Projet de développement applicatif.

Langage de modélisation : UML

Approche : Modèle en V

* **Besoin**.
* **Spécifications fonctionnelles**.
* **Spécifications techniques**.
* **Codage**.
* **Tests de fonctionnement**
* **Validation**
* **Mise en production**

++++++++++++++++++++++++++

**Besoins**

La direction de l’entreprise recherche un moyen plus efficace pour motiver ses employés. Ceci sur la base l’assiduité de ces derniers sur le lieu de travail. Il sera vérifié des critères tels que :

* + La durée passée en entreprise ;
  + La régularité en entreprise de l’employé ;
  + La ponctualité des employés.

Chacun des critères ci-dessus octroie une certaine note au personnel. Il sera alors fait un cumul de ces notes sur des intervalles de temps laissé à la discrétion de l’entreprise. La valeur de ces cumuls permettra de savoir si un employé :

* Soit écopera d’un rappel à l’ordre ;
* Soit recevra une motivation ;
* Soit bénéficiera d’un encouragement ;

Cette évaluation aura alors plus ou moins des répercutions sur les primes d’efficacité octroyé à l’employé.

**Spécifications fonctionnelles**

**Le système applicatif qui sera développé permettra :**

* + **A l’employer de s’identifier au moyen de son empreinte digital avant d’accéder aux locaux de l’entreprise**
  + **A l’employer de s’identifier également au moyen son empreinte digitale à chaque fois qu’il sort des locaux de l’entreprise.**
  + **Ces différentes navettes de l’employé doivent être enregistrées dans une base de données pour des traitements ultérieurs. Et les prises de décisions administratives.**

**Spécifications techniques**

Une analyse des besoins de l’entreprise, ainsi que des spécifications techniques ont conduit à l’élaboration d’un diagramme de classe UML. Comme le présente la figure suivante :

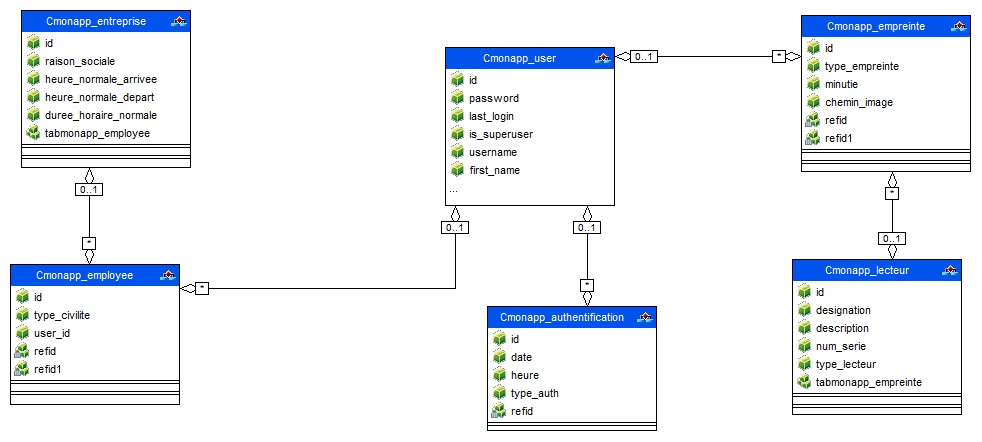


Figure N°1 : Diagramme de classe

Présentation des classes du diagramme.

|  |  |
| --- | --- |
| Classe | Description |
| Authentification | Recueille la date ainsi que l’heure de demande d’accès. Soit en entrée ou en sortie. |
| Employee | Contient la civilité de l’employer ainsi que son entreprise dans le cadre d’un building qui hébergerait plus d’une entreprise. |
| Empreinte | Dispose des informations des empreintes digitales des employés. |
| Entreprise | Héberge les informations sur l’entreprise et sa description |
| Lecteur | Récupère les informations sur les différents lecteurs utilisés pour le traitement des empreintes. |
| User | Contient les informations personnelles de l’employé. |

Il est à noter que de son importance, ce diagramme sert de tremplin pour l’étape suivante, celle du codage.

**Spécifications techniques**

**Il s’agit ici de** la phase de réalisation à proprement parler, pendant laquelle nous développons l’application.

LES OUTILS LOGICIELS

* Python 3.11 : Langage de programmation Backend
* Django 4.2.1 : Framework de développement web du langage python.
* Html, Javascript, css : Langages de programmation Frontend
* BootStrap : Framework de développement Frontend
* PIP 23 : Gestionnaire de modules et librairies python.
* SQLite 3 : Système léger de gestion de base de données.
* Vim 9.0, Notepad++ : Editeurs de code
* DP\_UareU\_WSDK\_223 : Pilote SDK facilitant la reconnaissance du « lecteur d’empreinte digitale » par le système d’exploitation hôte.

LES OUTILS MATERIELS

* Ordinateur portable Hp ProBook 450 G4, Ram 8Go, Core i5, 7th Gen : utilisé pour la programmation en local de l’application.
* Lecteur d’empreinte digitale « DigitaPersona U.are.U 4500 » : utilisé pour la lecture de l’empreinte digitale. Il s’agit du produit de la firme LotHill Solutions.

|  |
| --- |
| D:\Data\helios\Downloads\zzz-fingerprints images\4500-frontview-hi-res-small.png |
| Figure 2 : Lecteur d’empreinte digitale « DigitaPersona U.are.U 4500 » |

QUELQUES CODES DE L’APPLICATION

**(En cours de rédaction…)**

**Fonction checkFingerPrintMatch**

**Cette fonction fait une comparaison entre deux images: celle enregistrée dans la base de données à l'enrôlement, puis celle récupérée depuis le lecteur sur l’interface d'entrée ou à de sortie.**

|  |
| --- |
| **import os;**  **import cv2;**  **def checkFingerPrintMatch(original\_img\_folder, sample):**    **#Initialisation de variables**  **best\_maching\_score = 0;**  **filename = None;**  **image = None;**  **kp\_orig, kp\_samp, mp = None, None, None;**  **#Parcours des images originales pour comparer à celle du demandeur d'accès.**  **for file in [file for file in os.listdir(original\_img\_folder)]:**  **#Récupération de l'image en base par concaténation du dossier parent et du nom de l'image.**  **fingerprint\_image = cv2.imread(original\_img\_folder + "/" + file);**    **#Scale invariant Feature Transform (SIFT), permet d'extraire les points clés des images.**  **sift = cv2.SIFT\_create(); #Création de l'objet sift**  **#extraction des points clés de l'image en base**  **keypoints\_1, descriptors\_1 = sift.detectAndCompute(sample, None);**  **#extraction des points clés de l'image provenant du lecteur**  **keypoints\_2, descriptors\_2 = sift.detectAndCompute(fingerprint\_image, None);**    **#Pour ces points clés, trouvons les correspondances**  **#Récupérons dans "matches", l'ensemble de paramètres pour faire correspondre les points clés entre l'image 'sample' et 'fingerprint\_image'.**  **matches = cv2.FlannBasedMatcher({'algorithm':1, 'trees':10},**  **{}).knnMatch(descriptors\_1, descriptors\_2, k=2);**    **#Initialisons une liste vide pour stocker les points correspondants**  **match\_points = [];**    **#Vérifions dans la boucle, si le rapport de distance entre les deux matches est inférieur à un certain seuil. Alors cette correspondance est conservée dans match\_points.**  **for p, q in matches: #macthing 1:p , maching 2:q**  **if p.distance < 0.1 \* q.distance:**  **match\_points.append(p);**    **#Nous allons Comparer le nombre de points clés dans l'image 'sample' et 'fingerprint\_image'...**  **# pour déterminer le nombre minimum de points clés, à mettre dans "keypoints"**  **keypoints = 0;**  **if len(keypoints\_1) < len(keypoints\_2):**  **keypoints = len(keypoints\_1);**  **else:**  **keypoints = len(keypoints\_2);**    **#Si le score de correspondance est supérieur au best\_maching\_score**  **if len(match\_points) / keypoints \* 100 > best\_maching\_score:**  **best\_score = len(match\_points) / keypoints \* 100; #best\_score prend la valeur de la correspondance**  **filename = file; #le sample approximative le mieux correspondant est récupéré**  **image = fingerprint\_image;**  **kp\_orig, kp\_samp, mp = keypoints\_1, keypoints\_2, match\_points;**    **if filename == None: #Si le fichier est vide alors, recherche infructueuse**  **return None; #renvoie de None;**  **pass;**  **else: #Sinon**  **result = cv2.drawMatches(sample, kp\_orig, image, kp\_samp, mp, None);**  **result = cv2.resize(result, None, fx=4, fy=4);**  **resultat = {};**  **resultat['filename'] = filename;**  **resultat['score'] = best\_score;**  **return resultat; #Renvoie du nom du sample et du meilleur score.**  **pass;** |

**Cette fonction renvoie None si l'empreinte recherchée n'est pas retrouvée dans les échantillons. Mais si une correspondance est trouvée entre l'empreinte du lecteur et celle en base alors elle renvoie le nom du sample (fichier en base) et le meilleur score.**

**Les lignes suivantes présentent la liste des fonctions intervenant depuis la préparation de l’environnement, passant la sélection du lecteur, ainsi que la récupération de son numéro de série, jusqu’à la lecture et l’enregistrement de l’empreinte d’un doigt posé sur le lecteur.**

|  |
| --- |
| **'''**  **initJvm(chemin\_machine\_virtuelle\_java/Vide): renvoie None.**  **Mais démarre le JVM pour permettre son usage par jpype.**  **'''**  **def initJvm(jvm\_path=None):**  **if jpype.isJVMStarted():**  **pass**  **else:**  **jpype.startJVM(jpype.getDefaultJVMPath())**  **jpype.addClassPath('./dpuareu.jar')**  **jpype.addClassPath('./JavaFXSwingFingerprintApplication.jar')**  **jvmPath = jpype.getDefaultJVMPath()**  **path = jpype.getClassPath()**  **return None** |

|  |
| --- |
| **'''**  **selectedReader(chemin\_jvm/Vide): renvoie le lecteur sélectionné**  **'''**  **def selectedReader(jvm\_path=None):**  **initJvm(jvm\_path)**  **#Selection d'un lecteur**  **UareUGlobal = jpype.JClass("com.digitalpersona.uareu.UareUGlobal")**  **selectionObject = JClass('Selection')**  **selObj = selectionObject.Select(UareUGlobal.GetReaderCollection())**  **return selObj** |

|  |
| --- |
| **'''**  **readerSerialNumber(): renvoie le numéro de série du lecteur d'empreinte digitale.**  **'''**  **def readerSerialNumber():**  **sel\_obj = selectedReader()**  **serial\_number = None**  **if(sel\_obj is None):**  **print("Erreur Serial-Number")**  **else:**  **serial\_number = sel\_obj.GetDescription().serial\_number**  **return serial\_number** |

|  |
| --- |
| **'''**  **get\_fingerprints\_image\_directory(un\_dossier\_parent, un\_sous\_dossier): renvoie le**  **répertoire où sont stockées les images de l'empreinte.**  **'''**  **def get\_fingerprints\_image\_directory(main\_dir, sub\_dir):**  **filedirname = ''**  **absdirname = os.path.dirname(\_\_file\_\_)**    **#Remplacement des antislash (archi Windows) s'il en trouve par des slashs**  **tempname = absdirname.replace('\\', '//')**  **if sub\_dir is not None:**  **filedirname = tempname + '//media//' + main\_dir + '//' + sub\_dir + '//'**  **else:**  **filedirname = tempname + '//media//' + main\_dir + '//'**  **return filedirname** |

|  |
| --- |
| **'''**  **loadFingerprint(nom\_utilisateur, type\_de\_doigt): enregistre l'image de l'empreinte digitale lu par le lecteur**  **et renvoie cette image.**  **'''**  **def loadFingerprint(username, fingertype):**  **initJvm()**  **#Concaténation "username"\_"fingertype" pour créer le "fingername"**  **fingername = "{}\_{}".format(username, fingertype)**  **#Création de l'instance de la classe Java 'Selection' à l'aide de JClass**  **selectionObject = JClass('Selection')**  **#Création de l'instance de la classe Java 'Capture' à l'aide de JClass**  **captureObj = JClass('Capture')**  **#Création de l'instance de la classe Java 'ImagePanel' à l'aide de JClass**  **imgPanObj = JClass('ImagePanel')**  **#Création de l'instance de la classe Java 'com.digitalpersona.uareu.UareUGlobal' à l'aide de JClass**  **UareUGlobal = JClass("com.digitalpersona.uareu.UareUGlobal")**  **#Appelle la méthode 'Select' pour obtenir un objet lecteur à partir de 'UareUGlobal.GetReaderCollection()'.**  **readerObj = selectionObject.Select(UareUGlobal.GetReaderCollection())**  **if readerObj is not None: #Si le lecteur a été sélectionné**  **'''**  **Appelle la méthode 'Run' de 'captureObj' pour commencer à capturer des ...**  **images d'empreintes digitales à partir du lecteur sélectionné.**  **'''**  **captureObj.Run(readerObj, 0)**  **if captureObj.getM\_captureResult() is not None:**    **# Récupération du chemin du dossier dans lequel seront stockées les empreintes digitales.**  **xpath = get\_fingerprints\_image\_directory('fingerprints', None)**  **'''**  **Opération d'enregistrement l'image du doigt posé sur le lecteur d'empreinte digitale**  **"res" prendra True/False selon la réussite ou l'échec de cette opération**  **'''**  **imgPan = imgPanObj()**  **res = imgPan.getM\_Image(captureObj.getM\_captureResult().image, xpath, fingername)**  **if res == True: #Si un contenu a été récupéré du lecteur.**  **# Reconstruction du chemin absolu du fichier de l'image d'empreinte enregistrée**  **file = None**  **absdirname = os.path.dirname(\_\_file\_\_)**  **filepath = os.path.join(absdirname,'media', 'fingerprints', 'pngfiles', fingername+'.png')**    **if os.path.exists(filepath): #Si ce chemin existe**  **return filepath #Alors il est renvoyé pour d'autre traitement**  **else:**  **print('Le fichier n\'existe pas')**  **return None**  **else:**  **print('Erreur : Aucune empreinte enregistrée.')**  **return None** |

QUELQUES INTERFACES GRAPHIQUES

**(En cours de rédaction…)**

**Tests de fonctionnement**

**Afin de s’assurer de livrer un produit opérationnel, l’équipe de développement a intégré les tests de fonctionnement à la phase de développement. Permettant d’ajuster les comportements du produit au fur et à mesure du codage. Afin de corriger les éventuelles erreurs et rendre un produit répondant aux aspirations du client.**

**Validation**

**L’application a été testée sur un échantillon des potentiels utilisateurs. Et en concertation avec le maître d’ouvrage, le produit logiciel a été validé conforme aux besoins exprimés.**

**Mise en production**

Après une dernière phase de test en collaboration avec nos clients, et avec leur accord le produit applicatif a été mis en production. A cette étape du projet nous avons déployé l’application en ligne pour l’usage restreint du client.

Le déploiement s’est réalisé chez l’hébergeur PYTHONANYWHERE.COM.

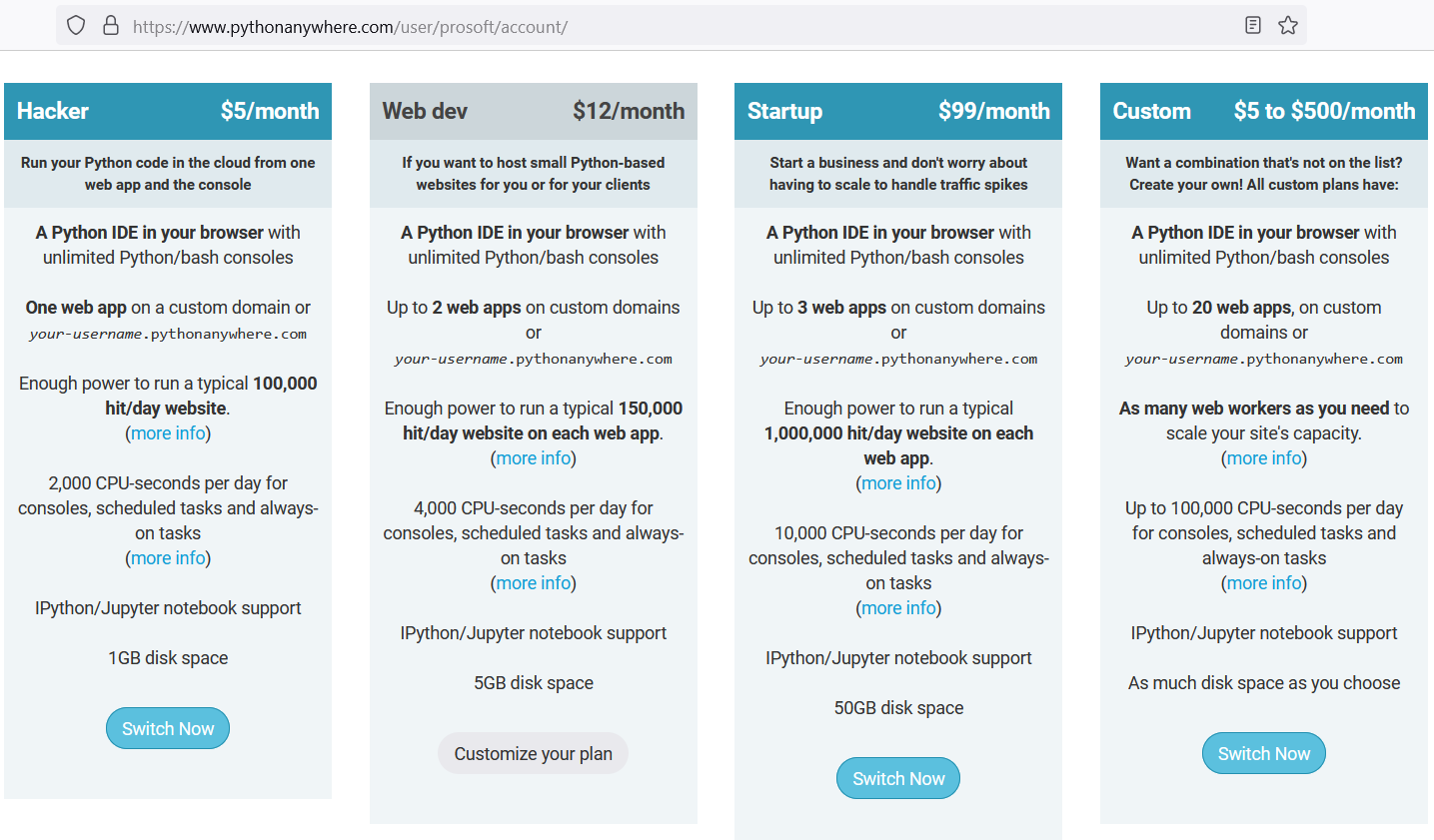


Figure 3 : Plans d’hébergement fournis par pythonanywhere.

Sur cette plateforme, un stockage a été acheté avec le plan « WebDev » pour les besoins de la mise en ligne de notre application. Rendant ainsi à la date de la redaction de ce document, l’application disponible à l’addresse <http://aristide-prosoft.pythonanywhere.com/> tout comme le présente la figure suivante :

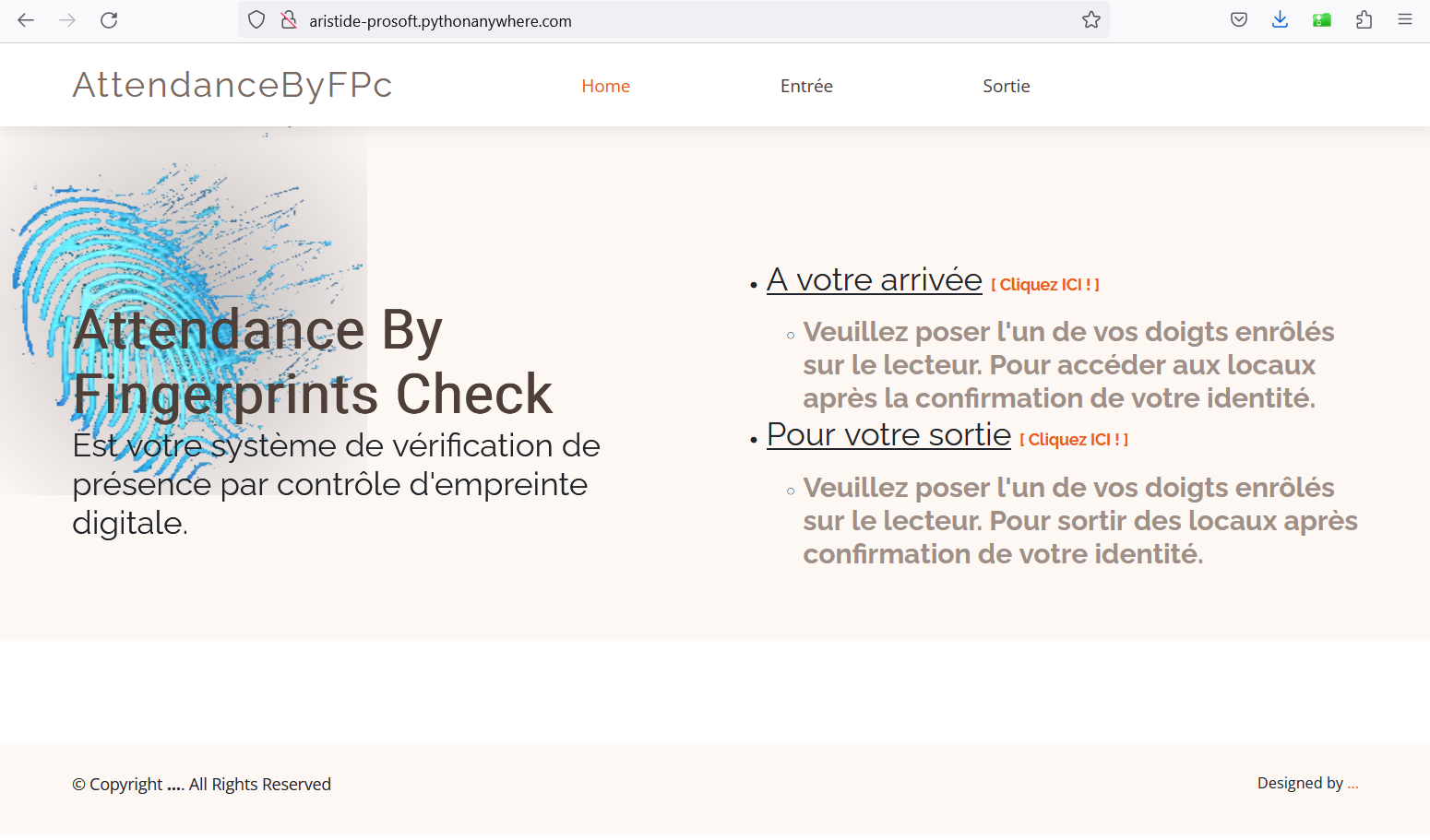


Figure N°X Page d’accueil de l’application après déploiement.