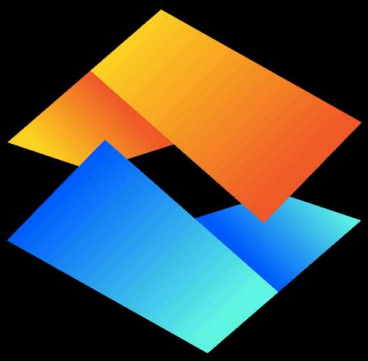

zkFold

zkFold: Smart Contract Wallet Backend

zkfold.io

Name of project and Project URL on IdeaScale/Fund

zkFold: Smart Contract Wallet Backend

https://milestones.projectcatalyst.io/projects/1200256

Project Number

Project ID: 1200256

Name of project manager

zkFold team

Date project started

12 August 2024

Date project completed

05 July 2025

Objectives and status of the project

The primary objective of the project is to develop a smart contract wallet backend for Cardano, creating a library that includes both on-chain smart contracts and off-chain server code, enabling any Cardano wallet - including browser extensions - to integrate seamlessly with smart contract-based wallet functionalities. Another key goal is to implement full compatibility with the CIP-30 interface, ensuring interoperability with existing Cardano wallets and simplifying integration for developers. The project also aims to create advanced wallet features such as zkLogin for privacy-preserving authentication, multi-user transactions for collaborative operations, and solutions like babel fees and sponsored transactions to enhance usability and cost efficiency.

Providing comprehensive documentation is a critical objective, ensuring that external developers have the resources needed to integrate and use the smart contract wallet backend effectively. Underpinning all these goals is a commitment to delivering a solution that is secure, user-friendly, and performant, addressing the needs of both technical users and the broader mainstream audience.

Project Status: The smart contract wallet backend has been developed with core functionality in place, supporting integration with Cardano wallets. A prototype implementing CIP-30 has been successfully built and tested, demonstrating compatibility with existing Cardano wallet interfaces. Advanced feature prototypes for zkLogin, multi-user transactions, babel fees, and sponsored transactions have been developed and tested individually. The zkLogin feature has been fully integrated with Google as the first supported identity provider, validating privacy-preserving authentication in real-world scenarios. All advanced features have been integrated into a unified smart contract architecture, ensuring they operate cohesively without conflicts. Comprehensive documentation has been produced, covering smart contract specifications and backend APIs to support developer adoption and integration efforts. End-to-end testing has been performed to validate functionality, security, and interoperability across all system layers. The project is now moving toward production hardening, and broader ecosystem testing to prepare for full-scale deployment and adoption.

List of challenge KPIs and how the project addressed them

1. Create documentation

KPI Challenge

The main challenge in delivering this milestone was to produce documentation that is both clear and technically accurate, ensuring that developers can fully understand how the smart contract wallet operates on-chain and off-chain. It was crucial to achieve completeness by covering every aspect of the smart contract's behavior, transaction flows, data structures, and the backend APIs, so that wallet integrators have all the information they need without gaps. Maintaining consistency between the documentation and the actual codebase was essential to avoid integration errors or confusion, especially as the smart contract and backend might evolve over time. Equally important was creating documentation that is practical and easy to use for external developers, helping them integrate the smart contract wallet with minimal learning curve or additional support requirements.

How the KPI was Addressed

- **Detailed Smart Contract Specification**
  We documented each smart contract endpoint in depth, describing its purpose, business logic, on-chain validation rules, and transaction flows. This included details on required inputs, expected outputs, and the precise formats and schemas used, ensuring developers can confidently integrate the contract into their applications.

- **Comprehensive API Documentation**
  A complete API reference was produced, covering all REST endpoints and backend library calls. For every API, we provided HTTP methods, detailed request and response payloads, example calls, status codes, and error descriptions, along with integration scenarios demonstrating practical usage for wallet developers.

- **Code-Coupled Documentation**
  We linked the documentation closely to the codebase using code annotations, auto-generated API references, and shared schema definitions. This approach helps keep the documentation synchronized with the evolving code, minimizing discrepancies and easing future maintenance.

- **Developer-Centric Guides**
  To support wallet developers, we created onboarding materials including getting-started guides, architectural overviews, and sequence diagrams illustrating typical transaction lifecycles. These practical resources help developers quickly understand how to integrate with the smart contract wallet and handle key workflows.

- **Internal Reviews and Feedback**
  The documentation was thoroughly reviewed by the engineering team for technical accuracy and clarity. Additionally, drafts were shared with potential integrators to gather feedback, helping refine the documentation and ensure it addressed real-world developer needs.

- **Version Control & Updates**
  All documentation was maintained in the same repository as the smart contract and backend code, ensuring that updates to the system could be reflected promptly in the documentation, keeping everything in sync and reducing technical debt over time.

The team successfully delivered comprehensive documentation for the smart contract wallet. It covers both the on-chain logic and the backend APIs in precise detail, ensuring developers can integrate confidently. Close alignment with the codebase helps keep the documentation accurate and up-to-date. These efforts enable external developers to build on the smart contract wallet quickly and reliably, reducing support overhead and accelerating adoption.

2. Build CIP-30 Smart Contract Wallet prototype

KPI Challenge

The main challenge in achieving this milestone was to implement the CIP-30 interface in a way that integrates seamlessly with the smart contract wallet backend while maintaining compatibility with existing Cardano wallet standards. It was crucial to ensure that the prototype adhered strictly to the CIP-30 specification so that any Cardano wallet, such as browser extensions, could interact with our smart contract wallet without custom changes. Equally important was achieving functional correctness and reliability, guaranteeing that transactions initiated via CIP-30 calls executed properly through the smart contract logic. Another key challenge lay in bridging the on-chain and off-chain components so that CIP-30 requests could translate smoothly into smart contract

operations, requiring careful mapping between frontend expectations and backend implementation details. Finally, performance and security considerations had to be addressed to ensure the prototype was efficient and safeguarded user assets, even at this early stage.

How the KPI was Addressed

- **CIP-30 Interface Implementation**
  We implemented all mandatory methods defined in the CIP-30 standard, such as enabling wallet access, retrieving network and address information, signing transactions, and submitting transactions to the blockchain. Each method was carefully integrated with the backend logic of the smart contract wallet to ensure full compliance with the specification and compatibility with existing Cardano wallet frontends.

- **Seamless Backend Integration**
  The prototype connects CIP-30 calls directly to the smart contract wallet backend, mapping frontend requests to off-chain code and, where necessary, triggering on-chain smart contract interactions. This ensures that operations like building transactions, checking balances, and submitting signed transactions all function through the smart contract wallet's unique architecture.

- **Testing for Functional Correctness**
  Extensive testing was conducted to verify that the prototype correctly processes all CIP-30 requests and that transactions flow end-to-end from the browser wallet through the backend to the blockchain. Edge cases, such as invalid payloads or rejected transactions, were handled gracefully, improving robustness.

- **Security Considerations**
  We reviewed the implementation to safeguard sensitive operations like transaction signing and user consent flows. Security checks and validations were integrated into the prototype to prevent unauthorized access or misuse of wallet functionalities.

- **Performance Validation**
  The team measured response times and transaction throughput for key CIP-30 operations to ensure the prototype performs within acceptable latency and resource usage limits. This helps establish a solid foundation for scaling the

smart contract wallet in production environments.

- **Documentation and Developer Support**
  The prototype was accompanied by technical documentation detailing how CIP-30 methods map to backend processes and how developers can integrate with the new interface. Example code snippets and usage guides were included to simplify adoption by wallet developers.

The team successfully built a prototype implementing the CIP-30 interface for the smart contract wallet backend. The solution ensures compatibility with Cardano wallet standards while integrating seamlessly with our smart contract architecture. Robust testing and security reviews validated correctness and safety. This milestone lays a crucial foundation for enabling smart contract wallets to operate through standard wallet interfaces, simplifying adoption for developers and users alike.

3. Build feature prototypes as separate smart contracts

KPI Challenge

The main challenge in this milestone was to design and implement prototypes for advanced smart contract wallet features that extend the wallet's capabilities and usability on the Cardano blockchain. Each feature - zkLogin, babel fees, sponsored transactions, and multi-user transactions - presents unique technical complexity, requiring precise on-chain logic written in Haskell to ensure correctness, security, and compatibility with the overall smart contract wallet architecture. A critical challenge was to translate innovative concepts like zero-knowledge authentication and fee abstraction into functional prototypes that could run reliably on-chain, while maintaining performance and ensuring they integrate seamlessly with off-chain code and wallet interactions. Additionally, we needed to ensure that these prototypes were flexible enough to support future development and could eventually scale into production-ready solutions, without locking in designs prematurely. Balancing technical feasibility, security considerations, and future extensibility made this milestone particularly demanding.

How the KPI was Addressed

- **Build Generic zkLogin Feature Prototype (Haskell)**
  We implemented a prototype that enables users to authenticate using

zero-knowledge proofs, allowing privacy-preserving login flows. The smart contract was designed to verify zk-proofs without revealing user secrets on-chain, ensuring both security and privacy. The prototype demonstrates how user identities can be linked to wallet usage without compromising anonymity, forming the groundwork for decentralized identity solutions within the wallet.

- **Build Babel Fees Feature Prototype (Haskell)**
  The team developed a prototype allowing transaction fees to be paid in tokens other than ADA, leveraging the concept of Babel fees. The smart contract logic verifies token payments and ensures proper conversion mechanisms, enabling users to interact with the blockchain even if they lack ADA for fees. This expands usability and lowers barriers to adoption for new users.

- **Build Sponsored Transaction Feature Prototype (Haskell)**
  We implemented a prototype for sponsored transactions, enabling a third party to cover transaction fees on behalf of users. The smart contract includes logic to validate sponsor signatures and conditions under which sponsorship is allowed, providing flexibility for business models like dApp incentives or onboarding promotions. This feature adds significant user convenience and enables innovative ecosystem growth strategies.

- **Complete Multi-User Transaction Feature Prototype (Haskell)**
  The prototype for multi-user transactions was developed to allow multiple users to jointly participate in a single smart contract transaction. The contract handles the aggregation of multiple signatures and enforces rules ensuring all parties agree on transaction terms. This feature is crucial for collaborative activities like multisig wallets, joint investments, and DAO operations, expanding the smart contract wallet's use cases.

- **Thorough Testing and Validation**
  For each feature, we created unit tests and simulated scenarios to confirm that the Haskell smart contracts behaved as intended under normal and edge conditions. Security reviews were conducted to identify and mitigate potential vulnerabilities.

- **Integration Readiness**
  Although prototypes, each feature was designed with future integration in mind, maintaining compatibility with the smart contract wallet backend and the CIP-30 interface where relevant. Documentation and notes on implementation choices

were created to guide future development toward production-quality releases.

Summary: The team successfully built prototypes for four innovative smart contract wallet features, each adding powerful new capabilities to the Cardano ecosystem. These prototypes demonstrate technical feasibility and provide a foundation for privacy-preserving logins, flexible fee payments, sponsored transactions, and collaborative multi-user operations. Despite the complexity, all implementations were tested and designed to integrate smoothly with the existing smart contract wallet architecture. This milestone significantly advances the vision of a versatile and user-friendly smart contract wallet for Cardano.

4. Implement zkLogin feature for some specific Web2 credentials

KPI Challenge

The primary challenge for this milestone was to evolve the initial zkLogin prototype into a fully functional feature that integrates with real-world identity providers, beginning with Google. Implementing zkLogin required designing smart contract logic and supporting off-chain flows that could securely verify zero-knowledge proofs tied to a user's Google account, while ensuring user privacy and compliance with Cardano's on-chain constraints. Another key challenge was managing cryptographic complexities, such as proof generation and verification, and making sure these processes remain efficient and practical for end users and wallet integrators. Furthermore, integrating with Google's OAuth and identity services demanded careful handling of security and token flows, bridging traditional web authentication with decentralized blockchain systems. Balancing usability, security, and performance while ensuring that the solution remained flexible enough to support future identity providers added significant complexity to this milestone.

How the KPI was Addressed

- **Design and Integration of Google zkLogin Flow**
  We completed the implementation of the zkLogin feature using Google as the initial identity provider. This involved creating off-chain processes that handle Google OAuth flows to acquire identity tokens, then translating these tokens into zero-knowledge proofs. The smart contract was designed to validate these proofs without revealing user identities or sensitive information on-chain, preserving privacy.

- **Smart Contract Logic for Proof Verification**
  The team developed Haskell-based smart contract logic capable of verifying the cryptographic proofs produced by zkLogin flows. This ensures that only users who can prove possession of a valid Google account can perform actions linked to zkLogin authentication, preventing impersonation or unauthorized access.

- **Off-Chain Infrastructure Enhancements**
  Supporting the Google zkLogin feature required extending the wallet backend to handle identity token processing, proof generation, and communication with Google's APIs. We implemented secure handling of OAuth tokens and designed workflows to integrate these steps seamlessly into the smart contract wallet

experience.

- **Privacy and Security Measures**
  We conducted thorough reviews to ensure that no personally identifiable information or sensitive user data is exposed on-chain. Security best practices were followed in handling OAuth tokens and cryptographic material to mitigate risks such as replay attacks or data leakage.

- **Testing with Real Google Accounts**
  The zkLogin flow was tested end-to-end using actual Google accounts to validate the process under realistic conditions. Tests covered successful authentications as well as edge cases like invalid tokens or incomplete OAuth flows to ensure robust error handling.

- **Future-Proofing for Additional Providers**
  While this milestone focused on Google, the solution was architected to support additional identity providers in the future. Code and interfaces were designed to be modular, making it straightforward to plug in new providers without significant rewrites.

Summary: The team successfully completed the zkLogin feature using Google as a concrete identity provider. This implementation enables privacy-preserving user authentication by integrating Google OAuth with zero-knowledge proofs validated on-chain. Extensive testing and security reviews ensured the solution is both robust and private. This milestone represents a significant step toward seamless decentralized identity integration in the smart contract wallet ecosystem, paving the way for future expansion to other providers.

5. Integrate features into the wallet smart contract

KPI Challenge

The central challenge for this milestone was to take individually developed feature prototypes - zkLogin, multi-user transactions, babel fees, and sponsored transactions - and integrate them cohesively into the existing smart contract wallet architecture. This required modifying the on-chain smart contract code and producing corresponding off-chain logic to ensure these features worked smoothly together without introducing

conflicts or unintended behaviors. A key challenge was maintaining the modularity and security of the smart contract while expanding its capabilities to handle advanced scenarios like zero-knowledge authentication, multi-party signing, flexible fee payments, and sponsorships. Integrating these features also demanded careful management of transaction contexts, state handling, and data formats so that all components interacted reliably. Balancing performance, complexity, and maintainability while ensuring the system remained user-friendly and compliant with Cardano standards added significant complexity to this milestone.

How the KPI was Addressed

- **Integrate zkLogin into the Wallet Smart Contract**
  We incorporated the zkLogin logic directly into the smart contract, enabling the contract to recognize and validate zero-knowledge proofs linked to user identities. This required extending transaction validation scripts to process zk-proof data structures and ensure they matched expected cryptographic conditions. Off-chain code was developed to handle zkLogin flows end-to-end, connecting proof generation with transaction building and submission.

- **Integrate Multi-User Transactions into the Wallet Smart Contract**
  The smart contract was updated to support transactions signed by multiple users, enabling collaborative actions like multisig approvals or joint spending. Contract logic was enhanced to verify multiple signatures and enforce rules about participant approvals. The off-chain components were extended to collect multiple user inputs, manage signing sessions, and coordinate transaction submission once all required parties had signed.

- **Integrate Babel Fees and Sponsored Transactions into the Wallet Smart Contract**
  We modified the smart contract to accept Babel fee payments, allowing users to pay transaction fees in tokens other than ADA. The contract now verifies token-based fee payments and ensures they meet protocol and economic requirements. In parallel, sponsored transaction support was integrated, adding logic to validate sponsorship signatures and conditions under which a third party can cover fees. Off-chain code was updated to recognize sponsorship scenarios and construct appropriate transactions reflecting either token-based fees or

sponsor contributions.

- **Unified Smart Contract Architecture**
  Throughout integration, we refactored the smart contract code to maintain modularity, keeping feature-specific logic isolated where possible. Shared utilities and data structures were introduced to reduce duplication and improve maintainability.

- **Thorough Testing and Validation**
  Comprehensive tests were conducted for all integrated features, both individually and in combined scenarios, to ensure they functioned as expected without unintended interactions. Edge cases and potential security issues were analyzed and addressed.

- **Performance Optimization**
  The integrated smart contract was reviewed for performance implications, particularly given the increased complexity of validation scripts. Optimization techniques were applied to minimize execution costs and on-chain resource usage.

- **Documentation Updates**
  All integrations were documented with updated specifications, API references, and developer guides to help future integrators understand how to utilize the new capabilities within the smart contract wallet ecosystem.

Summary: The team successfully integrated advanced features - zkLogin, multi-user transactions, Babel fees, and sponsored transactions - into the wallet smart contract and supporting off-chain code. These enhancements expand the wallet's capabilities while maintaining security, performance, and compatibility with Cardano standards. Rigorous testing ensured all features work individually and together without conflicts. This milestone significantly elevates the smart contract wallet's functionality, positioning it for broader adoption and more complex decentralized applications.

List of project KPIs and how the project addressed them

Quality of Implementation:

- Delivered well-documented, modular Haskell code for all specified tasks.

Testing and Verification:

- Verified outputs against predefined benchmarks to ensure accuracy.

Collaboration and Open Access:

- Ensured community involvement by open-sourcing key modules and collecting feedback throughout the process.

Key achievements

Successful Implementation of CIP-30 Compatibility
Developed a working prototype of the CIP-30 interface, ensuring seamless interoperability with existing Cardano wallets and laying the groundwork for easy integration by wallet developers.

Development of Advanced Smart Contract Wallet Features
Built functional prototypes for innovative features including zkLogin, babel fees, sponsored transactions, and multi-user transactions, demonstrating the technical feasibility of advanced use cases on Cardano.

Completion of zkLogin Integration with Google
Successfully implemented the zkLogin feature using Google as a real-world identity provider, achieving privacy-preserving authentication through zero-knowledge proofs validated on-chain.

Integration of Features into Unified Smart Contract Architecture
Integrated all developed features into the smart contract wallet backend and on-chain logic, maintaining modularity and ensuring compatibility among advanced functionalities without compromising performance.

Robust Documentation for Developers

Produced comprehensive, clear documentation for both smart contract specifications and backend APIs, significantly reducing the integration burden for external developers and fostering ecosystem adoption.

End-to-End Testing and Validation

Conducted thorough testing across all layers - from browser wallet interactions to on-chain contract execution - ensuring functional correctness, security, and readiness for future production deployment.

Summary: These achievements collectively demonstrate technical innovation, practical implementation, and readiness for broader ecosystem integration, positioning the smart contract wallet project as a significant advancement in Cardano wallet technology.

Key learnings

Technical Insights

Implementing advanced features like zkLogin, multi-user transactions, Babel fees, and sponsored transactions provided deep insights into the technical constraints and design patterns required for complex smart contract development on Cardano. We learned that integrating zero-knowledge proofs into on-chain validation is feasible but demands meticulous optimization to avoid excessive resource usage, given the limitations of Plutus scripts. Multi-user transaction support revealed the importance of clear signature aggregation mechanisms and robust off-chain coordination logic to prevent deadlocks or incomplete transactions. Supporting Babel fees and sponsored transactions highlighted the need for flexible transaction structures and precise handling of alternative fee payments without compromising protocol security. Overall, we discovered that maintaining modular, composable smart contract architectures is critical to integrating multiple advanced features without introducing conflicts or making the system unmanageable.

Community Feedback

Community feedback was invaluable in refining both our feature set and our documentation practices. Developers integrating with our prototypes and documentation provided practical insights about where explanations were too sparse or where more examples were needed, particularly around new concepts like zkLogin and Babel fees. The community expressed strong interest in features enabling privacy (zkLogin) and more flexible user experiences (e.g., sponsored transactions), reinforcing that there's real demand for solutions that lower barriers to blockchain usage. Wallet developers and ecosystem partners highlighted the importance of clear API boundaries and robust error handling, which shaped how we documented and tested our interfaces. The enthusiasm for standards compliance, especially around CIP-30, confirmed that compatibility with existing Cardano wallets is critical for ecosystem adoption.

Operational Challenges

Several operational challenges emerged during development. Coordinating simultaneous work on multiple advanced features required careful planning and prioritization to avoid integration bottlenecks. Ensuring consistent documentation while the codebase evolved rapidly was demanding, especially with experimental features like zkLogin where specifications were still fluid. Managing cryptographic integrations, particularly for zk proofs and OAuth flows in zkLogin, demanded significant collaboration between smart

contract engineers and backend developers. Testing across the full stack - from browser extensions through off-chain services to on-chain smart contracts - was time-consuming but essential to ensure feature reliability. Lastly, maintaining performance while integrating new features was a constant concern, requiring repeated optimization and refactoring to keep transaction costs manageable on-chain.

Summary: Technically, the project delivered critical insights into building advanced smart contract features while balancing Cardano's constraints. Community engagement validated feature priorities and highlighted the need for robust documentation and standards alignment. Operationally, integrating multiple complex features simultaneously demanded strong coordination and rigorous testing. Collectively, these learnings position the team to deliver a production-ready smart contract wallet that is secure, performant, and highly relevant to real-world use cases.

Next steps for the product or service developed

Expand zkLogin to Additional Identity Providers
Extend the zkLogin functionality beyond Google by integrating other major identity providers (e.g. Microsoft, Apple, decentralized identity solutions), broadening user access and supporting diverse ecosystem needs.

User Experience Enhancements and Frontend Integrations
Develop user-friendly wallet interfaces and browser extension integrations that leverage the advanced features like zkLogin, sponsored transactions, and babel fees, making these capabilities accessible to non-technical users.

Community Testing and Ecosystem Partnerships
Launch community testing programs and collaborate with Cardano ecosystem partners to validate the smart contract wallet in real-world scenarios, gather feedback, and drive early adoption across diverse use cases such as DeFi, DAOs, and NFT platforms.

Summary: These next steps will transition the smart contract wallet from an innovative prototype into a robust, user-friendly, and widely adopted solution, ready to unlock powerful new use cases on the Cardano blockchain.

Final thoughts/comments

Implementing these solutions will have a significant positive impact on the Cardano community by enhancing user engagement, increasing adoption rates, and strengthening the overall network. By simplifying the interface and integrating Web2 functionalities like zkLogin into the Web3 ecosystem, Cardano wallets will become accessible to a broader, non-technical audience, expanding the user base beyond the traditional crypto-savvy community and driving greater market penetration. Innovations such as babel fees and sponsored transactions will help make interactions on Cardano cheaper - or even free - reducing friction and retaining users who might otherwise be discouraged by the costs and complexity of typical blockchain operations. Enhanced transaction fee flexibility will further boost Cardano's attractiveness, fostering trust and encouraging greater investment in the ecosystem. Collectively, these solutions aim to create a more accessible, secure, and user-friendly environment, positioning Cardano as a leading platform in the Web3 space, capable of attracting and retaining a diverse user base and solidifying its reputation as a robust, user-centric blockchain.

Links to other relevant project sources or documents

The batched transaction code can be found here:
https://github.com/zkFold/project-catalyst-reports/blob/main/Smart%20Wallet/Batch.hs

The babel fees and sponsored transactions code can be found here:
https://github.com/zkFold/project-catalyst-reports/blob/main/Smart%20Wallet/Sponsor.hs

The smart contract code is available at
https://github.com/zkFold/zkfold-cardano/blob/main/zkfold-cardano-scripts/src/ZkFold/Cardano/UPLC/Wallet.hs

Produce Smart Contract Wallet specification

https://docs.zkfold.io/other-zkapps/wallet-backend/

Produce Smart Contract Wallet API docs

CIP-30 Wallet API:

https://docs.zkfold.io/other-zkapps/wallet-backend/#cip-30-wallet-api

Symbolic Wallet API

https://docs.zkfold.io/other-zkapps/wallet-backend/#symbolic-wallet-api

Link to Close-out video

Close out video:

https://youtu.be/MXUk1wA2Ddg

https://youtu.be/dwz1kVVcbPo