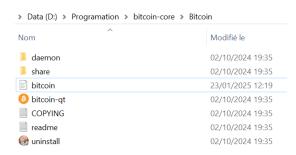
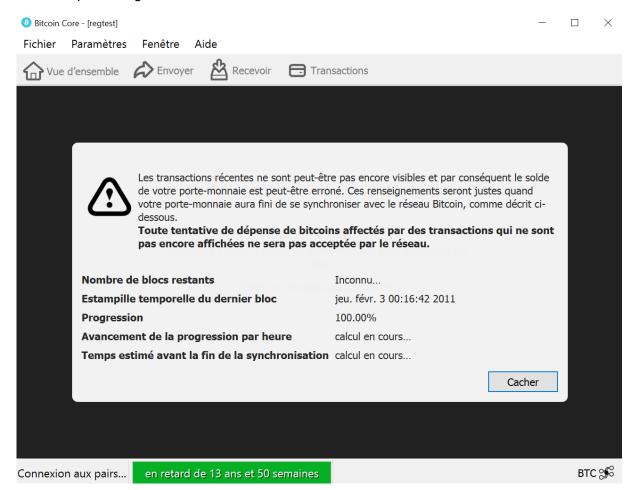
Rapport

- Lancement de bitcoin-core :



./bitcoin-qt.exe -regtest



- Lancement de lightning :

> Data (D:) > Programation > lightning > clightning-v24.08.2 > clightning-v24.08.2		
.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-F677GO04:/mnt/d\$ lightningd --version v24.08.2

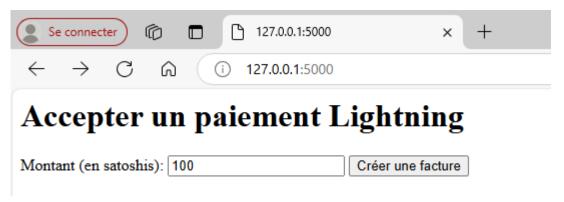
lightningd --network=regtest --daemon --log-file=/mnt/d/Programation/lightning/lightningd.log --lightning-dir=/mnt/d/Programation/lightning

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.

lci, on génère un label aléatoire en hexadécimal à partir de 4 octets aléatoires (os.urandom(4)).

- description = "Paiement 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 (chaine de caractères commençant par Inbc...).

"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 :
 - o "unpaid" (jamais payée et toujours valide),
 - o "paid" (déjà payée),
 - o "expired" (délai expiré, non payée).

- En fonction du statut

- Si c'est "paid", on affiche "La facture {label} est payée !".
- Sinon, on affiche "Statut: ...".