BitPay Backend Coding Assignment

You've been provided with a client, sorrychain.js, to a blockchain we've dubbed "Sorrychain" since it's a sorry excuse of a blockchain for its cryptocurrency, SorryCoin (SRC). Your task is to create a simple user client and an invoicing server.

First, you must start Sorrychain and leave it running. A user will need to be able to use your client to request an invoice from your server. The invoice should contain an id, an amount to be paid, an expiration time of something reasonable like 5 minutes, and an SRC address. The user then sends that amount to the address on the invoice via Sorrychain. The Sorrychain will return a transaction id to the client. Your server needs to subscribe to blocks and transactions on the Sorrychain to know when a payment is received and confirmed.

The basic functionality of your client and server are below:

- Client:
    - Get invoice from server
    - Create transaction on Sorrychain
- Server
    - Create an invoice
    - Receive payment info from Sorrychain
    - Mark the invoice as paid

Requirements:

- Use Node.JS
- For data storage you can either keep everything in memory or, if you want to use a database, use MongoDB.
- Do NOT modify sorrychain.js. We will use our own copy when evaluating your code.
    - If you encounter a bug, feel free to change it locally and make a note about it in your Readme
- Upload to a git repository named <first-initial+last-name>-<year> (e.g. lskywalker-2021)
    - Include a Readme file which describes how to use your project.

Tips:

- This should not take you more than a few hours
- Comments explaining **why** you did something are a big plus, but don't comment every line
- Sorrychain is not exhaustive and should be considered as if it were a major chain like Bitcoin. You should consider possibilities of the major chains when implementing. If/when you do so, leave a comment explaining why.
- Using npm libraries is fine, however the fewer the better. This assignment can be done with only native modules.
- Keep it simple, but expect and plan for us to try to break it.

- Adding a bunch of extra functionality won't really give you bonus points, just make sure you hit the requirements as best you can
- Including unit and/or integration tests is optional, but appreciated.
- Missing core functionality in the requirements will likely result in disqualification
- Have fun!

Sorrychain Docs:

Sorrychain is a simple blockchain that produces blocks on a set interval of 5 seconds. State is not maintained between restarts, meaning the blockchain gets wiped clean when you shut it down. Any transactions created will be included in the next block. Sorrychain is completely centralized, has a proof-of-none validation mechanism (meaning there is no block validation), and has zero cryptographic security.

API:

Sorrychain's HTTP API can be reached on port 5000.
Errors will return an object with an 'error' field describing the error.

GET: /block[?hash|height]
    Query: (optional) \`hash\` or \`height\` of the block you want. By default, this will return the latest block if no query parameter is given
    Response: <block: Object>

GET: /transaction?hash=<your tx hash>
    Query: (required) \`hash\` of the transaction you want
    Response: <transaction: Object>

POST: /
    Body: { address: String, amount: Number }
    Response: <transaction hash: String>

TCP:

You can listen for blocks and transactions on Sorrychain's TCP socket on port 4000.
The data will be serialized as a stringified object:

```
{
  type: <"block"|"transaction">,
  data: <hex-encoded block or transaction object>
}
```

Objects:

Block
```
{
    height: Number,
    blockTime: Number,
    prevBlockHash: String,
    transactions: String[]
    hash: String
}
```

Transaction
```
{
    address: String,
    amount: Number,
    timestamp: Number,
    hash: String
}
```