

1.2. Modèle logique de données :

Ce modèle conceptuel nous a aidé à établir un modèle logique de données (MLD) est la représentation des données d'un système d'information. Les données sont représentées en prenant en compte le modèle technologique qui sera utilisée pour leur gestion :

- Adhérent (#Id_adh, #id_ag Nom_adh, Prénom_adh, Adresse_adh, Nbr_pts, Type_adh, Date_nss, Montant_adh, Mdp_adh, Date_adh)
- Fiche emprunt (#Id_fiche_emprunt, #id_adh, Nom_emp, Prnm_emp, Date emp, Doc emp, Date_limite_emp)
- Commande (#Id_com, #num_four, #id_ag, Doc emp, Prix u, Quantite)
- Fournisseur (#num_four, Raison_social)
- Revendeur (#id_rev, Nom_rev, Tel_rev, Adresse_rev)
- Contrat (#id_rev, #num_four, Date_sign, Date_fin)
- Agent (#Id_ag, Nom_ag, Prnm_ag, Mdp_ag)
- Reserve (#id_adh, #Code_doc, Date_reservation)
- Famille (#code_famille, Type_fam, Nbre_doc)
- Emprunt (#code_doc, #id_adh, Nbre_docs_emp, Duree_de_l'emp)
- Chef d'établissement (#id_chef, #id_ag Nom_chef, Prénom_chef)
- Fiche problème (#code_fiche_prob, #id_adh, #id_ag, Motif_prob)
- Demande de prêt (#code_demande_pret, #id_adh, Date_demande_pret,)
- Documents (#code_doc, #code_famille Titre, Auteur, Année_de_sortie, Salle, Rayon, Type_de_doc)

2. La réalisation :

2.1. MySQL :

Afin de réaliser le projet et de lui faire passer d'un modèle conceptuel vers un modèle physique usuelle dans notre vie on a fait appel à plusieurs méthode, technique est des langage de programmation tout d'abord **MySQL** qui est un système de gestion de base de données relationnelle (SGBDR) largement utilisé. Il stocke les données dans des tables organisées de manière relationnelle et utilise le langage SQL pour interagir avec la base de données. Les applications et les utilisateurs se connectent à un serveur **MySQL** pour accéder aux données. **MySQL** utilise des index pour

optimiser les recherches, prend en charge les transactions pour garantir l'intégrité des données et dispose d'un optimiseur de requêtes pour améliorer les performances. Il offre également des fonctionnalités de sécurité et de sauvegarde pour protéger les données et prévenir la perte de données.

Grace a **MySQL Workbench** et après a voire réussir a se connecter à notre serveur- un serveur MySQL est un logiciel qui gère et fournit un accès à une base de données MySQL, permettant le stockage et la gestion de données- on a vas commencer par la création de notre base de donner d'abord et après on commence a faire la création des tables qui vont contenir toute les informations qu'on aura besoin dans notre projet grâce à l'ensemble des requêtes suivante :

2.Les requêtes :

```
CREATE TABLE inscription (  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  email VARCHAR(255) NOT NULL,  
  prenom VARCHAR(10) NOT NULL,  
  nom VARCHAR(10) NOT NULL,  
  password VARCHAR(255) NOT NULL,  
  rolle ENUM('student', 'teacher') NOT NULL  
);
```

```
CREATE TABLE document (  
  bid INT PRIMARY KEY,  
  titre VARCHAR(255),  
  TYPE ENUM('CD', 'Thèses', 'Article', 'Livres'),  
  ISBN VARCHAR(20),
```

```
Place VARCHAR(50),
Auteur VARCHAR(255),
Date_sortie YEAR,
Numero_copie INT,
    Empruntable BOOLEAN
);
```

```
CREATE TABLE borrowing (
    emprunt_id INT AUTO_INCREMENT PRIMARY KEY,
    book_id INT,
    person_id INT,
    periode VARCHAR(20),
    date_debut DATE,
    date_fin DATE,
    copies INT,
    FOREIGN KEY (book_id) REFERENCES document(bid),
    FOREIGN KEY (person_id) REFERENCES inscription(id)
);
```

```
INSERT INTO document(bid, titre, TYPE, ISBN, Place, Auteur, Date_sortie,
Numero_copie, Empruntable)
VALUES (2, 'titre2', 'Livres', '123967', 'Salle A1', 'Auteu1', '2022', 4, FALSE);
```

```
ALTER TABLE borrowing
ADD COLUMN posi ENUM('Active', 'Returned');
```

```
CREATE TABLE tempoborrowing (  
    emprunt_id INT AUTO_INCREMENT PRIMARY KEY,  
    book_id INT,  
    person_id INT,  
    periode VARCHAR(20),  
    date_debut DATE,  
    date_fin DATE,  
    copies INT,  
    FOREIGN KEY (book_id) REFERENCES document(bid),  
    FOREIGN KEY (person_id) REFERENCES inscription(id)  
);
```

```
CREATE TABLE deductions (  
    deduction_id INT AUTO_INCREMENT PRIMARY KEY,  
    person_id INT,  
    deduction_date DATE,  
    FOREIGN KEY (person_id) REFERENCES inscription (id)  
);
```

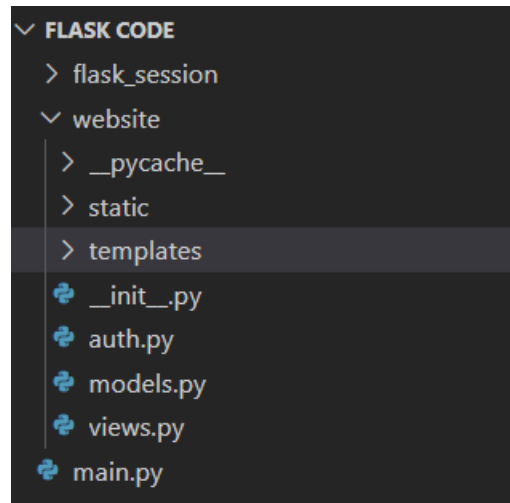
```
CREATE TABLE commande (  
    commande_id INT AUTO_INCREMENT PRIMARY KEY,  
    fournisseur VARCHAR(25),  
    commande_date DATE,  
    prix_totale INT  
);
```

```
select * from inscription;  
select * from document;  
select * from deductions;  
select * from borrowing;  
select * from tempoborrowing;  
select * from commande;
```

grâce a ces requête on a pu créer l'ensemble des tables qu'on vas utiliser lors de notre projet et lors de l'implémentation des autres langages a fin de crée une interface graphique qui vas aidez les différent utilisateurs a mieux utiliser le programme et même de concevoir et respecter les diffèrent mesure noter dans le cahier de charge, pour cela on a décidé d'avoir créé une application web on utilisons des technologie comme html, css pour le front end et du python pour la backend, notons bien sur un tout petit peu du javascript a fin d'établir des petite options dans nos page web et des différentes Framework comme Jinja -Jinja est un moteur de Template Python populaire pour générer des pages HTML dynamiques à partir de modèles prédéfinis - et Bootstrap -un Framework open-source largement utilisé pour le développement web. Il fournit un ensemble d'outils, de composants et de styles prédéfinis pour faciliter la création d'interfaces utilisateur attrayantes et réactives-

2.L'environement de développement

La première étape pour réussir notre application Web c'est de créer le dossier qui vas contenir l'ensemble des codes et fonctions qui gère cette dernière dans notre cas c'était le dossier nomme Flask Code :



Comme vous pouvez voir dans notre dossier on trouve un autre dossier et un fichier python qui s'appelle

- main.py : le fichier main.py est le point d'entrée principal de notre application Flask. Il initialise et configure l'application, définit les routes et les fonctions de vue, et gère la logique de traitement des requêtes et des réponses le fichier main.py a comme code :

```
• from website import create_app
•
• app = create_app()
•
• if __name__ == '__main__':
•     app.run(debug=True)
•
```

Ce code initialise et démarre votre application Flask en appelant la fonction `create_app()` pour créer l'objet Flask, puis en utilisant `app.run(debug=True)` pour démarrer le serveur de développement.

- Le code dans main.py c'est fait exporter à partir du fichier `__init__.py` dans le dossier `website` pour cela on trouve `from website import create_app` `website` ici désigne le dossier où il y a notre application et `create_app` et le nom de notre fonction qu'on a créée dans le fichier `__init__.py` au sein du dossier `website` :

```
• from flask import Flask
• from flask_sqlalchemy import SQLAlchemy
• from flask_session import Session
•
• def create_app():
•     app = Flask(__name__)
```

```

•     app.config['SECRET_KEY'] = 'password'
•     app.config['SESSION_TYPE'] = 'filesystem'
•     app.config['SESSION_COOKIE_SECURE'] = True
•     app.config['SESSION_COOKIE_HTTPONLY'] = True
•     Session(app)
•
•     from .views import views
•     from .auth import auth
•
•     app.register_blueprint(views, url_prefix='/')
•     app.register_blueprint(auth, url_prefix='/')
•
•     return app
•

```

Voici le code qui se trouve au sein de notre `__init__.py` afin de définir la fonction `create_app()` :

- `from flask import Flask`: Cette ligne importe la classe Flask du module Flask, qui est un framework web en Python.
- `from flask_session import Session`: Cette ligne importe la classe Session du module flask_session. Flask-Session est une extension Flask qui permet de gérer les sessions utilisateur.
- `app = Flask(__name__)`: Cette ligne crée une instance de la classe Flask et l'assigne à la variable `app`. L'argument `__name__` est utilisé pour définir le nom de l'application.
- `app.config['SECRET_KEY'] = 'pass'`: Cette ligne définit la clé secrète de l'application Flask, utilisée pour la sécurité des sessions et des cookies.
- `app.config['SESSION_TYPE'] = 'filesystem'`: Cette ligne définit le type de stockage des sessions, en l'occurrence ici, sur le système de fichiers.
- `app.config['SESSION_COOKIE_SECURE'] = True`: Cette ligne active la sécurité des cookies de session en spécifiant que le cookie doit être transmis uniquement via une connexion sécurisée (HTTPS).
- `app.config['SESSION_COOKIE_HTTPONLY'] = True`: Cette ligne indique que le cookie de session doit être accessible uniquement par le serveur, pas par JavaScript.
- `Session(app)`: Cette ligne initialise l'extension Flask-Session avec l'application Flask créée.
- `from .views import views`: Cette ligne importe le blueprint views depuis le module views situé dans le même package (ou dossier) que le fichier actuel.

- `from .auth import auth`: Cette ligne importe le blueprint `auth` depuis le module `auth` situé dans le même package (ou dossier) que le fichier actuel.
- `app.register_blueprint(views, url_prefix='/')`: Cette ligne enregistre le blueprint `views` avec l'application Flask, en spécifiant que les routes de ce blueprint seront préfixées par `'/'`.

Blueprint est un mécanisme de Flask permettant de structurer et d'organiser une application en modules réutilisables. Il offre une approche modulaire pour la définition des routes, des modèles, des vues et des fichiers statiques, facilitant ainsi le développement et la gestion de projets Flask plus complexes.

3. Connexion avec la base de données

Maintenant et après avoir créé nos fichiers de démarrage on doit commencer par la création des routes que l'utilisateur va utiliser pour naviguer dans notre site web. Pour cela on a créé des pages `views.py` et `auth.py` qui vont contenir toute la fonctionnalité nécessaire. La page `auth.py` va contenir tout ce qui est nécessaire pour ce que l'on a besoin d'une authentification pour ce que l'on doit tout d'abord connecter à notre base de données MySQL grâce à la commande suivante dans notre page `auth.py`

```
#####
import mysql.connector
from datetime import datetime

db = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="123456",
    database="testdatabase")
#####
```

Maintenant est grâce à la bibliothèque `mysql.connector` on a pu définir une variable avec le nom `db` qui contient la somme des informations nécessaires avec le nom de la base de données qu'on a `testdatabase` pour pouvoir manipuler et exécuter notre base de données.

Après on commence à définir toutes les routes nécessaires comme le login et le signup page comme suit :

```
@auth.route('/login', methods=['GET', 'POST'])
def login():
```



```

user = 'hamza'
if request.method == 'POST':
    email = request.form.get('email')
    password = request.form.get('password')
    cursor = db.cursor()
    query = "SELECT * FROM inscription WHERE email = %s AND password = %s"
    values = (email, password)
    cursor.execute(query, values)
    user = cursor.fetchone()
    if user :
        user_id = user[0]
        session['logged_in'] = True
        session['user_id'] = user_id
        flash('Login successful!',category='success')
        return redirect(url_for('auth.home'))
    else:
        flash('Invalid email or password.', category='error')
else :
    user = None
return render_template("login.html", user=current_user)

```

@auth.route('/login', methods=['GET', 'POST']) est un décorateur qui associe cette fonction de vue à la route '/login' dans le Blueprint 'auth' et spécifie les méthodes HTTP autorisées pour cette route (GET et POST dans ce cas).

def login(): est la fonction de vue elle-même. Elle est exécutée lorsque la route '/login' est accédée par un utilisateur.

À l'intérieur de la fonction de vue, on commence par initialiser une variable user avec la valeur 'hamza'. Cela peut être une valeur par défaut pour la variable utilisateur.

Ensuite, on vérifie si la méthode de la requête est POST. Cela signifie qu'un formulaire a été soumis par l'utilisateur pour se connecter.

Si la méthode est POST, on récupère les données du formulaire à l'aide de request.form.get() pour obtenir les valeurs de l'email et du mot de passe soumis par l'utilisateur.

Ensuite, une requête SQL est exécutée pour vérifier si les informations d'identification fournies (email et mot de passe) correspondent à un enregistrement dans la table "inscription" de la base de données.

Si un utilisateur correspondant est trouvé, son identifiant est récupéré et stocké dans la session Flask en utilisant `session['logged_in']` et `session['user_id']`. Un message de réussite est affiché à l'utilisateur en utilisant `flash()`.

En cas d'échec de connexion, un message d'erreur est affiché à l'utilisateur.

Si la méthode de la requête est GET (lorsque l'utilisateur accède simplement à la page de connexion sans soumettre de formulaire), la variable `user` reste `None`.

Enfin, la fonction de vue renvoie le modèle `'login.html'` en utilisant `render_template()`, en passant les variables nécessaires au modèle.

4. les routes et commande python :

La même chose on a pour toutes les autres fonctionnalités de l'application comme le sign-up page ou on trouve

```
@auth.route('/sign-up',methods=['GET', 'POST'])
def sign_up():
    user = 'hamza'
    if request.method == 'POST':
        email = request.form.get('email')
        firstName = request.form.get('firstName')
        lastName = request.form.get('lastName')
        password1 = request.form.get('password1')
        password2 = request.form.get('password2')
        role = request.form.get('role')

        if len(email) < 4:
            flash ('Email must be greater than 4 characters.', category='error')
        elif len(firstName) < 2:
            flash ('First name must be greater than 2 characters.',
category='error')
        elif len(lastName) < 2:
            flash ('First name must be greater than 2 characters.',
category='error')
        elif password1 != password2:
            flash ('Passwords dont match.', category='error')
        elif len(password1)<5:
            flash ('Passwords must be greater than 5.', category='error')
        elif role == None:
            flash ('Choose your role', category='error')
        else:
            cursor = db.cursor()
```

```

        query = "INSERT INTO inscription (email, prenom, nom, password,
roll1) VALUES (%s, %s, %s, %s, %s)"
        values = (email, firstName, lastName, password1, role)
        cursor.execute(query, values)
        db.commit()
        flash('Account Created !', category='sucess')

    return render_template("sign_up.html", user=current_user)

```

Le code vérifie et insère les données dans notre base de données pour gérer les inscriptions ou les adhérents et stocker leurs informations pour les réutiliser après.

5. Logout

```

@auth.route('/logout')
def logout():
    session.pop('logged_in', None)
    session.pop('user_id', None)
    return redirect(url_for('auth.login'))

```

Le logout détermine les variables `user_id` et `logged_in` comme nul et redirige vers la page login

6. Réservation :

La réservation est la partie la plus compliquée dans notre suite au vu des différentes consignes qu'il faut suivre afin de réaliser une réservation car il faut vérifier le nombre de réservations pour les adhérents et les diviser entre étudiant et professeur et tout cela en mettant en garde leurs nombres de scores et enregistrer leur demande et la rendre sous une forme PDF pour y avoir un reçu de réservations ou demande pour cela on a utilisé le code suivant :

```

@auth.route('/reservations', methods=['GET', 'POST'])
def reservations():
    cursor = db.cursor()

    if request.method == 'POST':
        titre = request.form.get('book-title')
        period = request.form.get('duration')
        nom = request.form.get('Nom')
        return_date = request.form.get('return_date')

        cursor.execute("SELECT id, roll1, score, regularization_request FROM
inscription WHERE nom = %s", (nom,))
        person_row = cursor.fetchone()

```

```

    if person_row is not None:
        person_id, rol11, person_points, regularization_request = person_row
    else:
        flash('Person not found.', 'error')
        return redirect(url_for('auth.reservations'))

    cursor.execute("SELECT bid, Emprutable FROM document WHERE titre = %s",
(titre,))
    document_row = cursor.fetchone()

    if document_row is not None:
        book_id, emprutable = document_row
    else:
        flash('Document not found.', 'error')
        return redirect(url_for('auth.reservations'))

    if emprutable == 'non':
        flash("This document cannot be reserved.", 'error')
        return redirect(url_for('auth.reservations'))

    if rol11 == 'student':
        cursor.execute("SELECT COUNT(*) FROM borrowing WHERE person_id = %s
AND posi = 'Active'", (person_id,))
        active_loans = cursor.fetchone()[0]
        cursor.execute("SELECT COUNT(*) FROM tempoborrowing WHERE person_id =
%s", (person_id,))
        temp_reservations = cursor.fetchone()[0]
        max_reservations = 2

        if active_loans + temp_reservations >= max_reservations:
            flash("You have reached the maximum reservation limit or have
active loans.", 'error')
            return redirect(url_for('auth.reservations'))

    elif rol11 == 'teacher':
        cursor.execute("SELECT COUNT(*) FROM borrowing WHERE person_id = %s
AND posi = 'Active'", (person_id,))
        active_loans = cursor.fetchone()[0]
        cursor.execute("SELECT COUNT(*) FROM tempoborrowing WHERE person_id =
%s", (person_id,))
        temp_reservations = cursor.fetchone()[0]
        max_reservations = 5

        if active_loans + temp_reservations >= max_reservations:

```

```

        flash("You have reached the maximum reservation limit or have
active loans.", 'error')
        return redirect(url_for('auth.reservations'))

    if person_points <= 0:
        if regularization_request is not None:
            flash("Your points are insufficient for making a reservation.",
'error')
            return render_template('regularization.html', user=current_user,
nom=nom)
        else:
            return render_template('regularization.html', user=current_user,
nom=nom)

    query = "INSERT INTO tempoborrowing (book_id, person_id, periode,
date_debut, date_fin) VALUES (%s, %s, %s, %s, %s)"
    values = (book_id, person_id, period, datetime.now(), return_date)
    cursor.execute(query, values)
    db.commit()

    flash('Reservation made successfully!', 'success')

    # Retrieve the reservation data for the PDF template
    reservation_info = [(cursor.lastrowid, book_id, person_id, period,
datetime.now())]

    # Render the template with the reservation information
    rendered_template = render_template(
        'pdf_template.html',
        reservation_info=reservation_info,
        configuration=config
    )

    # Generate the PDF file from the rendered template
    pdfkit.from_string(rendered_template, 'reservation_receipt.pdf',
configuration=config)

    cursor.execute("SELECT * FROM tempoborrowing")
    reservations = cursor.fetchall()

    return render_template('reservations.html', user=current_user,
reservations=reservations)

```

7. Home Page :

La page home et la page ou l'adhérent vas voir les livre disponible, les demandes actuelles et l'historique de c'est demande

```
@auth.route('/home')
def home():
    if 'user_id' in session:
        user_id = session['user_id']
        cursor = db.cursor()
        query = "SELECT prenom, nom FROM inscription WHERE id = %s"
        values = (user_id,)
        cursor.execute(query, values)
        user_data = cursor.fetchone()
        if user_data:
            first_name, last_name = user_data
            full_name = f"{first_name} {last_name}"

            cursor.execute("SELECT bid, titre, TYPE, ISBN, Place, Auteur,
Date_sortie, Numero_copie, Empruntable FROM document")
            documents = cursor.fetchall()

            cursor.execute("SELECT t.book_id, d.titre, t.periode FROM
tempoborrowing t JOIN document d ON t.book_id = d.bid WHERE t.person_id = %s",
(user_id,))
            demands = cursor.fetchall()

            cursor.execute("SELECT l.book_id, d.titre, p.nom, p.prenom,
l.periode, l.statu_s, l.posi FROM borrowing l JOIN document d ON l.book_id =
d.bid JOIN inscription p ON l.person_id = p.id WHERE l.person_id = %s",
(user_id,))
            loans = cursor.fetchall()

            return render_template("home.html", user=full_name,
documents=documents, demands=demands, loans=loans)

        return redirect(url_for('auth.login'))
```

pour cela on affecte les variables documents, demandes et loans pour prendre les valeurs du tableau a fin de les envoyer vers la page html et les rendre visuels

8. Page Admin :

La page admin a besoin de

```
@auth.route('/admin')
def admin():
```

```

cursor = db.cursor()
cursor.execute("SELECT * FROM inscription")
inscriptions = cursor.fetchall()

crs = db.cursor()
crs.execute("SELECT * FROM document")
documents = crs.fetchall()

cursor.execute("SELECT * FROM tempoborrowing")
reservations = cursor.fetchall()

return render_template("admin.html", inscriptions=inscriptions, documents =
documents, reservations=reservations)

```

Pour gérer toute cet dernière.

8.1 suppressions autant qu'un admin :

Il faut établir une fonction pour établir la suppression :

```

@auth.route('/delete_person/<int:person_id>', methods=['POST'])
def delete_person(person_id):
    cursor = db.cursor()

    cursor.execute("DELETE FROM borrowing WHERE person_id = %s", (person_id,))

    cursor.execute("DELETE FROM inscription WHERE id = %s", (person_id,))

    db.commit()

    flash('Person deleted successfully!', 'success')

    return redirect(url_for('auth.admin'))

```

La même chose pour établir l'ajout d'un livre :

```

@auth.route('/Ajoutlivre', methods=['GET', 'POST'])
def ajout():
    if request.method == 'POST':
        title = request.form.get('title')
        type = request.form.get('type')
        isbn = request.form.get('isbn')
        place = request.form.get('place')

```

```

author = request.form.get('author')
release_year = request.form.get('releaseYear')
num_copies = request.form.get('numCopies')
borrowable = request.form.get('borrowable')

cursor = db.cursor()

cursor.execute("SELECT MAX(bid) FROM document")
max_bid = cursor.fetchone()[0]

if max_bid is None:
    max_bid = 0

bid = max_bid + 1

query = "INSERT INTO document (bid, titre, TYPE, ISBN, Place, Auteur,
Date_sortie, Numero_copie, Empruntable) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)"
values = (bid, title, type, isbn, place, author, release_year,
num_copies, borrowable)
cursor.execute(query, values)

db.commit()

flash('Document added successfully!', 'success')

return render_template('add_book.html')

```

et la suppression d'un livre

```

@auth.route('/delete_document', methods=['POST'])
def delete_document():
    titre = request.form.get('titre')
    cursor = db.cursor()

    # Perform the deletion logic
    cursor.execute("DELETE FROM document WHERE bid = %s", (titre,))
    db.commit()

```



```
flash('Document deleted successfully!', 'success')

return redirect(url_for('auth.admin'))
```

Maintenant au niveau de l'admin quand il y a une demande pour emprunter un livre ou non l'admin a le choix soit d'accepter ou non et pour y faire il a les fonctions suivantes :

```
@auth.route('/approved', methods=['POST'])
def approved():
    demande_id = request.form.get('demande_id')
    demande_id1 = request.form.get('demande_id1')

    if demande_id is not None:
        demande_id = int(demande_id)
        # Get the details of the demand from the database using the demande_id
        cursor = db.cursor()
        cursor.execute("SELECT book_id, person_id, periode FROM tempoborrowing
WHERE emprunt_id = %s", (demande_id,))
        demand = cursor.fetchone()

        if demand:
            book_id = demand[0]
            person_id = demand[1]
            periode = demand[2]
            crsr = db.cursor()
            crsr.execute("SELECT prenom, nom FROM inscription WHERE id = %s",
(person_id,))
            perso = crsr.fetchone()
            nom = perso[0]
            prenom = perso[1]

            # Insert the approved demand into the borrowing table
            cursor.execute("INSERT INTO borrowing (book_id, person_id, periode,
statu_s, nom, prenom) VALUES (%s, %s, %s, 'approved', %s, %s)", (book_id,
person_id, periode, nom, prenom))
            cursor.execute("DELETE FROM tempoborrowing WHERE emprunt_id = %s",
(demande_id,))
            db.commit()
            db.commit()
            flash('Demand approved successfully!', 'success')
        else:
            flash('Demand not found!', 'error')
```

```

elif demande_id1 is not None:
    demande_id1 = int(demande_id1)
    # Get the details of the demand from the database using the demande_id
    cursor = db.cursor()
    cursor.execute("SELECT book_id, person_id, periode FROM tempoborrowing
WHERE emprunt_id = %s", (demande_id1,))
    demand = cursor.fetchone()

    if demand:
        book_id = demand[0]
        person_id = demand[1]
        periode = demand[2]
        crsr = db.cursor()
        crsr.execute("SELECT prenom, nom FROM inscription WHERE id = %s",
(person_id,))
        perso = crsr.fetchone()
        nom = perso[0]
        prenom = perso[1]

        # Insert the not approved demand into the borrowing table
        cursor.execute("INSERT INTO borrowing (book_id, person_id, periode,
statu_s, nom, prenom) VALUES (%s, %s, %s, 'not approved', %s, %s)", (book_id,
person_id, periode, nom, prenom))
        cursor.execute("DELETE FROM tempoborrowing WHERE emprunt_id = %s",
(demande_id1,))
        db.commit()
        flash('Demand not approved!', 'success')

    return redirect(url_for('auth.admin'))

```

ici l'admin si il clique sur approuver il vas supprimer le documents de la table temporaires des emprunts et il vas transférer toute les informations vers la table finale des emprunt soit avec une valeur de approuvé ou bien **not approved**.

9. Acceptation de la demande d'emprunte :

Maintenant lors de l'acceptation de la demande chez l'admin l'adhérant peut retourner un livre pour cela on a créer une fonction pour être capable à retourner les livres :

```

@auth.route('/return_book/<int:loan_id>', methods=['POST'])
def return_book(loan_id):
    loan_id = request.form.get('loan_id')

```

```

cursor = db.cursor()

if loan_id is not None:
    loan_id = int(loan_id)
    cursor.execute("SELECT person_id, date_fin FROM borrowing WHERE book_id = %s", (loan_id,))
    loan_data = cursor.fetchone()

    # Consume any unread results from the previous query
    cursor.fetchall()

    if loan_data:
        person_id, due_date = loan_data
        today = date.today()

        if due_date is not None and today > due_date:
            # Calculate the number of days late
            days_late = (today - due_date).days

            # Deduct points based on the number of days late
            points_to_deduct = days_late // 7 # Deduct 1 point for every 7
days_late
            if points_to_deduct > 0:
                cursor.execute("UPDATE inscription SET points = points - %s
WHERE id = %s", (points_to_deduct, person_id))
                db.commit()
                flash(f'{points_to_deduct} point(s) deducted for late
return.', 'info')

            cursor.execute(" UPDATE borrowing SET posi = %s WHERE book_id = %s",
('Returned',loan_id,))

        db.commit()
        flash('Book returned successfully!', 'success')

    if loan_data is None:
        # The loan record does not exist
        flash('Failed to return the book. Please try again.', 'error')

cursor.close()
return redirect(url_for('auth.home'))

```

Et cela marche avec le code complet ou chaque option a sa propre fonction :

10. Le code complet :

```
from flask import Blueprint, render_template, request, flash, session, redirect, url_for
from flask_login import logout_user, current_user
from datetime import date, timedelta
from flask import Flask
from flask_login import LoginManager
import jinja2
import pdfkit

path_wkhtmltopdf = b'C:\Program Files\wkhtmltopdf\bin\wkhtmltopdf.exe'
config = pdfkit.configuration(wkhtmltopdf=path_wkhtmltopdf)

#####
import mysql.connector
from datetime import datetime

db = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="123456",
    database="testdatabase")

#####

auth = Blueprint('auth', __name__)

@auth.route('/login', methods=['GET', 'POST'])
def login():
    user = 'hamza'
    if request.method == 'POST':
        email = request.form.get('email')
        password = request.form.get('password')
        cursor = db.cursor()
        query = "SELECT * FROM inscription WHERE email = %s AND password = %s"
        values = (email, password)
        cursor.execute(query, values)
        user = cursor.fetchone()
        if user :
```

```

        user_id = user[0]
        session['logged_in'] = True
        session['user_id'] = user_id
        flash('Login successful!',category='success')
        return redirect(url_for('auth.home'))
    else:
        flash('Invalid email or password.', category='error')
else :
    user = None
    return render_template("login.html", user=current_user)

@auth.route('/logout')
def logout():
    session.pop('logged_in', None)
    session.pop('user_id', None)
    return redirect(url_for('auth.login'))

@auth.route('/sign-up',methods=['GET', 'POST'])
def sign_up():
    user = 'hamza'
    if request.method == 'POST':
        email = request.form.get('email')
        firstName = request.form.get('firstName')
        lastName = request.form.get('lastName')
        password1 = request.form.get('password1')
        password2 = request.form.get('password2')
        role = request.form.get('role')

        if len(email) < 4:
            flash ('Email must be greater than 4 characters.', category='error')
        elif len(firstName) < 2:
            flash ('First name must be greater than 2 characters.',
category='error')
        elif len(lastName) < 2:
            flash ('First name must be greater than 2 characters.',
category='error')
        elif password1 != password2:
            flash ('Passwords dont match.', category='error')
        elif len(password1)<5:
            flash ('Passwords must be greater than 5.', category='error')
        elif role == None:
            flash ('Choose your role', category='error')
        else:
            cursor = db.cursor()

```

```

        query = "INSERT INTO inscription (email, prenom, nom, password,
roll1) VALUES (%s, %s, %s, %s, %s)"
        values = (email, firstName, lastName, password1, role)
        cursor.execute(query, values)
        db.commit()
        flash('Account Created !', category='sucess')

    return render_template("sign_up.html", user=current_user)

@auth.route('/home')
def home():
    if 'user_id' in session:
        user_id = session['user_id']
        cursor = db.cursor()
        query = "SELECT prenom, nom FROM inscription WHERE id = %s"
        values = (user_id,)
        cursor.execute(query, values)
        user_data = cursor.fetchone()
        if user_data:
            first_name, last_name = user_data
            full_name = f"{first_name} {last_name}"

            cursor.execute("SELECT bid, titre, TYPE, ISBN, Place, Auteur,
Date_sortie, Numero_copie, Empruntable FROM document")
            documents = cursor.fetchall()

            cursor.execute("SELECT t.book_id, d.titre, t.periode FROM
tempoborrowing t JOIN document d ON t.book_id = d.bid WHERE t.person_id = %s",
(user_id,))
            demands = cursor.fetchall()

            cursor.execute("SELECT l.book_id, d.titre, p.nom, p.prenom,
l.periode, l.statu_s, l.posi FROM borrowing l JOIN document d ON l.book_id =
d.bid JOIN inscription p ON l.person_id = p.id WHERE l.person_id = %s",
(user_id,))
            loans = cursor.fetchall()

            return render_template("home.html", user=full_name,
documents=documents, demands=demands, loans=loans)

    return redirect(url_for('auth.login'))

```

```

@auth.route('/reservations', methods=['GET', 'POST'])
def reservations():
    cursor = db.cursor()

    if request.method == 'POST':
        titre = request.form.get('book-title')
        period = request.form.get('duration')
        nom = request.form.get('Nom')
        return_date = request.form.get('return_date')

        cursor.execute("SELECT id, rol11, score, regularization_request FROM
inscription WHERE nom = %s", (nom,))
        person_row = cursor.fetchone()

        if person_row is not None:
            person_id, rol11, person_points, regularization_request = person_row
        else:
            flash('Person not found.', 'error')
            return redirect(url_for('auth.reservations'))

        cursor.execute("SELECT bid, Empruntable FROM document WHERE titre = %s",
(titre,))
        document_row = cursor.fetchone()

        if document_row is not None:
            book_id, empruntable = document_row
        else:
            flash('Document not found.', 'error')
            return redirect(url_for('auth.reservations'))

        if empruntable == 'non':
            flash("This document cannot be reserved.", 'error')
            return redirect(url_for('auth.reservations'))

        if rol11 == 'student':
            cursor.execute("SELECT COUNT(*) FROM borrowing WHERE person_id = %s
AND posi = 'Active'", (person_id,))
            active_loans = cursor.fetchone()[0]
            cursor.execute("SELECT COUNT(*) FROM tempoborrowing WHERE person_id =
%s", (person_id,))

```

```

        temp_reservations = cursor.fetchone()[0]
        max_reservations = 2

        if active_loans + temp_reservations >= max_reservations:
            flash("You have reached the maximum reservation limit or have
active loans.", 'error')
            return redirect(url_for('auth.reservations'))

        elif rol11 == 'teacher':
            cursor.execute("SELECT COUNT(*) FROM borrowing WHERE person_id = %s
AND posi = 'Active'", (person_id,))
            active_loans = cursor.fetchone()[0]
            cursor.execute("SELECT COUNT(*) FROM tempoborrowing WHERE person_id =
%s", (person_id,))
            temp_reservations = cursor.fetchone()[0]
            max_reservations = 5

            if active_loans + temp_reservations >= max_reservations:
                flash("You have reached the maximum reservation limit or have
active loans.", 'error')
                return redirect(url_for('auth.reservations'))

        if person_points <= 0:
            if regularization_request is not None:
                flash("Your points are insufficient for making a reservation.",
'error')
                return render_template('regularization.html', user=current_user,
nom=nom)
            else:
                return render_template('regularization.html', user=current_user,
nom=nom)

        query = "INSERT INTO tempoborrowing (book_id, person_id, periode,
date_debut, date_fin) VALUES (%s, %s, %s, %s, %s)"
        values = (book_id, person_id, period, datetime.now(), return_date)
        cursor.execute(query, values)
        db.commit()

        flash('Reservation made successfully!', 'success')

        # Retrieve the reservation data for the PDF template
        reservation_info = [(cursor.lastrowid, book_id, person_id, period,
datetime.now())]

        # Render the template with the reservation information

```



```

        rendered_template = render_template(
            'pdf_template.html',
            reservation_info=reservation_info,
            configuration=config
        )

        # Generate the PDF file from the rendered template
        pdfkit.from_string(rendered_template, 'reservation_receipt.pdf',
configuration=config)

        cursor.execute("SELECT * FROM tempoborrowing")
        reservations = cursor.fetchall()

        return render_template('reservations.html', user=current_user,
reservations=reservations)

@auth.route('/admin')
def admin():
    cursor = db.cursor()
    cursor.execute("SELECT * FROM inscription")
    inscriptions = cursor.fetchall()

    crs = db.cursor()
    crs.execute("SELECT * FROM document")
    documents = crs.fetchall()

    cursor.execute("SELECT * FROM tempoborrowing")
    reservations = cursor.fetchall()

    return render_template("admin.html", inscriptions=inscriptions, documents =
documents, reservations=reservations)

@auth.route('/Test')
def show_test_table():
    cursor = db.cursor()
    cursor.execute('SELECT * FROM person')
    data = cursor.fetchall()
    columns = [column[0] for column in cursor.description]
    return render_template('test.html', data=data, columns=columns)

```

```

@auth.route('/delete_person/<int:person_id>', methods=['POST'])
def delete_person(person_id):
    cursor = db.cursor()

    cursor.execute("DELETE FROM borrowing WHERE person_id = %s", (person_id,))

    cursor.execute("DELETE FROM inscription WHERE id = %s", (person_id,))

    db.commit()

    flash('Person deleted successfully!', 'success')

    return redirect(url_for('auth.admin'))


@auth.route('/Ajoutlivre', methods=['GET', 'POST'])
def ajout():
    if request.method == 'POST':
        title = request.form.get('title')
        type = request.form.get('type')
        isbn = request.form.get('isbn')
        place = request.form.get('place')
        author = request.form.get('author')
        release_year = request.form.get('releaseYear')
        num_copies = request.form.get('numCopies')
        borrowable = request.form.get('borrowable')

        cursor = db.cursor()

        cursor.execute("SELECT MAX(bid) FROM document")
        max_bid = cursor.fetchone()[0]

        if max_bid is None:
            max_bid = 0

        bid = max_bid + 1

```

```

        query = "INSERT INTO document (bid, titre, TYPE, ISBN, Place, Auteur,
Date_sortie, Numero_copie, Emprutable) VALUES (%s, %s, %s, %s, %s, %s, %s, %s,
%s)"
        values = (bid, title, type, isbn, place, author, release_year,
num_copies, borrowable)
        cursor.execute(query, values)

        db.commit()

        flash('Document added successfully!', 'success')

    return render_template('add_book.html')

@auth.route('/delete_document', methods=['POST'])
def delete_document():
    titre = request.form.get('titre')
    cursor = db.cursor()

    # Perform the deletion logic
    cursor.execute("DELETE FROM document WHERE bid = %s", (titre,))
    db.commit()

    flash('Document deleted successfully!', 'success')

    return redirect(url_for('auth.admin'))

@auth.route('/approved', methods=['POST'])
def approved():
    demande_id = request.form.get('demande_id')
    demande_id1 = request.form.get('demande_id1')

    if demande_id is not None:
        demande_id = int(demande_id)
        # Get the details of the demand from the database using the demande_id
        cursor = db.cursor()
        cursor.execute("SELECT book_id, person_id, periode FROM tempoborrowing
WHERE emprunt_id = %s", (demande_id,))
        demand = cursor.fetchone()

        if demand:
            book_id = demand[0]
            person_id = demand[1]

```

```

        periode = demand[2]
        crsr = db.cursor()
        crsr.execute("SELECT prenom, nom FROM inscription WHERE id = %s",
(person_id,))
        perso = crsr.fetchone()
        nom = perso[0]
        prenom = perso[1]

        # Insert the approved demand into the borrowing table
        cursor.execute("INSERT INTO borrowing (book_id, person_id, periode,
statu_s, nom, prenom) VALUES (%s, %s, %s, 'approved', %s, %s)", (book_id,
person_id, periode, nom, prenom))
        cursor.execute("DELETE FROM tempoborrowing WHERE emprunt_id = %s",
(demande_id,))
        db.commit()
        db.commit()
        flash('Demand approved successfully!', 'success')
    else:
        flash('Demand not found!', 'error')

elif demande_id1 is not None:
    demande_id1 = int(demande_id1)
    # Get the details of the demand from the database using the demande_id
    cursor = db.cursor()
    cursor.execute("SELECT book_id, person_id, periode FROM tempoborrowing
WHERE emprunt_id = %s", (demande_id1,))
    demand = cursor.fetchone()

    if demand:
        book_id = demand[0]
        person_id = demand[1]
        periode = demand[2]
        crsr = db.cursor()
        crsr.execute("SELECT prenom, nom FROM inscription WHERE id = %s",
(person_id,))
        perso = crsr.fetchone()
        nom = perso[0]
        prenom = perso[1]

        # Insert the not approved demand into the borrowing table
        cursor.execute("INSERT INTO borrowing (book_id, person_id, periode,
statu_s, nom, prenom) VALUES (%s, %s, %s, 'not approved', %s, %s)", (book_id,
person_id, periode, nom, prenom))
        cursor.execute("DELETE FROM tempoborrowing WHERE emprunt_id = %s",
(demande_id1,))

```

```

        db.commit()
        flash('Demand not approved!', 'success')

    return redirect(url_for('auth.admin'))

@auth.route('/return_book/<int:loan_id>', methods=['POST'])
def return_book(loan_id):
    loan_id = request.form.get('loan_id')
    cursor = db.cursor()

    if loan_id is not None:
        loan_id = int(loan_id)
        cursor.execute("SELECT person_id, date_fin FROM borrowing WHERE book_id = %s", (loan_id,))
        loan_data = cursor.fetchone()

        # Consume any unread results from the previous query
        cursor.fetchall()

        if loan_data:
            person_id, due_date = loan_data
            today = date.today()

            if due_date is not None and today > due_date:
                # Calculate the number of days late
                days_late = (today - due_date).days

                # Deduct points based on the number of days late
                points_to_deduct = days_late // 7 # Deduct 1 point for every 7
days late

                if points_to_deduct > 0:
                    cursor.execute("UPDATE inscription SET points = points - %s
WHERE id = %s", (points_to_deduct, person_id))
                    db.commit()
                    flash(f'{points_to_deduct} point(s) deducted for late
return.', 'info')

                    cursor.execute(" UPDATE borrowing SET posi = %s WHERE book_id = %s",
('Returned', loan_id,))

```

```

        db.commit()
        flash('Book returned successfully!', 'success')

    if loan_data is None:
        # The loan record does not exist
        flash('Failed to return the book. Please try again.', 'error')

    cursor.close()
    return redirect(url_for('auth.home'))

@auth.route('/update_score/<int:person_id>', methods=['POST'])
def update_score(person_id):
    cursor = db.cursor()
    new_score = int(request.form.get('score'))

    # Update the score in the inscription table
    cursor.execute("UPDATE inscription SET score = %s WHERE id = %s", (new_score,
person_id))
    db.commit()

    flash('Score updated successfully!', 'success')
    return redirect(url_for('auth.admin'))

@auth.route('/submit_regularization', methods=['POST'])
def submit_regularization():
    cursor = db.cursor()

    if request.method == 'POST':
        # Get the user ID from the session or any other means
        user_id = session.get('user_id') # Modify this line according to your
session management

        # Get the regularization request form data
        motif1 = request.form.get('motif1')
        motif2 = request.form.get('motif2')

```

```

        motif3 = request.form.get('motif3')

        # Update the regularization_request column in the inscription table
        query = "UPDATE inscription SET regularization_request = %s WHERE id = %s"
        values = (f"{motif1}\n{motif2}\n{motif3}", user_id)
        cursor.execute(query, values)
        db.commit()

        flash('Regularization request submitted successfully!', 'success')

        return redirect(url_for('auth.reservations'))

@auth.route('/regularization', methods=['POST'])
def regularization():
    person_id = request.form.get('person_id')
    regularization_text = request.form.get('regularization_text')

    cursor = db.cursor()
    query = "UPDATE inscription SET regularization_request = %s WHERE id = %s"
    values = (regularization_text, person_id)
    cursor.execute(query, values)
    db.commit()

    flash('Regularization request submitted successfully!', 'success')
    return redirect(url_for('auth.reservations'))

@auth.route('/delete_regularization/<int:inscription_id>', methods=['POST'])
def delete_regularization(inscription_id):
    cursor = db.cursor()

    # Delete the regularization request from the database
    query = "UPDATE inscription SET regularization_request = NULL WHERE id = %s"
    cursor.execute(query, (inscription_id,))
    db.commit()

    flash('Regularization request deleted successfully!', 'success')
    return redirect(url_for('auth.admin'))

```

```

import pdfkit
from flask import render_template, make_response
import pdfkit

@auth.route('/generate_pdf', methods=['POST', 'GET'])
def generate_pdf():
    # Retrieve the reservation information from the request
    reservation_info = request.get_json()

    # Render the HTML template with the reservation information
    pdf_content = render_template('pdf_template.html',
reservation_info=reservation_info)

    # Configure PDFkit options (optional)
    options = {
        'page-size': 'A4',
        'encoding': 'UTF-8',
    }

    # Generate the PDF file from the HTML content
    pdf_file = pdfkit.from_string(pdf_content, False, options=options)

    # Return the PDF file as a response
    response = make_response(pdf_file)
    response.headers['Content-Type'] = 'application/pdf'
    response.headers['Content-Disposition'] = 'attachment;
filename="recu_demande.pdf"'
    return response

@auth.route('/commande', methods=['GET', 'POST'])
def commande():
    if request.method == 'POST':
        fournisseur = request.form.get('fournisseur')
        prix_totale = request.form.get('prix_totale')

        if prix_totale is None or prix_totale == '':

```



```

        flash('Please enter the total price.', category='error')
        return redirect(url_for('auth.commande'))

    try:
        prix_totale = float(prix_totale)
    except ValueError:
        flash('Invalid total price. Please enter a valid number.',
category='error')
        return redirect(url_for('auth.commande'))

    # Insert the command record into the database
    cursor = db.cursor()
    query = "INSERT INTO commande (fournisseur, commande_date, prix_totale)
VALUES (%s, CURDATE(), %s)"
    values = (fournisseur, prix_totale)
    cursor.execute(query, values)
    db.commit()

    # Get the list of products and calculate the total price
    products = []
    total_price = 0.0

    for i in range(1, 6):
        product_name = request.form.get(f'product{i}')
        product_price = request.form.get(f'price{i}')

        if product_name and product_price:
            products.append({'name': product_name, 'price':
float(product_price)})
            total_price += float(product_price)

    # Render the receipt template with the dynamic data
    html = render_template('receipt.html', fournisseur=fournisseur,
products=products, total_price=total_price, configuration=config)

    # Generate PDF using pdfkit
    pdf = pdfkit.from_string(html, False, configuration=config)

    # Save the PDF file
    file_path = 'C:/Users/Hamza/Desktop/receipt.pdf'
    with open(file_path, 'wb') as f:
        f.write(pdf)

    flash('Command created successfully!', category='success')
    return redirect(url_for('auth.commande'))

```

```

        return render_template('commande.html')

def generate_receipt_pdf(fournisseur, products, total_price):
    # Render the receipt HTML template with the provided data
    rendered_template = render_template(
        'receipt.html',
        fournisseur=fournisseur,
        products=products,
        total_price=total_price,
        configuration=config
    )

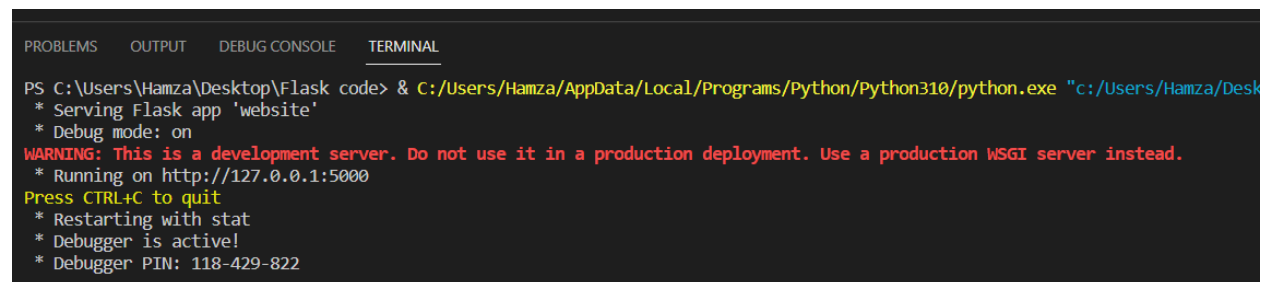
    # Set up PDF options
    options = {
        'page-size': 'A4',
        'encoding': 'UTF-8',
        'no-outline': None
    }

    # Generate PDF from the rendered template
    pdf = pdfkit.from_string(rendered_template, False,
options=options,configuration=config)

    return pdf

```

11. Manipulation :



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
PS C:\Users\Hamza\Desktop\Flask code> & C:/Users/Hamza/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Hamza/Desktop/Flask code/main.py"
* Serving Flask app 'website'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 118-429-822

```

On commence par le lancement de notre serveur est on se dirige vers 127.0.0.1:5000



Supposons qu'on n'a pas un compte on vas se diriger vers sign up :

Sign Up

Email Address
manipu@test.com

First Name
Manipu

Last Name
lation

Password

Password (Confirm)

Role
Student

Submit

Est-on remplis le formulaire et on clique submit cela vas afficher account created est si on se dirige vers notre base de donne on vas trouver le compte qu'on a crée

```

72 • select * from inscription;
73 • select * from document;
74 • select * from deductions;
75 • select * from borrowing;
76 • select * from tempoborrowing;
77 • select * from commande;
78

```

result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:								
id	email	prenom	nom	password	roll	score	regularization_request	
1	test@test.com	Hamza	Rahhali	123456	teacher	20	NULL	
2	test1@test.com	Ines	Sardi	123456	student	20	NULL	
8	test55@test.com	Hamza	Afif	123456	student	20	NULL	
9	manipu@test.com	Manipu	lation	123456	student	20	NULL	
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	

Maintenant on peut établir une connexion avec le login :

[Home](#)
[Logout](#)

Login successful!

Welcome, Manipu lation!

Current Demands:

ID	Book Title	Status
----	------------	--------

Active Book Loans:

ID	Book Title	First Name	Last Name	Period	Status	Action
----	------------	------------	-----------	--------	--------	--------

Available Documents:

Title	Type	ISBN	Author	Place	Release Year	Number of Copies	Empruntable
titre1	Livres	1234567	Auteur	Salle A1	2023	6	oui
titre2	Livres	123967	Auteu1	Salle A1	2022	4	non

Make a Demand

Cela doit nous afficher les demande établit ainsi que les commande accepter et déjà fait et les livres disponibles ou on peut faire une demande grâce à make a demand :

[Home](#)
[Logout](#)

Book Reservation

Book Title:

Nom :

Reservation Duration:

Return Date:

[Confirm Reservation](#)

Après cliquer confirme réservation on trouve que la demande est enregistrer dans notre base donner temporaire

```

76 • select * from tempoborrowing;
77 • select * from commande;
78

```

emprunt_id	book_id	person_id	periode	date_debut	date_fin	copies
60	1	1	short	2023-05-31	2023-06-15	NULL
61	1	2	long	2023-06-01	2023-07-01	NULL
63	1	9	short	2023-06-03	2023-06-18	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL

tempoborrowing 2 x

Ainsi que dans notre page home :

Welcome, Manipulation!						
Current Demands:			Active Book Loans:			
ID	Book Title	Status	Book ID	First Name	Last Name	Period
1	titre1	Pending				
Available Documents:						
Title	Type	ISBN	Author	Place	Release Year	Number of Copies
titre1	Livres	1234567	Auteur	Salle A1	2023	6
titre2	Livres	123967	Auteu1	Salle A1	2022	4
Make a Demand						

Si on visite la page admin maintenant on vas y trouver la demande ainsi que les informations à propos de cette dernière :

2	test1@test.com	Ines	Sardi	student	<input type="text" value="20"/>	<button>Update</button>	<button>Delete</button>
8	test55@test.com	Hamza	Afif	student	<input type="text" value="20"/>	<button>Update</button>	<button>Delete</button>
9	manipu@test.com	Manipu	lation	student	<input type="text" value="20"/>	<button>Update</button>	<button>Delete</button>

Add Person

Les livres

Title	Type	ISBN	Author	Place	Release Year	Number of Copies	Emprutable	Action
titre1	Livres	1234567	Auteur	Salle A1	2023	6	oui	<button>Delete</button>
titre2	Livres	123967	Auteu1	Salle A1	2022	4	non	<button>Delete</button>

Add Document

Demande de Réservations

Demande ID	Livre ID	Person ID	Period	Date de début	Action
60	1	1	short	2023-05-31	<button>Approved</button> <button>Not Approved</button>
61	1	2	long	2023-06-01	<button>Approved</button> <button>Not Approved</button>
63	1	9	short	2023-06-03	<button>Approved</button> <button>Not Approved</button>

Après avoir cliquer sur approuver on trouve que la demande est maintenat dans la table finale des prêt :

```

74 • select * from deductions;
75 • select * from borrowing;
76 • select * from tempoborrowing;
77 • select * from commande;
78

```

Result Grid

Filter Rows:

Edit:

Export/Import:

Wrap Cell Content:

	emprunt_id	book_id	person_id	periode	date_debut	date_fin	copies	statu_s	nom	prenom	posi
	51	1	1	short	NULL	NULL	NULL	approved	Hamza	Rahhali	Returned
	52	1	2	short	NULL	NULL	NULL	approved	Ines	Sardi	Returned
	53	1	8	long	NULL	NULL	NULL	approved	Hamza	Afif	NULL
	54	1	9	short	NULL	NULL	NULL	approved	Manipu	lation	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Est au même lieux on trouve dans notre home qu'il y a une demande active :

Welcome, Manipulation!

Current Demands:

ID	Book Title	Status
----	------------	--------

Active Book Loans:

Book ID	Book Title	First Name	Last Name	Period	Status	Action
1	titre1	lacion	Manipu	short	approved	Return Book

Available Documents:

Title	Type	ISBN	Author	Place	Release Year	Number of Copies	Emprutable
titre1	Livres	1234567	Auteur	Salle A1	2023	6	oui
titre2	Livres	123967	Auteu1	Salle A1	2022	4	non

[Make a Demand](#)

Maintenant si on clique sur return book le livre vas être retourner est on vas-y pouvoir le changement même dans la table :

Welcome, Manipulation!

Current Demands:

ID	Book Title	Status
----	------------	--------

Active Book Loans:

Book ID	Book Title	First Name	Last Name	Period	Status	Action
1	titre1	lacion	Manipu	short	approved	Returned

Available Documents:

Title	Type	ISBN	Author	Place	Release Year	Number of Copies	Emprutable
titre1	Livres	1234567	Auteur	Salle A1	2023	6	oui
titre2	Livres	123967	Auteu1	Salle A1	2022	4	non

[Make a Demand](#)

```
74 • select * from deductions;  
75 • select * from borrowing;  
76 • select * from tempoborrowing;  
77 • select * from commande;  
78
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: [IA](#)

emprunt_id	book_id	person_id	periode	date_debut	date_fin	copies	statu_s	nom	prenom	posi
51	1	1	short	NULL	NULL	NULL	approved	Hamza	Rahhali	Returned
52	1	2	short	NULL	NULL	NULL	approved	Ines	Sardi	Returned
53	1	8	long	NULL	NULL	NULL	approved	Hamza	Afif	Returned
54	1	9	short	NULL	NULL	NULL	approved	Manipu	lacion	Returned
*	NULL	NULL	NULL	NULL	NULL	NULL	approved	NULL	NULL	NULL

borrowing 5 x

Est si on vérifie notre dossier ou le fichier est localiser on vas trouver un fichier PDF contenant les information de la réservations :

Reservation Receipt

Reservation Information:

Reservation ID: 63

Book ID: 1

Person ID: 9

Period: short

Date de dÃ©but: 2023-06-03 12:55:04.974720

Maintenant on peut même supprimer l'utilisateur depuis la page admin grâce à :

Inscriptions

ID	Email	Prénom	Nom	Role	Score	Action
1	test@test.com	Hamza	Rahhali	teacher	<input type="text" value="20"/>	Update Delete
2	test1@test.com	Ines	Sardi	student	<input type="text" value="20"/>	Update Delete
8	test55@test.com	Hamza	Afif	student	<input type="text" value="20"/>	Update Delete
9	manipu@test.com	Manipu	lation	student	<input type="text" value="20"/>	Update Delete

Cela vas supprimer l'utilisateur de notre base de donner et depuis la page admin on peut établir une commande est générer un reçu PDF pour cette dernière :

Commande

Fournisseur:

Product:

Price:

Add Product

Make Commande

Afin de réaliser tout cela on a eu besoin de plusieurs page html comme base.html pour gérer l'ensemble des appels a Bootstrap notre Framework :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <link
      rel="stylesheet"
      href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.
css"
      integrity="sha384-
Vkoo8x4CGs03+HhXv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh"
      crossorigin="anonymous"
    />
    <link
      rel="stylesheet"
      href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-
awesome.min.css"
      crossorigin="anonymous"
    />

    <title>{% block title %}Home{% endblock %}</title>
  </head>
  <body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
      <button
        class="navbar-toggler"
        type="button"
        data-toggle="collapse"
        data-target="#navbar"
      >
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbar">
        <div class="navbar-nav">
          {% if 'logged_in' in session %}
            <a class="nav-item nav-link" id="home" href="/home">Home</a>
            <a class="nav-item nav-link" id="logout" href="{%
url_for('auth.logout') %}">Logout</a>
          {% else %}
            <a class="nav-item nav-link" id="login" href="/login">Login</a>
            <a class="nav-item nav-link" id="signUp" href="/sign-up">Sign Up</a>
          {% endif %}
        </div>
      </div>
    </nav>
  </body>
</html>
```

```

        </div>
    </div>
</nav>

{% with messages = get_flashed_messages(with_categories=true) %} {% if
messages %} {% for category, message in messages %} {% if category ==
'error' %}
<div class="alert alert-danger alert-dismissible fade show" role="alert">
    {{ message }}
    <button type="button" class="close" data-dismiss="alert">
        <span aria-hidden="true">&times;</span>
    </button>
</div>
{% else %}
<div class="alert alert-success alert-dismissible fade show" role="alert">
    {{ message }}
    <button type="button" class="close" data-dismiss="alert">
        <span aria-hidden="true">&times;</span>
    </button>
</div>
{% endif %} {% endfor %} {% endif %} {% endwith %}

<div class="container">{% block content %} {% endblock %}</div>
<script
    src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
    integrity="sha384-
KJ3o2DKtIkVYIK3UENzmM7KCKRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
    crossorigin="anonymous"
></script>
<script
    src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min
.js"
    integrity="sha384-
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
    crossorigin="anonymous"
></script>
<script
    src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"
    integrity="sha384-
JZR6Spejh4U02d8j0t6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmY1"
    crossorigin="anonymous"
></script>

{% block javascript %}
<script type="text/javascript">

```

```

        function deleteNote(noteId) {
        fetch("/delete-note", {
            method: "POST",
            body: JSON.stringify({ noteId: noteId }),
        }).then((_res) => {
            window.location.href = "/";
        });
        }
    </script>
{% endblock %}
</body>
</html>

```

La page admin.html:

```

{% extends 'base.html' %}

{% block content %}
<div class="container">
    <h2>Inscriptions</h2>
    <table class="table inscription-table">
        <thead>
            <tr>
                <th>ID</th>
                <th>Email</th>
                <th>Prénom</th>
                <th>Nom</th>
                <th>Role</th>
                <th>Score</th>
                <th>Action</th>
            </tr>
        </thead>
        <tbody>
            {% for inscription in inscriptions %}
            <tr>
                <td>{{ inscription[0] }}</td>
                <td>{{ inscription[1] }}</td>
                <td>{{ inscription[2] }}</td>
                <td>{{ inscription[3] }}</td>
                <td>{{ inscription[5] }}</td>
                <td>
                    <form action="{{ url_for('auth.update_score', person_id=inscription[0]) }}" method="POST" style="display:inline;">

```

```

        <input type="number" name="score" value="{{ inscription[6] }}"
min="0">
        <button type="submit" class="btn btn-sm btn-primary">Update</button>
    </form>
</td>
<td>
    <form action="{{ url_for('auth.delete_person',
person_id=inscription[0]) }}" method="POST" style="display:inline;">
        <button type="submit" class="btn btn-sm btn-danger">Delete</button>
    </form>
</td>
</tr>
{% endfor %}
</tbody>
</table>

<a href="{{ url_for('auth.sign_up') }}" class="btn btn-sm btn-primary">Add
Person</a>

<h2>Les livres</h2>
<table class="table document-table">
    <thead>
        <tr>
            <th>Title</th>
            <th>Type</th>
            <th>ISBN</th>
            <th>Author</th>
            <th>Place</th>
            <th>Release Year</th>
            <th>Number of Copies</th>
            <th>Emprunable</th>
            <th>Action</th>
        </tr>
    </thead>
    <tbody>
        {% for document in documents %}
        <tr>
            <td>{{ document[1] }}</td>
            <td>{{ document[2] }}</td>
            <td>{{ document[3] }}</td>
            <td>{{ document[5] }}</td>
            <td>{{ document[4] }}</td>
            <td>{{ document[6] }}</td>
            <td>{{ document[7] }}</td>
            <td>{{ document[8] }}</td>

```

```

        <td>
            <form action="{{ url_for('auth.delete_document') }}" method="POST"
style="display:inline;">
                <input type="hidden" name="titre" value="{{ document[0] }}">
                <button type="submit" class="btn btn-sm btn-danger">Delete</button>
            </form>
        </td>
    </tr>
    {% endfor %}
</tbody>
</table>

<a href="{{ url_for('auth.ajout') }}" class="btn btn-sm btn-primary">Add
Document</a>

<h2>Demande de Réservations</h2>
<table class="table reservation-table">
    <thead>
        <tr>
            <th>Demande ID</th>
            <th>Livre ID</th>
            <th>Person ID</th>
            <th>Period</th>
            <th>Date de début</th>
            <th>Action</th>
        </tr>
    </thead>
    <tbody>
        {% for reservation in reservations %}
        <tr>
            <td>{{ reservation[0] }}</td>
            <td>{{ reservation[1] }}</td>
            <td>{{ reservation[2] }}</td>
            <td>{{ reservation[3] }}</td>
            <td>{{ reservation[4] }}</td>
            <td>
                <form action="{{ url_for('auth.approved') }}" method="POST"
style="display:inline;">
                    <input type="hidden" name="demande_id" value="{{ reservation[0] }}">
                    <button type="submit" class="btn btn-sm btn-
success">Approved</button>
                </form>
                <form action="{{ url_for('auth.approved') }}" method="POST"
style="display:inline;">
                    <input type="hidden" name="demande_id1" value="{{ reservation[0] }}">

```

```

        <button type="submit" class="btn btn-sm btn-danger">Not
Approved</button>
    </form>
</td>
</tr>
{% endfor %}
</tbody>
</table>

<h2>Regularization Requests</h2>
<table class="table regularization-table">
    <thead>
        <tr>
            <th>ID</th>
            <th>Nom</th>
            <th>Regularization Request</th>
            <th>Action</th>
        </tr>
    </thead>
    <tbody>
        {% for inscription in inscriptions %}
        <tr>
            <td>{{ inscription[0] }}</td>
            <td>{{ inscription[3] }}</td>
            <td>
                {% if inscription[7] %}
                {{ inscription[7] }}
                {% else %}
                No regularization request
                {% endif %}
            </td>
            <td>
                <form action="{{ url_for('auth.delete_regularization',
inscription_id=inscription[0]) }}" method="POST" style="display:inline;">
                    <button type="submit" class="btn btn-sm btn-danger">Delete</button>
                </form>
            </td>
        </tr>
        {% endfor %}
    </tbody>
</table>
    <a href="{{ url_for('auth.commande') }}" class="btn btn-sm btn-primary">Make a
Command</a>
</div>
{% endblock %}

```

La page home.html :

```
{% extends "base.html" %}

{% block content %}
<div class="container">
  <h2 class="text-center">Welcome, {{ user }}!</h2>

  <div class="row">
    <div class="col-md-6">
      <div class="card mt-4">
        <div class="card-body">
          <h3 class="card-title">Current Demands:</h3>
          <table class="table demand-table">
            <thead>
              <tr>
                <th>ID</th>
                <th>Book Title</th>
                <th>Status</th>
              </tr>
            </thead>
            <tbody>
              {% for demand in demands %}
              <tr>
                <td>{{ demand[0] }}</td>
                <td>{{ demand[1] }}</td>
                <td>Pending</td>
              </tr>
              {% endfor %}
            </tbody>
          </table>
        </div>
      </div>
    </div>

    <div class="col-md-6">
      <div class="card mt-4">
        <div class="card-body">
          <h3 class="card-title">Active Book Loans:</h3>
          <table class="table loan-table">
            <thead>
              <tr>
                <th>ID</th>
                <th>Book Title</th>
                <th>First Name</th>
              </tr>
            </thead>
            <tbody>
              {% for loan in loans %}
              <tr>
                <td>{{ loan[0] }}</td>
                <td>{{ loan[1] }}</td>
                <td>{{ loan[2] }}</td>
              </tr>
              {% endfor %}
            </tbody>
          </table>
        </div>
      </div>
    </div>
  </div>
</div>
```

```

        <th>Last Name</th>
        <th>Period</th>
        <th>Status</th>
        <th>Action</th>
    </tr>
</thead>
<tbody>
    {% for loan in loans %}
    <tr>
        <td>{{ loan[0] }}</td>
        <td>{{ loan[1] }}</td>
        <td>{{ loan[2] }}</td>
        <td>{{ loan[3] }}</td>
        <td>{{ loan[4] }}</td>
        <td>{{ loan[5] }}</td>
        <td>
            {% if loan[5] == "approved" and loan[6] is none %}
            <form action="{{ url_for('auth.return_book', loan_id=loan[0])
}}}" method="POST" style="display: inline;">
                <input type="hidden" name="loan_id" value="{{ loan[0] }}">
                <button type="submit" class="btn btn-sm btn-primary">Return
Book</button>
            </form>
            {% else %}
            Returned
            {% endif %}
        </td>
    </tr>
    {% endfor %}
</tbody>
</table>
</div>
</div>
</div>
</div>

<h3 class="mt-4">Available Documents:</h3>
<table class="table document-table">
    <thead>
        <tr>
            <th>Title</th>
            <th>Type</th>
            <th>ISBN</th>
            <th>Author</th>
            <th>Place</th>

```



```

        <th>Release Year</th>
        <th>Number of Copies</th>
        <th>Empruntable</th>
    </tr>
</thead>
<tbody>
    {% for document in documents %}
    <tr>
        <td>{{ document[1] }}</td>
        <td>{{ document[2] }}</td>
        <td>{{ document[3] }}</td>
        <td>{{ document[5] }}</td>
        <td>{{ document[4] }}</td>
        <td>{{ document[6] }}</td>
        <td>{{ document[7] }}</td>
        <td>{{ document[8] }}</td>
    </tr>
    {% endfor %}
</tbody>
</table>

<div class="text-center mt-4">
    <a href="{{ url_for('auth.reservations') }}" class="btn btn-primary
reservation-btn">Make a Demand</a>
</div>
</div>
{% endblock %}

```

La page login :

```

{% extends "base.html" %}

{% block title %}Login{% endblock %}

{% block content %}
<div class="container">
    <div class="row justify-content-center">
        <div class="col-lg-6">
            <div class="card mt-5">
                
                <div class="card-body">
                    <h3 class="card-title text-center">Login</h3>
                    <form method="POST">
                        <div class="form-group">

```

```

        <label for="email">Email Address</label>
        <input type="email" class="form-control" id="email" name="email"
placeholder="Enter email">
    </div>
    <div class="form-group">
        <label for="password">Password</label>
        <input type="password" class="form-control" id="password"
name="password" placeholder="Enter password">
    </div>
    <div class="text-center">
        <button type="submit" class="btn btn-primary">Login</button>
    </div>
</form>
</div>
</div>
</div>
</div>
</div>
{% endblock %}

```

La page sign Up :

```

{% extends "base.html" %}
{% block title %}Sign Up{% endblock %}
{% block content %}
<div class="container">
    <div class="row justify-content-center">
        <div class="col-md-6">
            <div class="card mt-4">
                <div class="card-body">
                    <h3 class="card-title text-center">Sign Up</h3>
                    <form method="POST">
                        <div class="form-group">
                            <label for="email">Email Address</label>
                            <input
                                type="email"
                                class="form-control"
                                id="email"
                                name="email"
                                placeholder="Enter email"
                            />
                        </div>
                        <div class="form-group">
                            <label for="firstName">First Name</label>

```

```

        <input
            type="text"
            class="form-control"
            id="firstName"
            name="firstName"
            placeholder="Enter first name"
        />
    </div>
    <div class="form-group">
        <label for="lastName">Last Name</label>
        <input
            type="text"
            class="form-control"
            id="lastName"
            name="lastName"
            placeholder="Enter last name"
        />
    </div>
    <div class="form-group">
        <label for="password1">Password</label>
        <input
            type="password"
            class="form-control"
            id="password1"
            name="password1"
            placeholder="Enter password"
        />
    </div>
    <div class="form-group">
        <label for="password2">Password (Confirm)</label>
        <input
            type="password"
            class="form-control"
            id="password2"
            name="password2"
            placeholder="Confirm password"
        />
    </div>
    <div class="form-group">
        <label for="role">Role</label>
        <select class="form-control" id="role" name="role">
            <option value="student">Student</option>
            <option value="teacher">Teacher</option>
        </select>
    </div>

```

```

        <br />
        <div class="text-center">
            <button type="submit" class="btn btn-primary">Submit</button>
        </div>
    </form>
</div>
</div>
</div>
</div>
</div>
{% endblock %}

```

La page reservation :

```

{% extends "base.html" %}

{% block title %}Book Reservation{% endblock %}

{% block content %}
    <div class="container">
        <h2>Book Reservation</h2>
        <form action="{{ url_for('auth.reservations') }}" method="POST">
            <div class="form-group">
                <label for="book-title">Book Title:</label>
                <input type="text" id="book-title" name="book-title" class="form-control"
required>
            </div>
            <div class="form-group">
                <label for="Nom">Nom :</label>
                <input type="text" id="Nom" name="Nom" class="form-control" required>
            </div>
            <div class="form-group">
                <label for="duration">Reservation Duration:</label>
                <select id="duration" name="duration" class="form-control" required>
                    <option value="short">Short</option>
                    <option value="long">Long</option>
                    <option value="annual">Annual</option>
                </select>
            </div>
            <div class="form-group">
                <label for="return_date">Return Date:</label>
                <input type="text" id="return_date" name="return_date" class="form-
control" readonly>
            </div>
        </form>
    </div>
{% endblock %}

```

```

    </div>
    <button type="submit" class="btn btn-primary">Confirm Reservation</button>
  </form>
</div>

<script>
var durationSelect = document.getElementById("duration");
var returnDateInput = document.getElementById("return_date");

durationSelect.addEventListener("change", function() {
  var selectedDuration = durationSelect.value;
  var currentDate = new Date();
  var returnDate = new Date();

  if (selectedDuration === "short") {
    returnDate.setDate(currentDate.getDate() + 15);
  } else if (selectedDuration === "long") {
    returnDate.setDate(currentDate.getDate() + 30);
  } else if (selectedDuration === "annual") {
    returnDate.setFullYear(currentDate.getFullYear() + 1, 11, 31);
  }

  var formattedReturnDate = returnDate.toISOString().split("T")[0];
  returnDateInput.value = formattedReturnDate;
});
</script>
{% endblock %}

```

Comme vous remarquer on a fait appel a javascript pour etablir les differente operation de calculation de date

La page commande :

```

document.querySelector('form').addEventListener('submit', function(event) {
  var totalPrices = document.querySelectorAll('input[name^="price"]');
  var totalPrice = 0;
  for (var i = 0; i < totalPrices.length; i++) {
    if (totalPrices[i].value !== '') {
      totalPrice += parseFloat(totalPrices[i].value);
    }
  }
  var totalPriceInput = document.createElement('input');
  totalPriceInput.type = 'hidden';

```

```

        totalPriceInput.name = 'prix_totale';
        totalPriceInput.value = totalPrice.toFixed(2);
        this.appendChild(totalPriceInput);
    });
});
</script>
{% endblock %}

```

La page ajout livre :

```

{% extends "base.html" %}
{% block title %}Add Document{% endblock %}
{% block content %}
<form method="POST">
  <h3 align="center">Add Document</h3>
  <div class="form-group">
    <label for="title">Title</label>
    <input
      type="text"
      class="form-control"
      id="title"
      name="title"
      placeholder="Enter title"
    />
  </div>
  <div class="form-group">
    <label for="type">Type</label>
    <select class="form-control" id="type" name="type">
      <option value="CD">CD</option>
      <option value="Thèses">Thèses</option>
      <option value="Article">Article</option>
      <option value="Livres">Livres</option>
    </select>
  </div>
  <div class="form-group">
    <label for="isbn">ISBN</label>
    <input
      type="text"
      class="form-control"
      id="isbn"
      name="isbn"
      placeholder="Enter ISBN"
    />
  </div>
</form>

```

```

</div>
<div class="form-group">
  <label for="place">Place</label>
  <input
    type="text"
    class="form-control"
    id="place"
    name="place"
    placeholder="Enter place"
  />
</div>
<div class="form-group">
  <label for="author">Author</label>
  <input
    type="text"
    class="form-control"
    id="author"
    name="author"
    placeholder="Enter author"
  />
</div>
<div class="form-group">
  <label for="releaseYear">Release Year</label>
  <input
    type="text"
    class="form-control"
    id="releaseYear"
    name="releaseYear"
    placeholder="Enter release year"
  />
</div>
<div class="form-group">
  <label for="numCopies">Number of Copies</label>
  <input
    type="text"
    class="form-control"
    id="numCopies"
    name="numCopies"
    placeholder="Enter number of copies"
  />
</div>
<div class="form-group">
  <label for="borrowable">Borrowable</label>
  <select class="form-control" id="borrowable" name="borrowable">
    <option value="1">Yes</option>

```

```

        <option value="0">No</option>
    </select>
</div>
<br />
<button type="submit" class="btn btn-primary">Submit</button>
</form>
{% endblock %}

```

Et bien sur les formes des pdf a generer :

```

<!DOCTYPE html>
<html>
<head>
    <title>Reservation Receipt</title>
    <style>
        body {
            font-family: Arial, sans-serif;
        }

        h1 {
            color: #333;
            font-size: 24px;
            margin-bottom: 20px;
        }

        ul {
            list-style: none;
            padding: 0;
        }

        li {
            margin-bottom: 10px;
        }

        .reservation-info {
            margin-bottom: 30px;
        }
    </style>
</head>
<body>
    <div class="reservation-info">
        <h1>Reservation Receipt</h1>
        <p>Reservation Information:</p>
    </div>

```



```

        <ul>
            {% for reservation in reservation_info %}
                <li><strong>Reservation ID:</strong> {{ reservation[0] }}</li>
                <li><strong>Book ID:</strong> {{ reservation[1] }}</li>
                <li><strong>Person ID:</strong> {{ reservation[2] }}</li>
                <li><strong>Period:</strong> {{ reservation[3] }}</li>
                <li><strong>Date de début:</strong> {{ reservation[4] }}</li>
            {% endfor %}
        </ul>
    </div>
</body>
</html>

```

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Receipt</title>
    <style>
        body {
            font-family: Arial, sans-serif;
        }
        h1 {
            text-align: center;
        }
        table {
            width: 100%;
            border-collapse: collapse;
        }
        th, td {
            padding: 8px;
            text-align: left;
            border-bottom: 1px solid #ddd;
        }
        tfoot td {
            font-weight: bold;
        }
    </style>
</head>
<body>
    <h1>Receipt</h1>
    <h2>Fournisseur: {{ fournisseur }}</h2>

```

```

<table>
  <thead>
    <tr>
      <th>Product</th>
      <th>Price</th>
    </tr>
  </thead>
  <tbody>
    {% for product in products %}
      <tr>
        <td>{{ product['name'] }}</td>
        <td>{{ product['price'] }}</td>
      </tr>
    {% endfor %}
  </tbody>
  <tfoot>
    <tr>
      <td colspan="2">Total Price: {{ total_price }}</td>
    </tr>
  </tfoot>
</table>
</body>
</html>

```

12. Une requête qui permet la modification des mots de passe de tous les utilisateurs

```

1  @auth.route('/changepassword/<int:person_id>', methods=['POST'])
2  def changepassword(person_id):
3      cursor = db.cursor()
4      update_query = "UPDATE inscription SET password = CONCAT(nom,
5      prenom) WHERE id = %s"
6      cursor.execute(update_query, (person_id,))
7
8      cursor.fetchall()
9      db.commit()
10     flash('Passwords changed successfully!', category='success')
11     return redirect(url_for('auth.admin'))

```

En conclusion, ce rapport a abordé la conception et le développement d'une application web utilisant le framework Flask en Python. L'objectif principal du projet était de créer une application de gestion d'une bibliothèque, mettant en œuvre des fonctionnalités telles que l'inscription des adhérents, la gestion des documents, les emprunts et les retours, ainsi que l'authentification des utilisateurs.

L'intégration de composants tels que Jinja2 et Bootstrap a permis de créer des pages web dynamiques et esthétiquement agréables, offrant une expérience utilisateur conviviale. De plus, l'utilisation d'une base de données MySQL a permis de stocker et de récupérer efficacement les données liées à la bibliothèque et aux utilisateurs.