BuskoGuard

Système de gestion de contrôle de présence par QR Code et NFC

Projet d'intégration des connaissances en mathématique et informatique

INF5000

Présenté par :
Joseph Samuel Jonathan
Étudiant en 4ème année de génie informatique

Dépôt du projet: https://github.com/josephsamijona/busko_guard_v2_mvp.git

Plan du projet BuskoGuard

1. Présentation du projet

A. **Qu'est-ce que BuskoGuard?**
B. **À quoi sert ce système de gestion de contrôle de présence?**
C. **Quel problème ce projet vient-il résoudre?**

2. Guide de démarrage

A. **Comment lancer le système?**
B. **Quelles sont les informations essentielles à connaître?**
C. **Quelles sont les prérequis techniques?**

3. Architecture et fonctionnement

4. Maintenance et support

5. Perspectives d'évolution

A. **Quelles sont les améliorations futures possibles?**
B. **Comment le système pourrait-il être étendu?**
C. **Quelles autres technologies pourraient être intégrées?**

6. Conclusion

A. **Quels sont les principaux apports de ce projet?**
B. **Comment ce projet a-t-il permis d'intégrer les connaissances en mathématique et informatique?**

1. PRESENTATION DU PROJET

Qu'est-ce que BuskoGuard?

BuskoGuard est un système de contrôle de présence et de gestion d'employés qui utilise les technologies QR Code et NFC. Il s'agit d'une solution simple, non sophistiquée, conçue dans l'optique d'être facilement déployable et portable.

Le projet est né d'une observation personnelle dès mon arrivée à l'ISTEAH, où j'ai constaté que l'administration utilisait un système d'appointage manuel pour gérer la présence. Cette méthode traditionnelle, bien que fonctionnelle, présentait plusieurs limitations et opportunités d'amélioration grâce aux technologies modernes.

À quoi sert ce système de gestion de contrôle de présence?

BuskoGuard sert à:

- Automatiser le processus de pointage des présences
- Réduire les erreurs liées à la saisie manuelle
- Générer des rapports de présence précis et en temps réel
- Faciliter la gestion administrative du personnel
- Offrir une interface simple pour les utilisateurs et les administrateurs

Le système exploite deux technologies complémentaires:

- Les QRCodes, qui peuvent être scannés via un smartphone ou un lecteur dédié
- La technologie NFC (Near Field Communication), qui permet une validation de présence par simple proximité d'une carte ou d'un badge

Quel problème ce projet vient-il résoudre?

BuskoGuard répond à plusieurs problématiques concrètes:

1. **Inefficacité du système manuel**: Les systèmes d'appointage manuels sont chronophages, sujets aux erreurs humaines et difficiles à analyser a posteriori.
2. **Besoin de solutions adaptées au contexte local**: En créant BuskoGuard, j'ai voulu proposer une alternative qui soit:
    - Adaptée aux ressources disponibles
    - Facile à déployer sans infrastructure complexe
    - Accessible même avec des contraintes techniques ou budgétaires
3. **Complexité excessive des solutions existantes**: De nombreux systèmes de pointage commerciaux sont:
    - Trop coûteux
    - Excessivement complexes à configurer et maintenir

- Surdimensionnés par rapport aux besoins réels

BuskoGuard représente donc une innovation simple mais efficace, pensée pour apporter une solution directement applicable à un problème quotidien observé dans un contexte académique ou professionnel.

2. GUIDE DE DEMARRAGE

Dans cette section, je vais vous montrer comment faire pour démarrer le projet BuskoGuard. Je sais bien qu'il est souvent difficile de suivre ce type de guide car on rencontre fréquemment des problèmes de dépendances liés à des librairies. Je ferai de mon possible pour résumer les points importants et simplifier le processus. Le but est que même une personne non initiée ou peu à l'aise avec la technologie puisse lancer ce système par elle-même.

Structure générale du système

Avant de commencer, il est important de comprendre que BuskoGuard est composé de deux applications distinctes:

1. **L'application principale Django** (core et config) - Il s'agit du backend contenant:
   - La logique du code de l'API
   - Les templates des interfaces administrateur pour la gestion générale
   - L'interface kiosk qui est le système de pointage
   - L'interface dashboard destinée aux employés

   Tous ces éléments se trouvent dans le backend Django.

2. **L'application mobile** - Située dans le dossier `employee_mobile_app`, il s'agit du client mobile développé avec Flutter qui sert de carte ID digitale pour l'employé lors du pointage.

Méthodes de déploiement

Il existe deux cas de figure pour déployer BuskoGuard:

1. **Déploiement en local** - Cette méthode est recommandée lorsque l'application se trouve sur un serveur ou une machine qui n'est pas constamment connectée à internet. C'est le déploiement qui sera le plus simple car la gestion des médias, des fichiers statiques et de la base de données est faite de manière locale (SQLite).

2. **Conteneurisation Docker** - Cette seconde méthode offre plus de possibilités car elle permet de mettre le projet dans un conteneur Docker qui pourra être lancé sur un serveur cloud, depuis GitHub, ou sur un système embarqué comme un Raspberry Pi.

Peu importe votre choix, nous verrons les deux cas de figure dans les sections suivantes.

Comment lancer le système?

Méthode de déploiement en local

Pour déployer BuskoGuard en local, suivez attentivement les étapes ci-dessous. Ce guide suppose que vous utilisez Windows.

Prérequis

1. **Installation de Python**

   Assurez-vous d'avoir installé Python et ajouté Python au PATH système. Sans cette configuration, vous ne pourrez pas lancer les commandes Python depuis n'importe quel répertoire.

   Pour vérifier si Python est correctement installé, ouvrez un terminal et tapez:

   ```
   python --version
   ```

2. **Installation de Git**

   Téléchargez et installez la dernière version de Git depuis git-scm.com.

3. **Configuration de PowerShell (si nécessaire)**

   Si vous utilisez PowerShell, vous devrez peut-être configurer la politique d'exécution pour permettre la création d'environnements virtuels. Ouvrez PowerShell en tant qu'administrateur et exécutez:

   ```
   Set-ExecutionPolicy RemoteSigned
   ```

   Confirmez le changement lorsque vous y êtes invité.

Étapes d'installation

1. **Cloner le dépôt**

Ouvrez un terminal (PowerShell ou le nouveau Terminal Windows) et exécutez:

```
git clone https://github.com/josephsamijona/busko_guard_v2_mvp.git
cd busko_guard_v2_mvp
```

Pour vous assurer que vous êtes dans le bon répertoire, exécutez `ls` (ou `dir` dans cmd). Si vous voyez le fichier `manage.py`, vous êtes au bon endroit. (Si vous le voyez, félicitations, vous avez franchi la première étape sans embûche!)

2. **Création d'un environnement virtuel**

   Cette étape est cruciale car elle sépare votre projet de l'environnement Python global, évitant ainsi les conflits potentiels de dépendances:

   ```
   python -m venv venv
   ```

   Pour activer l'environnement virtuel:

   ```
   # Sur Windows avec PowerShell ou Terminal
   .\venv\Scripts\activate
   ```

   Vous devriez voir le nom de l'environnement `(venv)` en vert au début de votre ligne de commande, indiquant que l'environnement virtuel est actif.

3. **Installation des dépendances**

   Dans le répertoire principal du projet, avec l'environnement virtuel activé, exécutez:

   ```
   pip install -r requirements.txt
   ```

4. **Configuration du fichier .env**

   Puisque vous avez téléchargé le projet depuis GitHub, certains fichiers sensibles ne sont pas inclus, notamment le fichier `.env` qui contient toutes les informations d'identification du projet.

   Créez un fichier nommé `.env` (attention: avec le point devant, sans extension) à la racine du projet et copiez-y le contenu suivant:

```
 # Base Django Settings
DEBUG=True
SECRET_KEY="niojokoojojojojji990009uhnnmiioojy167ikw)vs85$idff_%^6o&mg5
8+*yd3ldb&4v6fpssat&7="
ALLOWED_HOSTS=localhost,127.0.0.1

# JWT Settings
JWT_SECRET_KEY=L9kNv2$Pm5@Rw8^biijion1*Fy6+Tb4_Cn9!Zd7%Ga2$
JWT_ACCESS_TOKEN_LIFETIME=3600
JWT_REFRESH_TOKEN_LIFETIME=86400

# Database (MySQL)
MYSQL_URL=mysql://root:zpdAFkCBZPzBOmmdZJsPINkhTsggcuCg@gondola.proxy.r
lwy.net:31884/railway

# CORS & CSRF Settings
CORS_ALLOWED_ORIGINS=https://*.railway.app,https://*.vercel.app,https:/
/*.herokuapp.com,https://*.onrender.com,http://localhost:3000,http://12
7.0.0.1:3000,https://appjhbridge.up.railway.app
CORS_ALLOW_CREDENTIALS=True
CSRF_TRUSTED_ORIGINS=https://*.railway.app,https://*.vercel.app,https:/
/*.herokuapp.com,https://*.onrender.com,http://localhost:8000,http://12
7.0.0.1:8000,https://appjhbridge.up.railway.app
CORS_ALLOW_HEADERS=accept,accept-encoding,accept-
language,authorization,content-type,dnt,origin,user-agent,x-
csrftoken,x-requested-with,access-control-allow-origin,access-control-
allow-headers,access-control-allow-methods,cache-control,pragma,sec-
fetch-dest,sec-fetch-mode,sec-fetch-site

# Cache Settings
CACHE_TTL=300
POSITION_CACHE_TTL=60
SCHEDULE_CACHE_TTL=3600

# Redis & Celery
REDIS_URL=redis://default:pZDsUIIsUwUSWsNvlrfgeHezxUxwPTxD@shinkansen.p
roxy.rlwy.net:10800
CELERY_BROKER_URL=redis://default:pZDsUIIsUwUSWsNvlrfgeHezxUxwPTxD@shin
kansen.proxy.rlwy.net:10800
CELERY_RESULT_BACKEND=redis://default:pZDsUIIsUwUSWsNvlrfgeHezxUxwPTxD@
shinkansen.proxy.rlwy.net:10800

# Email Configuration
EMAIL_HOST=smtp.gmail.com
EMAIL_PORT=587
EMAIL_HOST_USER=jhbridgetranslation@gmail.com
EMAIL_HOST_PASSWORD=vmwa dbks phrc wfnk
EMAIL_USE_TLS=True
EMAIL_USE_SSL=False
DEFAULT_FROM_EMAIL=jhbridgetranslation@gmail.com

B2_ACCESS_KEY_ID=0057c06ea39a0970000000001
B2_SECRET_ACCESS_KEY=K005IwR8fXYEw3qng/kkQLhrAESDr/0
```

```
B2_BUCKET_NAME=jhbridgestockagesystem
B2_ENDPOINT_URL=https://s3.us-east-005.backblazeb2.com
B2_REGION_NAME=us-east-005

# Paramètres S3
B2_DEFAULT_ACL=public-read
B2_QUERYSTRING_AUTH=False
B2_FILE_OVERWRITE=False
B2_LOCATION=media
DEFAULT_FILE_STORAGE=custom_storages.MediaStorage

IS_DEVELOPMENT=True
```

Explication des variables d'environnement:

- **DEBUG**: En mode développement, mettez `True` pour obtenir des messages d'erreur détaillés. **ATTENTION**: Ne jamais laisser `DEBUG=True` en production!
- **SECRET_KEY**: Clé secrète utilisée par Django pour la sécurité (signatures, tokens, etc.)
- **ALLOWED_HOSTS**: Liste des hôtes autorisés à servir l'application
- **JWT_***: Paramètres pour les JSON Web Tokens utilisés dans l'authentification
- **MYSQL_URL**: URL de connexion à la base de données MySQL (non utilisée en mode local avec SQLite)
- *CORS_ et CSRF_***: Paramètres de sécurité pour les requêtes cross-origin
- **CACHE_***: Paramètres de cache pour optimiser les performances
- *REDIS_ et CELERY_***: Configuration pour les tâches asynchrones (non utilisées en développement local)
- **EMAIL_***: Configuration pour l'envoi d'emails
- **B2_***: Paramètres pour le stockage des fichiers sur Backblaze B2 (stockage de type S3)
- **IS_DEVELOPMENT**: Indique si l'application est en mode développement

5. **Configuration de la base de données**

   Comme vous êtes en mode local, je recommande d'utiliser SQLite comme base de données. C'est déjà intégré au code et ne nécessite aucune configuration supplémentaire.

   Si vous souhaitez utiliser d'autres services (base de données externe, Redis, etc.), mettez `IS_DEVELOPMENT=False` dans le fichier `.env`. Notez que cela nécessitera des configurations supplémentaires.

   Attention: Gardez `DEBUG=True` uniquement pour les tests locaux. Pour un serveur de production, il faut absolument mettre `DEBUG=False` pour des raisons de sécurité.

6. **Appliquer les migrations**

   Exécutez les commandes suivantes pour préparer et appliquer les migrations de la base de données:

   ```
   python manage.py makemigrations
   ```

```
python manage.py migrate
```

Si vous rencontrez des erreurs, vérifiez le mode de fonctionnement dans le fichier `.env`. Il est possible que le système attende des informations d'identification supplémentaires.

7. **Création d'un compte superutilisateur**

   Pour accéder à l'interface d'administration avec tous les privilèges:

   ```
   python manage.py createsuperuser
   ```

   Suivez les instructions pour créer vos identifiants d'administrateur.

8. **Lancement du serveur**

   Si vous avez passé toutes ces étapes sans encombre, bravo! (Vous méritez une petite pause café ☕)

   Lancez le serveur avec:

   ```
   python manage.py runserver
   ```

Navigation dans le système

- **Interface Kiosk (par défaut)**: L'URL par défaut (`http://127.0.0.1:8000/`) vous amène à l'interface kiosk qui permet de réaliser les scans.
- **Interface Admin**: Pour gérer le système, effectuer des opérations CRUD, gérer le contrôle de présence et les employés, allez à `http://127.0.0.1:8000/admin/` et connectez-vous avec vos identifiants superutilisateur.
- **Interface Employé**: Accessible via `http://127.0.0.1:8000/login/`. Les employés y ont accès à leur carte ID numérique, leur historique de présence, et peuvent demander et suivre leurs congés.

Configuration de l'application mobile

Pour connecter l'application mobile Flutter à votre instance locale Django:

1. **Exposer votre localhost à Internet**

   Vous pouvez utiliser ngrok pour créer un tunnel vers votre serveur local:

   - Téléchargez et installez ngrok
   - Exécutez: `ngrok http 8000`
   - Notez l'URL fournie (ex: `https://abc123def456.ngrok.io`)

2. **Mettre à jour le fichier .env**

Ajoutez l'URL ngrok à votre liste `ALLOWED_HOSTS` dans le fichier `.env`:

```
ALLOWED_HOSTS=localhost,127.0.0.1,abc123def456.ngrok.io
```

3. **Configurer l'application Flutter**
   - Naviguez vers le dossier de l'application mobile: `cd employee_mobile_app`
   - Installez les packages: `flutter pub get`
   - Ouvrez `lib/constants.dart` et remplacez les URLs par l'URL ngrok obtenue
   - Lancez l'application en mode debug: `flutter run`

   Ou créez un APK:

   ```
   flutter build apk --release
   ```

   L'APK sera généré dans `build/app/outputs/flutter-apk/app-release.apk`

Création d'utilisateurs et d'employés

Pour configurer rapidement le système:

1. **Création manuelle**:
   - Accédez à l'interface d'administration
   - Créez d'abord un utilisateur
   - Créez ensuite un employé associé à cet utilisateur
   - Attribuez-lui un rôle, un département et un horaire
   - Le système générera automatiquement son QR code
2. **Utilisation des données de test**:

   Pour un démarrage rapide, le projet inclut un script qui remplit la base de données avec des informations fictives:

   ```
   python populate_db.py
   ```

   Les identifiants de connexion seront stockés dans le fichier `user_credentials.json`.

Félicitations! Vous avez maintenant une instance locale fonctionnelle de BuskoGuard prête à être utilisée.

Méthode de déploiement avec Docker

Docker permet de conteneuriser l'application BuskoGuard, la rendant portable et facilement déployable sur différentes plateformes. Le projet dispose déjà des fichiers nécessaires pour la containerisation (Dockerfile, runtime.txt et Procfile), simplifiant grandement le processus.

Prérequis généraux

- Docker et Docker Compose installés
- Un compte GitHub pour forker le projet
- Connaissances de base en Docker et en ligne de commande

Fichiers de configuration existants

Le projet BuskoGuard contient déjà les fichiers nécessaires à la containerisation:

1. **Dockerfile**: Définit comment l'image Docker doit être construite
2. **runtime.txt**: Spécifie la version de Python utilisée
3. **Procfile**: Définit les commandes à exécuter au démarrage (utilisé principalement dans les environnements cloud)

Déploiement sur trois environnements différents

1. Sur un VPS ou serveur privé

Un VPS (Virtual Private Server) ou un serveur privé offre un contrôle total sur l'environnement d'exécution.

Étapes:

1. **Préparation du serveur**

```
# Installer Docker et Docker Compose
sudo apt update
sudo apt install docker.io docker-compose
sudo systemctl enable docker
sudo systemctl start docker
```

7. **Forker et cloner le projet**

```
git clone https://github.com/VOTRE-USERNAME/busko_guard_v2_mvp.git
cd busko_guard_v2_mvp
```

10. **Créer un fichier docker-compose.yml**

```
version: '3'

services:
  web:
    build: .
    restart: always
    ports:
      - "8000:8000"
    environment:
      - DEBUG=False
      - SECRET_KEY=votre_cle_secrete_tres_longue
      - ALLOWED_HOSTS=votre-domaine.com,www.votre-domaine.com
      - IS_DEVELOPMENT=False
      # Autres variables d'environnement...
    depends_on:
      - db
```

```
      - redis

  db:
    image: mysql:8.0
    restart: always
    volumes:
      - mysql_data:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=votre_mot_de_passe_mysql
      - MYSQL_DATABASE=buskoguard

  redis:
    image: redis:alpine
    restart: always
    volumes:
      - redis_data:/data

volumes:
  mysql_data:
  redis_data:
```

Lancer les conteneurs

```
sudo docker-compose up -d
```

Appliquer les migrations et créer un superutilisateur

```
sudo docker-compose exec web python manage.py migrate
sudo docker-compose exec web python manage.py createsuperuser
```

Configurer un proxy inverse (Nginx) pour HTTPS

```
sudo apt install nginx certbot python3-certbot-nginx
```

Créer un fichier de configuration Nginx:

```
# /etc/nginx/sites-available/buskoguard
server {
    server_name votre-domaine.com www.votre-domaine.com;

    location / {
        proxy_pass http://localhost:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

Activer le site et configurer HTTPS:

```
sudo ln -s /etc/nginx/sites-available/buskoguard /etc/nginx/sites-
enabled/
sudo nginx -t
sudo systemctl reload nginx
sudo certbot --nginx -d votre-domaine.com -d www.votre-domaine.com
```

2. Sur un Raspberry Pi

Le Raspberry Pi offre une solution économique pour déployer BuskoGuard, idéale pour les petites organisations.

Étapes:

1. **Préparation du Raspberry Pi**

```
# S'assurer que le système est à jour
sudo apt update
sudo apt upgrade

# Installer Docker (script officiel)
curl -sSL https://get.docker.com | sh

# Ajouter l'utilisateur au groupe docker
 sudo usermod -aG docker pi

 # Installer Docker Compose
 sudo apt install python3-pip
 sudo pip3 install docker-compose
```

15. **Cloner le projet et préparer l'environnement**

```
 git clone https://github.com/VOTRE-USERNAME/busko_guard_v2_mvp.git
 cd busko_guard_v2_mvp
```

18. **Créer un fichier docker-compose.yml adapté aux ressources limitées du Raspberry Pi**

```
 version: '3'

 services:
   web:
     build: .
     restart: always
     ports:
       - "8000:8000"
     environment:
       - DEBUG=False
       - SECRET_KEY=votre_cle_secrete_tres_longue
       - ALLOWED_HOSTS=raspberrypi.local,192.168.1.X
       - IS_DEVELOPMENT=False
       # Autres variables...
     depends_on:
       - db
       - redis

   db:
     image: mariadb:10.5  # Plus léger que MySQL
     restart: always
     volumes:
       - mysql_data:/var/lib/mysql
     environment:
       - MYSQL_ROOT_PASSWORD=votre_mot_de_passe
       - MYSQL_DATABASE=buskoguard

   redis:
     image: redis:alpine
     restart: always
     volumes:
       - redis_data:/data

 volumes:
```

```
    mysql_data:
    redis_data:
```

5. **Lancer les conteneurs**

```
docker-compose up -d
```

7. **Configurer l'accès local**

   Pour rendre le Raspberry Pi accessible sur le réseau local:

```
sudo apt install avahi-daemon
sudo systemctl enable avahi-daemon
sudo systemctl start avahi-daemon
```

   L'application sera accessible via `http://raspberrypi.local:8000` ou `http://192.168.1.X:8000`

3. Sur un service cloud (Railway)

Railway est une plateforme cloud moderne qui simplifie considérablement le déploiement d'applications conteneurisées. Je recommande cette approche pour sa simplicité et son infrastructure évolutive.

Étapes:

1. **Forker le projet sur GitHub**

   Rendez-vous sur https://github.com/josephsamijona/busko_guard_v2_mvp et cliquez sur "Fork" pour créer votre propre copie du projet.

2. **Créer un compte Railway**

   Inscrivez-vous sur Railway et connectez votre compte GitHub.

3. **Déployer les quatre instances requises:**

   a. **Instance principale du projet Django**

   - Dans Railway, cliquez sur "New Project" > "Deploy from GitHub repo"
   - Sélectionnez votre fork de BuskoGuard
   - Railway détectera automatiquement le Dockerfile et le Procfile
   - Dans l'onglet "Variables", ajoutez toutes les variables d'environnement nécessaires:
   - `DEBUG=FalseSECRET_KEY=votre_cle_secrete_tres_longueALLOWED_HOSTS=votre-app.up.railway.appIS_DEVELOPMENT=False`

   b. **Instance MySQL**

   - Dans le même projet, cliquez sur "New Service" > "Database" > "MySQL"
   - Railway vous fournira automatiquement l'URL de connexion dans le format:

- `MYSQL_URL=mysql://root:password@container-name.railway.app:port/railway`
- Ajoutez cette URL aux variables d'environnement de votre service Django

c. **Deux instances Redis** (une pour le cache, une pour Celery)

- Ajoutez deux services Redis: "New Service" > "Database" > "Redis"
- Pour chaque instance, récupérez l'URL de connexion:
- `REDIS_URL=redis://default:password@container-name.railway.app:port`
- Configurez les variables d'environnement de votre service Django:
- `REDIS_URL=url_de_la_premiere_instanceCELERY_BROKER_URL=url_de_la_seconde_instanceCELERY_RESULT_BACKEND=url_de_la_seconde_instance`

4. **Configurer les variables d'environnement finales**

   Mettez à jour toutes les variables d'environnement dans l'onglet "Variables" de votre service Django principal, notamment:

```
CORS_ALLOWED_ORIGINS=https://votre-app.up.railway.app
CSRF_TRUSTED_ORIGINS=https://votre-app.up.railway.app
```

5. **Vérifier le déploiement**
   - Railway génère automatiquement une URL pour votre application (exemple: https://votre-app.up.railway.app)
   - Accédez à cette URL pour vérifier que l'application est bien déployée
   - Appliquez les migrations via la console Railway:
   - `python manage.py migratepython manage.py createsuperuser`

Configuration de l'application mobile Flutter

Pour que l'application mobile Flutter puisse se connecter à votre nouvelle instance déployée:

1. **Mettre à jour les constantes d'URL**
   - Naviguez vers le dossier `employee_mobile_app/lib/`
   - Ouvrez le fichier `constants.dart`
   - Remplacez les URLs existantes par l'URL de votre nouvelle instance:
   - `// Exemple avant:const String baseUrl = 'http://localhost:8000';// Exemple après:const String baseUrl = 'https://votre-app.up.railway.app';`

Créer un dépôt Git séparé pour l'application Flutter

```
2. cd employee_mobile_app
3. git init
4. git add .
5. git commit -m "Initial commit of Flutter app"
6. git remote add origin https://github.com/VOTRE-USERNAME/buskoguard_mobile_app.git
7. git push -u origin master
8. # BuskoGuard Mobile App
```

```
9.
10.  Application mobile pour le système de contrôle de présence BuskoGuard.
11.
12.  ## Installation de Flutter
13.
14.  ### 1. Télécharger Flutter SDK
15.
16.  Téléchargez la dernière version de Flutter depuis le site officiel:
17.  - Windows: https://docs.flutter.dev/get-started/install/windows
18.  - macOS: https://docs.flutter.dev/get-started/install/macos
19.  - Linux: https://docs.flutter.dev/get-started/install/linux
20.
21.  ### 2. Extraire l'archive
22.
23.  Extrayez l'archive téléchargée dans un dossier de votre choix (évitez
   les chemins avec des espaces ou des caractères spéciaux).
24.
25.  ### 3. Ajouter Flutter aux variables d'environnement
26.
27.  Ajoutez le dossier `flutter/bin` à votre PATH:
28.
29.  - **Windows**:
30.    - Ouvrez les Paramètres > Recherchez "variables d'environnement"
31.    - Modifiez la variable PATH et ajoutez le chemin complet vers
   flutter\bin
32.
33.  - **macOS/Linux**:
34.    ```bash
35.    export PATH="$PATH:`pwd`/flutter/bin"
```

Ajoutez cette ligne à votre fichier ~/.bashrc, ~/.zshrc ou équivalent

4. Vérifier l'installation

```
flutter doctor
```

Suivez les instructions pour installer les dépendances manquantes.

5. Configurer un éditeur

Installez Android Studio, VS Code ou IntelliJ avec les plugins Flutter.

Configuration du projet

1. Cloner ce dépôt:
```
2. git clone https://github.com/VOTRE-
   USERNAME/buskoguard_mobile_app.git
3. cd buskoguard_mobile_app
```
4. Installer les dépendances:
```
5. flutter pub get
```
6. Mettre à jour l'URL API dans lib/constants.dart:
```
7. const String baseUrl = 'https://votre-serveur-buskoguard.com';
```

8. Lancer l'application en mode debug:
9. `flutter run`

Génération d'APK

Pour créer un APK de release:

```
flutter build apk --release
```

L'APK sera disponible dans: `build/app/outputs/flutter-apk/app-release.apk`

Points importants à retenir

Quel que soit le mode de déploiement choisi:

1. **Sécurité**:
   - Passez `DEBUG=False` en production
   - Utilisez des mots de passe forts et uniques
   - Limitez les `ALLOWED_HOSTS` aux domaines nécessaires
   - Activez HTTPS lorsque possible
2. **Sauvegarde**:
   - Configurez des sauvegardes régulières de la base de données
   - Pour Railway, utilisez la fonctionnalité intégrée de sauvegarde
3. **Surveillance**:
   - Mettez en place un système de monitoring pour vérifier la disponibilité
   - Consultez régulièrement les journaux pour détecter d'éventuels problèmes
4. **Mise à jour**:
   - Pour mettre à jour l'application, tirez les dernières modifications et reconstruisez les conteneurs:
   - `git pulldocker-compose downdocker-compose up -d --build`
   - Sur Railway, les mises à jour se font automatiquement lorsque vous poussez sur GitHub

En suivant ces instructions, vous aurez un système BuskoGuard pleinement fonctionnel, déployé dans un environnement de production sécurisé et accessible depuis l'application mobile.

3 . ARCHITECTURE ET FONCTIONNEMENT

A. Comment fonctionne le système?

BuskoGuard fonctionne selon une architecture simple et efficace:

1. **Système monolithique Django** qui gère:
    - L'interface administrateur pour la gestion des employés, départements, et présences
    - L'interface kiosk pour effectuer les pointages
    - L'API REST pour l'application mobile
    - Le système d'authentification et d'autorisation
2. **Application mobile Flutter** qui sert de:
    - Carte d'identité numérique personnelle
    - Moyen de consulter l'historique des présences
    - Interface pour soumettre des demandes de congé
3. **Flux de fonctionnement**:
    - Le personnel administratif gère les employés via l'interface d'administration
    - Les employés utilisent l'application mobile ou se présentent au kiosk
    - Le système vérifie l'identité via QR code ou NFC
    - Les pointages sont enregistrés dans la base de données
    - Les rapports et historiques sont générés automatiquement

B. Quelles sont les structures et technologies utilisées?

Le projet repose sur une pile technologique simple mais robuste:

1. **Backend**:
    - Django (framework Python)
    - Django REST Framework pour l'API
    - SQLite en local / MySQL en production
    - Système d'authentification par token JWT
2. **Frontend**:
    - Templates Django avec Bootstrap pour l'interface administrateur et kiosk
    - JavaScript pour les fonctionnalités interactives (scan QR, lecture NFC)
3. **Application mobile**:
    - Flutter (framework Dart)
    - HTTP pour les appels API
    - Services d'authentification personnalisés

4. **Structure du projet**:

```
buskoguard_v2_mvp/
├── manage.py
├── Dockerfile, runtime.txt, .env, Procfile, entrypoint.sh
├── requirements.txt
├── populate_db.py
├── config/
│   └── (fichiers de configuration Django)
├── core/
│   ├── admin.py
│   ├── models.py
│   ├── views/
│   │   ├── employee_view.py
│   │   ├── kiosk_view.py
│   │   └── mobile_api_view.py
│   ├── serializers/
│   └── utils/
│       └── generate_qr_code.py
├── templates/
├── static/
│   └── js/
│       ├── qrcodescanner.js
│       └── NFCReader.js
└── employee_mobile_app/
    └── lib/
        ├── main.dart
        ├── constants.dart
        ├── services/
        │   └── authservice.dart
        ├── models/
        │   └── employee.dart
        └── screens/
            ├── loginscreen.dart
            └── employee_profilescreen.dart
```

C. Comment les composants QRCode et NFC sont-ils intégrés?

L'intégration des technologies QRCode et NFC est réalisée de manière minimaliste et efficace:

1. **QR Code**:
    - Génération: Le système génère automatiquement un QR code unique pour chaque employé via `generate_qr_code.py`
    - Stockage: Le code est stocké dans le modèle `Employee` en base de données
    - Lecture: L'interface kiosk utilise le fichier JavaScript `qrcodescanner.js` pour lire les codes présents sur l'application mobile ou imprimés
    - L'application mobile affiche le QR code de l'employé pour le pointage
2. **NFC**:
    - Identifiant: Chaque carte NFC possède un identifiant unique enregistré dans le modèle `Employee`
    - Lecture: Le système utilise `NFCReader.js` pour interfacer avec les lecteurs NFC physiques
    - Le système vérifie l'identifiant NFC contre la base de données pour valider l'identité
3. **Simplicité d'implémentation**:
    - Les traitements sont principalement réalisés côté client en JavaScript
    - L'authentification est gérée par de simples requêtes HTTP au backend
    - Cette approche réduit la complexité du système et facilite la maintenance

D. Quelle est l'architecture de la base de données ?

La base de données suit une structure relationnelle classique, optimisée pour la gestion des présences:

1. **Modèles principaux**:
    - `Department`: Départements de l'entreprise
    - `Role`: Postes ou fonctions des employés
    - `User`: Extension du modèle Django User avec informations personnelles
    - `Employee`: Informations professionnelles liées à un utilisateur
    - `Schedule`: Horaires planifiés pour chaque employé
    - `AttendanceRecord`: Enregistrements des pointages (entrée, sortie, pause)
    - `LeaveRequest`: Demandes et gestion des congés
2. **Relations clés**:
    - Un `User` correspond à un `Employee` (relation one-to-one)
    - Un `Employee` appartient à un `Department` et occupe un `Role`
    - Un `Employee` peut avoir plusieurs `Schedule` (horaires hebdomadaires)
    - Un `Employee` génère plusieurs `AttendanceRecord` (pointages)
    - Un `Employee` peut soumettre plusieurs `LeaveRequest` (congés)
3. **Particularités du modèle Employee**:

- Stockage des identifiants uniques (employee_id, nfc_id, qr_code)
- Relation directe avec le système d'authentification (User)
- Point central du système de pointage

Cette architecture de base de données simple mais complète permet de gérer efficacement les présences tout en maintenant une séparation claire entre les différentes entités du système.

4. Maintenance et support

A. Comment maintenir le système?

La maintenance du système BuskoGuard est conçue pour être simple et peu contraignante, conformément à la philosophie générale du projet. Voici les principaux aspects à considérer:

Mises à jour régulières

1. **Dépendances Python**:

```
bash
# Activation de l'environnement virtuel
source venv/bin/activate  # Linux/Mac
.\venv\Scripts\activate   # Windows

# Mise à jour des packages
pip install -r requirements.txt --upgrade

# Génération d'un nouveau requirements.txt après mises à jour
pip freeze > requirements.txt
```

2. **Application Flutter**:

```
bash
# Dans le dossier employee_mobile_app
flutter pub upgrade
```

3. **Django et sécurité**:

- Vérifiez régulièrement les bulletins de sécurité Django: https://www.djangoproject.com/weblog/
- Appliquez les correctifs critiques sans délai

Surveillance des performances

1. **Métriques à surveiller**:
   - Temps de réponse des requêtes API
   - Utilisation de la base de données
   - Taux d'erreur des scans QR et NFC
2. **Outils recommandés**:
   - Pour le déploiement local: Django Debug Toolbar
   - Pour le déploiement cloud: Outils de surveillance intégrés (Railway, Heroku, etc.)
   - Journaux d'application (logs)

Maintenance préventive

1. **Nettoyage périodique**:

```
python
# Script de nettoyage des données obsolètes (à exécuter via cron/tâche planifiée)
python manage.py clean_old_attendance_records --older-than=365  # Nettoie les enregistrements > 1 an
```

2. **Vérification d'intégrité**:

```
bash
# Vérification de la cohérence des données
python manage.py validate_employee_data
```

B. Quelles sont les procédures de sauvegarde et de récupération?

Une stratégie de sauvegarde fiable est essentielle, même pour un système simple comme BuskoGuard.

Sauvegarde de la base de données

1. **SQLite (déploiement local)**:

```
bash
# Sauvegarde manuelle
cp db.sqlite3 db.sqlite3.backup-$(date +%Y%m%d)
```

```
# Script automatisé de sauvegarde
#!/bin/bash
BACKUP_DIR="/path/to/backups"
DATE=$(date +%Y%m%d_%H%M%S)
cp db.sqlite3 "$BACKUP_DIR/db.sqlite3.backup-$DATE"
# Conserver uniquement les 10 dernières sauvegardes
ls -t "$BACKUP_DIR"/db.sqlite3.backup-* | tail -n +11 | xargs rm -f
```

2. **MySQL (déploiement cloud/Docker)**:

```
bash
# Sauvegarde
mysqldump -u username -p buskoguard > buskoguard_backup_$(date +%Y%m%d).sql

# Restauration
mysql -u username -p buskoguard < buskoguard_backup_20240515.sql
```

Sauvegarde des médias et configurations

1. **Fichiers média**:

```
bash
# Sauvegarde des fichiers média (photos de profil, QR codes générés)
rsync -av --delete /path/to/media/ /path/to/backup/media/
```

2. **Fichier .env et configurations**:
   - Sauvegardez régulièrement le fichier `.env` en lieu sûr
   - Documentez toutes les modifications de configuration

Plan de récupération

1. **Procédure de restauration complète**:

```
bash
# 1. Reconstruire l'environnement
python -m venv venv
source venv/bin/activate
pip install -r requirements.txt

# 2. Restaurer la base de données
```

```
cp /path/to/backup/db.sqlite3.backup-20240515 ./db.sqlite3

# 3. Restaurer les fichiers média
rsync -av /path/to/backup/media/ ./media/

# 4. Appliquer les migrations manquantes (si nécessaire)
python manage.py migrate
```

2. **Test de restauration**:
   - Testez régulièrement la procédure de restauration (au moins trimestriellement)
   - Documentez les résultats des tests de restauration

C. Comment résoudre les problèmes courants?

Problèmes d'authentification

1. **Utilisateur ne peut pas se connecter**:
   - Vérifiez l'exactitude des identifiants
   - Réinitialisez le mot de passe via l'interface d'administration
   - Vérifiez que le compte est actif (`is_active=True`)

```
python
# Dans une console Django (python manage.py shell)
from django.contrib.auth import get_user_model
User = get_user_model()
user = User.objects.get(username='utilisateur_problematique')
user.is_active = True
user.set_password('nouveau_mot_de_passe')
user.save()
```

2. **Problèmes de token JWT**:
   - Vérifiez la validité du token dans les en-têtes de requête
   - Assurez-vous que les horloges serveur/client sont synchronisées
   - Régénérez un token en cas de doute

Problèmes de scan QR/NFC

1. **Échec de lecture QR**:
   - Assurez-vous que l'éclairage est adéquat
   - Vérifiez que le QR code n'est pas endommagé
   - Testez avec l'identifiant manuel en alternative
   - Vérifiez les permissions de caméra dans le navigateur

2. **Problèmes de lecteur NFC**:
   - Redémarrez le lecteur NFC
   - Vérifiez les pilotes du lecteur
   - Testez avec une carte NFC connue pour fonctionner
   - Vérifiez la compatibilité du navigateur avec l'API Web NFC

```javascript
// Test de disponibilité NFC dans la console du navigateur
if ('NDEFReader' in window) {
  console.log('Web NFC API est disponible');
} else {
  console.error('Web NFC API n\'est pas prise en charge par ce navigateur');
}
```

Problèmes de performance

1. **Lenteur de l'interface d'administration**:
   - Activez le mode DEBUG temporairement pour identifier les requêtes lentes
   - Vérifiez les requêtes N+1 dans les modèles liés
   - Ajoutez des index à la base de données si nécessaire

```python
# Exemple d'optimisation de requête dans views.py
from django.db.models import Prefetch

# Avant: peut générer des requêtes N+1
employees = Employee.objects.all()

# Après: optimisé avec prefetch_related
employees = Employee.objects.prefetch_related(
    'user', 'department', 'role'
).all()
```

2. **Problèmes de l'application mobile**:
   - Vérifiez la connectivité réseau
   - Effacez le cache de l'application
   - Assurez-vous que l'URL de l'API est correcte dans `constants.dart`
   - Vérifiez les logs de l'application Flutter

```dart
// Ajoutez temporairement des logs de débogage dans les services
print('Tentative de connexion à: $baseUrl');
```

```
      print('Réponse reçue: ${response.statusCode}');
```

Journal des erreurs

Maintenez un journal des problèmes rencontrés et de leurs solutions pour accélérer la résolution future:

```
| Date       | Problème                     | Solution                          | Prévention                    |
|------------|------------------------------|-----------------------------------|-------------------------------|
| 2024-05-10 | Échec des scans QR           | Mise à jour de la bibliothèque JS | Vérification mensuelle des MAJ   |
| 2024-05-12 | Lenteur de l'interface admin | Optimisation des requêtes         | Monitoring des performances      |
```

5. Perspectives d'évolution

A. Quelles sont les améliorations futures possibles?

Bien que BuskoGuard soit conçu comme une solution simple face au problème de d'appointage, mais plusieurs améliorations pourraient enrichir ses fonctionnalités tout en préservant sa philosophie minimaliste.

Améliorations fonctionnelles

1. **Tableau de bord analytique**
    - Statistiques de présence en temps réel
    - Visualisation des tendances (retards, absences, heures supplémentaires)
    - Rapports automatisés envoyés par email aux responsables
2. **Système de notification avancé**
    - Alertes par SMS/email pour les retards
    - Rappels automatiques pour les gestionnaires concernant les demandes de congé en attente
    - Notifications push dans l'application mobile pour les changements d'horaire
3. **Gestion des horaires flexible**
    - Support des horaires flexibles avec plages de présence obligatoire
    - Possibilité d'accumulation d'heures supplémentaires compensées
    - Planification d'équipes avec rotation automatique

Améliorations techniques

1. **Optimisation des performances**
    - Mise en cache avancée pour réduire les temps de chargement
    - Optimisation des requêtes pour supporter un plus grand nombre d'employés
    - Compression des données d'historique pour une conservation à long terme
2. **Renforcement de la sécurité**
    - Validation multi-facteurs pour les fonctions administratives
    - Chiffrement de bout en bout des données sensibles
    - Journalisation avancée des actions administratives (audit trail)
3. **Amélioration de la fiabilité**
    - Mode hors ligne pour l'application mobile avec synchronisation ultérieure
    - Système de détection et prévention des fraudes (double pointage, pointage par procuration)
    - Tolérance aux pannes améliorée pour les environnements instables

B. Comment le système pourrait-il être étendu?

Au-delà des améliorations, BuskoGuard pourrait évoluer vers un système plus complet de gestion des ressources humaines et du temps de travail.

Extensions fonctionnelles

1. **Module de gestion du temps complet**
    - Intégration avec la paie basée sur les heures travaillées
    - Gestion des heures supplémentaires et calcul automatique
    - Suivi du temps par projet/tâche pour la facturation client
2. **Extension vers la gestion RH**
    - Suivi des évaluations de performance
    - Gestion des formations et certifications
    - Module de recrutement intégré avec suivi des candidatures
3. **Contrôle d'accès physique**
    - Intégration avec des serrures électroniques
    - Gestion des zones d'accès restreint
    - Journal des entrées/sorties pour la sécurité physique

Extensions d'échelle

1. **Support multi-site**
    - Gestion centralisée avec déploiement distribué
    - Synchronisation inter-sites pour les employés mobiles
    - Rapports consolidés pour les entreprises multi-succursales
2. **Architecture microservices**
    - Décomposition du monolithe en services spécialisés
    - API Gateway pour unifier l'accès aux différents services
    - Évolutivité indépendante pour chaque composant du système
3. **Version SaaS (Software as a Service)**
    - Offrir BuskoGuard comme service cloud multi-tenant

- Modèle d'abonnement basé sur le nombre d'employés
- Personnalisation par entreprise tout en partageant l'infrastructure

C. Quelles autres technologies pourraient être intégrées?

L'intégration de nouvelles technologies pourrait renforcer la valeur et les capacités de BuskoGuard.

Technologies d'identification avancées

1. **Biométrie**
   - Reconnaissance faciale pour le pointage sans contact
   - Empreintes digitales comme alternative aux cartes NFC
   - Reconnaissance vocale pour les confirmations de présence
2. **Géolocalisation intelligente**
   - Pointage automatique basé sur la présence dans une zone géographique
   - Vérification de la position GPS lors du scan de QR code
   - Zones virtuelles (geofencing) pour les équipes mobiles
3. **Blockchain pour l'intégrité des données**
   - Horodatage infalsifiable des pointages
   - Preuve cryptographique de présence
   - Audit transparent des modifications de données

Intelligence artificielle et automatisation

1. **Prévision et optimisation**
   - Prédiction des absences et retards récurrents
   - Planification optimisée des horaires selon les tendances historiques
   - Détection d'anomalies dans les habitudes de pointage
2. **Assistants virtuels**
   - Chatbot pour répondre aux questions fréquentes des employés
   - Traitement automatisé des demandes de congé simples
   - Rappels personnalisés basés sur les habitudes de l'employé
3. **Vision par ordinateur**
   - Analyse de l'occupation des espaces de travail
   - Détection automatique de présence dans les salles de réunion
   - Comptage anonyme pour optimiser l'utilisation des locaux

Intégrations avec l'écosystème d'entreprise

1. **API étendues**
   - Intégration avec les principaux ERP (SAP, Oracle, etc.)
   - Connecteurs pour les systèmes de paie populaires
   - Webhooks pour déclencher des actions dans d'autres systèmes
2. **IoT (Internet des Objets)**

    - Capteurs de présence dans les espaces de travail
    - Badges intelligents avec affichage e-ink
    - Intégration avec les systèmes domotiques de bureau
3. **Outils collaboratifs**
    - Synchronisation avec les calendriers d'équipe (Google Calendar, Outlook)
    - Intégration avec les plateformes de communication (Slack, Microsoft Teams)
    - Statut de présence automatiquement mis à jour dans les outils collaboratifs

6. Conclusion

Au terme de ce projet, je peux affirmer que BuskoGuard répond à son objectif fondamental : transformer un processus d'appointage manuel observé à l'ISTEAH en un système numérique léger et accessible. Cette réalisation démontre qu'une approche minimaliste, privilégiant la simplicité plutôt que la complexité, peut effectivement moderniser des processus administratifs quotidiens.

Ce projet m'a permis d'appliquer diverses connaissances acquises durant mon parcours académique. Plusieurs cours ont été particulièrement utiles dans cette réalisation:

- INF3300 (Base de données) pour la conception et l'optimisation du modèle de données
- INF1000, INF2000 et INF3500 qui m'ont introduit aux fondamentaux de la programmation
- LOG3300 (Développement web) pour la création de l'interface avec Django
- INF4100 (Systèmes répartis et cloud) pour les aspects de déploiement et de scalabilité

La réalisation de BuskoGuard démontre qu'une solution technique bien pensée, même simple, peut apporter une amélioration significative à un processus quotidien.

Le système actuel représente une base solide sur laquelle construire. Les perspectives d'évolution détaillées précédemment offrent un chemin de développement qui permettra d'enrichir progressivement les fonctionnalités tout en préservant l'accessibilité et la facilité de maintenance qui constituent l'essence même de BuskoGuard.