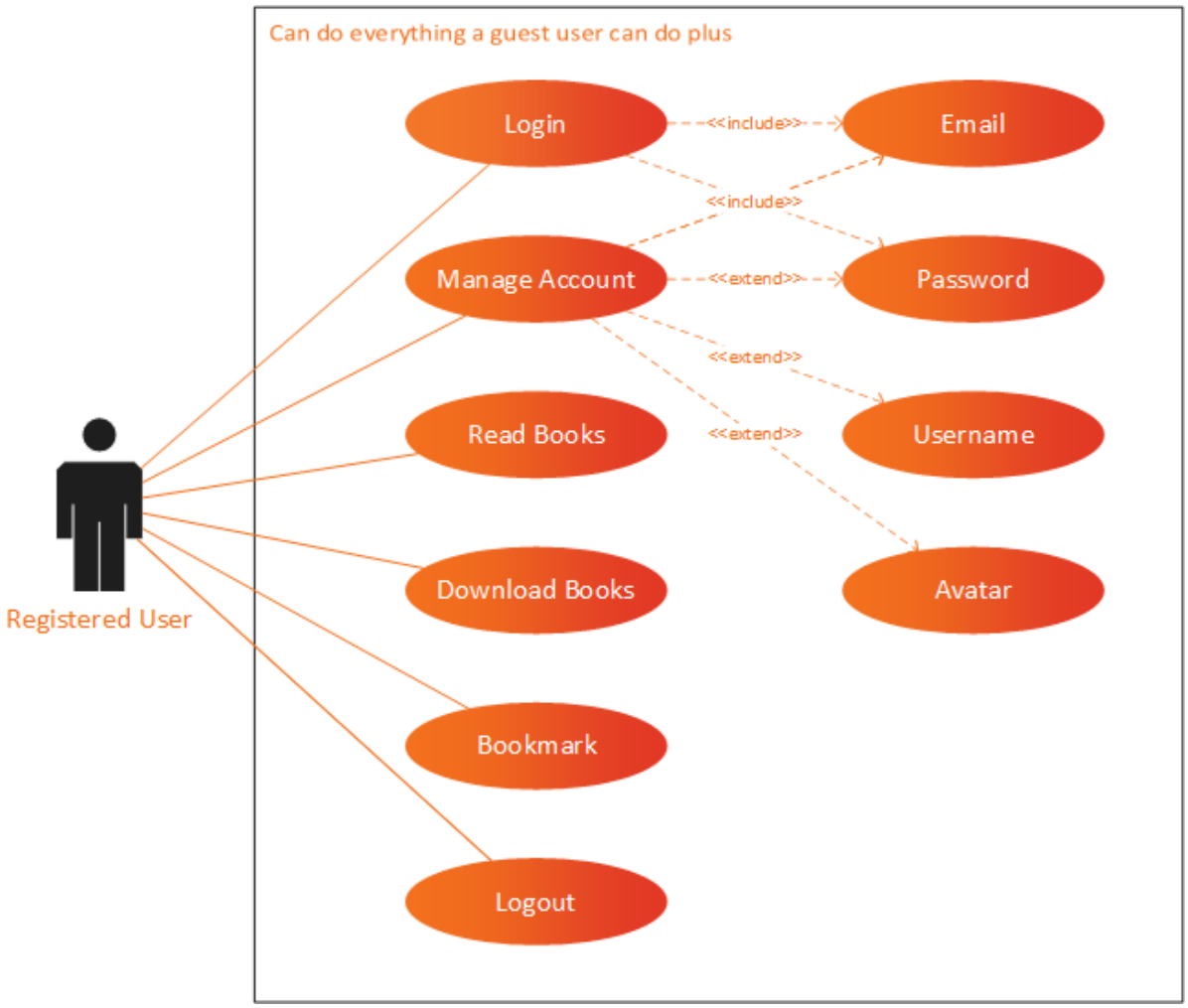


Use-Case Diagrams

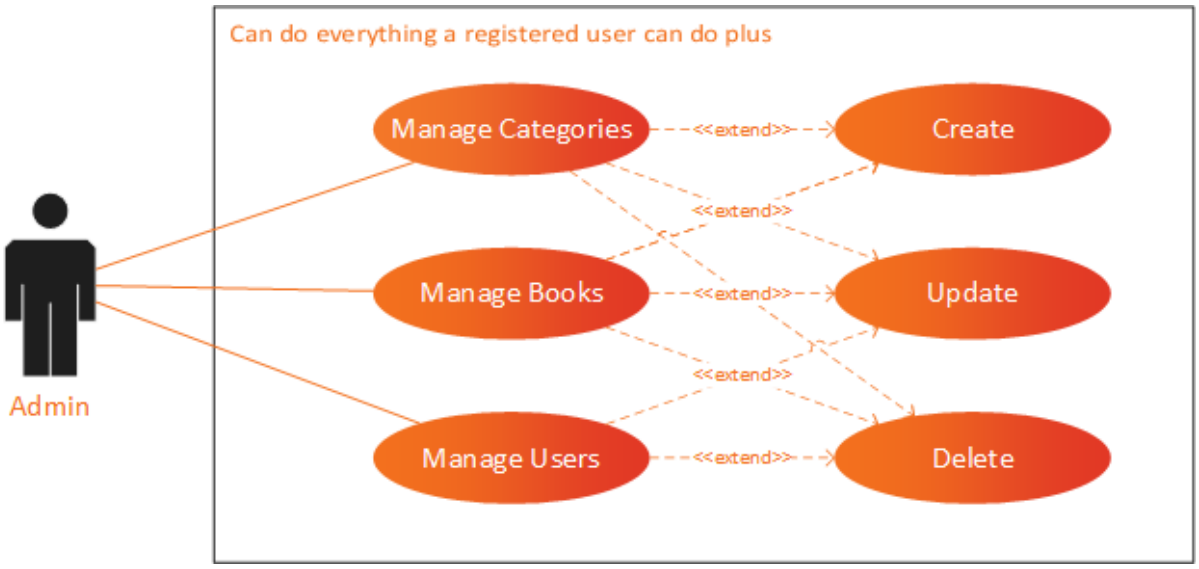
1. Unregistered Users



2. Registered Users



3. Admin


Can do everything a registered user can do plus
Manage Categories
Manage Books
Manage Users
Create
Update
Delete
<<extend>>
<<extend>>
<<extend>>
<<extend>>
Admin

Database Design & Structure

Collection	Field	Field Type
Users	_id	ObjectId
	username	String
	email	String
	password	String
	created_on	Date
	verified	Boolean
	avatar	String
	token	String
	updated_on	Date
	roles	Array
	bookmarks	Array
Categories	_id	ObjectId
	category	String
	icon	String
	image	String

Collection	Field	Field Type
Books	_id	ObjectId
	type	String
	category_id	ObjectId
	title	String
	year	Int32
	author	Array
	publisher	String
	summary	String
	book	String
	downloadable	Boolean
	cover	String
	video	String
	view_count	Int32
	download_count	Int32

The MongoDB database for the E-Library web application is well-organized, consisting of three main collections: Users, Categories, and Books. Each collection captures essential information related to users, book categories, and individual books. Below is a summary of the key fields and their types in each collection:

1. Users Collection

Fields:

- _id: Object ID for user identification. Auto generated.
- username: String representing the username.
- email: String for the user's email.
- password: String storing the user's password.
- created_on: Date indicating the user's registration date.
- verified: Boolean indicating whether the user is verified.
- avatar: String for storing the user's avatar.
- token: String for user authentication.
- updated_on: Date indicating the last update date.
- roles: Array storing user roles.
- bookmarks: Array storing bookmarked items.

Sample Data:


```
_id: ObjectId('65512270765d2c3b115fa64e')
username: "Admin"
email: "admin@elibrary.com"
password: "47a1861e8cf81c814ac62148bc80b14b"
created_on: 2023-11-12T19:07:28.981+00:00
verified: true
avatar: "/assets/img/user.jpg"
token: "hLp5ajPC7GXqSs_-dmPRhO5HQgSz6p52VrA8OGr7XL8"
updated_on: 2023-11-14T11:42:41.207+00:00
▾ roles: Array (2)
    0: "staff"
    1: "admin"
▾ bookmarks: Array (5)
    0: "65590181ce7ec25358e85ea9"
    1: "6558e9a4ce7ec25358e85ea2"
    2: "6558f3e8ce7ec25358e85ea7"
    3: "6558f0d1ce7ec25358e85ea5"
    4: "6558217c3ca85721dcdd1837"
```


2. Categories Collection

Fields:

- _id: Object ID for category identification. Auto generated.
- category: String representing the category name.
- icon: String for storing the category icon.
- image: String for storing the category image.

Sample data:


```
_id: ObjectId('65566c8cc894f21a67023700')
category: "Biography"
icon: "user-tie-hair"
image: "https://i.ibb.co/BzF9Cpg/Biography.png"
```

3. Books Collection

Fields:

- _id: Object ID for book identification. Auto generated.
- type: String indicating whether the entry is a book or video.
- category_id: Object ID linking to the Categories collection.
- title: String representing the title of the book or video.
- year: Int32 indicating the publication year.
- author: Array storing author information.
- publisher: String representing the book publisher.
- summary: String providing a brief summary of the book or video.
- book: String for storing the book file (if applicable).
- downloadable: Boolean indicating if the book is downloadable.
- cover: String storing the URL for the book cover image.
- video: String storing the URL for the video.
- view_count: Int32 indicating the number of views.
- download_count: Int32 indicating the number of downloads.

Sample data:


```
_id: ObjectId('6557bf8f0d3b10045f048e9b')
type: "book"
category_id: ObjectId('65566d4dc894f21a67023704')
title: "The Arabian Nights"
year: 2008
▾ author: Array (1)
    0: "Andrew Lang"
publisher: "Project Gutenberg"
summary: "A medieval Middle-Eastern literary epic which tells the story of Scheh…"
book: "https://cdn.filestackcontent.com/VuxSSDV0SiGLjFWYCsRD"
downloadable: true
cover: "https://i.ibb.co/X5tprZK/The-Arabian-Nights.jpg"
video: ""
view_count: 146
download_count: 44
```

Key Observations:

- Relationships are established using Object IDs, linking different collections.
- Arrays are utilized for fields like authors, roles, and bookmarks to handle multiple values.
- Date types are used for tracking registration and update timestamps.
- Boolean types such as 'verified' and 'downloadable' provide binary information.
- Strings are utilized for textual data like usernames, emails, and text summaries.

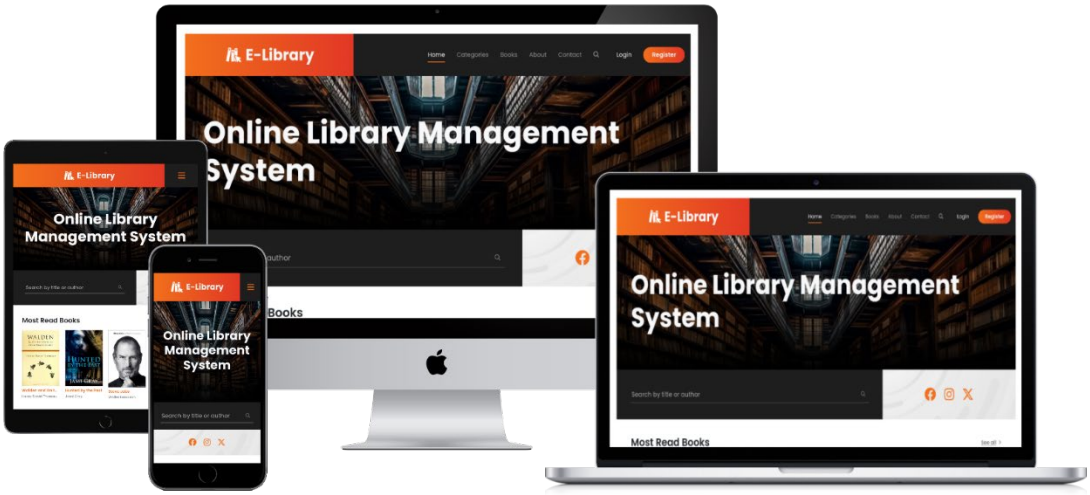
This database structure is designed to efficiently manage user data, book categories, and individual books, providing a solid foundation for the E-Library web application. The use of MongoDB allows for flexibility and scalability as the application evolves.

(Anon., n.d.)

User Guide

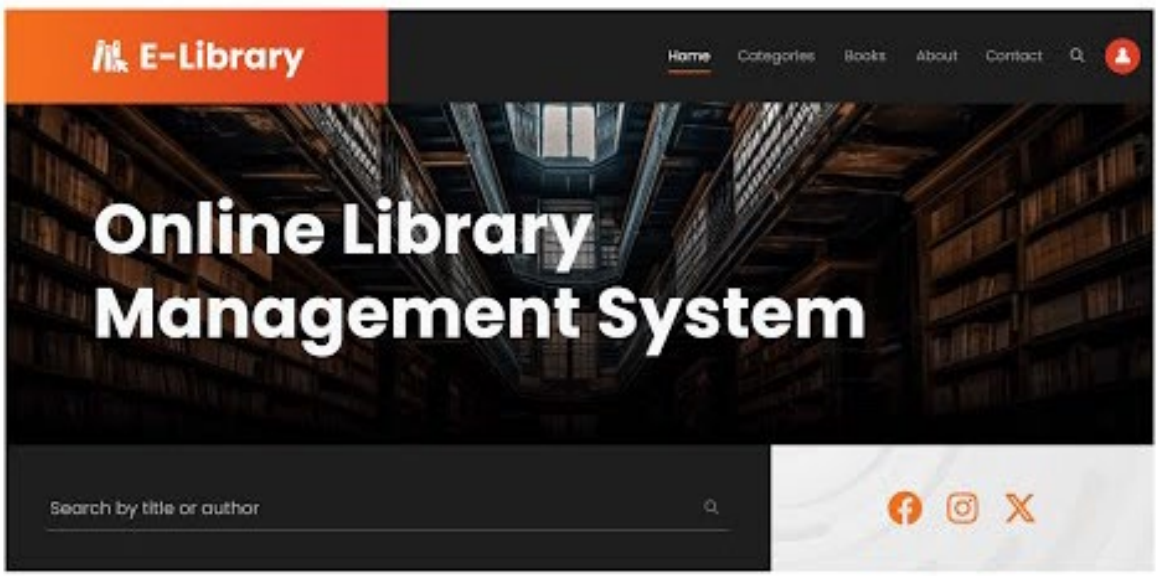
You can view the website online by going to:

https://elibrary.almir.info/



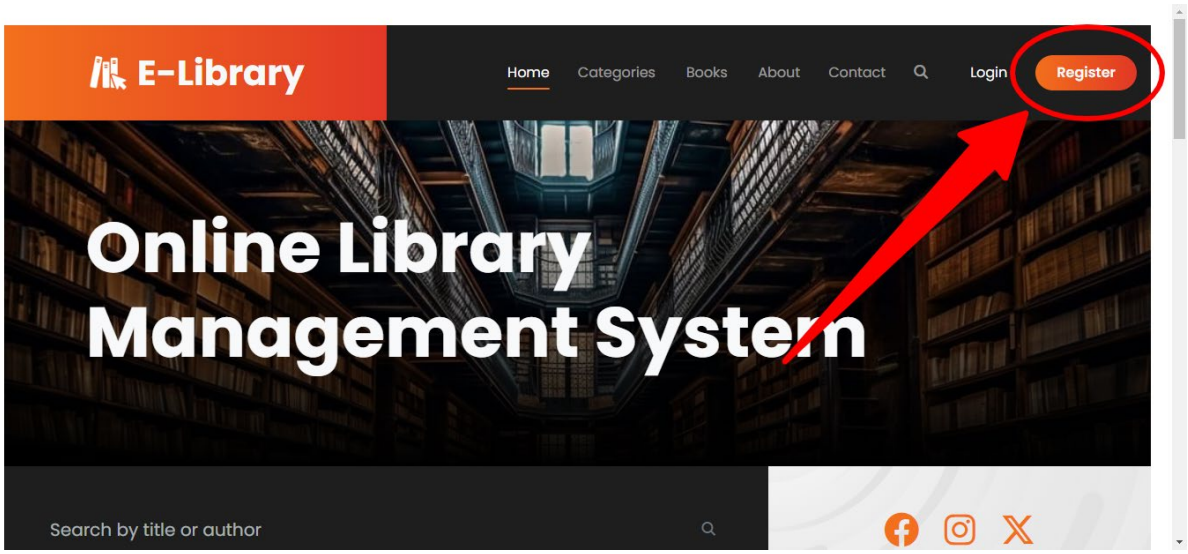
Video Demonstration:

Link: https://www.youtube.com/watch?v=bo6WEnQ8j3g



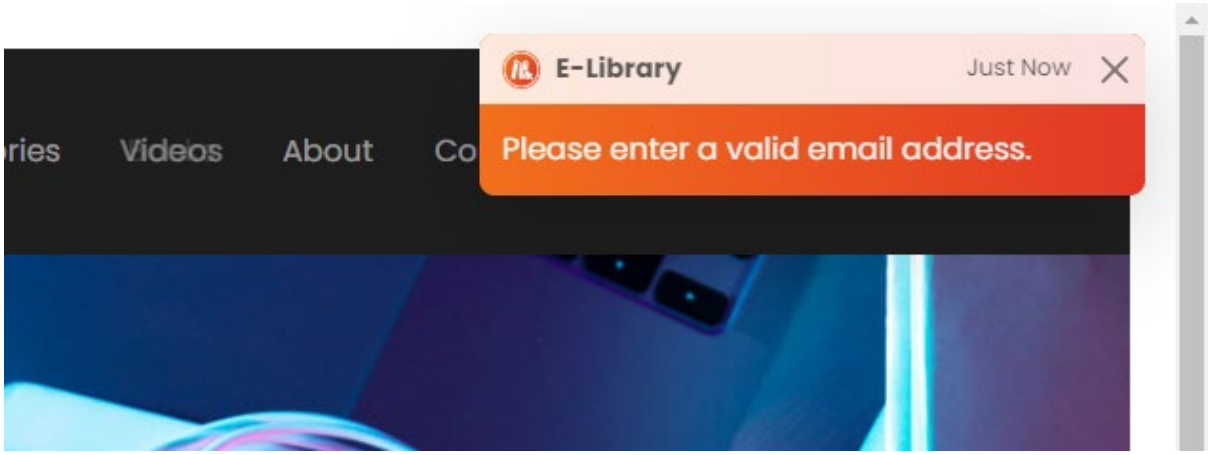
1. Register

If this is first time visiting the website and you want to create an account, go to the register page by clicking on the Register button

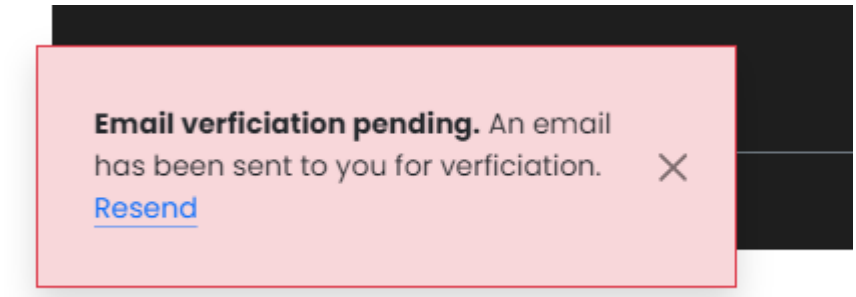


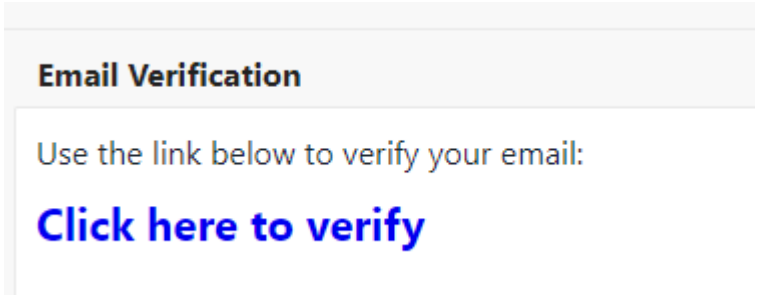
Fill in the details and click on the register to create a new account.

You may receive notifications if an error occurred while creating a new account.

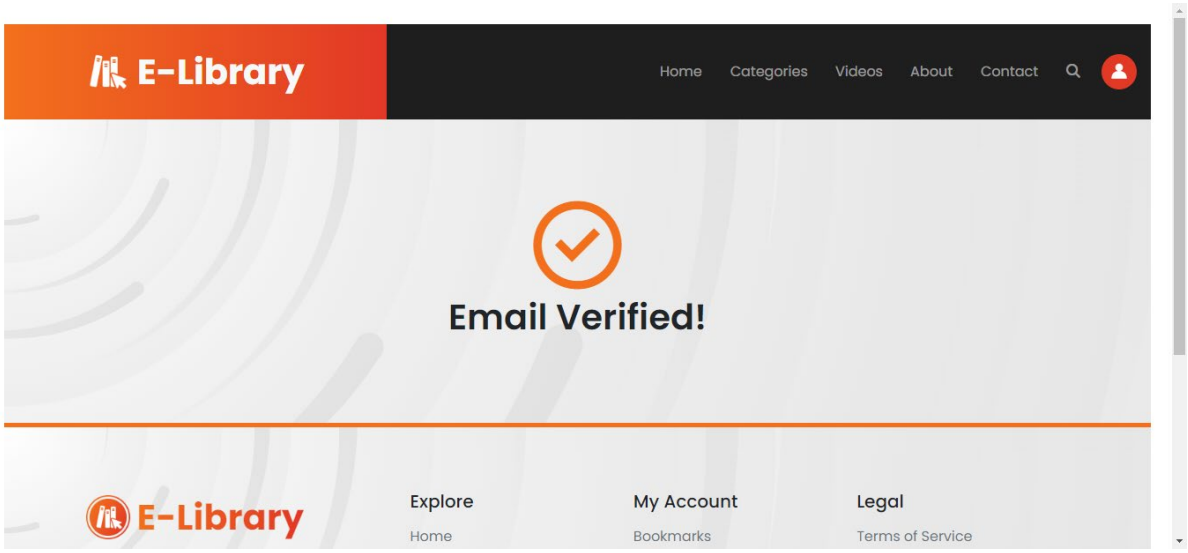


After successfully creating an account, you will receive an email to verify your email address. Check your mail to inbox and open the verification link to verify your email.





Upon clicking the verification link, your email address will be verified.

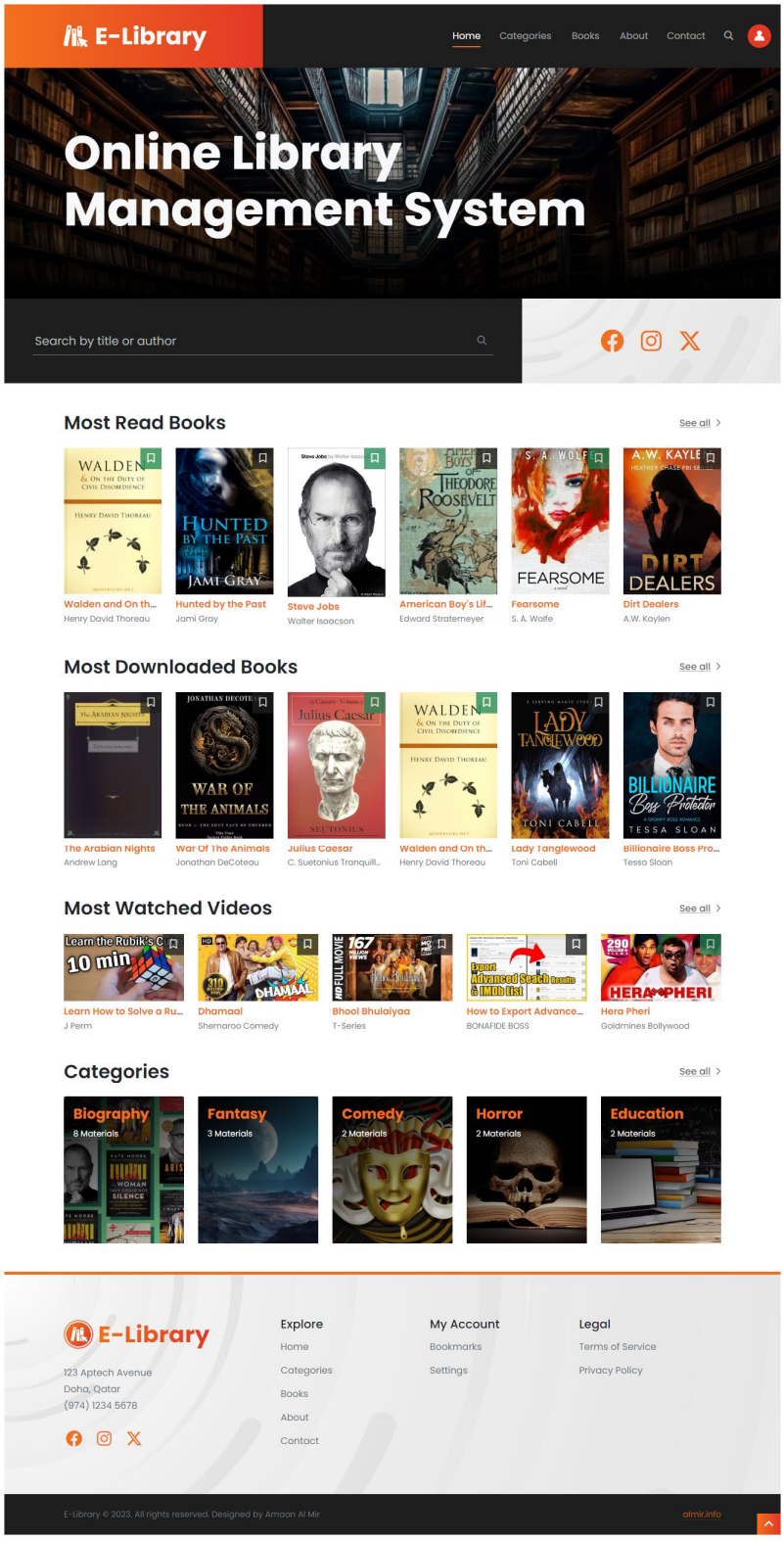


2. Login

If you're a returning user, you can use the login page to get into the website. Enter your email and p

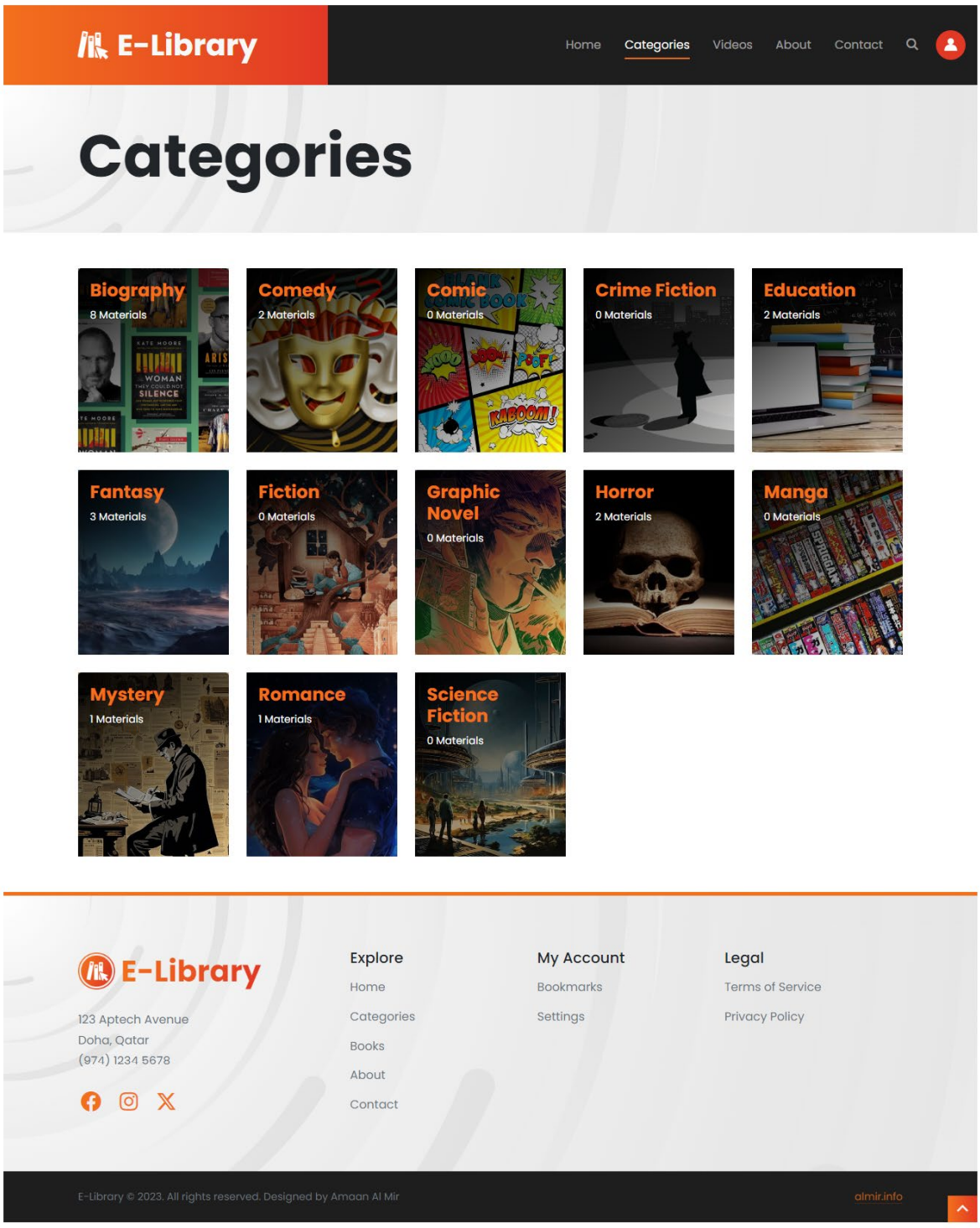
3. Homepage

The home page displays some trending books and videos.



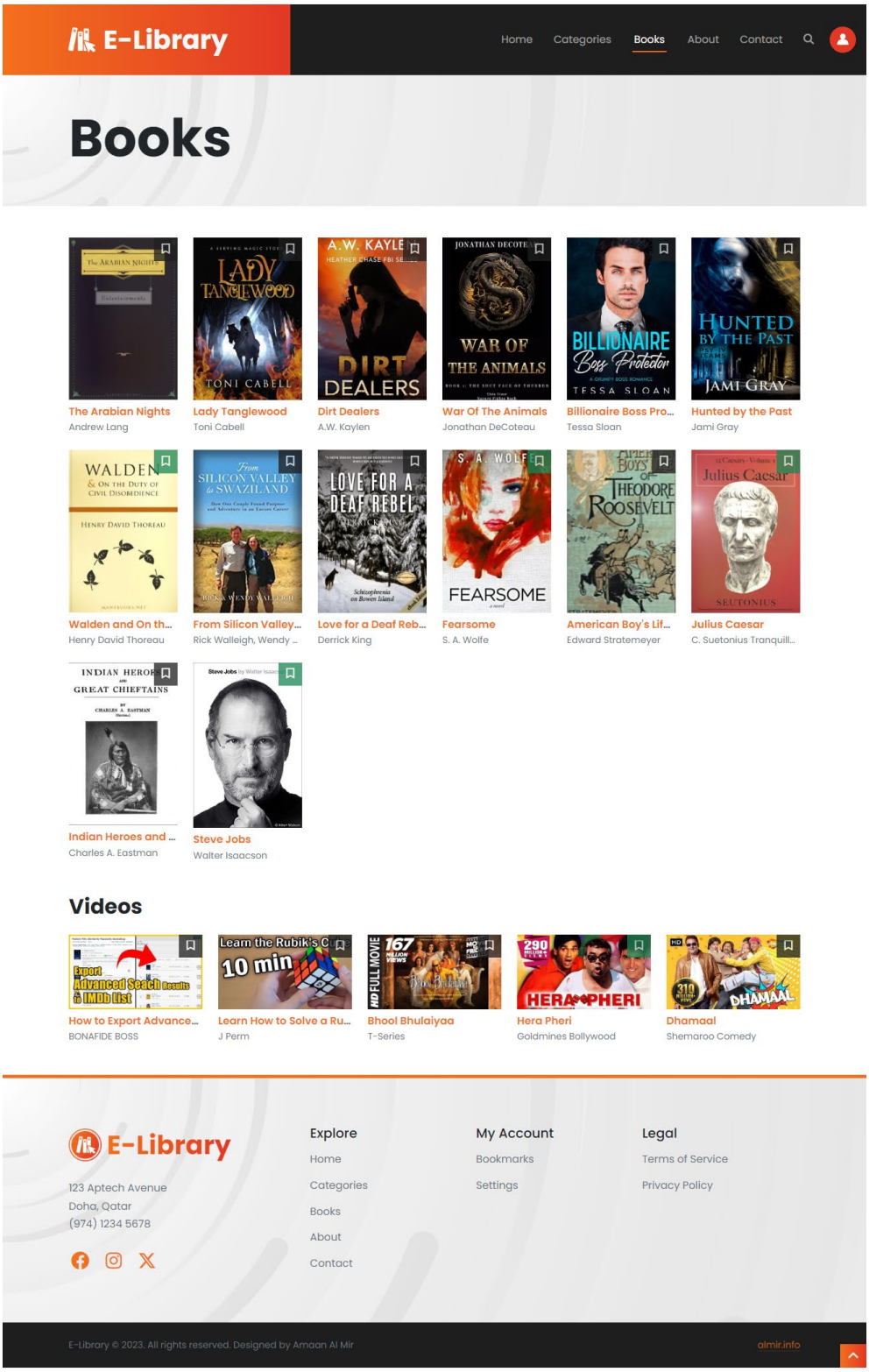
4. Categories

On this page you can find a list of categories available.



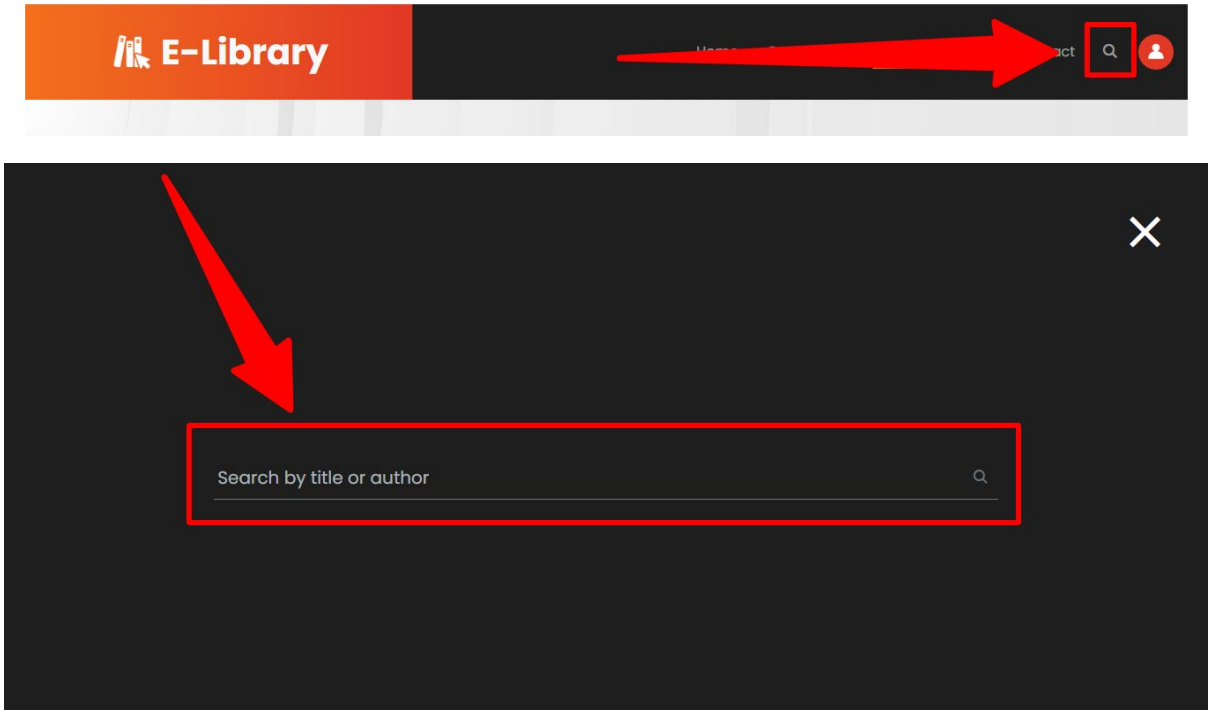
5. Books & Videos

On this page you can find list of books and videos.



6. Search

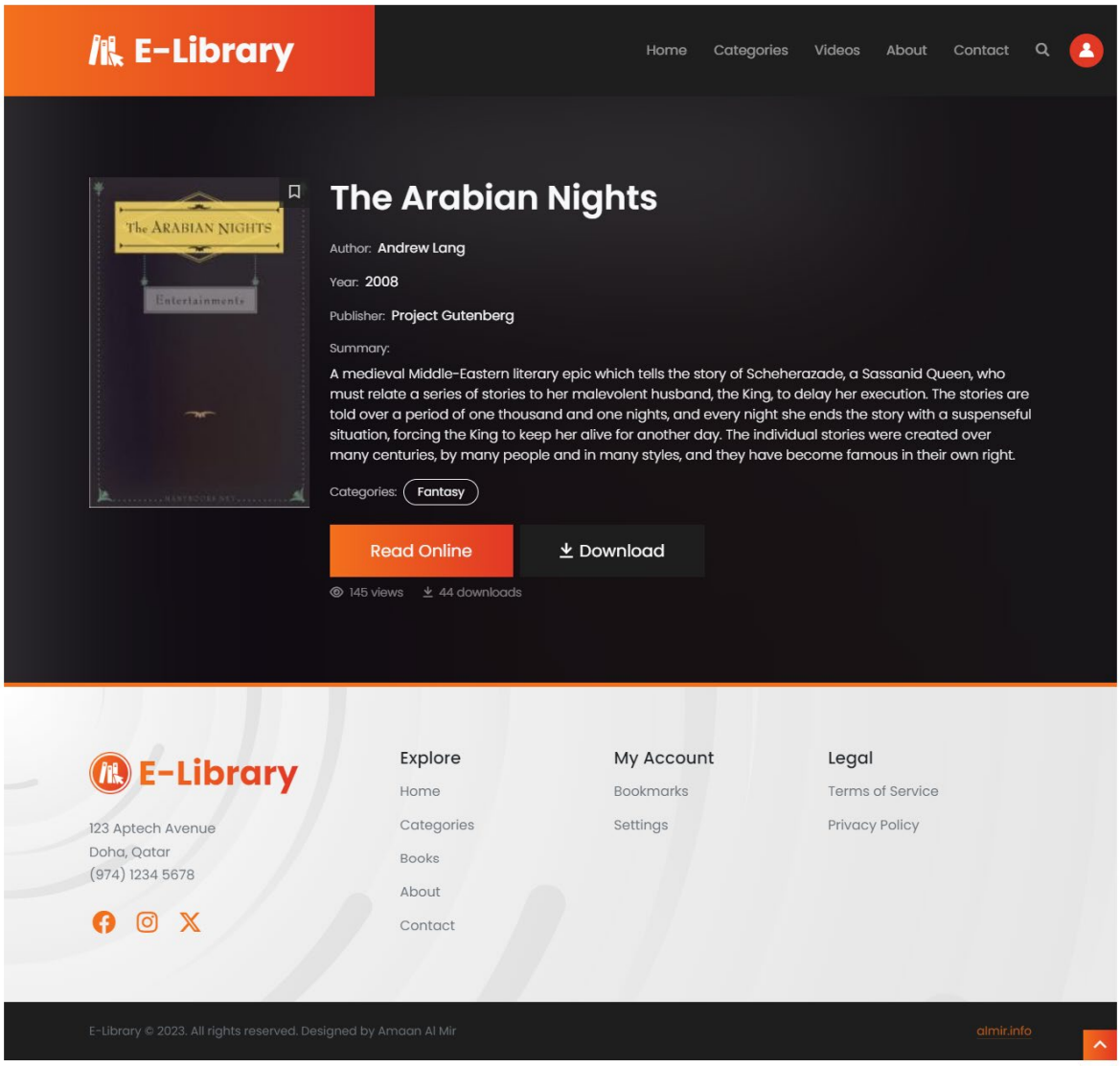
You can open the search menu from any page click on the search button given in the top navigation bar.



You can search for a book or a video by its title or author.

7. Book Details

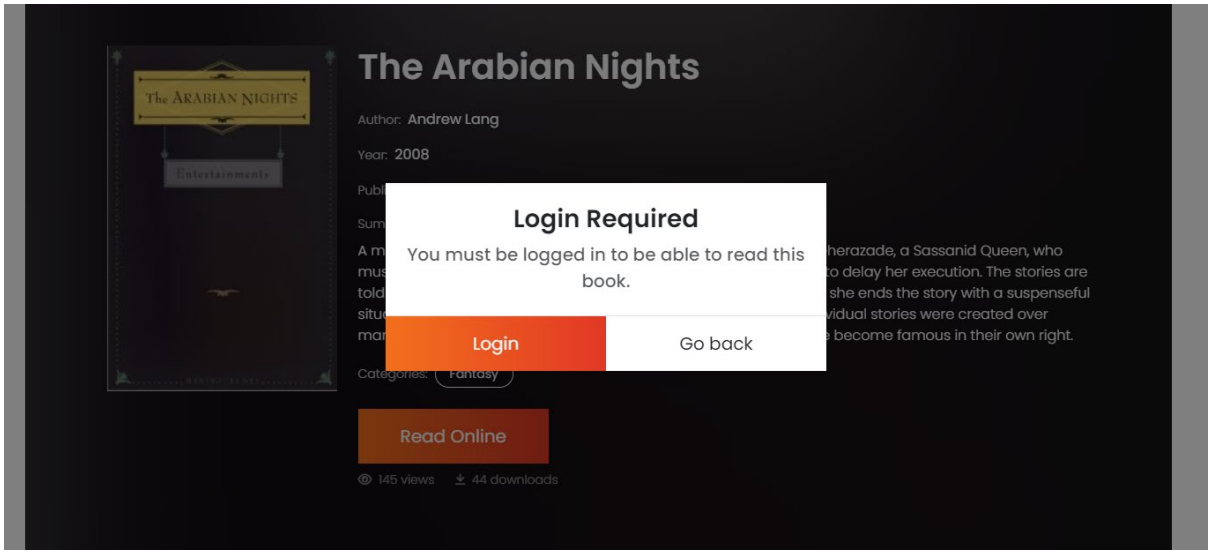
You can view any book or video details by clicking on it.



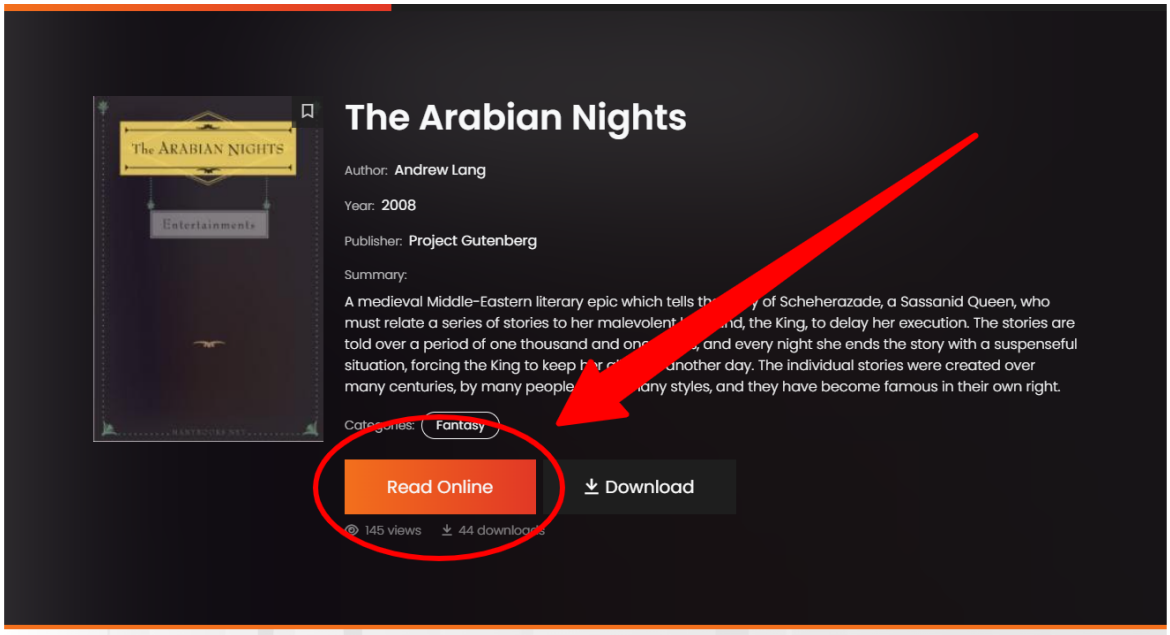
Note: Not all books can be downloaded.

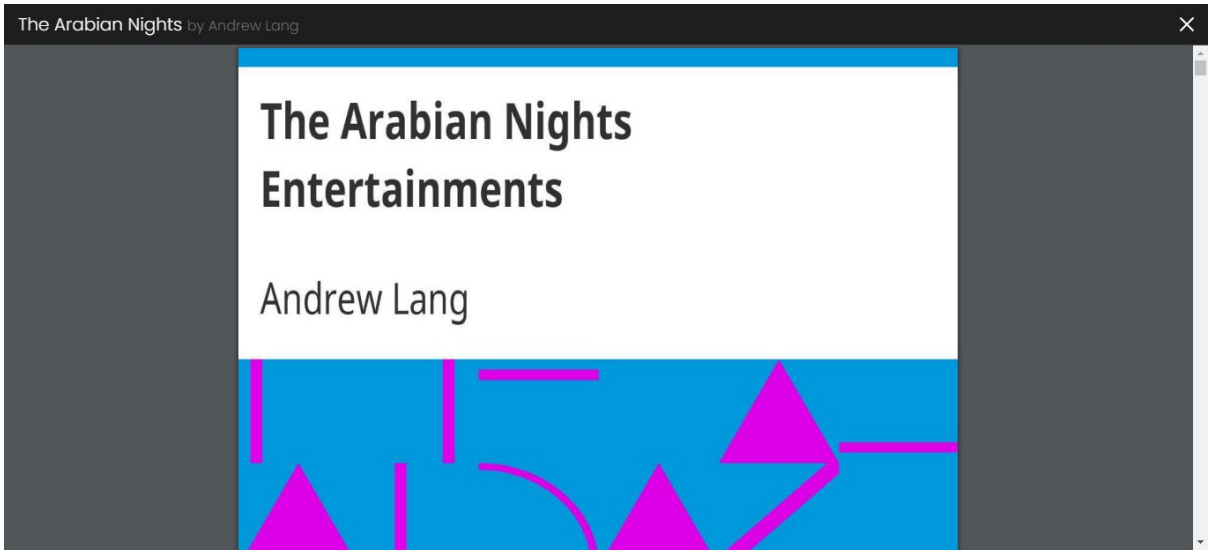
8. Read a Book Online

Non-logged in users will be asked to login before they can read the book online.



Logged-in users can click on the read button the read the book.

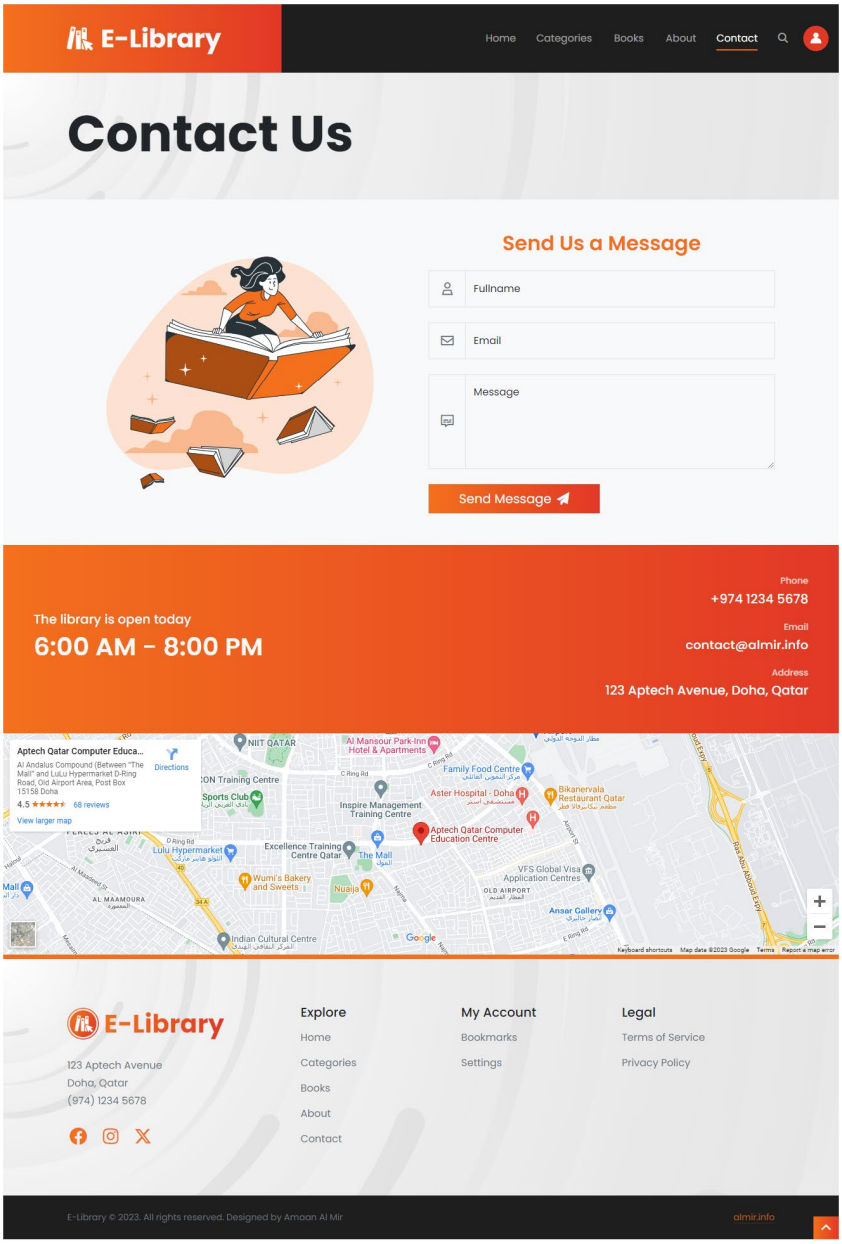




Some books are also allowed to be downloaded by the librarian.

9. About & Contact pages

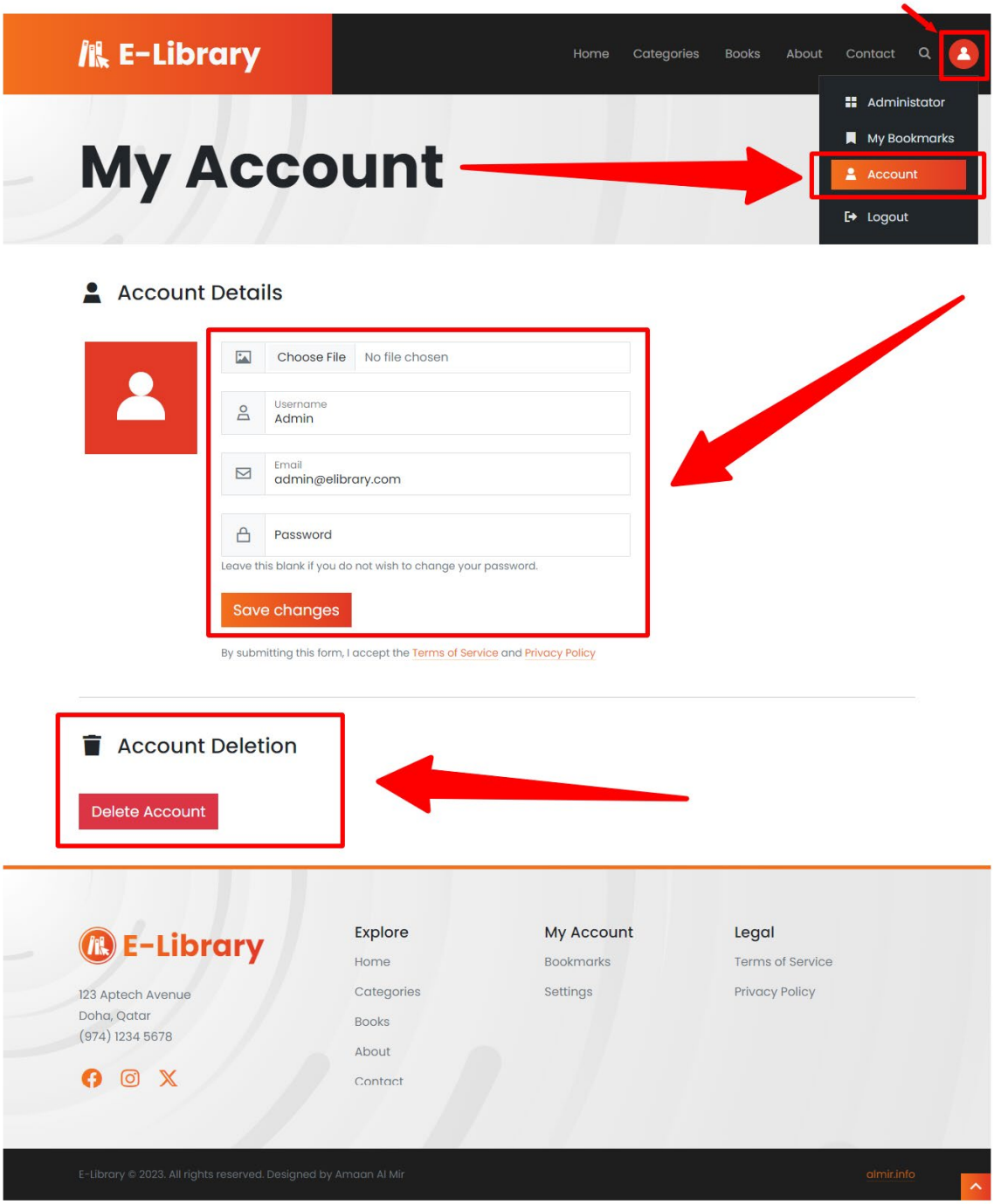
On these pages, you can find general information about the library like timings, no. of books available online, contact information, and location.



You can also use the contact form to send message to the librarian.

10. User Account

On this page, you can update your account details like avatar, email and password.

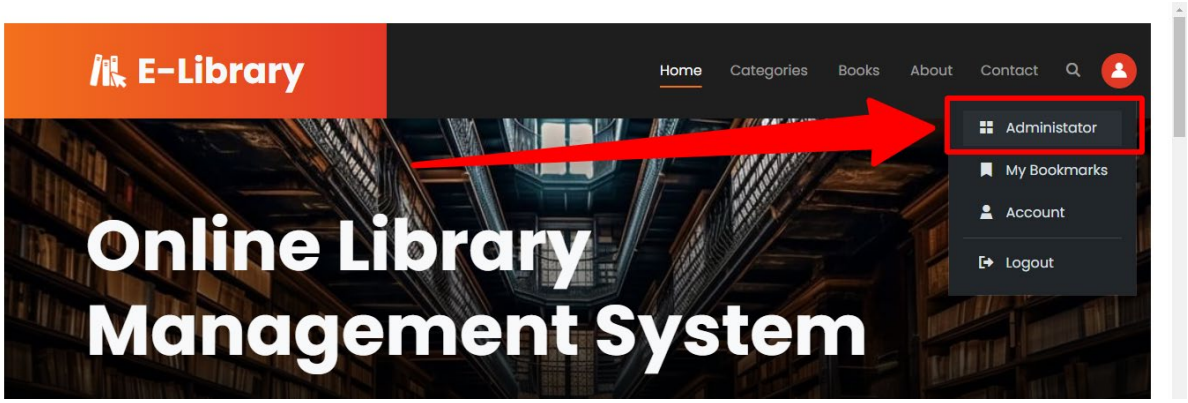


You also have the option to delete your account.

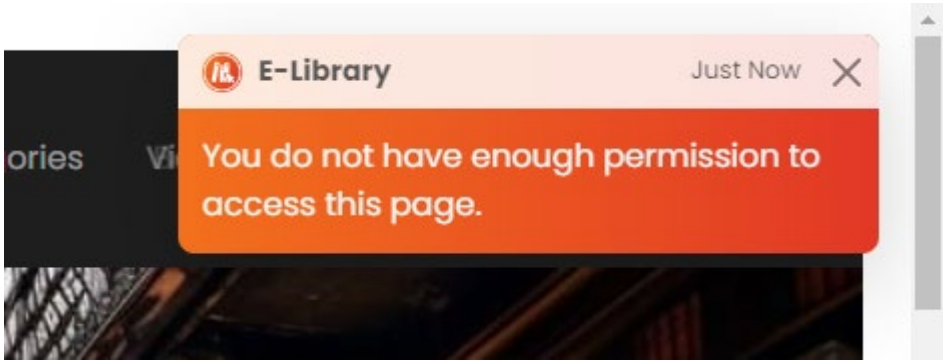
11. Website Administration

The website offers administration services that can the librarian manage the E-Library's categories, books, and users.

To go to this Admin dashboard, click on the button shown in the picture below:

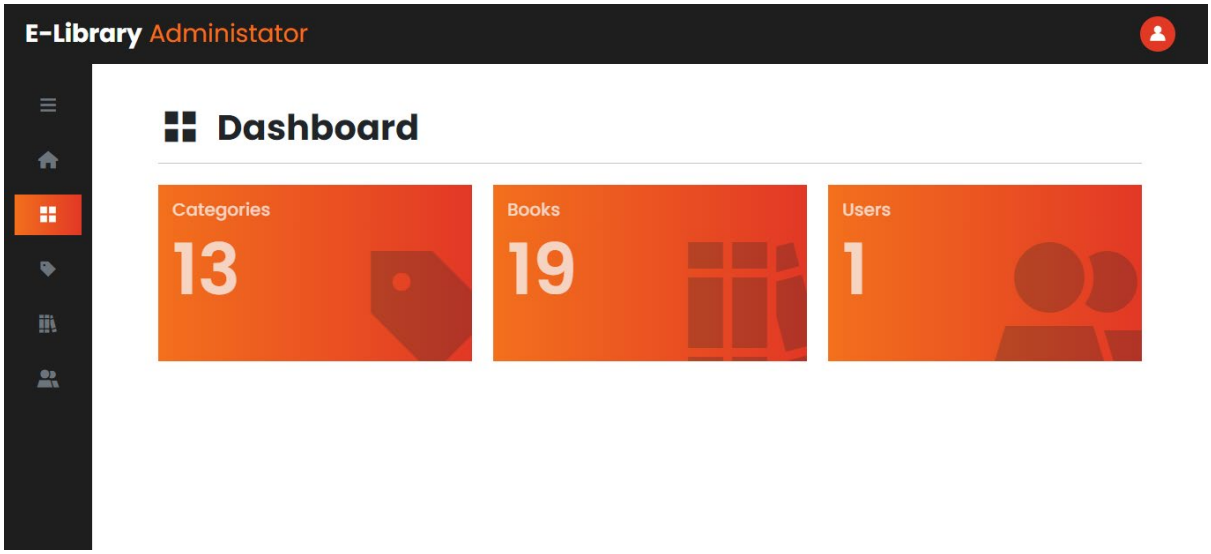


Users without admin permission will be denied access to the website administration pages.

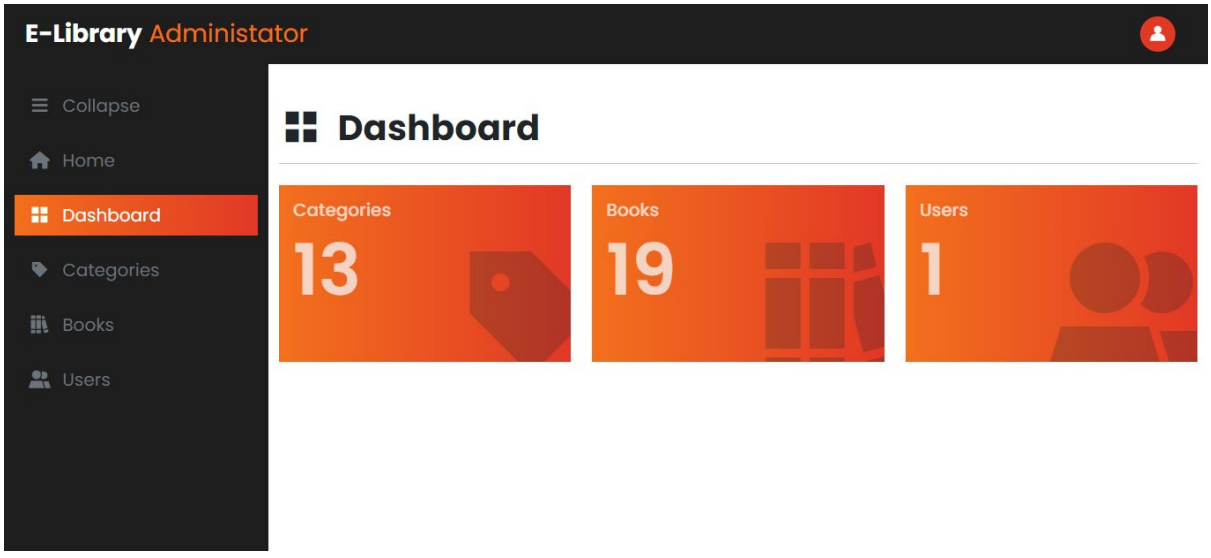


1. Dashboard

On this page, admin can see an overview of categories, books, and users in the online database.

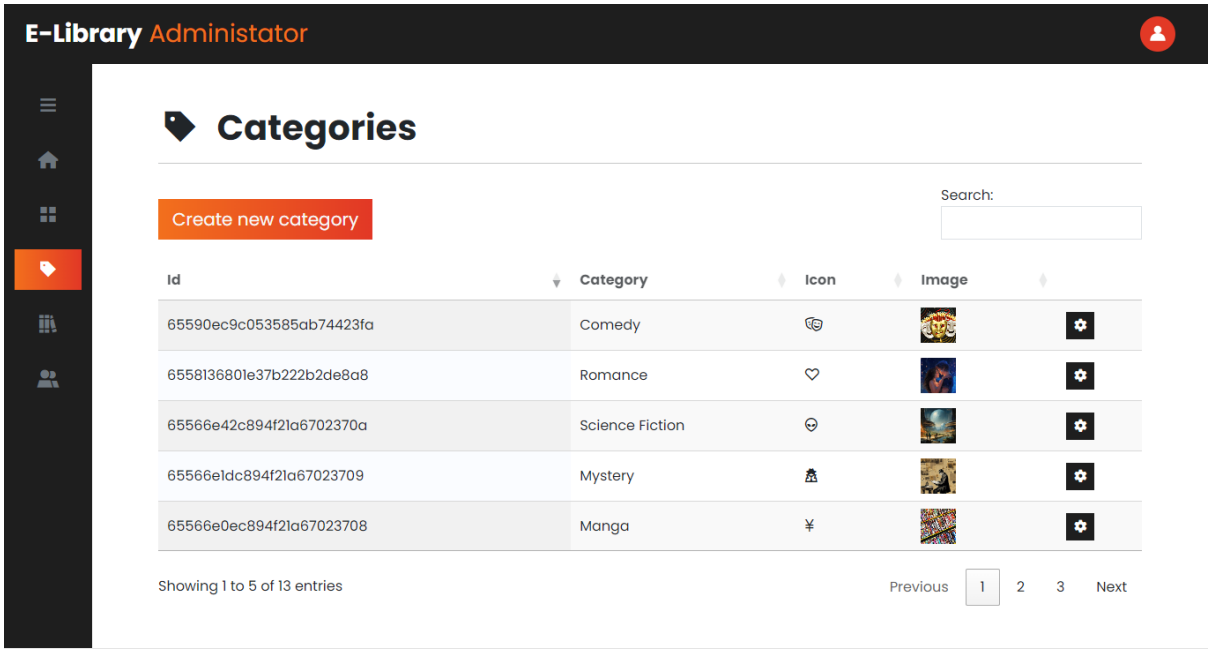


You can use side column to navigate through admin pages.



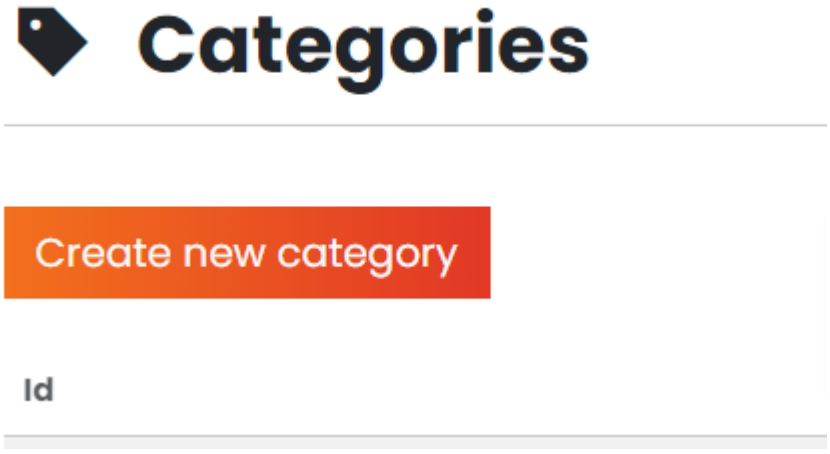
2. Manage Categories

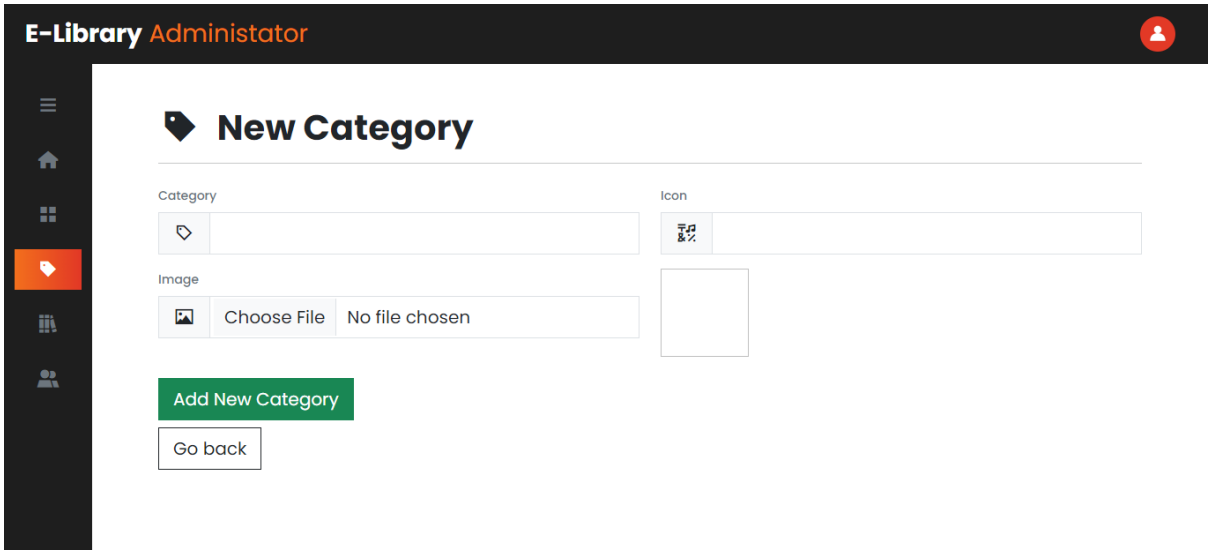
On this page, admin can create a new category or update and delete existing categories.



Create New Category

To create a new category, click on the “Create new category” button.

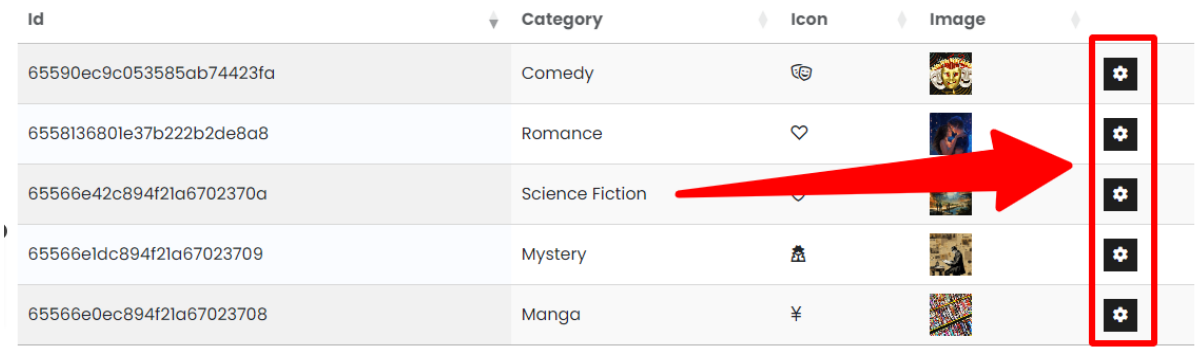


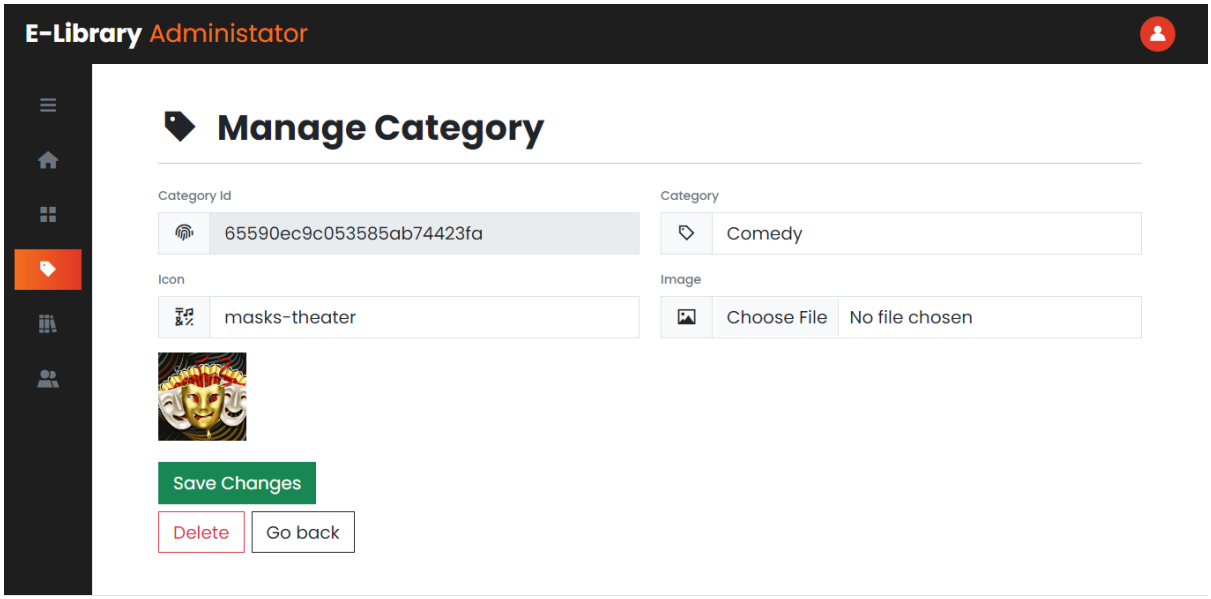


Fill the necessary details like category title, icon and an image, then click on the add button to add the category to the database.

Update/Delete a Category

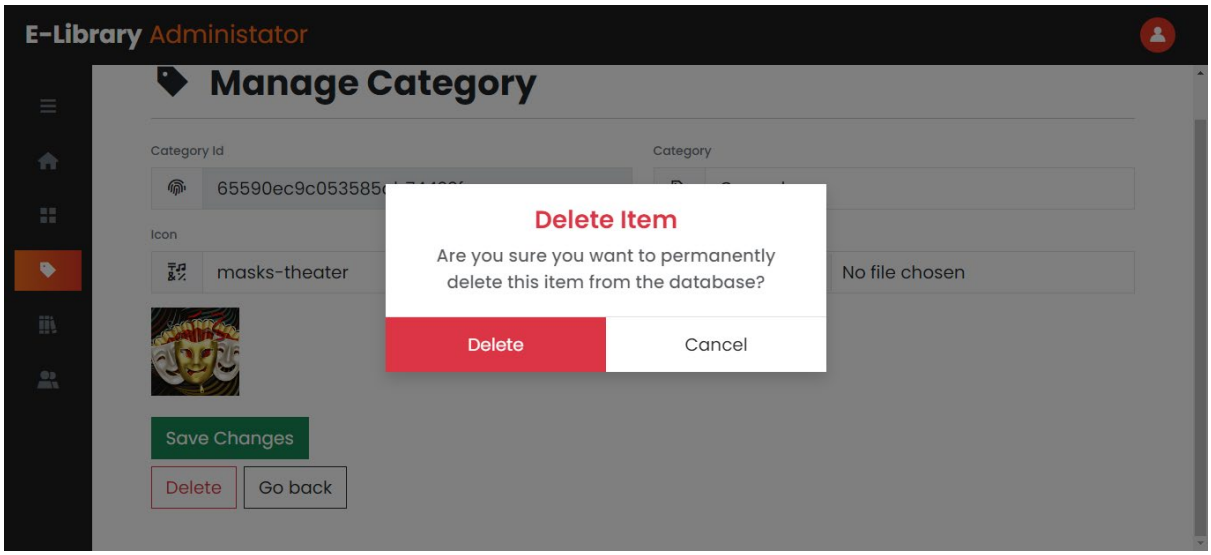
To update or delete a category, click the gear icon next to the category to open the edit page where you can edit or delete it.





Make changes you want to make and click the green button to save them.

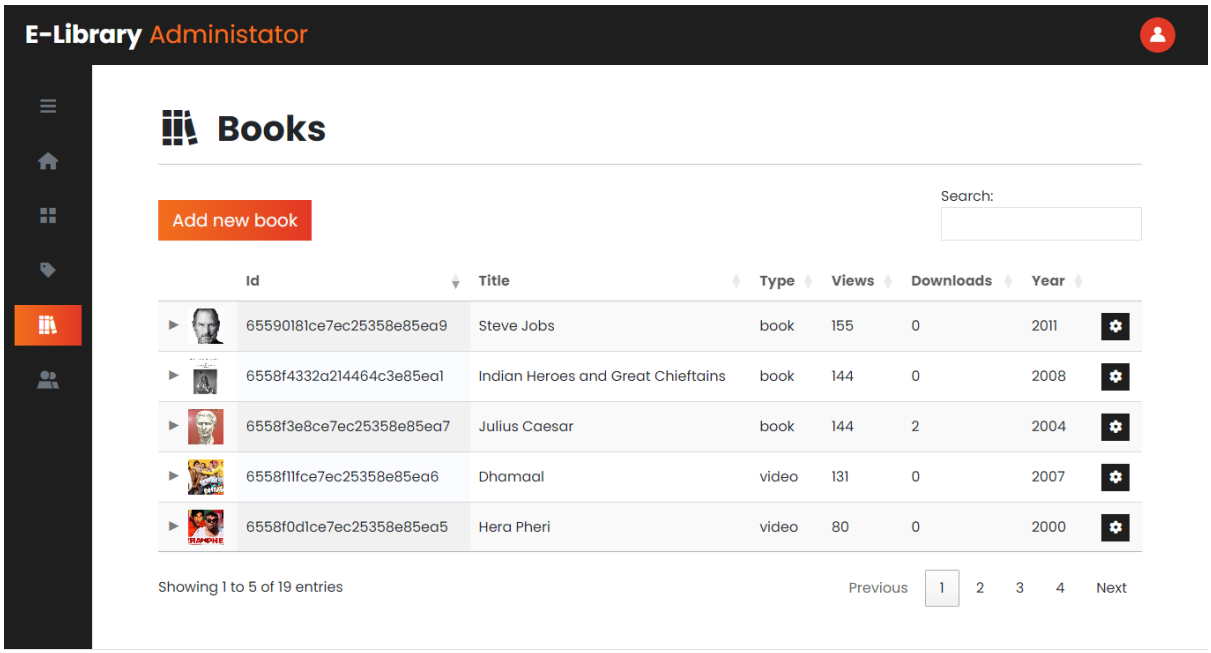
To delete it, click the delete button next to it.



Then confirm the deletion by clicking on the delete button again.

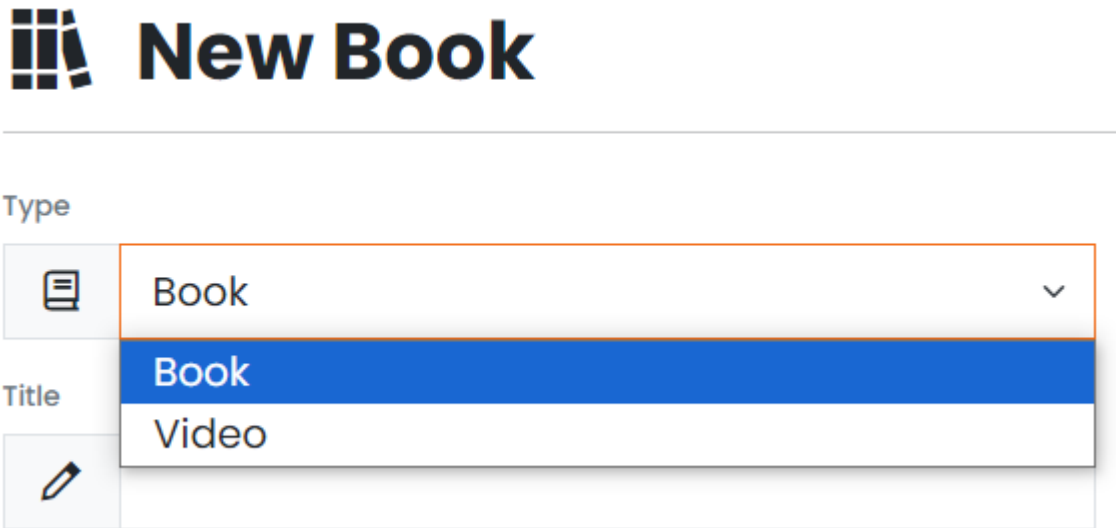
3. Managing Books and Videos

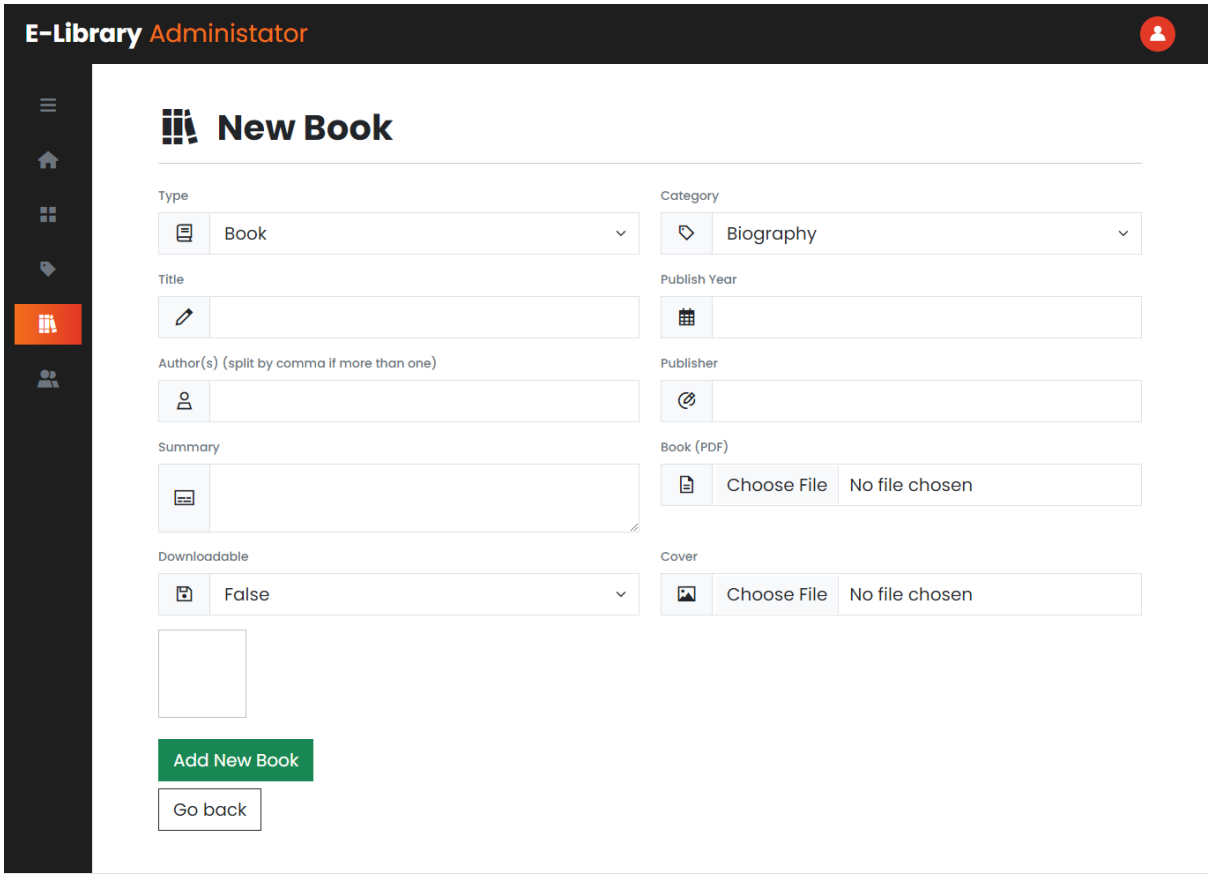
This is also similar to categories, only data is different.



Add New Book or Video

Select book or video by changing type.





Fill up the rest of the details to add a book or a video.

Note: Only PDFs are supported for books. And for videos, enter an **embed YouTube URL**.

Example of an embedded YouTube URL:

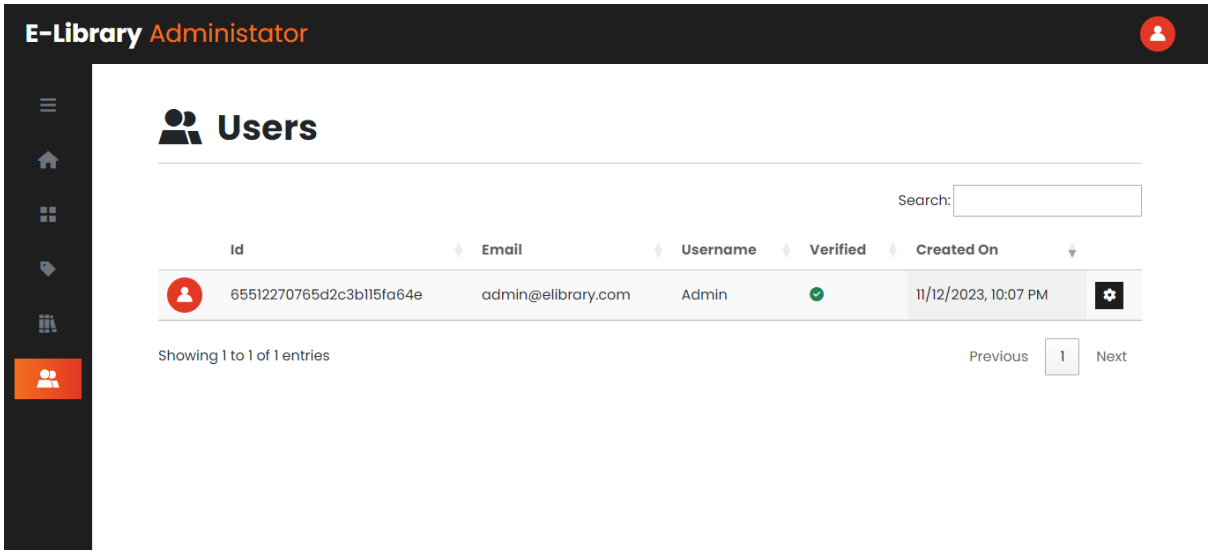
https://www.youtube.com/**embed**/jzYxbnHHhY4

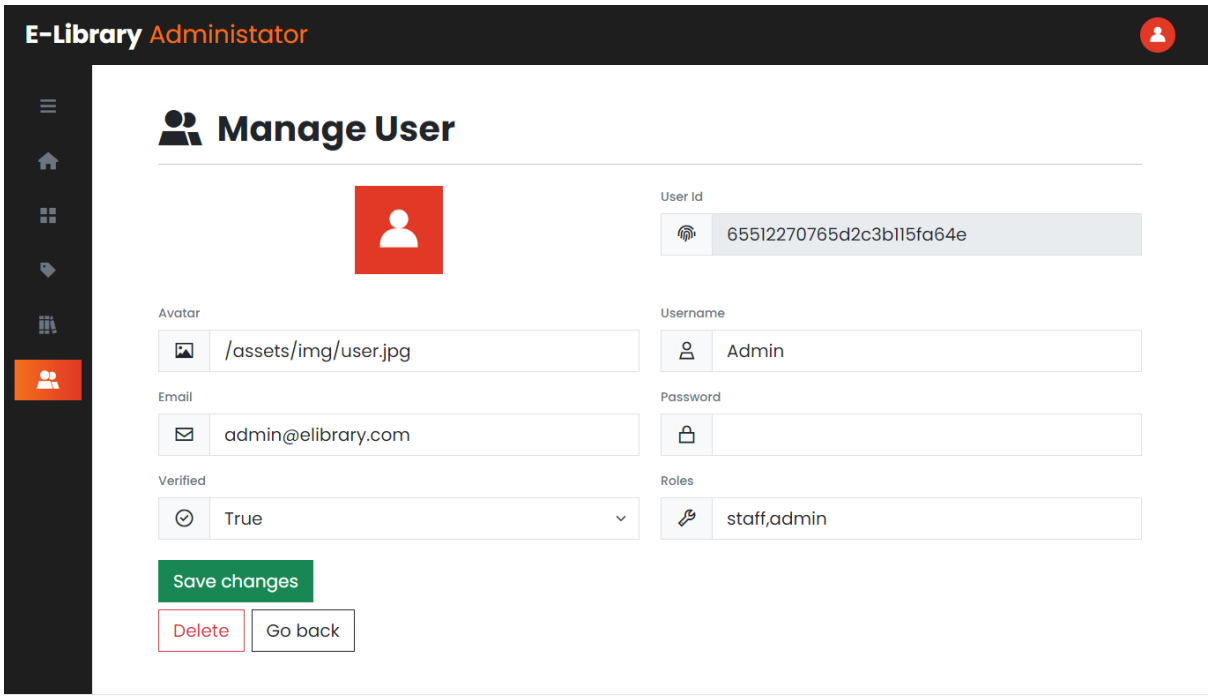
Update/Delete a Book or Video

This is similar to updating or deleting a category.

4. Managing Users

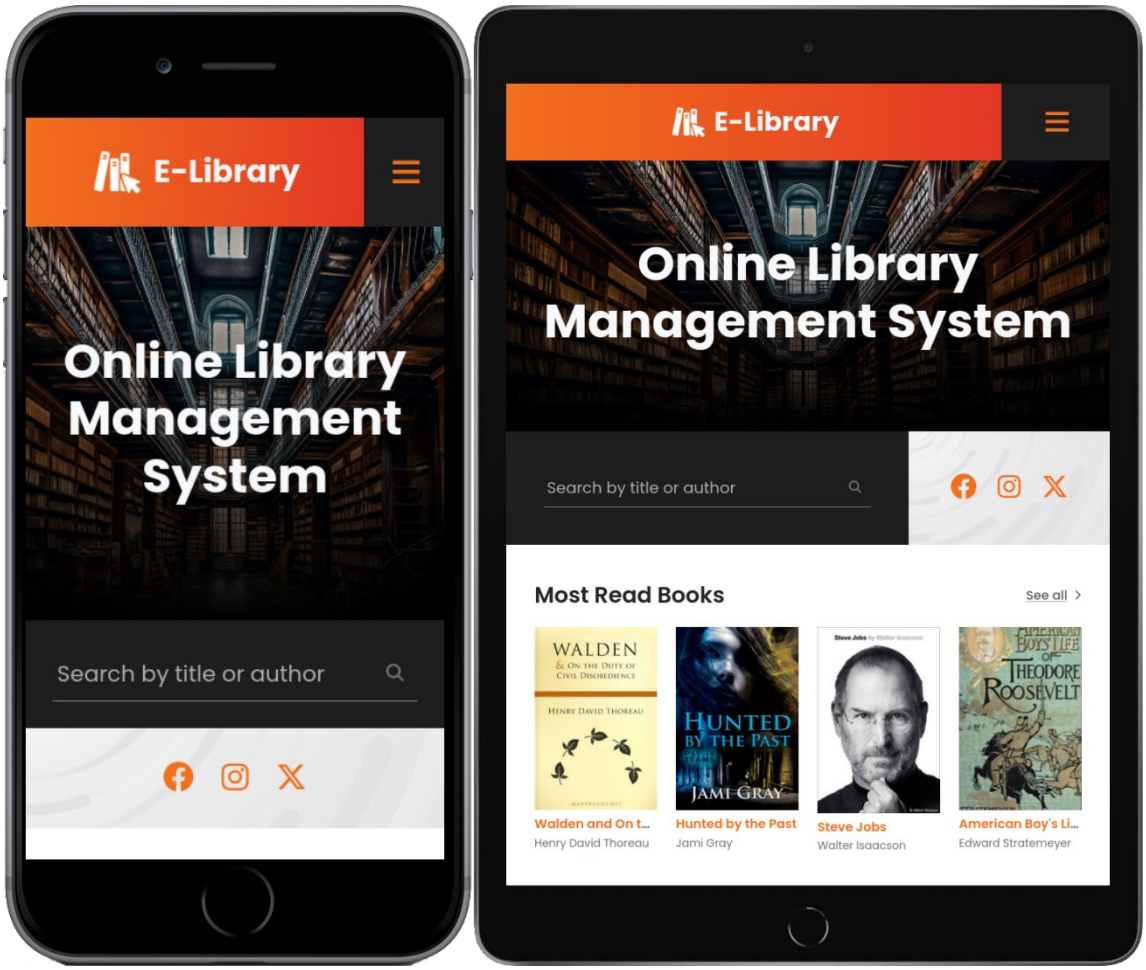
Managing users is also similar to managing categories and books.



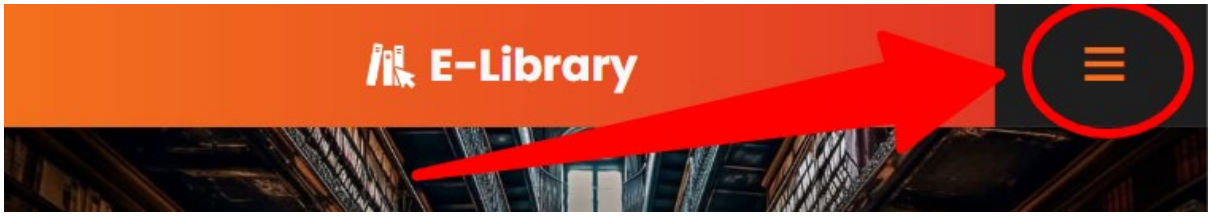


12. Mobile & Tablet View

The website responsive and mobile-friendly.



You can open the nav by clicking on the button shown below:



Developer Guide

Technology Stack

Frontend:

- HTML
- CSS
- JavaScript
- jQuery v3.7.1
- Bootstrap v5.3.1
- Font Awesome v6.4.2

Backend:

- Python v3.11.6
- Flask v3.0.0
- Flask-PyMongo v2.3.0
- Flask-WTF v1.2.1
- Email-Validator v2.1.0
- Flask-Mail v0.9.1
- Requests v2.31.0
- Filestack-Python v3.5.0

Database:

- MongoDB

Web Services:

- ImgBB – Image storage
    - Used to store book covers, video thumbnails, and user avatars
- Filestack – File Storage
    - Used to store books
- YouTube – Video Storage
    - Used to store videos
- Google SMTP (Mail) Server
    - Used to send email address verification and password recovery emails

How To Run the Application?

Prerequisites

Make sure you have the following downloaded and installed in your system:

- Python 3.9 - 3.11
- If you didn't get the project files along with this documentation, you can download them from: https://github.com/BonaFideBOSS/E-Library
- Request the **config.py** file from me which contains all the secrets like database connection string and APIs

Method 1

Simple double-click on the start_app.bat file to start the application. It will take a few seconds to start up in the first run.

Note: If this is your time, it will take much longer to initialize and start up the application.

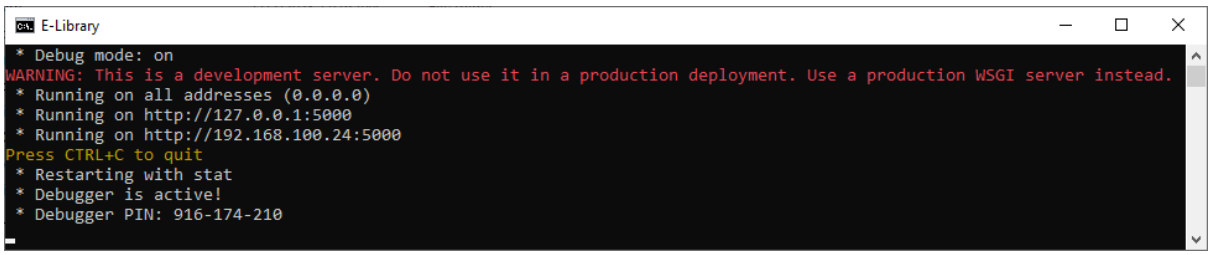
Method 2

If method 1 fails to start the application, open CMD inside the project directory and run the following commands:

```
py -m venv venv
venv\scripts\activate
pip install -r requirements.txt
py -m flask --debug run --host=0.0.0.0
```

Note: Each command should be executed separately in a new line.

If successful, you should see something like this on your screen:



Then, simply open a browser and go the address shown in your terminal.

Source Code

Website Structure

website	app.py	config.py
assets This folder contains CSS, JavaScript, and image files views This folder contains HTML files __init__.py admin.py admin_forms.py auth.py forms.py helpers.py mailer.py user.py views.py		

1. Homepage

The below function is used to render the home page.

website/views.py

```python
# Home page
@views.route("/")
def home():
    return render_template("home.html")
```

The home page also displays a list of few books and categories which are pulled through API via ajax.

website/views.py

```python
# API to get list of books and videos to display on the home page
@views.route("/home-books-api", methods=["POST"])
def home_books_api():
    book_type = request.form.get("book_type", type=str)
    sort = request.form.get("sort", type=str)
    limit = 6
    books = list(db.Books.find({"type": book_type}).sort(sort, -1).limit(limit))
    return render_template("partial/home-features.html", books=books)


# API to get list of categories to display on the home page
@views.route("/home-categories-api", methods=["POST"])
def home_categories_api():
    categories = get_categories("book_count", -1, limit=5)
    return render_template("partial/home-categories.html", categories=categories)
```

website/assets/custom/js/home.js

```javascript
async function get_books(book_type, sort, container) {
  const url = '/home-books-api'
  const data = { 'book_type': book_type, 'sort': sort }
  await get_api_data(url, data, container)
  all_loaded++
  if (all_loaded == 3 && is_user_logged_in) { bookmark() }
}

function get_categories() {
  const url = '/home-categories-api'
  get_api_data(url, {}, '#featured-categories')
}

async function get_api_data(url, data, container) {
  await $.post(url, data).done(function (response) { $(container).html(response) })
}
```

2. Registration

The register form is used to render and validate the registration form you see on the register page.

website/forms.py

```python
# Registration Form - Used to validate new account creation
class RegisterForm(FlaskForm):
    # Username field
    username = StringField(
        "Username",
        validators=[
            # Required field
            DataRequired(message="Please enter a username."),
            # Minimum length = 4 | Maximum length = 26
            Length(min=4, max=26, message="Username must be 4 to 26 characters long."),
        ],
    )
    # Email field
    email = EmailField(
        "Email",
        validators=[
            # Required field
            DataRequired(message="Please enter an email address."),
            # Must be valid email address
            Email(message="Please enter a valid email address."),
        ],
    )
    # Password field
    password = PasswordField(
        "Password",
        validators=[
            # Requiredd field
            DataRequired(message="Please enter a password."),
            # Minimum length = 8
            Length(min=8, message="Password must be at least 8 characters long."),
        ],
    )

    # Function to validate if email is unique
    def validate_email(self, email):
        # Find any user with the same email
        user = get_user_by_email(email.data)
        if user:
            # Raise an error if a user is found
            raise ValidationError("This email is already is use.")
```

The below code is used to render the register page and validate the register form on POST request.

website/auth.py

```python
# Register page
@auth.route("/register/", methods=["GET", "POST"])
def register():
    # Initialize Form
    form = RegisterForm()

    # Validate form on submit
    if form.validate_on_submit():
        user = form.data
        user.pop("csrf_token")
        user["password"] = encrypt_message(user["password"])
        user["created_on"] = datetime.utcnow()
        user["verified"] = False
        user["avatar"] = DEFAULT_AVATAR
        db.Users.insert_one(user)
        send_email_verification_mail(user["email"])
        add_user_to_session(user)
        flash("Successfully created a new account.")
        return redirect(url_for("views.home"))

    return render_template("auth/register.html", form=form)
```

The below function is used to mail a verification link after user successfully creates a new account.

website/mailer.py

```python
# Function to send email verification link to user's email address
def send_email_verification_mail(user_email: str):
    @copy_current_request_context
    def send_mail(to):
        token = generate_token()
        user = db.Users.find_one_and_update({"email": to}, {"$set": {"token": token}})

        if user:
            v_link = f"{request.host_url}verify-email?_id={user['_id']}&token={token}"
            email = Message("Email Verification", recipients=[to])
            email.html = f"""
            <p>Use the link below to verify your email:</p>
            <h1><a href="{v_link}">Click here to verify</a></h1>
            """
            mail.send(email)

    threading.Thread(target=send_mail, args=(user_email,)).start()
```

3. User Login

The below code is used to render the login page.

website/auth.py

```python
# Login page
@auth.route("/login/", methods=["GET", "POST"])
def login():
    # Initialize Login Form
    form = LoginForm()

    # Validate form on submit
    if form.validate_on_submit():
        user = db.Users.find_one({"email": form.email.data})
        remember = True if request.form.get("remember") == "on" else False
        add_user_to_session(user, remember)
        flash("Successfully logged in.")
        return redirect(url_for("views.home"))

    return render_template("auth/login.html", form=form)
```

The login form is used for rendering and validating the user login form passed to the login view.

website/forms.py

```python
# Login Form - Used to validate existing account credentials
class LoginForm(FlaskForm):
    email = EmailField(
        "Email", validators=[DataRequired(message="Please enter your email address.")]
    )
    password = PasswordField(
        "Password", validators=[DataRequired(message="Please enter your password.")]
    )

    # Function to validate login credentials
    def validate(self, extra_validators=None):
        if not super().validate():
            return False

        user = get_user_by_email(self.email.data)

        if not user:
            self.email.errors.append("Email not found.")
            return False

        # Check if password matches with saved password
        password = encrypt_message(self.password.data)
        if password != user["password"]:
            self.password.errors.append("Password is incorrect.")
            return False

        return True
```

After successfully logging in, the user is added to the session using the below function.

website/auth.py

```python
# Function to add user to the session
def add_user_to_session(user: dict, remember: bool = False):
    user["_id"] = str(user["_id"])
    session["user"] = user
    session.permanent = remember
```

4. Password Recovery

The below function is used to render a page to let user submit a request to reset their account password.

website/auth.py

```python
# Page to send forgot password request
@auth.route("/forgot-password/", methods=["GET", "POST"])
def forgot_password():
    form = ForgotPassword()

    if form.validate_on_submit():
        send_password_reset_mail(form.email.data)
        flash("Password reset link sent to your email.")
        return redirect(url_for("auth.login"))

    return render_template("auth/forgot-password.html", form=form)
```

4.1. Mail

The below function is used to send a password recovery mail to the user.

website/auth.py

```python
# Function to send a password recovery mail
def send_password_reset_mail(user_email: str):
    @copy_current_request_context
    def send_mail(to):
        token = generate_token()
        user = db.Users.find_one_and_update({"email": to}, {"$set": {"token": token}})

        if user:
            v_link = f"{request.host_url}reset-password?_id={user['_id']}&token={token}"
            email = Message("Reset Password", recipients=[to])
            email.html = f"""
            <p>Use the link below to reset your password:</p>
            <h1><a href="{v_link}">Click here to reset your password</a></h1>
            <p>Please ignore this email if it wasn't requested by you.</p>
            """
            mail.send(email)

    threading.Thread(target=send_mail, args=(user_email,)).start()
```

4.2. Reset Password

The below function is used validate the password recovery link and allows user to reset their account password.

website/auth.py

```python
# Page to reset password
@auth.route("/reset-password/", methods=["GET", "POST"])
def reset_password():
    user_id = request.args.get("_id")
    token = request.args.get("token")
    form = ResetPassword(user_id=user_id, token=token)

    if form.validate_on_submit():
        db.Users.find_one_and_update(
            {"_id": ObjectId(user_id), "token": token},
            {"$set": {"password": encrypt_message(form.password.data)}},
        )
        flash("Your password has been successfully updated.")
        return redirect(url_for("auth.login"))

    return render_template("auth/reset-password.html", form=form)
```

The forms used to validate user inputs are inside the forms.py file.

5. Categories

The below function is used to retrieve a list of categories along with the number of books in each of them.

website/views.py

```python
# Function to categories along with number of books in them
def get_categories(sort: str = "category", sort_order: int = 1, limit: int = None):
    limit = [{"$limit": limit}] if limit else []
    pipeline = [
        {
            "$lookup": {
                "from": "Books",
                "localField": "_id",
                "foreignField": "category_id",
                "as": "books",
            }
        },
        {"$addFields": {"book_count": {"$size": "$books"}}},
        {"$sort": {sort: sort_order}},
    ] + limit
    categories = list(db.Categories.aggregate(pipeline))
    return categories
```

Then, the below function is used to render the categories page.

website/views.py

```python
# Categories page
@views.route("/categories/")
def categories():
    categories = get_categories()
    return render_template("categories.html", categories=categories)
```

6. Books

The below function is used to retrieve a list of books and videos from the database and render the books page. The search function is also included in this code.

website/views.py

```python
# Books page
@views.route("/books/")
@views.route("/books/<category_id>/")
def books(category_id=None):
    # Get search parameters if given
    search = request.args.get("search", default=None, type=str)
    query = {"category_id": ObjectId(category_id)} if category_id else {}
    # Filter out database if search parameters are given
    if search:
        search = db_searcher(["title", "author"], search)
        query.update(search)
    books = list(db.Books.find(query))
    return render_template("books.html", books=books)
```

7. Book Details

The below function is used to display the details of a book.

website/views.py

```python
# Book Details page
@views.route("/books/<book_id>")
def view_book(book_id):
    pipeline = [
        {"$match": {"_id": ObjectId(book_id)}},
        {
            "$lookup": {
                "from": "Categories",
                "localField": "category_id",
                "foreignField": "_id",
                "as": "category",
            }
        },
    ]
    book = list(db.Books.aggregate(pipeline))[0]
    return render_template("book-details.html", book=book)
```

7.1. Download

The below API route is used to let users download a book.

website/views.py

```python
# Book Details page
@views.route("/books/<book_id>")
def view_book(book_id):
    pipeline = [
        {"$match": {"_id": ObjectId(book_id)}},
        {
            "$lookup": {
                "from": "Categories",
                "localField": "category_id",
                "foreignField": "_id",
                "as": "category",
            }
        },
    ]
    book = list(db.Books.aggregate(pipeline))[0]
    return render_template("book-details.html", book=book)
```

7.2. Updating Book Stats

The below function is used to update a book or video view and download count.

website/views.py

```python
# API to update book stats
# Example: When the user clicks on read/download button,
# a request is sent to this URL via ajax to increment the view/download count
@views.route("/books/update-stats/", methods=["POST"])
def update_book_stats():
    book_id = request.form.get("book_id")
    stats = request.form.get("stats")
    if book_id and stats in ["view", "download"]:
        db.Books.find_one_and_update(
            {"_id": ObjectId(book_id)},
            {"$inc": {f"{stats}_count": 1}},
        )
    return "", 200
```

8. User Account

The below function is used to render the user account page and process user details on POST request.

website/user.py

```python
# Function to render user's account page
@user.route("/account", methods=["GET", "POST"])
def account():
    form = AccountForm()
    user = session["user"]

    if form.is_submitted():
        form.user_id.data = ObjectId(user["_id"])
        if not form.password.data:
            form.password.data = session["user"]["password"]

    if form.validate_on_submit():
        data = form.data
        data.pop("csrf_token")
        data.pop("user_id")

        if data["email"] != user["email"]:
            data["verified"] = False

        if data["password"] != user["password"]:
            data["password"] = encrypt_message(data["password"])

        if data["avatar"]:
            # Upload avatar to cloud
            avatar_url = upload_image_to_cloud(data["avatar"], form.username.data)
            data["avatar"] = avatar_url if avatar_url else user["avatar"]
        else:
            data["avatar"] = user["avatar"]

        # Update user details
        user = db.Users.find_one_and_update(
            {"_id": ObjectId(user["_id"])},
            {"$set": data},
            return_document=True,
        )
        add_user_to_session(user)
        if not user["verified"]:
            send_email_verification_mail(user["email"])
        flash("Successfully updated account details.")

    return render_template("user/account.html", form=form)
```

9. Website Administration

The admin functions are located inside the admin.py and admin_forms.py files.

9.1. User Permissions

Before every request, the below function is used to check if the user has admin roles.

website/admin.py

```python
# Function to check if user is admin
@admin.before_request
def is_user_admin():
    userrolees = []

    # Check if user is logged in
    if not "user" in session:
        return redirect(url_for("auth.login"))

    # Check if user has any role
    if "roles" in session["user"]:
        userrolees = session["user"]["roles"]

    # Check if user has admin role
    if not "staff" in userrolees:
        flash("You do not have enough permission to access this page.")
        return redirect(url_for("views.home"))
```

If the user does not have the required role, they are redirected to the homepage.

9.2. Role Authorization

Not all users who can view the admin dashboard are permitted to use all features of the website.

For example, any staff can create a new record, but only an admin can update or delete an item from the database.

The below function is used to authorize certain roles before the start of any function.

website/admin.py

```python
# Decorator to authorize roles
def authorize_roles(*roles):
    def decorator(f):
        @wraps(f)
        def decorated_function(*args, **kwargs):
            # Check if user has any of the required roles
            if not any(role in session["user"]["roles"] for role in roles):
                flash("You do not have enough permission to perform this action.")
                return redirect(url_for("admin.dashboard"))
            return f(*args, **kwargs)

        return decorated_function

    return decorator
```

9.3. Insert Records

All admin pages perform a similar method of processing data. This documentation only includes the processing of Books.

The below function is used to render the page where the admin can add new books or videos.

website/admin.py

```python
# New Book Page
@admin.route("/books/new/", methods=["GET", "POST"])
def new_book():
    # Initialize the form
    form = NewBookForm()

    # Validate the form on submit
    if form.validate_on_submit():
        data = form.data
        data.pop("book_id")
        data.pop("csrf_token")
        data.pop("submit")
        if data["type"] == "book":
            # Upload book to cloud
            data["book"] = upload_pdf_to_cloud(data["book"])
        data["category_id"] = ObjectId(data["category_id"])
        data["author"] = data["author"].split(",")
        # Upload cover to cloud
        data["cover"] = upload_image_to_cloud(data["cover"], data["title"])
        data["downloadable"] = eval(data["downloadable"])
        data["view_count"] = 0
        data["download_count"] = 0
        # Insert record to the database
        db.Books.insert_one(data)
        flash("Successfully added a new book.")
        return redirect(url_for("admin.books"))

    return render_template("admin/manage-books.html", form=form)
```

The below form is used to validate book details.

website/admin.py

```python
# Books Form - Used to validate books and videos when updating them
class BooksForm(FlaskForm):
    # Book Id
    book_id = StringField("Book Id", render_kw={"disabled": True})
    # Type of content - Book or Video
    type = SelectField( "Type", choices=[("book", "Book"), ("video", "Video")],
        validators=[DataRequired(message="Material type is required.")])
    # Category Id
    category_id = SelectField( "Category", validators=[DataRequired("Category is required.")])
    # Title of the book or Video
    title = StringField("Title", validators=[DataRequired("Title is required.")])
    # Publish year of the book or video
    year = IntegerField( "Publish Year",
        validators=[DataRequired("Year is required."),NumberRange(min=1000,
                # Cannot be greater than current year
                max=datetime.now().year, message="Year cannot be greater than current year.")])
    # Author name
    author = StringField("Author(s) (split by comma if more than one)",
        validators=[DataRequired("Author is required.")])
    # Publisher name
    publisher = StringField("Publisher", validators=[DataRequired("Publisher is required.")])
    # Book/Video summary or description
    summary = TextAreaField("Summary",
        validators=[DataRequired("Summary is required."),
            Length(
                # Maximum of 1000 characters
                min=10, max=1000, message="Summary must be 10 to 1000 characters long.")])
    # Book cover or video thumbnail
    cover = FileField("Cover", validators=file_validators)
    # Book - PDF format
    book = FileField("Book (PDF)", validators=file_validators)
    # Video - URL string
    video = StringField("Video (YouTube Embed URL)")
    # Downloadable - True or False
    downloadable = SelectField("Downloadable", choices=[(False, "False"), (True, "True")],
        validators=[DataRequired(message="Downloadable status is required.")])
    # Submit button
    submit = SubmitField("Save Changes")

    # Function to show a list of categories
    def is_submitted(self):
        self.category_id.choices = [(c["_id"], c["category"]) for c in db.Categories.find({})]
        return super().is_submitted()
```

9.4. Update Records

The below function is used to update book details.

website/admin.py

```python
# Page to edit books and videos
@admin.route("/books/manage/<book_id>", methods=["GET", "POST"])
def manage_book(book_id):
    book = db.Books.find_one({"_id": ObjectId(book_id)})
    form = BooksForm(
        book_id=book["_id"],
        title=book["title"],
        type=book["type"],
        category_id=book["category_id"],
        year=book["year"],
        cover=book["cover"],
        author=",".join(book["author"]),
        publisher=book["publisher"],
        summary=book["summary"],
        downloadable=book["downloadable"],
        book=book["book"] if "book" in book else "",
        video=book["video"] if "video" in book else "",
    )

    @authorize_roles("admin")
    def update_book():
        data = form.data
        data.pop("book_id")
        data.pop("csrf_token")
        data.pop("submit")
        data["category_id"] = ObjectId(data["category_id"])
        data["author"] = data["author"].split(",")
        data["downloadable"] = eval(data["downloadable"])
        if data["book"] != book["book"]:
            data["book"] = upload_pdf_to_cloud(data["book"])
            form.book.data = data["book"]
        if data["cover"] != book["cover"]:
            data["cover"] = upload_image_to_cloud(data["cover"], data["title"])
            form.cover.data = data["cover"]
        db.Books.update_one(book, {"$set": data})
        flash("Details updated successfully!")

    if form.is_submitted():
        form.book_id.data = book["_id"]

    if form.validate_on_submit():
        update_book()

    return render_template("admin/manage-books.html", book=book, form=form)
```

9.5. Delete Records

The below function is used to delete a document from the database.

website/admin.py

```python
# API to delete an item from the database
@admin.route("/database/<collection>/delete/<_id>", methods=["POST"])
@authorize_roles("admin")
def delete_db_record(collection: str, _id: ObjectId):
    if not (collection == "Users" and _id == "65512270765d2c3b115fa64e"):
        db[collection].delete_one({"_id": ObjectId(_id)})
        flash(f"Successfully deleted entry from the {collection} collection.")
    return redirect(url_for(f"admin.{collection.lower()}"))
```

9.6. View Records

The below function is used to retrieve data from the database and send them to admin dashboard.

website/admin.py

```python
@admin.route("/get-books", methods=["POST"])
def get_books():
    params = search_params("title")
    data = get_records("Books", params)
    return data
```

Datatables server-side rendering feature is used to pull the data from the above URL and display them in a tabular format to the admin.

References

Anon., n.d. *How to Set Up Flask with MongoDB.* [Online]

Available at: https://www.mongodb.com/compatibility/setting-up-flask-with-mongodb

[Accessed 2023].

Lord, D., 2023. *Flask Documentation.* [Online]

Available at: https://flask.palletsprojects.com/en/3.0.x/

Otto, M., 2023. *Bootstrap.* [Online]

Available at: https://getbootstrap.com/docs/5.3/