PeerPay: A Blockchain-Based Transaction Visualization System

Phase 3 Report: Final Report & Submission

Harsh Dayal
Registration Number: 22BCE10564

Course: IBM blockchain
Institution: VIT

April 19, 2025

Project Repository: https://github.com/Kaos599/PeerPay

Contents

1 Project Objective Recap

The primary objective I set for the PeerPay project was to design, develop, and implement a full-stack application demonstrating fundamental blockchain concepts using a practical, simplified peer-to-peer transaction ledger. My goal was to create a system where transactions are recorded as blocks in a cryptographically linked chain, providing a visual representation of immutability and transparency principles. I aimed for this project to serve as a hands-on learning experience in both blockchain fundamentals (hashing, block structure, chaining) and modern web development practices (React, Node.js, APIs, database integration). This final report summarizes the project's journey, architecture, implemented features, key learnings, and overall outcomes.

2 Tools & Technologies Utilized

The successful development of PeerPay relied on a carefully selected stack of modern web technologies. Table 1 provides a comprehensive list, and the subsequent subsection elaborates on the rationale for key choices.

Table 1: Final Technology Stack Used in PeerPay.

Category	Specific Technology/Tool
Frontend Library	React 18+
Frontend Build Tool	Vite
UI Component Library	Chakra UI
Data Fetching & State	TanStack Query (React Query) v5
Form Management	React Hook Form
Schema Validation	Zod
Animation	Framer Motion
Icons	Lucide React
Backend Runtime	Node.js
Backend Framework	Express.js
Cryptographic Hashing	Node.js 'crypto' module (SHA-256)
Database	MongoDB (via Atlas)
ODM	Mongoose
Development	npm, Git/GitHub, VS Code / Cursor

2.1 Technology Rationale Elaboration

My selection of the technology stack was driven by efficiency, learning goals, and modern development practices:

- **React & Vite:** React's component-based structure allowed for modular UI development. Vite provided an exceptionally fast development environment, significantly speeding up the build and refresh cycle compared to older tools.
- **Chakra UI:** This library was instrumental in building a polished and responsive user interface quickly. Its comprehensive set of pre-built, accessible components, along with its intuitive styling

props and built-in theme support (including dark mode), saved considerable time and effort on UI design and implementation.

- **TanStack Query:** Managing asynchronous server state is often complex. TanStack Query simplified fetching, caching, and synchronizing data from the backend API immensely. Features like automatic refetching on window focus and interval polling were crucial for creating the near real-time feel of the block list updating automatically.
- **React Hook Form & Zod:** These libraries provided a powerful combination for handling form submissions. React Hook Form optimized form state management, while Zod allowed for clear, server-independent schema definition and validation, ensuring data integrity before it even reached the backend.
- **Node.js & Express.js:** Using Node.js allowed me to leverage JavaScript across the full stack. Express.js provided a minimalist yet powerful framework for rapidly setting up the RESTful API endpoints needed for the frontend interaction.
- **MongoDB & Mongoose:** MongoDB's flexible document model suited the structure of blockchain blocks well. Mongoose added a valuable layer of structure through schema definition and validation, simplifying database interactions within the Node.js environment.

3 Summary of Work Across Phases

My work on the PeerPay project progressed through three distinct phases:

- **Phase 1 (Proposal & Idea):** I conducted initial research, defined the problem statement focusing on educational blockchain visualization, proposed the PeerPay solution, outlined objectives and features, selected the technology stack, and established the project scope. The output was a detailed proposal document.
- **Phase 2 (Implementation & Execution):** I translated the proposal into code, developing the core application. This involved building the Node.js backend with Express and Mongoose, implementing the blockchain logic (block structure, hashing, chaining), setting up MongoDB persistence, creating the REST API, developing the React frontend with Chakra UI and other libraries, integrating frontend-backend communication, and performing initial testing and debugging.
- **Phase 3 (Testing & Finalization):** I focused on ensuring robustness and completeness. This included conducting more thorough end-to-end testing, refining the UI/UX based on usage, verifying all core requirements were met, adding code comments, documenting the final system in this report, and preparing the project artifacts for submission.

4 Implemented Features and Functionality

The final version of PeerPay includes the following key features that I implemented:

- **Custom Blockchain Structure:** My backend manages a chain of blocks with index, timestamp, transaction data, previous hash, and SHA-256 hash.
- **MongoDB Persistence:** The entire blockchain is reliably stored in a MongoDB database, ensuring data survives server restarts.
- **RESTful API:** My backend provides well-defined endpoints:
  - ‘GET /blocks‘: Retrieves the entire current blockchain.
  - ‘POST /mine‘: Adds a new transaction as a new block to the chain.

  - 'GET /validate': Checks the cryptographic integrity of the entire chain.

- **Interactive React Frontend:** Built with Vite, providing a dynamic user interface for interacting with the blockchain data.
- **Block Visualization:** Displays fetched blocks sequentially in a clear card format, showing index, timestamp, transaction details, hash, and previous hash.
- **Validated Transaction Submission:** A user-friendly form utilizing React Hook Form and Zod allows users to submit valid transaction data (sender, recipient, amount).
- **Near Real-time Updates:** Automatic polling via TanStack Query refreshes the displayed block list periodically, giving the appearance of a live feed.
- **User Feedback:** Chakra UI toast notifications provide immediate feedback on the success or failure of transaction submissions.
- **Modern UI/UX:** Responsive design adapts to different screen sizes, and includes a toggle for light/dark mode preference.
- **Chain Validation Logic:** The '/validate' endpoint provides an explicit way to verify the chain's integrity, demonstrating the tamper-evident nature of the hash linking.

5 Application Workflow

The typical user interaction and data flow within PeerPay proceeds as follows:

1. **Initial Load:** The user accesses the web application. The React frontend mounts and triggers a data fetch using TanStack Query to the backend's 'GET /blocks' endpoint.
2. **Backend Fetch:** The Node.js/Express backend receives the request, queries the MongoDB database via Mongoose to retrieve all block documents, sorted by index.
3. **Display Blocks:** The backend responds with the blockchain data (usually as a JSON array). TanStack Query updates the frontend state, and React renders the list of 'BlockCard' components, displaying the current chain.
4. **User Submits Transaction:** The user fills in the 'Sender', 'Recipient', and 'Amount' fields in the 'TransactionForm'. Client-side validation using Zod runs automatically.
5. **Frontend Sends Data:** Upon submitting a valid form, the frontend triggers a mutation via TanStack Query, sending a 'POST' request to the backend's '/mine' endpoint with the transaction data in the request body.
6. **Backend Processes Transaction:**
   - The backend receives the transaction data.
   - It fetches the current latest block from MongoDB to get its index and hash (which will be the new block's 'previousHash').
   - It creates a new block object with the next index, current timestamp, the received transaction data, and the 'previousHash'.
   - It calls the 'calculateHash' method to compute the SHA-256 hash for this new block based on its contents.
   - It saves the new block document to MongoDB using Mongoose.

7. **Backend Responds:** The backend sends a success response (e.g., status 201) back to the frontend, possibly including the newly created block data.

8. **Frontend Feedback:** The frontend receives the success response. The ‘useMutation‘ hook updates, triggering a success toast notification via Chakra UI.

9. **Automatic Update:** Independently, TanStack Query’s polling mechanism eventually triggers another ‘GET /blocks‘ request. The backend returns the updated chain (including the newly added block). React re-renders the block list, and the user sees their transaction appear as a new block.

This flow demonstrates the interaction between the user interface, API calls, backend logic, and database persistence.

6 Blockchain Concepts Implementation

While PeerPay uses a centralized database, I implemented core blockchain concepts to achieve the project’s educational goals:

- **Hashing:** Each block’s integrity is maintained by a SHA-256 hash. I implemented a ‘calculateHash‘ method within the Mongoose schema definition in ‘server.js‘. This method computes the hash based on the block’s index, timestamp, data payload (the transaction), and the hash of the previous block. This ensures that any change to the block’s content would result in a different hash.

- **Chaining:** Blocks are linked cryptographically. Each block (except the Genesis block) stores the ‘previousHash‘ field, which contains the exact hash value of the preceding block in the sequence. This creates an unbroken chain; if a block’s hash changes, the ‘previousHash‘ link in the *next* block becomes invalid.

- **Tamper-Evidence (Simulated Immutability):** The combination of unique block hashes and the ‘previousHash‘ link makes the chain tamper-evident. If an attacker were to modify data within a past block in the MongoDB database, recalculating its hash would change it. This change would break the ‘previousHash‘ link of the subsequent block. My ‘GET /validate‘ API endpoint explicitly checks these conditions: it recalculates the hash of each block and compares it to the stored hash, and it verifies that each block’s ‘previousHash‘ matches the actual hash of the preceding block. This allows demonstrating how tampering would be detected, simulating the immutability concept found in decentralized ledgers.

- **Genesis Block:** An initial block (index 0) with a predefined ‘previousHash‘ (e.g., "0") is automatically created by the ‘ensureGenesisBlock‘ function if the database is empty. This provides the starting point for the chain’s hash links.

7 Sample Block Structure (JSON)

The following shows the typical structure of a block as stored in MongoDB and returned by the API:
"$_id$" : "...", $//MongoDBObjectId$
"index": 2,
"timestamp": 1710000000000, // Example epoch timestamp
"data":
"sender": "Charlie",
"recipient": "David",
"amount": 50
,
"previousHash": "abc123def456...", // Hash of block with index 1

"hash": "def789ghi012...", // Calculated SHA-256 hash of this block (index 2)

"$_v$":0//$Mongooseversionkey$

```
"_id" : "...", //MongoDBObjectId"index" : 2, "timestamp" : 1710000000000, //Exampleepochtimestamp"data" : "sender" : "Charlie", "recipient" : "David", "amount" : 50, "previousHash" : "abc123def456...", //Hashofblockwithindex1"hash" : "def789ghi012...", //CalculatedSHA − 256hashofthisblock(index2)"_v":0//Mongooseversionkey
```

Listing 1: Example Final Block Data Structure

8 Final Results and Testing Summary

I performed final testing across the application's features to ensure stability and correctness. The key test results, confirming that all primary functionalities operate as expected, are summarized in Table 2.

Table 2: Summary of Final Testing Results.

Test Case	Expected Result	Status
Fetch Initial Blocks	UI displays Genesis block (and any existing blocks) correctly on load.	Passed
Add Valid Transaction	Transaction submitted, success toast shown, new block appears in UI after polling, block saved in DB with correct index/hashes.	Passed
Add Invalid Transaction	Form validation prevents submission and displays appropriate error messages.	Passed
Chain Integrity Check (‘/validate‘)	API returns ‘ "isValid": true, ... ‘ for an unmodified chain.	Passed
Tamper Detection (‘/validate‘)	After manually altering data/hash in DB, API returns ‘ "isValid": false, ... ‘ indicating the specific failure.	Passed
UI Responsiveness	Layout adjusts correctly on various screen sizes.	Passed
Theme Toggling	Light/Dark mode switch functions correctly, changing UI theme.	Passed
Polling Update	Newly added blocks appear in the UI without manual refresh within the polling interval.	Passed

9 Screenshots

(I will attach final screenshots here from my completed project)

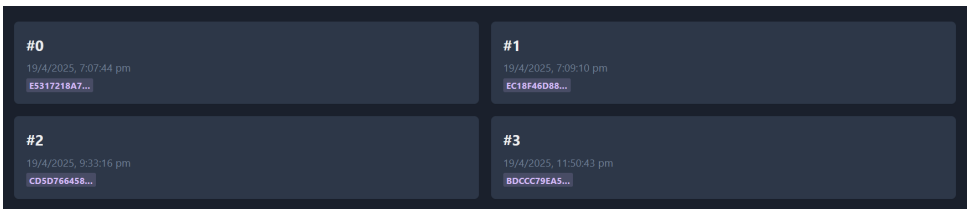

Figure 1: Final UI - Block List View.

Figure 2: Final UI - Transaction Form.

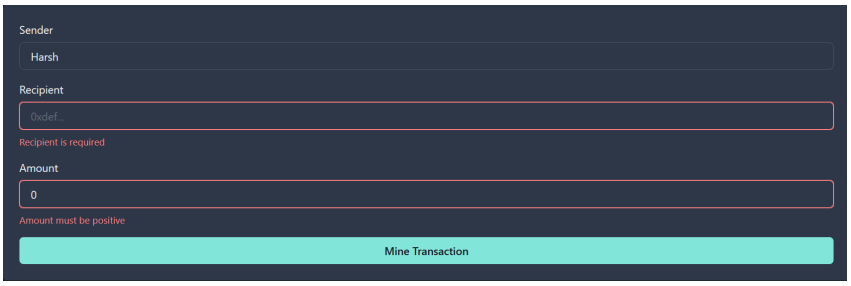

Figure 3: Example Form Validation Error.

Figure 4: Example Success Toast Notification.

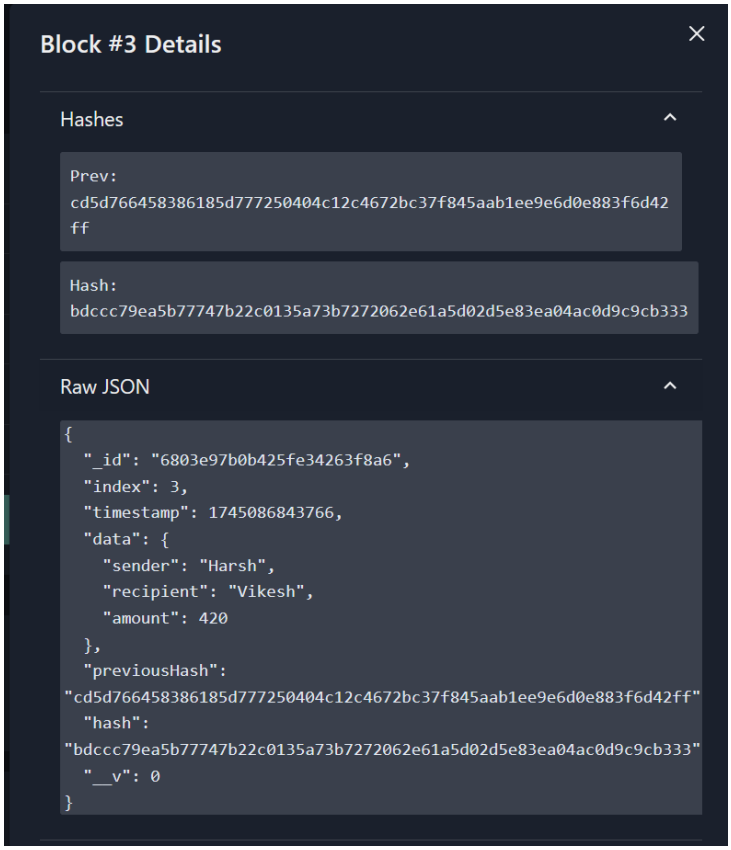

Figure 5: Example Chain Validation Result (API).

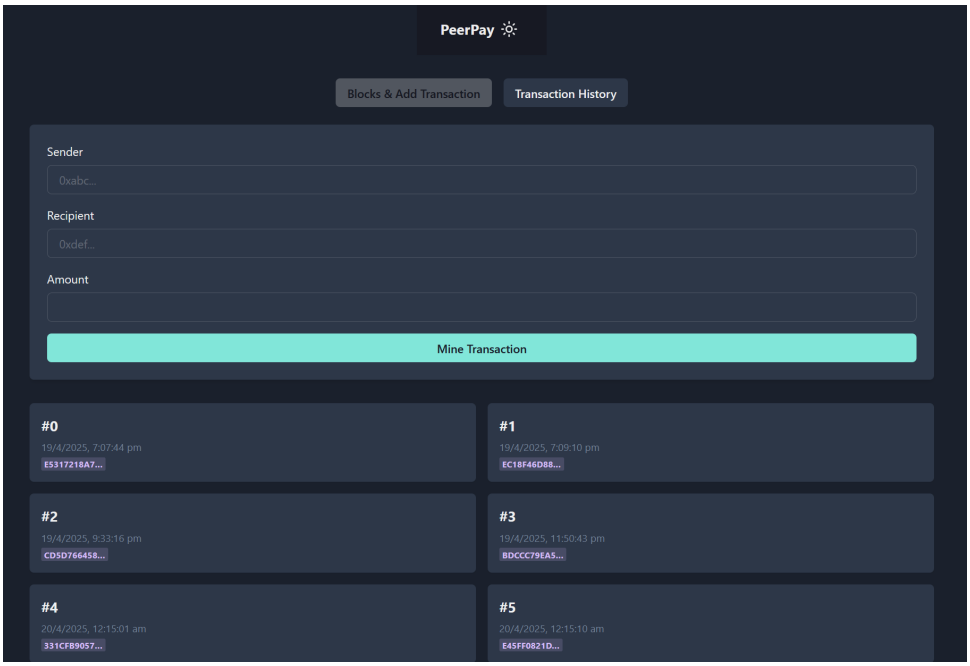

Figure 6: Example Dark Mode UI.

10 Key Learnings

This project provided me with significant learning experiences across various domains:

- **Blockchain Fundamentals:** I moved beyond theoretical knowledge to practically implement block structures, SHA-256 hashing for integrity, cryptographic chaining via previous hashes, and the concept of tamper evidence. Building the validation logic reinforced my understanding of how these elements work together.
- **Full-Stack Development Workflow:** I gained valuable hands-on experience managing the entire development lifecycle, from designing the backend API and database schema to building the frontend UI components and integrating them seamlessly. This involved understanding the request-response cycle and managing state across client and server.
- **Modern Frontend Ecosystem:** I became significantly more proficient with the React ecosystem, particularly Vite for rapid development, Chakra UI for efficient and aesthetic UI construction, TanStack Query for sophisticated server state management (which dramatically simplified data fetching and synchronization), and the React Hook Form/Zod combination for robust form handling.
- **Node.js Backend Development:** I strengthened my skills in building RESTful APIs using Express, handling asynchronous database operations effectively with Mongoose and ‘async/await‘, utilizing built-in Node.js modules like ‘crypto‘, and managing server configurations like CORS.
- **Problem Solving & Debugging:** The project required debugging across the full stack. I developed better strategies for identifying issues, whether they were frontend rendering bugs, API request/response mismatches, backend logic errors, or database interaction problems.

11 Project Highlights

I consider the following to be the main achievements and highlights of the PeerPay project:

- **Successful End-to-End Implementation:** I delivered a fully functional application that successfully meets all the core objectives outlined in the Phase 1 proposal.
- **Clear Demonstration of Concepts:** The application serves its primary educational purpose by effectively visualizing basic blockchain mechanics (hashing, chaining, block structure) in an accessible and interactive manner.
- **Robust Technology Choices:** I utilized a modern, efficient, and widely-used technology stack (React/Vite/Node/MongoDB), ensuring the project reflects current web development practices.
- **High-Quality User Interface:** The frontend, built with Chakra UI, offers a clean, responsive, aesthetically pleasing, and interactive experience with useful feedback mechanisms and features like theme toggling.
- **Persistent and Validatable Chain:** Transactions are reliably stored in MongoDB, and the cryptographic integrity of the stored chain can be programmatically verified via the ‘/validate‘ API endpoint, demonstrating the tamper-evident nature of the structure.

12 Future Scope Revisited

Based on the completed project, several interesting avenues exist for future enhancements or related projects:

- **User Authentication:** Implementing user accounts (e.g., using JWT or OAuth) would allow associating transactions with specific authenticated users and potentially adding permissions.
- **Enhanced Block Explorer:** Adding features like searching for specific transactions or blocks by hash/index, implementing pagination for large chains, and providing more detailed block views.
- **Consensus Mechanism Simulation:** Introducing a simplified Proof-of-Work (PoW) simulation where adding a block requires some computational effort (e.g., finding a hash with a certain number of leading zeros), rather than immediate addition.
- **Transaction Signing:** Incorporating basic digital signature verification using public/private key cryptography to ensure the authenticity of the transaction sender.
- **Networking Simulation:** Exploring basic peer-to-peer concepts, perhaps by running multiple backend instances and simulating block propagation (this would significantly increase complexity).
- **Containerization and Deployment:** Packaging the frontend and backend applications using Docker and deploying them to cloud platforms (e.g., Vercel/Netlify for frontend, Heroku/AWS/Fly.io for backend) for public accessibility.

13 Conclusion

The PeerPay project successfully concluded with the delivery of a functional, full-stack web application that achieves its initial objective: demonstrating fundamental blockchain concepts through an interactive transaction visualization tool. Through the three distinct phases, I designed the system architecture, implemented the backend API with Node.js, Express, and MongoDB, built the interactive React frontend using modern libraries like Chakra UI and TanStack Query, and conducted thorough testing to ensure correctness and usability.

This project served as an invaluable learning experience, significantly deepening my understanding of blockchain principles like hashing and chaining, while also strengthening my practical skills in contemporary full-stack web development. PeerPay stands as a clear, working example of how foundational blockchain ideas can be visualized and interacted with, fulfilling all the project requirements set forth in the initial proposal and providing a solid base for potential future explorations.