








Rapport









- Lancement de bitcoin-core :

| Data (D:) > Programation > bitcoin-core > Bitcoin | |
|--|------------------|
| Nom | Modifié le |
|  daemon | 02/10/2024 19:35 |
|  share | 02/10/2024 19:35 |
|  bitcoin | 23/01/2025 12:19 |
|  bitcoin-qt | 02/10/2024 19:35 |
|  COPYING | 02/10/2024 19:35 |
|  readme | 02/10/2024 19:35 |
|  uninstall | 02/10/2024 19:35 |

```
./bitcoin-qt.exe -regtest
```

The screenshot shows the Bitcoin Core GUI. At the top is a menu bar with 'Fichier', 'Paramètres', 'Fenêtre', and 'Aide'. Below it is a toolbar with icons for 'Vue d'ensemble', 'Envoyer', 'Recevoir', and 'Transactions'. The main area displays a warning dialog box with a yellow triangle icon containing an exclamation mark. The text in the dialog reads: 'Les transactions récentes ne sont peut-être pas encore visibles et par conséquent le solde de votre porte-monnaie est peut-être erroné. Ces renseignements seront justes quand votre porte-monnaie aura fini de se synchroniser avec le réseau Bitcoin, comme décrit ci-dessous. **Toute tentative de dépense de bitcoins affectés par des transactions qui ne sont pas encore affichées ne sera pas acceptée par le réseau.**' Below this text is a table with two columns. The first column lists synchronization status items, and the second column shows their current values. A 'Cacher' button is located at the bottom right of the dialog box. At the very bottom of the screenshot, a status bar shows 'Connexion aux pairs...' followed by a green box containing the text 'en retard de 13 ans et 50 semaines', and the Bitcoin logo on the right.

- Lancement de lightning :

| » Data (D:) » Programmation » lightning » clightning-v24.08.2 » clightning-v24.08.2 | | |
|--|------------------|---------------------|
| Nom | Modifié le | Type |
|  .github | 06/11/2024 15:15 | Dossier de fichiers |
|  bitcoin | 06/11/2024 15:15 | Dossier de fichiers |
|  ccan | 06/11/2024 15:15 | Dossier de fichiers |
|  channeld | 06/11/2024 15:15 | Dossier de fichiers |
|  cli | 06/11/2024 15:15 | Dossier de fichiers |
|  cln-grpc | 06/11/2024 15:15 | Dossier de fichiers |
|  cln-rpc | 06/11/2024 15:15 | Dossier de fichiers |
|  closingd | 06/11/2024 15:15 | Dossier de fichiers |

```
marc@LAPTOP-F677G004:/mnt/d$ lightningd --version
v24.08.2
```

```
lightningd --network=regtest --daemon --log-file=/mnt/d/Programation/lightning/lightningd.log
--lightning-dir=/mnt/d/Programation/lightning
```

Website

- pip install flask pylightning

Pour interagir avec core lightning

- python app.py



Explication du code

```
from flask import Flask, request, jsonify
from lightning import LightningRpc
import os
```

```
app = Flask(__name__)
```

- **LightningRpc** est la classe de la librairie **pylightning** qui permet de communiquer avec **Core Lightning (c-lightning)** via un fichier socket local (lightning-rpc).

```
LIGHTNING_RPC_PATH = "/mnt/d/Programation/lightning/lightning-rpc"
```

```
# Initialisation de la connexion RPC
lightning = LightningRpc(LIGHTNING_RPC_PATH)
```

- **LIGHTNING_RPC_PATH** : c'est le chemin local vers le fichier lightning-rpc que crée le démon lightningd (Core Lightning) lorsqu'il tourne.
- **LightningRpc(LIGHTNING_RPC_PATH)**
On crée un objet lightning qui sert de **client** pour toutes les opérations JSON-RPC avec le nœud c-lightning.
Grâce à cet objet, on peut appeler des méthodes comme invoice(...), listinvoices(...), etc.

1. Route d'accueil

```
@app.route("/")
def index():
```

- Page d'accueil par défaut

2. Création d'une facture Lightning : "/create_invoice"

```
@app.route("/create_invoice", methods=["POST"])
def create_invoice():
```

- **@app.route("/create_invoice", methods=["POST"]) :**
Cette route est appelée quand le formulaire de la page d'accueil est soumis via POST.
- **amount_sats = request.form.get("amount", "1000")**
On récupère la valeur du champ amount transmis par le formulaire.
Si ce champ est vide, on utilise par défaut "1000" satoshis.
- **label = "facture_" + os.urandom(4).hex()**
Pour c-lightning, le paramètre label doit être **unique** pour chaque facture.
Ici, on génère un label aléatoire en hexadécimal à partir de 4 octets aléatoires (os.urandom(4)).
- **description = "Paielement de test depuis Flask"**
C'est la description qui apparaîtra dans la facture (certains wallets LN l'affichent).
- **lightning.invoice(...)**
Méthode c-lightning pour créer une nouvelle facture.
Paramètres :
 - o **msatoshi=int(amount_sats) * 1000**
 - c-lightning attend un montant en **millisatoshis** (msatoshi).
 - Un satoshi = 1000 millisatoshis, donc on multiplie par 1000 pour passer de sats à msats.
 - o **label=label**
 - Le label unique qu'on vient de créer.
 - o **description=description**
 - La description textuelle à inclure dans la facture.
- **invoice = lightning.invoice(...)** renvoie un objet/dictionnaire contenant notamment :
"bolt11" : la facture BOLT11 elle-même (chaîne de caractères commençant par lnbc...).
"payment_hash", "expires_at", etc.
- **bolt11 = invoice["bolt11"]**
On stocke la facture BOLT11 dans la variable bolt11.
- **Retour d'un bloc HTML :**
On affiche la facture BOLT11 pour que l'utilisateur puisse la copier/coller ou la scanner.
On ajoute un lien "Vérifier le statut de la facture" pointant vers **/check_invoice/{label}**.
{label} est le label unique, qui permettra de retrouver la facture pour savoir si elle est payée.

2. Vérification du statut de la facture :

```
@app.route("/check_invoice/<label>", methods=["GET"])
def check_invoice(label):
```

- **@app.route("/check_invoice/<label>", methods=["GET"])**
 - On déclare une route où label est un paramètre variable dans l'URL.
 - Par exemple, si on visite /check_invoice/facture_abcd1234, alors label = "facture_abcd1234" dans la fonction.
- **invoices = lightning.listinvoices(label)["invoices"]**
 - On appelle la commande listinvoices de c-lightning pour récupérer toutes les factures portant ce label.
 - Le résultat est un dictionnaire qui contient une clé "invoices" (une liste).
 - Ici, on l'extrait directement : ["invoices"].
- **if not invoices:**
 - Si la liste est vide, c'est qu'il n'y a pas de facture correspondant à ce label.
 - On renvoie un message "Aucune facture trouvée avec ce label."
- **invoice = invoices[0]**
 - Sinon, on prend la première (normalement il ne devrait y en avoir qu'une).
- **status = invoice["status"]**
 - Le statut d'une facture c-lightning peut être :
 - "unpaid" (jamais payée et toujours valide),
 - "paid" (déjà payée),
 - "expired" (délai expiré, non payée).
- **En fonction du statut**
 - Si c'est "paid", on affiche "La facture {label} est payée !".
 - Sinon, on affiche "Statut : ...".