OpenData

Présentation

Le programme permet de visualiser la disponibilité des parkings en temps réel. Il effectue les actions suivantes :

- Récupère régulièrement des données depuis une source et les sauvegarde sous forme de fichier JSON.
- Extraction des données :
  - Nom du parking
  - Places disponibles
  - Capacité maximale
- **Génération d'un histogramme** montrant la disponibilité de tout les parkings actuel (Ce graphique est mis a jour lors de chaque collecte de données)

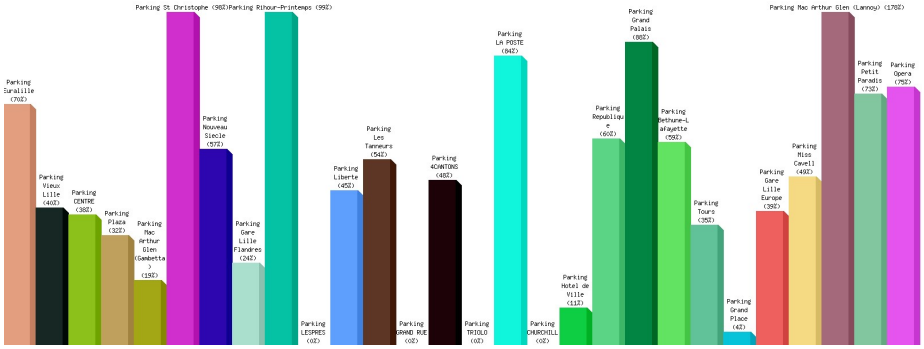

Figure 1: PARKING_REMOVE

- **Sauvegarde des données** : chaque parking a un fichier .json correspondant dans le répertoire **Data_parkings** contenant un maximum de 13 entrées sauvegardant les 13 dernieres collectes de données (Places_Dispo/Places_Max/Data et heure de la collecte).

Note: À son lancement, le programme affiche des graphiques pour chaque parking ainsi que le decompte avant la prochaine collecte de donnée **Prochaine collecte dans 9 minutes 58 secondes...**

Détails Techniques

Source des Données

- Sujet choisi : Disponibilité temps réel des parkings MEL
- Formats disponibles : .json
- URL stable pour le fichier .json : Lien direct

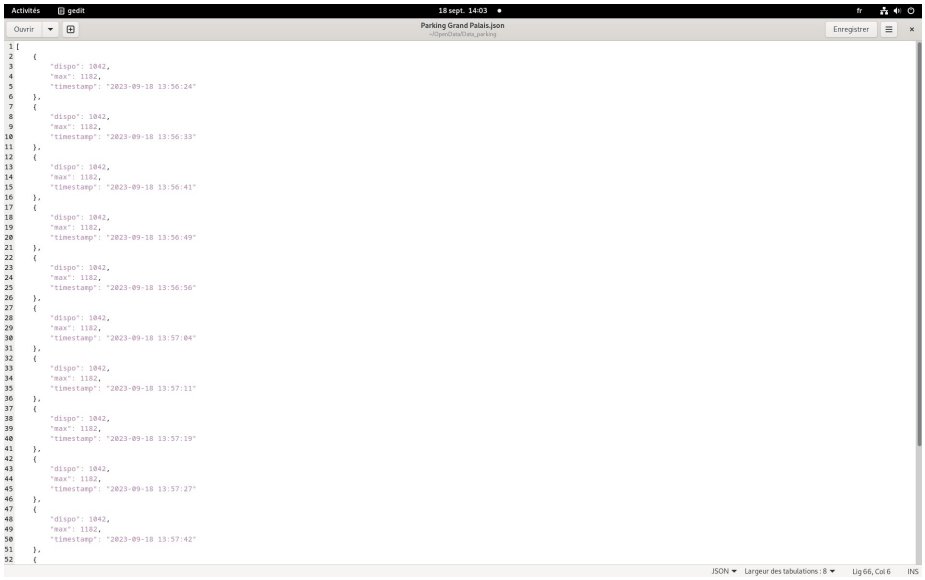

Figure 2: parking_data

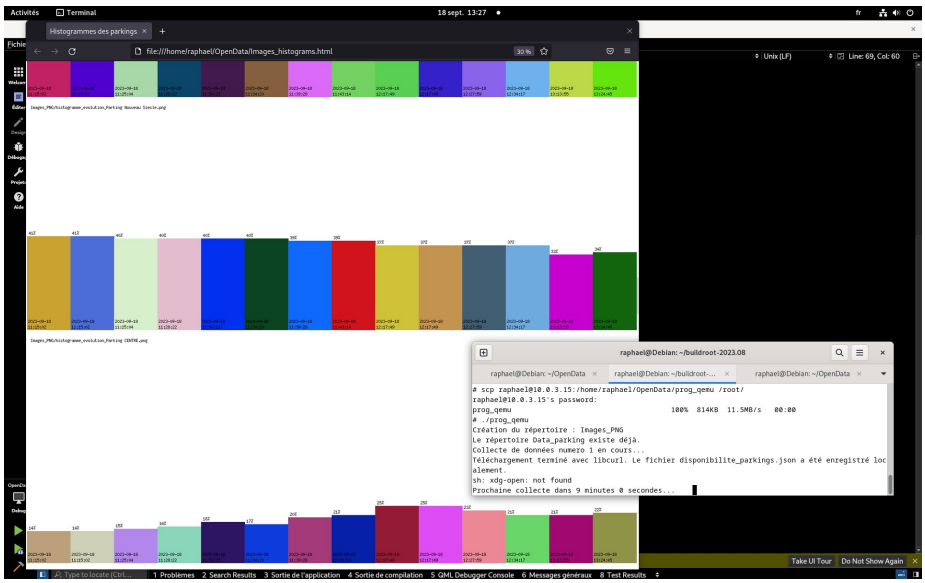

Figure 3: parking9

Méthodologie

- **Téléchargement** : Utilise `libcurl` pour télécharger le fichier .json.
- **Parsing** : Le fichier .JSON est ensuite parsé pour en extraire les données nécessaires.
- **Génération du graphique** : Un histogramme est créé à partir des données récupérées, en utilisant la bibliothèque GD.

Objectif

Deux fichiers principaux sont générés :

1. **Taux_dispo_actuel_TOUT_parkings.png** : Un histogramme montrant la disponibilité de chaque parking de la derniere collecte de donnée.

Figure 4: parking11

2. **Nom_du_parking_evolution.png** (exemple de nom) : Ce fichier, situé dans le répertoire **Image_PNG**, est mis à jour à chaque exécution du programme et a chaque collecte de donnée pour afficher l'évolution de la disponibilité du parking correspondant.

Documentation Technique : OpenData Parking MEL

Executer cette commande : `git clone https://github.com/raph5640/OpenData.git`

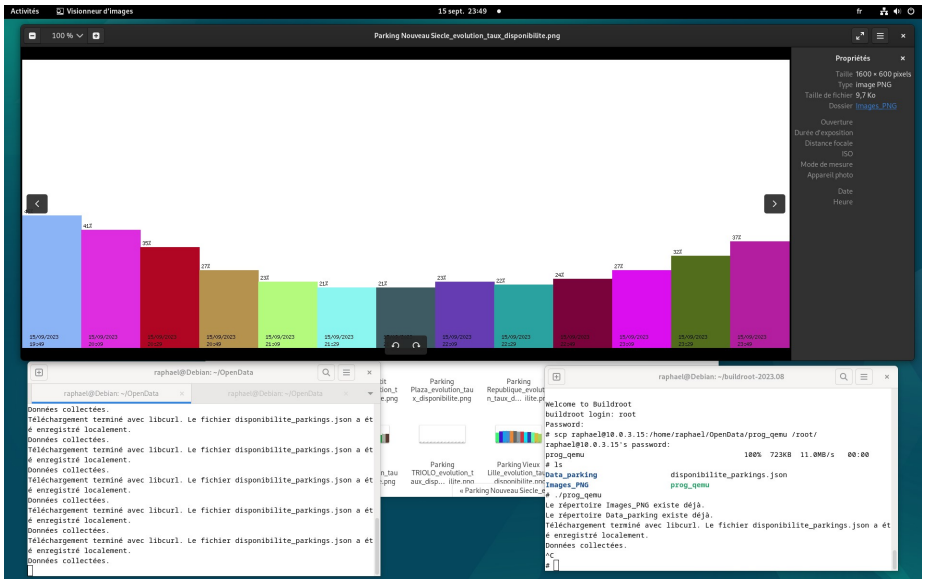

Figure 5: evolution_parking5

Pour visualiser la documentation **Doxygen** depuis le repertoire `/OpenData` : `firefox docs/html/index.html`

1. Présentation du projet

- **Fonction principale** : Ce programme vise à récupérer des données en temps réel sur les parkings depuis une URL spécifiée et à les afficher sous forme d'histogrammes. Les histogrammes montrent la disponibilité actuelle des parkings et l'évolution de la disponibilité pour chaque parking.
- **Source des données** : https://opendata.lillemetropole.fr//explore/dataset/disponibilite-parkings/download?format=json&timezone=Europe/Berlin&use_labels_for_header=false

2. Dépendances

- **libcurl** : Pour télécharger le fichier .json depuis l'URL
- **nlohmann/json.hpp** : Une bibliothèque JSON pour C++ utilisée pour parser le fichier .json
- **gd** : Bibliothèque pour la génération de graphiques.

3. Compilation

Debian:

1. Cloner le dépôt: `git clone https://github.com/raph5640/OpenData.git`
2. Accéder au répertoire: `cd OpenData/`

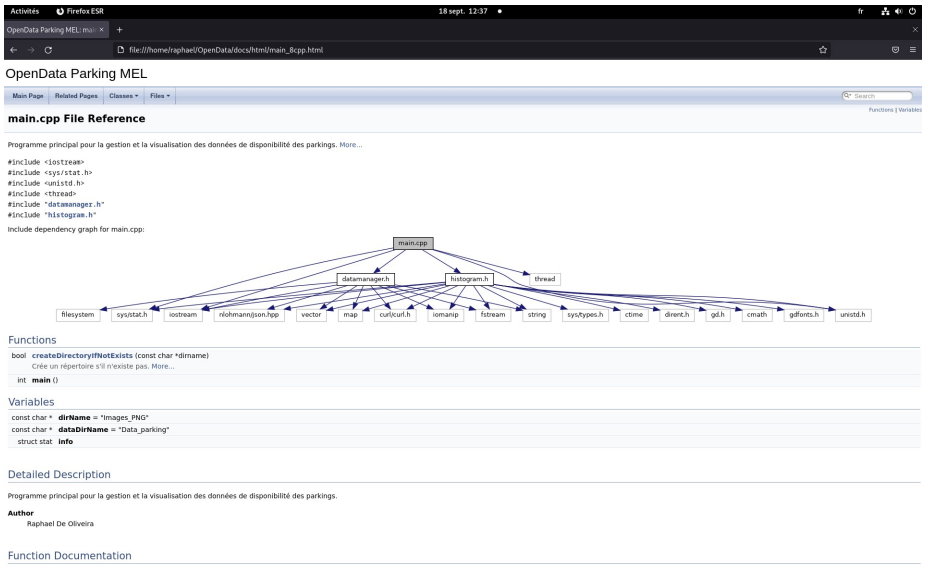

Figure 6: doxygen3

3. Télécharger et installer la dépendance JSON: `git clone https://github.com/nlohmann/json` ou `sudo apt install ljsoncpp`
4. **Compiler: `g++ -o prog_debian main.cpp histogram.cpp datamanager.cpp -lgd -lcurl -ljsoncpp -I/home/raphael/json/include`**
5. Exécuter: `./prog_debian`

Configuration de Buildroot et QEMU

1. Configuration de Buildroot

1. Accédez à votre répertoire Buildroot : `cd /home/raphael/buildroot-2023.08`
2. Lancez `make xconfig`.
3. Activez les options suivantes :
   - `libcurl`
   - `json-for-modern-cpp` et `libjsoncpp` (BR2_PACKAGE_JSON_FOR_MODERN_CPP et BR2_PACKAGE_LIBJSON)
   - Bibliothèque `gd` (BR2_PACKAGE_GD) et `gdtopng`
   - `Enable C++ support`
   - Faire un `make linux-xconfig` et activez également les options liées au framebuffer :
     - Support for frame buffer devices (CONFIG_FB)
     - Frame buffer hardware drivers (sélectionnez le matériel spécifique que vous utilisez)
     - Virtio GPU driver (CONFIG_DRM_VIRTIO_GPU)

4. Lancez la compilation : `make`.
5. Pour la compilation croisée : `~/buildroot-2023.08/output/host/bin/aarch64-buildroot-linux-gnu-g++ ~/OpenData/main.cpp ~/OpenData/histogram.cpp ~/OpenData/datamanager.cpp -o ~/OpenData/prog_qemu -lgd -lcurl -lstdc++fs -ljsoncpp -I/home/raphael/json/include`

2. Transfert et Exécution sur Buildroot

1. Démarrer Buildroot : `./go` ou `./start_buildroot.sh`.
2. Transférez `prog_qemu` sur Buildroot : `scp raphael@10.0.3.15:/home/raphael/OpenData/prog_qemu /root/`
3. Dans Buildroot, naviguez vers `/root/` et exécutez : `./prog_qemu`.
4. Le programme est en mesure d'afficher le taux de disponibilité de chaques parking lors de la collecte de donnée sur le terminale.

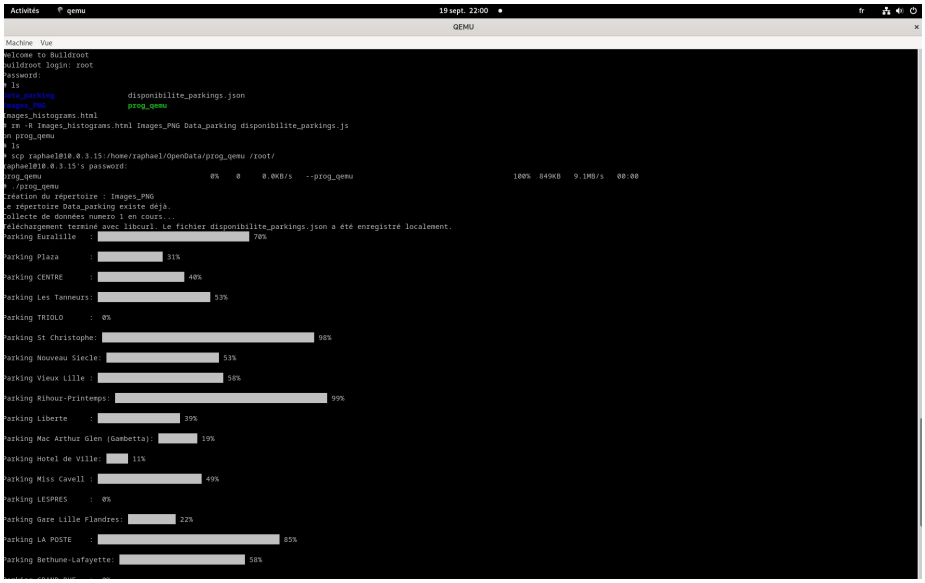

Figure 7: TERMINALESHOOOW

Note: Lorsque votre terminale QEMU s'affiche aller dans le terminale **BarMenu -> Vue -> Activer serial0**.

3. Affichage avec Framebuffer

1. Démarrez votre machine virtuelle Buildroot en utilisant le framebuffer dans votre shellscript de démarrage `./start_VM.sh` :

```
#! /bin/sh

#author Raphael De Oliveira
```

```
qemu-system-aarch64 -M virt \  -cpu cortex-a57 \  -smp 1 \  -kernel
output/images/Image \    -append "root=/dev/vda console=ttyAMA0" \
-netdev user,id=eth0,hostfwd=tcp::2222-:22,hostfwd=tcp::8888-:80
-device virtio-net-device,netdev=eth0 \-drive file=output/images/rootfs.ext4,if=none,format
\  -device virtio-blk-device,drive=hd0 \  -device virtio-gpu-pci \
-usb \-device nec-usb-xhci \-device usb-tablet
```

1) exécutez : `fbset -g 1600 600 1600 600 32`
2) Vous pouvez utiliser la commande : `fbv Taux_dispo_actuel_TOUT_parkings.png` pour afficher l'image maintenant.
3) Pour connaitre les dimensions de votre frambuffer utiliser : `fbset`
4) si l'image dépasse la résolution defini par QEMU faites un `convert Taux_dispo_actuel_TOUT_parkings.png -resize 978x600! Taux_dispo_actuel_TOUT_parkings_re` pour redefinir les dimensions de votre images.

- Pour utiliser convert il faut activer avant dans `make xconfig -> ImageMagick`

4. Affichage avec adresse http, configuration du serveur Web lighttpd

(Si vous voulez afficher le contenu généré par prog_qemu de votre machine buildroot dans un navigateur depuis votre machine hote)

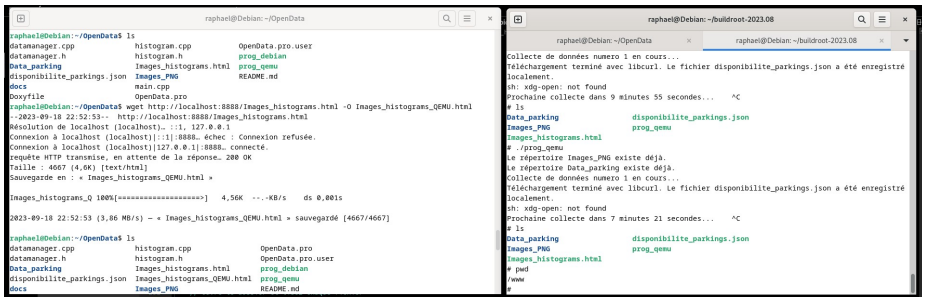

Figure 8: serveur_web

1. Démarrez votre machine virtuelle Buildroot :

```
bash qemu-system-aarch64 -M virt \ -cpu cortex-a57 \ -nographic
\ -smp 1 \ -kernel output/images/Image \ -append "root=/dev/vda
console=ttyAMA0" \ -netdev user,id=eth0,hostfwd=tcp::2222-:22,hostfwd=tcp::8888-:80
-device virtio-net-device,netdev=eth0 \ -drive file=output/images/rootfs.ext4,if=none,format
\ -device virtio-blk-device,drive=hd00
```

2. Lancez `lighttpd` : `lighttpd -f /etc/lighttpd/lighttpd.conf` (Activer le avec un make xconfig avant)
3. Créez un répertoire pour les fichiers web : `mkdir /www`
4. Déplacez les fichiers nécessaires :

```
mv /root/Data_parking /www

mv /root/Images_PNG /www

mv /root/prog_qemu /www

mv /root/Images_histograms.html /www

mv /root/disponibilite_parkings.json /www
```

5. Ajustez les autorisations : `chmod -R 755 /www/`
6. Modifiez le fichier de configuration de lighttpd : `vi /etc/lighttpd/lighttpd.conf` et ajoutez/modifiez la ligne : `server.document-root = "/www"`
7. Redémarrez lighttpd : `lighttpd -f /etc/lighttpd/lighttpd.conf`
8. Testez depuis la machine hôte : `wget http://localhost:8888/Images_histograms.html -O Images_histograms_QEMU.html`
9. Ouvrir dans un navigateur web de votre choix, depuis votre machine hôte (Linux/Debian) **`http://localhost:8888/Images_histograms.html`** :

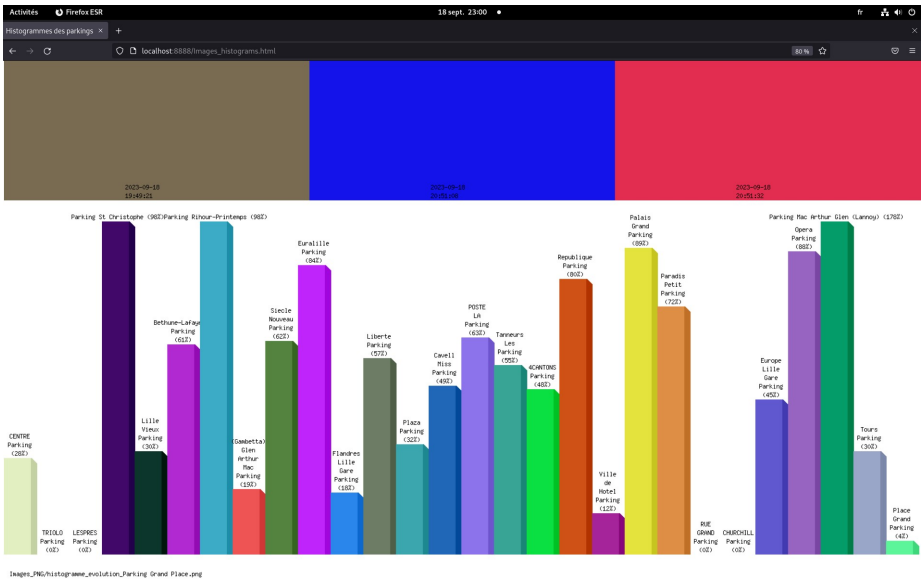

Figure 9: serveur_web2

5. Annexes

- **Images** : Consultez les images exemple sur GitHub dans le répertoire `Images_PNG`.