The traditional models of software architectures don't apply to decentralized dapps. We need a new paradigm. Here, I outline a new architecture for DApps that are completely decentralized. It's called SACI, pronounced Sassy (because it is).

The 4 components in this architecture are

- Decentralized S

torage

- Decentralized A

pp

- Smart C

ontract

- Decentralized I

ndexer

So - Storage, App, Contract, Indexer = SACI

Storage

In the architecture, the data is not stored on a centralized database. The data is stored as Blobs on a decentralized storage service. These blobs are then referenced by their unique identifier (e.g. MD5Sum, IPFS Hash)

This storage stores

- The App itself, which is served for execution on the client side (more on this in the next section)
- User generated data - both encrypted and un-encrypted depending on the data type

Encrypted Data storage

The data is stored by encrypting it with the public keys of the users who are allowed to decrypt the data. This data can be retrieved by an application by the reference identifier and then decrypted on the client side using the private keys.

Services

IPFS, FileCoin, Arweave

App

The app is one that runs completely on client side. E.g. React apps without SSR, mobile applications (APK/IPA), single page websites.

What is not an application is one that is served by a server where there is some computation and authentication that happens on the backend. For example, in the MVC model, the controller has access to modifying the views on the backend. In SACI all the business logic is executed strictly on the client side / frontend.

The app is usually responsible for

UI/UX

All the business logic and user interfaces are displayed by the application. If this is a web application, it should be noted that this comes with its own security constraints. Code that executes on a browser is insecure. That brings us to the next responsibility of the App.

Wallet integration

All the user interactions necessarily happens on chain (ideally). So it requires the user to sign every transaction that happens. Even for transactions that aren't necessarily posted on to the main chain to save gas, they must be signed and stored on an L2 or rollup.

Every user interaction is valid only if it has a signature to it.

API integration

The business logic requires data. This data can be fetched via RPCs for on chain data or APIs exposed by the indexer.

Available services

React, Flutter (without SSR)

Contract

As discussed in the section above, all the user interactions ideally are recorded on-chain. If the user is creating a post, it's on chain. If the user is creating a profile, it's on chain …

The way to save data is to create a transaction on the chain. This contract should emit an event when the transaction is completed.

The best way to save the data is to create a blob of data, put it on a decentralized storage service and save the reference to that blob. When the contract emits the reference to the blob, the event is listened to by the indexer.

The data can be retrieved by a combination of querying the contract/indexer and the decentralized storage service.

The best way to store data is to emit events. This event can then be indexed by the indexer and made available for later use. This is better than saving the data completely on chain, to save gas fees.

Services available

Ethereum, Optimism, Polygon, StarkNet and all other programmable L1/L2/Rollups

Indexer

The indexer is responsible for capturing all the data that is emitted by the contract. The indexer is the closest to a controller on the MVC model. The indexer is responsible for capturing the data from the contract, processing it and saving it in a way that can be queried later. Ideally the indexer is also a decentralized system that allows for the system to not have a single point of failure.

Verification

This is an optional step to optimize resource consumption.

The indexer will store information only when the data emitted by the contract is in the correct schema. The indexer will query the verification service to check the sanctity of the data before storing it.

The verification service allows the data to be consistent and clean.

Service available

Graph, Covalent

Conclusion

This architecture is optimized for complete decentralization. The point here being that the service provided by your dapp has the following properties.

Composable & Permissionless

It requires all the data to be available for any service to consume. It is not required that the product used only via the application. The product can be used by other services via a different UI, or other contracts that bypass the app completely.

Trustless

All the components are open sourced. The contract, indexer logic & the app (if it is a web app) are by definition open sourced. No one needs to trust this dapp - they can see all the code and be sure about the business logic.

Secure

All the data that is generated by the user requires a wallet signature and/or encryption. Meaning that the data is always in the control of the end user and cannot be censored, manipulated or used without consent.