NFT DAO Akkadia Project Audit Report 2021-06-03

Key findings

The Akkadia code was not built to any master plan. That defect has resulted in the existing components not connecting to each other and crucial functionality never being built and cannot be easily fixed. For this reason, we believe the most cost-effective option is to build Akkadia from scratch.

Summary

The Akkadia code base is supposed to be a website that supports the sale of membership tokens for the NFT DAO. It is made of three parts: infrastructure, backend and frontend. The three parts were made independently by three different developers, who have all since left.

Each part has merits of its own, yet the project as a whole has serious issues. Most importantly, the parts fail to connect to each other. Indeed, there is apparently not any master plan, data schema, or specification for how the pieces are supposed to connect. In addition, entire required pieces of functionality are missing, such as data persistence in the backend, and identity management in the frontend.

Because the current components are not integrated with each other and there is no master plan in place, completing the project will require more or less building the project from scratch. We recommend hiring a professional development team, including a project manager and with in-house expertise in UX design, front-end development, backend development, and operations development.

While some of the existing code may be reusable, it will likely be more cost effective to create the master plan and then build the necessary components to the requirements of the plan instead of creating a plan around the old codebase. We also propose a simplified architecture that we believe will reduce the development time of the project.

Overview

Purpose

The goal for the upcoming Akkadia release is to implement the sale of 10,000 NFTs in 10 batches of 1000. The first batch is sold at 10 ADA/ticket, the next at 20 ADA/ticket, etc., until the last batch is sold at 100 ADA/ticket. One cold wallet gathers the ADA and is watched for incoming transactions. One hot wallet owns the NFTs and sends them to whoever sent enough ADA to the cold wallet. Each ticket will be associated with a specific picture. The site is supposed to launch in June 2021.

Code Repository

Original: https://github.com/NFT-DAO/Akkadia
MuKn internal Fork: https://github.com/MuKnIO/Akkadia

Lack of Documentation

No significant documentation to complement or support the code was found. We did not find, nor were we made aware of:

- any formal or informal specification document for what the software is supposed to do;
- any formal or informal specification document describing how the software is architected, how it is supposed to work, or why specific architectural choices were made;
- any formal or informal specification for any of the individual components of the software.

Architecture

/ ~6700 of combined .js and .ts loc.

front_end/ Frontend. 4.4 kloc.
forkable/ Backend. 0.77 kloc.
cdk/ AWS infrastructure. 1.5 kloc.

front_end/
The front_end directory is a static website. It was written by Taylor Yoon aka yoont4. It doesn't look like it connects to the backend via any endpoint. It uses placeholder assets for data meant to be retrieved from the backend.

forkable/
The forkable directory is the backend service, to be packaged into a lambda function. It was written by Quinn Parkinson aka logicalmechanism. This code is a decent first pass at fleshing out the functionality, but it is overall low quality. More importantly, the code crucially omits essential parts of the functionality, such as persisting data to the database, or having a mechanism to share configuration with the infrastructure or frontend.

cdk/
The cdk/ directory is the AWS infrastructure. It was written by Dmitri Safine. It looks like a clean job, though not documented with an audit in mind. The infrastructure follows a design that is broadly sensible, but may or may not be optimal in light of the current state of the project or the intended state for initial deployment. It also needs to be updated with the latest version of the source repository..

ASSESSMENT:

front_end/

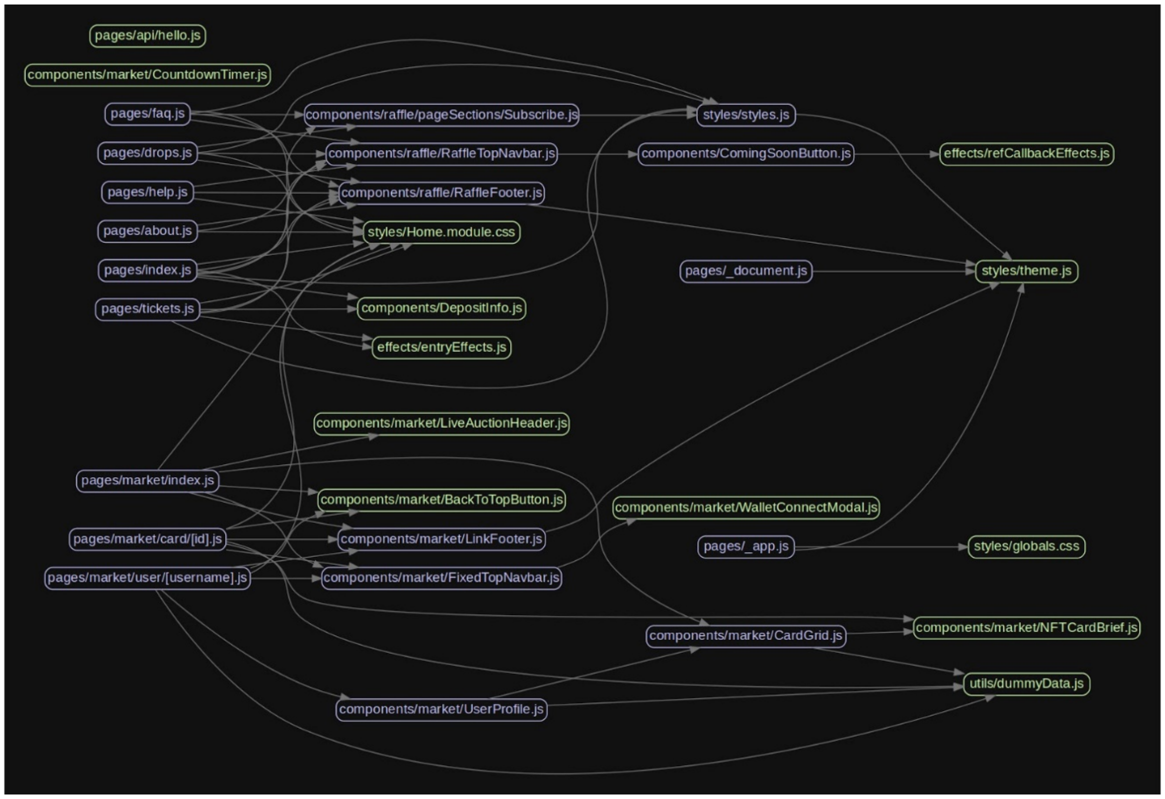
Description

The frontend for the Akkadia NFT sale. Contains landing page, purchasing NFTs, token galleries and other front-facing components.

Remarks

Contains static content with placeholder values for backend data. Purchases cannot currently be made from the site. Missing backend calls are marked with TODOs by the original developer. **Most / all backend calls are not implemented, but are currently marked with placeholders.** Eventually Infra should support AWS lambdas which serve entry points of the backend. In turn, the frontend should call these AWS lambdas for required functionality.

Frontend Dependency Graph


pages/api/hello.js
components/market/CountdownTimer.js
pages/faq.js
pages/drops.js
pages/help.js
pages/about.js
pages/index.js
pages/tickets.js
components/raffle/pageSections/Subscribe.js
components/raffle/RaffleTopNavbar.js
components/raffle/RaffleFooter.js
styles/Home.module.css
components/DepositInfo.js
effects/entryEffects.js
styles/styles.js
components/ComingSoonButton.js
effects/refCallbackEffects.js
pages/_document.js
styles/theme.js
components/market/LiveAuctionHeader.js
pages/market/index.js
pages/market/card/[id].js
pages/market/user/[username].js
components/market/BackToTopButton.js
components/market/LinkFooter.js
components/market/FixedTopNavbar.js
components/market/WalletConnectModal.js
pages/_app.js
styles/globals.css
components/market/CardGrid.js
components/market/NFTCardBrief.js
utils/dummyData.js
components/market/UserProfile.js

Backend functionality required by Frontend

The following functionalities are currently proxied using placeholders.

Purchasing tickets

Retrieves depositAmount, depositAddress from backend.
pages/tickets (BuyTicketSection)
pages/index (TicketPage)
Note: Purchasing will happen directly via transactions to the Sender's wallet. Thus, the frontend does not query the backend with a purchase request. Instead, the backend listens to the transactions on the Sender's address, and sends out NFTs to Buyers with valid transactions.

Displaying Card Samples

Extracts Cards metadata (id, title, author, etc…) from backend.
components/market/CardGrid (CardGrid)
components/market/NFTCardBrief (NFTCardGridPreview)

Displaying Card Author info

Extracts Author info from backend.
components/market/NFTCardBrief (NFTCardBrief) -> pages/market/user/[username]
pages/market/user/[username] (UserProfilePage)

Drops

Extracts Card drop from backend. Currently this is hardcoded and not marked with TODOs.
pages/drops (UpcomingDropsSection)

Login

Not implemented.

forkable/

Description

The backend of the Akkadia NFT sale. Consists of a collection of script files implementing core logic to handle transacting (**auto_rec_send.js)** and minting Akkadia NFTs (**minting.js)**.

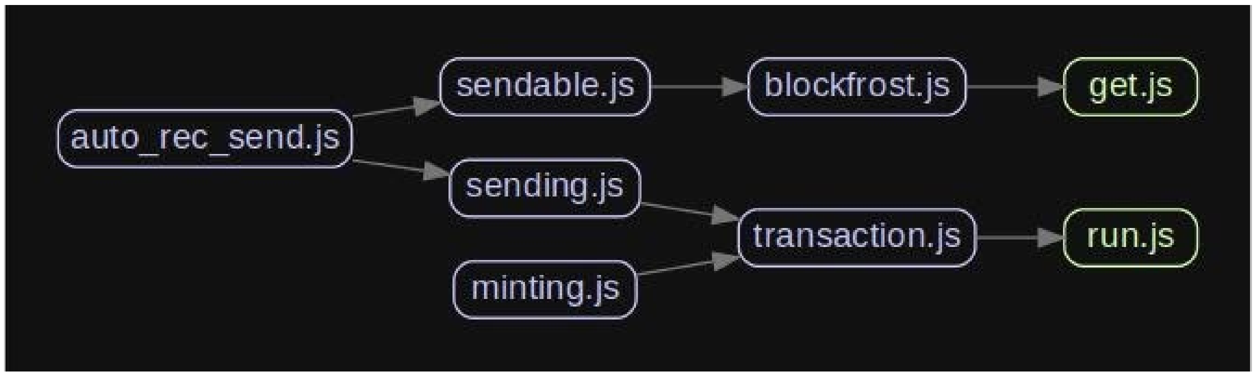
Remarks

NFT transaction is not fully implemented, because Database functionality is unimplemented.
NFT Minting is complete but not polished.

Minting uses cardano-cli which -- apparently through a standard rpc endpoint-- sends transactions to the testnet, instead of using the cardano node stood up by the AWS infrastructure or otherwise configured. It is unclear how this would work in production on the mainnet; we speculate that an operator would reproduce those steps on his machine, while connecting to the actual mainnet.

Watching buyer transactions happens by querying blockfrost.io, a different mechanism than the minting mechanism described above. There are valid reasons why in a highly adversarial environment one might want to use different mechanisms for listening to the blockchain and posting to it; but at the scale of this project, this is only an opportunity for more breakage and higher maintenance.

Backend dependency graph



Backend code review

Inline audit: https://github.com/NFT-DAO/Akkadia/pull/20/files

There is currently no documentation on setting up backend dependencies and running the script files.

High level functionality is roughly implemented. However, significant work remains to make it production ready.

General issues with the backend code:

- There was little or no error handling.
- Validation logic was convoluted or missing.
- Logging is not implemented for production and development.
- No test cases or instructions to verify that complex and/or critical functionalities work.

Backend components review

Transacting NFTs is done via **auto_rec_send**.
Minting NFTs is done via **minting**.

Module	Functionality	Issues
auto_rec_send NFT transaction to Buyer	Gets transaction history via blockfrost.io API using the sender's public address (**sendable.find_all_trx(addr)**). The transaction history contains a batch of Buyers' addresses. For each batch, it sequentially sends NFTs to all Buyers addresses (**sendable.helper(res, addr, [])**). This job runs every 60s.	**autoReceiveThenAddToDB** Functionality for adding the data into the DB is not ready.
sendable	Utilities for finding transaction history of Sender's public address, sending NFTs to Buyers. **find_all_trx** Finds and returns all transactions of a public address by querying Blockfrost API. **helper** Sends NFTs to each Buyer address sequentially using **build_sendable**. **build_sendable** Obtains inputs and outputs of a transaction. Assumes the address of the first input in inputs is the buyer's address. It also checks outputs, to see if the price was paid in full using lovelace **within one output**. If this holds, it uses **confirm_single** to do further validation and then to dispatch the NFT. **confirm_single** If it passes the validations, dispatch the NFT to the Buyer's address, in the "to be sent" database. This is yet to be implemented.	**find_all_trx** Extracts without specifying page, count, order parameters. Can lead to some addresses never receiving their NFTs, if their transactions are not within the first 100 returned by the Blockfrost endpoint. **helper** Address not properly validated. Only checks if it is a non-empty string. **build_sendable** Validation takes place between here and **confirm_single.** These should be moved to **confirm_single.** **confirm_single** Validation logic is convoluted. See inline code comments.
blockfrost	Utilities for the blockfrost.io API.	**getAddressHistory** Uses deprecated blockfrost API.
get	Web utils	-

Module	**Functionality**	**Issues**
minting Minting of NFTs	**mint** Uses **cardano-cli** to interface with the Mainnet. It does the following: 1. Get Protocols 2. Get Sender UTXO 3. Get balance, tx-in-count, tx-in 4. Get chain tip 5. Build draft transaction 6. Estimate fee 7. Get new txout 8. Build raw transaction 9. Sign transaction 10. Submit transactions	Validation logic is convoluted. Parsing logic is convoluted. Test cases should be implemented for complex sections.
transaction	Utility functions which provide transaction functionality via **cardano-cli**.	Currently this is not wired up to the NFT DAO node, only a testnet.
run	Access a shell process via node.	-

Glossary

Sender: Buyers send ADA to this entity's public address. This entity then sends NFTs to the transacting addresses.
Buyer: Those who are purchasing the NFT.

cdk/

Description

The AWS infrastructure for Akkadia NFT Sale. Uses AWS CDK to define cloud application resources with Typescript code that can stand a network.

Remarks

Overall the configuration is optimized for a moderate to heavy continuous load. This appears to greatly exceed current needs. This could be a mostly static website, with low volume load for NFT issuance only: 10000 queries max total, perhaps a small multiple with timer-based retries.

There does not appear to be any document explaining the current AWS infrastructure, how it does or doesn't match the rest of the code as it is currently (or as it is intended to be), and/or why specific decisions were made.

It is unclear where the backend code currently in forkable/ is supposed to run: as part of the "frontend" pipelines, or of the cardano node, or of a "stack" of its own that is yet to be written.

Components

The main executable is bin/nft-mvp.ts that relies on the following files in lib/:

- infrastructure-pipeline-stack.ts seems to configure the entire AWS infrastructure.
- cardano-binaries-pipeline-stack.ts and cardano-node-bootstrap.ts are two different but overlapping build pipelines. It might be possible to merge them or eliminate one (or both?).
- cardano-node-stack.ts stands a cardano node. It isn't currently used by the rest of the code. More importantly, it might be costly to maintain in the long run, since *it will require regular updates*. Also, the current code doesn't deal with smooth upgrade, setup time needed to initially synch with the blockchain, or preservation of blockchain data across upgrades. This all adds complexity that might not be necessary if the system is going to rely on the external service blockfrost.io for watching the blockchain, anyway.
- core-stack.ts stands a small bastion host. May not be needed if the infrastructure is simplified away.
- rds-stack.ts starts a postgres database, currently unused, but that would presumably be used by the backend, and possibly by the front-end. Alternatively, the limited use of a database could be wholly replaced by simple append-only access to s3 buckets.
- frontend-pipeline-stack.ts starts some lambda service for the front-end. This would be most appropriate for a database-driven website, but the current website is wholly static, and might be better served by a s3-backed static website, with the few dynamic requests sent to the backend. Considering that the identities that matter are cryptographic, the unimplemented login functionality can possibly be wholly done away with. There remains, though, the issue of whether and how to customize the site depending on the cryptographic identities of the user.
- There isn't currently a defined service for the backend, but it might actually look very much like the current "frontend pipeline stack" after the front-end was made a static website, and/or it could be merged with the cardano node if the cost of using and maintaining a node is deemed necessary. One problem with making it a lambda service will be making sure that the use of the cardano cli fits within the space and time requirements of a lambda service.

The infrastructure does not seem to match the code. It is also out-of-date in that it seems to assume the various parts of the code are in different branches. However, it looks like the code is now all in the same branch, in different sub-directories, and the infrastructure should be updated accordingly.

Building the Missing Pieces

Blueprints

First and foremost, the project needs blueprints:

- A clear specification of what needs to be done
    - what are the resource and security constraints for the system as a whole,
    - What are the resource and security constraints for each component of the system.
- For the UI, wireframes fully describing each step of each possible interaction flow between the users and the website. A professional UX designer will likely be needed for that.
- The data schema for any persistent data, including any metadata to be associated with NFTs, needs to be defined.
- Responsibilities need to be clearly distributed regarding
    - who is going to build what (code, graphics, infrastructure, etc.),
    - who will make sure the pieces fit together,
    - who will watch over the moving pieces (e.g. cardano node if we stand up one, blockfrost.io connection if we use their service)—and
    - who will be rewarded which way for their trouble.
- A clear understanding of what graphics should be used for the website in general, and to associate with the NFTs in particular.
    - It appears that what is in the repository would mostly not be usable with the author's consent anymore, and that an artist might need to be paid for new designs.

There is no point in resuming any coding until the above have been resolved.

Additional Issue: Centralized and/or Decentralized Identity

The current frontend alludes to a wholly unimplemented login mechanism. It is not obvious such a mechanism would be useful or necessary, but it could significantly affect the structure of the website.

First, there is therefore no *need* for centralized identity managed by the website because payment and ownership of the NFTs are completely decentralized, managed on the Cardano blockchain, which is authoritative.

Such identity *could* be useful, but only if besides selling the NFT's there are other services offered whereby the users could interact with each other, and the website would somehow provide a centralized way to anchor decentralized identities. Even then, in the future, we can imagine decentralized alternatives to this centralized service. Thus, implementing such a login system seems unnecessary in either the short term or the long run.

While it is true that the people who purchase NFTs will want some way when browsing the website to distinguish between those NFTs they possess and those they do not, that will require integration with the Daedalus or Yoroi wallet rather than a centralized login system. Integrating with a wallet is not trivial, and is on its own probably a multiple man-month project, independent from the work required to get the NFT sale working as such.

Whether identities are going to be managed centrally and/or decentrally, a lot of work is required to get it right. Happily, that work is not necessary to launch the NFT sale —but the result will be much less appealing if decentralized identities are not at least minimally supported.

Proposed Simplification

If there is no login and no login-driven behavior, the entire website could be a static website serving a Javascript interface that runs purely on the client side, querying s3-backed static data as needed.

The backend would update said s3-backed static data in append-only mode when NFTs are minted—if needed at all. No database would be required beside s3 buckets.

The frontend could be a wholly static website, with lambda functions just for invoking the backend. If blockfrost.io is used, no cardano node is required. With no permanent server running, no bastion host is required, either.

With such simplifications, it might be possible to complete the website, its test and deployment by the end of June 2021.

Conclusion

The failure to build Akkadia to any master plan is a critical flaw in the code. It resulted in significant wasted effort in the form of components that do not connect to each other, crucial functionality never being built, and left no easy way to integrate the code components that were built. For these reasons, we believe the most cost-effective option for NFT DAO is to design and build Akkadia from scratch.