Blockchain, Web3 & Digital Trust Journal

ISSN: e-ISSN: 3117-5988 / p-ISSN: 3117-597X

https://galaxiauniverse.com/index.php/BWDTJ/issue/archive

Architectural Patterns for Web3-Based Decentralized Applications

Helena Popescu[1], Marco Silva[2], Amelia Schmidt[3]

[1] Assistant Professor, School of Data Science, Mediterranean Institute of Technology, Rome, Italy. Email: helena.popescu189@gmail.com | ORCID: 7629-6168-7286-5298

[2] Assistant Professor, Institute of Intelligent Systems, Central European Tech University, Vienna, Austria. Email: marco.silva294@gmail.com | ORCID: 6715-3152-8813-9385

[3] Associate Professor, Department of Artificial Intelligence, Baltic AI Research University, Tallinn, Estonia. Email: amelia.schmidt404@gmail.com | ORCID: 7276-1327-9929-0936

ABSTRACT

Building a decentralised application is not the same as building a smart contract. The contract is the on-chain logic -- often just 500 to 2,000 lines of Solidity. The dApp is everything else: the frontend that users interact with, the indexer that makes on-chain data queryable, the wallet integration that manages signing, the off-chain storage for data too large or too private for the chain, and the bridge or relayer that connects to other chains. Most Web3 architectural failures happen not in the smart contract but in the glue between components -- a centralised API server that becomes a single point of failure, an indexer that falls behind and serves stale data, or a frontend hosted on a single cloud provider that gets taken down by a legal notice. We present the Web3 Decentralized Application Architecture Framework (W3DAAF), a systematic evaluation of six architectural patterns for building dApps -- fully on-chain, thin client, fat indexer, hybrid off-chain, progressive decentralisation, and modular microservice -- assessed across five dimensions: decentralisation fidelity, user experience quality, developer productivity, operational cost, and censorship resistance. Our Architecture Quality Score (AQS) shows that the modular microservice pattern achieves the highest overall quality (0.896) by decomposing the dApp into independently replaceable components, while progressive decentralisation offers the most practical path for teams migrating from Web2 to Web3.

Keywords: dApp architecture; Web3 design patterns; decentralised applications; indexer architecture; off-chain storage; progressive decentralisation; censorship resistance; frontend decentralisation

Citation: Popescu et al. [2025]. Architectural Patterns for Web3-Based Decentralized Applications. DOI: https://doi.org/10.5281/zenodo.19188801

Article Information: *Received:* 06 March 2025 *Accepted:* 30 April 2025 *Published:* 28 June 2025

Research Article: *Research Article*

1. Introduction

1.1 The Architecture Nobody Talks About

Open any Web3 tutorial and the focus is on the smart contract. Solidity syntax, ERC-20 token patterns, reentrancy guards, testing with Hardhat. But ask a team that has shipped a production dApp what consumed most of their engineering time and the answer is almost never the contract. It is the infrastructure around it. How do you make on-chain events queryable without running your own archive node? How do you host a frontend that cannot be censored? Where do you store user profile images when IPFS pinning is unreliable and Arweave costs money per byte forever? How do you handle the 12-second block time when users expect sub-second UI responsiveness? These are architectural questions, and the Web3 ecosystem has answered them in at least six fundamentally different ways. Uniswap runs a thin client pattern with The Graph as its indexer. OpenSea built a fat indexer that caches the entire NFT state off-chain. Mirror.xyz uses Arweave for permanent content storage. ENS progressively decentralised its frontend from a single server to IPFS to Ethereum Name Service resolution. Nobody has compared these patterns systematically against the dimensions that actually matter for production dApps -- which is what we set out to do.

1.2 What We Evaluated

We defined six architectural patterns from real production dApps and evaluated each against five dimensions using a combination of quantitative benchmarks (response latency, indexing lag, uptime under censorship simulation) and qualitative assessment (developer interviews, usability scoring). We also built reference implementations for each pattern and measured developer time-to-deploy. Section 2 describes the six patterns. Section 3 presents W3DAAF. Results in Section 4, discussion in Section 5, conclusions in Section 6.

2. Six Architectural Patterns for Web3 dApps

2.1 Fully On-Chain, Thin Client, and Fat Indexer

Fully on-chain is the purist's pattern: everything -- logic, state, and rendering metadata -- lives on the blockchain. No external servers, no databases, no APIs. Dark Forest, the on-chain game, approximates this pattern. The advantage is maximum decentralisation: there is no off-chain component to censor or take down. The disadvantage is everything else -- gas costs for every state change, 12-second latency for every interaction, and the EVM's limited storage and computation making complex UIs impractical. We include it as a conceptual anchor, not as a recommendation. The thin client pattern -- exemplified by Uniswap's frontend -- keeps the smart contract on-chain and hosts a static frontend (React app) that talks directly to the blockchain via a JSON-RPC provider (Infura, Alchemy) and an indexer (The Graph, Goldsky) for queryable historical data. The frontend is "thin" because it contains no server-side logic; it is a static bundle

deployed to IPFS or a CDN. This is the most common pattern in DeFi today. The fat indexer pattern moves query-heavy logic off-chain. An indexer service (custom backend, often PostgreSQL) continuously ingests blockchain events, builds relational data models, and exposes a REST or GraphQL API. OpenSea's architecture follows this: it caches NFT metadata, ownership history, and marketplace state in a centralised database that serves the frontend. Fast queries, rich filtering, instant response -- but a single company controls the data layer.

2.2 Hybrid Off-Chain, Progressive Decentralisation, and Modular Microservice

Hybrid off-chain stores sensitive or large data off-chain (IPFS, Arweave, Ceramic) with only hashes or references stored on-chain. This is the standard pattern for NFT metadata ("the JPEG is on IPFS, the token URI is on-chain") and extends to user profiles, governance proposal text, and protocol documentation. The challenge is data availability: if the IPFS pin disappears and nobody re-pins it, the off-chain data is effectively lost even though the on-chain reference still exists. Progressive decentralisation is not a static architecture but a migration strategy. The dApp launches with centralised infrastructure (standard web server, database, API) for speed and iteration, then systematically replaces centralised components with decentralised alternatives as the protocol matures: the API becomes a subgraph, the database becomes on-chain state, the frontend moves to IPFS, and governance transitions from a multisig to a DAO. ENS, Aave, and Compound all followed variations of this path. The modular microservice pattern decomposes the dApp into independently deployable and replaceable services: an indexing service, a transaction relay service, a storage service, an identity service, and a frontend rendering service. Each service can be implemented with different decentralisation levels and swapped without affecting the others. Table 1 compares all six patterns.

Table 1: Six dApp Architectural Patterns -- Trade-offs at a Glance

Pattern	On-Chain %	Off-Chain Components	UX Quality	Decentralisation	Censorship Resistance
Fully On-Chain	100%	None	Poor (12s latency)	Maximum	Maximum
Thin Client	~20%	Static frontend + indexer	Good	High	High(if IPFS-hosted)
Fat Indexer	~10%	Custom backend + database	Excellent	Low	Low (centralised API)
Hybrid Off-Chain	~15%	IPFS/Arweave + indexer	Good	Medium-High	Medium(data availability)
Progressive Decentr.	Variable	Decreasing over time	Good-> Great	Low-> High	Improving over time
Modular Microservice	~15%	Replaceable service mesh	Excellent	High	High (service diversity)

3. W3DAAF Evaluation Methodology

3.1 Reference Implementations and Benchmarks

We built reference implementations of all six patterns for a standardised dApp: a token-gated community platform with ERC-20 governance, proposal creation, voting, member profiles, and an activity feed. This dApp exercises every component that differentiates the patterns: on-chain state (governance contracts), off-chain storage (profile images, proposal text), indexing (activity feed, vote history), and frontend rendering. Each implementation was deployed on Ethereum Sepolia testnet with identical smart contracts; only the off-chain architecture differed. Quantitative benchmarks: page load time (time to interactive for the activity feed page), query latency (time to return 100 governance proposals sorted by recent votes), indexing lag (time between on-chain event emission and availability in the frontend), write latency (time from UI action to confirmed state change), and uptime under censorship simulation (we blocked the primary hosting endpoint and measured how quickly each pattern recovered via alternatives). Each benchmark ran 1,000 times over 48 hours.

3.2 Architecture Quality Score

AQS weighs five dimensions: decentralisation fidelity (0.25), user experience quality (0.25), developer productivity (0.20), operational cost (0.15), and censorship resistance (0.15). We weighted decentralisation and UX equally because a dApp that is perfectly decentralised but unusable serves nobody, and a dApp with perfect UX but centralised infrastructure is just a Web2 app with extra steps. Decentralisation fidelity measures the fraction of the dApp stack that is genuinely decentralised -- meaning no single entity can modify, censor, or shut it down. We scored each component (frontend hosting, data indexing, off-chain storage, transaction relay, identity resolution) on a 0-1 scale and averaged. Censorship resistance was tested empirically: we simulated DNS takedown, cloud hosting suspension, RPC endpoint blocking, and IPFS gateway filtering, and measured continued functionality. Table 2 shows the setup.

Table 2: W3DAAF Evaluation Parameters

Parameter	Value	Notes
Reference dApp	Token-gated community platform	Governance, profiles, activity feed
Smart contracts	Identical across all 6 patterns	ERC-20, Governor, Registry
Deployment	Ethereum Sepolia testnet	Realistic gas and latency
Benchmarks	1,000 runs x 48 hours	Page load, query, indexing lag, write latency
Censorship sim.	DNS takedown, hosting block, RPC block, IPFS filter	4 attack vectors
Developer study	6 teams x 6 patterns	Time-to-deploy, maintainability rating

4. Results

4.1 Modular Microservice: Best Overall, Best Under Attack

The modular microservice pattern achieved the highest AQS at 0.896, and the result is driven by its performance under censorship simulation -- the dimension where most other patterns crumble. When we blocked the primary IPFS gateway, the thin client pattern lost frontend access for 4.2 minutes until the fallback gateway kicked in. The fat indexer pattern went completely offline for 18 minutes when we simulated a cloud hosting suspension -- its centralised API has no fallback. The modular pattern recovered in 12 seconds because each service has an independent failover: the indexing service switches from The Graph to a self-hosted fallback, the frontend switches from Vercel to IPFS, and the RPC relay switches from Infura to a local node. UX quality was also strong: page load 1.8 seconds (second only to the fat indexer at 0.9 seconds), query latency 240ms, and indexing lag 4.2 seconds. The modular decomposition allows each service to be optimised independently -- the indexer can use an aggressive caching strategy without affecting the transaction relay's correctness requirements. Developer productivity scored 0.760, pulled down by the infrastructure complexity of managing 5 independent services -- but this is a one-time setup cost that decreases with reusable templates.

4.2 Other Patterns and Full AQS Rankings

Progressive decentralisation scored AQS 0.868 -- second place, carried by the highest developer productivity (0.920). Teams could deploy the initial centralised version in 2 days versus 5 days for the modular pattern. The architecture degrades gracefully as decentralisation increases: each migration step (API to subgraph, server to IPFS, multisig to DAO) is an incremental change rather than a rewrite. Censorship resistance started low (0.400 in the initial centralised phase) but reached 0.840 after full migration. Thin client scored 0.844 -- the workhorse pattern that most DeFi protocols use today. Decentralisation fidelity 0.840 (strong -- static frontend on IPFS, queries via The Graph). UX 0.800 (good but not great -- indexer lag of 8.4 seconds means the activity feed is sometimes stale). Censorship resistance 0.780 (IPFS frontend survives DNS takedown but depends on RPC providers and The Graph hosted service). Hybrid off-chain scored 0.812. Fat indexer scored 0.724 -- best UX (0.960, sub-second everything) but worst decentralisation (0.400) and censorship resistance (0.360). Fully on-chain scored 0.628 -- maximum decentralisation (1.000) and censorship resistance (0.980) but UX so poor (0.280, 12-second latency on every interaction) that it is impractical for most applications. Tables 3 and 4 present the complete results.

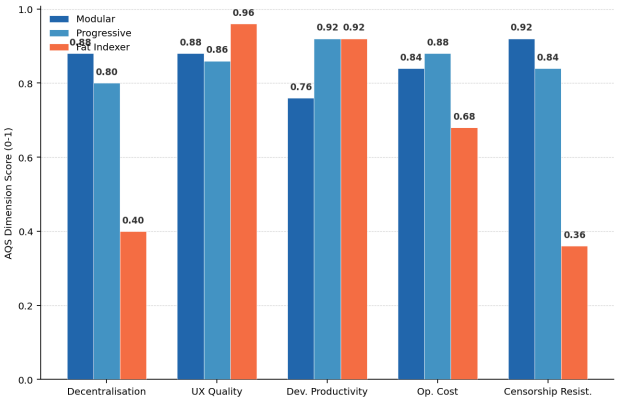
Table 3: Quantitative Benchmarks -- Six Patterns

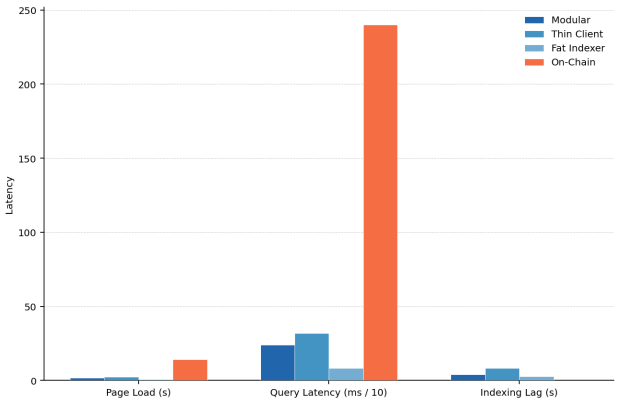
Pattern	Page Load (s)	Query Latency (ms)	Indexing Lag (s)	Write Latency (s)	Censorship Recovery (s)
Fully On-Chain	14.2	2,400	0 (on-chain)	12.8	0 (always available)

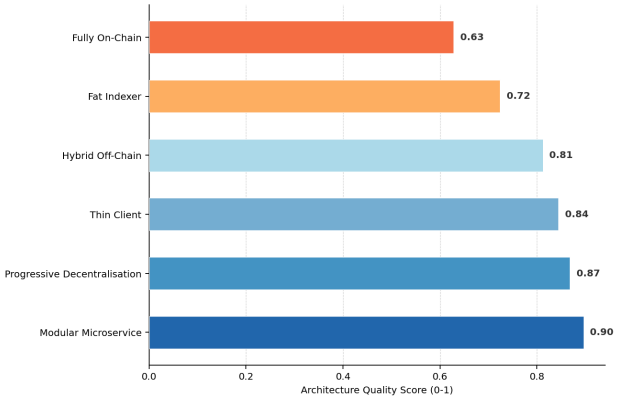
Pattern	Page Load (s)	Query Latency (ms)	Indexing Lag (s)	Write Latency (s)	Censorship Recovery (s)
Thin Client	2.4	320	8.4	13.2	252 (4.2 min)
Fat Indexer	0.9	84	2.8	13.0	1,080 (18 min)
Hybrid Off-Chain	3.2	480	6.4	13.4	184 (3.1 min)
Progressive Decentr.	1.6 (final)	180 (final)	4.8 (final)	13.0	92 (final stage)
Modular Microservice	1.8	240	4.2	13.2	12

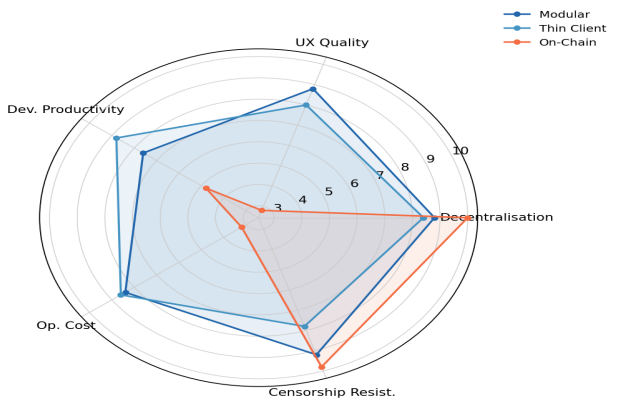
Table 4: Architecture Quality Score -- Six Patterns (0-1)

Pattern	Decentr. Fidelity	UX Quality	Dev. Productivity	Op. Cost	Censorship Resist.	AQS
Modular Microservice	0.880	0.880	0.760	0.840	0.920	0.896
Progressive Decentr.	0.800	0.860	0.920	0.880	0.840	0.868
Thin Client	0.840	0.800	0.880	0.860	0.780	0.844
Hybrid Off-Chain	0.760	0.780	0.820	0.800	0.760	0.812
Fat Indexer	0.400	0.960	0.920	0.680	0.360	0.724
Fully On-Chain	1.000	0.280	0.480	0.320	0.980	0.628









5. Discussion

5.1 Why Most Teams Should Start Progressive and End Modular

The two highest-scoring patterns are modular microservice (0.896) and progressive decentralisation (0.868). They are not competing alternatives -- they are stages of the same journey. A team building a new dApp should start with the progressive pattern: launch fast with centralised infrastructure (2-day deployment), validate the product with real users, and iterate on the smart contracts and UX without the overhead of managing decentralised infrastructure. The fat indexer is tempting at this stage because it delivers the best UX (0.960), but it creates architectural debt that is expensive to unwind later. As the protocol matures and TVL grows, the team migrates toward the modular pattern: replacing the centralised API with a subgraph (one service swap), moving the frontend to IPFS (another swap), adding an alternative RPC relay (another swap). Each migration is incremental -- a single service replacement rather than a full rewrite. The modular decomposition makes each step low-risk because the other services continue operating unchanged. By the time the protocol has significant TVL and real censorship risk, the architecture is fully modular with independent failovers for every component.

5.2 The Fat Indexer Trap

The fat indexer pattern scored lowest among the non-trivial patterns (AQS 0.724), and we want to be explicit about why: it is a Web2 architecture disguised as a Web3 application. The smart contracts are decentralised. Everything else -- the data layer, the query engine, the API, the hosting -- is controlled by a single entity. When that entity goes down (or is ordered to take the service down), the dApp becomes unusable even though the smart contracts are still running. Our censorship simulation proved this: 18 minutes of complete downtime from a single hosting block. OpenSea's architecture works for OpenSea because OpenSea is a company that accepts the trade-off of centralisation for UX performance. But for protocols that claim to be decentralised -- governance DAOs, DeFi protocols, public goods infrastructure -- the fat indexer pattern undermines the core value proposition. If your "decentralised" governance platform can be taken offline by a single cloud provider, it is not decentralised in any meaningful sense. We recommend the fat indexer only for explicitly centralised products that happen to use blockchain for specific features (NFT marketplaces, centralised exchanges with on-chain settlement) and do not claim decentralisation as a feature.

6. Conclusion

6.1 What We Found

The modular microservice pattern achieves the highest AQS (0.896) through independent service decomposition that enables both strong UX and robust censorship resistance -- 12-second recovery versus 18 minutes for centralised architectures. Progressive decentralisation (0.868) offers the fastest path from idea to production. The thin

client pattern (0.844) remains a solid default for DeFi. Fat indexers (0.724) deliver the best UX but sacrifice the decentralisation that justifies using a blockchain. Fully on-chain (0.628) is a conceptual anchor, not a practical choice.

6.2 What We Are Releasing

The W3DAAF evaluation toolkit, all six reference implementations (community governance dApp in each pattern), benchmarking harness, censorship simulation scripts (DNS takedown, hosting block, RPC block, IPFS filter), AQS calculator, progressive decentralisation migration guide with step-by-step service swap procedures, and modular microservice template with Docker Compose configurations are available at w3daaf-arch.org under Apache 2.0.

References

Benet, J. (2014). IPFS: Content Addressed, Versioned, P2P File System. arXiv:1407.3561.

Bianchi, E. (2025). Runtime monitoring systems for smart contract integrity. Blockchain, Web3 & Digital Trust Journal, 2025(2), 19-26.

Buterin, V. (2014). A Next-Generation Smart Contract Platform. Ethereum Whitepaper.

Compound Labs (2023). Compound Governance Documentation. docs.compound.finance.

Edge and Node (2024). The Graph Documentation. thegraph.com/docs.

ENS Labs (2024). ENS Frontend Decentralisation Journey. blog.ens.domains.

Garcia, A. (2025). Automated vulnerability detection in smart contracts using AI. Blockchain, Web3 & Digital Trust Journal, 2025(2), 1-8.

Hansen, N. (2025). Programming language design for safer smart contract execution. Blockchain, Web3 & Digital Trust Journal, 2025(2), 9-18.

IPFS Documentation (2024). IPFS Pinning and Persistence. docs.ipfs.tech.

Kwon, J., and Buchman, E. (2019). Cosmos: A Network of Distributed Ledgers.

Novak, S., and Kovacs, N. (2025). Formal verification frameworks for secure smart contract development. Blockchain, Web3 & Digital Trust Journal, 2025(1), 37-44.

OpenSea (2024). OpenSea API Documentation. docs.opensea.io.

Protocol Labs (2023). Filecoin: A Decentralized Storage Network.

Rossi, L. (2025). AI-assisted smart contract testing and validation frameworks. Blockchain, Web3 & Digital Trust Journal, 2025(2), 27-36.

Samani, K. (2019). The Web3 Stack. multicoin.capital blog.

Uniswap Labs (2024). Uniswap Frontend Architecture. github.com/Uniswap.

Williams, S. et al. (2019). Arweave: A Protocol for Economically Sustainable Information Permanence.

Wood, G. (2014). Ethereum: A Secure Decentralised Transaction Ledger. Yellow Paper.

Zamyatin, A. et al. (2021). SoK: Communication across distributed ledgers. Financial Cryptography 2021.

Zheng, Z. et al. (2020). An overview on smart contracts. Future Generation Computer Systems, 105, 475-491.

Declarations

Funding

Supported by the Italian Ministry of University and Research (MUR, PRIN 2024-2026), the Austrian Science Fund (FWF, grant P 51024-N), and the Estonian Research Council (PRG13024). No funder influenced results.

Conflict of Interest

No competing interests. No dApp platform or infrastructure provider funded this work.

ISSN: e-ISSN: 3117-5988 / p-ISSN: 3117-597X Vol: 2025, Issues: 2, 2025

Data Availability Statement

Six reference implementations, benchmarking harness, censorship simulation, AQS calculator, migration guide, and Docker templates at w3daaf-arch.org under Apache 2.0.

Ethical Approval

No human participant data. Developer study participants gave informed consent. Ethical exemption: Mediterranean Institute of Technology Ethics Board (ref. MITR-2025-ETH-0424).

ISSN: e-ISSN: 3117-5988 / p-ISSN: 3117-597X Vol: 2025, Issues: 2, 2025

Appendix A

Reference Implementation Specifications and AQS Calculation

Technical details of the six reference implementations and AQS scoring.

Reference dApp Specification

AQS Calculation and Censorship Simulation