

❖ Objectif

Développer une application web conviviale et sécurisée permettant aux utilisateurs de gérer leur CV en ligne. L'objectif est de faciliter la création de CV et d'améliorer leur accessibilité pour les employeurs ou toute autre partie intéressée.

❖ Exigences fonctionnelles

Création de compte et connexion

- Les utilisateurs doivent pouvoir s'inscrire avec une adresse e-mail et un mot de passe.
- Une option de connexion avec l'e-mail et le mot de passe doit être disponible.

Gestion du compte personnel

- L'utilisateur doit pouvoir modifier ses informations personnelles (nom, e-mail, etc.).
- L'utilisateur doit pouvoir télécharger et changer sa photo de profil.

Gestion du CV

- Le système doit permettre à l'utilisateur d'ajouter des informations sur son éducation, expériences professionnelles, compétences, projets et langues.
- L'utilisateur doit pouvoir ajouter, modifier et supprimer chaque élément de ces catégories.
- Chaque section du CV doit avoir une interface dédiée pour saisir et modifier les données.

Validation des entrées

- Les données saisies doivent être validées (vérification de l'e-mail et du mot de passe)
- Les images téléchargées doivent être conformes aux formats autorisés (JPEG, PNG, etc.).

Affichage du CV

- L'utilisateur doit pouvoir visualiser son CV complet après avoir saisi les informations.
- Une option pour télécharger le CV en PDF ou dans un format imprimable doit être disponible.

Gestion des sessions

- L'utilisateur doit pouvoir se connecter et se déconnecter facilement.
- L'application doit être sécurisée avec des méthodes standard (ex. : chiffrement des mots de passe).

Notifications

- L'utilisateur doit recevoir des notifications pour les opérations réussies ou échouées (inscription, modification du compte, ajout de section CV, etc.).

Interface frontale (Frontend)

- L'interface utilisateur doit être simple et intuitive.
- Elle doit inclure des formulaires de saisie et des boutons de contrôle (enregistrer, modifier, supprimer).

Gestion des fichiers

- L'utilisateur doit pouvoir télécharger des photos de profil et des images liées à ses projets.

❖ Exigences techniques

Technologies utilisées

- **Frontend** : HTML5, CSS3 (avec Bootstrap), JavaScript.
- **Backend** : Python avec Flask.
- **Base de données** : SQLite.
- **Validation des entrées** : Flask-WTF et WTForms.
- **Gestion des sessions** : Flask-Login.
- **Gestion des fichiers** : Flask-Uploads.
- **Notifications** : Flask-Flash.



Outils de développement supplémentaires

- **Validation des entrées** : Utiliser Flask-WTF et WTForms pour vérifier les données.
- **Sécurité** : Utiliser des techniques comme le chiffrement des mots de passe (via Werkzeug).

5. Exigences non fonctionnelles

Performance

- Le système doit être réactif, avec un temps de chargement des pages inférieur à 3 secondes.

Sécurité

- L'application doit être sécurisée avec des techniques comme le chiffrement des mots de passe.
- Le système doit empêcher tout accès non autorisé aux données personnelles et aux CV.

Évolutivité

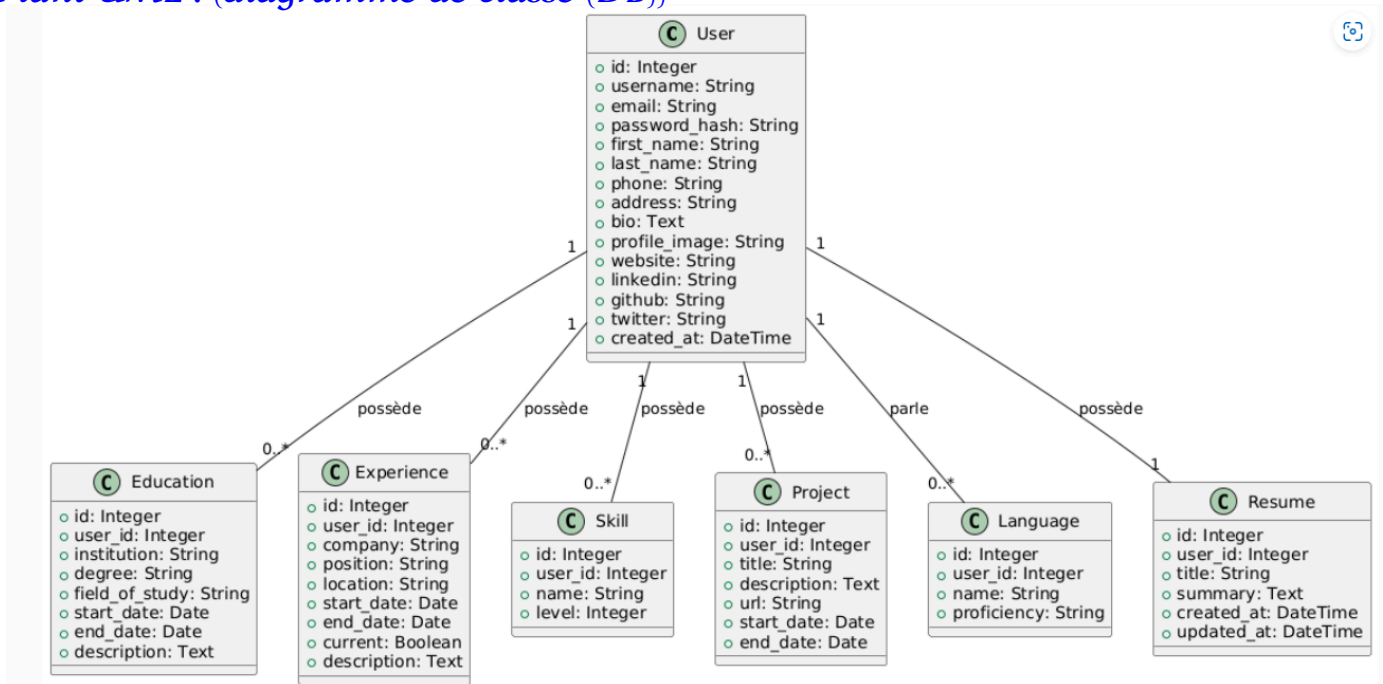
- Le système doit être évolutif pour permettre l'ajout de fonctionnalités futures ex : multilingue, intégration avec d'autres plateformes.

Phase de réalisation

Phase 1 : Planification et conception (Conception et architecture)

- Définition de la structure des données (base de données, modèles).
- Conception des interfaces utilisateur de base. (les formulaire)

★ Plant UML : (diagramme de classe (DB))



exemple de modele User avec flask :

```
from app import db
from flask_login import UserMixin
from werkzeug.security import generate_password_hash, check_password_hash
from datetime import datetime

# Modèle représentant un utilisateur
class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(64), unique=True, nullable=False) # Nom d'utilisateur unique et obligatoire
    email = db.Column(db.String(120), unique=True, nullable=False) # Email unique et obligatoire
    password_hash = db.Column(db.String(128)) # Hash du mot de passe pour la sécurité
    first_name = db.Column(db.String(64), nullable=True)
    last_name = db.Column(db.String(64), nullable=True)
    phone = db.Column(db.String(20), nullable=True)
    address = db.Column(db.String(200), nullable=True)
    bio = db.Column(db.Text, nullable=True) # Biographie de l'utilisateur
    profile_image = db.Column(db.String(200), nullable=True) # Image de profil de l'utilisateur
    website = db.Column(db.String(200), nullable=True)
    linkedin = db.Column(db.String(200), nullable=True)
    github = db.Column(db.String(200), nullable=True)
    twitter = db.Column(db.String(200), nullable=True)
    created_at = db.Column(db.DateTime, default=datetime.utcnow) # Date de création du compte

    # Méthode pour définir le mot de passe de manière sécurisée (en le hachant)
    def set_password(self, password):
        self.password_hash = generate_password_hash(password)

    # Méthode pour vérifier si le mot de passe entré correspond au mot de passe haché
    def check_password(self, password):
        return check_password_hash(self.password_hash, password)

    # Représentation de l'utilisateur sous forme de chaîne de caractères
    def __repr__(self):
        return f'<User {self.username}>'
```

Exemple de Formuler en flask :

```
from flask_wtf import FlaskForm
from wtforms import StringField, PasswordField, BooleanField, SubmitField
from wtforms.validators import DataRequired, Length, Email, EqualTo, ValidationError
from app.models.user import User

class RegistrationForm(FlaskForm):
    # Champs pour l'inscription avec des validations
    first_name = StringField('Prénom', validators=[DataRequired(), Length(min=2, max=25)])
    last_name = StringField('Nom de famille', validators=[DataRequired(), Length(min=2, max=25)])
    username = StringField('User_name', validators=[DataRequired(), Length(min=2, max=25)])
    email = StringField('Adresse email', validators=[DataRequired(), Email()])
    password = PasswordField('Mot de passe', validators=[DataRequired(), Length(min=6, max=10)])
    confirm_password = PasswordField('Confirmer le mot de passe', validators=[DataRequired(), EqualTo('password'),
    Length(min=6, max=10)])
    submit = SubmitField('S\'inscrire')

    # Vérification de l'unicité du nom d'utilisateur
    def validate_username(self, username):
        user = User.query.filter_by(username=username.data).first()
        if user:
            raise ValidationError("Ce nom d'utilisateur est déjà pris. Veuillez en choisir un autre.")

    # Vérification de l'unicité de l'email
    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user:
            raise ValidationError("Cet e-mail est déjà utilisé. Veuillez en utiliser un autre.")

class LoginForm(FlaskForm):
    # Champs pour la connexion avec des validations
    email = StringField('Adresse email', validators=[DataRequired(), Email()])
    password = PasswordField('Mot de passe', validators=[DataRequired(), Length(min=6, max=10)])
    remember = BooleanField('Rester connecté')
    submit = SubmitField('Se connecter')
```

Les formulaires dans Flask sont utilisés comme un moyen de collecter les données des utilisateurs et de vérifier leur validité avant de les traiter ou de les enregistrer dans la base de données. Dans le code que j'ai fourni, il y a deux formulaires principaux :

Formulaire d'inscription (RegistrationForm) : Ce formulaire est utilisé lorsque l'utilisateur souhaite créer un nouveau compte sur l'application. Il demande à l'utilisateur de remplir plusieurs champs, tels que : prénom, nom de famille, nom d'utilisateur, adresse e-mail, mot de passe et confirmation du mot de passe.

Objectif : L'objectif de ce formulaire est de collecter les informations de base de l'utilisateur et de les vérifier (comme s'assurer que l'adresse e-mail est valide et que le nom d'utilisateur n'est pas déjà utilisé dans la base de données).

Formulaire de connexion (LoginForm) : Ce formulaire est utilisé lorsque l'utilisateur souhaite se connecter à un compte déjà existant dans l'application. Il nécessite uniquement l'adresse e-mail et le mot de passe.

Objectif : L'objectif de ce formulaire est de vérifier que l'utilisateur qui essaie de se connecter est déjà inscrit dans l'application, en validant les informations de connexion (adresse e-mail et mot de passe).

Vérification des données (Validators) :

La vérification des données est le processus qui permet de s'assurer que les informations saisies par l'utilisateur respectent les règles définies par l'application. Dans Flask, on utilise des validators pour valider les champs des formulaires. Dans le code, plusieurs validators sont utilisés, tels que :

1. **DataRequired()** : Vérifie que le champ n'est pas vide. Par exemple, si l'utilisateur tente de laisser le champ "prénom" vide, l'application rejettera l'entrée et affichera un erreur
2. **Length(min=2, max=25)** : Vérifie que la longueur de l'entrée dans le champ ne dépasse pas une certaine valeur maximale ni ne soit inférieure à une certaine valeur minimale. Par exemple, si l'utilisateur entre un prénom trop long ou trop court, l'entrée sera rejetée.
3. **Email()** : Vérifie que l'entrée dans le champ e-mail est une adresse e-mail valide. Si l'e-mail est invalide , un message d'erreur sera affiché.
4. **EqualTo('password')** : Vérifier que le champ "Confirmer le mot de passe" correspond bien à celui du champ "Mot de passe". Si les deux ne sont pas identiques, un message d'erreur sera affiché demandant à l'utilisateur de saisir correctement le mot de passe.

Vérification de l'unicité (Unique Validation) :

Il est important que l'application garantisse qu'aucune donnée sensible, telle que le nom d'utilisateur ou l'adresse e-mail, ne soit dupliquée dans la base de données, afin qu'un autre utilisateur ne puisse pas utiliser les mêmes informations. Dans le code, l'unicité est vérifiée à l'aide de fonctions personnalisées telles que :

1. **validate_username(self, username)** : Lorsque l'utilisateur entre un nom d'utilisateur, l'application vérifie dans la base de données si ce nom est déjà pris. Si le nom d'utilisateur existe déjà, une exception (**ValidationError**) est levée et un message est affiché pour informer l'utilisateur qu'il doit choisir un autre nom d'utilisateur.
2. **validate_email(self, email)** : Il en va de même pour l'adresse e-mail : lorsque l'utilisateur entre un e-mail, l'application vérifie s'il est déjà utilisé. Si l'e-mail existe dans la base de données, une exception est levée et l'utilisateur est invité à entrer un e-mail différent.

Cette opération garantit que chaque nom d'utilisateur et chaque adresse e-mail dans l'application sont uniques et non dupliqués, ce qui améliore l'expérience de l'utilisateur et empêche les problèmes pouvant découler de la répétition des données.

Template de base :

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>{% block title %}Système de CV{% endblock %}</title> <!-- Titre de la page, défini par un bloc dynamique -->
  <link rel="stylesheet" href="{{ url_for('static', filename='css/main.css') }}"><!-- Lien vers les style CSS principales -->
  <link rel="stylesheet" href="{{ url_for('static', filename='css/resume.css') }}"><!-- style CSS spécifique au CV -->
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/font-awesome@6.4.0/css/all.min.css"><!-- icônes social -->
  {% block styles %}{% endblock %} <!-- Bloc de styles personnalisés -->
</head>
<body>
  <!-- Barre de navigation principale -->
  <nav class="navbar">
    <div class="container">
      <!-- Lien vers la page d'affichage du CV -->
      <a class="navbar-brand" href="{{ url_for('resume.view_resume') }}">CVPro <span class="plus-icon">+</span></a>
      <ul class="nav-links" id="navbarNav">
        <!-- Affichage des liens uniquement si l'utilisateur est connecté -->
        {% if current_user.is_authenticated %}
          <li><a href="{{ url_for('resume.view_resume') }}">CV</a></li>
          <li><a href="{{ url_for('profile.view_profile') }}">Profil</a></li>
          {% endif %}
        </ul>
        <ul class="nav-auth">
          <!-- Lien vers la page d'accueil -->
          <li><a href="{{ url_for('home') }}">Accueil</a></li>
          {% if current_user.is_authenticated %}
            <li class="dropdown">
              <a href="#" class="dropbtn">
                <!-- Affichage de l'image de profil si elle est disponible -->
                {% if current_user.profile_image %}
                  
                {% endif %}
                {{ current_user.username }}
              </a>
              <div class="dropdown-content"> <!-- Menu déroulant avec des options de profil -->
                <a href="{{ url_for('profile.profile') }}">Modifier le profil</a>
                <a href="{{ url_for('auth.logout') }}">Se déconnecter</a>
              </div>
            </li>
            {% else %}
              <!-- Liens pour la connexion et l'inscription si l'utilisateur n'est pas connecté -->
              <li><a href="{{ url_for('auth.login') }}">Se connecter</a></li>
              <li><a href="{{ url_for('auth.register') }}">Créer un compte</a></li>
              {% endif %}
            </ul>
          </div>
        </nav>

        <div class="container mt-4">
          <!-- Gestion des messages flash, utilisés pour afficher des notifications -->
          {% with messages = get_flashed_messages(with_categories=true) %}
            {% if messages %}
              {% for category, message in messages %}
                <div class="alert alert-{{ category }}">
                  {{ message }}
                </div>
              {% endfor %}
            {% endif %}
          </div>
```

```

    {% endif %}
{% endwhile %}
{% block content %}{% endblock %} <!-- Contenu principal de la page, défini par un bloc dynamique -->
</div>

<!-- Pied de page avec des liens vers les réseaux sociaux -->
<footer class="footer">
    <div class="footer-content">
        <p>&copy; 2025 Application de gestion des utilisateurs</p>
        <div class="social-links"> <!-- Liens vers les icônes de réseaux sociaux -->
            <a href="#"><i class="fab fa-github"></i></a>
            <a href="#"><i class="fab fa-twitter"></i></a>
            <a href="#"><i class="fab fa-linkedin"></i></a>
        </div>
    </div>
</footer>

<!-- Script JavaScript pour cacher les messages flash après un délai -->
<script>
    document.addEventListener("DOMContentLoaded", function () {
        let flashMessages = document.querySelectorAll(".alert");
        flashMessages.forEach(function (message) {
            setTimeout(function () {
                message.style.transition = "opacity 0.5s"; // Transition de disparition
                message.style.opacity = "0"; // Réduction de l'opacité
                setTimeout(() => message.remove(), 500); // Suppression du message après la transition
            }, 3000); // Délai avant de commencer la transition
        });
    });
</script>
{% block scripts %}{% endblock %}<!-- Bloc de scripts personnalisés -->
</body>
</html>

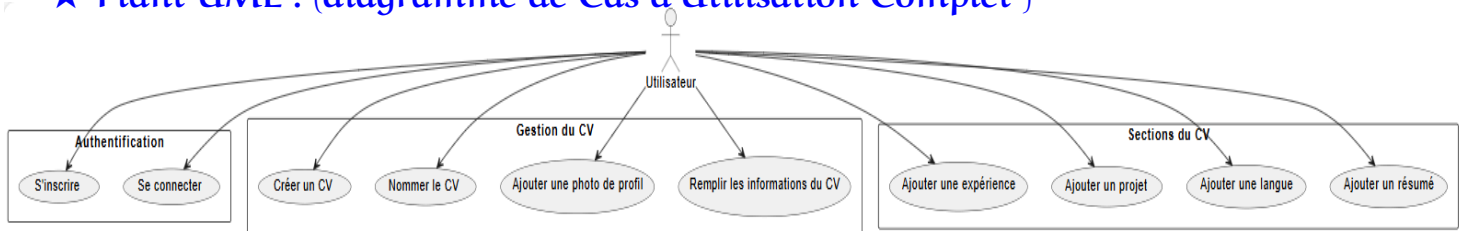
```

Ce code est de fournir une structure de base pour une page web en utilisant Flask. Il comprend :

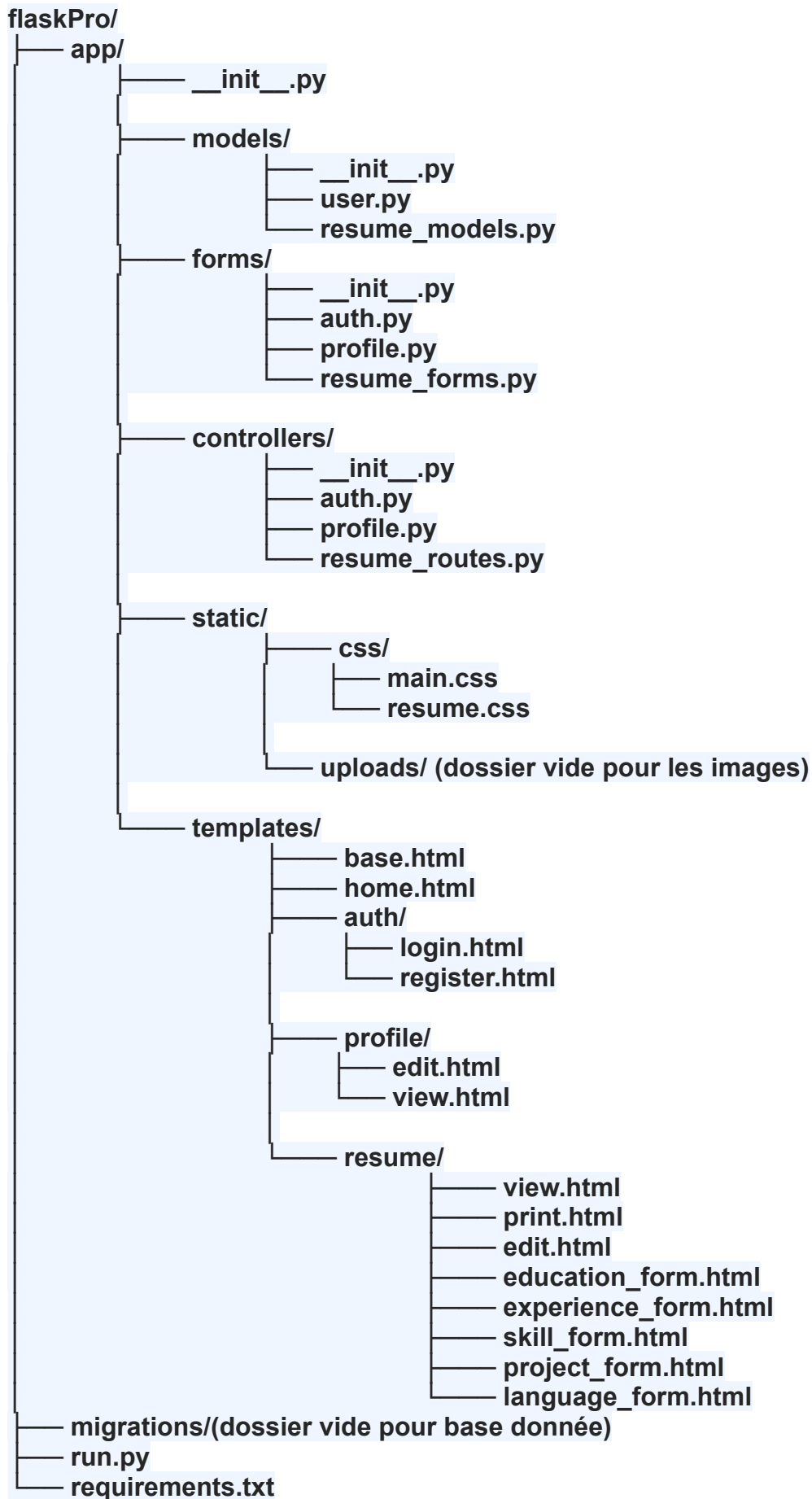
1. **Paramètres de la page** : Liens vers les fichiers CSS et FontAwesome pour la mise en forme
2. **Barre de navigation** : Contient des liens vers différentes pages comme la page de visualisation du CV et le profil utilisateur, avec une gestion dynamique de la navigation selon l'état de la connexion
3. **Gestion des messages flash** : Affiche des notifications (messages flash) à l'utilisateur après certaines actions (comme le succès ou l'échec d'une opération).
4. **Contenu principal** : Ce bloc est personnalisable avec des blocs dynamiques Flask pour afficher le contenu approprié.
5. **Pied de page** : Inclut des liens vers les réseaux sociaux.
6. **Masquage des messages** : Utilisation de JavaScript pour masquer les messages flash après un délai.

En résumé, l'objectif est de créer une page web avec une structure **flexible** et complète pour gérer l'authentification, l'interaction avec l'utilisateur, et l'affichage des notifications, tout en offrant une interface utilisateur personnalisable.

★ Plant UML : (diagramme de Cas d'Utilisation Complet)



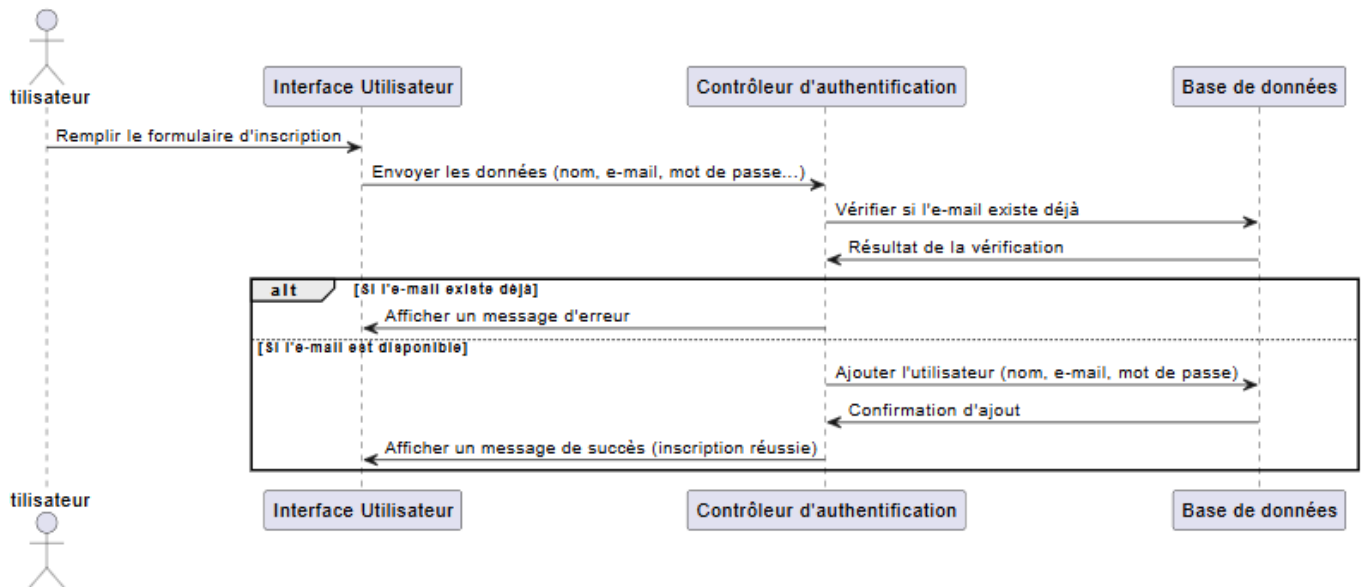
★ Explication de l'architecture du projet :



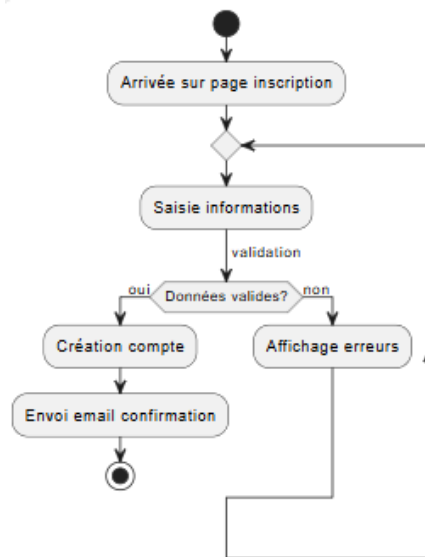
Phase 2 : Développement du système

- Développement des fonctionnalités principales (inscription, connexion, gestion de compte).
- Développement des interfaces CV et des options de modification.

★ Plant UML : Diagramme de Séquence (register)



★ Plant UML : Diagramme d'Activité (register)



★ Interface Utilisateur :

```
{% extends 'base.html' %} <!--Héritage du template de base-->
{% block title %}Créer un compte{% endblock %} <!--le titre de la page-->
{% block content %} <!--Le bloc content contient l'élément principal de la page-->
<div class="container">
    <div class="card">
        <h1 class="card-title">Créer un compte</h1>
        <!-- Formulaire de création de compte -->
        <form method="post" class="auth-form">
            {{ form.hidden_tag() }}

            <div class="form-row">
                <!-- Champ pour le nom de famille -->
                <div class="form-group col-md-6">
                    {{ form.last_name.label(class="form-label") }}
                    {{ form.last_name(class="form-control") }}
                    {% if form.last_name.errors %}
                        <div class="invalid-feedback">
                            {% for error in form.last_name.errors %}
                                <span>{{ error }}</span>
                            {% endfor %}
                        </div>
                    {% endif %}
                </div>
            </div>
        </form>
    </div>
</div>
```

```

</div>
{% endif %}
</div>

<!-- Champ pour le prénom -->
<div class="form-group col-md-6">
  {{ form.first_name.label(class="form-label") }}
  {{ form.first_name(class="form-control") }}
  {% if form.first_name.errors %}
    <div class="invalid-feedback">
      {% for error in form.first_name.errors %}
        <span>{{ error }}</span>
      {% endfor %}
    </div>
  {% endif %}
</div>
</div>

<!-- Champ pour le nom d'utilisateur -->
..... suit de code

```

CVPro

Accueil

Se connecter

Créer un compte

Créer un compte

Nom de famille

Prénom

User_name

Adresse email

Mot de passe

Confirmer le mot de passe

S'inscrire

Vous avez déjà un compte ? [Se connecter](#)

© 2025 Application de gestion des utilisateurs

L'objectif est de collecter les données saisies par l'utilisateur pour créer un nouveau compte. Ces données sont envoyées au serveur via la méthode "POST".

Vérification des erreurs : La validité des données saisies est vérifiée . En cas d'erreurs, des messages sont affichés pour indiquer à l'utilisateur ce qu'il doit corriger.

Protection contre les attaques CSRF : La balise `{{ form.hidden_tag() }}` est utilisée pour protéger le formulaire contre les attaques **CSRF** (falsification de requêtes inter-sites).

Mise en forme : Une bibliothèque CSS est utilisée (comme `form-control` et `invalid-feedback`) pour styliser les champs de manière à ce que le formulaire soit facile à lire et à utiliser.

Lien vers la page de connexion : En bas du formulaire, un lien est fourni pour les utilisateurs ayant déjà un compte afin qu'ils puissent se connecter au lieu de créer un nouveau compte.

=> **En résumé,** l'objectif de ce code est de créer une interface utilisateur conviviale et sécurisée pour l'inscription d'un nouveau compte, tout en vérifiant la validité des données et en protégeant contre les menaces de sécurité.

★ Controller (Auth):

```
from flask import Blueprint, render_template, redirect, url_for, flash, request
from flask_login import login_user, logout_user, login_required, current_user
from app import db
from app.models.user import User
from app.forms.auth import LoginForm, RegistrationForm

auth_bp = Blueprint('auth', __name__, url_prefix='/auth')

@auth_bp.route('/register', methods=['GET', 'POST'])
def register():
    # Vérification si l'utilisateur est déjà connecté et redirection vers la page de résumé
    if current_user.is_authenticated:
        return redirect(url_for('resume.view_resume'))
    form = RegistrationForm()
    if form.validate_on_submit(): # Validation du formulaire lors de la soumission
        # Création d'un nouvel utilisateur avec les données du formulaire
        user = User(
            username=form.username.data,
            email=form.email.data,
            first_name=form.first_name.data,
            last_name=form.last_name.data
        )
        user.set_password(form.password.data) # Hachage et stockage du mot de passe
        db.session.add(user)
        db.session.commit()
        flash('Enregistrement réussi ! Vous pouvez maintenant vous connecter.', 'success')
        return redirect(url_for('auth.login'))
    return render_template('auth/register.html', form=form)

@auth_bp.route('/login', methods=['GET', 'POST'])
def login():
    # Vérification si l'utilisateur est déjà connecté et redirection vers la page de résumé
    if current_user.is_authenticated:
        return redirect(url_for('resume.view_resume'))

    form = LoginForm()
    if form.validate_on_submit(): # Validation du formulaire lors de la soumission
        # Recherche de l'utilisateur en fonction de l'email
        user = User.query.filter_by(email=form.email.data).first()
        if user and user.check_password(form.password.data): # Vérification du mot de passe
            login_user(user, remember=form.remember.data) # Connexion de l'utilisateur
            next_page = request.args.get('next') # Récupération de la page suivante à rediriger
            return redirect(next_page or url_for('resume.view_resume')) # Redirection vers
la page de résumé ou la page suivante
        else:
            flash('Échec de la connexion. Veuillez vérifier votre email et votre mot de passe.', 'danger')
    return render_template('auth/login.html', form=form)

@auth_bp.route('/logout')
@login_required
def logout():
    # Déconnexion de l'utilisateur
    logout_user()
    flash('Déconnexion réussie.', 'info')
    return redirect(url_for('auth.login')) # Redirection vers la page de connexion
```

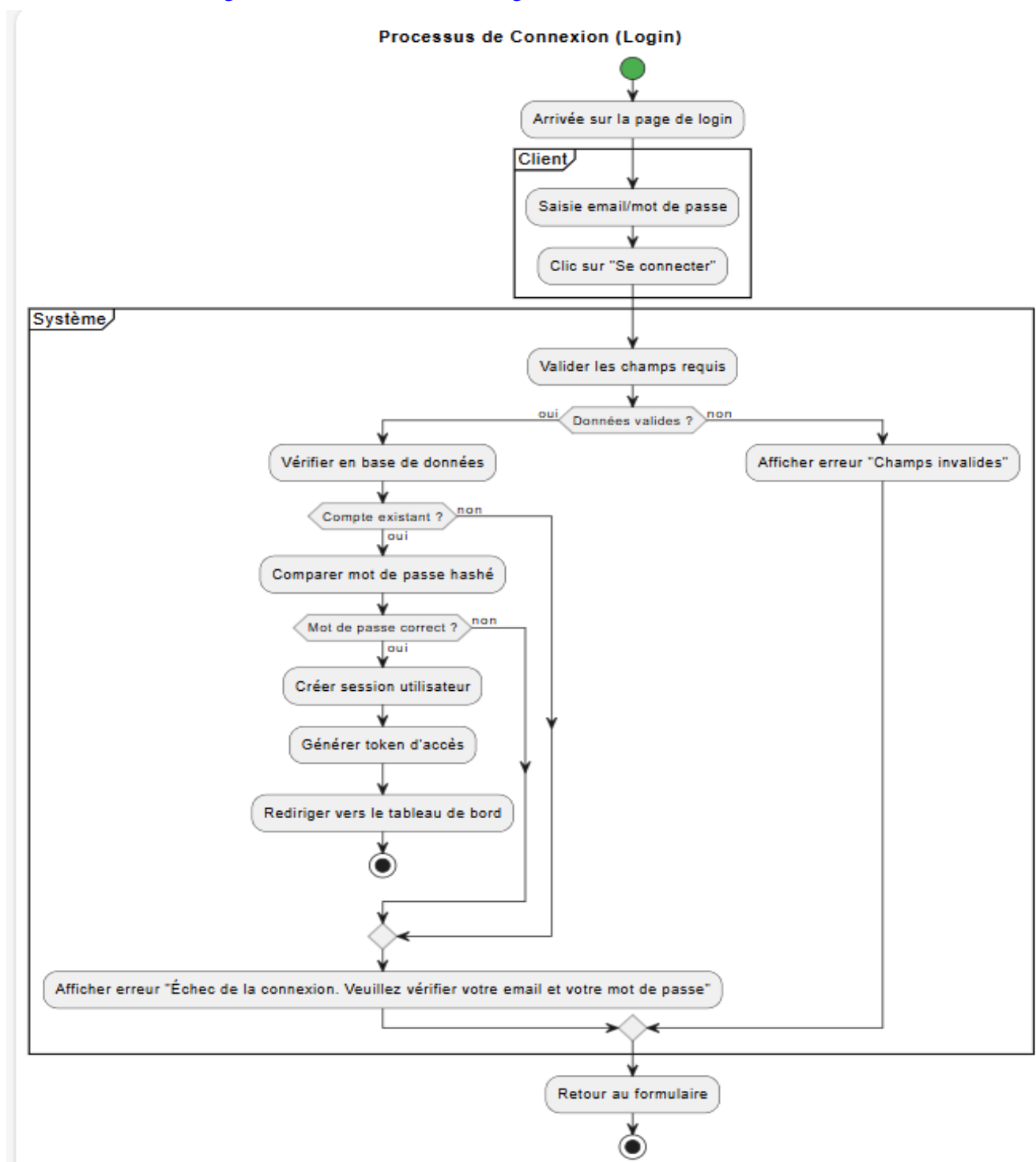
Ce code définit un blueprint Flask pour la gestion de l'authentification des utilisateurs. Il contient trois routes principales :

/register : Permet aux utilisateurs de s'inscrire. Si l'utilisateur est déjà connecté, il est redirigé vers la page de résumé. Lors de la soumission du formulaire, un nouvel utilisateur est créé et ajouté à la base de données après validation du formulaire

/login : Permet à un utilisateur de se connecter. Si l'utilisateur est déjà connecté, il est redirigé vers la page de résumé. Après validation du formulaire de connexion, le mot de passe de l'utilisateur est vérifié, et s'il est correct, l'utilisateur est connecté.

/logout : Permet à l'utilisateur de se déconnecter. Après la déconnexion, il est redirigé vers la page de connexion.

★ Plant UML : Diagramme d'Activité (login)



```
{% extends 'base.html' %}
{% block title %}Connexion{% endblock %}
{% block content %}
<div class="container">
  <div class="card">
    <h1 class="card-title">Connexion</h1>

    <!-- Formulaire de connexion -->
    <form method="post" class="auth-form">
      <!-- Inclusion du champ caché CSRF pour la sécurité du formulaire -->
      {{ form.hidden_tag() }}

      <!-- Champ pour l'email -->
      <div class="form-group">
        {{ form.email.label(class="form-label") }}
        {{ form.email(class="form-control") }}
        <!-- Affichage des erreurs de validation de l'email -->
        {% if form.email.errors %}
          <div class="invalid-feedback">
            {% for error in form.email.errors %}
              <span>{{ error }}</span>
            {% endfor %}
          </div>
        {% endif %}
      </div>

      <!-- Champ pour le mot de passe -->
    </form>
  </div>
</div>

... SUITE DE CODE...
```

En résumé, ce modèle fournit une interface pour que les utilisateurs puissent **se connecter** à leur compte, avec une gestion des erreurs et une protection de sécurité CSRF. (même principe comme **register**)

CVPro

Accueil

Se connecter

Créer un compte

Connexion

Adresse email

Mot de passe

☐ Rester connecté

Se connecter

Pas encore de compte ? [Inscrivez-vous maintenant](#)

© 2025 Application de gestion des utilisateurs

SOS: Je souhaite également d'autres pages qui fonctionnent avec la même logique, comme l'ajout d'expériences, l'ajout de langues et l'ajout d'éducation.

page home :

CVPro

Accueil

Se connecter

Créer un compte

Créez votre CV

Créez facilement un CV professionnel en quelques minutes.

Fonctionnalités

Facile à utiliser

Notre site propose une interface simple et intuitive pour vous aider à créer votre CV en quelques clics.

Modèles préconçus

Choisissez parmi une large gamme de modèles professionnels adaptés à vos besoins.

Prêt à imprimer

Vous pouvez télécharger votre CV au format PDF, prêt à être imprimé.

Commencez maintenant

Créez votre CV facilement en vous inscrivant ou en vous connectant.

Créer un compte

Se connecter

page : cree CV :

Créer le CV

Titre du CV

Ex : Développeur Full Stack, Marketing Digital, Développeur Web, Chef de Produit

Résumé

Resume profil..

Sauvegarder

Annuler

page Modifier profile :

CVPro

CV

Profil

Accueil

marouan_99

Modifier le profil

Nom d'utilisateur

marouan_99

Adresse email

marwanben@gmail.com

Prénom

Benali

Nom de famille

Marouan

Numéro de téléphone

Adresse

À propos de moi

Image de profil

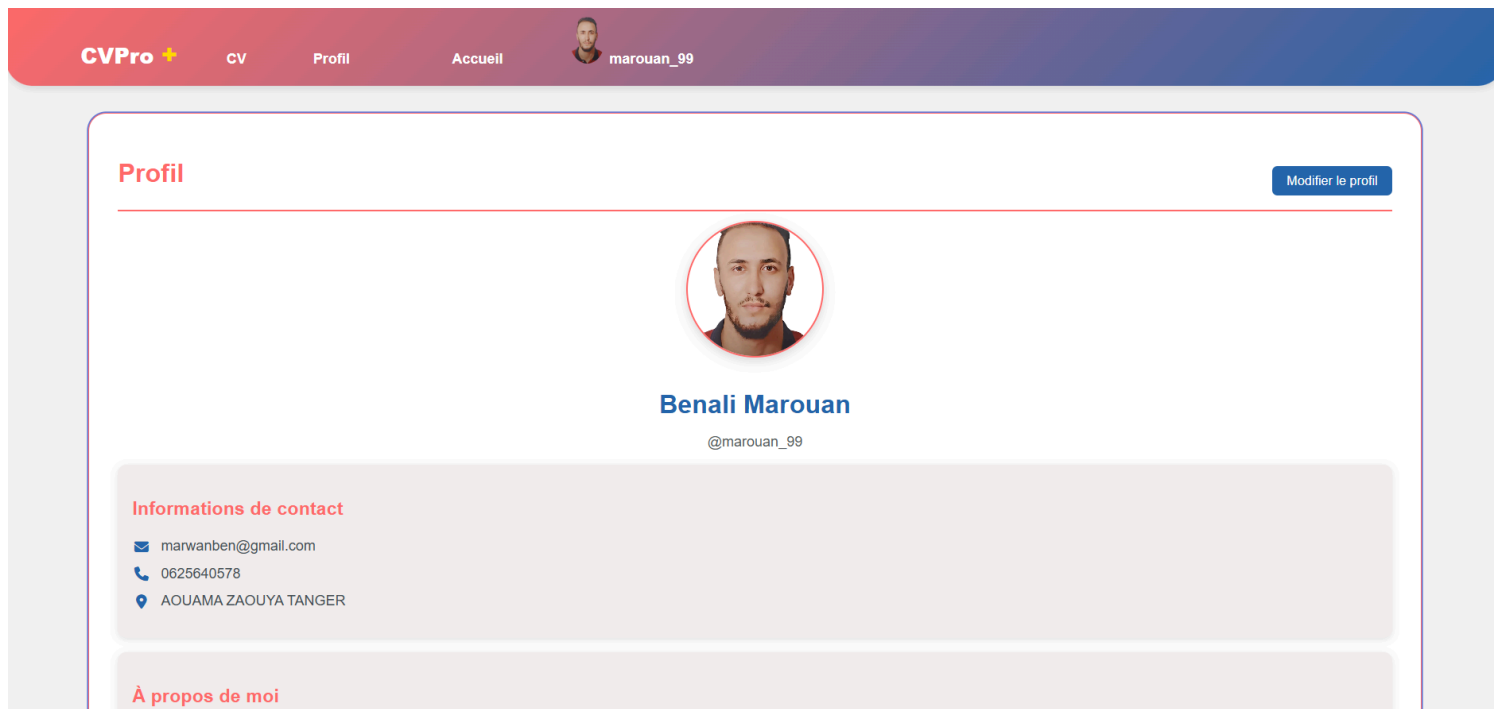
Choisir un fichier

Aucun fichier n'a été sélectionné

Site web

https://

page Affichage information :



Code de page profil/view.html

```
{% extends 'base.html' %}
{% block title %}Profil{% endblock %}
{% block content %}
<div class="container">
  <!-- Card Container for Profile -->
  <div class="card">
    <div class="profile-header">
      <h1 class="card-title">Profil</h1>
      <!-- Link to modify the profile -->
      <a href="{{ url_for('profile.profile') }}" class="btn btn-primary">Modifier le profil</a>
    </div>

    <div class="profile-content">
      <!-- Profile Image Section -->
      <div class="profile-image-container">
        {% if current_user.profile_image %}
          <!-- Display the user's profile image if it exists -->
          
        {% else %}
          <!-- Display a placeholder icon if the user doesn't have a profile image -->
          <div class="profile-image-placeholder">
            <i class="fas fa-user"></i>
          </div>
        {% endif %}
      </div>

      <div class="profile-details">
        <!-- User's Full Name -->
        <h2>{{ current_user.first_name }} {{ current_user.last_name }}</h2>
        <!-- Display the username with the @ symbol -->
        <p class="username">@{{ current_user.username }}</p>

        <div class="profile-section">
```

```

<h3>Informations de contact</h3>
<ul class="profile-info-list">
  <!-- Display the user's email address -->
  <li><i class="fas fa-envelope"></i> {{ current_user.email }}</li>
  {% if current_user.phone %}
  <!-- Display the user's phone number if available -->
  <li><i class="fas fa-phone"></i> {{ current_user.phone }}</li>
  {% endif %}
  {% if current_user.address %}
  <!-- Display the user's address if available -->
  <li><i class="fas fa-map-marker-alt"></i> {{ current_user.address }}</li>
  {% endif %}
</ul>
</div>

{% if current_user.bio %}
<!-- Display the user's bio if available -->
<div class="profile-section">
  <h3>À propos de moi</h3>
  <p>{{ current_user.bio }}</p>
</div>
{% endif %}

<div class="profile-section">
  <h3>Liens</h3>
  <ul class="profile-links">
    {% if current_user.website %}
    <!-- Display the user's website if available -->
    <li><a href="{{ current_user.website }}" target="_blank"><i class="fas fa-globe"></i> Site Web</a></li>
    {% endif %}
    {% if current_user.linkedin %}
    <!-- Display the user's LinkedIn profile if available -->
    <li><a href="{{ current_user.linkedin }}" target="_blank"><i class="fab fa-linkedin"></i> LinkedIn</a></li>
    {% endif %}
    {% if current_user.github %}
    <!-- Display the user's GitHub profile if available -->
    <li><a href="{{ current_user.github }}" target="_blank"><i class="fab fa-github"></i> GitHub</a></li>
    {% endif %}
    {% if current_user.twitter %}
    <!-- Display the user's Twitter profile if available -->
    <li><a href="{{ current_user.twitter }}" target="_blank"><i class="fab fa-twitter"></i> Twitter</a></li>
    {% endif %}
  </ul>
</div>
</div>
</div>

<div class="profile-actions">
  <!-- Button to view the user's resume -->
  <a href="{{ url_for('resume.view_resume') }}" class="btn btn-primary">Voir le CV</a>
</div>
</div>
{% endblock %}

```

Ce code est un modèle HTML pour afficher un profil utilisateur dans une application Flask.

Voici un résumé de ses composants :

- **Héritage du modèle de base** : Le modèle étend **base.html**, et les blocs de contenu spécifiques sont définis pour le titre de la page et le contenu.
- **En-tête du profil** : Le titre "Profil" est affiché avec un bouton pour modifier le profil, redirigeant vers une page dédiée à la modification.
- **Image de profil** : Si l'utilisateur a une image de profil, elle est affichée. Sinon, un icône de remplacement (utilisateur) est affiché.
- **Informations personnelles** :
 - Le nom complet de l'utilisateur, son nom d'utilisateur (précédé du symbole @), et son adresse email sont affichés.
 - Si disponibles, d'autres informations telles que le numéro de téléphone et l'adresse de l'utilisateur sont également affichées.
 - Si l'utilisateur a une biographie, elle est présentée sous la section "À propos de moi"
- **Liens externes** : Des liens vers les profils de l'utilisateur sur des plateformes externes (par exemple, site web, LinkedIn, GitHub, Twitter) sont affichés si ces informations sont disponibles.
- **Actions** : Un bouton permet à l'utilisateur de voir son CV.

⇒ En résumé, cette page affiche les informations de profil d'un utilisateur, y compris ses détails personnels, ses liens sociaux, et un accès pour modifier le profil ou consulter le CV.


page résumer/view.html pour voir le CV

CVPro +


CV

Profil

Accueil

 marouan_99

Développeur



Benali Marouan

✉ marwanben@gmail.com

☎ 0625640578

📍 AOUMAMA ZAOUYA TANGER

🌐 MAROUAN-BEN-ALI.COM

🌐 LINKEDIN-MAROUANBENALI.COM

🌐 GITHUB-MAROUAN_BENALI.COM

🐦 MAROUANBENALI.MA

Résumé

full stack devlopere marouan ben ali informatique par exemple

Éducation

Aucune éducation ajoutée. Ajoutez-en une maintenant

Expérience Professionnelle

les pages d'ajouter - modifier : (même concepte comme login & register)

Education :

CVPro+CVProilAccueilmarouan_99

Ajouter Éducation

Établissement

Université Mohammed V, Université Paris-Saclay, ISG Tunis...

Diplôme

Licence en Informatique, Master en Génie Logiciel, Doctorat en IA...

Domaine d'études

Informatique, Génie Civil, Sciences et Technologies...

Date de début

jj/mm/aaaa

Date de fin

jj/mm/aaaa

Description

SauvegarderAnnuler

Experience :

CVPro+CVProilAccueilmarouan_99

Ajouter Expérience

Entreprise

Ex : Google, Microsoft, Startup X...

Poste

Ex : Ingénieur Logiciel, Responsable Marketing, Développeur Web...

Lieu

Ex : Casablanca, Paris, Tanger...

Date de début

jj/mm/aaaa

Actuellement employé

Date de fin

jj/mm/aaaa

Description

SauvegarderAnnuler

Langage :

CVPro+CVProilAccueilmarouan_99

Ajouter Langue

Langue

Ex : Français, Anglais, Espagnol, Arabe...

Niveau de compétence

Débutant

SauvegarderAnnuler

© 2025 Application de gestion des utilisateurs


competence :

CVPro

CV

Profil

Accueil

 marouan_99

Ajouter une compétence

Compétence

Ex : Développement Web, Gestion de projets, Analyse de données, Leadership...

Niveau

Débutant




Débutant

Intermédiaire

Bon

Avancé

Expert




projet :

CVPro

CV

Profil

Accueil

 marouan_99

Ajouter un projet

Titre du projet

Ex : Application de Gestion, Site Web e-commerce...

Description

URL du projet

https://

Date de début

jj/mm/aaaa

Date de fin

jj/mm/aaaa

Sauvegarder


Annuler

resultat de CV :


Développeur


Modifier le CV


Imprimer





Benali Marouan


 marwanben@gmail.com


 0625640578

 AOUMAMA ZAOUYA TANGER

 MAROUAN-BEN-ALI.COM

 LINKEDIN.MAROUAN.BENALI.COM

 GITHUN.MAROUAN.BENALI.COM

 MAROUANBENALI.MA

Résumé

full stack devlopere marouan ben ali informatique par exemple

Éducation

Ajouter

email sont affichés. en Informations personnelles :

email sont affichés. : 2002-03 - 2009-02

Modifier

Supprimer

Le nom complet de l'utilisateur, son nom d'utilisateur (précédé du symbole @), et son adresse email sont affichés. Si disponibles, d'autres informations telles que le numéro de téléphone et l'adresse de l'utilisateur sont également affichées.

Expérience Professionnelle

Ajouter

son adresse email

imprimante :

Imprimer

Total : 2 pages

Imprimante

Enregistre au format PDF

Pages

Tout

Pages impaires uniquement

Pages paires uniquement

par exemple : 1-5, 8, 11-13

Paramètres supplémentaires

Imprimer à l'aide de la boîte de dialogue système...

Enregistrer

Annuler

Développer

Benali Marouan

marwanben@gmail.com

0625640578

AOUAMA ZAOUYA TANGER

marouan-ben-ali.com

Linkden/Marouanbenali.com

github/marouan_benali.com

marouanbenali.ma

Résumé

full stack devlopere marouan ben ali informatique par exemple

2002-02 - 2004-02

Le nom complet de l'utilisateur, son nom d'utilisateur (précédé du symbole @), et son adresse email sont affichés. Le nom complet de l'utilisateur, son nom d'utilisateur (précédé du symbole @), et son adresse email sont affichés. Le nom complet de l'utilisateur, son nom d'utilisateur (précédé du symbole @) et son adresse email sont affichés. Si disponibles, d'autres informations telles que le numéro de téléphone et l'adresse de

Phase 3 : Tests- Tests de sécurité et de performance.

- Tests de toutes les fonctionnalités (ajout, modification et suppression de données).

Fin du rapport :

Nous n'avons pas expliqué les codes des éléments qui ont été ajoutés car ils ne nous étaient pas demandés, et l'explication et la clarification seront développées. Par conséquent, notre explication et le rapport que nous vous présentons se sont limités à l'explication des données d'inscription et de connexion. Quant au code, il contient des commentaires sur presque toutes les parties.

الحمد لله رب العالمين