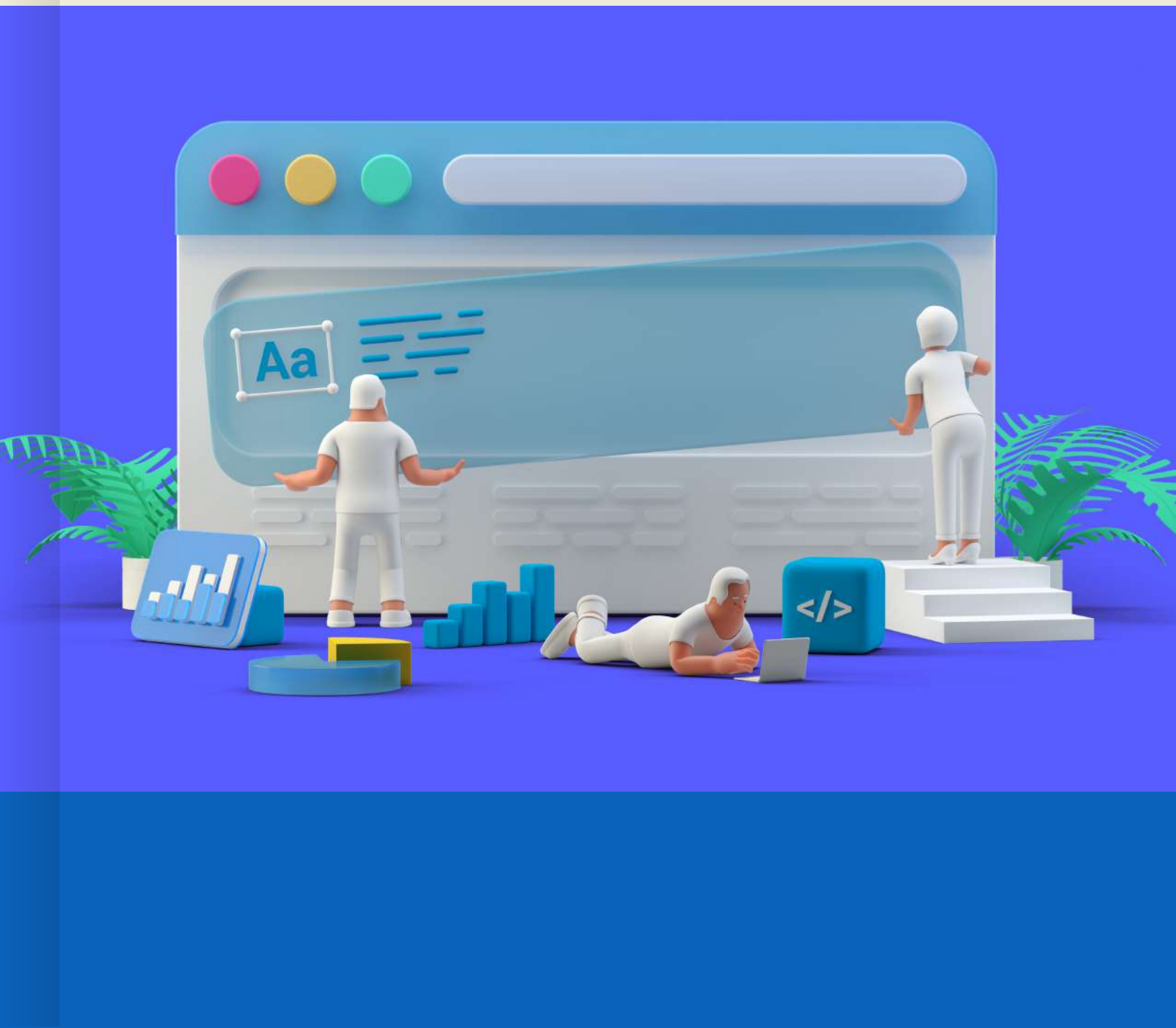


The State of Web3 User and Developer Experience



CryptoEQ™ stands at the forefront of the cryptocurrency analysis and ratings sector, offering unparalleled insights and expertise. Our mission is to demystify the complexities of the crypto market, providing investors and traders with the most trusted, objective, and detailed analysis available. Through our proprietary algorithms, exhaustive research, and a vibrant community, we empower our users to navigate their investment journey with confidence and precision.

We're excited to present this in-depth research report on the State of Web3 User and Developer Experience. Combining our deep analysis skills with insights from Radix contributors, this research pushes the boundaries of blockchain knowledge. It aims to be both informative and enjoyable, and breaks down the complex stuff so you can join in the blockchain adventure with ease and confidence.

01

**User Experience
(UX) and Developer
Experience (DX)**

Introduction

The blockchain economy has relentlessly expanded over the past several years, rising from a single smart contract platform in Ethereum to a robust industry of **hundreds** of blockchains, ecosystems, and applications. One of the key factors determining which platforms will be adopted, or those that fall to the wayside, are the User Experience (UX) and Developer Experience (DX) they offer, as that determines how easy, intuitive, and safe it is to use and build blockchain dApps.

Why is User Experience and Developer Experience Important?



As blockchain ventures from its niche origins into mainstream consciousness, the imperative of a seamless UX has never been greater. Potential users - especially those that haven't yet used Web3 - expect intuitiveness and security akin to the conventional Web2 digital experience. But over the last few years, blockchain has been anything but intuitive and secure.

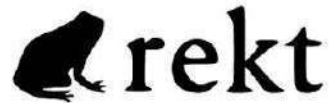
Users have to contend with complex technologies, intricate cryptographic procedures, and confusing and opaque transactions. The challenge for blockchains today is abstracting away these complexities without compromising on the core principles of decentralization, security, and transparency. For platforms like Ethereum, Solana, Aptos, Radix and others, achieving a UX that is intuitive and safe enough for mainstream adoption is not just desirable - it's indispensable.

Yet, an intuitive UX is only half the equation. The backbone of any successful blockchain platform is its community of developers, the architects who build, innovate, and expand its ecosystem of decentralized applications (dApps). The programming language, tools, resources, and support given to developers is paramount to allowing these developers to build dApps quickly, allow new developers to come into the space, offer dApps that resonate with end-users, and do so in a way that is secure.

With millions of dollars lost to hacks in crypto and DeFi still occurring every week, much needs to be done to give developers the right tools to build securely. Web3 and DeFi will find it hard to appeal to both mainstream retail users, and institutional adopters, with the security practices of today.

The symbiotic relationship between a thriving developer community and a growing user base cannot be overstated. Platforms prioritizing UX and DX thus position themselves to initiate a flywheel of user and developer growth as they offer the best possible experiences for both demographics.

Top 10 crypto and DeFi hacks: [Source](#)



-
1. **Ronin Network - REKT** *Unaudited*
\$624,000,000 | 03/23/2022
 2. **Poly Network - REKT** *Unaudited*
\$611,000,000 | 08/10/2021
 3. **BNB Bridge - REKT** *Unaudited*
\$586,000,000 | 10/06/2022
 4. **SBF - MASK OFF** *N/A*
\$477,000,000 | 11/12/22
 5. **Wormhole - REKT** *Neodyme*
\$326,000,000 | 02/02/2022
 6. **Mixin Network - REKT** *N/A*
\$200,000,000 | 09/23/2023
 7. **Euler Finance - REKT** *Sherlock*
\$197,000,000 | 03/13/2023
 8. **BitMart - REKT** *N/A*
\$196,000,000 | 12/04/2021
 9. **Nomad Bridge - REKT** *N/A*
\$190,000,000 | 08/01/2022
 10. **Beanstalk - REKT** *Unaudited*
\$181,000,000 | 04/17/2022

Scope

This report provides a holistic analysis of the UX and DX offered by a selection of leading Web3 platforms chosen to represent the offerings across the market.

These are:

- Ethereum, the most popular smart contract platform today, and whose Ethereum Virtual Machine (EVM) is the most widely used programming environment used not just by Ethereum, but by many other layer 1 and layer 2 smart contract platforms;
- Solana, a high throughput chain with its own Sealevel Virtual Machine (SVM), considered by many to be Ethereum's most prominent challenger;
- Aptos, with the Move programming language and MoveVM, offers a new object-based architecture that promises to improve developer productivity and dApp security, and introduces multiple user experience innovations as a result.
- Radix, with its Scrypto programming language and Radix Engine Virtual Machine, brings a new "asset-oriented" architecture that also promises to improve developer productivity and provide a more intuitive and secure user experience.

To compare the UX and DX across these platforms, this report focuses on how tokens, accounts, smart contracts, and transactions work within the respective virtual machines, as this is what ultimately determines the UX and DX that the platforms can offer.

For DX, this report focuses on the programming languages used by developers to build dApps, examining how these have been architected to support how quickly, intuitively, and securely developers can build dApps.

For UX, this report focuses on the capabilities of the mobile and desktop wallets available for these platforms, how the underlying platforms support those capabilities, and whether this results in an intuitive, easy, and secure experience.

Last, we also compare the tooling and resources available to each of the respective developer ecosystems to help them learn and build Web3 and DeFi dApps.

Executive Summary

Ethereum, long recognized as a leader in the blockchain world, has crafted a diverse and mature ecosystem that has become the industry benchmark.

Despite its wide adoption, Ethereum faces challenges in user-friendliness and security, with users required to navigate a lot of the complexity of using blockchain directly, such as blind signed transactions, reliance on seed phrases, and potential for wallet drains. This has resulted in millions of dollars of user funds being lost, and many of these issues cannot be fixed easily due to Ethereum's architecture.

From a development perspective, Ethereum offers a mature platform for dApp development and developer tooling, but as first mover, there are complexities in Solidity and the EVM that results in a steep learning curve to building secure dApps, with the need to perform significant validation and audit.

Overall, Ethereum and the EVM has a significant lead in adoption, with by far the most users, liquidity, and dApps. Its momentum will carry it forward for many years to come and we can expect more and more third party tools such as Truffle and Hardhat to help cover shortfalls in the underlying architecture.

Solana

Solana, with the SVM virtual machine, stands out with its high-speed and low-cost transactions, non-EVM design, and dedication to the monolithic/integrated design structure, improving composability.

Solana's approach to user experience, characterized by low fees and a unified Token Program, simplifies transactions. But due to its account-based architecture, blind signed transactions and wallet drains are still all too common, meaning that users must be wary of using the wrong account, interacting with the wrong smart contract, impacting trust.

On the developer side, Solana has a fast-growing developer community, and Rust is much loved. Developers are however required to deal with some low-level intricacies associated with Solana's parallelized execution model.

Solana is a formidable challenger and continues to gain the most adoption outside the EVM, with significant user and liquidity growth over 2023.

Aptos

Aptos shares much of the same high-level objectives and blockchain design concepts as Solana but significantly differs in its implementation thanks in large part to the Move programming language and MoveVM, which has an object-based model to assets, as opposed to the account-based model for Ethereum and Solana.

Aptos offers an innovative token standard, emphasizing ease of use and flexibility, and offers an enhanced user experience by making tokens more predictable, allowing wallets to recognize the tokens in a user's account (which Solana and Ethereum are not able to do), and provides enhanced security through rotatable signing factors.

On the developer side, Aptos' Move programming language offers an intuitive experience with its object model, solving issues caused by the account model used by Ethereum and Solana. Aptos' Move Prover tool further supports the integrity of smart contracts, bolstering security.

Aptos is an up and coming ecosystem, offering an improved UX and DX but has a smaller ecosystem than Ethereum or Solana.

Radix

Radix has taken what it calls a “full stack” approach to designing its layer 1, building its own consensus algorithm, execution environment, programming language, and wallet, with the aim of offering an improved UX and DX. Many features that are tacked-on on other platforms via third party tooling are enshrined as native features of Radix, improving trust and reliability. Similar but different to Aptos, Radix uses an object-centric design that it styles as an “asset-oriented” programming environment.

This results in a more transparent and predictable user experience as the platform natively understands and governs tokens and transactions. Radix’s architecture also solves other UX issues such as wallet drains, blind signed transactions, or seed phrases. On the developer side, many commonly used features, such as tokens, NFTs, how transactions are accounted, and how permissions are built into smart contracts, are provided as native first class features of Radix’s Scrypto programming language and Radix Engine execution environment, improving developer productivity and security as these features do not need to be built by developers, but instead are customized by them, reducing the chances for error, while still providing the flexibility to build any dApp they desire.

This positions Radix strongly for increased user and developer adoption, but, as Radix’s smart contract capabilities only just recently launched in Q3 2023, the ecosystem is behind others so far in adoption.

Conclusion

While Ethereum is the most widely adopted smart contract blockchain today, the limitations of the Ethereum Virtual Machine (EVM) hold back its ability to offer mainstream-ready user and developer experiences. This has burdened users with insecure security practices, resulting in frequent hacks and exploits. Numerous third-party dApps and wallets look to help improve the user's experience and safety, but ultimately, there is only so much that can be done due to the inherent constraints of the EVM design.

Solana, a “next generation” smart contract blockchain, offers user experience (UX) and developer experience (DX) innovations with its Phantom wallet, Rust programming language, and low transaction fees. Similarly, Aptos, with MoveVM, introduces several further innovations to make tokens more predictable and development more intuitive. Radix, with its recent mainnet upgrade, takes it all one step further with assets as a native first-class feature of the programming environment, improving not only transparency and confidence for users but also developer productivity and security, setting a new benchmark for Web3 UX and DX. By solving the most comprehensive UX and DX challenges, Radix sets itself up as a clear leader for user and developer experience. While Radix's ecosystem has grown quickly since the launch of its mainnet upgrade, it requires significant user, developer, and liquidity growth before it can catch up to Ethereum and Solana.

A high-level breakdown of how each of the Layer 1s in scope approach a variety of UX and DX design challenges has been summarized in the table below.

Executive Summary - Comparison of Platforms' User Experience and Developer Experience

Tokens, NFTs, and Accounts				
	Ethereum	Solana	Aptos	Radix
Summary	<p>A token-holding account is a line item in a smart contract.</p> <p>The behaviors of the token are governed by the smart contract developer.</p> <p>As a result:</p>	<p>A token-holding account is a line item in a smart contract created by a developer e.g. SPL.</p> <p>As a result:</p>	<p>A token-holding account is a line item in a smart contract object created by a developer.</p> <p>As a result:</p>	<p>A token-holding account is a container that holds tokens inside it, as if it were a physical object.</p> <p>The behaviors of the token are governed by the virtual machine.</p> <p>As a result:</p>
Token Behavior	The behavior of each token is unpredictable.		Tokens follow less unpredictable behaviors.	Tokens follow predictable behaviors.
Malicious Tokens	Tokens with hidden malicious behavior that can drain a user's account are commonplace, and users have to know to avoid interacting with these tokens.		Aptos Fungible Asset Standard and pre-signature transparency removes tokens as smart contract code, making malicious tokens less prevalent.	Malicious tokens are impossible.
Understanding a Token	A user would have to read the underlying smart contract code to know what behaviors a token is capable of, and if it was secure.		Varies depending on wallet.	The wallet instantly displays all the possible behaviors of a token.
Import Token	Wallets do not know what tokens a user holds. Users have to manually "import tokens" into their wallet.		Wallets instantly know what tokens a user holds. There is no need to "import tokens".	Wallets instantly know what tokens a user holds. There is no need to "import tokens".
Custody	Users never actually custody a token in their account, as their account is a line item in a developer's smart contract (that the developer may control).		Users custody the token/object directly.	Users custody tokens inside their account, as if it was a physical object in a container.
Seed Phrases	Access to an account is controlled by a single seed phrase.		Consists of key pairs but can be rotated and recovered unlike traditional BTC or ETH wallets.	Access to an account is controlled by an arbitrary combination of signing factors that can be rotated. (While live on the ledger, not yet enabled in wallet)
Spend Approvals	Users are required to provide approval to smart contracts to spend their tokens in other smart contracts.		Spend approvals are not needed, as tokens are represented as objects.	Spend approvals are not needed, as tokens are represented as objects.

Transactions				
	Ethereum	Solana	Aptos	Radix
Summary	<p>A transaction is a signed hash requesting a single smart contract to execute an instruction.</p> <p>As a result:</p>	<p>On-chain state is organized into resources and modules. These are then stored within the individual accounts. Transactions are state changes to these elements.</p> <p>As a result:</p>	<p>On-chain state is organized into resources and modules. These are then stored within the individual accounts. Transactions are state changes to these elements.</p> <p>As a result:</p>	<p>A transaction is a signed manifest of human-readable instructions defining asset movements and method calls between smart contracts/accounts.</p> <p>As a result:</p>
Blind Signing	<p>Transactions are “blind signed” without a user knowing what a transaction will do.</p>	<p>Transactions are blind signed although there are some additional protective measures in place over and above Ethereum and Solana.</p>	<p>Transactions are blind signed although there are some additional protective measures in place over and above Ethereum and Solana.</p>	<p>Transactions are expressed in human-readable language.</p>
Contract Calls	<p>A transaction can only call one smart contract at once. Complex transactions require a custom smart contract to be deployed to orchestrate a series of downstream calls for the transaction.</p>	<p>A single transaction can call a single module or can be a complex transaction called by multiple modules.</p>	<p>A single transaction can call a single module or can be a complex transaction called by multiple modules.</p>	<p>A transaction can call multiple smart contracts all at once. No custom smart contracts need to be deployed to execute complex transactions.</p>
Guarantees	<p>Users can't set network-guaranteed conditions that a transaction must achieve.</p>	<p>Users can set protective measures to ensure only certain outcomes are possible.</p>	<p>Users can set protective measures to ensure only certain outcomes are possible.</p>	<p>Users can set network-guaranteed transaction outcomes, e.g. this swap must return 100 tokens, otherwise the transaction is aborted.</p>
Delegated Transaction Fees	<p>Transaction fees must be paid by the account signing the transaction</p>	<p>Transaction fees can be paid by other accounts.</p>	<p>Transaction fees can be paid by other accounts.</p>	<p>Any account can pay a transaction fee.</p>

Wallets & Login				
	Ethereum	Solana	Aptos	Radix
Wallets and UI	<p>A multitude of mobile/desktop wallets exist. Users can add their accounts to a combination of browser-based or mobile wallets.</p>	<p>A multitude of mobile/desktop wallets exist. Users can add their accounts to a combination of browser-based or mobile wallets.</p>	<p>A handful of wallet options, including third-party development.</p>	<p>A handful of wallet options, including the mobile-first Radix Wallet that can connect to desktop when needed.</p>
Identity and Login	<p>Users log in to Web3 with an account that can hold tokens.</p> <p>Secured by a seed phrase.</p>	<p>Users log in to Web3 with an account that can hold tokens.</p> <p>Secured by a seed phrase.</p>	<p>Users log in to Web3 with an account that can hold tokens.</p> <p>Secured by a seed phrase.</p>	<p>Users log in to Web3 with a dedicated smart contract that represents their persona.</p> <p>Secured by multifactor authentication and recovery. (While live on the ledger, not yet enabled in wallet)</p>

Executive Summary - Comparison of Platforms' DX

Developer Experience	Ethereum	Solana	Aptos	Radix
Language Adoption & Usability				
How are tokens created?	Copy + paste of ERC20 (or equivalent code)	Copy + paste of SPL token standard (or equivalent code)	Copy + paste of the Aptos Coin Standard (or equivalent code)	Function or API call to platform, with parameters, to create a token - as tokens on Radix are native
Does the platform enforce asset standardization and guarantee how tokens and NFTs behave, reducing the chances for hacks and exploits?	No	No	Yes	Yes - as tokens on Radix are native
Do transactions include guardrails to ensure accounting is correct e.g. tokens don't get lost?	No - the developer is responsible for defining how the accounting for a token is done	No	Yes - some	Yes - the execution environment Radix Engine natively handles accounting
Can you call multiple smart contracts with a single atomic transaction?	No - you would have to deploy a specific smart contract which would then call other smart contracts	No - you would have to deploy a specific smart contract which would then call other smart contracts	No - you would have to deploy a specific smart contract which would then call other smart contracts	Yes - Transaction Manifest can call multiple smart contracts in one atomic transaction.
Is re-entrancy possible?	Yes	Yes	No	No
Is there an on-chain way to reuse common logic?	No	Yes - Solana Programs	Yes- reusable Modules	Yes - Radix Blueprints
Are there native features to easily allow for the creation of complex authorization and access systems?	No	No	No	Yes - Badges and native role based access control
Developer Tooling and Ecosystem				
How large is the developer ecosystem?	Huge	Large	Medium	Medium
Third party code repositories	Plentiful	Medium	Limited	Limited

02

**User Experience
(UX)**

User Experience

The experience for users of Web3 and DeFi today is complex and high risk.

To do things such as set up a new account, or perform a swap between two tokens, users are confronted with deeply technical requirements that they need to understand and accept, or if not, they could lose their assets.

To assess the user experience offered by our selection of leading platforms, we break this down into broadly five questions:

- Tokens and NFTs - are assets predictable, transparent, and secure?
- Transactions - are transactions predictable, transparent, and secure?
- Accounts - are accounts easy to set up and secure?
- Wallets - is the user interface easy to use and intuitive?
- Identity and Login - is my Web3 identity private and secure?

Each of these areas represents a core part of the crypto, Web3 and DeFi experience.

Tokens and NFTs

The role of tokens and NFTs in the blockchain ecosystem is undeniable. They are representations of value, be it monetary or symbolic. The simplicity with which a user can understand, trust, and interact with these assets directly impacts the adoption rate of the platform they're hosted on. A complicated and untrustworthy system discourages mainstream users and institutions alike.

After all, nobody wants to wade through complex smart contract code to ensure the safety of their assets, nor should they have to fear hidden vulnerabilities that could be exploited by malicious entities. In other words, they can simply be a question: do I know and trust what I am holding?

Ethereum

Ethereum pioneered how assets are represented on decentralized networks by allowing developers to deploy smart contracts with arbitrary logic and hold arbitrary state. No longer were tokens limited just to the primary, native token of the network, such as BTC for Bitcoin, or LTC for Litecoin; but instead, anyone could deploy their own token to the Ethereum network, giving rise to the ICO boom of 2017.

With this logic and state, Ethereum utilizes a model where a token is a smart contract. The smart contract contains a list of balances inside it, and those balances are controlled by logic deployed by that smart contract's developer.

The Ethereum community has coalesced around a series of smart contract templates, known primarily as ERC (Ethereum Request for Comment) standards, to create some standards and conformance across the tokens created. The most popular standards include:

ERC-20 - The Standard for Fungible Tokens: This is the most widely used standard for creating fungible tokens on Ethereum. A fungible token is one where each unit is interchangeable and indistinguishable from another unit. Examples include cryptocurrencies like DAI or USDC. ERC-20 defines a set of functions that the token's smart contract must implement, enabling functionalities like balance inquiries, token transfers, and getting the total supply of tokens.

ERC-721- Non-Fungible Tokens (NFTs): Unlike ERC-20 tokens, which are interchangeable, ERC-721 tokens are unique. Each token has distinct information or attributes, making them distinguishable. This property is ideal for digital collectibles, art, or any digital asset where individuality and provenance are crucial. The standard ensures that each token has a unique identifier, allowing for the transfer and inquiry of individual tokens.

ERC-1155 - Multi-Token Standard: An evolution in Ethereum's token standards, ERC-1155 allows a smart contract to produce fungible and non-fungible tokens. It is optimized for scenarios where users may need to batch multiple items, reducing the gas costs associated with deploying and managing multiple token types.

Since tokens on Ethereum are governed by smart contract code, they are transparent by nature. Anyone is able to read the code to understand how the token behaves. However, this requires someone to actually be able to read and understand that code in order to ascertain whether the token is in fact secure. This is a severe limiter in establishing a user-friendly UX.


Furthermore this approach to representing tokens has given rise to a multitude of user experience issues, such as:


Requiring users to provide “spend approval”, which is kind of like giving a third party fintech access to spend the money in your bank account, for it to provide you services.

An example spend approval

MetaMask Notification - Mozilla Firef... — □ ✕

1 of 2
requests waiting to be acknowledged

 0xab61...b14e ● Main Ethereum Network




Allow

Https://app.uniswap.org to spend your BAT?

Do you trust this site? By granting this permission, you're allowing Htps://app.uniswap.org to withdraw your BAT and automate transactions for you.

[Edit Permission](#)

 **Transaction Fee** [Edit](#)

A fee is associated with this request.

\$1.12
0.003271 ETH

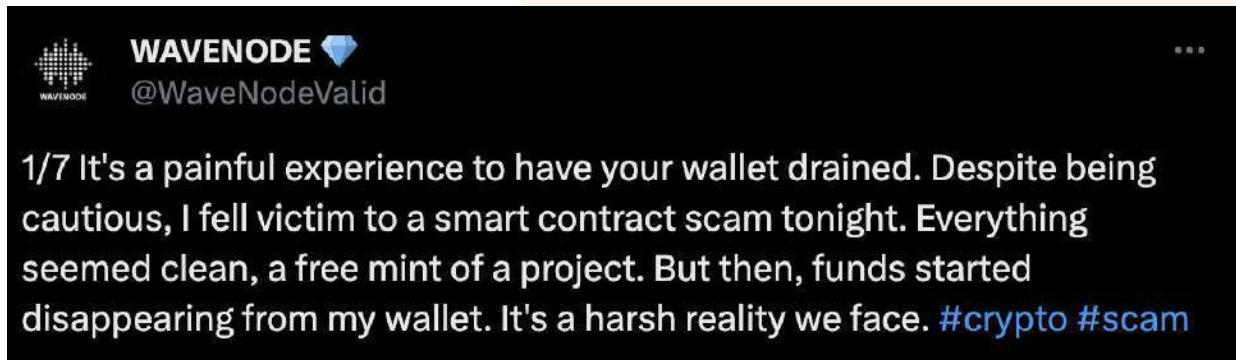
[View full transaction details](#) ▼

[Reject](#) [Confirm](#)

[REJECT 2 TRANSACTIONS](#) ▼

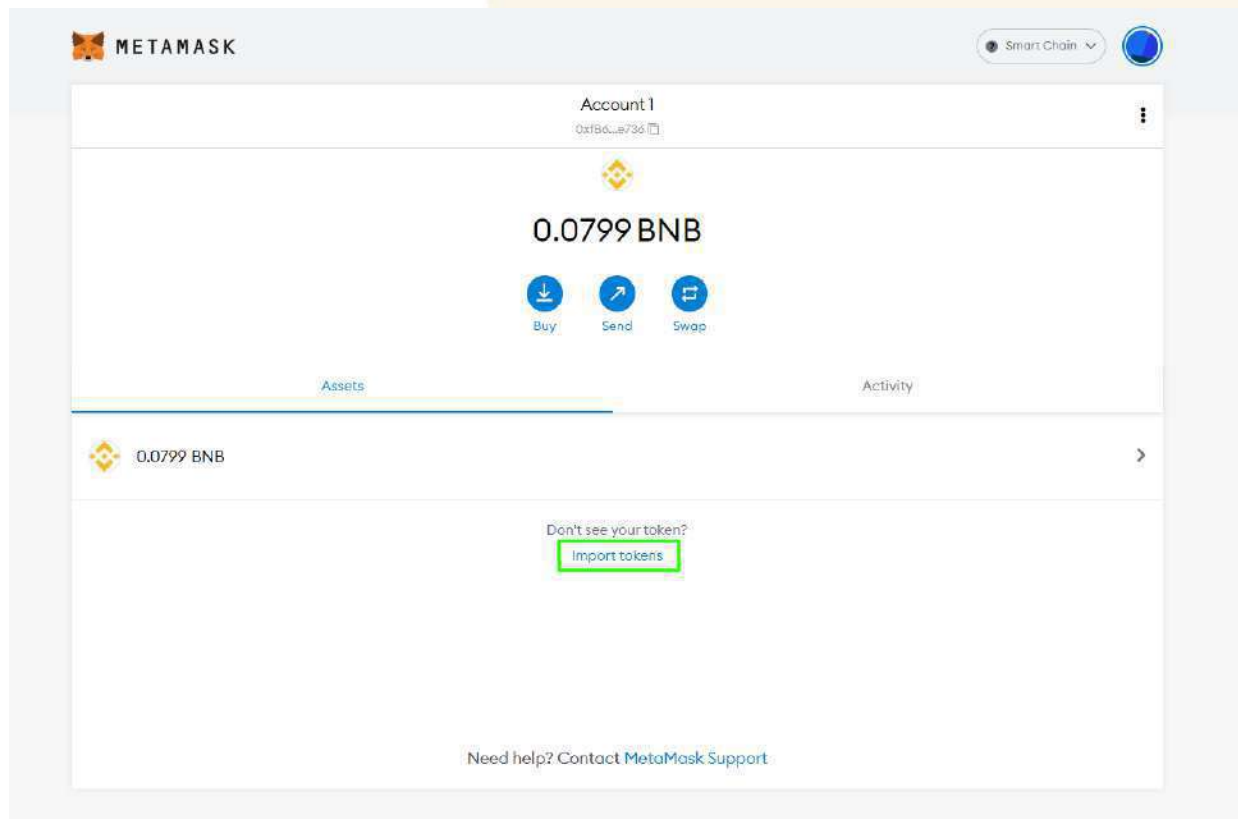
The State of Web3 User and Developer Experience

Users needing to be wary not to interact with a “malicious” token, that could potentially drain the user’s account if a transaction is signed interacting with that account.



Source

- Users not actually custodying their tokens in their account, as their account is just a record inside a third party developer’s smart contract.
- Wallets not knowing immediately all the tokens in a user’s account, requiring users to manually “import” the less common tokens to wallets like MetaMask.



Manually importing a token

Of course, it is worth stating that as the emphasis and adoption of L2 platforms continue to expand (Optimism, Arbitrum, etc.), Ethereum will continue to trend towards a base settlement/security layer, meaning that it is far less likely to need a comprehensive UX. But these UX challenges are presented by the capabilities of the EVM itself. And so any L2 that utilizes the EVM will also face these same UX issues.

Solana

While most blockchain platforms use the term "smart contract," Solana prefers the term "program." Similarly, instead of the commonly used "token standard," Solana introduces the concept of a "token program." These programs are primarily written in Rust and are housed in the Solana Program Library (SPL). SPL serves as a repository of on-chain programs curated by Solana's core team. Tokens minted on this platform are often referred to as SPL tokens.

The SPL standard allows for creating fungible tokens on the Solana blockchain, allowing developers and projects to create new assets using the standard easily. This is expanded to NFTs through the Metaplex protocol. Metaplex allows the creation of unique, non-fungible tokens. Solana's ecosystem has several wallets that support SPL tokens and Metaplex NFTs. The Phantom wallet, for instance, offers a smooth user experience and integrates both SPL tokens and NFTs seamlessly.

A notable feature of Solana's system is the unified Token Program that facilitates the minting, transferring, and burning of both fungible assets and NFTs. This means that for every new asset, there's no need to deploy a separate smart contract. Instead, an original "mint authority" address is designated, which acts similarly to a smart contract address in the Ethereum ecosystem.

Solana's Token Program improves upon Ethereum's ERC approach because the SPL is on-chain and there is less likelihood for a developer to make a mistake instantiating a token from the SPL, as opposed to Ethereum's manual approach. But while the SPL provides a base layer of security, the implementation specifics can introduce vulnerabilities. Thus, users have to be cautious and, in many cases, place trust in developers and their code.

Other user experience issues still manifest due to Solana and Ethereum sharing a similar smart contract account-based model to asset representation. Issues like wallet drains, needing to read the underlying code to truly understand how a token behaves, and the need to manually import tokens into your wallet still persist.

Aptos

To reimagine the blockchain experience, Aptos introduces a new approach toward token and NFT standardization that, in their eyes, optimizes for ease of use, security, and flexibility over competing blockchains. Similar to Solana, Aptos incorporated its own distinct token standard, separate from the EVM, named the Aptos Coin Standard. Unlike with the EVM (but similar to the Solana approach), the MoveVM removes the need for a new smart contract for every new token issuance, greatly reducing the associated gas costs. Within this design, there are two primary token standards:

Aptos Fungible Asset Standard: This standard helps to streamline the tokenization of diverse assets such as commodities, real estate, and in-game assets. Compared to Ethereum and Solana, Aptos simplifies creating and managing fungible assets into objects.

Aptos Digital Asset Standard (NFTs): This standard offers a standardized approach to defining unique digital asset ownership on the Aptos blockchain. Aptos redefines NFT creation and management by utilizing object-oriented models. With an emphasis on flexibility, composability, and scalability, the standard sets itself apart by allowing tokens to possess other NFTs, introducing a new layer of composability. It also promotes extensibility, supporting custom data and functionalities without altering the core framework.

The result of this object model to representing assets means that Aptos solves many of the aforementioned issues, for example:

- tokens on Aptos follow less unpredictable behaviors
- the Aptos Fungible Asset Standard and pre-signature transparency removes tokens as smart contract code, making malicious tokens less prevalent
- spend approvals are no longer necessary as token objects are transferred between accounts as opposed to the Ethereum and Solana model of tokens being a balance in a smart contract
- wallets on Aptos now instantly know what tokens a user holds: there is no longer a need to “import tokens”, as users actually custody their tokens in their account directly.

Using objects rather than the traditional account based model, Aptos offers an enhanced user experience over Ethereum and Solana, simplifying asset transfers and ownership.

Radix

Radix takes Aptos’ object model even further, enshrining tokens and NFTs as [native first class objects](#), understood and governed by the platform, entirely distinct and separate from smart contract code.

Unlike many existing platforms that treat assets as secondary elements often externalized from the core system (see for instance how tokens on Ethereum are represented as ERC20 contracts built by a developer, not enshrined in the core system), Radix integrates assets directly within the core functionality of its network.

Here, assets such as tokens or NFTs are not just additional entities; they are integral "resources" that Radix’s virtual machine, Radix Engine, recognizes, controls, and guarantees the behavior of.

This emphasis on native assets addresses a critical challenge in the DeFi landscape: clarity and trust. In many traditional Web3 wallets, users face ambiguity regarding their holdings. Tokens, especially those of the LP kind, frequently lack transparent details, necessitating users to rely on external platforms or decentralized apps for insights. This approach not only complicates the user experience but also introduces potential vulnerabilities from external dependencies.

Radix's native assets, however, ensure a direct and transparent representation. Each asset detail, from basic token attributes to complex NFT definitions, resides within the native asset's resource configuration. This approach eliminates the need for users to traverse intricate smart contract codes or external sites to understand how a token behaves. Because the behavior of assets on Radix are guaranteed by the Radix Engine virtual machine, this offers a clearer, more direct understanding of what a token is capable of doing, shielding users from potential risks associated with external asset verifications, and improving trust. This profoundly transforms the utility of the Radix Wallet. Armed to access each asset's configuration directly, the wallet offers transparent details on the details associated with each asset and the behaviors that its capable of, enhancing the user experience.



Radix labels itself the “Full Stack for DeFi” [Source](#)

By making assets a foundational aspect of its platform, this solves many of the aforementioned user experience issues:

- Spend approvals are no longer necessary (like Aptos)
- Users no longer need to be wary of “malicious” tokens that could drain a user’s account, as the behavior of tokens on Radix is governed by the Radix Engine virtual machine.
- Because token behavior is controlled by Radix Engine, the Radix Wallet can show exactly the behaviors that a token is capable of, such as if a token can be burned, recalled, or new supply minted.
- Users actually custody their tokens in their account, as a token is like a physical object inside the account.
- The Radix Wallet knows all the tokens in a user’s account immediately, solving the problem of having to manually import tokens.

Transactions

A transaction, in essence, is when a user updates the state of the ledger by signing a set of instructions. Most transactions involve transferring of funds from one account to another. It’s important for these fund movements to be correct, as if there is misapplied logic, then users could lose funds (or some users gain funds unexpectedly). Most transactions require a “gas” fee, which pays for the work performed by nodes to process the transaction.

Here’s a summary of how transactions work on our selected platforms.

Ethereum

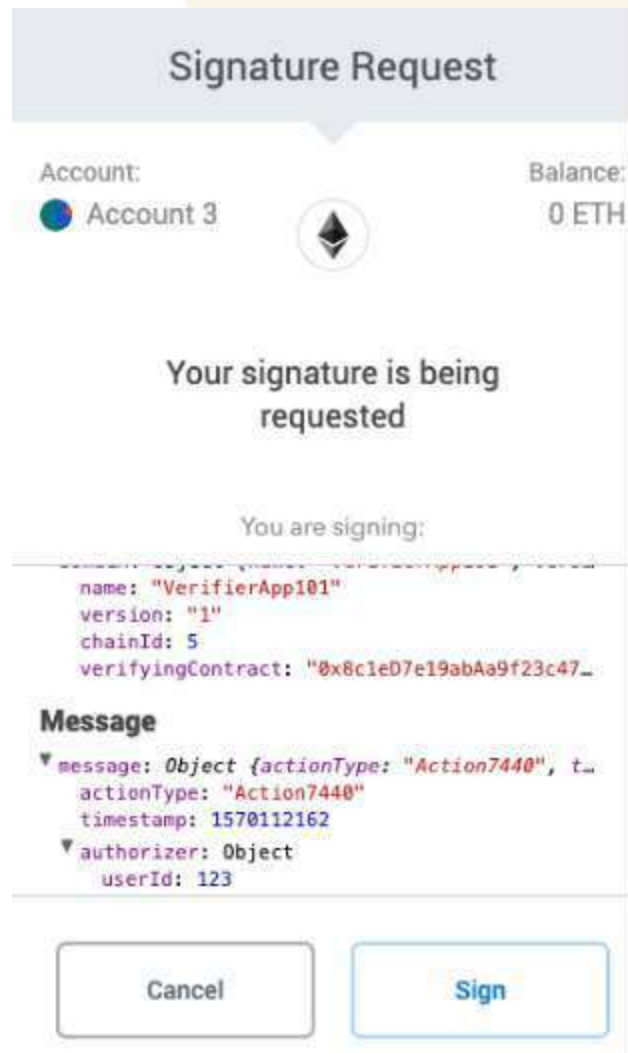
Transactions on Ethereum can either be for the native asset ETH only; or they are calls to contracts, such as ERC20 contracts. In both cases, the user signing the transaction must pay gas fees from the account they wish to use, meaning that users must have some ETH in each account they wish to use.

For contract calls, a transaction can call only one smart contract. That smart contract then serves as the gateway to further downstream contract calls. Some transactions can involve more than 10 different smart contracts calling further downstream contracts to update their own internal state, forming long chains of “delegate calls”. A major limitation of this approach is that if one of those downstream contract calls fails, all the smart contracts have to be able to cater to those failure modes, which can cause issues for users and cause additional complexity for developers.

Another barrier facing Ethereum’s UX is that the platform doesn’t understand, natively, what any token is other than ETH. To Ethereum, an ERC20 token is an arbitrary set of numbers that, if updated, must obey the logic defined by the smart contract. There is no on-chain requirement for ERC20 tokens to follow the same behaviors. It’s just a standard that a developer can manually choose to follow, if they want. This reduces the trust in ERC20 tokens as users don’t actually have guarantees on what a token will do, e.g. if the accounting will be performed correctly, or worse, if signing a transaction with it means that other tokens could be drained from the user’s account.

Furthermore this model means that when users interact with any smart contract, they need to “blind sign” their transactions. Blind signing a transaction means that wallets present a hash to the user - a string of what appear to be unintelligible letters and numbers, that the user signs. The wallet can’t present what the transaction will do in human-readable terms, because neither Ethereum nor its wallets knows what a token is, as it is just an arbitrary number that obeys arbitrary code in a smart contract.

Users can therefore be easily tricked into signing something they didn’t actually mean to, as a hash could be to approve the transfer of tokens inside a smart contract, or it could be to approve a smart contract to have approval to spend the user’s tokens, without their consent, commonly known as a “wallet drain”.



Metamask's non-user-friendly UI for signing a message.

From a safety angle, an oblivious user might unintentionally approve malicious activities. Moreover, from an adoption perspective, the daunting deluge of cryptographic information might repel prospective enthusiasts. While MetaMask has been a game-changer, its mode of transaction depiction might hinder more expansive adoption.

Third-party solutions like WalletGuard.app have entered the scene to counter these challenges. WalletGuard aims to render Ethereum's enigmatic transaction prompts into more digestible, human-friendly formats. This enhancement aids users in grasping the repercussions of their actions. Yet, these remedial tools, although laudable, are more of a band-aid to the existing infrastructure rather than a fundamental, systemic remedy, as the issue stems from how tokens are represented in the EVM. Depending on such external utilities introduces another layer of complexity for developers, often leading to increased integration labor and possible vulnerability spots.

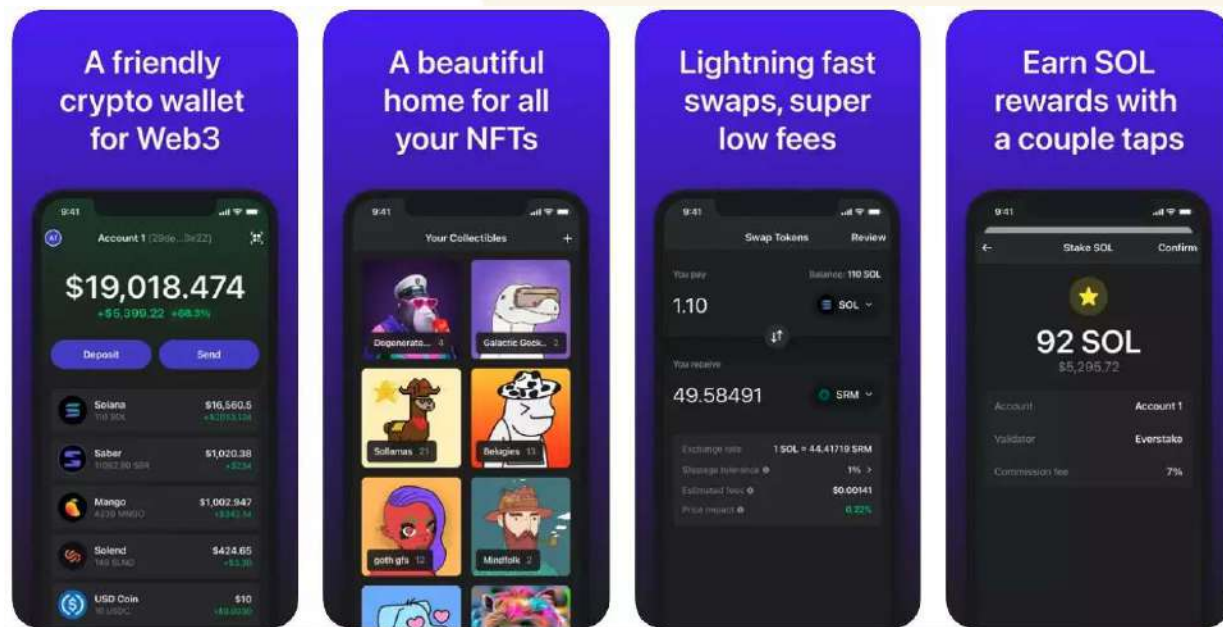
Solana

Solana's Phantom wallet aims to improve the user experience in earlier blockchain wallets. This design philosophy is evident in how it presents transaction details. Unlike MetaMask's often cryptic messages, Phantom lays out information in a more digestible manner, reducing the knowledge gap. However, it does not completely solve for the issues outlined above for Ethereum, such as wallet drains or blind signing, as can be seen in the example below.



Source

Phantom Wallet also features customizable gas fees, allowing users to make informed decisions that optimize their transactions based on the prevailing network conditions.



Source

Like Ethereum, a transaction can only call one smart contract at once. Complex transactions require a custom smart contract to be deployed to orchestrate a series of downstream calls for the transaction.

Aptos

Due to Aptos' new approach to accounts and "resources" on its blockchain, it is able to provide several improvements to transaction UX over Ethereum and Solana.

Notably, Aptos integrates transaction viability protection, which safeguards users against inadvertent transactions by placing constraints on each transaction's viability via a sequence number, expiration time, and chain identifier, shielding users from risks such as replaying a transaction that has already been committed, has expired, or has been committed on a different blockchain, such as a test network.

Additionally, the mechanism of pre-signature transaction transparency allows wallets to interpret transaction results in a readable format before the user signs it. This is pivotal in mitigating potential threats associated with signing harmful transactions. This feature achieves this by offering a detailed depiction of potential transaction outcomes before endorsement. To further fortify against fraudulent activities, this function can assimilate data on previous attack patterns and potentially harmful smart contracts.

Radix

Radix has developed an entirely novel approach to how transactions work on top of a public ledger, called the “Transaction Manifest”.

The Transaction Manifest is structured as a series of calls to each account or smart contract, specifying how native tokens should move between those accounts or smart contracts, and specifying what methods to call on smart contracts.

This opens up a number of advantages, namely:

- Transactions on Radix aren't limited to just calling a single smart contract - it's possible for a transaction on Radix to call many accounts and smart contracts atomically, all at once, and if any of those calls fails for some reason, the entire transaction is aborted without needing any special logic built by the smart contract developer to handle such an occurrence.
- Safer transactions because the Radix Engine virtual machine natively understands what tokens are, and natively handles the safe and accurate accounting that must occur during a transaction, such as tokens shouldn't be spent twice, or tokens shouldn't go missing during a transaction, which is logic that developers on Ethereum and other platforms must implement themselves. Radix Engine handles this accounting via containers for tokens called vaults (while assets are at rest) and buckets (while assets are on the move), and these must resolve correctly before the end of a transaction if the transaction is to be accepted.
- The Transaction Manifest also allows applications to be built that can compose multiple smart contracts atomically on the fly, without needing a special smart contract to be deployed just to orchestrate the transaction, opening up new use cases for applications built only in website front ends.

The State of Web3 User and Developer Experience

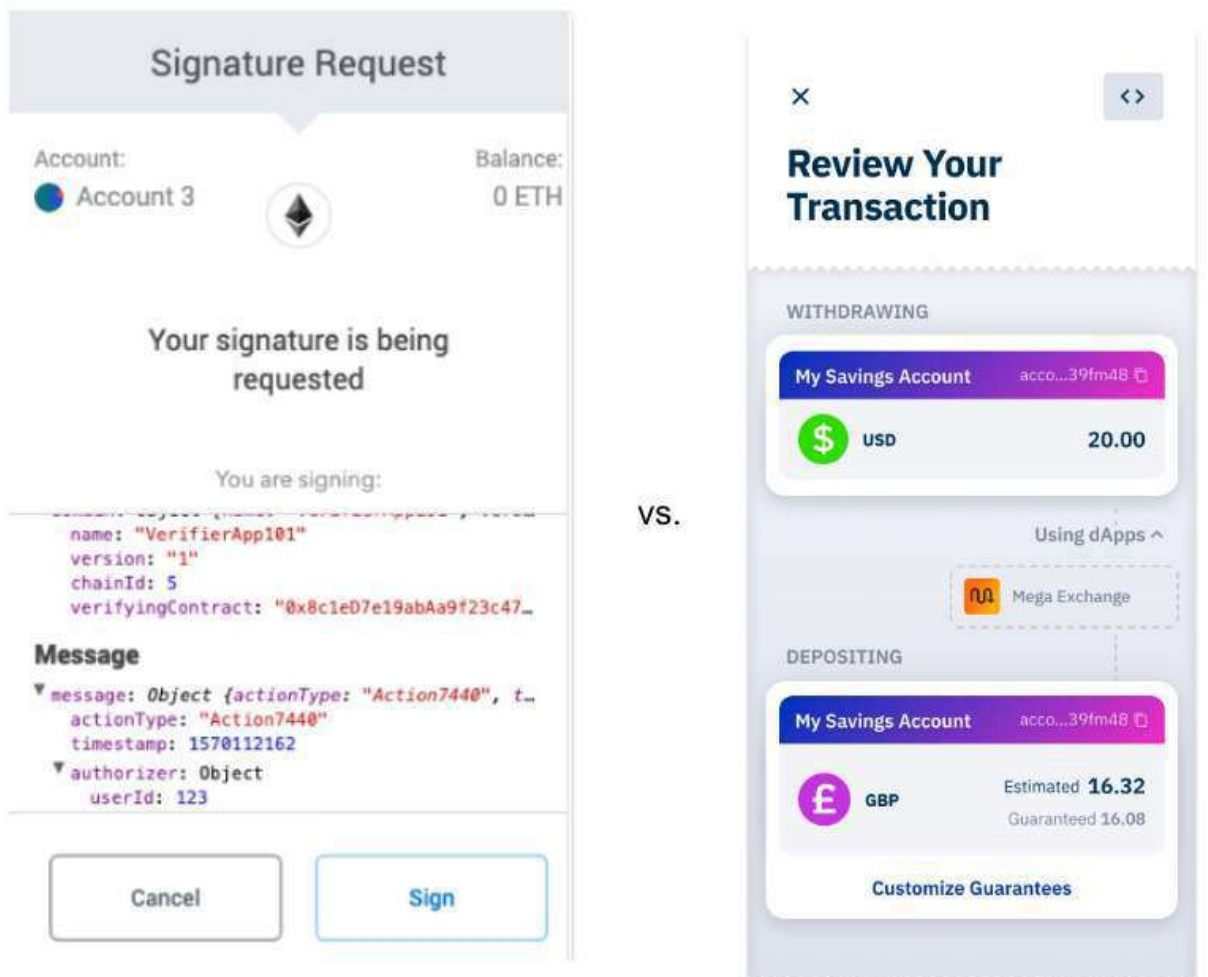


On Radix, the transaction manifest describes how assets are transferred between components. For most use cases, there is no need for a smart contract such as Furucombo to send messages to other smart contracts.

Source

Furthermore, as the Transaction Manifest specifies how tokens move between accounts, all transactions on Radix are human-readable. They do not need to be blind signed, as the Radix platform natively understands how tokens behave, and transactions are expressed to the Radix platform in human-readable terms.

This innovation ensures that users can comprehend the essence of their transactions irrespective of their familiarity with blockchain intricacies. A simple transaction like withdraw 20 USD tokens from my account and deposit 16.32 GBP tokens in my account is presented in human-readable-language, eliminating the ambiguities of blind signing on other platforms (see graphic below).



Metamask vs. RadixWallet reviewing a transaction UI.

Last, Radix users can also set customizable “Guarantees”, such as if they are conducting a swap between two assets, the network will guarantee that at least a certain amount of tokens are deposited into their account, or if that guarantee isn’t met, the transaction is aborted.

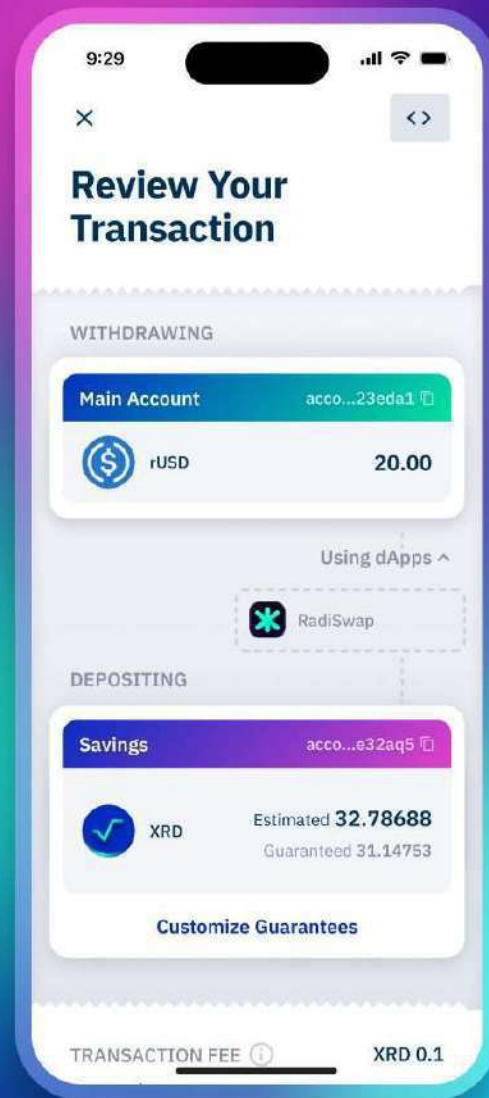


Human-Readable Transactions

Review your transaction before it happens with simple and clear language.

No more “blind signing” your assets away.

 RADIX



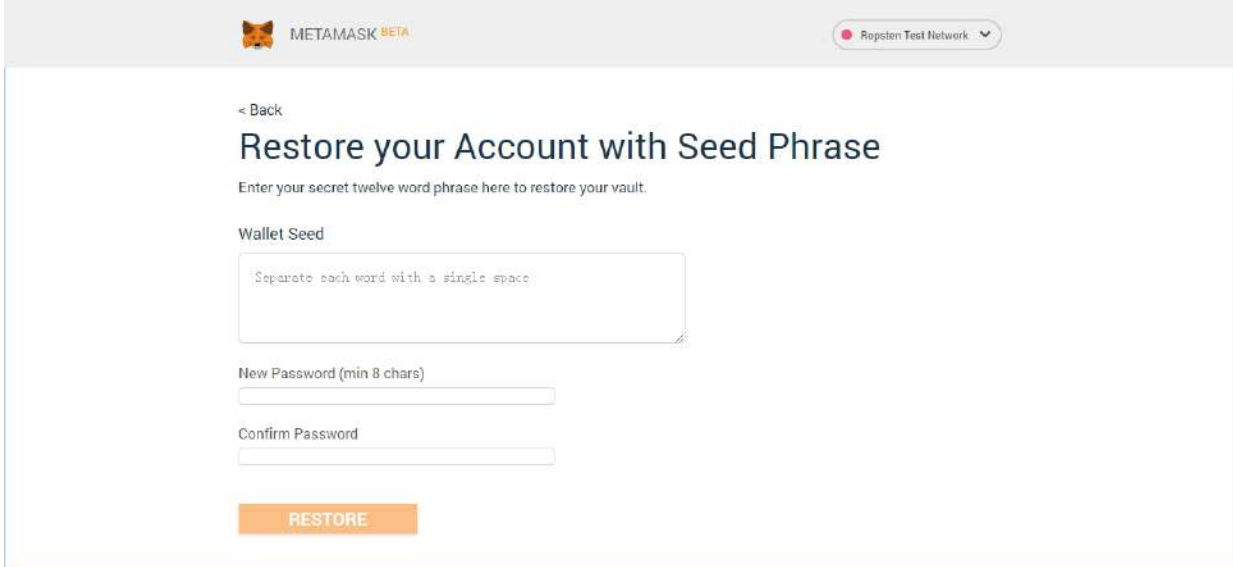
Source

Account Management

Accounts are where a user stores their assets. Accounts have typically been secured by users with a private/public key pair, where the private key is secret and used to secure the account, and the public key is mapped to the account's publicly known address, such as "0x...".

But private keys are long strings and hard to remember and prone to error (one incorrect character would invalidate it). To solve for this, users have seed phrases, mnemonics that are typically 12 to 24 words, that are used to recover a private key.

However, the primary issue with seed phrases is the "all or nothing" access they provide. While they grant users full control over their assets, the loss of this seed phrase translates to the irreversible loss of the assets contained within the associated account. Users are thus burdened with the monumental task of securely storing these phrases, often leading to anxiety and potential errors. The onus of security entirely lies on the user, which can be a deterrent for less tech-savvy individuals. Additionally, in targeted attacks, the seed phrase becomes a single point of failure: once compromised, an attacker gains full access to the user's assets.



The screenshot shows the MetaMask interface for restoring an account. At the top, there is a header with the MetaMask logo and 'BETA' text, and a dropdown menu for the network set to 'Ropsten Test Network'. Below the header, there is a '< Back' link and the main heading 'Restore your Account with Seed Phrase'. A sub-heading reads 'Enter your secret twelve word phrase here to restore your vault.' The form includes a 'Wallet Seed' section with a text input field containing the placeholder 'Separate each word with a single space'. Below this are two password fields: 'New Password (min 8 chars)' and 'Confirm Password'. At the bottom of the form is an orange 'RESTORE' button.

[Source](#)

Ethereum

Ethereum utilizes a dual account system comprising Externally Owned Accounts (EOAs) and contract accounts. At its core, EOAs, governed by private keys, facilitate simple transfers and serve as the foundational interface for most individual users. They facilitate routine transactions and the management of ETH, with popular wallets such as MetaMask, MyEtherWallet, and Ledger Live primarily engaging with these EOAs.

In contrast to EOAs, contract accounts are more complex as they, by definition, have smart contract code. While they're capable of holding and dispatching Ether, their true potency lies in crafting new contracts and executing operations as stipulated by their code.

For users, Ethereum accounts are typically secured by a single seed phrase that poses security challenges and makes the process of onboarding new users more difficult.

Ethereum ERC-4337 Account Abstraction promises to solve this problem by allowing users to use smart contracts for their accounts and identity instead of just a single account-private key pair. This will allow users to use multiple signing factors when authenticating to their account. While this does technically solve the issue, it introduces a great deal of new complexity such as a new transaction type, separate mempool, special “bundler” nodes, new “entry point” smart contracts, and with potentially higher transaction fees. It remains to be seen whether Ethereum can implement ERC-4337 effectively.

Solana

Solana takes a unique approach to account management, integrating both data storage and token ownership within its accounts.

In Solana's ecosystem, an account is more than just an address with a balance. It is a mutable data structure that holds information like account ownership, data for a particular program (akin to smart contracts in other ecosystems), or token balances for various SPL (Solana Program Library) tokens. This flexible account model allows Solana to execute high-speed transactions and supports its scalability goals.

While accounts in Solana can be created by anyone, they must maintain a minimum balance, known as "rent," to remain active. This mechanism ensures efficient memory usage and encourages optimal data management within the blockchain, but can cause issues in user experience given the requirement for maintaining a minimum balance.

Like Ethereum, accounts on Solana are secured by a seed phrase.

Aptos

At the core of the Aptos blockchain's asset management strategy is the "resource," a fundamental construct from the Move language. The resource approach introduces scarcity in representing assets and more easily enables access control over the resource. Every account housed within the Aptos blockchain is distinguishable by a specific 32-byte account address. What makes these accounts more than mere repositories of assets is their ability to store data in resources.

When an account is created, the only original resource are the core account details, specifically the authentication key and the sequence number. As accounts evolve, they accommodate more intricate resources, including Move Modules (code) and state/data termed Move Resources

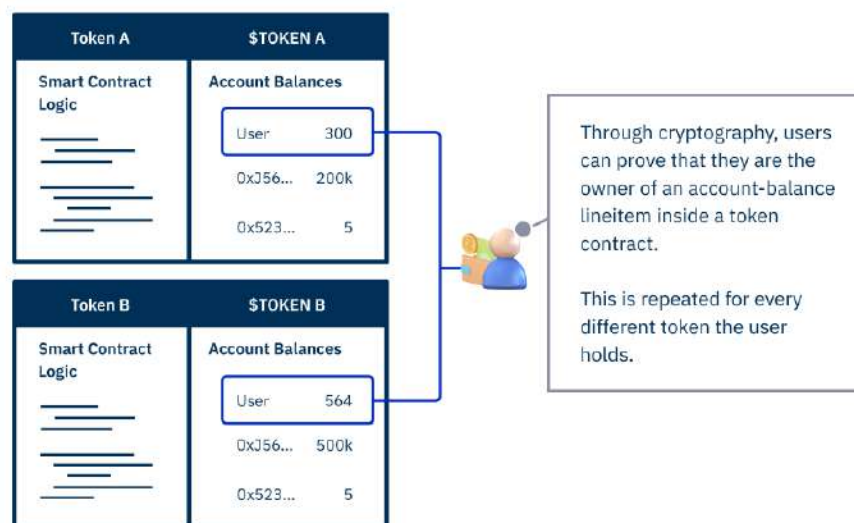
- **Move Modules:** These are crucial code devoid of data. Their primary function is to encode the protocols that dictate the updates to the Aptos blockchain's overarching state.
- **Move Resources:** In contrast to Move modules, Move resources are centered around data, without incorporating any code. Every piece of data stored as a resource aligns with a particular type, a definition that is firmly anchored in a module residing in the blockchain's distributed database.

Like Ethereum and Solana, accounts are secured by a seed phrase, but on Aptos there is the added ability to rotate keys, which allows a user to be able to re-secure their account if their seed phrase or private keys are compromised.

Radix

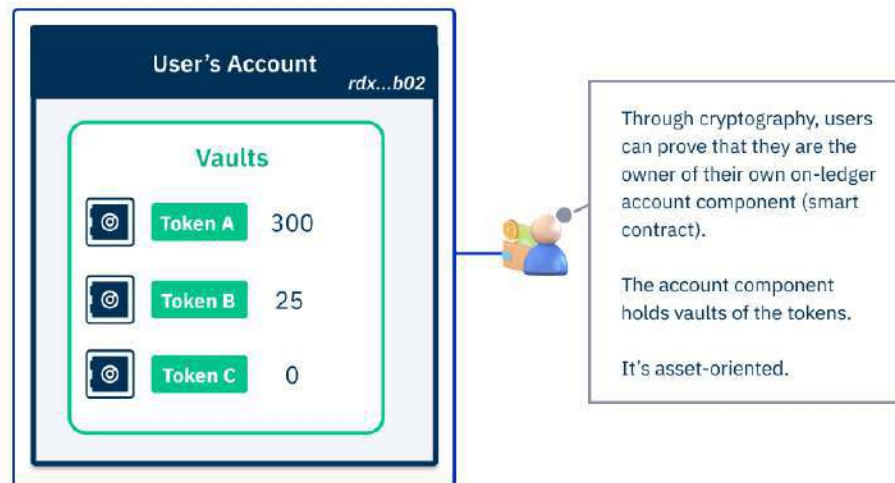
In Radix's ecosystem, an account is not just a private-public key pair that is linked to a record inside a developers' smart contract. Instead, an account on Radix is an on-chain container that “physically” holds tokens with customizable permissions. This grants users full control over their assets, as the native tokens of Radix physically live inside the account. Radix calls these “Smart Accounts”.

A1. What is an account? (Messages Only)



An account on Ethereum. [Source](#)

A1. What is an account? (Asset-Oriented)



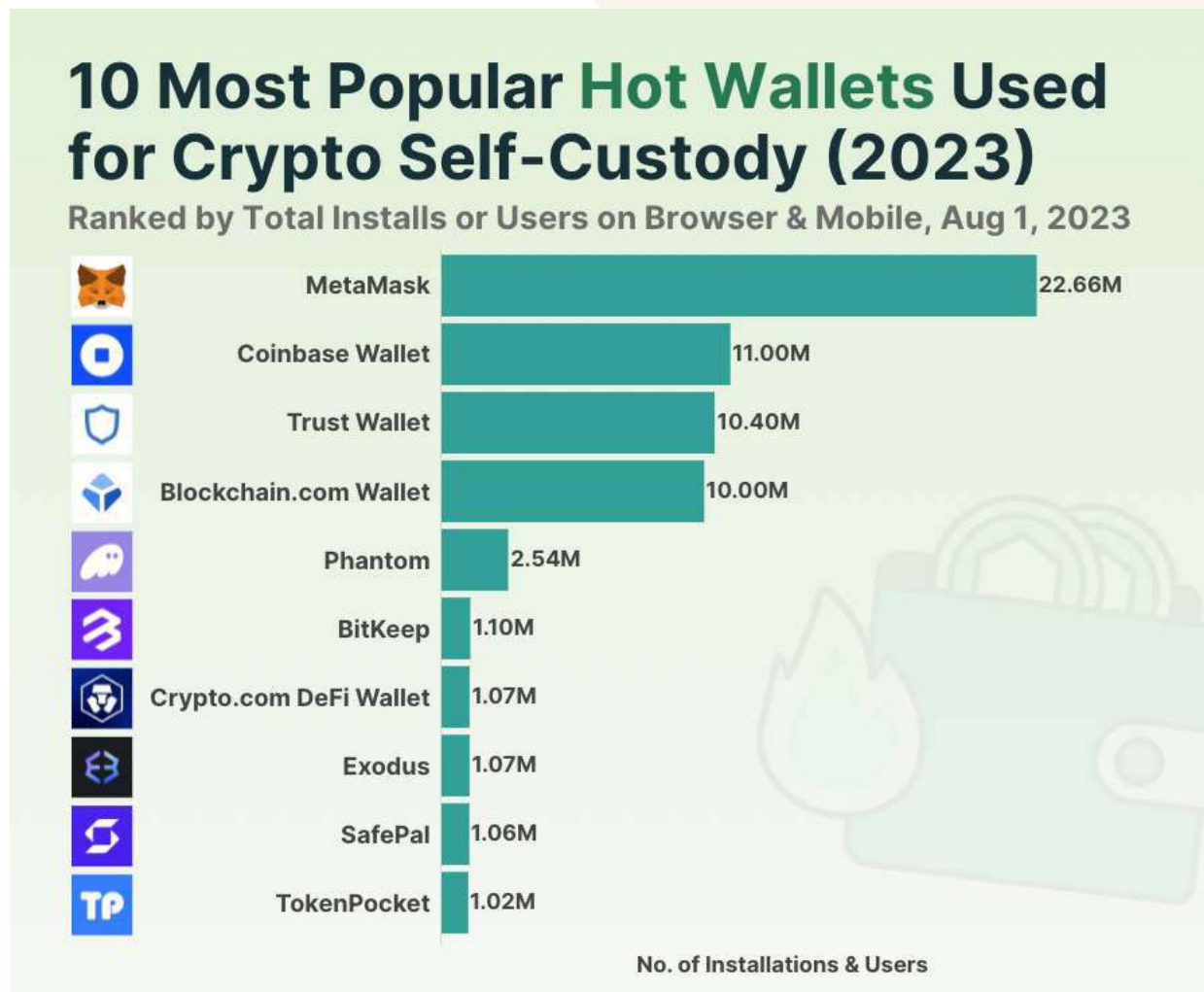
An account on Radix. [Source](#)

Radix's Smart Accounts allow users to set up multifactor authentication and recovery for their accounts with a combination of signing devices, such as requiring multiple mobile phones or other devices such as Yubikeys to sign a transaction or recover an account if one of the devices is lost or stolen. This native multifactor functionality diminishes the vulnerability linked to a single seed phrase. While seed phrases can still be used as a signing factor in the Smart Account permissioning system, Smart Accounts can require multiple seed phrases, or multiple other signing factors.

Smart Accounts' multifactor authentication and recovery functionality is currently live on the Radix Network today, although not yet implemented in the UX of the wallet. It promises to be a groundbreaking innovation for end user safety.

Wallets

Ethereum's MetaMask is the industry's most commonly used digital wallet. For many users venturing into decentralized finance or collectibles, their first point of interaction is often MetaMask, given its wide adoption and integration across platforms.



Source: CoinGecko

MetaMask is a pivotal gateway to the Ethereum ecosystem. At its essence, it's a multi-dimensional Ethereum wallet that functions as a conduit between users and the Ethereum blockchain. With MetaMask, users can effortlessly manage their ETH, engage with many Ethereum-based tokens like ERC-20 and ERC-721, and fluidly interact with decentralized applications (dApps). Its design goes beyond mere wallet functionalities, serving also as an interface for executing smart contract functions and dApp interactions. This dual nature of being both a wallet and an interaction portal makes it an invaluable tool for Ethereum.

The desktop version of MetaMask, available as a browser extension, boasts a minimalist and intuitive interface. Its design elements prioritize user ease, offering a clear view of balances, transaction histories, and token assets. Positioned conveniently at the top-right corner of browsers like Chrome and Firefox, it ensures users have swift access to Ethereum-based functionalities without needing a standalone platform. One of its standout features is the seamless dApp connectivity, where users can link their MetaMask wallet to various dApps, enabling immediate interactions without redundant logins.

In parallel, the MetaMask mobile application offers a more comprehensive experience tailored for on-the-go users. Upon launching the app, users encounter a neatly organized dashboard that succinctly displays their ETH balance, recent transaction activities, and a quick overview of token holdings. The mobile iteration has also incorporated QR code functionality, a feature that significantly simplifies asset transfers and dApp connections.

However, non-crypto natives may find storing large sums of funds in a browser extension to be a strange and insecure experience. And there are challenges associated with MetaMask such as the need to “import token”, as MetaMask does not know what tokens a user holds, as it does not know which smart contracts to look in.

Solana/Phantom

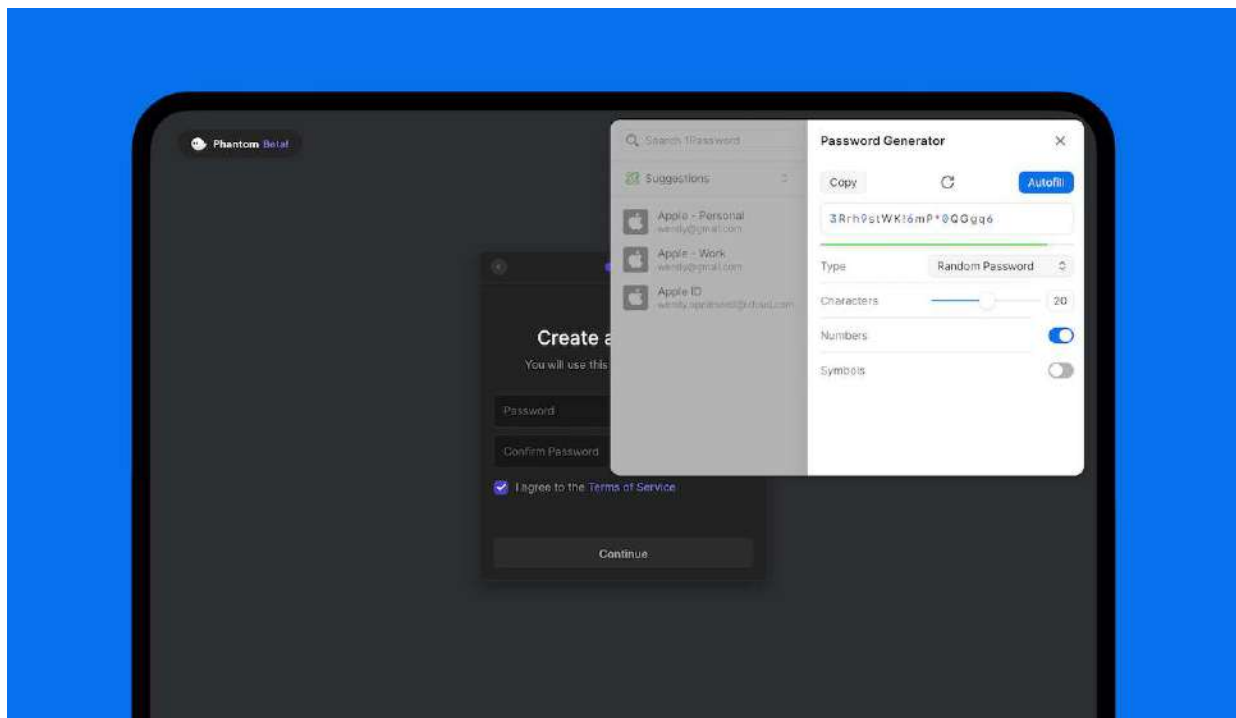
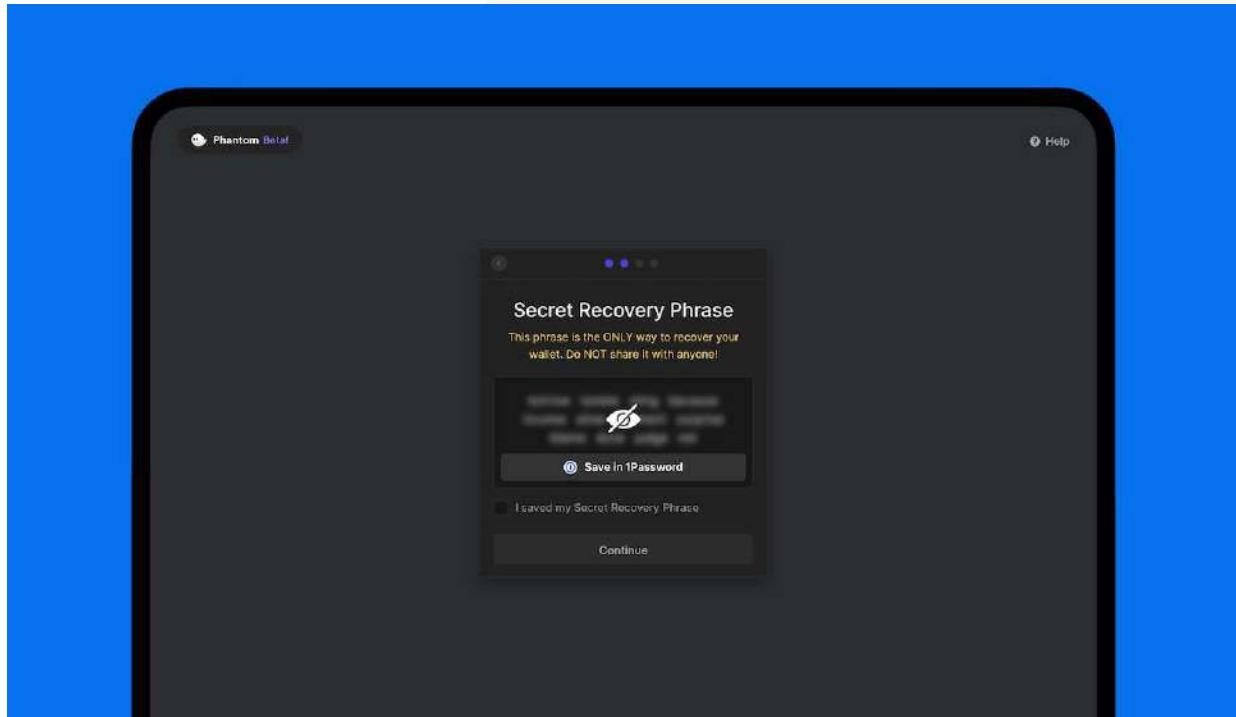
Navigating Solana, users rely on various wallets tailored for the platform. Phantom and Sollet are among the most popular. These wallets provide a user-friendly interface for managing SOL, Solana's native token, and other SPL tokens. They also support interactions with decentralized applications built on Solana, making them indispensable tools for the community.

Phantom, in particular, has garnered attention for its sleek design and seamless integration with popular web browsers, making it a go-to for many users. Additionally, with the growth of Solana's ecosystem, these wallets continually evolve, introducing features like NFT displays and staking capabilities, enhancing the overall user experience on Solana.

Recognizing the diverse usage patterns of modern users, Phantom offers versatility across platforms. For web users, Phantom integrates by injecting a 'phantom object' into the JavaScript context. This allows web apps to engage seamlessly with the wallet. Simultaneously, Phantom utilizes universal and deep linking techniques for mobile users, facilitating smooth interaction between mobile applications and the wallet.

The collaboration between Phantom Wallet and 1Password brings forth several advantages over Ethereum. First, access is notably simplified. With 1Password's "Save in 1Password" API, users can save their entire set of Phantom wallet credentials, including private keys, directly to 1Password. This reduces the hassle of remembering complex login details, offering a unified access point to digital assets. However, this UX improvement comes at the cost of introducing additional attack vectors, compromising a user's self-sovereign control of their assets, which kind of misses the point of Web3! The end user is always responsible for deciding which ratio of convenience-to-security is appropriate for their needs.

The State of Web3 User and Developer Experience



Source

Aptos/Pontem

[Pontem Wallet](#) is the leading Aptos wallet, offering a mobile version and a browser extension. The mobile version maintains most of the wallet's primary features, including NFT management, a dApp exploration page, and in-wallet swaps. All of this, combined with the wallet's native liquid staking and Ledger support, allows users to have self-custodial control over their digital assets while navigating the Aptos ecosystem.

The screenshot shows a swap interface for APT to USDC. At the top, there is a logo with a Santa hat and the text "DEVELOPED BY PONTEM NETWORK". The main swap area has a "Swap" title and a settings gear icon. The first input field shows "0.001" APT with a balance of "0.00951999". The second input field shows "0.003305" USDC with a balance of "0.050536". Below the inputs, it states "1 APT = 3.305 USDC (including fee)". The "Price Impact" is "0.00%", the "Fee (0.2%)" is "0.000002 APT", and the "Curve Type" is "Uncorrelated". A large purple button says "Swap APT to USDC". At the bottom, the "Currency Reserves" section shows "134,615.66" APT and "445,796.79" USDC.

DEVELOPED BY PONTEM NETWORK

Swap

0.001 | APT Balance: 0.00951999

0.003305 | USDC Balance: 0.050536

1 APT = 3.305 USDC (including fee)

Price Impact 0.00%

Fee (0.2%) 0.000002 APT

Curve Type Uncorrelated

Swap APT to USDC

Currency Reserves

APT	134,615.66
USDC	445,796.79

Source

Additionally, there's a high degree of emphasis placed on delegation with the Aptos design, giving users flexibility in their overall management and involvement on the Aptos blockchain. Besides the APT serving as a delegation token to validators, Aptos also supports private key delegation, an option not available on most other blockchains. The private key delegation allows for the temporary transfer of control over a wallet to a third party. The third party that the wallet has been delegated to holds a specified level of authority over the wallet, but the original user ultimately has full control and ownership over the wallet and all associated assets.

Pontem displays a comprehensive and forward-thinking approach to digital wallets, especially in terms of its user-centric features. Both platforms, while distinct in their offerings, underline the necessity of evolving user expectations concerning digital wallets.

Radix Wallet

The Radix Wallet mobile app is where the groundbreaking features of the Radix Network are exposed to the user, such as the Transaction Manifest's human readable transactions and guarantees on deposits, native assets with transparent behaviors, and soon-to-come to the Radix Wallet, the multifactor authentication and recovery capabilities of Radix's Smart Accounts, obsoleting seed phrases.

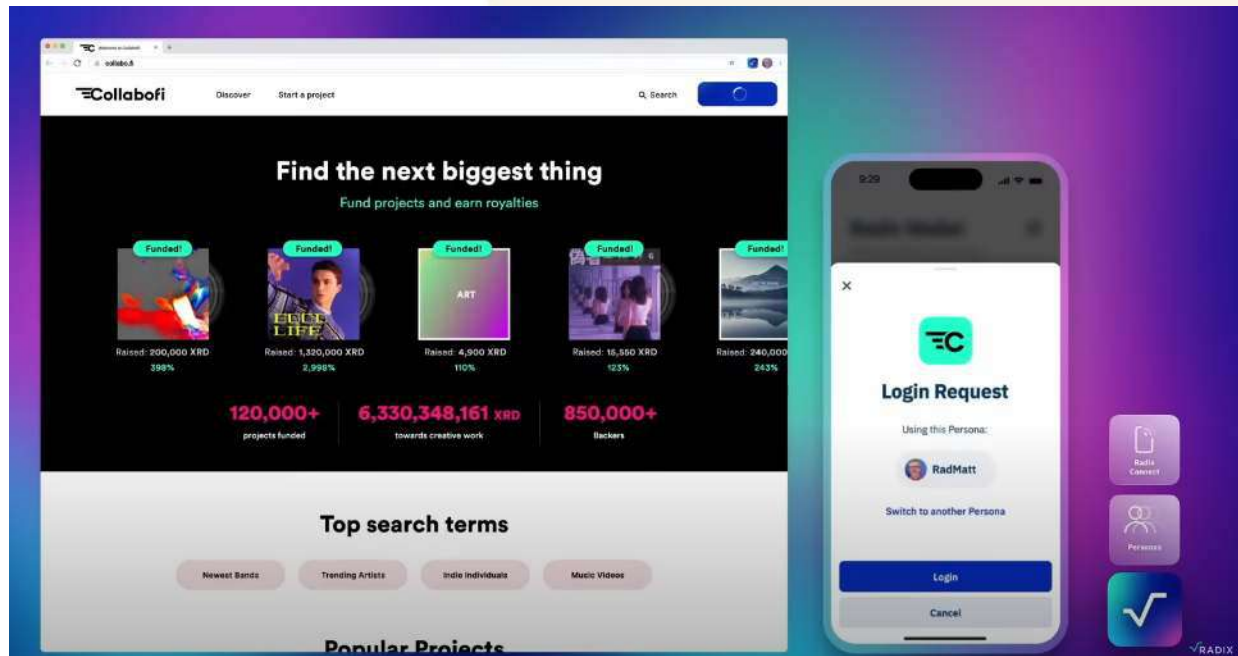
The Radix Wallet was developed in response to the problem of users putting large sums in browser extensions, which are often insecure, and that users switching between desktop and mobile sessions could not have a seamless, unified experience.

To solve this, the Radix Wallet is a mobile app only, but can still provide users the full power of a full screen desktop experience through Radix Connect. Radix Connect is a desktop browser extension that can create an encrypted P2P link between the mobile Radix Wallet app and a desktop session. This connector extension doesn't hold any funds - it serves only as a means to connect the mobile wallet and desktop session when needed.

With the [Radix Wallet](#), users can leverage full screen desktop Web3/DeFi dApps, but when it comes to signing a transaction, the desktop session will automatically request the mobile app, and the user authorizes any requests on the mobile combined with any relevant hardware wallets.

This improves security because the mobile app authorizing transactions is segregated from the desktop browser navigating Web3 and DeFi websites. And it creates a single unified experience where the wallet is always and only on your phone. There is no desktop wallet - just the connector extension, therefore users will always have their primary wallet with them at all times - even when they're on the go.

In future, Radix will also support deep linking so that users can experience Web3 and DeFi web dApps only on the phone.



Source

Web3 Identity and Login

Ethereum does not natively offer much within the Web3 identity or login space, instead relying on builders and third-parties to experiment and develop solutions atop the protocol. One such example is SpruceID. Within the decentralized identity space, SpruceID has introduced an authentication solution known as SIWE (Sign In With Ethereum). SIWE facilitates users to autonomously manage their digital identity by leveraging EVM-compatible wallets. By adopting a standardized sign-in workflow, SIWE can integrate with both existing identity services and Web3 applications. This methodology doesn't only cater to the emerging Web3 applications but also provides potential integration points for established Web2 platforms. Importantly, SIWE provides this service without requiring users to engage in blockchain transactions, thus removing any additional associated costs. As an added layer, Web2 platforms integrating SIWE can provide an entry point for users unfamiliar with web3, without altering the core user experience.

The most popular implementation of Sign in with Ethereum is Metamask, which can use one of your accounts to sign in to popular Web3 dApps. For example, if you had account 0x123..., then you would authenticate to the dApp with this account without needing an on-chain transaction, using just cryptography. This creates the potential for a passwordless future.

However, some drawbacks of this implementation on Ethereum is that, as with any token holding account, authentication to the account is controlled by a single seed phrase. If you lose that seed phrase, then you've permanently lost your account, your identity, and your login.

Another prominent Ethereum-based Web3 identity service is Ethereum Name Service (ENS), a new naming system that translates user-friendly addresses, such as 'alice.eth', into machine-recognized formats like Ethereum addresses and content hashes. While similar in concept to the Internet's Domain Name Service (DNS), ENS's structure is distinct, shaped by Ethereum's capabilities. Owners of domains, like '.eth', can manage subdomains via smart contracts called registrars. Moreover, ENS offers integration of pre-existing DNS names and allows owners to configure subdomains at their discretion.

Solana

The most prominent wallet on Solana, Phantom, provides a multi-wallet system that allows users to consolidate multiple Secret Recovery Phrases and private keys from chains like Solana, Ethereum, and Polygon into a single wallet, with the capability to manage up to 100 accounts. Users can easily migrate their existing addresses from widely-used wallets such as MetaMask, Coinbase Wallet, and Rainbow into Phantom.

For Web3 login and identity, Solana provides [Sign-in With Solana](#), which operates similarly to Sign-in With Ethereum.

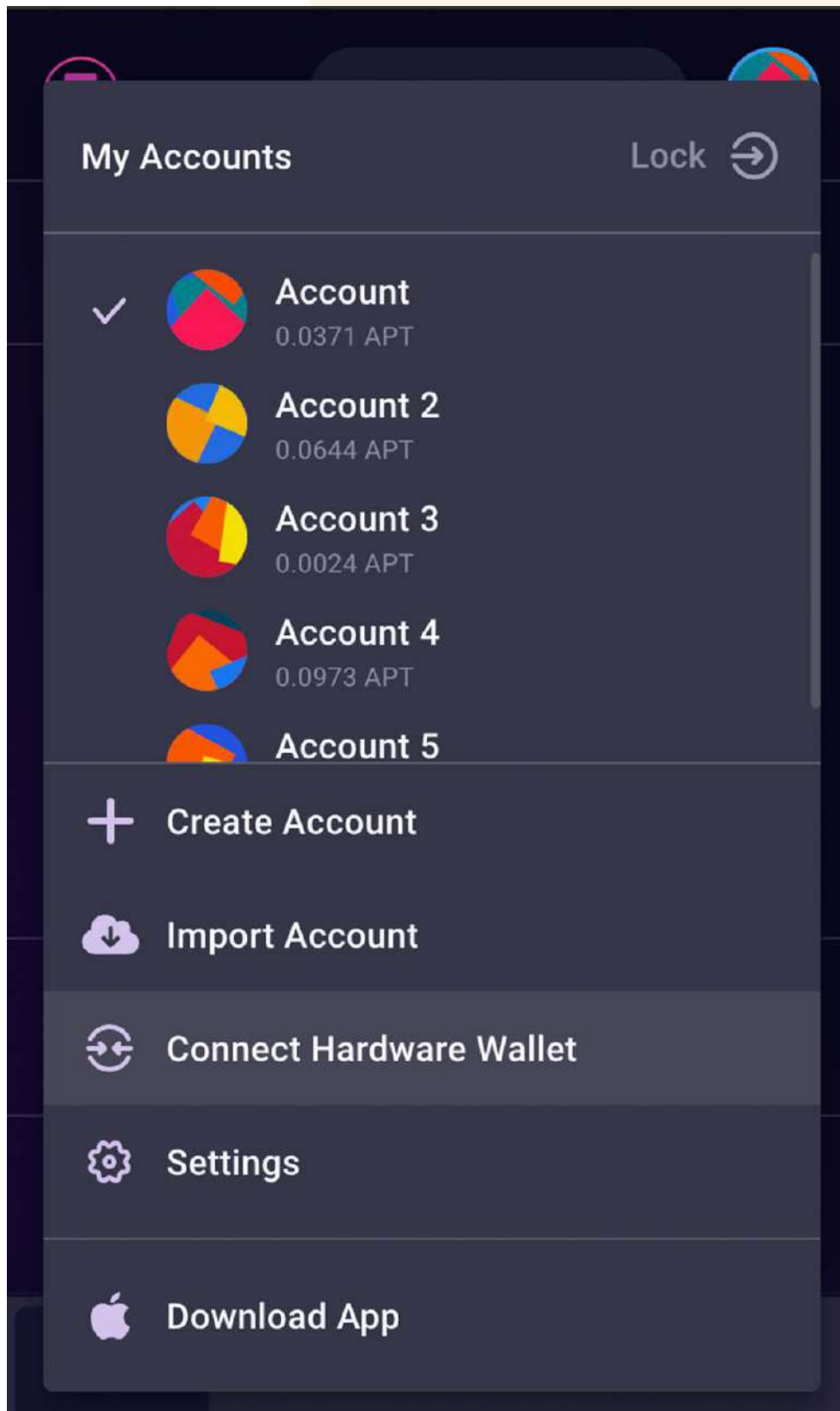
The same drawbacks to Web3 login and identity on Ethereum still apply to Solana.

Aptos

With traditional blockchains like Ethereum and Bitcoin, users' on-chain identity is tied to their private keys. As empowering as it may sound, binding one's identity to a private key poses significant risks. Misplacing or compromising the key doesn't just jeopardize your assets; it threatens your entire digital identity. While temporary identities exist for privacy or development reasons, other solutions beyond initial key creation are available. Aptos is one blockchain attempting to incorporate these new solutions.

With Aptos, a user is not bound by a singular, immutable private key. Rather, Aptos users can rotate or change the key (like in a scenario where that key has been compromised), ensuring the security of their digital identity.

Additionally, Aptos incorporates multi-factor authentication across accounts. Multi-signature authentication bolsters security and can act as a redundancy system for blockchain data.



Pontem wallet with multiple accounts and hardware wallet connectivity. [Source](#)

As blockchain moves from a niche domain to mainstream acceptance, platforms like Aptos, which prioritize the seamless integration of identity and security, will lead the way. By catering to modern requirements, Aptos not only ensures the safety of its users but also positions itself as a front-runner in the mission to introduce the next billion users to Web3.

For signing into Web3 applications with Aptos, Aptos has Identity Connect, which allows users to sign in with their account (a specific address on-chain with a corresponding key pair). [Source](#)

Persona Creation on Radix

Web3 identity has grown in importance in the blockchain domain, focusing on user privacy, authenticity, and autonomy. Radix stands out, particularly with its "Personas" feature. Many traditional platforms link users to one digital identity, potentially merging their activities across different sectors. Radix's "Personas" offers a solution to this limitation.

Users can set up distinct digital identities on the Radix platform for specific purposes. For example, one persona could be for gaming, tracking achievements and interactions, while another could be tailored for DeFi actions. Importantly, these personas allow users to maintain privacy, ensuring different activities don't overlap. Additionally, as Personas are dedicated identities on the Radix Ledger, there is no conflating of a user's token-holding account and their digital identity. Those two concepts on Radix are separate and distinct, supporting more private use of Web3.

Furthermore unlike Ethereum or Solana, Personas will be able to be configured with multifactor authentication and recovery, allowing them to be freed from the all or nothing seed phrase that controls a user's Ethereum or Solana identity. This gives users far more assurance, as if they lose one device, there are still multiple backups (or combinations of other signing factors) that can provide a user with access to their account, bolstering security and recoverability.

Radix also offers storage of personal data in the wallet so that when a user needs to share information with a dApp, they can do so at the time of the request. Web3 identity, as shown by Radix's "Personas," emphasizes interaction, privacy, and autonomy. As the decentralized space advances, features focusing on user-centric needs will likely set industry standards, with the user's needs and rights at the forefront.

Summary

The Web3 user experience to date has unquestionably been defined by Ethereum and its ecosystem of EVM L2s, which utilize MetaMask and other wallets; and most recently with the surge in popularity of Solana and its wallets including Phantom. With the most users, liquidity, and dApps, these are the two user experiences that lead the way in terms of adoption.

However, these user experiences do not come without significant challenges. Frequent hacks, frequent stories of users getting their wallet drained, and stories of users losing their funds through lost or compromised seed phrases or blind signed transactions are all too common.

Aptos and Radix bring a multitude of user experience benefits, including more transparent transactions with Aptos' transaction viability protection, and Radix's human-readable Transaction Manifest, which are fundamental to building confidence in use of Web3. With native assets and dedicated Personas for identity and login Radix's tokens follow predictable and transparent behaviors, further increasing user's trust in Web3.

The State of Web3 User and Developer Experience

By solving core problems in transparency, trust, and security holding back mainstream adoption, Radix's full suite of UX improvements stands it in a strong position to onboard the next wave of non-crypto natives to Web3.

03

**Developer
Experience (DX)**

Developer Experience (DX)

Developers, the architects who build decentralized applications and systems, are key to enabling the next phase of Web3's growth. Their experience, DX, is what allows them to build their business and application ideas into reality, with the aim of doing so quickly, intuitively, easily, and securely. A good DX means a developer can enter an ecosystem and construct projects, tools, protocols, and applications quickly and safely.

To dissect the Developer Experience (DX) offerings of Ethereum, Solana, Aptos, and Radix, we have assessed them against the following two criteria:

Language Adoption and Usability: The ease with which developers can learn or use a programming language to build Web3 and DeFi dApps.

Developer Tooling and Ecosystem: The tools, SDKs, documentation, community support, and other resources supporting developers. A rich ecosystem can significantly reduce development time and enhance the quality of dApps and other blockchain projects.

Ethereum

Ethereum has significantly influenced the developer experience in the blockchain space. Central to Ethereum's architecture are its smart contracts, computer logic that executes autonomously on top of a public ledger.

Developers begin their journey on Ethereum by writing smart contracts using a specific programming language called Solidity. This high-level, statically typed language is tailored for the Ethereum Virtual Machine (EVM) and provides the syntax and tools required to create intricate decentralized applications (dApps). Solidity's development environment offers features like inheritance, libraries, and user-defined types, making it powerful and flexible.

Once a smart contract is drafted in Solidity, it's compiled into bytecode, which is then deployed onto the Ethereum network. This deployment acts as a transaction, sending the contract code to a special address. Once deployed, a smart contract gets its permanent address on the blockchain.

Every smart contract action, whether updating a state or invoking a function, requires a transaction. Developers use web3 libraries, such as web3.js or ethers.js, to interact with their smart contracts from external applications. These libraries provide the necessary tools to create, sign, and send transactions and listen for events emitted by smart contracts.

Furthermore, developers often streamline their workflow by using platforms like Truffle or Hardhat. These frameworks offer testing environments, smart contract compilation tools, and network management capabilities, making deploying and managing contracts more intuitive.

While Ethereum has provided a robust dApp development platform, developers face challenges. The dynamic nature of gas prices, occasional network congestion, and the complexities introduced by smart contracts and composable dApps have led to both developer frustration but also [numerous bugs and hacks](#). Some common vulnerabilities in Ethereum smart contracts that have been exploited are:

1. **Re-Entrancy:** One of the most common attacks in smart contracts, re-entrancy consists of an attacker calling a function recursively in order to damage the protocol, often by stealing funds.
2. **Simple Code/Math Bugs:** These occur when there is an error in a mathematical formula or in the calculation process, such as rounding mistakes.
3. **Faulty Proof Verification:** Especially relevant in bridges and other cross-chain protocols, this occurs when there is a faulty verification proof on one chain which allows the attacker to falsify actions on the other paired chain.
4. **Incorrect Call Permissions Check:** This vulnerability arises when the caller's ability to execute the function is not properly set. For example, a function that should be executed only by certain roles is left open for anyone to call.

As a result of these vulnerabilities, while writing your first Solidity smart contract can take only a matter of hours, it takes years to learn all the intricacies and vulnerabilities in Solidity to be able to build secure Web3 dApps, and even then, new vulnerabilities continue to emerge, with user funds perpetually at risk, and developers needing to dedicate substantial time and resources to validating and auditing their code.

The impact of these issues cannot be overstated. Over \$3Bn was lost in DeFi hacks in 2022, and \$1.7Bn in 2023. With such a large percentage of the ecosystem at risk, regular and institutional users will find it hard pressed to find a compelling reason to put significant sums of their assets into Web3 and DeFi, until these issues are permanently solved.

Solana

The Solana development experience is significantly influenced by its unique architecture, particularly its parallel smart contract runtime known as Sealevel. This technology sets Solana apart from other blockchain platforms like Ethereum, which primarily use single-threaded runtimes.

A key aspect of Solana is its focus on optimizing transaction throughput on high-performance, multi-core machines. This approach is driven by the observation of an exponential growth in the number of cores in computers, prompting Solana to design transactions that can be easily parallelized. This parallelization is a fundamental feature of Solana's architecture, enabling it to process transactions more efficiently compared to traditional, single-threaded blockchain platforms.

The account model in Solana is notably different from Ethereum. While Ethereum assigns each smart contract its own storage within its account, Solana employs a different approach. On-chain programs in Solana are immutable accounts used only to store executable bytecode. The actual state for these programs is stored in separate, non-executable accounts. This segregation allows Solana developers to design their smart contracts to be parallelizable. By spreading the smart contract state across multiple accounts, these accounts can be used in parallel transactions without data conflicts.

Solana's transaction processing is sophisticated and involves multiple components. Each transaction must list all the accounts it will read from, write to, and invoke as a program. This pre-listing enables Solana validators to know which transactions can be processed simultaneously without conflicts. This method allows for efficient processing of transactions, leading to higher throughput and lower fees.

Developing on Solana, however, poses certain challenges and complexities due to this architecture. The need to consider how to parallelize transactions and manage multiple accounts adds a layer of complexity that is less prevalent in platforms like Ethereum. Developers are required to think meticulously about the design of their on-chain programs and the associated accounts they will manipulate. This leads to a steeper learning curve, but also opens up possibilities for more efficient and scalable applications.

While the Solana developer experience offers significant benefits in terms of scalability and efficiency due to its parallel smart contract runtime, it also introduces additional complexities in the development process. Developers must adapt to a different account model and the need for careful planning of parallelizable transactions and state management.

Aptos

To address specific development challenges present in other blockchains, Aptos embarked on the implementation of the Move language built as part of Facebook's Libra/Diem project.

At its core, Move introduces several novel concepts and features that distinguish it from other blockchain programming languages. Firstly, it provides a type system and resource-oriented programming model that ensures digital assets cannot be cloned or accidentally destroyed. In Move, assets are represented as "resources," a type that cannot be copied or implicitly discarded, ensuring their conservation and integrity.

Moreover, Move is designed to be expressive and flexible, allowing developers to define custom resource types and write complex business logic for decentralized applications. Additionally, Move's module system and ability to update code dynamically make it adaptable, enabling continuous improvement and evolution of smart contracts on the Aptos blockchain. These features make Move a powerful tool for developers looking to build secure, efficient, and scalable decentralized applications.

One of the standout features of Aptos is its commitment to formal verification, a process that mathematically validates that software code aligns with predetermined rules or properties. Through the Move Prover tool, Aptos provides tooling for developers to test the integrity of their smart contracts. This tool is adept at neutralizing potential threats within smart contracts, such as the aforementioned double-spending and reentrancy attacks. By employing formal verification, Move Prover not only mitigates these specific threats but also detects other coding anomalies, bolstering the overall stability of the system.

Furthermore, MoveVM's adoption of static dispatch for function calls offers an additional layer of security against reentrancy attacks. This contrasts with the Ethereum Virtual Machine's (EVM) dynamic dispatch approach. The distinction lies in the timing of function calls during the program's execution. With static dispatch, function calls are made at compile-time, allowing for an early error-checking phase. This means that in MoveVM, smart contracts are verified at an earlier stage, effectively nipping potential reentrancy attacks in the bud.

Another key distinction between Aptos and Ethereum is the dynamic gas management system. While Ethereum's gas fees can be unpredictable due to network congestion, Aptos has integrated features like built-in gas price estimators to offer a more real-time, dynamic method to gauge transaction costs. This dynamic estimation method gives developers a clearer picture of the anticipated gas fees, allowing for better financial and technical planning.

Aptos also offers an advantage in terms of transaction prioritization. The platform's bucket-based prioritization system enables developers to adjust transaction priority through the gas unit price, granting them greater control over the urgency of transaction processing. This stands in contrast to Ethereum, where the gas fees directly impact transaction prioritization but often at the cost of higher unpredictability.

When it comes to security during transaction simulation, Aptos mandates that all signatures in a transaction must be set to zero. This security measure ensures no potential misuse of a valid signature during the simulation phase, drawing a line between simulation and real transaction deployment. Plus, in terms of the developer environment, the integration of Aptos CLI's gas profiling feature is notable. It offers a method for developers to understand gas consumption without necessarily resorting to on-chain simulations, simplifying the overall process.

Radix

Radix has built its own custom programming language, called Scrypto (based on Rust), and execution environment, called the Radix Engine, which have been tailored to meet the needs of Web3 and DeFi developers.

Scrypto and Radix Engine were designed and implemented over 2019-2023, following interviews with hundreds of dApp developers. The core principle underpinning this design is the concept that on blockchains and public ledgers, the key thing that matters is the asset, i.e. a token or NFT, and that their safety and correct handling are paramount.

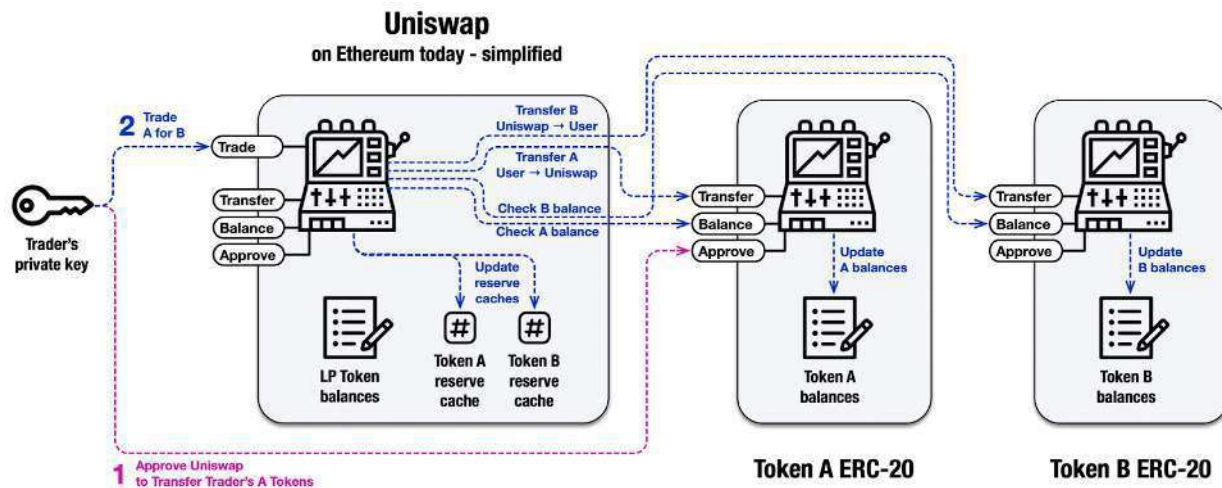
Scrypto and Radix Engine have therefore been designed specifically to include tokens and NFTs as first class core primitives of the programming environment. Radix describes this as “asset-oriented” programming. Here, tokens are not smart contracts, but are distinct objects on the ledger, called “resources”. Similar to how a physical object would behave, resources have properties and behaviors natively understood and enforced by Radix Engine, ensuring that they obey certain properties intrinsic to all assets, such as they shouldn’t be spent twice, transactions should net to zero, and that they must physically live somewhere, in containers called vaults for tokens at rest, and buckets while tokens are on the move. Tokens and NFTs are therefore resources that sit inside smart contract accounts or components, and they get passed between those accounts. Any time an asset gets passed between an account, Radix Engine is guaranteeing the accounting behind the transaction - ensuring that the correct assets get passed and they don’t go missing or double counted.

This brings a number of advantages. First, standard things that developers need time and again, such as minting a new token, defining the conditions under which a token can be sent, whether a token can be recalled, or have its supply changed, how a liquidity pool works, or how permissions are built, are all native features built into the programming environment. This speeds up development time, makes development more intuitive, and makes development more secure, as Scrypto developers can draw upon this suite of native tools and configure them to do what they need to do, knowing that these features will work out of the box. It’s the difference between having a predefined set of tools available to support you, as opposed to having to build each tool yourself. The Ethereum or Solana developer experience instead requires developers to either build equivalent functionality themselves, or copy and paste pre-existing code from online libraries, such as OpenZeppelin, but this can also create security vulnerabilities if any modifications need to be made.

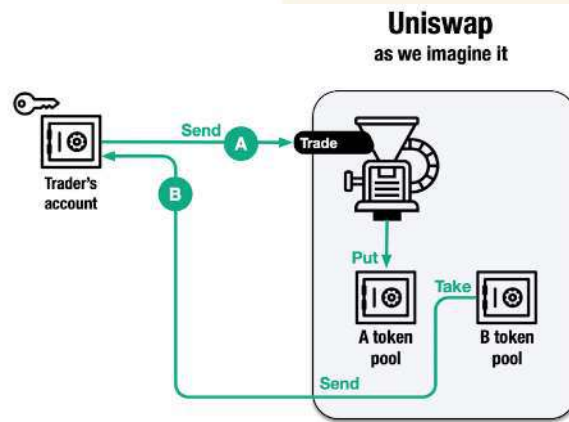
The State of Web3 User and Developer Experience

The aim of providing this asset-oriented programming environment is to lower the learning curve and time it takes for developers to build Web3 and DeFi dApps, reduce the likelihood for developers to make mistakes (as asset logic and transactional accounting is handled by the engine, and other tools to speed up developer productivity, such as permissioning, are available off the shelf), and improve security, as the engine is handling assets and permissions instead of a developer's custom-built smart contract.

In Scrypto, business logic and asset logic are segregated. The business logic is still Turing complete and allows developers the full range of functionality to build any dApp they want, but they can draw upon the suite of asset-oriented tools when they need. This is similar to how game engines, such as Unreal Engine, provide game developers native tooling for graphics and physics and libraries of in-game elements, which still allows the game developer to build any game they want, but the tooling significantly reduces their learning curve, time required, and improves the reliability of games.



How Uniswap works on Ethereum - passing complex chains of messages to downstream smart contracts to update their internal balances, including the need for “approve” transactions which introduces security vulnerabilities. [Source](#)



How Uniswap would be built on Radix - “physically” sending tokens to the DEX, and it physically sends other tokens back. [Source](#)

The added benefit of having assets being natively understood by the execution environment is that Radix transactions are not just a signed hash to a smart contract, as with Ethereum, but are written in human-readable language, defining how tokens move between accounts, and which smart contract calls to make.

```
CALL_METHOD
  Address("<my main account address>")
  "withdraw"
  Address("<USD address>")
  Decimal("20");

TAKE_FROM_WORKTOP_BY_AMOUNT
  Decimal("20")
  Address("<USD address>")
  Bucket("my USD");

CALL_METHOD
  Address("<exchange address>")
  "trade"
  Bucket("my USD");

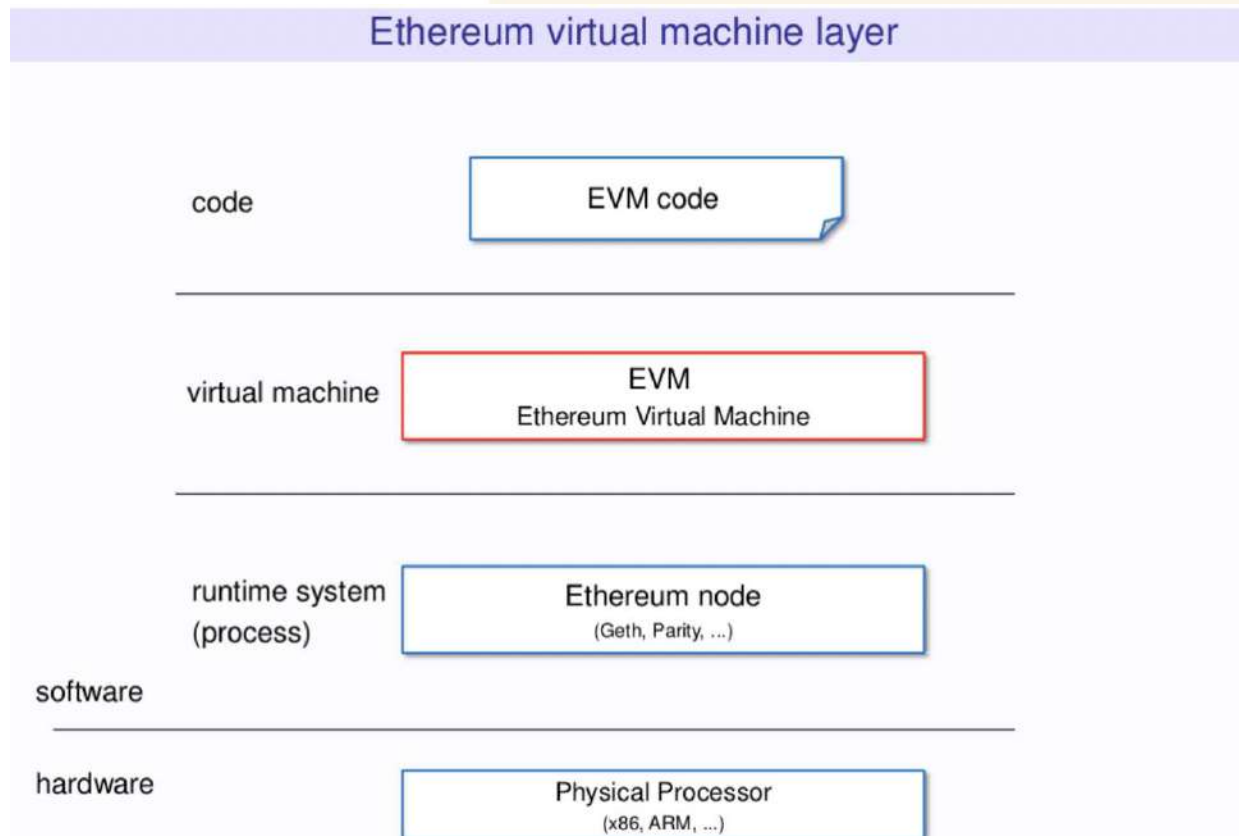
CALL_METHOD
  Address("<my main account address>")
  "deposit_batch"
  Expression("ENTIRE_WORKTOP");
```

An actual transaction manifest. [Source](#)

Language Adoption and Usability

Solidity was crafted with a vision to bridge the gap between traditional programming paradigms and the nascent realm of blockchain. The design of Solidity was influenced by ECMAScript (JavaScript), allowing developers to transition with relative ease. Solidity thrives on static typing, demanding explicit declaration of variable types, a departure from contemporary languages. This nuance enables early error detection at the compile-time, bolstering code integrity, but as we have seen with billions of dollars of DeFi hacks, this alone has not been enough to foster a secure Web3 experience.

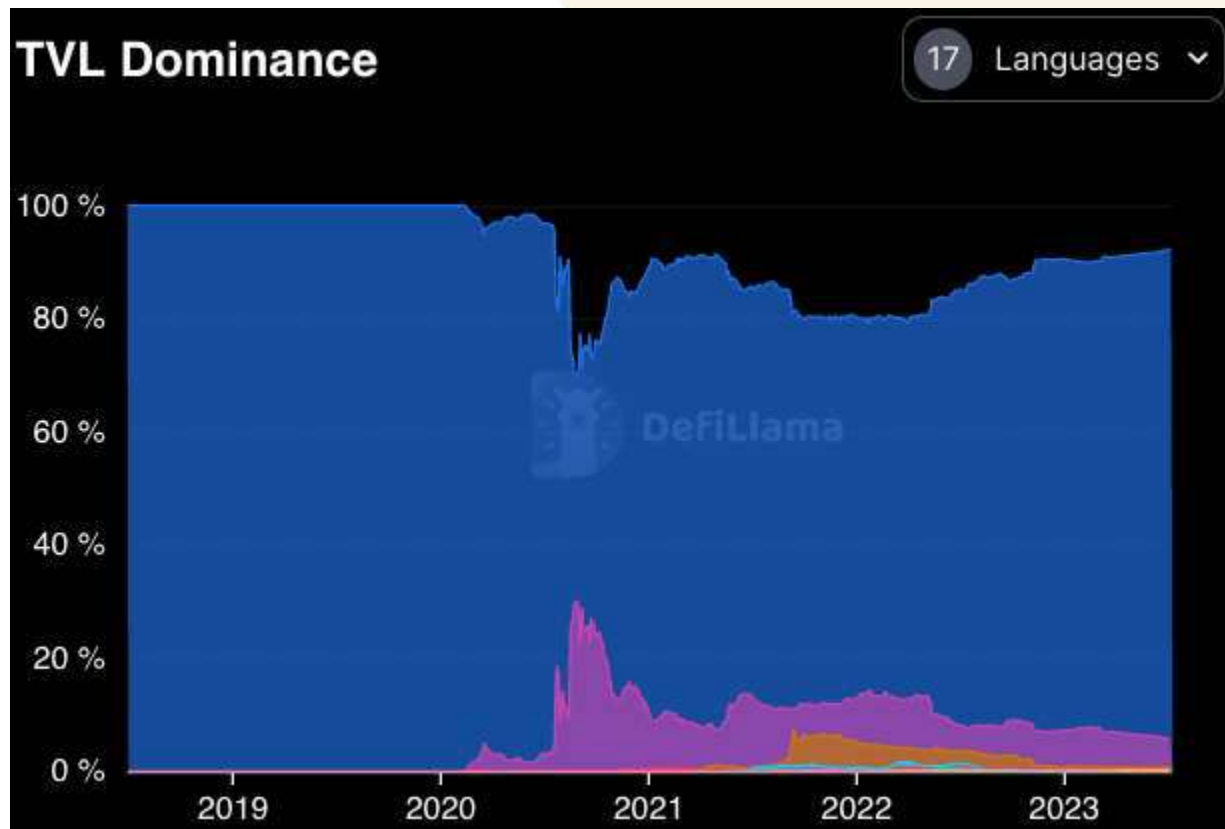
Central to Solidity's ideology is the concept of 'contracts.' In the Solidity domain, this term, reminiscent of age-old legal documents, refers to structured blocks of code. These blocks range from simple constructs, like tokens, to more elaborate structures characteristic of extensive decentralized applications. Solidity also refines the developer experience by introducing mechanisms such as code inheritance. This feature enables foundational contracts to grow organically, extending their functionalities through derivative contracts.



Source

Though Solidity is not without its challenges, its syntactic resemblance to JavaScript has allowed new developers to easily familiarize themselves with the language, and whilst it may be easy to build an initial smart contract, JavaScript was not designed for the safe handling financial applications handling billions of dollars, resulting in Solidity being difficult to build complex and secure applications in.

As an illustration of its dominance, Solidity has carved itself a niche, becoming the primary language for scripting smart contracts. Its open-source nature on Ethereum equips developers to build atop these contracts, evolving them into entirely distinct decentralized applications. Statistics indicate that contracts written in Solidity govern over 90% of DeFi's Total Value Locked (TVL).



Solidity contracts hold 90%+ of DeFi TVL. [Source](#)

The Ethereum ecosystem's depth extends beyond Solidity. It has been the genesis for several globally recognized standards, including:

ERC20/ERC721:

Ethereum's token benchmarks

Ethereum's JSON-RPC Client API:

A blueprint for communicating seamlessly with Ethereum nodes

Ethers.js:

A foundational web library tailored for Ethereum

Pioneering Cryptographic Practices:

including the Keccak256 hash algorithm and ECDSA signatures applied over Secp256k1

While it's feasible to architect an EVM-compatible chain devoid of these standards, adherence amplifies developmental efficacy and user experience. This standardization fosters composability, equipping developers to leverage various benefits.

Examples include Ethereum's Social Graph, facilitating shared metadata across EVM chains, and interoperability elements, like cross-chain governance. Furthermore, toolkits like OpenZeppelin, Hardhat, and Foundry, along with infrastructural tools such as Gnosis Safe, WalletConnect, Zerion, Metamask, and Etherscan, have cemented Ethereum's reputation, with 8 out of the top 20 blockchain ecosystems rooted in EVM.

Utility in the Ethereum Ecosystem

Thanks to Solidity's "first-mover advantage" compared to newer chains, it has developed certain advantages and offers more developer support tools than other blockchains. Backed by a reservoir of comprehensive documentation and the industry's largest developer community, Solidity is the current de facto smart contracting language. Tools like Remix, an online integrated development environment, and Truffle, offering frameworks to ease the Solidity development lifecycle further, have become staples in the developer tool kit. Furthermore, Ethereum's growing prominence paved the way for a deluge of educational content centered on Solidity, creating a positive feedback loop for the developer community.

Limitations

However, amid its accolades, Solidity faces its own set of tribulations. Despite its widespread adoption, Solidity has been marred by security breaches, with the infamous DAO attack of 2016 serving as a stark reminder. Although the entire space (and code) has come a long way since then, the DeFi sector within the EVM ecosystem continues to witness reentrancy attacks. In fact, re-entrancy is a core enabling feature that many EVM dApps require in order to work, and so re-entrancy exploits will continue to occur. These persistent threats underscore the vulnerabilities that lurk within Solidity and the EVM.

The root cause of these issues stems from the platform's lack of asset standardization and guarantee on how tokens and NFTs behave, and complexity thrust upon developers to learn the specifics of how Solidity and the EVM work. EVM developers are required to build much of this complexity themselves from scratch as they can't re-use common logic, and thus even the slightest mistake made can have disastrous results.

The result is that to build a complex and secure dApp, Solidity often presents a steep learning curve, with the need to build things like permissions to each smart contract from scratch - coding which addresses have access to which methods directly within the smart contract. Lastly, while the mathematical veracity of contracts is paramount, Solidity's innate toolset falls short in offering rigorous formal verification, though external tools have emerged to bridge this gap.

Solidity will continue to see many developers flock to it despite these issues, due to the EVM's dominance, but more robust programming environments are needed if Web3 and DeFi are to gain the trust of mainstream users, and institutions, for their day to day transactions.

Rust Language on Solana

Rust has captured significant attention within the developer community for its blend of performance, concurrency, and safety. Originating from Mozilla, it was designed to deliver software that is not only fast but also immune to a vast array of bugs which plague other languages. Rust's ownership system, along with its borrow checker, ensures memory safety without necessitating a garbage collector. This results in an inherently safer language while still achieving the performance of languages like C and C++.

Unlike Ethereum and other blockchain platforms that have coalesced around the Ethereum Virtual Machine (EVM) and its associated languages, Solana has charted a distinct trajectory. Eschewing EVM compatibility, Solana harnesses the Low-Level Virtual Machine (LLVM). The LLVM, a pivotal innovation in the programming world, is a collection of modular and reusable compiler and toolchain technologies. It acts as an intermediate representation, bridging the gap between high-level code and machine code, allowing for optimal performance across various hardware architectures.

Within Solana's architecture, Rust plays a pivotal role. Instead of conventional "smart contracts", Solana employs the concept of "programs." These programs, the equivalents of Ethereum's smart contracts, are predominantly written in Rust, though C and C++ are also supported.

Solana's choice to utilize Rust as its primary programming language is deeply rooted in its overarching objectives and the distinct advantages that Rust brings to the table. One of the most compelling attributes of Rust is its prowess in performance and concurrency. The language is engineered with efficient memory management and zero-cost abstractions, ensuring that any code executed is optimized for speed. This complements Solana's emphasis on scalability, a vital attribute in the blockchain sphere, where transaction speed and throughput are paramount.

Rust, by nature, presents a steeper learning curve compared to some of its counterparts, like Solidity. This complexity is a sieve, naturally appealing to more experienced and professional developers. Solana Labs has been transparent about its intention here: cultivating an ecosystem enriched by innovative, high-quality projects. This strategy is in stark contrast to other platforms that sometimes grapple with a deluge of derivative projects.



Solana contracts deployed over the last 3 years. [Source](#)

However, Rust on Solana, like Ethereum, does not provide developers with many of the native tooling to reduce hacks and exploits and as a result, improve developer productivity. Tokens on Solana are still smart contracts, which can have mistakes and thus result in users' loss of funds. Solana still relies on re-entrancy, which introduces vulnerabilities.

Like Ethereum, much of the logic around how assets behave and how transactions get accounted for is still built by each smart contract developer, meaning that Solana developers are still required to spend significant amounts of time validating that their contracts are correctly implemented, and do not open up security vulnerabilities.








Case in point, the root cause behind the \$48m Cashio hack on Solana was because a smart contract was fooled into believing that a fake asset had been deposited. [Source](#), [Source](#)




Move Programming Language on Aptos

With the blockchain economy facing many security breaches from smart contract loopholes, persistent security concerns have led developers to search for more appealing designs and programming languages. For this reason, Aptos has chosen to use the Move programming language. Move is also used by other projects, including:

- Sui Network
- Parachain
- Celo

NEW LAYER 1 BLOCKCHAINS

Blockchain	Type	Consensus	TPS	Programming	Virtual Machine	Node Requirement
 Aptos	Monolithic	DiemBFTv4	160,000	Move	Move VM	CPU: 8 cores RAM: 32GB Storage: 1TB
 Sui	Monolithic	Narwhal & Bullshark	120,000	SuiMove	Move VM	CPU: 2 cores RAM: 8GB Storage: 50GB
 Fuel	Roll-up	n/a	n/a	Sway	FuelVM	n/a
 Shardeum	Monolithic	PoQ + PoS	n/a	Solidity & Vyper	EVM-Compatible	n/a
 Quai	Monolithic	PoW 2.0	7,500	n/a	EVM-Compatible	CPU: 4 cores RAM: 16GB Storage: 100GB
 Lina	Monolithic	n/a	n/a	n/a	n/a	n/a
 Sei	App Chain	Tendermint	18,000	Cosmos SDK	CosmWasm	CPU: 8 cores RAM: 32GB Storage: 1TB

 Updated: Oct 27th, 2022   @Coin98Analytics

Move is a new smart contract programming language that emphasizes safety and flexibility for developers. In Move, re-entrancy is not required, and thus it's disabled, solving for many types of hack and exploit. Furthermore Move integrates formal verification within its development process, providing an advantage to projects such as Aptos, as it helps to ensure the security of the code being used within the earliest stages of product development. A core function of the language, Move Prover, is the verification tool and assures developers that their code is correct.

Move is memory-safe, expressive, and based on the widely used Rust programming language. This helps to make it a more attractive option for developers as there's crossover knowledge from Rust to Move. This is less applicable to other languages, such as Solidity. Theoretically, this can help Aptos attract more development talent to its ecosystem.

The State of Web3 User and Developer Experience

Furthermore, Move's design facilitates parallel transaction processing. On the other hand, EVM's sequential transaction processing, a measure to counteract double-spending and reentrancy threats, often leads to congestion.

Solidity holds the upper hand in terms of smart contract upgradability, as Solidity provides developers with the latitude to upgrade protocol and contract codes, an option that Move purposely curtails.

Comparing Aptos Move/MoveVM & Other Languages and VMs

	Move / MoveVM (Aptos)	Solidity / EVM	Rust / WASM
Potential Double-Spending	Prevented with Move Prover's formal verification	Possible, but prevented through the consensus algorithm	Possible, but prevented through the consensus algorithm
Parallel Execution	Enabled by Block-STM & distributed execution	Impossible	Only when designating access to all data
Token Standard	Digital Asset Standard from August, 2023 (Separate smart contract not needed)	Ethereum Request for Comment (Separate smart contract needed)	SPL(Solana Programmable Ledger) (Separate smart contract not needed)
Smart Contract Simplicity	High	Low	Low
Function Call	Static dispatch	Dynamic dispatch	Static dispatch

Source: Aptos, Xangle

Xangle

Move brings many safety innovations to the table, while at the same time improving developer productivity, with its use of objects (as opposed to the account-based model for Ethereum and Solana). With the Aptos Coin Standard, Aptos works similarly to Solana. It uses a common set of code (APC) for different types of tokens like digital currencies, collectible items, and semi-fungible tokens (which are like a mix of currencies and collectibles). Instead of writing new code for each token, developers run a function in the existing code with the details about their token.

However, ultimately these tokens are not native to the underlying blockchain, and thus developers are still responsible for ultimately ensuring that their tokens behave correctly, get accounted for correctly, and are validated correctly. This ultimately slows the development process as developers are responsible for building this code, validating it, and auditing it. Issues in the implementation of tokens are still possible, and from a UX perspective still raises the possibility for malicious tokens.

Scripto Smart Contract Language on Radix

Scripto retains most of Rust's features while incorporating specific functions and syntax for the Radix Engine. However, Scripto is more than just Rust operating on a public Distributed Ledger Technology (DLT) network. It's an asset-oriented language that allows Rust-style logic to interact with data and assets natively, making it a perfect match for building decentralized finance (DeFi) applications. As mentioned in the introduction to Scripto and Radix Engine, together they provide developers first class native tools and features to speed up the development process, reduce the chances for bugs and hacks, and to onboard new developers more quickly with a more intuitive “physical” approach to representing assets. Like Move, re-entrancy is not required in Scripto.

A few of these features include:

Resources, Buckets, and Vaults: In Scripto, tokens and non-fungible tokens (NFTs) are not smart contracts or components but are treated as resources. To create a new resource, like a token or an NFT, you can utilize a built-in Scripto function to specify the parameters of that resource. The newly created resources are immediately stored in a temporary container called a bucket, which eventually needs to be transferred into a more permanent container known as a vault for further storage. Vaults have special logic in place that validates whether tokens received are valid.

Blueprints, Components and Methods: Radix has two types of smart contract, which begins life as a “blueprint” which can be instantiated into one or many "components."

A blueprint is like a template for smart contract functionality. They don't hold any state - they are purely a definition of what functionality a smart contract should do.

Once a blueprint has been deployed, components may then be instantiated from that blueprint. The Scrypto code is present in the blueprint; but the component's actual state - its data and resources - belongs exclusively to that individual component, not the blueprint. Once a component is instantiated from a blueprint, it becomes active for use on the network by users or other components. Blueprints foster the reusability of Scrypto code and offer developers vast flexibility in performing various setup and configuration actions.

As an example, you may have a blueprint for a DEX liquidity pool, that defines how swaps between two assets may be conducted. And then a developer may instantiate that blueprint into a component for an XRD:USDT pair, or another component for an XRD:ETH pair, etc. This way you can have greater assurance over how each component will behave, as you are directly instantiating it on ledger (as opposed to copying and pasting and then directly editing smart contract code on Ethereum and other platforms).

One special feature of components is that they have special methods that allow them to directly accept resources, i.e. a token. This new approach provides a much more intuitive way of handling assets, as components are physical containers for tokens, and provides a far safer way to transfer assets, as the execution environment is handling the transaction accounting.

Permissions: One pain point that Scrypto solves elegantly is in the use of “badges” for permissioning. A badge is like any other resource, i.e. a token or NFT, but it is used for permissions. So for example, a user could hold a badge NFT that gives them special discounts on trading fees for a DEX. A developer could hold a special administrator badge that provides them privileged access to a smart contract, e.g. to update an important parameter. Furthermore Radix makes use of native role based access control, which maps access to specific methods to roles, simplifying the process of defining which roles can do what when it comes to accessing a smart contract component. This is in stark contrast to how permissions are set up on the other three platforms in this report, where developers are required to build the low level systems for how permissioning works themselves, with mistakes in these low level systems sometimes resulting in hacks, exploits, and loss of developer productivity.

Transaction Manifest: As previously alluded, transactions on Radix take advantage of the unique capabilities of the Transaction Manifest, which is how transactions on Radix are written. Unlike Ethereum, Solana or Aptos, where a transaction is a call to a single smart contract, which then executes calls downstream, on Radix, the Transaction Manifest is able to interface with each smart contract component in turn, specifying how tokens move between accounts/components, and specifying which component methods to call.

All of these actions must execute successfully, or the entire transaction is aborted by Radix Engine. This allows Radix developers to build far more composable applications, as the standardization of how the Transaction Manifest defines how tokens move reduces complexity. It also provides Radix developers the ability to build applications purely in the front end of a website that can interact with multiple smart contracts, specifying conditions that must be met, such as this transaction must return at least X tokens, otherwise the entire transaction fails safely, in an “atomic” manner.

Royalties: Radix has also implemented an automated Developer Royalties System, that pays developers on-chain when their smart contract code gets used. This creates an additional and recurring incentive mechanism for developers to create core blueprints and components that get used.

Overall, Scrypto provides a syntax that has been specifically designed to make it easy for developers to be able to create and manage tokens and NFTs in an intuitive and safe way, clearly separating the underlying logic that governs assets from the business logic that defines how their dApp should function. Features such as badges and native role based access control for permissioning ensure that access rights to smart contracts are more intuitive, easily set up, and safely configured. The Transaction Manifest provides developers a powerful way to build certain types of dApp purely in the front end of websites, without needing to deploy smart contracts just to execute complex transactions.

04

Developer Tooling and Ecosystem

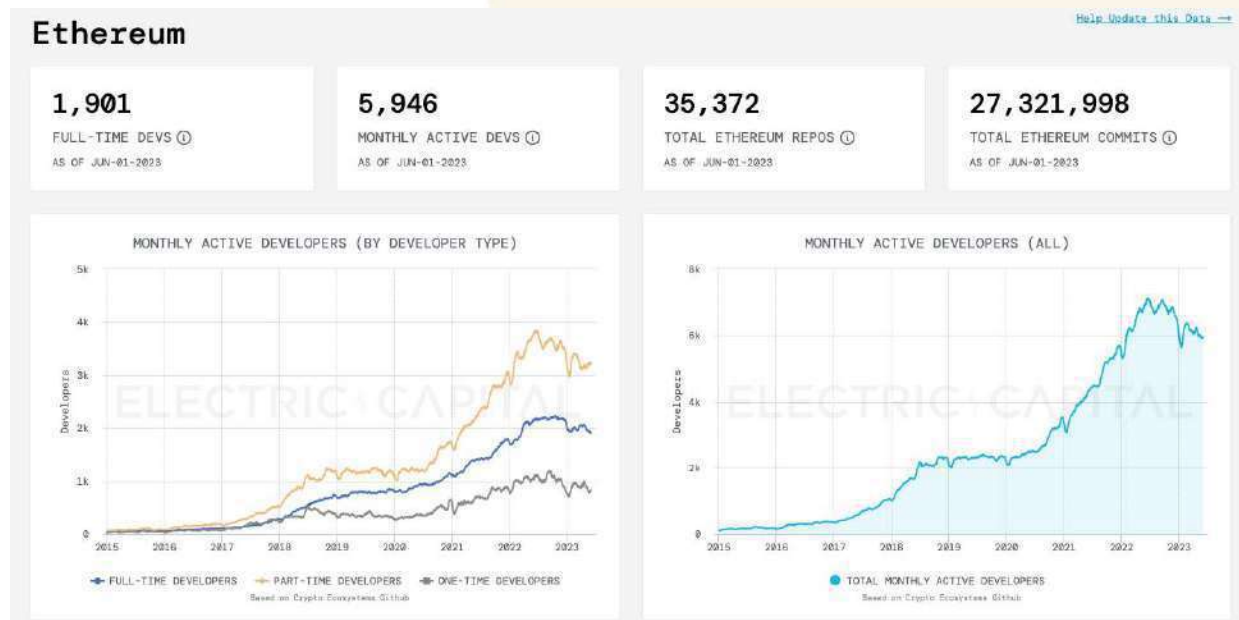
Ethereum

Ethereum's prominence in the blockchain landscape is underpinned by its expansive developer ecosystem, tailored to accommodate the diverse needs of decentralized application (DApp) development. The platform's development frameworks offer comprehensive infrastructures, making it accessible for developers of various backgrounds, whether from a JavaScript or Python milieu. This versatility ensures a smoother and more efficient DApp development process, inviting novice and veteran developers to innovate on the platform.

In addition to these frameworks, Ethereum boasts a suite of tools geared towards seamless compilation and effective dependency management. Developers can utilize preprocessors, flattening tools, and specialized libraries to ensure smart contracts are efficiently compiled, and their interdependencies are seamlessly integrated. Given the criticality of security in a decentralized environment, Ethereum's toolkit also emphasizes robust security assurance through bytecode scrutinizers, static analysis, and symbolic execution analyzers. Such tools are invaluable in ensuring the smart contracts deployed are resilient against vulnerabilities.

Recognizing the dynamic nature of blockchain-based applications, Ethereum also offers real-time monitoring and analytics tools. These solutions provide granular insights, allowing developers to identify and address any operational challenges quickly.

Standardization remains a pivotal aspect of Ethereum's developer ecosystem, with the ERC (Ethereum Request for Comments) repository playing a vital role. These standards, which span diverse facets from token specifications to blockchain identities, foster consistency and interoperability across applications. The rich tapestry of libraries in the ecosystem offers developers pre-built modules and smart contracts, optimizing development time and ensuring adherence to best practices.



Source

Solana

The emergence of Solana as a viable alternative is noteworthy due to its design, transaction speeds, and less saturated developer space. One of Solana's significant moves has been partnering with Alchemy. This collaboration aims to enhance the development experience on the Solana blockchain, with Alchemy offering full support and resources to developers working with Solana.

Key components in Solana's development toolset include:

Solana Tool Suite:

An essential set of tools for any developer looking to build on Solana. It's a foundational step, setting up the environment and enabling interaction with various other tools like Anchor.

Rust:

Given that Solana is built on Rust, understanding this language is crucial. Rust is known for its speed and stability and is the primary language for Solana-based development.

Anchor:

This is a development framework tailored for Solana. It offers pre-written code structures and security protocols, simplifying the process of creating and deploying applications on the Solana blockchain.

Additionally, other tools facilitate specialized functionalities:

Web3.js:

A JavaScript API that allows developers to interact with the Solana blockchain

SPL-Token:

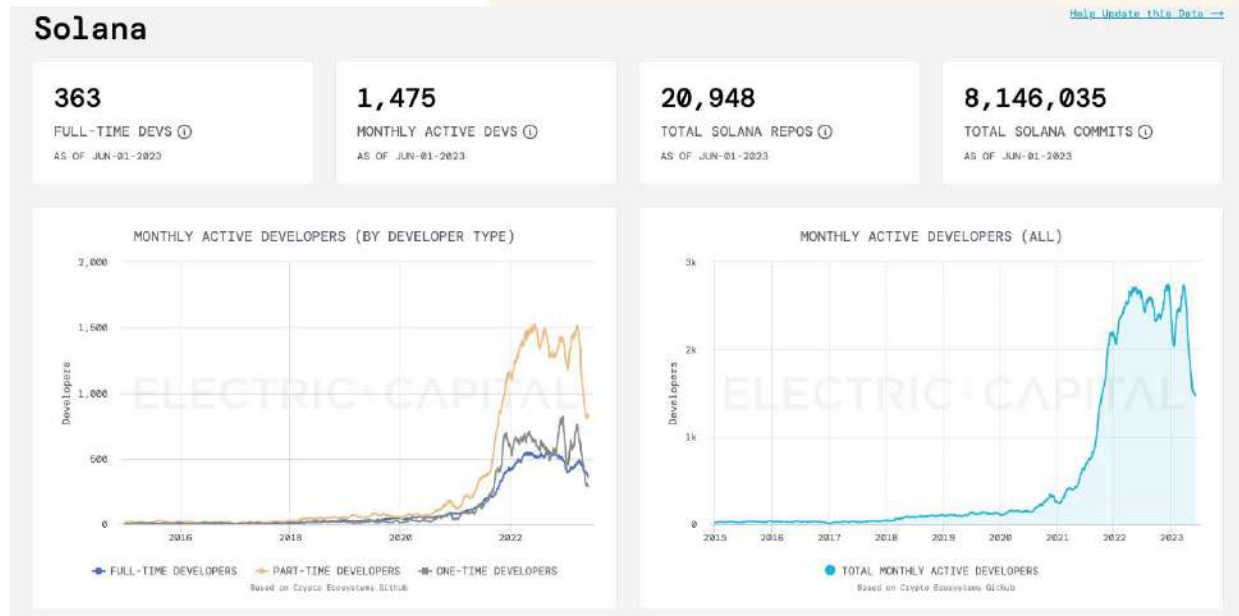
A package enabling developers to mint, transfer, and manage tokens on the Solana platform

Wallet-Adapter:

This facilitates the integration of Solana-based wallets into decentralized applications

Support for developers is abundant in the Solana ecosystem. Platforms like the Solana and Anchor Discords provide community-based assistance, while resources such as Solana's tutorials and hackathons offer practical learning experiences.

The State of Web3 User and Developer Experience



Source

Aptos

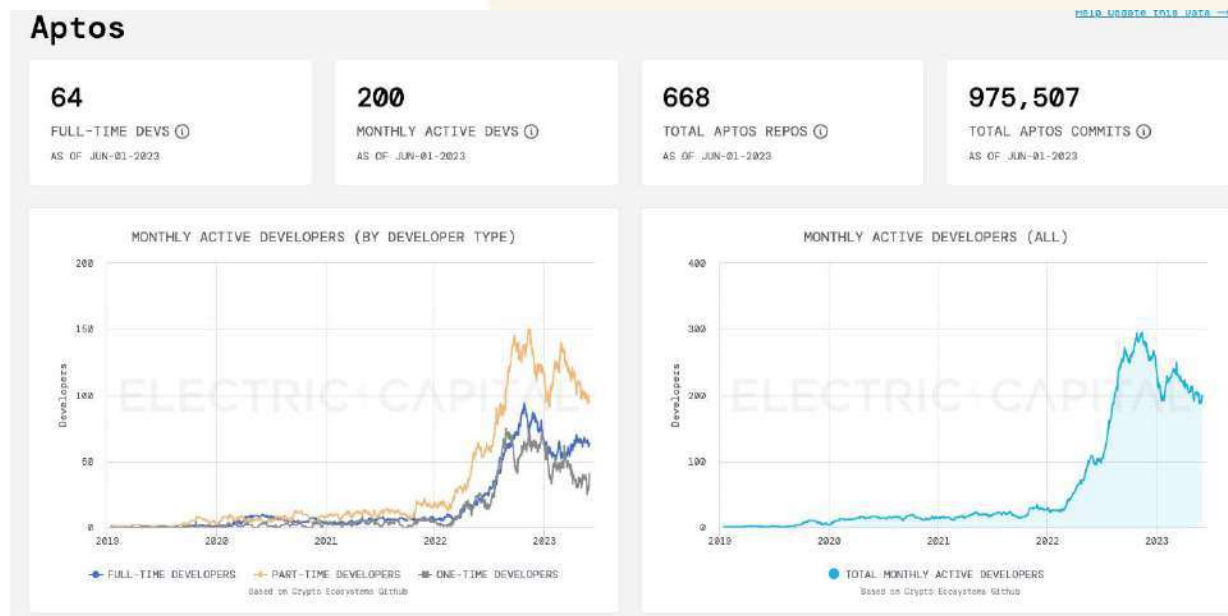
Aptos has meticulously cultivated an expansive developer ecosystem, mirroring a tree's vast roots. It provides a fertile ground for projects to grow, thanks to a comprehensive set of developer tools tailored for versatility and innovation.

Understanding the importance of an efficient Integrated Development Environment (IDE), Aptos has seamlessly integrated with renowned platforms like VSCode and IntelliJ IDEA. These platforms not only offer familiarity but also optimize the workflow for developers. Complementing this is Aptos's SDK Development Framework, which presents a rich tapestry of options. The Move Playground, for instance, offers a sandboxed space for innovative experimentation. In terms of language support, Aptos proudly offers SDKs in popular languages such as Typescript, Python, and Rust, ensuring wide accessibility.

For a deeper dive into the blockchain, the Blockchain Access Browser introduces tools like Aptos Explorer, TraceMove, and Aptoscan. These are crucial for visualizing intricate on-chain activities and interactions. In a decentralized ecosystem, data accessibility and retrievability are paramount. Aptos addresses this by integrating robust Node and Archive Services, including NodeReal, Ankr, and Moralis.

The Aptos Developer Documentation further emphasizes their dedication to a thriving developer community. This comprehensive guide assists developers at every skill level, covering a gamut of topics from basic setups to the nuances of the Move programming language. Aptos champions community engagement through platforms like Discord, Stack Overflow, and Twitter, fostering collaboration and open-source development.

The State of Web3 User and Developer Experience



Source

Radix

Radix has recognized the immense potential of decentralized finance (DeFi) and aims to make it accessible and scalable. It has created a seamless synergy between its primary smart contract language, Scrypto, and a suite of robust toolkits, laying a comprehensive foundation for developers.

The Radix dApp Toolkit exemplifies the platform's commitment to user-centric design. It bridges the user experience between the Radix Wallet and decentralized applications (dApps). The integration of the Wallet SDK and Gateway SDK within this toolkit facilitates a consistent and intuitive interaction model. With the Wallet SDK, dApps are able to maintain a bi-directional communication channel with the Radix Wallet, ensuring users can fluidly transact and interact with dApps while retaining the security and integrity of their holdings.

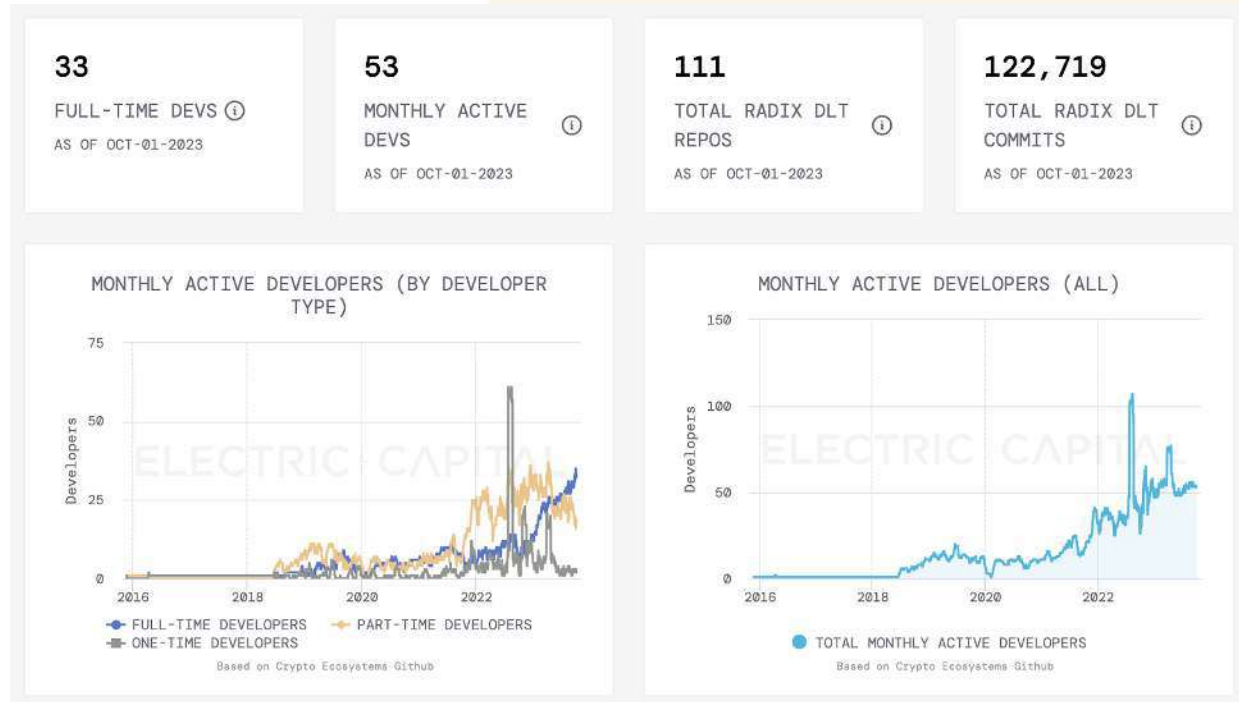
In addition to front-end utilities, Radix's Radix Engine Toolkit (RET) stands out as a pivotal asset for back-end development. Crafted in the Rust programming language, it streamlines complex tasks such as transaction manifest creation and SBOR encoding/decoding. What's noteworthy is RET's adoption of UniFFI, broadening its compatibility to encompass languages like C#, Kotlin, Swift, and Python. This flexibility showcases Radix's vision of fostering a diversified developer ecosystem, recognizing that innovation often stems from a varied technological background.

One intriguing innovation is the introduction of ROLA (Radix Off-Ledger Authentication). Devising a system that operates off-ledger while harnessing on-ledger data is a testament to Radix's ability to blend decentralized and traditional technologies. ROLA allows dApps to authenticate their users against on-chain identities, streamlining dApp creation. It functions similarly to the new Web2 passkeys marking "the beginning of the end of the password" - but by using Radix's on-ledger recovery, it doesn't require trusting a centralized service with key backup and syncing. This introduces a dimension of trust and security that stands apart in the blockchain landscape.

It's also worth noting Radix's commitment to fostering a strong, engaged developer community. The Radix Developer Program, paired with active engagement on platforms like Discord and Telegram, exemplifies this. It's not merely about providing tools; it's about facilitating knowledge exchange, ideation, and collaborative growth. The Technical Docs empower developers by elucidating the intricate features and specifications of the platform.

The Babylon release itself was a monumental step, paving the way for the deployment of Scrypto-based smart contracts and heralding an era of enhanced DeFi capabilities on Radix. However, Radix's forward-looking approach doesn't stop there. The subsequent Xi'an release promises even greater strides, with the complete implementation of the Cerberus consensus protocol, a potential game-changer in the quest for boundless scalability.

The State of Web3 User and Developer Experience



Source

Summary

For DX, Ethereum's ecosystem is comprehensive, catering to a diverse range of developers from various backgrounds. Aptos offers a multi-language supportive environment coupled with renowned Integrated Development Environment (IDE) platforms, making it a fertile ground for developer innovation. Solana's collaboration with industry players like Alchemy indicates its ambition to provide top-tier tooling capabilities. However, Radix sets itself apart with native tooling for assets, transactions and permissions that greatly simplifies and accelerates the development process, making it easier to build secure dApps in a fraction of the time.

Disclaimer:

<https://www.cryptoeq.io/disclaimer>

The content in this PDF is for informational purposes only and is not intended to provide tax, legal, accounting, financial, investment, or professional advice. The document is not an investment advisor nor any sort of financial advisor, nor is it undertaking to provide investment advice, act as an advisor to, or give advice in a fiduciary capacity to any reader or individual with respect to the information presented herein. You should not construe the information provided in this PDF as investment advice, financial advice, trading advice, or any other advice. The information contained in this document is general information, and personalized information is not provided. The document does not give individual or personalized recommendations that any cryptocurrency be bought, sold, or held by any readers. None of the information or content in this document is (or is intended to be) a solicitation or offering of any security to any person or entity.

Investing involves substantial risk, and the document makes no guarantee or promise as to any results that may be obtained using the information from the PDF, either directly or indirectly. Different types of investments involve varying degrees of risk, and there can be no assurance that any specific investment will either be suitable or profitable for any reader's portfolio. Readers should consult their financial advisor or investment advisor and conduct their own research and due diligence before making any investment decisions.

The information provided in this PDF (including any separate documents that may be referenced within this PDF) is not directed at any investor or category of investors and is provided solely as general information. The document provides all information "as is" and without warranties of any kind. The publisher will strive to provide accurate information, but it is not responsible for any inaccurate or missing information. By accessing the information in this PDF, you expressly acknowledge and accept all risks and liability associated with the use of the information provided. To the maximum extent permitted by law, the publisher disclaims any and all liability in the event any information, commentary, analysis, opinions, or other information provided in the PDF prove to be inaccurate, incomplete, or unreliable or result in any investment or other losses.

This publication is sponsored. CryptoEQ neither endorses nor assumes responsibility for any content, accuracy, quality, advertising, products, or other materials presented on this page. Readers are advised to conduct their own research before engaging in any activities related to the company. CryptoEQ disclaims any direct or indirect liability for any damage or loss arising from or allegedly caused by the use of or reliance on any content, goods, or services mentioned in the PDF.



About CryptoEQ

Crypto is complex. We make it simple.

CryptoEQ is an independent cryptocurrency analysis and rating agency that provides unbiased, objective, and transparent research you can trust. We help people navigate their investment journey and trading decisions. Our proprietary algorithms, exhaustive research and helpful community are key to our success as we follow strict principles and ethics to deliver honest information. We actively seek to identify scams and low quality nefarious projects relieving you of that burden.

