Spike UFC-414: API Proxy Prototype Design

Acceptance Criteria

For reference, the following are the acceptance criteria for the API Proxy:

- CRUD accounts
- CRUD transactions
  - Bulk insert
- API key authentication
- Encrypt/decrypt data

Design Brainstorming

- What if, like Redux, we use the encryption middleware *as* a middleware, but an Express one this time?
  - That way, we can intercept data for encryption/decryption at the route level.
- Should just copy in the encryption middleware code for this separate package until we can have it hosted publicly and shared.
- Should also re-use the Feathers API client from the Frontend.
- Need API key authentication.
  - Like the most basic shit: just read in an API key (UUID v4) from an env var and use that to authenticate requests.
  - This way the proxy can at least be exposed on the internet kinda safely.
- I'm thinking it's easier to just use Feathers as the API rather than Express since Express apparently doesn't have 'after' middlewares? So we wouldn't be able to put a middleware in place after a response is made to decrypt or encrypt the response??
  - But then it'll get confusing whether we're calling Feathers functions for the proxy or for the API...
  - Meh, just brute force it with Express. It's just a prototype after all.
- CRUD:
  - Create: encrypt request

    - Read: decrypt response
    - Update: encrypt request
    - Delete: no encryption/decryption
- Decided to add the transactions as a property on the accounts encryption schema.
    - This way, when we fetch accounts from the Backend, we can just decrypt the transactions alongside them without doing anything extra.
    - Will have to backport this to the Frontend...

In Progress Thoughts

- I'm having some real issues dealing with the crypto worker pool. The `workerize-loader` is throwing things for a loop in Node land.
- Honestly, I'm starting to think that we should just throw it out in favour of just going single-threaded. Frankly, I doubt there's going to be much of a performance hit (will have to benchmark), but it'll make my life so much easier.
- However, there's another potential solution: using the native Webpack 5 functionality for workers Web Workers | webpack.
    - This way, we can get rid of workerize-loader (or any webpack loader syntax) in favour of just instantiating a `Worker` directly. Noteworthily, `Workers` are cross-compatible with Node, so it should just work.
    - The obvious downside is that we lose the nice promise based syntax, but we might be able to reverse engineer it from `workerize-loader`; doesn't seem too complicated.
        - Or maybe not...
- Of course, upgrading `cra` to v5 (to get Webpack 5) is out of scope of this ticket. So... for now, the code will just be copied into the proxy repo.

Todo

- Initialize crypto
- Encryption/Decryption middleware
- CRUD accounts
    - Create accounts
    - Read accounts

    - Update accounts
    - Delete accounts
- CRUD transactions
    - Create transactions
    - Read transactions
    - Update transactions
    - Delete transactions
- Pass email/password as env vars
- API key authentication middleware
- Refreshing auth JWT
    - Can implement this as a middleware?
- Bulk insert transactions