Truth Bounties: Combating Misinformation Through Economic Incentives

Technical Research Paper
Presented to the Faculty of the
School of Engineering and Applied Science
University of Virginia

By

Jianming Li, Christopher Cicero, Sankalpa Banjade

April 25, 2025

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

ADVISOR

Jack W. Davidson, Professor, Department of Computer Science

Introduction:

The proliferation of misinformation in digital spaces represents one of the most pressing challenges of our time. In response, the Truth Bounties platform emerged from a collaboration between the University of Virginia School of Law and Computer Science Department as an innovative solution that harnesses economic incentives to promote content verification. This web-based application enables content authors to place monetary bounties on their claims, effectively transforming traditional content verification into a market-driven process.

Truth Bounties operates through a workflow where content authors can place financial stakes on their published content's accuracy, creating tangible incentives for potential challengers to scrutinize content and identify false or misleading information. The platform manages the entire process from initial content submission through final arbitration and payment processing, creating a self-sustaining ecosystem for content verification.

The development demanded careful consideration of several critical technical objectives: content integrity, system security, scalability, and user experience. These objectives presented significant challenges, from creating reliable mechanisms for content verification to implementing secure payment processing and managing complex system integrations. To address these challenges, Truth Bounties was built on a modern technology stack including Django, PostgreSQL, Heroku, and various other services, chosen for reliability, scalability, and maintainability.

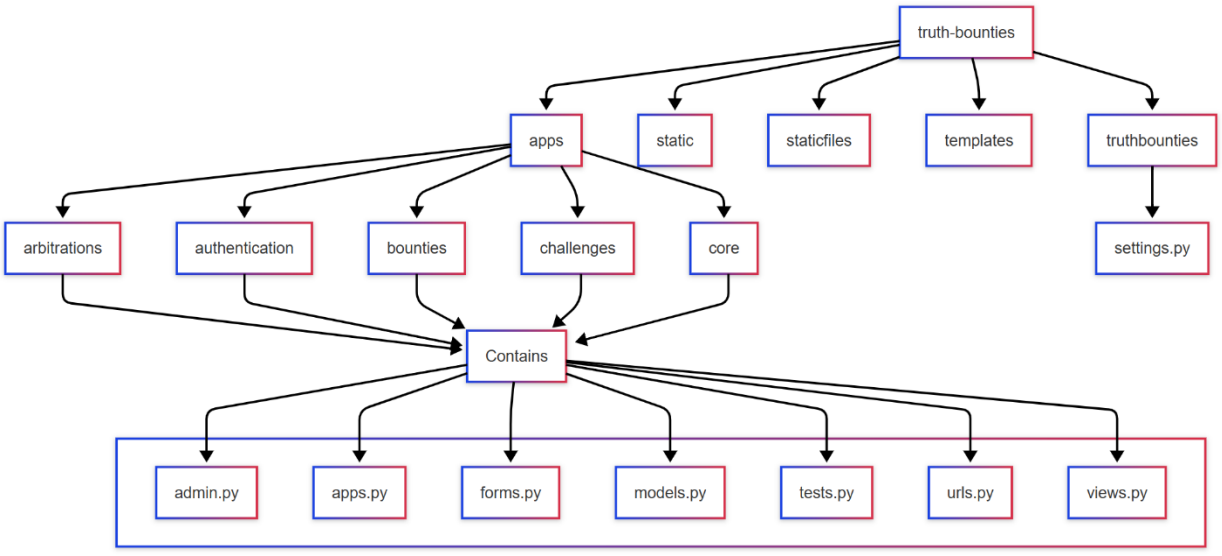
This technical documentation provides comprehensive guidance for developers working with the Truth Bounties platform, detailing system architecture, core functionalities, development workflows, and security implementations. Through careful architecture decisions

and robust implementation practices, we have created a system that not only serves its immediate purpose but also provides a foundation for future innovation in content verification technologies.

System Architecture:

The Truth Bounties platform employs a three-tier architecture that balances scalability, maintainability, and security. Each tier serves a specific purpose while maintaining clear interfaces with other components. At the presentation tier, users interact with a responsive web interface built using Bootstrap and custom JavaScript. The frontend implements a dynamic status system that provides real-time updates on bounties and challenges.

The application tier, powered by Django 5.1.6, forms the core of our system. It consists of several key services:

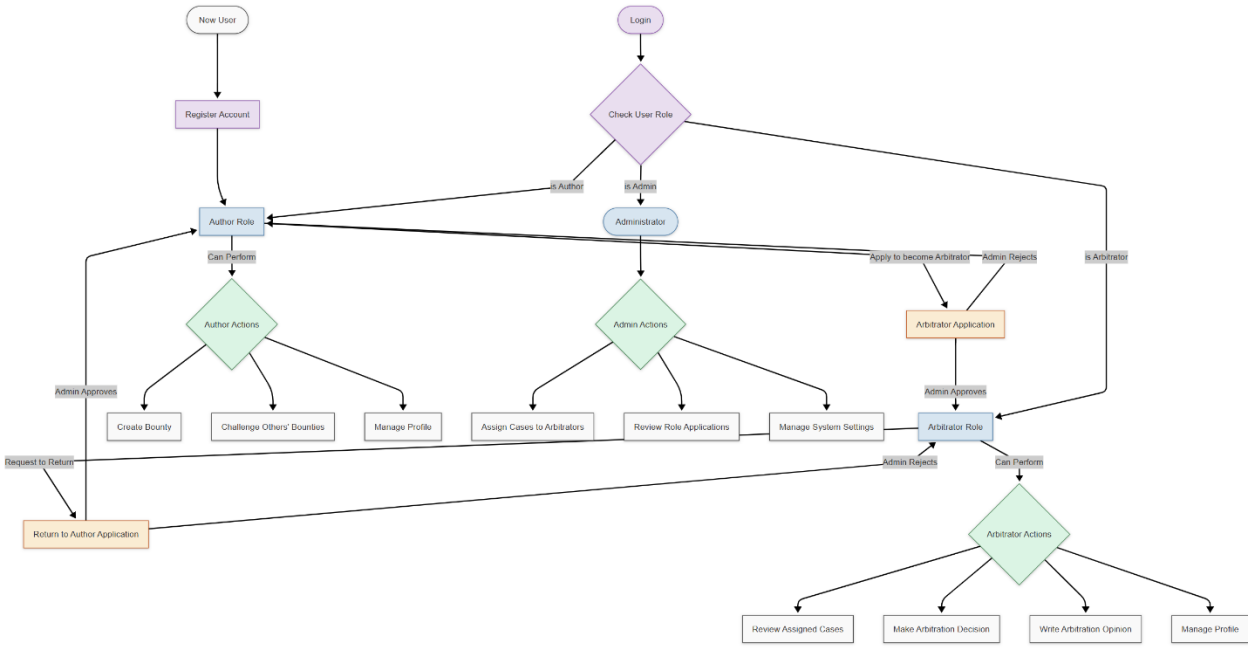


These services communicate through well-defined interfaces, using Django's signal system for event handling. When a new challenge is submitted, the system triggers a series of coordinated actions. The data tier utilizes PostgreSQL 16 as the primary database. Content artifacts and metadata are stored directly in PostgreSQL, ensuring data consistency and simplifying the architecture. Payment processing is implemented through direct integration with the Stripe API.

The entire system runs on Heroku's platform, utilizing a multi-environment setup. This architecture has proven effective during initial testing, with the modular design allowing for independent scaling of components as needed, while maintaining clear separation of concerns for simplified maintenance and updates.

Core Components:

The Truth Bounties platform is built around several key components that work together to manage the content verification process. Each component is designed with modularity and maintainability in mind, while ensuring seamless integration with other parts of the system. At the heart of the platform lies the Authentication component, which manages user identity and access control. Built upon Django's robust authentication system, we extended it to support different user roles including Authors, Challengers, and Arbitrators. Each role carries specific permissions and capabilities within the system:



For instance, Authors can create bounties and respond to challenges, while Challengers can dispute content and provide evidence. The authentication system also tracks user reputation scores, which influence their ability to participate in various platform activities.

The Content Management component serves as the backbone of our bounty system. It handles the creation, storage, and tracking of bounties placed on content. When an Author creates a bounty, the system captures essential information including the content itself, source URL, bounty amount, and current status. This component maintains the complete lifecycle of a bounty, from its initial active state through potential challenges and ultimate resolution. A sophisticated status tracking system ensures that all participants can monitor the current state of any bounty in the system.

Challenge Processing forms another crucial component of the platform. When a Challenger disputes content, this component manages the entire workflow from submission through resolution. It stores the Challenger's explanation and evidence, handles the notification system for all involved parties, and maintains the challenge status throughout the process. The component includes validation mechanisms to ensure challenges meet platform requirements and are submitted within the designated timeframe for each bounty.

Payment Integration represents a critical component that handles all financial transactions within the platform. Through direct integration with the Stripe API, we implement secure payment processing for bounty creation and distribution. When an Author creates a bounty, the system securely processes their payment and holds the funds until the challenge period expires or a challenge is resolved. Similarly, when challenges are successful, this component manages the distribution of funds to the appropriate parties.

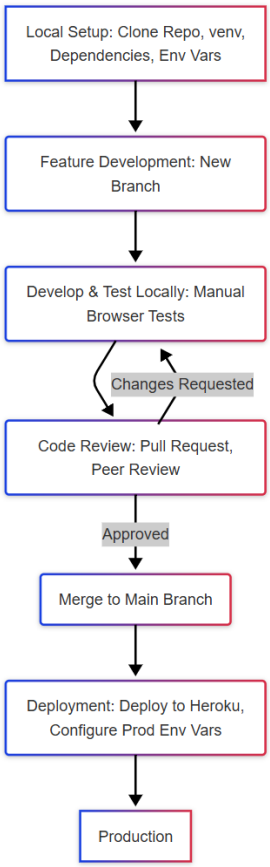
The User Interface components tie these core functionalities together through a series of intuitive views and templates. We've implemented a clean, responsive design that guides users through complex processes like bounty creation and challenge submission. The interface provides real-time feedback on actions and maintains consistent status information across all platform features.

These components are orchestrated through a well-defined event system that ensures proper coordination of activities across the platform. For example, when a challenge is submitted, the system automatically updates the bounty status, notifies relevant parties, and prepares for potential arbitration. This tight integration between components, while maintaining clear boundaries, allows the platform to handle complex workflows while remaining maintainable and extensible.

The design of these core components reflects our commitment to creating a robust platform that can evolve with changing requirements while maintaining high standards of security and usability. Each component's clear responsibilities and well-defined interfaces make it possible to modify or enhance functionality without disrupting the overall system operation.

Development Workflow:

The development workflow for Truth Bounties follows modern software development practices, emphasizing code quality, collaboration, and maintainability. Our workflow is designed to support multiple developers working concurrently while maintaining system stability and reliability. Development begins with local environment setup:



Each developer clones the repository from GitHub and creates a virtual environment using Python's venv module. The project dependencies are managed through requirements.txt, ensuring consistency across development environments.

For local development, developers first clone the repository, create a virtual environment, and install dependencies. The project includes a detailed setup guide in the README.md file that walks through the process of configuring environment variables, particularly for sensitive information like Stripe API keys and database credentials. We maintain separate settings files for development and production environments, allowing developers to work with local databases and test API keys without affecting the production system.

Feature development follows a branch-based workflow. When starting work on a new feature or bug fix, developers create a new branch from the main development branch. We

follow a naming convention that clearly identifies the type of change and the feature being implemented (e.g., ‘sprint6-payment-integration’ or ‘sprint4-s3-Challenger’). This branching strategy helps maintain clean version control and facilitates code review.

Testing is conducted locally using Django's development server (localhost:8000). Developers manually test new features and changes through the browser, paying particular attention to: User authentication and authorization, Bounty creation and management, Challenge submission and processing, Payment integration, API endpoints. Code reviews are conducted through GitHub's pull request system. Each pull request must be reviewed by at least one other team member before merging. Reviewers check for code quality and adherence to project standards, testing the changes locally to verify functionality.

For deployment, we use Heroku's platform, which provides a straightforward way to move from development to production. The transition from local development to production is managed through Heroku's deployment pipeline, with environment variables configured appropriately for the production environment. This streamlined workflow has proven effective for our team, allowing us to maintain steady development progress while ensuring functionality through direct testing. The simple structure and documented processes make it straightforward for new developers to join the project and begin contributing effectively.

Adding New Features:

The Truth Bounties platform was designed with extensibility in mind, making the process of adding new features straightforward and systematic. To illustrate how developers can expand the platform's functionality, let's explore the implementation of a hypothetical feature: a notification system for bounties approaching expiration. Adding a new feature begins with understanding how it fits into the existing architecture. In our platform, most features interact

with the core bounty system and user management components. For our expiration notification example, we would build upon these existing foundations rather than creating entirely separate systems. This approach maintains the platform's cohesive design while expanding its capabilities.

The implementation process starts with creating a new feature branch in Git, following our naming conventions to clearly identify the feature's purpose. This branching strategy allows developers to work independently without affecting the main codebase. For the notification system, we would extend the existing User model to include notification preferences and create appropriate database fields to track notification status and user preferences.

Our straightforward architecture means that most new features can be implemented by modifying existing components rather than creating entirely new services. The notification system, for instance, would integrate directly with our existing views and templates. We would add notification display elements to the user dashboard and create preference settings in the user profile section, maintaining consistency with our Bootstrap-based design.

Local development and testing play a crucial role in feature implementation. Using Django's development server, developers can immediately see how their changes affect the system. For the notification feature, this would involve testing the timing of notifications, verifying the display in the user interface, and ensuring proper integration with existing bounty functionality. This direct testing approach allows for rapid iteration and immediate feedback.

The final steps involve code review through GitHub's pull request system and deployment through Heroku. We emphasize clean, maintainable code that follows the project's existing patterns. Once approved, new features are merged into the main branch and deployed, making them available to all users. This process demonstrates how Truth Bounties can evolve while maintaining its core simplicity and reliability. By following these patterns, developers can

confidently add new features that enhance the platform's functionality while preserving its architectural integrity.

Database Schema:

The Truth Bounties platform utilizes a PostgreSQL database with a schema designed for clarity and efficiency. Our database structure reflects the core relationships between users, bounties, and challenges while maintaining data integrity and supporting the platform's key features. At the foundation of our schema is the User table, which extends Django's built-in authentication system. This table stores essential user information including usernames, email addresses, and hashed passwords.

We extended the default user model to include additional fields specific to our platform. The Bounty table serves as the central element of our database, storing information about content submissions and their associated bounties. Challenges are tracked in their own table, maintaining relationships with both bounties and users.

The relationships between these tables enable efficient querying for common operations like displaying active bounties, tracking challenge status, and managing user interactions. We utilize foreign key constraints to maintain referential integrity and ensure that related records are handled appropriately when updates or deletions occur.

This straightforward schema design supports all of our core functionality while remaining simple enough to maintain and modify as needed. The structure allows for easy querying of common scenarios, such as finding all challenges for a particular bounty or listing all bounties created by a specific user. Future enhancements to the platform, such as adding user reputation systems or expanding the challenge verification process, can be accommodated through simple additions to the existing schema without requiring significant restructuring of the database.

Security Implementation:

The Truth Bounties platform implements several layers of security measures, leveraging both Django's built-in security features and additional custom protections to safeguard user data and financial transactions. Authentication and authorization form the first line of defense. We utilize Django's authentication system, which provides secure password hashing, session management, and protection against common vulnerabilities like session hijacking. User passwords are never stored in plaintext but are instead hashed using Django's password hashers. The system enforces role-based access control, ensuring users can only access and modify resources appropriate to their role.

For financial transactions, we integrate with Stripe's secure payment processing system. All payment information is handled directly by Stripe through their JavaScript library, meaning sensitive credit card data never touches our servers. The integration uses Stripe's client-side tokenization, where payment details are sent directly to Stripe's servers, and our system only receives a secure token to process the transaction:

Cross-Site Request Forgery (CSRF) protection is enabled globally across the platform through Django's middleware. Every form submission includes a CSRF token, and the system validates these tokens on all POST requests. Form validation and input sanitization are handled at both the frontend and backend levels. All user inputs are validated and sanitized before being stored in the database, protecting against SQL injection and cross-site scripting (XSS) attacks. Django's template system automatically escapes HTML in user-submitted content when rendering pages. These security measures create a robust foundation for protecting user data and financial transactions while maintaining a smooth user experience. We prioritized implementing

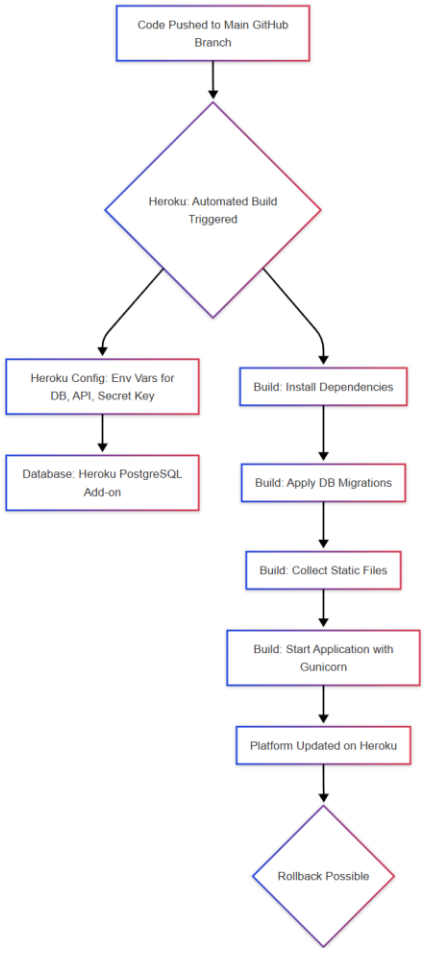
essential security features that directly protect our users' data and financial transactions, laying the groundwork for additional security enhancements as the platform grows.

Deployment Process:

The Truth Bounties platform employs a streamlined deployment process utilizing Heroku's cloud platform, chosen for its reliability and straightforward implementation. This approach allows our team to focus on development while maintaining confident and consistent deployments to the production environment. Our deployment workflow centers around the GitHub repository, where the main branch serves as the source of production-ready code. Critical to our deployment strategy is the management of configuration information through Heroku's environment variables system. This approach keeps sensitive data, such as database credentials, Stripe API keys, and Django's secret key, secure and separate from the codebase. We maintain distinct configurations for development and production environments, with development using local environment variables for testing and production utilizing secure Heroku config vars for live operation.

The database management strategy relies on Heroku's PostgreSQL add-on, providing a robust and scalable solution for our data storage needs. Database migrations are handled automatically during the deployment process through Heroku's release phase, ensuring that the database schema stays in sync with application code. This automation reduces the risk of deployment failures due to missed or incorrect database updates.

When deploying updates to the platform, the process follows a consistent pattern:



Code pushed to the main branch on GitHub triggers Heroku's automated build process. During this build, the system installs necessary dependencies from our requirements.txt file, applies any pending database migrations, and starts the application using gunicorn as our production server. Static file handling is managed through Heroku's built-in systems, with Django's collectstatic command gathering all static assets for efficient serving in the production environment.

This deployment approach has proven both reliable and maintainable for our current needs. Its simplicity helps minimize deployment-related issues while providing the flexibility to roll back changes if necessary. The straightforward nature of the process makes it accessible to all team members, ensuring that deployments can be managed effectively without requiring specialized expertise.

Personal Reflection and Future Considerations:

Working on the Truth Bounties platform has been an invaluable learning experience, particularly in understanding the challenges of building a web application that handles financial transactions and content verification. As a key contributor to this project, I've gained significant insights into both technical implementation and project management aspects of web development.

One of the most significant learnings came from implementing the Stripe payment integration. While I had previous experience with Django, integrating a third-party payment processor revealed the complexities of handling financial transactions securely. The importance of proper error handling and transaction validation became particularly apparent when dealing with real monetary values. This experience has given me a deeper appreciation for the intricacies of financial technology implementations.

The project also highlighted the importance of user experience design. Initially, we focused primarily on functionality, but as the project progressed, we realized that the success of the platform heavily depends on making complex processes, like creating bounties and submitting challenges, intuitive for users. If I were to start over, I would advocate for more upfront investment in user interface design and user testing before implementing features.

Looking forward, several areas of the platform could benefit from further development. First, the notification system needs expansion to keep users better informed about bounty and challenge status changes. Currently, users must actively check for updates, which could limit engagement. Implementing real-time notifications through WebSockets or a similar technology would significantly improve the user experience. Another area for improvement is the content verification system. While our current implementation captures and stores content effectively,

we could enhance it by adding automated screening tools to detect potential manipulation or duplicative content. This enhancement would help maintain the integrity of the bounty system and reduce the burden on human arbitrators. The challenge review process could also be refined. Currently, it's relatively straightforward, but adding features like structured evidence submission forms and standardized evaluation criteria would make the process more consistent and transparent. Additionally, implementing a reputation system for both authors and challengers would help build trust within the platform community.

One aspect I would approach differently is our testing strategy. While we relied primarily on manual testing through localhost, implementing automated tests from the start would have saved time in the long run and provided better confidence in our code changes. This became particularly apparent as the codebase grew and manual testing became more time-consuming. The deployment process, while functional, could be enhanced with automated staging environments and more comprehensive pre-deployment checks. This would provide an additional layer of quality assurance before changes reach production.

Working with Professor Gilbert and Mr. Chau provided valuable insights into how legal and technical requirements intersect in real-world applications. Their expertise helped shape the platform's features to serve practical needs while maintaining technical feasibility. This collaboration demonstrated the importance of cross-disciplinary communication in building successful applications.

Despite these areas for improvement, I'm proud of what we accomplished. The Truth Bounties platform represents a novel approach to content verification, and building it has been both challenging and rewarding. The experience has strengthened my full-stack development skills and given me practical experience in building mission-critical features like payment

processing and user authentication. The project has also reinforced the importance of starting with a minimum viable product and iterating based on feedback. While there are many features we could add, focusing on core functionality first allowed us to create a working platform that can be enhanced over time. This approach proved especially valuable given the project's academic timeline and resource constraints. Moving forward, I believe Truth Bounties has the potential to make a significant impact in the fight against misinformation. The technical foundation we've built is solid, and with continued development focusing on user experience, automated testing, and enhanced features, the platform could become an important tool for content verification across the internet.