Rapport : Machine Learning & Internet of Things

Système de surveillance d'un examen en ligne

Réalisé par : Traian BATOG et Aghilas KESSAÏ

Etudiants en Master 2 EID² à l'USPN

Encadré par : Massinissa HAMIDI et Aomar OSMANI

Table des matières

I- Introduction

Dans le cadre du cours de Machine Learning & Internet of Things à l'Université Sorbonne Paris Nord en Master 2 EID², nous avons réalisé un système informatique permettant de détecter les tentatives de triche lors d'un examen en ligne. Etant donné que le phénomène des examens en ligne s'est répandu très rapidement au cours de l'année 2020, il est important de pouvoir vérifier à distance que le candidat ne triche pas.

Ainsi, le but du projet était de détecter les tentatives de triche en utilisant différents moyens. D'abord, on utilise la caméra de l'ordinateur où se déroule l'examen (cette caméra filme le visage du candidat). Le candidat est aussi équipé de 2 téléphones : un téléphone B fixé sur son bras et un téléphone T fixé sur sa tête. Le téléphone B nous permet donc d'analyser les données liées aux mouvements du bras du candidat grâce aux différents capteurs du téléphone (l'accéléromètre en particulier). Le téléphone T permet de filmer l'ordinateur du candidat ainsi que ses bras, les objets autour de lui et la pièce dans laquelle il se trouve.

Grâce à ses différents moyens, nous avons élaboré un système informatique permettant de détecter les tentatives de fraude. Notre système sera détaillé dans la suite de ce rapport.

II- Etat de l'art

A) L'aspect commercial

Avant de se lancer dans un tel projet informatique, il est important de se renseigner sur les différentes solutions présentes sur le marché ainsi que sur l'avancement technique liée à cette problématique.

D'abord nous allons nous plonger dans l'aspect commercial et voir quelles solutions sont mises à disposition des établissements d'enseignements.

En cherchant sur internet, on retrouve une multitude de solutions de 'proctoring' (un terme anglais signifiant : surveiller des personnes faisant un examen et s'assurer qu'ils ne trichent pas).

Intéressons-nous d'abord aux solutions utilisées par de grandes universités tel que Sorbonne Université, l'Université d'Amsterdam ou Polytechnique Montréal : ces 3 universités se retrouvent sur le site de ProctorExam. D'après la démo de **ProctorExam**, l'étudiant doit montrer sa pièce d'identité avant de débuter et son visage pendant la durée de l'examen. Un téléphone est aussi placé derrière le candidat afin d'avoir une vue de la pièce sur laquelle il se trouve. Cependant, on ne trouve pas de détails concernant la façon dont la détection de triche est faite, mais nous avons contacté l'assistance technique de Proctor Exam, **qui nous a indiqué que la reconnaissance faciale ainsi que la surveillance de la pièce est faite de manière continue durant tout le déroulement de l'examen. A la fin de l'examen, le professeur reçoit un rapport avec tous les incidents relevés par ProctorExam.** Nous n'avons pas trouvé d'information concernant le prix de la solution, étant donné qu'il faut demander une démo et un devis afin d'avoir une réponse. Nous notons que cette solution nous semble très professionnelle, avec par exemple des assistants techniques sur le chat du site disponibles pour toute question.

On retrouve une autre solution de proctoring répandue particulièrement aux Etats-Unis et classée par plusieurs sites web comme étant l'une des meilleures solutions présentes sur le marché : ProctorU. Il est à noter que depuis le 28 février 2022, cette solution a fusionné avec Yardstick Assessment Solutions qui est une autre entreprise de proctoring et est devenue **Meazure Learning**. Cela laisse présager une croissance rapide du chiffre d'affaires de cette nouvelle entreprise. Cette solution permet 2 sortes de surveillance : automatisée grâce à l'IA ou/et faite par une personne. On peut lire sur le site 'Automated monitoring paired with human review and validation', cela signifie que **la vérification automatique grâce à l'IA accompagne la vérification humaine**. Le produit 'ProctorU Platform' permet par exemple d'offrir une aide supplémentaire à un professeur qui surveille ses élèves lors d'un examen en ligne. La plateforme détecte les comportements suspects et par la suite un surveillant certifié vérifie s'il y a une tentative de triche ou non. On retrouve de multiples options qu'on peut personnaliser avec différents produits disponibles. Ainsi l'équipe de Meazure Learning s'occupe de tous les processus liés à un examen en ligne (assister le professeur, créer un horaire d'examen,

surveillance, …). D'après un article sur le site de l'Université de Louisiane, un étudiant passant un examen en ligne avec cette solution doit s'acquitter d'une somme entre 15 et 30$. Ainsi, cette somme est assez élevée étant donné que l'IA offre une grande assistance lors de la détection de triche, ce qui réduit le temps de travail d'un employé qui vérifie les tentatives de triche. On suppose que lors d'un examen en ligne d'une heure et de 20 personnes, le prix est de 20 * 15 = 300 $. On suppose aussi que lors d'un examen d'une heure, il y a environ 2 évènements suspects par candidat, ce qui fait un total de 80 événements suspects pour cette session d'une heure. Or, si l'employé de l'entreprise prends 1 minute pour vérifier un incident et indiquer si c'est de la triche ou non, alors en 80 minutes il pourra vérifier l'intégralité des événements. Supposons que cet employé est payé à 20$/heure (ce qui est une estimation élevée), alors il sera payé 27 $ pour cette prestation. Ainsi, l'entreprise peut potentiellement avoir un bénéfice de 300 – 27 = 273 $ pour une heure. Cependant, ce bénéfice dépend de l'efficacité et des capacités du système d'IA développé par Meazure Learning.

Sur le marché français, on retrouve **Mereos** qui est d'après forbes.fr 'le discret leader français des examens en ligne'. Dorone Parienti qui est le co-fondateur de cette solution indique que sa solution permet le blocage du navigateur, la détection d'un double écran ainsi que l'enregistrement de la caméra du candidat et de son écran. 'Une équipe de surveillant de Mereos visionne ensuite les flux enregistrés et édite un rapport de fraude le cas échéant' déclare le co-fondateur. Nous avons contacté Mereos afin de nous renseigner sur le prix de la solution mais il ne la communique pas publiquement tant que nous n'avons pas demandé un devis. Parmi ses clients on retrouve l'ESAM Paris, le Haut Comité pour la Défense Civile ou TonAvenir. Tout comme ProctorU, cette solution est basée sur une IA mais aussi sur une vérification humaine. Une autre entreprise française, **TestWe**, qui est l'un des leaders du marché du secteur des examens en ligne d'après Le Figaro, a développé une solution de surveillance nommée ProctorWe. Cette solution prend en photo le candidat chaque intervalle de quelques secondes (on n'a pas une vidéo en continu), et à la fin on obtient un film du déroulement de l'examen. Parmi ses clients, on retrouve des universités brésiliennes ou le groupe BPCE.

En somme, une multitude de solutions se trouvent sur le marché, nous en avons sélectionné quelques-unes, qui ont été cités par plusieurs articles et qui ont le meilleur SEO. Les prix ne sont pas indiqués clairement sur le site web des solutions. Les solutions sont en général très personnalisables, ce qui à une influence sur le prix.

B) L'aspect technique

On se plonge à présent dans la recherche et l'avancée concernant les solutions de surveillance d'un examen en ligne. En faisant des recherches sur ResearchGate, on retrouve l'article '**Automated Online Exam Proctoring**', réalisé par Atoum, Chen, Liu et Hsu et Xioming Liu de l'Université de Michigan. Ils proposent un système de surveillance entièrement automatisé (contrairement aux solutions présentées précédemment qui requièrent une intervention humaine pour valider ou non le bon déroulement de l'examen). Le système est composé d'une caméra (de l'ordinateur), d'un microphone et d'une autre caméra qui filme la pièce. Il est basé sur la vérification de l'utilisateur, de la détection du texte et de la voix, du regard, de la vérification des fenêtres ouvertes sur l'ordinateur et de la détection de téléphone. Ils évaluent leur système grâce à un dataset généré par 24 personnes qui trichent de diverses manières pendant un examen en ligne. Ils démontrent un taux de réussite de leur modèle de 87%.

On retrouve un autre article : 'An intelligent system for online exam monitoring' publié en 2016 par Pratish, Narayanan et Bijalani. Ils utilisent la caméra de l'ordinateur ainsi que son micro et l'écran du candidat. Ils évaluent le degré de l'angle du visage du candidat (head pose) et se basent aussi sur le son ainsi que sur l'écran pour voir si le candidat triche ou non. D'après les chercheurs, leurs résultats d'expérience ont montré une meilleure efficacité que les systèmes existants déjà en 2016.

Il existe peu d'études qui ont réalisé un système de surveillance d'examen entier (de l'authentification, à la détection, …). Cependant, les systèmes réalisés par les chercheurs sont efficaces et coutent moins chers que les systèmes présents sur le marché : ils ne nécessitent pas d'intervention humaine.

III- Présentation générale de notre solution Examonitor

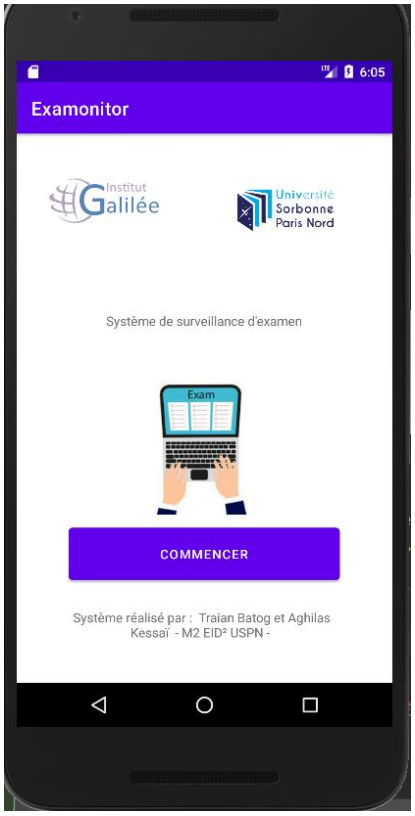
Notre système de surveillance d'examen en ligne est basé sur une application mobile et une application web. Le candidat doit placer un téléphone sur son bras et un autre téléphone sur sa tête. Le téléphone sur le bras permet d'utiliser le capteur de l'accéléromètre tandis que le téléphone de la tête permet de visualiser l'ordinateur du candidat ainsi que les objets qui sont autour de lui. L'application web accède à la caméra de l'ordinateur du candidat et procède à une reconnaissance faciale en continu tout au long de l'examen.

Sur l'application web, le candidat doit s'inscrire et se connecter. Il ne peut pas débuter l'examen si les 2 téléphones (du bras et de la tête) ne sont pas connectés à l'application. Après avoir placé et connecté les 2 téléphones, le candidat peut débuter l'examen sur l'interface web. Il y a 4 types de détections qui ont été mises en place : détection faciale (caméra de l'ordinateur), détection de son (de l'ordinateur), détection d'objets (du téléphone placé sur la tête) et détection du mouvement du bras (du téléphone placé sur le bras).

Notre système est donc composé d'une application mobile (Java) et d'une interface web réalisée avec Flask qui nous a permis de faire du développement web avec Python. Toutes les détections sont réalisées grâce à au programme Python. Plus de détails sont donnés dans les parties ci-dessous.

A) L'application mobile

Notre application mobile a été réalisée avec le langage Java en utilisant l'IDE Android Studio. Elle est simple et intuitive. Le candidat ouvre l'application et se retrouve sur cet écran d'accueil :



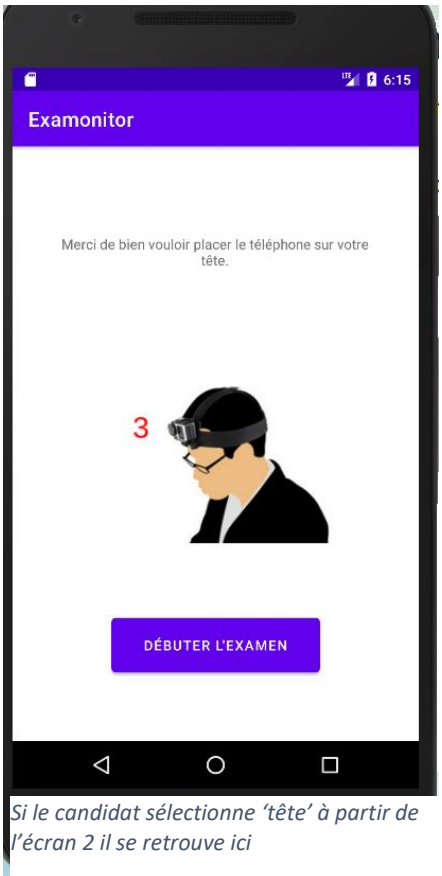
Ecran 1 d'accueil de l'application mobile

Le candidat procède en cliquant sur le bouton 'commencer'. Il se retrouve ensuite sur la 2ème activity

Ecran 2 informatif

Ensuite, le candidat choisi si ce téléphone sera placé sur son bras ou sur sa tête.

Si le candidat sélectionne 'bras' à partir de l'écran 2 il se retrouve ici



Si le candidat sélectionne 'tête' à partir de l'écran 2 il se retrouve ici

1) La détection des objets par le téléphone placé sur la tête

Lorsque le candidat débute l'examen à partir de l'écran du téléphone de la tête, il est envoyé sur l'application IP Webcam qui envoie le flux de la vidéo à une adresse IP précise. Cette adresse IP est récupérée par le programme Python qui effectue la détection des objets. Le programme a la capacité de détecter des objets tel qu'un téléphone, ce qui est considéré comme une tentative de triche. Parallèlement, un message est envoyé au programme Python (qui est sur Flask) grâce à une socket : ainsi le programme peut déterminer que le téléphone de la tête est bien connecté.

2) La détection des mouvements par le téléphone placé sur le bras

Lorsque le candidat débute l'examen à partir du téléphone du bras, il se retrouve sur cet écran :

Ecran de l'examen en cours (bras)

Ici des données de l'accéléromètre sont envoyées au programme Python grâce à des sockets. Nous avons une socket JAVA qui communique avec une socket Python. Le SensorManager d'Android nous permet d'accéder à de multiples données liées aux capteurs présent sur le téléphone. On utilise donc les données de l'accéléromètre. Nous avons donc l'accélération lié aux 3 axes X,Y et Z en m/s. La méthode JAVA 'onSensorChanged' est appelée à chaque changement d'accélération lié à un des 3 axes.

Notre stratégie était donc de regrouper les données en groupe de 10. Il est à noter que le capteur de l'accéléromètre est très sensible, la méthode onSensorChanged est donc appelée de manière très fréquente. Ainsi, en quelques secondes, on récupère 10 données.

Par exemple :

X	Y	Z	
1.38833	3.8373	4.9834	---- donnée 1
5.38833	7.3873	4.4844	--- donnée 2
5.38833	12.38373	8.9844	--- donnée 3

…

Ainsi, quand on récupère ses 10 données, on les envoie au programme Python. Sur le programme Python, on génère un dataset fictif de 1000 données composées de 4 variables (X, Y, Z et CHEAT). Afin de créer le modèle, on a ajouté la variable target CHEAT qui prends comme valeur 0 = PAS DE TRICHE et 1 = TRICHE. Voici la fonction Python qui nous permet de générer le dataset :

```python
def generer_dataset_bras():
    X = np.random.uniform(0.1,15,1000)
    Y = np.random.uniform(0.1,15,1000)
    Z = np.random.uniform(0.0,15,1000)
    df = pd.DataFrame({'X':X,'Y':Y,'Z':Z})

    #suppositions de triche, si il y a une acceleration supérieure a 3.5 m/s le long de x et y
    # ou si la somme de x+y+z dépasse 25
    df['CHEAT'] = np.where(   ((df['X']> 3.5) & (df['Y']> 3.5)) | ( (df['X']+df['Y']+df['Z']) > 30)  ,1,0)
    return df
```

```python
#dfg= data frame généré
dfg = generer_dataset_bras()
```

Voici un exemple d'un dataset généré aléatoirement :

```
Voici le dataset généré:

              X          Y          Z  CHEAT
0     12.168825   5.385987   0.472285      1
1      1.493077   6.199089   3.017228      0
2     10.067114   2.621523   5.283800      0
3     13.476836   0.635458  12.659752      0
4      7.828701  14.272813   3.832331      1
..          ...        ...        ...    ...
995   13.447676  14.883966  14.730554      1
996   13.058951   3.445016  14.205105      1
997    1.298202  14.680396  11.778710      0
998    6.811284   9.660931   3.612869      1
999   10.846836  13.218751   4.207956      1

[1000 rows x 4 columns]
Sur le dataframe généré, on a 618/1000 cas de triche et 382/1000 de cas de non triche.
```

On utilise ensuite la bibliothèque Sklearn de Python qui nous permet de faire du Machine Learning. On utilise l'algorithme de classification K Neighbors classifier qui nous permet d'implémenter la classification des k plus proches voisins. Ici, on va utiliser k = 3. Ainsi si notre donnée à 2 plus proches voisins qui sont classifié comme CHEAT=1 alors elle sera classifiée comme CHEAT.

On utilise ensuite une fonction qui va permettre de faire la classification de nos 10 données. Ainsi, on obtient un array composé de 10 éléments (des 0 et des 1) par exemple : [0 0 1 1 1 1 0 0 1 1] → ici nous avons 6 données classifiées comme triche, ainsi le niveau de triche affiché sur l'application web sera de 60%. Ce pourcentage fluctue en temps réel en fonction du mouvement du bras.

Le résultat de la classification est indiqué dans une variable 'fraud_arms' qui va être récupérée par une fonction AJAX : c'est cela qui va nous permettre d'afficher en temps réel le taux de triche en fonction du mouvement du bras. Voici la fonction Python qui permet de faire la détection en temps réel :

```python
def detect_cheat_arm(data):
    global fraud_arms
    if((data.partition('\n')[0])=='Bras'):

        df = data.split("\n",1)[1]
        df = (df.split(" "))

        ndf = np.empty(0)
        for el in df:
            el = el.split("\n")
            for e in el:
                ndf = np.append(ndf,float(e))
        ndf = np.reshape(ndf,(10,3))
        print(ndf)
        knc = classification_triche(dfg)

        knc.fit(X_train, y_train.values.ravel())

        y_pred = knc.predict(ndf)
        print(y_pred)
        nb =  np.count_nonzero(y_pred == 1)
        nb=nb*10
        snb = str(nb)
        file = open('fraud/log.txt', 'a+')
        now = datetime.now()
        dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
        file.write(f"Arm activity detected at  {dt_string} with {snb}% cheating level. \n")
        file.close()
        fraud_arms = "Arm activity cheating level :" + str(nb) + '%'

        return ndf
```

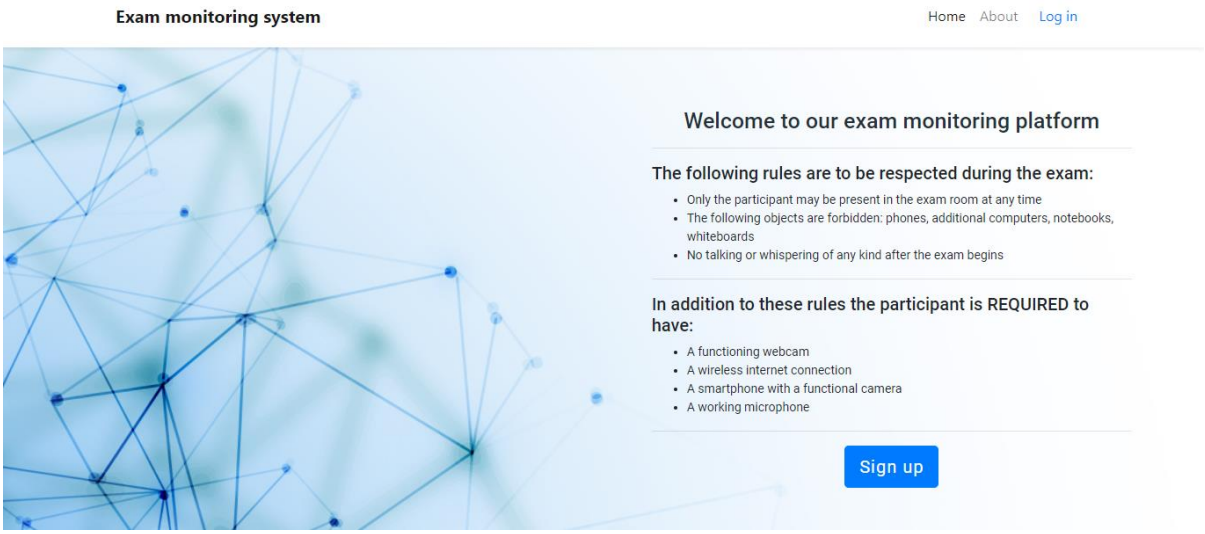
A chaque fois que les 10 données sont envoyées, la fonction detect_cheat_arms est exécutée. On vérifie ensuite que les données reçues (une string) par la socket ont bien été envoyées par le téléphone du bras. On prépare ensuite les données avant de les classifier.

B) L'application web

Pour le site web, nous avons choisi de faire une application web Flask. Ceci nous a permis d'utiliser Python afin de récupérer les données et de faire le traitement. On voulait créer une version améliorée du site evalbox, un site de surveillance d'examen déjà connu. La différence entre evalbox et notre solution est l'absence d'un blocage du navigateur et du chronomètre. On a décidé de se focaliser principalement sur la détection automatique de triche.

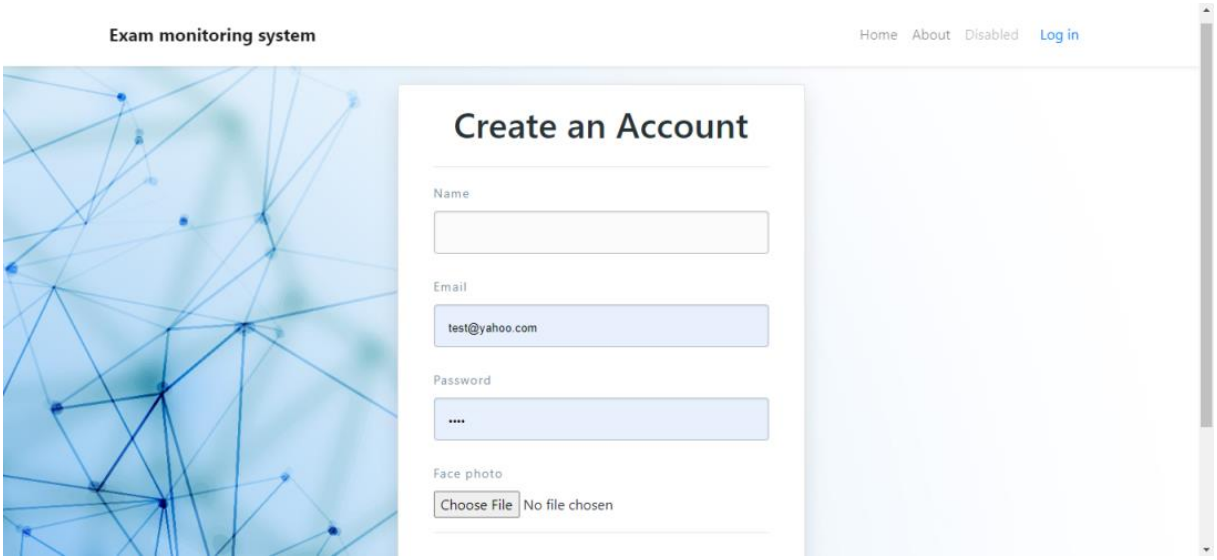
Page d’acceuil :

Lorsque l’utilisateur se rends sur le site, il tombe sur une page qui lui donne toutes les informations concernant son examen, d’une liste des objets/mouvements interdits et il a l’option de s’inscrire ou se connecter.



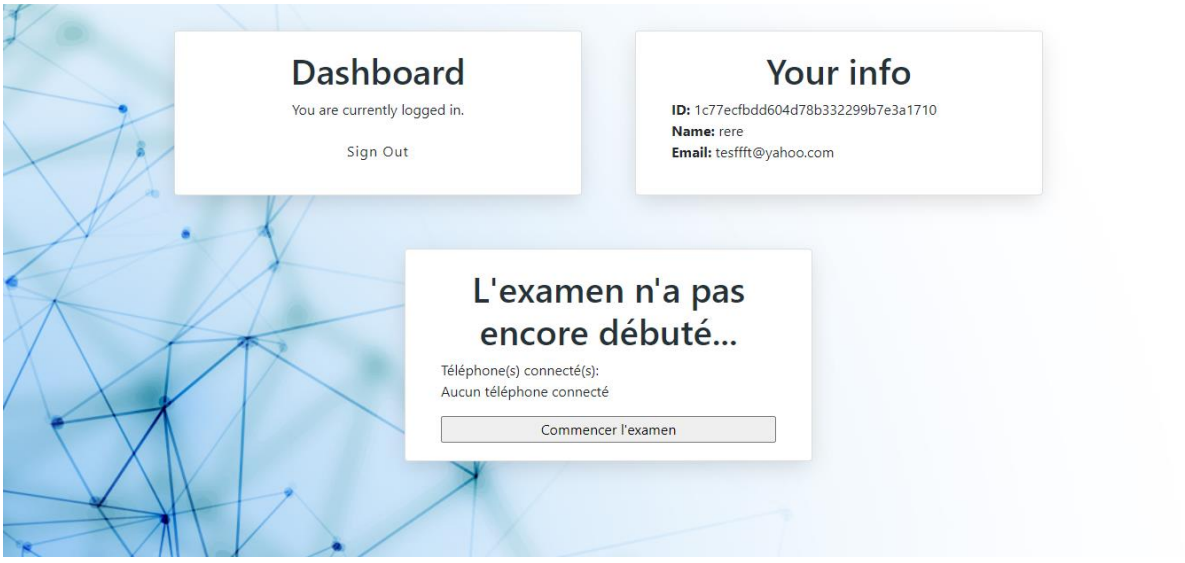
Inscription :

La page d’inscription permet au candidat de créer un compte afin de passer son examen. On vérifie qu’il entre le bon format pour l’adresse e-mail, ensuite on utilise un module Python qui permet d’encoder le mot de passe afin d’avoir une bonne sécurité. Si une adresse e-mail X est déjà enregistrée dans notre système, un autre utilisateur ne pourra pas s’inscrire avec cette même adresse X. Finalement, l’utilisateur doit ajouter une photo de lui, qui permettra de faire la reconnaissance faciale. On enregistre ensuite les informations dans notre base de donnée MongoDB.



La page de connexion est simple, on vérifie que les données entrées sont bien enregistrées dans notre système, et on vérifie si le mot de passe correspond. Une autre fonctionnalité ajoutée est le fait que l'utilisateur ne peut pas accéder à la page d'examen s'il n'est pas connecté. Ceci a été possible en utilisant Sessions. Si le candidat essaye d'aller sur la page d'examen sans être connecté, alors il sera directement dirigé vers la page d'acceuil et on lui demandera de créer un compte.

<u>Dashboard</u>



Le dashboard est la page ou on prépare tout avant de commencer l'examen. Dès que cette page se charge, le serveur prépare l'image que l'utilisateur a ajouté lors de l'inscription pour la reconnaissance faciale (encodage, BGR → RGB).

Un autre point important est que l'utilisateur ne peut pas commencer l'examen si il n'a pas connecté ses 2 téléphones à l'application que nous avons développée. Après avoir connecté ses 2 téléphones, il peut ensuite débuter l'examen.

Nous allons maintenant détailler comment les 3 autres types détections de triche ont été implémentées (la détection par les mouvements du bras a été détaillée dans la partie application mobile).

Pour rappel, il y a 4 types de détection de triche :

- Reconnaissance faciale
- Détection d'objets
- Reconnaissance vocale
- Capteur du bras (déjà évoquée)

<u>Reconnaissance faciale :</u>

Cette partie était difficile à implémenter dans notre application web. Il y avait de nombreux tutoriels Python pour implémenter la reconnaissance faciale sur Python. Cependant, le plus difficile était de faire la reconnaissance sur Flask (sur le site web) et de s'assurer que cela fonctionne correctement. On a décidé d'utiliser la libraire Python nommée FaceRecognition : https://github.com/ageitgey/face_recognition

qui est la librairie la plus performante (en utilisant peu d'images) avec un taux de réussite de 99.7%.

Au début, nous avons suivi un tutoriel sur PySource qui a utilisé FaceRecognition et OpenCV pour identifier rapidement une personne avec son visage : https://github.com/ageitgey/face_recognition .

Pour implémenter la reconnaissance sur Flask, nous avons combiné le tutoriel précèdent avec ce tutoriel 'How to implement functionalities inside Flask' :

https://github.com/krishnaik06/Flask-Web-Framework/tree/main/Tutorial%208

Pour permettre que l'encodage et la détection s'exécute au bon moment, nous avons utilisé Python executor threads. Ainsi, nous n'avons pas de temps d'attente en chargeant la page.

Détection d'objets :

Pour la détection d'objets on a utilisé un tutoriel par NeuralNine. On avait initialement planifié de faire notre propre modèle en utilisant des centaines d'images trouvées dans la bibliothèque de reconnaissance de Google. On a utilisé Darknet et YOLO V5 (you only look once) afin d'entraîner un modèle sur le cloud. Cependant, après avoir testé la performance, on a conclu que c'était trop lent. On a alors été menés à choisir le COCO dataset, qui a considérablement amélioré la performance.

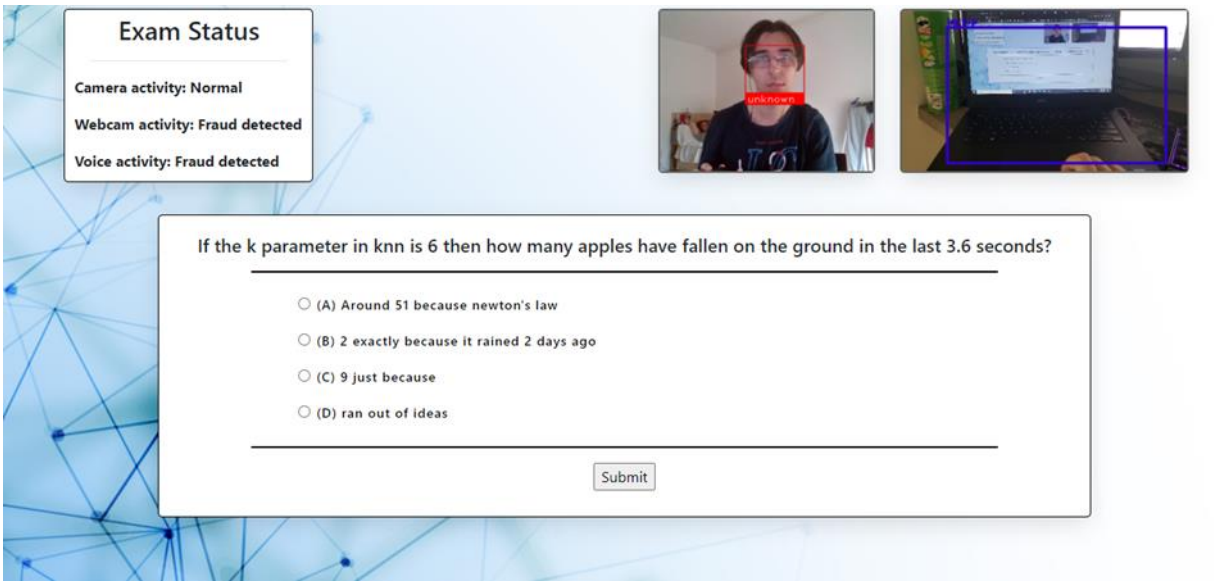
Reconnaissance vocale :

Le but de cette fonctionnalité était de détecter si des personnes parlaient pendant l'examen. On a implémenté cela en utilisant deux modules Python :

Speech_recognition et PyAudio. Ainsi, notre application accède au microphone de l'ordinateur et on utilise le détecteur de Google afin de vérifier s'il y a des personnes qui parlent.

La page d'examen :

Afin de voir un exemple de déroulement d'examen, regardez la vidéo Youtube de notre projet : https://www.youtube.com/watch?v=ZQbcUqdNra4



Sur la page d'examen, on a 2 vidéos qui s'affichent : une pour la reconnaissance faciale et une pour les détections d'objets. C'est aussi la page ou se déclenchent les détections des mouvements et de la parole. On a aussi un formulaire où l'examen doit répondre à une question.

C) <u>Affichage des informations de triche en temps réel et enregistrement des détections dans un fichier log</u>

Afin d'afficher les informations en temps réel sans recharger la page, nous utilisons des requêtes AJAX. Le principe est simple, on fait une requête à une fonction Python qui est sur Flask, ainsi on récupère la réponse et on l'affiche sur le endroits dédiés (en haut à gauche de la page d'examen). Les requêtes sont faites chaque seconde. Voici le code Javascript :

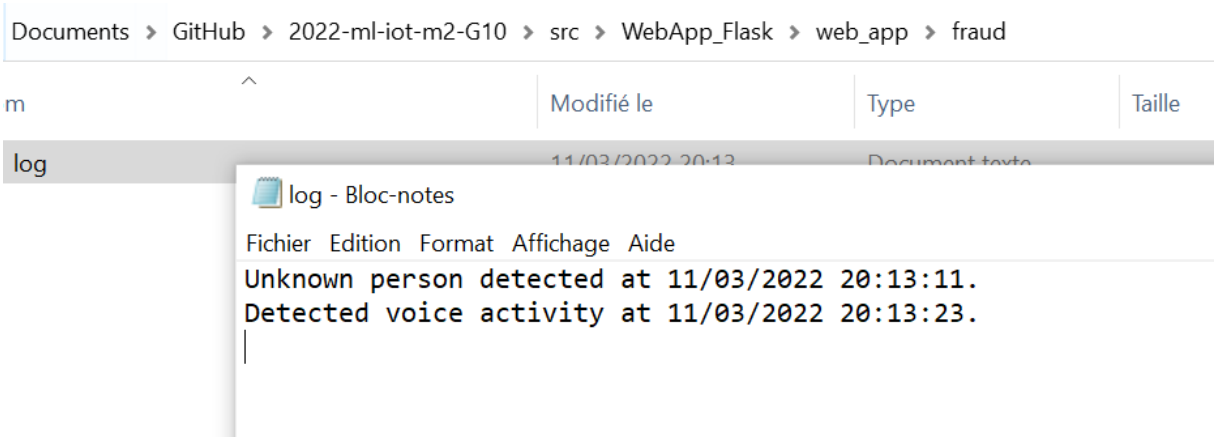
```javascript
function detect_fraud_cam_web() {
    $.ajax({
        url: '/fraud_cam_web',
        success: function(data) {
            console.log(data);
            $("#fraud_cam_web_detected").text(data)
        },complete:function(){
            setTimeout(detect_fraud_cam_web, 1000);
        }
    });
}
function detect_fraud_cam_phone() {
    $.ajax({
        url: '/fraud_cam_phone',
        success: function(data) {
            console.log(data);
            $("#fraud_cam_phone_detected").text(data)
        },complete:function(){
            setTimeout(detect_fraud_cam_phone, 1000);
        }
    });
}
```

Ici, on a un exemple de 2 fonctions javascript (sur le fichier exam.html) qui concernent la détection avec la caméra du téléphone et la caméra de l'ordinateur. La requête est faite à des fonctions sur le code sur Flask, (app.py) :

```python
@app.route('/fraud_cam_web')
def fraud_cam_web():
    if(fraud_cam_web==True):
        return jsonify("Webcam activity: Fraud detected")
    else:
        return jsonify("Webcam activity: Normal")

@app.route('/fraud_cam_phone')
def fraud_cam_phone():
    if(fraud_cam_phone==True):
        return jsonify("Camera activity: Fraud detected")
    else:
        return jsonify("Camera activity: Normal")
```

C’est ce processus qui nous a permis d’afficher les informations en temps réel. Etant donné que nous avons la capacité de savoir à quel moment une triche a été détectée, on peut l’enregistrer dans un fichier log qui est dans le dossier ‘fraud’ (qui est dans WebAppFlask/web_app). A la fin de l’examen on pourrait avoir par exemple :



Ceci a été fait grâce à ce code :

```
if 'unknown' in face_names:
    file = open('fraud/log.txt', 'a+')
    now = datetime.now()
    dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
    file.write(f"Unknown person detected at {dt_string}. \n")
    file.close()
    break
```

C’est le même code pour le reste des détections.

IV- Liste des outils et technologies utilisées :

Voici la liste détaillée :

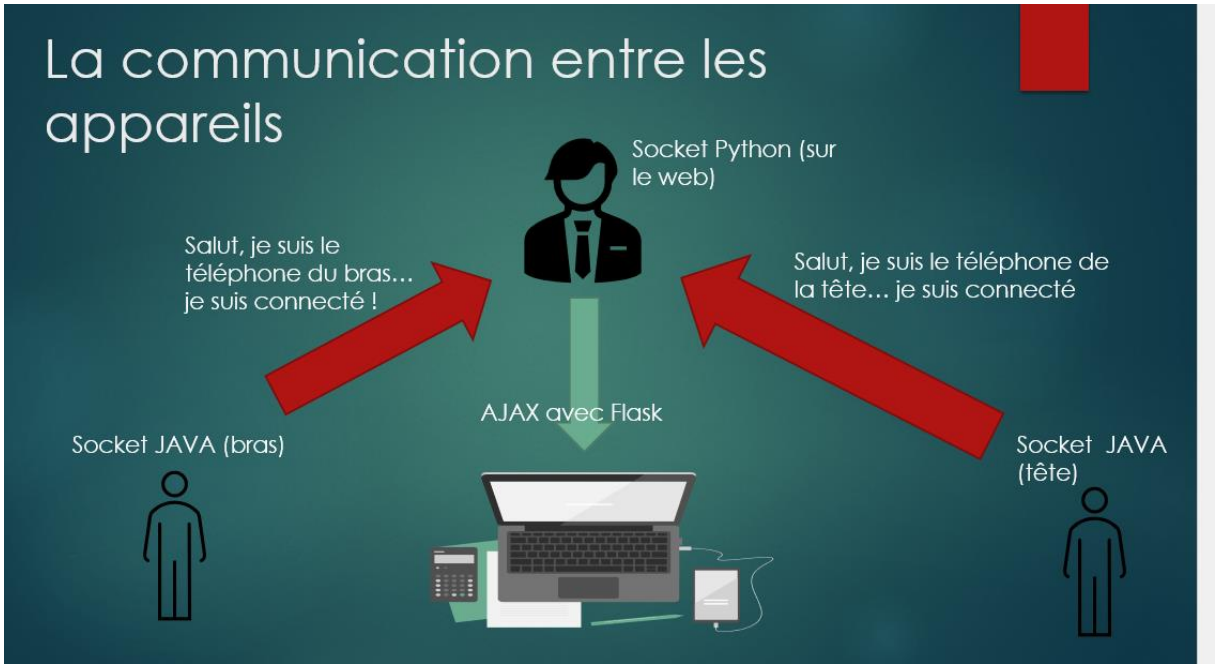
Web : Flask, Python, Javascript (Ajax, Jquery), HTML, CSS, Bootstrap

Base de données : MongoDB

Mobile : Java, XML

Reconnaissance et détection : Python (speech_recognition, face_recognition, sci-kit learn)

Connexions entre l'ordinateur et les téléphones : sockets Python et sockets JAVA. Un fonctionnement simple :



V- Conclusion :

Notre projet de surveillance d'examen à distance a été réalisé sur une application web et une application mobile. Le candidat peut s'inscrire et se connecter. Après sa connexion, il connecte les 2 téléphones et peut commencer l'examen. On utilise des sockets pour permettre les connexions entre les appareils. On affiche ensuite en temps réel sur le navigateur (grâce aux requêtes AJAX → Python) les informations de détection. A la fin, nous avons aussi un fichier log qui contient toutes les informations de triche.

Lien de la vidéo du projet :
https://www.youtube.com/watch?v=ZQbcUqdNra4