Lesson 7: Building on LND

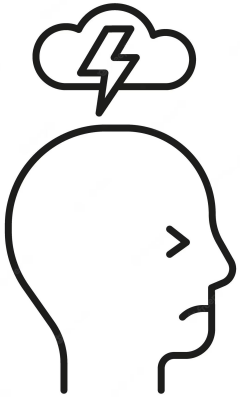
What is a Lightning App / Talking to your node / LND API / Building our LND methods

What is a Lightning App

A Lightning App is just like any other application but with Lightning added.

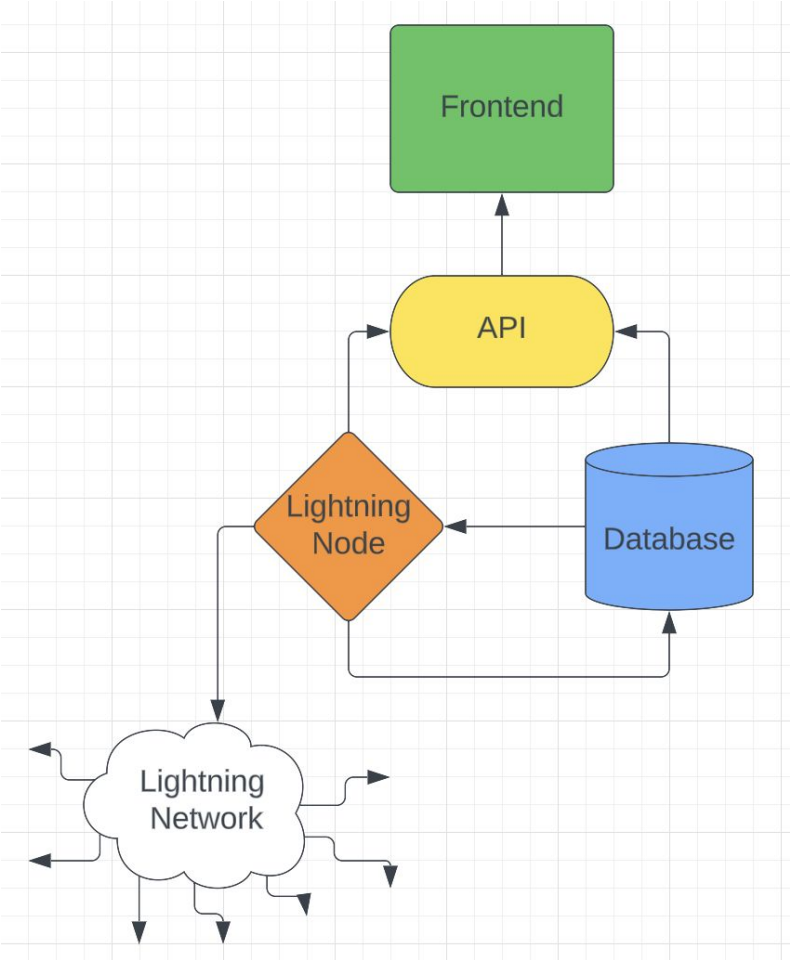
This can mean a lot of different things though:

- Does your Lightning app need a dedicated node?
- Will your Lightning app use some kind of LSP (Lightning Service Provider) for the required features?
- Will your user need to run a node and connect it somehow to use your app?
- Will the user be given a custodial wallet by you?
- Will the user need to connect their own wallet?
- Will the user be given a non-custodial wallet by you?
- Will the user be given a node by you?
- Will the user even know they're using Lightning?



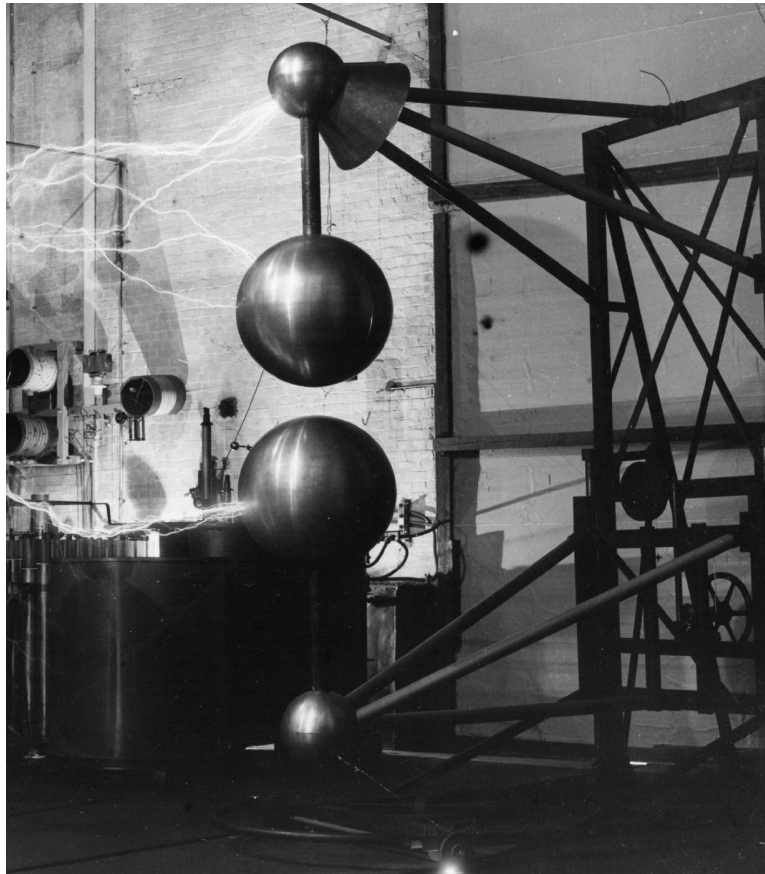
A typical fullstack Lightning App

- **Frontend:**
  - What your users are interacting with **UX**
  - How it looks and functions **UI**
- **API:**
  - Your apps communication layer (likely HTTP)
  - get, post, update, delete data from your database
  - Call methods on your lightning node (with websockets, GRPC, or HTTP)
  - Possibly middleware (some kind of server-side actions / functions / logic)
- **Database:**
  - Potentially optional depending on the app
  - Will store all app data
  - You may need to store Lightning node data as well
- **Lightning Node:**
  - Functions as wallet, node, and payment db.
  - It may be your node (custodial)
  - It may require your users to connect their node.



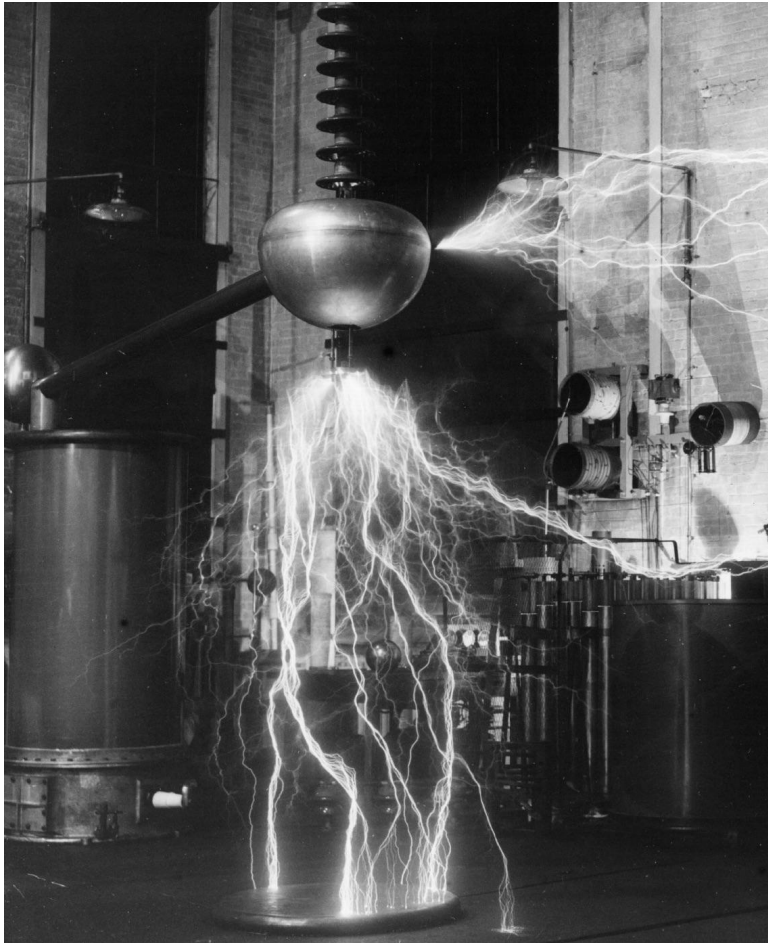
Common hurdles to building your first Lightning App

- Running a node / hosting / maintaining
- Talking to your node
- Developing with a node
- Getting liquidity
- Wallet Management



Tools to help with these common hurdles

- Running a node / hosting / maintaining
  - Voltage
  - Umbrel
- Talking to your node
  - gRPC wrappers (LND-GRPC, In-service)
  - Interfaces (LNBits, RTL, WebLN, LNC)
- Developing with a node
  - Polar
  - Workbench (LND / CLN)
- Getting liquidity
  - FLOW by Voltage
  - Magma by Amboss
- Wallet management
  - LNBits
  - LNPay
  - RTL

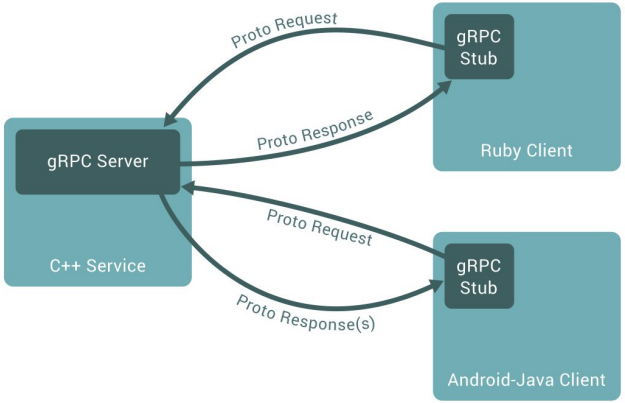


Talking to your node

One of my greatest hurdles when learning how to build Lightning apps was figuring out how to setup a connection to my node so I could call methods on it and read data from it.

gRPC

- Stands for "gRPC Remote Procedure Calls"
- Modern, high-performance framework for communication between services
- Based on Protocol Buffers: A language-neutral, platform-neutral data serialization format
- Efficient binary format: Results in smaller payloads and faster data transmission compared to text-based formats like JSON
- Strongly-typed: Ensures data consistency and reduces the likelihood of errors
- Supports multiple programming languages: gRPC libraries are available for popular languages like Python, JavaScript, Java, and more
- Streaming support: Allows sending multiple messages between client and server efficiently, with lower latency
- Wrapper libraries: Simplify gRPC usage and reduce boilerplate code, making it easier to interface with services like the Lightning Network implementations



Benefits of using a gRPC wrapper library

- Less code
- Easier to learn
- Exposes the full LND API via simple async methods
- Focus on the API / types / params, not the boilerplate

```
1   const LndGrpc = require("lnd-grpc");
2   const dotenv = require("dotenv");
3
4   dotenv.config();
5
6   const options = {
7     host: process.env.LND_HOST,
8     cert: process.env.LND_CERT,
9     macaroon: process.env.LND_MACAROON,
10  };
11
12  const lnd = new LndGrpc(options);
13
14  const connect = async () => {
15    await lnd.connect();
16
17    console.log("LND gRPC client is ready to use");
18  };
```

Making a request for getInfo without a gRPC wrapper

```
const fs = require('fs');
const grpc = require('@grpc/grpc-js');
const protoLoader = require('@grpc/proto-loader');

const GRPC_HOST = 'localhost:10009'
const MACAROON_PATH = 'LND_DIR/data/chain/bitcoin/regtest/admin.macaroon'
const TLS_PATH = 'LND_DIR/tls.cert'

const loaderOptions = {
  keepCase: true,
  longs: String,
  enums: String,
  defaults: true,
  oneofs: true,
};
const packageDefinition = protoLoader.loadSync('lightning.proto', loaderOptions);
const lnrpc = grpc.loadPackageDefinition(packageDefinition).lnrpc;
process.env.GRPC_SSL_CIPHER_SUITES = 'HIGH+ECDSA';
const tlsCert = fs.readFileSync(TLS_PATH);
const sslCreds = grpc.credentials.createSsl(tlsCert);
const macaroon = fs.readFileSync(MACAROON_PATH).toString('hex');
const macaroonCreds = grpc.credentials.createFromMetadataGenerator(function(args,
callback) {
  let metadata = new grpc.Metadata();
  metadata.add('macaroon', macaroon);
  callback(null, metadata);
});
let creds = grpc.credentials.combineChannelCredentials(sslCreds, macaroonCreds);
let client = new lnrpc.Lightning(GRPC_HOST, creds);
let request = {};
client.getInfo(request, function(err, response) {
  console.log(response);
});
```

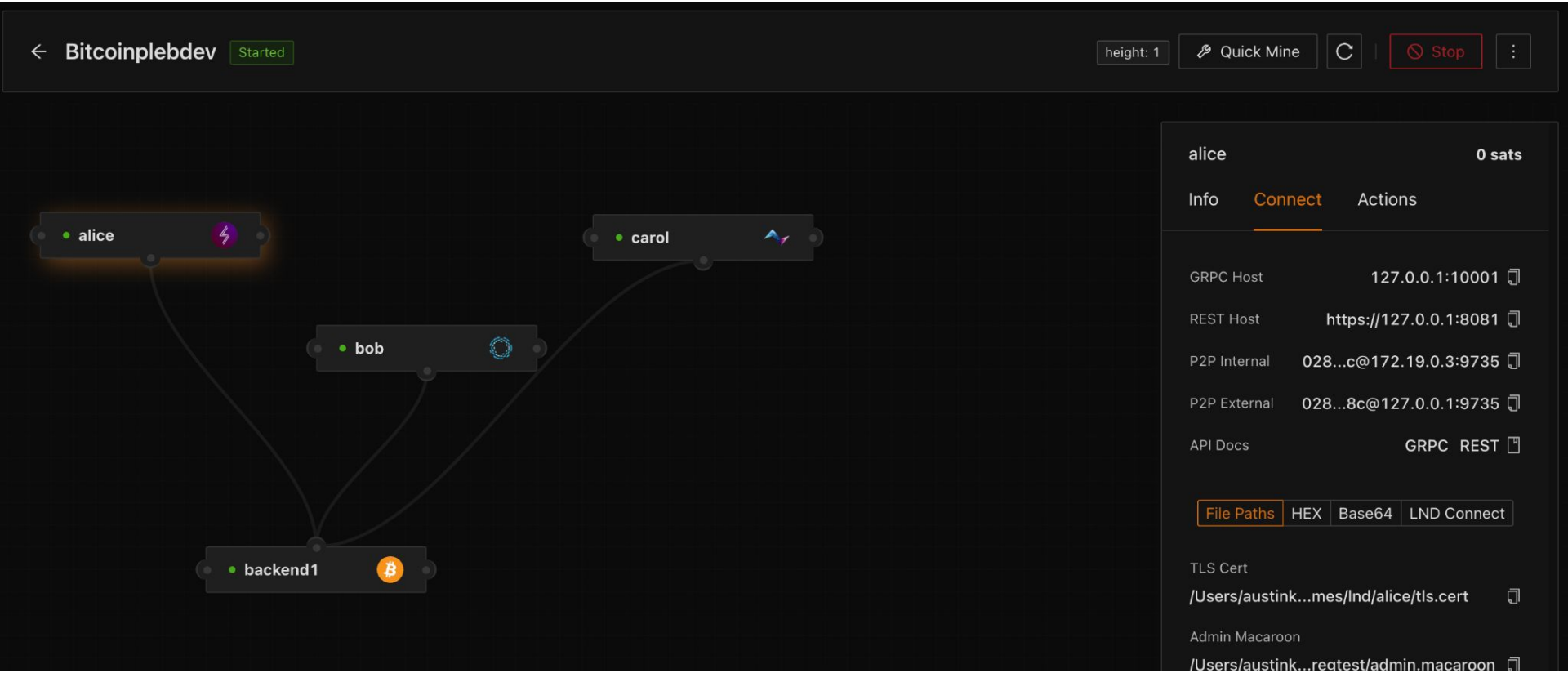
getInfo request in pleb-node with LND-GRPC

```
20  const getInfo = async () => {
21    const info = await lnd.services.Lightning.getInfo();
22
23    return info;
24  };
```

HOST, CERT, and MACAROON

- HOST: The address (IP or domain) of the Lightning node you are connecting to, allowing your app to communicate with the node's API
- CERT (Certificate): A digital certificate used to establish a secure, encrypted connection (TLS) between your app and the Lightning node, ensuring data privacy and integrity
- MACAROON: A unique, encrypted authentication token that grants specific permissions (e.g., read, write, or invoice) to your app when interacting with the Lightning node's API, allowing for fine-grained access control and increased security

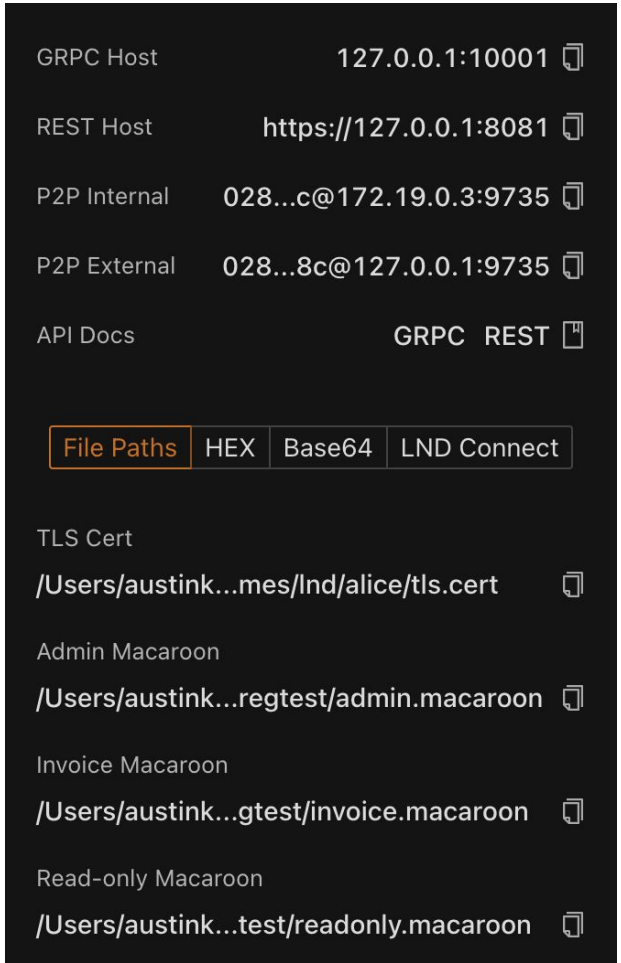
Click on a node, and select the connect tab



Grab your HOST, CERT, and MACAROON.

Grab these 3 variables and add them into your .env:

- GRPC Host
- TLS Cert (File Path)
- Admin Macaroon (File Path)



Add your LND credentials to your .env

```
# secret variable for the port of our server (used in index.js)
PORT=5501
# secret variable for our JWT secret (used in usersRouter.js)
SECRET=keepitsecretkeepitsafu
# secret variable for admin key (used in authenticateAdmin.js)
ADMIN_KEY=1234
# secret variable for LND gRPC host (used in lnd.js)
HOST=127.0.0.1:10001
# secret variable for LND TLS CERT (used in lnd.js)
CERT=/Users/austinkelsay/.polar/networks/1/volumes/lnd/alice/tls.cert
# secret variable for LND admin macaroon (used in lnd.js)
MACAROON=/Users/austinkelsay/.polar/networks/1/volumes/lnd/alice/data/chain/bitcoin/regtest/admin.macaroon
```

Install lnd-grpc

LND-GRPC

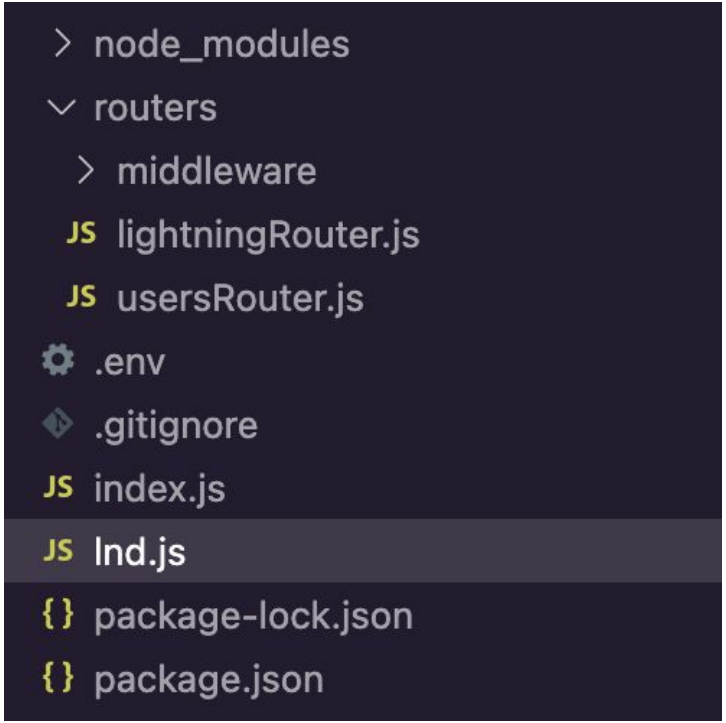
- Run `npm i lnd-grpc`

> Easy to use gRPC wrapper for lnd. ⚡

This package provides and easy to use gRPC wrapper for lnd.

- Supports all lnd versions
- Supports all lnd gRPC sub services
- Automatic async/promise support
- Automatic lnd version detection
- lndconnect support

Create a new file at the root of your project called lnd.js



Lets setup the connection to Alice with lnd-grpc

```
const LndGrpc = require("lnd-grpc");
const dotenv = require("dotenv");

dotenv.config();

const options = {
 host: process.env.HOST,
 cert: process.env.CERT,
 macaroon: process.env.MACAROON,
};

const lnd = new LndGrpc(options);

const connect = async () => {
 try {
   await lnd.connect();

   if (lnd.state !== "active") {
     throw new Error(
       "LND did not reach 'active' state within the expected time"
     );
   }

   console.log(`LND gRPC connection state:${lnd.state}`);
 } catch (e) {
   console.log("error", e);
 }
};

module.exports = { connect };
```

We can now call this connect function when our server starts up in index.js

```
const { connect } = require("./lnd")

...

// Use the built-in JSON middleware to parse incoming JSON requests
server.use(express.json());

// Connect to our LND node
connect()
```

Let's start the server and see if we connect successfully

(it's ok if you're getting the "DeprecationWarning like I am)

```
austinkelsay@Austins-MacBook-Pro backend-course-walkthrough % npm run start

> backend-course-walkthrough@1.0.0 start
> nodemon index.js

[nodemon] 2.0.21
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
Server listening on port 5501
(node:12359) [DEP0123] DeprecationWarning: Setting the TLS ServerName to an IP address is not permitted by RFC 6066. This will be ignored in a future ver
sion.
(Use `node --trace-deprecation ...` to show where the warning was created)
LND gRPC client is ready to use
```

Let's Look at the LND docs

So we can see how to build the required LND methods for pleb-wallet

LND docs

Create balance methods in lnd.js

```
const getBalance = async () => {
 const balance = await lnd.services.Lightning.walletBalance();
 return balance;
};

const getChannelBalance = async () => {
 const channelBalance = await lnd.services.Lightning.channelBalance();
 return channelBalance;
};

module.exports = {
 connect,
 getBalance,
 getChannelBalance,
};
```

Add createInvoice method in lnd.js

```
const createInvoice = async ({ value, memo }) => {
 const invoice = await lnd.services.Lightning.addInvoice({
   value: value,
   memo: memo,
 });

 // Save invoice to DB

 return invoice;
};

module.exports = {
 connect,
 getBalance,
 getChannelBalance,
 createInvoice,
};
```

Add payInvoice method in lnd.js

```
const payInvoice = async ({ payment_request }) => {
 const paidInvoice = await lnd.services.Lightning.sendPaymentSync({
   payment_request: payment_request,
 });

 return paidInvoice;
};

module.exports = {
 connect,
 getBalance,
 getChannelBalance,
 createInvoice,
 payInvoice,
};
```

How do we know when our invoices are paid?

Event streams!

Understanding event streams

- Event streams: A continuous flow of events or data, enabling real-time processing and updating of information
- gRPC streaming: Utilizes gRPC to establish a persistent connection for receiving real-time updates from a server (e.g., Lightning node)
- Subscribing to events: Specify the type of events or data to listen for, such as new invoices, settled invoices, or payment updates
- Callback functions: Define actions to perform when specific events occur, like updating a database or triggering a user notification
- Error handling: Implement error callbacks to manage unexpected issues during event stream processing, ensuring graceful failure and proper logging
- Real-time responsiveness: Event streams allow applications to react immediately to changes in data, providing a seamless and dynamic user experience
- Efficient resource usage: Event-driven architecture reduces the need for constant polling, lowering resource consumption and improving performance

Build an Invoice event stream

```
const invoiceEventStream = async () => {
 await grpc.services.Lightning.subscribeInvoices({
   add_index: 0,
   settle_index: 0,
 })
   .on("data", async (data) => {
     if (data.settled) {
       // Check if the invoice exists in the database
       const existingInvoice = False;

       // If the invoice exists, update it in the database
       if (existingInvoice) {
         // update db
       } else {
         console.log("Invoice not found in the database");
       }
     }
   })
   .on("error", (err) => {
     console.log(err);
   });
};
```

Start the invoice event stream when connecting

```
const connect = async () => {
 try {
   await lnd.connect();

   // Start the invoice event stream on successful connection
   // We want to always be listening for invoice events while the server is running
   invoiceEventStream();

   console.log("LND gRPC client is ready to use");
 } catch (e) {
   console.log("error", e);
 }
};
```

Add the balance endpoint in lightningRouter

```
const {
 getBalance,
 createInvoice,
 getChannelBalance,
 payInvoice,
} = require("../lnd.js");

// GET the onchain balance
router.get("/balance", (req, res) => {
 getBalance()
   .then((balance) => {
     res.status(200).json(balance);
   })
   .catch((err) => {
     res.status(500).json(err);
   });
});
```

Add the channelBalance in lightningRouter

```
// GET the lightning wallet balance
router.get("/channelbalance", (req, res) => {
 getChannelBalance()
   .then((channelBalance) => {
     res.status(200).json(channelBalance);
   })
   .catch((err) => {
     res.status(500).json(err);
   });
});
```

Add POST endpoint for creating an invoice

```
// POST required info to create an invoice
router.post("/invoice", authenticate, (req, res) => {
 const { value, memo } = req.body;

 createInvoice({ value, memo, user_id })
   .then((invoice) => {
     res.status(200).json(invoice);
   })
   .catch((err) => {
     res.status(500).json(err);
   });
});
```

Add POST endpoint for paying an invoice

```
// POST an invoice to pay
router.post("/pay", authenticateAdmin, async (req, res) => {
 const { payment_request } = req.body;

 const pay = await payInvoice({ payment_request });

 if (pay.payment_error) {
   res.status(500).json(pay.payment_error);
 }

 if (pay?.payment_route) {
   // Save to DB

   res.status(200).json(pay);
 }
});
```

Now we can check our newly created endpoints

With Insomnia!

Resources

- Express / React Lightning app template - https://github.com/AustinKelsay/pleb-node-template
- Lightning Labs Build Your First LAPP - https://docs.lightning.engineering/lapps/guides/polar-lapps
- Build Bitcoin into Your App: Getting Started with the Lightning Network - https://www.youtube.com/watch?v=6P0DZ74DmFA
- A crash course in Lightning App Development - https://medium.com/@rheedio/a-crash-course-in-lightning-app-development-5be5b8d2d558
- LND Overview and Developer Guide - https://dev.lightning.community/overview/