Blockchains and Overlay Networks
Challenge Task Report
Team Zeus

Alessandro Motta, Akos Istvan Imets, Noah Isaak,
Gökberk Beydemir, Annamaria Vass

May 2025

1 Abstract

Online-tournament organisers frequently face accusations of prize-money withholding, bracket manipulation, and opaque match outcomes. To address these shortcomings, we present a Decentralised Tournament Management app that leverages blockchain technology to ensure a platform that is (1) transparent, (2) resistant to fraud and match manipulation, and (3) capable of fair, delay-free prize distribution. Transparency is achieved through a public ledger that provides a global, immutable, and viewable record of every match and tournament; fraud resistance is provided by blockchain-enabled cryptographic signing of match results by both participants; and fair prize distribution is guaranteed by smart contracts that automatically pay out the prize pool upon tournament completion. The platform demonstrates that transparent, self-executing tournaments can be realised without compromising user experience.

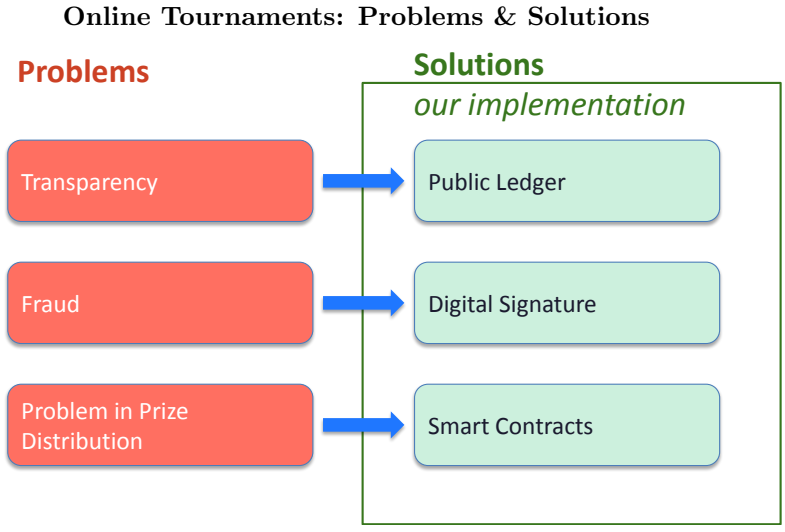

Figure 1: Problem-solution overview of online tournaments. Common issues such as lack of transparency, fraud, and unfair prize distribution are addressed in our implementation by using blockchain primitives: public ledgers, digital signatures, and smart contracts.

2 Introduction

We present a Decentralized Tournament Manager built on the Ethereum blockchain, designed to ensure fair, transparent, and trustless handling of online tournaments. By leveraging smart contracts, the system automates bracket progression, prize distribution, and dispute resolution without relying on centralized authorities.

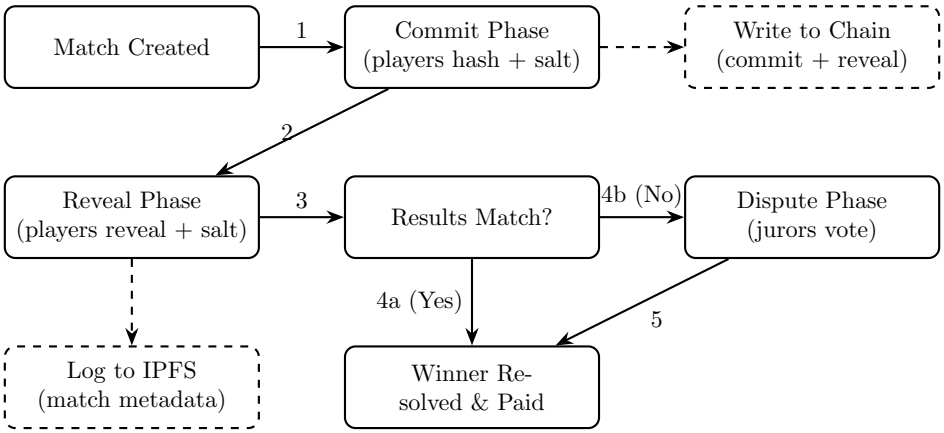

Figure 2: Overview of the numbered match lifecycle stages: from creation (1), commit (2), reveal (3), to resolution via direct agreement (4a) or juror dispute (4b–5). Dashed arrows show off-chain logging and on-chain events.

Figure 2 outlines the lifecycle of a single match in our platform. A match begins with creation (step 1), followed by a commit-reveal protocol to ensure integrity and fairness. In the commit phase (step 2), both players independently submit a hashed version of their result along with a secret salt. This prevents either party from adjusting their response after seeing the opponent's.

Match results are verified in the reveal phase (step 3), where players disclose their actual result and salt. If the revealed outcomes match, the smart contract automatically determines the winner and triggers a payout (step 4a). If results diverge, the match enters a dispute phase (step 4b), where a decentralized jury is invoked to vote on the outcome.

Disputes are resolved via an on-chain jury mechanism (step 5), where external participants stake tokens, and their reputation to vote on the correct result. To support fair decisions of the juries and maintain auditability, the system logs match metadata and relevant evidence to the InterPlanetary File System (IPFS)—a decentralized, content-addressable storage network that captures the progression of each match in a verifiable way. These actions (indicated by the dashed arrows in the diagram) are auxiliary but essential for transparency.

Alternative approaches, such as centralized moderators or direct game API integrations, reintroduce trust assumptions and often limit applicability across games. In contrast, our solution is fully decentralized and game-independent, ensuring fair play through digital signatures, commit-reveal schemes, and community-driven dispute resolution.

By integrating smart contracts, content-addressable storage (IPFS), and decentralized arbitration, the platform addresses key pain points in online tournaments—such as unverifiable match results, delayed or unfair prize distribution, and potential manipulation—without relying on centralized entities. The remainder of this paper introduces the platform's architecture, outlines the smart contract logic, and discusses how transparency, fraud resistance, and fairness are technically achieved.

3 Method / Implementation

3.1 Transparency: Public Ledger Events

To achieve transparency, every critical interaction within a tournament is recorded on the Ethereum blockchain via event logs. This includes player registration, match creation, result commitment and reveal, and prize payout. These events form an immutable, tamper-proof audit trail that anyone can inspect using blockchain explorers. By anchoring key actions in a public ledger, we ensure accountability and allow both participants and observers to verify the legitimacy of the tournament flow.

3.2 Fraud Prevention: Commit-Reveal and Jury System

To ensure integrity in match result reporting and prevent fraud, we implemented a two-phase commit-reveal scheme combined with a decentralized jury-based dispute resolution mechanism, fully integrated in the *MatchContract.sol* smart contract.

In the commit phase, each player submits a cryptographic commitment—computed as a Keccak-256 hash of their claimed result plus a secret salt—via the *commitResul*t function. This commitment conceals the actual result but is immutable. During the reveal phase, players call *revealResult*, submitting the claimed result and salt, which the contract verifies against the commitment. If results match, the contract finalizes the outcome and disburses rewards accordingly.

However, if there is a discrepancy in the revealed results or if one of the players fails to reveal their commitment, the contract transitions into a dispute state. To resolve this, a decentralized jury system is invoked. Jurors in this system are external participants who voluntarily choose to vote on the outcome of the disputed match. They do so by interacting with the contract's voting function and are required to lock a stake as collateral. This economic stake serves as a deterrent against dishonest behavior, as jurors who are found to have voted inconsistently with the majority may lose their staked amount and suffer a reduction in their on-chain reputation, which is managed in a separate *ReputationRegistry.sol* contract.

Currently, the system requires exactly three jurors to reach a decision in any given dispute. This specific number is dictated by the constraints of the test network environment, which allows only twenty accounts. Within this limited setting, jurors are not randomly selected but instead self-elect to participate based on their own incentives. While this approach functions adequately for prototyping, it does pose a potential vulnerability to collusion or bribery.

3.3 Threat Model and Potential Attacks

The main attack vector targets the jury system's multi-signature-like decision process. With only three jurors who self-elect, collusion or bribery could manipulate dispute outcomes. An attacker controlling or influencing a majority of jurors can bias the resolution, undermining fairness.

To mitigate this, jurors must stake collateral and risk reputation loss, discouraging dishonest votes. Additionally, players and jurors losing too many reputation points can be banned, limiting repeated malicious behavior.

In a production environment, these vulnerabilities can be addressed by increasing the number of jurors and introducing random selection from a large, staked pool—potentially using verifiable random functions (VRFs)—to reduce manipulation risk and improve resilience.

3.4 IPFS Logging

To assist in disputes and support verifiability, each match logs metadata and gameplay-related artifacts on the InterPlanetary File System (IPFS). These logs may include links to replays, screenshots, or summaries of game state. The content-addressable nature of IPFS ensures the immutability and

integrity of submitted evidence: any tampering would result in a different hash and thus be immediately detectable. During disputes, jurors access these off-chain records through the on-chain CIDs to evaluate the claims based on verifiable match data. This mechanism strengthens the trustworthiness of the resolution process and discourages false reporting strategies.

The smart contract stores only the content hashes (CIDs) on-chain, enabling jurors and observers to access and verify the data off-chain without bloating the blockchain. This evidence can be used by jurors during disputes to make informed, transparent decisions.

3.5 Fair Prize Distribution: Smart Contract Escrow

All entry fees are held in escrow by smart contracts at the time of tournament registration. Upon resolution of each match, the prize money flows automatically to the winner, either directly or via the tournament bracket contract. In the event of disputes, the winner as determined by the jury receives the funds, while jurors are compensated from the dispute stake pool. This guarantees timely, fair, and automatic prize distribution without manual intervention.

3.6 Solution Architecture

The contract stack is composed of four primary components:

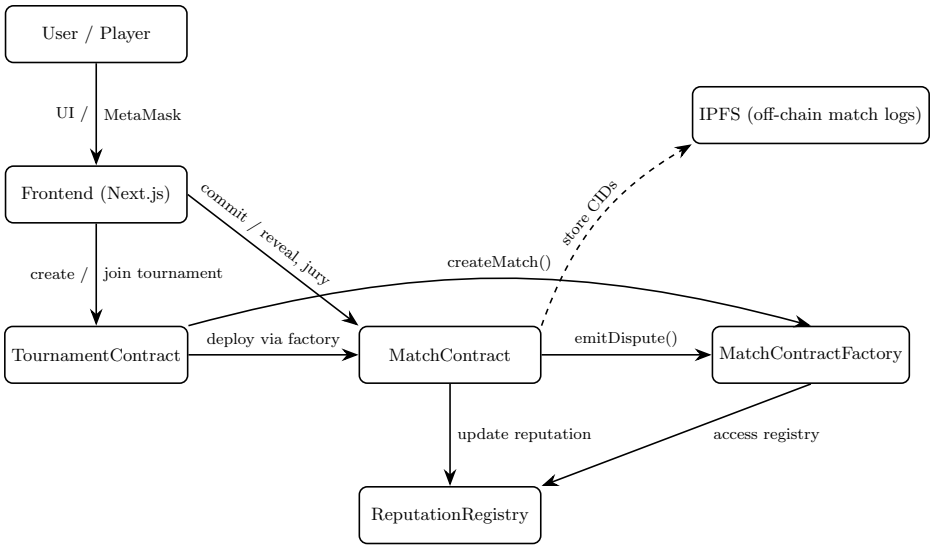

Figure 3: Architecture of smart contracts and their interactions. Players interact through the frontend, which connects to smart contracts for tournament control, match logic, reputation tracking, and off-chain data logging. IPFS is used to store verifiable match metadata.

- **TournamentContract**: Serves as the central coordinator for the tournament system, managing the full lifecycle from creation to prize distribution. It supports single-elimination brackets with 4, 8, or 16 participants and enforces structured progression through defined states (Registration, InProgress, Completed, Cancelled). The contract handles entry fee collection into a secure prize pool, and integrates with the ReputationRegistry to ensure only reputable players may register. It automatically deploys match instances via the MatchContractFactory, tracks results, and advances rounds accordingly. Upon tournament completion,

it transfers the prize to the winner and updates reputations based on performance. The contract exposes view functions to support frontend rendering of tournament and match data, and emits events for all key transitions to enable efficient UI updates.

- **MatchContract**: Created per match via a minimal proxy factory pattern (EIP-1167) [1], this contract governs the full lifecycle of a game using a robust state machine, from player entry and cryptographic commitments to result revelation, dispute detection, and resolution. Designed to support both standalone and tournament-linked matches, it implements a trustless commit-reveal scheme, integrates IPFS for off-chain evidence logging, and escalates disputes to the factory for jury-based resolution with staking and reputation-based juror selection. Tournament compatibility ensures seamless bracket advancement and prize coordination, while user-centric features like combined commit-reveal and streamlined jury voting enhance gas efficiency and player experience.
- **ReputationRegistry**: Tracks player and juror reputations. Winning, truthful revealing, and valid jury votes increase reputation, while no-shows and dishonest behavior result in penalties or bans. The registry determines stake requirements and access to participation roles.
- **MatchContractFactory**: Deploys and initializes new match contracts and relays reputation updates. It reduces gas costs via minimal proxy cloning and provides global events for dispute handling.

Each of these components is modular and upgradeable, facilitating future extensions such as support for other tournament types or deeper integration with external games.

3.7 Functional Requirements

To clarify the intended system capabilities from a user and operational perspective, Table 1 summarizes the core functional requirements of the platform. These requirements are derived from typical user interactions with the tournament manager through the browser-based frontend.

Functional Requirements
Users should be able to connect securely their wallet to the dApp.
Users should be able to browse available tournaments.
Users should be able to create a new tournament.
Users should be able to enter tournaments and pay entry fees.
Users can see (and report) tournament/match results.
After a tournament concludes, the winner should receive the prize money.
All transactions should be securely processed and recorded on the blockchain.

Table 1: Core functional requirements for interacting with the tournament system.

The platform fully supports the core functional requirements expected by users interacting with the tournament system. Through the browser-based frontend, users can securely connect their wallets, browse and create tournaments, enter tournaments with entry fee payments, and view or report match and tournament results. Upon tournament completion, winners are automatically awarded their prize money. All user actions and financial transactions are securely processed and immutably recorded on the blockchain, ensuring transparency and trustworthiness throughout the system.

3.8 Frontend & UX

The frontend is built with Next.js (React framework) and leverages the `shadcn/ui` component library and Tailwind CSS to provide a clean, modern, and responsive interface that works seamlessly

across desktop and mobile devices. State management is handled efficiently using React Hooks (e.g., `useState`, `useEffect`), ensuring a dynamic user experience. Usability is prioritized through an intuitive layout, featuring dedicated components like `TournamentCard` for browsing, `MatchCard` for match details, and interactive `TournamentBracket` visualizations. Navigation is managed by Next.js's App Router, guiding users through functionalities such as tournament discovery, creation, a personalized dashboard, and a jury dispute resolution section.

Secure wallet authentication via browser extensions like MetaMask is the entry point for users. Throughout the application, clear transaction feedback and error messages are provided using `sonner` toast notifications, informing users about initiated transactions, confirmations, or any issues encountered. Visual cues, such as color-coded status badges (e.g., "Open", "Active", "Completed" for tournaments; "Pending", "Disputed" for matches), help users quickly understand the current state of tournaments and individual matches. When creating a tournament or submitting a match result, forms built with `react-hook-form` and `Zod` for schema validation ensure data integrity before smart contract interactions. The platform also includes dialogs for actions like match reporting (`MatchReportDialog`) and jury voting (`JuryVoteDialog`).

All interactions with the Ethereum smart contracts, like fetching tournament data, joining events, creating new tournaments, reporting match outcomes, and participating in jury votes, are streamlined through a custom `useWeb3` hook. This abstraction layer simplifies communication with the blockchain and helps optimize smart contract calls to minimize gas usage and latency. The clear separation between the frontend and the backend contract logic also enables easy future extension with new game types or tournament formats.

3.9 Deployment and Testnet Compatibility

The dApp and all smart contracts have been successfully deployed and tested on a local Hardhat [2] development node using MetaMask for interaction. This setup enables full end-to-end testing, including tournament creation, match resolution, and prize distribution, within a sandboxed environment. The complete source code and deployment instructions are provided in the accompanying GitHub repository.

4 Conclusion

The Decentralized Tournament Manager presented in this work provides a robust and extensible framework for managing online tournaments in a trustless and decentralized manner. It offers a game-agnostic foundation capable of supporting fair tournament progression, transparent match verification, and automatic prize distribution. Through the use of smart contracts, commit-reveal result submission, IPFS-based match logging, and a reputation-backed jury mechanism, the system achieves its goals without relying on central authorities or off-chain enforcement.

While the current implementation supports single-elimination brackets, it is designed with extensibility in mind. Future versions could incorporate support for alternative formats such as round-robin or Swiss-style tournaments, broadening the framework's applicability. One of the primary areas identified for improvement is the jury selection process: currently jurors self-select by staking, which may lead to selection bias or low participation in low-stakes games. Introducing verifiable random selection of jurors—using VRF or similar methods—would enhance fairness and scalability.

Another avenue for future development is the integration of game APIs or trusted data feeds to partially automate match result verification. While this introduces some trade-offs in trust assumptions, it may be useful in contexts where games themselves can provide signed outcomes or zero-knowledge attestations. Additionally, improvements to the frontend experience and streamlining contract deployment would enhance accessibility for non-technical users.

Overall, the system lays a solid groundwork for decentralized tournament management. It separates the tournament logic from game-specific mechanics, allowing future developers to integrate their own games while inheriting a secure, auditable, and autonomous competitive structure.

5 Source Code

The complete source code, including smart contracts, frontend interface, and deployment scripts, is available at:

`https://github.com/guilloboi1917/BCOLN_dApp/tree/main`

References

[1] Ethereum Improvement Proposals (EIP). Eip-1167: Minimal proxy contract. `https://eips.ethereum.org/EIPS/eip-1167`. Accessed: 2025-04-11.

[2] Hardhat. Documentation. `https://hardhat.org/docs`. Accessed: 2025-04-18.

A Frontend Screenshots

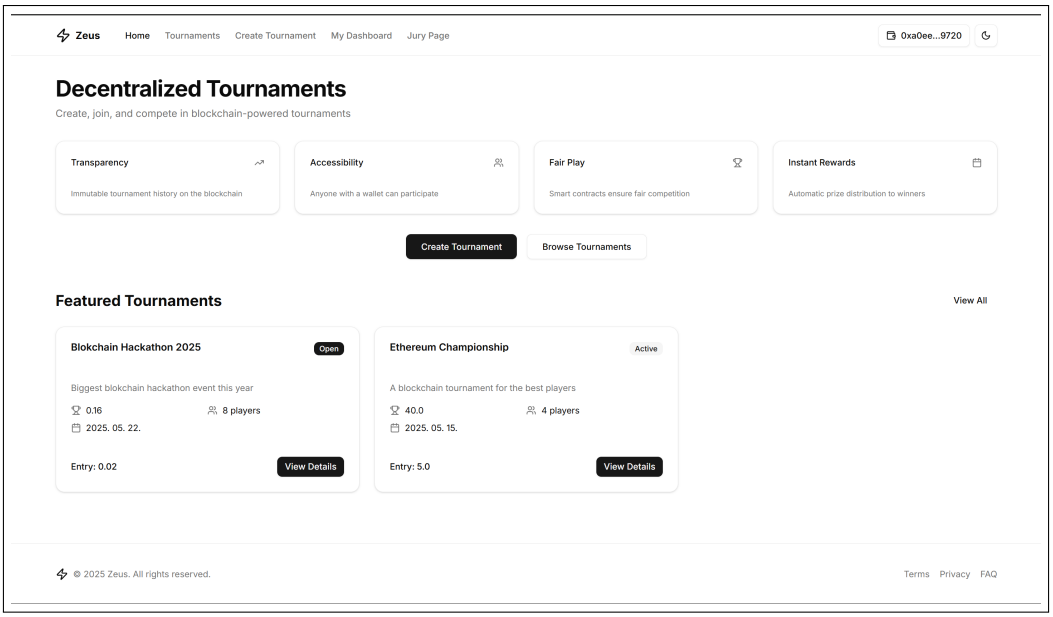

Figure 4: The landing page providing an overview of featured tournaments.

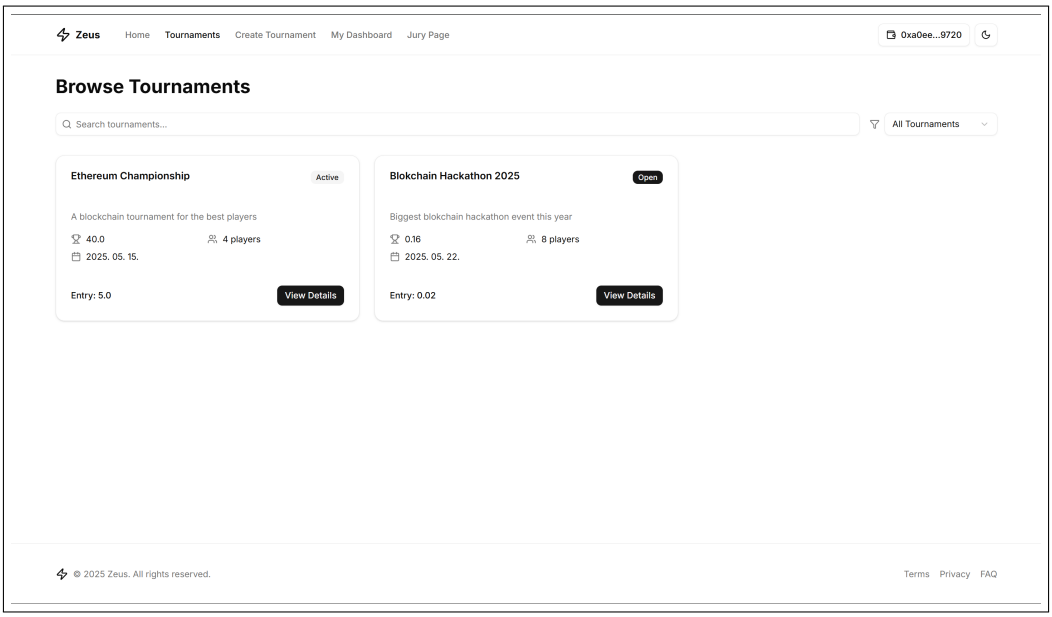

Figure 5: The tournaments browsing page with search and filter functionalities.

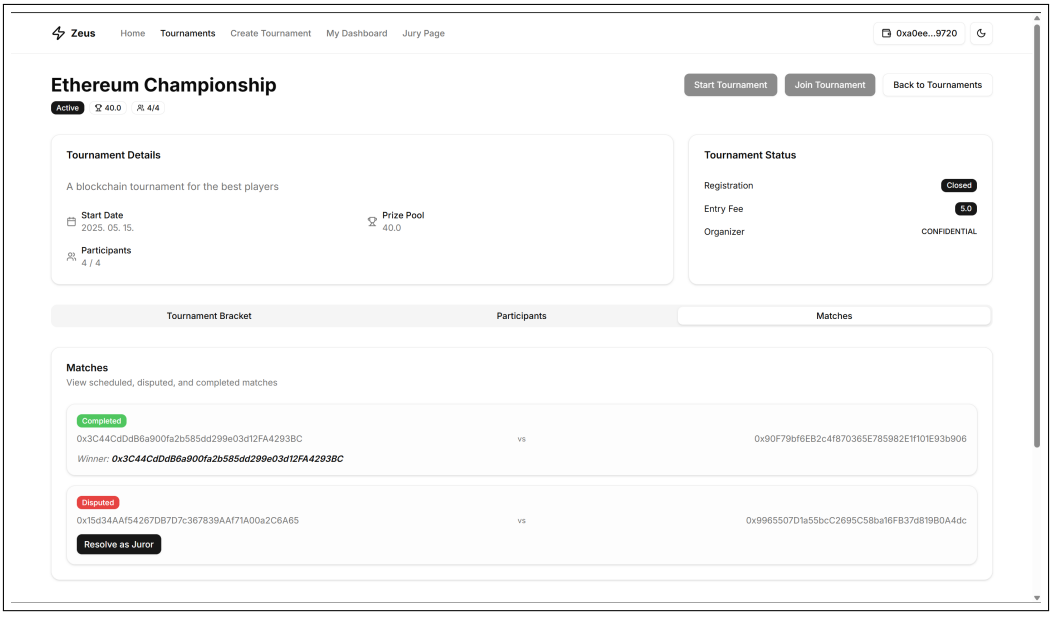

Figure 6: A tournament detail page showing status, description, participants, bracket, and matches.

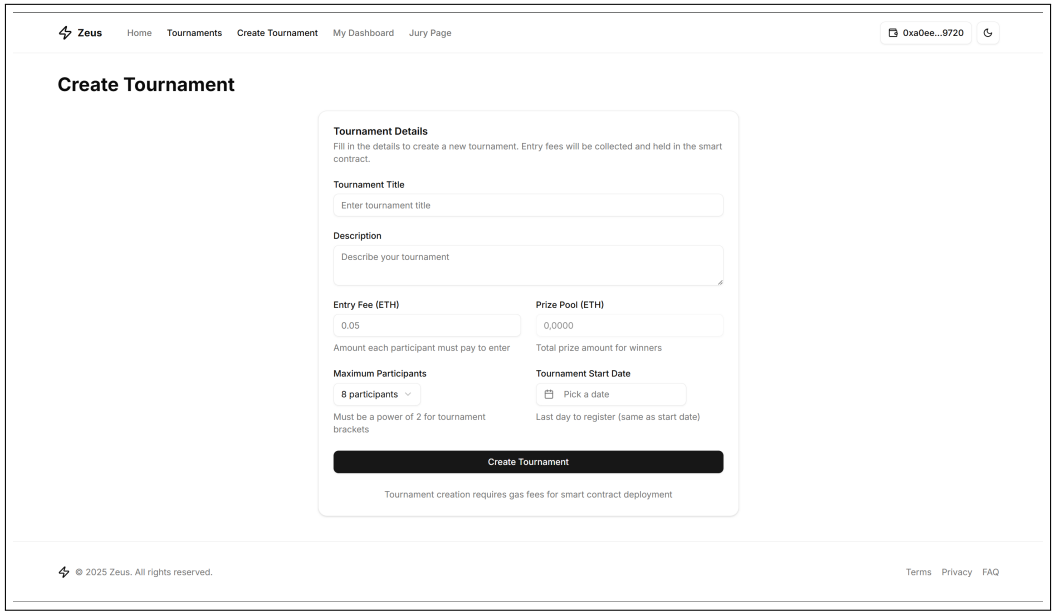

Figure 7: The form for creating a new tournament.

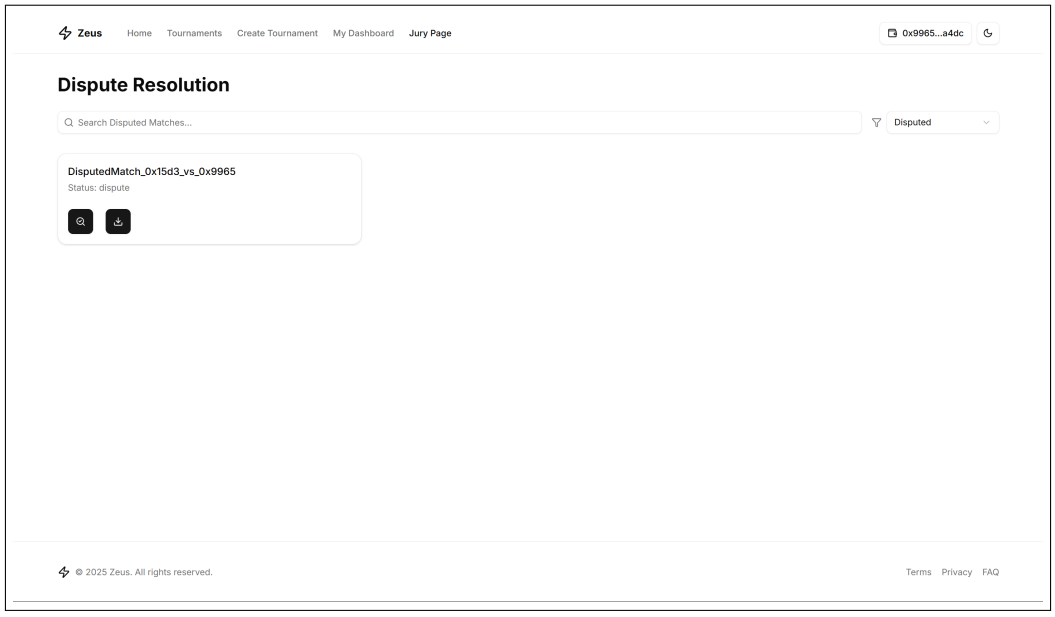

Figure 8: The jury page for viewing and participating in dispute resolutions.

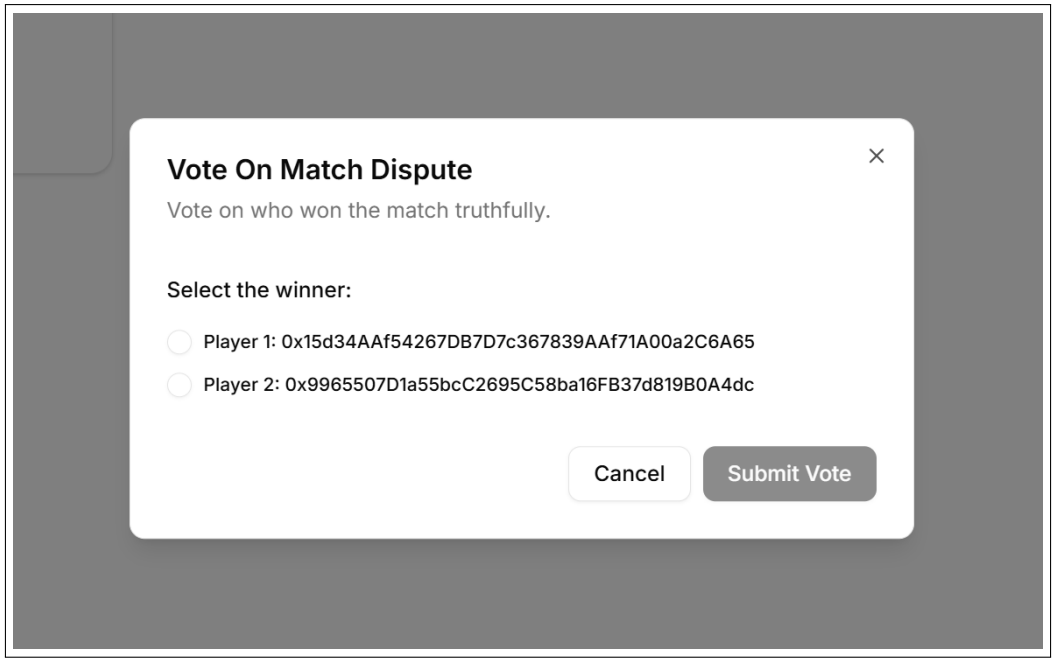

Figure 9: The vote dialog to resolve disputes.

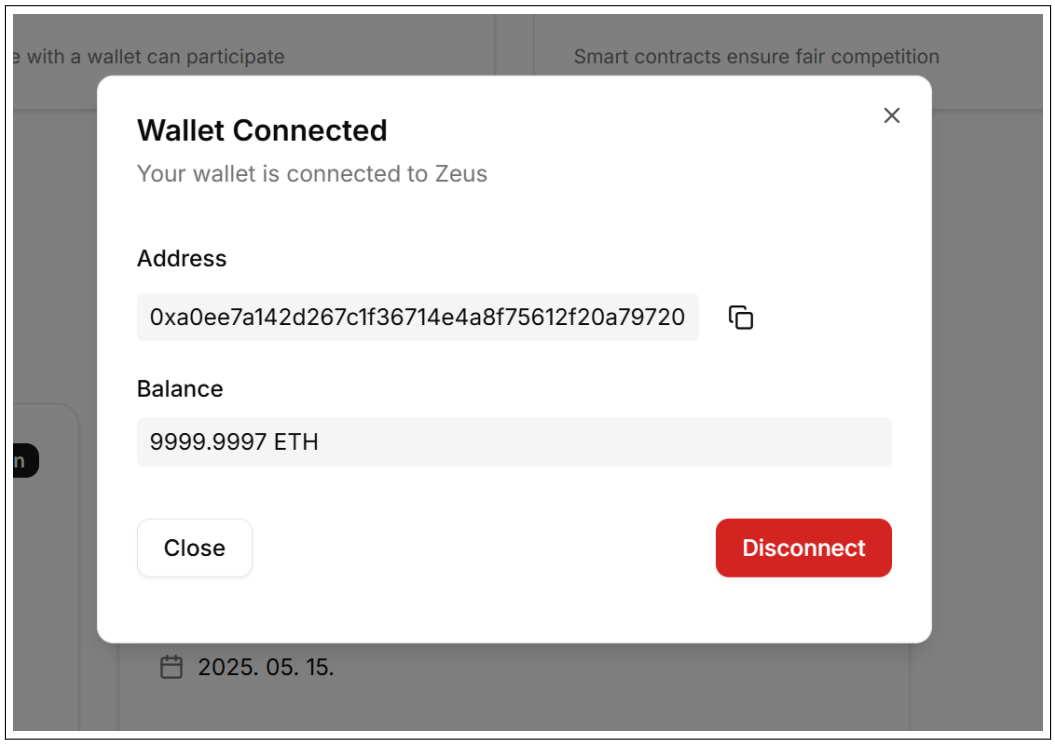

Figure 10: Dialog for the connected wallet.