Technical Specifications

TeosPump

1. INTRODUCTION

1.1 EXECUTIVE SUMMARY

1.1.1 Brief Overview of the Project

TeosPump is a decentralized launchpad platform built on the Solana blockchain that enables users to create and launch meme/cultural tokens using the $TEOS token as the primary payment and reward mechanism. The platform functions as a Solana launchpad that allows users to pre-sell tokens before launching liquidity pools, specifically targeting the creation of culturally-themed tokens with strong Egyptian branding backed by Elmahrosa International.

1.1.2 Core Business Problem Being Solved

Traditional token launches require significant capital for liquidity pools and lack community-driven pre-sale mechanisms that help creators collect liquidity and make successful launches. The crypto world suffers from rug pulls disguised as community projects, launchpads that promise significant returns but deliver minimal value, and excessive hype-driven vaporware. TeosPump addresses these challenges by providing a secure, transparent, and culturally-focused token creation platform that integrates mobile mining rewards and ensures proper liquidity management.

1.1.3 Key Stakeholders and Users

Stakeholder Group	Description	Primary Interests
Token Creators	Individuals and teams creating meme/cultural tokens	Easy token creation, fair launch mechanisms, community building

Stakeholder Group	Description	Primary Interests
Token Investors	Users purchasing tokens during pre-sale phases	Early access to promising projects, transparent tokenomics, secure transactions
Mobile Miners	Users earning $TEOS rewards through mobile mining	Reward accumulation, seamless wallet integration, fair distribution
Elmahrosa International	Project backing organization	Brand promotion, Egyptian cultural representation, platform success

1.1.4 Expected Business Impact and Value Proposition

The platform enables creators to raise 50-100 SOL or more through secure token sales with minimal costs (0.1 SOL and 2.5% of sales), while leveraging Solana's capability to make token creation highly accessible through tools that allow quick launches riding on current trends or memes. The integration of Egyptian cultural branding and mobile mining rewards creates a unique value proposition that differentiates TeosPump from generic launchpad platforms.

1.2 SYSTEM OVERVIEW

1.2.1 Project Context

Business Context and Market Positioning

Solana meme coins have established themselves as the best performing cryptocurrencies in the current bull cycle, with strong market momentum. Solana has provided meme coin enthusiasts with a cheaper, faster, and smoother way to create and trade tokens, with the Solana community actively supporting increased chain activity and providing a warm welcome to meme coins. TeosPump positions itself within this thriving ecosystem by focusing on culturally-themed tokens with Egyptian heritage, targeting the growing demand for meaningful meme coins beyond generic animal-themed tokens.

Current System Limitations

Meme coin trading on other chains has previously included high fees and slow, clunky token creation processes. The Solana blockchain offers significantly lower transaction fees than Ethereum, with approximately $0.005 per transaction on Solana compared to around $1 per transaction on Ethereum. However, existing launchpads lack integration with mobile mining rewards and cultural branding focus that TeosPump provides.

Integration with Existing Enterprise Landscape

The platform integrates with established Solana infrastructure including Phantom wallet's multi-chain capabilities supporting Solana, Ethereum, and Polygon, and leverages Token-2022 program extensions that provide more flexible and extensible token standards for complex tokenomics. The system connects with GitHub for version control and Vercel for automated deployment, ensuring seamless development and production workflows.

1.2.2 High-Level Description

Primary System Capabilities

Capability	Description	Technical Implementation
Token Creation	SPL token minting with custom metadata	Solana SPL Token Program integration
Payment Processing	$TEOS token-based fee collection	Smart contract automation

Capability	Description	Technical Implementation
Mobile Mining Integration	Reward distribution to mobile miners	Express.js API backend
Wallet Integration	Phantom wallet connectivity	@solana/web3.js library

Major System Components

The system comprises a Next.js frontend application with TypeScript and TailwindCSS, an Express.js backend API for mobile synchronization and token management, Solana blockchain integration for SPL token creation, and automated deployment through GitHub and Vercel infrastructure.

Core Technical Approach

The platform operates on the Solana blockchain using the SPL (Solana Program Library) token standard, functioning similarly to other cryptocurrencies while relying on decentralized technology to facilitate secure and fast transactions. The architecture follows a modern web3 stack with client-side wallet integration, server-side API management, and blockchain interaction through established Solana development tools.

1.2.3 Success Criteria

Measurable Objectives

Objective	Target Metric	Measurement Method
Token Launch Volume	100+ tokens created in first quarter	Platform analytics tracking
Transaction Processing	<2 second average confirmation time	Blockchain monitoring

Objective	Target Metric	Measurement Method
User Adoption	1,000+ active wallet connections	Wallet integration metrics
Revenue Generation	10+ SOL in platform fees monthly	Smart contract fee collection

Critical Success Factors

- Seamless Phantom wallet integration with zero-friction user experience
- Reliable SPL token creation with 99.9% success rate
- Secure $TEOS payment processing without transaction failures
- Responsive mobile mining reward distribution
- Stable platform performance during high-volume token launches

Key Performance Indicators (KPIs)

- Average time from wallet connection to token creation completion
- Platform uptime percentage during peak usage periods
- User retention rate for repeat token creators
- Mobile mining reward distribution accuracy
- Community engagement metrics on launched tokens

1.3 SCOPE

1.3.1 In-Scope

Core Features and Functionalities

Feature Category	Specific Capabilities
Wallet Integration	Phantom wallet connection, multi-chain support

Feature Category	Specific Capabilities
Token Creation	SPL token minting, metadata configuration, supply management
Payment System	$TEOS fee collection, SOL payment processing
Mobile Integration	Mining reward synchronization, mobile app backend API

Primary User Workflows

- User connects Phantom wallet to platform
- User configures token parameters (name, symbol, supply, metadata)
- User pays creation fee in $TEOS or SOL
- System mints SPL token on Solana blockchain
- Backend logs transaction and syncs with mobile application
- Mobile miners receive $TEOS rewards through API distribution

Essential Integrations

- Phantom wallet SDK for secure authentication and transaction signing
- Solana blockchain for SPL token creation and management
- GitHub repository for source code version control
- Vercel platform for automated deployment and hosting

Key Technical Requirements

The platform must support over 15 million potential active users with seamless interface management across various networks, providing streamlined user experience for both newcomers and experienced crypto veterans. The system requires real-time blockchain interaction, secure private key management, and reliable API endpoints for mobile synchronization.

1.3.2 Implementation Boundaries

System Boundaries

The platform operates within the Solana blockchain ecosystem, integrating with established wallet providers and deployment platforms. The system boundary includes frontend user interface, backend API services, blockchain smart contracts, and mobile application synchronization endpoints.

User Groups Covered

- Primary users: Token creators seeking to launch meme/cultural tokens
- Secondary users: Token investors participating in pre-sales
- Tertiary users: Mobile miners earning $TEOS rewards
- Administrative users: Platform operators managing system health

Geographic/Market Coverage

The platform targets global users with internet access and Solana wallet capabilities, with specific focus on Egyptian cultural themes and Elmahrosa International branding. No geographic restrictions apply for core functionality.

Data Domains Included

- User wallet addresses and transaction histories
- Token metadata including names, symbols, and supply information
- Payment records for $TEOS and SOL transactions
- Mobile mining reward distribution logs
- Platform analytics and usage metrics

1.3.3 Out-of-Scope

Explicitly Excluded Features/Capabilities

- Fiat currency payment processing or bank account integration

- Advanced DeFi features such as liquidity pool creation or yield farming
- NFT minting or marketplace functionality beyond basic token creation
- Multi-language localization beyond English interface
- Advanced trading features or order book management
- Governance token functionality or DAO implementation

Future Phase Considerations

- Integration with additional blockchain networks beyond Solana
- Advanced tokenomics features including vesting schedules and burn mechanisms
- Social features such as token creator profiles and community forums
- Analytics dashboard for token performance tracking
- Mobile application development for iOS and Android platforms

Integration Points Not Covered

- Traditional banking systems or payment processors
- Social media platforms for automated marketing
- Email marketing services or notification systems
- Third-party analytics platforms beyond basic usage tracking
- Customer support ticketing systems or live chat functionality

Unsupported Use Cases

- Enterprise-grade token launches requiring extensive compliance features
- High-frequency trading or algorithmic trading interfaces
- Institutional investor onboarding with KYC/AML requirements
- Cross-chain bridge functionality for token transfers
- Advanced security features such as multi-signature wallet support

2. PRODUCT REQUIREMENTS

2.1 FEATURE CATALOG

2.1.1 Core Platform Features

Feature ID	Feature Name	Category	Priority	Status
F-001	Phantom Wallet I ntegration	Authenticatio n	Critical	Propose d
F-002	SPL Token Creati on	Token Manag ement	Critical	Propose d
F-003	Payment Processi ng	Financial	Critical	Propose d
F-004	Mobile Mining Int egration	Rewards	High	Propose d

F-001: Phantom Wallet Integration

Description

- **Overview**: Phantom is a crypto wallet that can be used to manage digital assets and access decentralized applications on Solana, Bitcoin, Ethereum, Base, and Polygon. To interact with web applications, the Phantom extension and mobile in-app browser injects a phantom object into the javascript context of every site the user visits. A given web app may then interact with Phantom, and ask for the user's permission to perform transactions, through this injected provider.
- **Business Value**: Enables secure user authentication and transaction signing without requiring users to manage private keys directly
- **User Benefits**: Seamless wallet connection with industry-standard security practices and multi-chain support
- **Technical Context**: Phantom offers multichain compatibility (Solana, Ethereum, Bitcoin, Polygon, Base, Sui, among others), enabling users to manage a diverse portfolio of digital assets within a single wallet interface. Phantom features an integrated built-in swap function that

allows users to exchange one cryptocurrency for another directly within the wallet.

Dependencies

- **Prerequisite Features**: None (foundational feature)
- **System Dependencies**: Next.js 14+ frontend framework, TypeScript support
- **External Dependencies**: @solana/web3.js library, Phantom wallet browser extension
- **Integration Requirements**: Browser-based wallet detection and connection protocols

F-002: SPL Token Creation

Description

- **Overview**: Tokens on Solana are referred to as SPL (Solana Program Library) Tokens. A Mint Account represents a specific token and stores global metadata about the token such as the total supply and mint authority (address authorized to create new units of a token). The MintTo instruction on the Token Program creates new tokens. The mint authority must sign the transaction. The instruction mints tokens to a Token Account and increases the total supply on the Mint Account.
- **Business Value**: Core revenue-generating feature enabling token creators to launch projects on Solana blockchain
- **User Benefits**: Simple token creation process with customizable metadata and supply management
- **Technical Context**: Creating tokens and accounts requires SOL for account rent deposits and transaction fees. Creating a new Mint Account requires a transaction with two instructions.

Dependencies

- **Prerequisite Features**: F-001 (Phantom Wallet Integration)

- **System Dependencies**: Solana blockchain connection, SPL Token Program
- **External Dependencies**: @solana/web3.js, @solana/spl-token libraries
- **Integration Requirements**: Solana RPC endpoint configuration, transaction signing capabilities

F-003: Payment Processing

Description

- **Overview**: Fee collection system using $TEOS tokens and SOL for token creation services
- **Business Value**: Revenue generation through platform fees (0.1 SOL + 2.5% of sales)
- **User Benefits**: Transparent fee structure with multiple payment options
- **Technical Context**: Integration with existing $TEOS token contract (AhXBUQmbhv9dNoZCiMYmXF4Gyi1cjQthWHFhTL2CJaSo)

Dependencies

- **Prerequisite Features**: F-001 (Phantom Wallet Integration), F-002 (SPL Token Creation)
- **System Dependencies**: Smart contract interaction capabilities
- **External Dependencies**: $TEOS token contract, Solana blockchain
- **Integration Requirements**: Token transfer instructions, fee calculation logic

F-004: Mobile Mining Integration

Description

- **Overview**: Backend API system for synchronizing mobile mining rewards with web platform

- **Business Value**: User retention through reward mechanisms and mobile app integration
- **User Benefits**: Seamless reward distribution and cross-platform synchronization
- **Technical Context**: Express.js backend API handling mobile synchronization and $TEOS distribution

Dependencies

- **Prerequisite Features**: F-003 (Payment Processing)
- **System Dependencies**: Express.js backend, database for transaction logging
- **External Dependencies**: Mobile application API endpoints
- **Integration Requirements**: RESTful API design, authentication mechanisms

2.2 FUNCTIONAL REQUIREMENTS TABLE

2.2.1 F-001: Phantom Wallet Integration Requirements

Requirement ID	Description	Acceptance Criteria	Priority	Complexity
F-001-RQ-001	Wallet Connection	User can connect Phantom wallet to platform	Must-Have	Medium
F-001-RQ-002	Multi-chain Support	Support Solana, Ethereum, Bitcoin networks	Should-Have	High
F-001-RQ-003	Transaction Signing	Enable secure transaction signing through wallet	Must-Have	Medium

Requirement ID	Description	Acceptance Criteria	Priority	Complexity
F-001-RQ-004	Wallet Disconnection	Allow users to disconnect wallet safely	Must-Have	Low

Technical Specifications

- **Input Parameters**: Wallet connection request, transaction data for signing
- **Output/Response**: Wallet address, signed transaction objects, connection status
- **Performance Criteria**: <2 second connection time, 99.9% transaction signing success rate
- **Data Requirements**: Wallet address storage, transaction history logging

Validation Rules

- **Business Rules**: Only verified Phantom wallet connections accepted
- **Data Validation**: Valid Solana address format verification
- **Security Requirements**: Security features such as encryption, biometric authentication and hardware wallet integration are provided, but users must safeguard their secret recovery phrase to prevent unauthorized access. The wallet employs advanced encryption techniques to protect private keys and offers features like biometric authentication on mobile devices. Additionally, Phantom integrates with hardware wallets such as Ledger, providing an extra layer of security by keeping private keys offline.
- **Compliance Requirements**: Non-custodial wallet standards compliance

2.2.2 F-002: SPL Token Creation Requirements

Requirement ID	Description	Acceptance Criteria	Priority	Complexity
F-002-RQ-001	Token Metadata Configuration	Configure name, symbol, decimals, supply	Must-Have	Medium
F-002-RQ-002	Mint Account Creation	Create SPL token mint on Solana blockchain	Must-Have	High
F-002-RQ-003	Token Account Management	Handle associated token accounts	Must-Have	High
F-002-RQ-004	Metadata Upload	Store token metadata on-chain or IPFS	Should-Have	Medium

Technical Specifications

- **Input Parameters**: Token name, symbol, decimals (0-9), total supply, metadata URI
- **Output/Response**: Mint address, transaction signature, token account addresses
- **Performance Criteria**: Fast Transaction Speeds: Solana's architecture allows for fast transaction speeds, processing thousands of transactions per second. Low Fees: The low fees associated with transactions on Solana make it economically viable for token creation.
- **Data Requirements**: Token registry, metadata storage, mint authority management

Validation Rules

- **Business Rules**: The supply is one of the most important parts, as we have to organize the financial landscape that the SPL token will have. This decision is closely linked to the tokenomics of your project, in fact, we have a guide on how to design a successful tokenomics. Other than that, a supply of 1 B 0 10 B is the most normal, but it all depends on the specifications you want for your project.

- **Data Validation**: Token name length (1-32 characters), symbol length (1-10 characters), valid supply range
- **Security Requirements**: Mint authority verification, secure metadata handling
- **Compliance Requirements**: SPL token standard compliance

2.2.3 F-003: Payment Processing Requirements

Requirement ID	Description	Acceptance Criteria	Priority	Complexity
F-003-RQ-001	$TEOS Fee Collection	Accept $TEOS tokens for platform fees	Must-Have	Medium
F-003-RQ-002	SOL Fee Collection	Accept SOL for alternative payment method	Must-Have	Medium
F-003-RQ-003	Fee Calculation	Calculate 0.1 SOL + 2.5% sales commission	Must-Have	Low
F-003-RQ-004	Payment Verification	Verify payment completion before token creation	Must-Have	Medium

Technical Specifications

- **Input Parameters**: Payment amount, payment token type, recipient address
- **Output/Response**: Payment confirmation, transaction signature, fee breakdown
- **Performance Criteria**: <1 second fee calculation, 99.9% payment processing success
- **Data Requirements**: Fee structure configuration, payment history, owner wallet address

Validation Rules

- **Business Rules**: Minimum fee requirements, owner wallet verification (Akvm3CbDN448fyD8qmQjowgBGpcYZtjuKFL4xT8PZhbF)
- **Data Validation**: Sufficient balance verification, valid payment amounts
- **Security Requirements**: Payment atomicity, double-spending prevention
- **Compliance Requirements**: Financial transaction logging requirements

2.2.4 F-004: Mobile Mining Integration Requirements

Requirement ID	Description	Acceptance Criteria	Priority	Complexity
F-004-RQ-001	API Endpoint Creation	RESTful endpoints for mobile synchronization	Must-Have	Medium
F-004-RQ-002	Reward Distribution	Send $TEOS rewards to mobile miners	Should-Have	High
F-004-RQ-003	Transaction Logging	Log all mobile-related transactions	Must-Have	Low
F-004-RQ-004	Sync Status Tracking	Track synchronization status with mobile app	Should-Have	Medium

Technical Specifications

- **Input Parameters**: Mobile user ID, reward amount, wallet address
- **Output/Response**: API response status, transaction confirmation, sync status
- **Performance Criteria**: <500ms API response time, 99.5% uptime

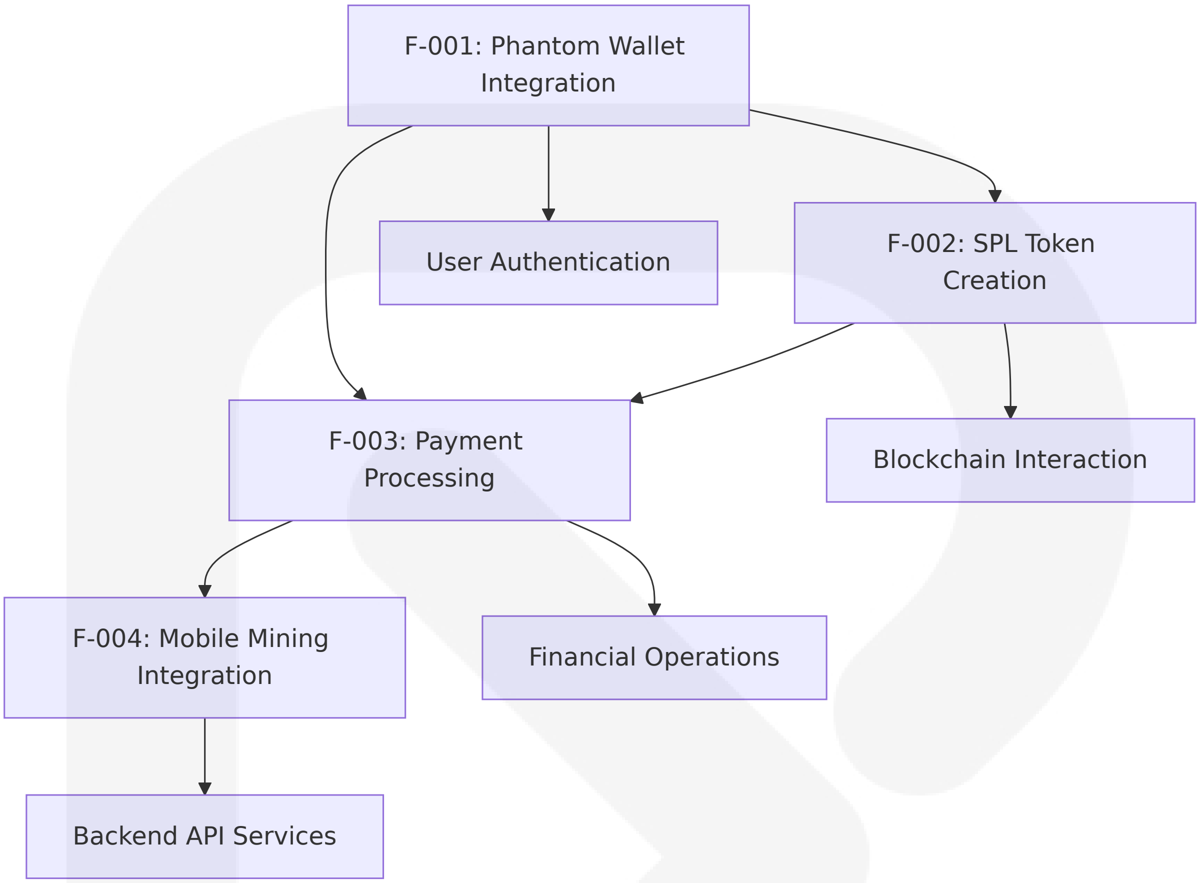
- **Data Requirements**: Mobile user registry, reward calculation logic, sync logs

Validation Rules

- **Business Rules**: Verified mobile mining activity, reward eligibility criteria
- **Data Validation**: Valid mobile user authentication, reward amount limits
- **Security Requirements**: API authentication, rate limiting, secure data transmission
- **Compliance Requirements**: User data privacy, reward distribution transparency

2.3 FEATURE RELATIONSHIPS

2.3.1 Feature Dependencies Map



2.3.2 Integration Points

Integration Point	Features Involved	Description	Technical Requirements
Wallet-Token Creation	F-001, F-002	Wallet signs token creation transactions	Transaction signing, account verification
Payment-Token Creation	F-002, F-003	Payment verification before token minting	Atomic transaction processing
Mobile-Payment Sync	F-003, F-004	Reward distribution through payment system	API integration, database synchronization

2.3.3 Shared Components

Component	Features Using	Purpose	Implementation
Solana Connection	F-001, F-002, F-003	Blockchain interaction	@solana/web3.js Connection class
Transaction Builder	F-002, F-003	Transaction construction	Custom transaction utility functions
Error Handling	All Features	Consistent error management	Global error handling middleware
Logging System	All Features	Activity tracking	Winston or similar logging library

2.3.4 Common Services

Service	Description	Features Served	Technical Stack
Authentication Service	User wallet verification	F-001, F-004	JWT tokens, wallet signature verification
Blockchain Service	Solana network interaction	F-001, F-002, F-003	@solana/web3.js, RPC endpoints
Database Service	Data persistence	F-003, F-004	PostgreSQL or MongoDB
API Service	Backend endpoints	F-004	Express.js, RESTful design

2.4 IMPLEMENTATION CONSIDERATIONS

2.4.1 Technical Constraints

Feature	Constraints	Impact	Mitigation Strategy
F-001	Browser wallet dependency	Limited to Phantom-compatible browsers	Provide clear browser requirements
F-002	Creating tokens and accounts requires SOL for account rent deposits and transaction fees	Users need SOL for gas fees	Implement fee estimation, user guidance
F-003	$TEOS token contract dependency	Reliance on external token contract	Contract verification, fallback mechanisms
F-004	Mobile app integration	Cross-platform synchronization complexity	Robust API design, error handling

2.4.2 Performance Requirements

Feature	Performance Metric	Target	Monitoring Method
F-001	Wallet connection time	<2 seconds	Frontend performance monitoring
F-002	Token creation time	<10 seconds	Blockchain transaction tracking
F-003	Payment processing	<5 seconds	Transaction confirmation monitoring
F-004	API response time	<500ms	Backend performance metrics

2.4.3 Scalability Considerations

Feature	Scalability Factor	Current Limit	Scaling Strategy
F-001	Concurrent wallet connectio	Browser limitations	Connection pooling, rate limiting

Feature	Scalability Factor	Current Limit	Scaling Strategy
	ns		
F-002	Token creation volume	Solana can handle thousands of transactions per second	Batch processing, queue management
F-003	Payment processing	Network throughput	Transaction batching, retry mechanisms
F-004	Mobile API requests	Server capacity	Horizontal scaling, load balancing

2.4.4 Security Implications

Feature	Security Risk	Mitigation	Implementation
F-001	Wallet compromise	Non-custodial design	Phantom is a non-custodial wallet, meaning you control your private keys. As a non-custodial wallet, you control your private keys, which are never shared or stored on a central server.
F-002	Unauthorized token creation	Payment verification	Atomic transaction processing
F-003	Payment fraud	Transaction verification	Multi-signature validation
F-004	API security	Authentication & authorization	JWT tokens, rate limiting

2.4.5 Maintenance Requirements

Feature	Maintenance Task	Frequency	Responsibility
F-001	Wallet SDK updates	Monthly	Frontend team
F-002	Solana network compatibility	Quarterly	Blockchain team
F-003	Fee structure updates	As needed	Business team
F-004	API endpoint maintenance	Weekly	Backend team

2.5 TRACEABILITY MATRIX

Business Requirement	Feature ID	Functional Requirement	Test Case	Priority
User wallet connection	F-001	F-001-RQ-001	TC-001-001	Critical
Token creation capability	F-002	F-002-RQ-001, F-002-RQ-002	TC-002-001, TC-002-002	Critical
Fee collection system	F-003	F-003-RQ-001, F-003-RQ-002	TC-003-001, TC-003-002	Critical
Mobile reward integration	F-004	F-004-RQ-001, F-004-RQ-002	TC-004-001, TC-004-002	High
Egyptian cultural branding	UI/UX	Design requirements	TC-UI-001	Medium
GitHub/Vercel deployment	DevOps	Deployment pipeline	TC-DEV-001	High

3. TECHNOLOGY STACK

3.1 PROGRAMMING LANGUAGES

3.1.1 Frontend Languages

Language	Version	Platform	Justification
TypeScript	5.8.3	Frontend/Backend	Latest stable version providing enhanced type safety, improved developer experience, and modern JavaScript features
JavaScript	ES2022+	Runtime	TypeScript compiles to JavaScript which runs anywhere JavaScript runs: In a browser, on Node.js, Deno, Bun and in your apps

3.1.2 Backend Languages

Language	Version	Platform	Justification
TypeScript	5.8.3	Node.js Backend	Consistent language across frontend and backend for shared type definitions and reduced context switching
JavaScript	ES2022+	Node.js Runtime	Express.js latest version 5.1.0 provides fast, unopinionated, minimalist web framework capabilities

3.1.3 Selection Criteria

Type Safety Requirements

- TypeScript extends JavaScript by adding types to the language and speeds up development experience by catching errors and providing fixes before running code

- Critical for blockchain applications where transaction errors can result in financial losses
- Enhanced IDE support with autocomplete and refactoring capabilities

Ecosystem Compatibility

- Solana web3.js latest version 1.98.2 with 4056 other projects using the library
- Phantom wallet integration requires JavaScript/TypeScript compatibility
- Next.js framework built specifically for TypeScript development

Performance Considerations

- TypeScript Native Previews achieving 10x speed-up on most projects through native compilation and shared memory parallelism
- Modern JavaScript features including BigInt support essential for Solana's u64 number handling

3.2 FRAMEWORKS & LIBRARIES

3.2.1 Frontend Framework

Framework	Version	Purpose	Justification
Next.js	14.2+	React Framework	Next.js 14.2 includes development, production, and caching improvements with 99.8% Turbopack tests passing for next dev --turbo
React	18+	UI Library	Integrated with Next.js for component-based architecture
TailwindCSS	4.1.11	CSS Framework	Latest version providing utility-first CSS framework for rapidly building custom user interfaces

3.2.2 Backend Framework

Framework	Version	Purpose	Justification
Express.js	5.1.0	Web Framework	Express v5 release designed to be boring but unblocks the ecosystem and enables more impactful changes in future releases
Node.js	18+	Runtime Environment	Express v5 dropped support for Node.js versions before v18

3.2.3 Blockchain Integration Libraries

Library	Version	Purpose	Justification
@solana/web3.js	1.98.2	Solana Blockchain	Maintenance branch for 1.x line with successor @solana/kit available
@solana/spl-token	Latest	Token Operations	SPL token creation and management functionality

3.2.4 Compatibility Requirements

Next.js 14.2 Features

- Build and production improvements with reduced build memory usage and CSS optimizations, plus configurable invalidation periods with staleTimes
- Turbopack Release Candidate for improved development performance
- Enhanced error handling and developer experience improvements

TailwindCSS 4.x Architecture

- Ground-up rewrite optimized for performance with full builds up to 5x faster and incremental builds over 100x faster
- Simplified installation with fewer dependencies, zero configuration, and automatic content detection

- Modern CSS features including cascade layers and registered custom properties

Express.js 5.x Security Enhancements

- CVE-2024-45590 mitigation with customizable urlencoded body depth defaulting to 32
- Updated to path-to-regexp@8.x removing sub-expression regex patterns for ReDoS mitigation and promise support for middleware

3.3 OPEN SOURCE DEPENDENCIES

3.3.1 Core Dependencies

Package	Version	Registry	Purpose
next	^14.2.0	npm	React framework with SSR/SSG capabilities
react	^18.0.0	npm	UI component library
react-dom	^18.0.0	npm	React DOM rendering
typescript	^5.8.3	npm	Type checking and compilation
tailwindcss	^4.1.11	npm	Utility-first CSS framework
express	^5.1.0	npm	Backend web framework

3.3.2 Blockchain Dependencies

Package	Version	Registry	Purpose
@solana/web3.js	^1.98.2	npm	Solana blockchain interaction
@solana/spl-token	^0.4.8	npm	SPL token operations

Package	Version	Registry	Purpose
@solana/wallet-adapter-react	^0.15.35	npm	Wallet integration utilities
@solana/wallet-adapter-phantom	^0.9.24	npm	Phantom wallet specific adapter

3.3.3 Development Dependencies

Package	Version	Registry	Purpose
@types/node	^20.0.0	npm	Node.js type definitions
@types/react	^18.0.0	npm	React type definitions
@types/react-dom	^18.0.0	npm	React DOM type definitions
eslint	^8.0.0	npm	Code linting and quality
prettier	^3.0.0	npm	Code formatting

3.3.4 Version Management Strategy

Semantic Versioning Compliance

- All dependencies follow semantic versioning for predictable updates
- Major version updates require explicit testing and validation
- Security patches applied automatically for patch versions

Dependency Security

- Recent @solana/web3.js security incident with compromised publish-access account requiring upgrade to version 1.95.8
- Regular security audits using npm audit and Dependabot
- Automated dependency updates for security patches

Package Registry Strategy

- Primary registry: npm (npmjs.com)
- Backup registry configuration for high availability
- Package-lock.json committed for reproducible builds

3.4 THIRD-PARTY SERVICES

3.4.1 Blockchain Infrastructure

Service	Purpose	Integration Method	Justification
Solana RPC	Blockchain connectivity	@solana/web3.js Connection	BigInt support for handling large numbers accurately, important for Solana programming as Rust's u64 type can represent numbers up to 2^64-1
Phantom Wallet	User authentication	Browser extension injection	Phantom wallet supports multichain compatibility (Solana, Ethereum, Bitcoin, Polygon, Base, Sui) with integrated built-in swap function

3.4.2 Development Services

Service	Purpose	Integration Method	Justification
GitHub	Version control	Git repository	Source code management and collaboration
Vercel	Deployment platform	Git repository integration with automatic deployment triggers on commits and pull requests	Ease of use with intuitive platform, free tier for personal projects, automatic SSL certificates, and automatic scalability handling

3.4.3 Token Services

Service	Purpose	Integration Method	Justification
$TEOS Token Contract	Payment processing	SPL token interaction	Contract address: AhXBUQmbhv9dNoZCiMYmXF4Gyi1cjQthWHFhTL2CJaSo
Owner Wallet	Fee collection	Direct SOL transfers	Wallet address: Akvm3CbDN448fyD8qmQjowgBGpcYZtjuKFL4xT8PZhbF

3.4.4 Integration Requirements

Vercel Deployment Pipeline

- Three default environments: Local Development for testing code changes, Preview for testing/QA/collaboration, and Production for final user-facing deployment
- Automatic SSL certificate provisioning
- Environment variable management for sensitive configuration
- Build optimization and caching strategies

GitHub Integration

- Automated deployment triggers on main branch commits
- Pull request preview deployments
- Issue tracking and project management
- Dependabot security updates

3.5 DATABASES & STORAGE

3.5.1 Primary Data Storage

Storage Type	Technology	Purpose	Justification
Blockchain	Solana	Token metadata and transactions	Immutable, decentralized storage for financial data
File System	Vercel Edge	Static assets and build artifacts	Static assets (HTML, CSS, JS) with size tracking and function deployment with type, runtime, size, and region information

3.5.2 Temporary Storage

Storage Type	Technology	Purpose	Justification
Memory	Node.js Runtime	Session data and caching	Ephemeral data for API request processing
Local Storage	Browser	User preferences	Client-side settings persistence

3.5.3 Data Persistence Strategy

Blockchain-First Architecture

- All financial transactions stored on Solana blockchain
- Token creation records immutably stored via SPL token program
- Payment records tracked through blockchain transaction history
- No traditional database required for core functionality

Stateless Application Design

- Backend API designed as stateless microservices
- No persistent server-side session storage
- Client-side state management through React hooks
- Wallet connection state managed by browser extension

3.5.4 Caching Solutions

Next.js Built-in Caching

- Caching improvements with configurable invalidation periods using staleTimes
- Static page generation for improved performance
- API route caching for blockchain data

Browser Caching

- Service worker implementation for offline capability
- Local storage for user preferences and settings
- IndexedDB for complex client-side data structures

3.6 DEVELOPMENT & DEPLOYMENT

3.6.1 Development Tools

Tool	Version	Purpose	Justification
Visual Studio Code	Latest	IDE	VS Code ships with recent stable TypeScript language service and provides IntelliSense without worry for most common cases
Git	2.40+	Version control	Distributed version control system
Node.js	18+	Runtime environment	Express v5 requires Node.js v18+ minimum
npm	9+	Package manager	Dependency management and script execution

3.6.2 Build System

Component	Technology	Purpose	Justification
TypeScript Compiler	tsc 5.8.3	Type checking	TypeScript 5.8 avoids re-validating options when edits don't change fundamental project structure, making edits in large projects feel more responsive
Next.js Build	Built-in	Application bundling	Build and production improvements with reduced build memory usage and CSS optimizations
TailwindCSS	PostCSS	CSS processing	Incremental builds over 100x faster completing in microseconds for previously used classes

3.6.3 Deployment Pipeline

Stage	Platform	Process	Configuration
Development	Local	Hot reload development server	Next.js dev server with Turbopack
Preview	Vercel	Automatic deployment on pull requests	Branch-based preview URLs
Production	Vercel	Automatic deployment on main branch commits	Custom domain with SSL

3.6.4 CI/CD Requirements

Automated Testing

- TypeScript compilation validation
- ESLint code quality checks
- Prettier code formatting verification
- Build process validation

Deployment Automation

- GitHub Actions CI/CD service for automating workflows including building, testing, and deploying code with automatic triggers on repository changes
- Environment variable injection
- Build artifact optimization
- Rollback capabilities for failed deployments

Quality Assurance

- Pre-commit hooks for code quality
- Automated dependency vulnerability scanning
- Performance monitoring and alerting
- Error tracking and logging

3.6.5 Infrastructure as Code

Vercel Configuration

```
{
  "framework": "nextjs",
  "buildCommand": "npm run build",
  "outputDirectory": ".next",
  "installCommand": "npm install",
  "devCommand": "npm run dev"
}
```

Environment Management

- Development: Local environment variables
- Preview: Branch-specific configuration
- Production: Secure environment variable storage

Monitoring and Observability

- Vercel dashboard integration with Logs, Analytics, Speed Insights, and Observability tabs

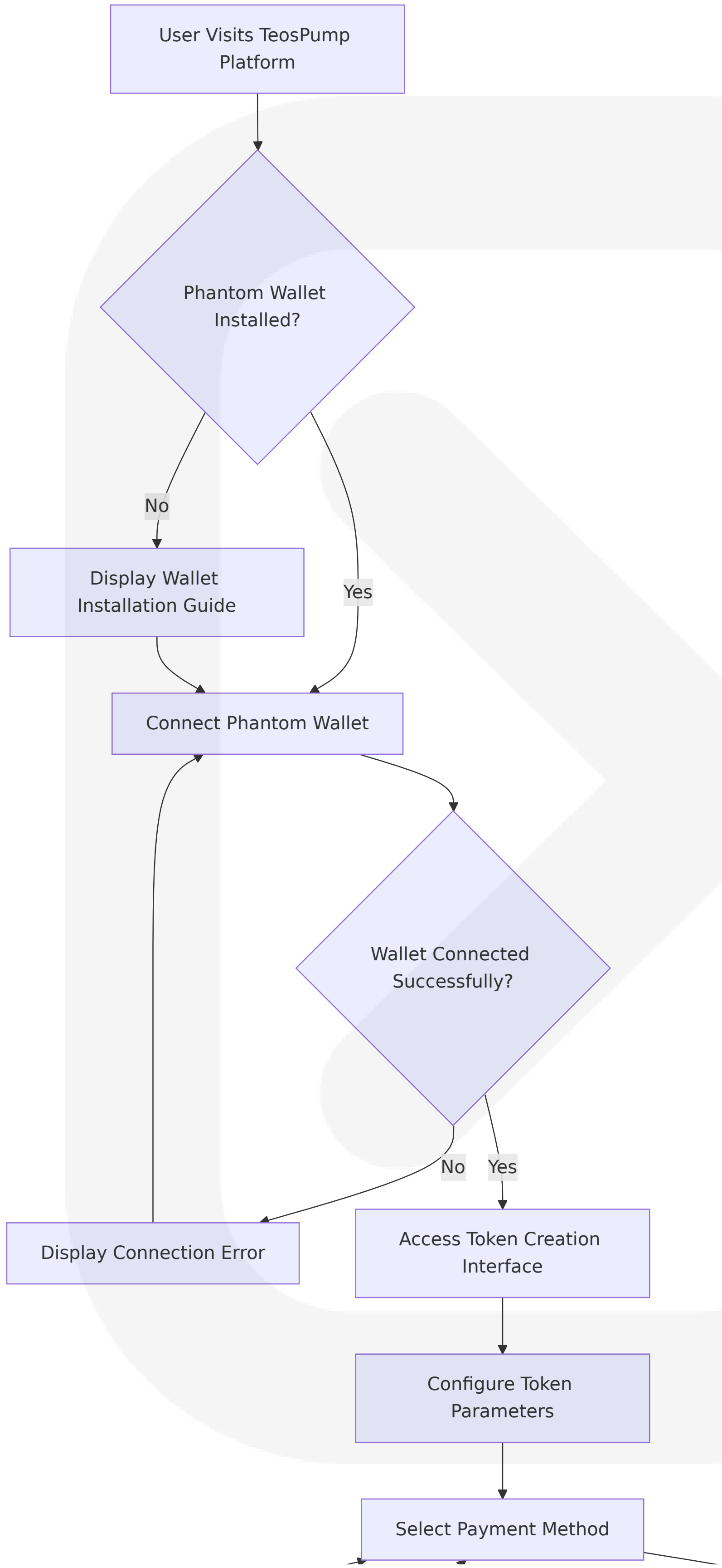
- Real-time deployment status monitoring
- Performance metrics and error tracking
- User analytics and usage patterns

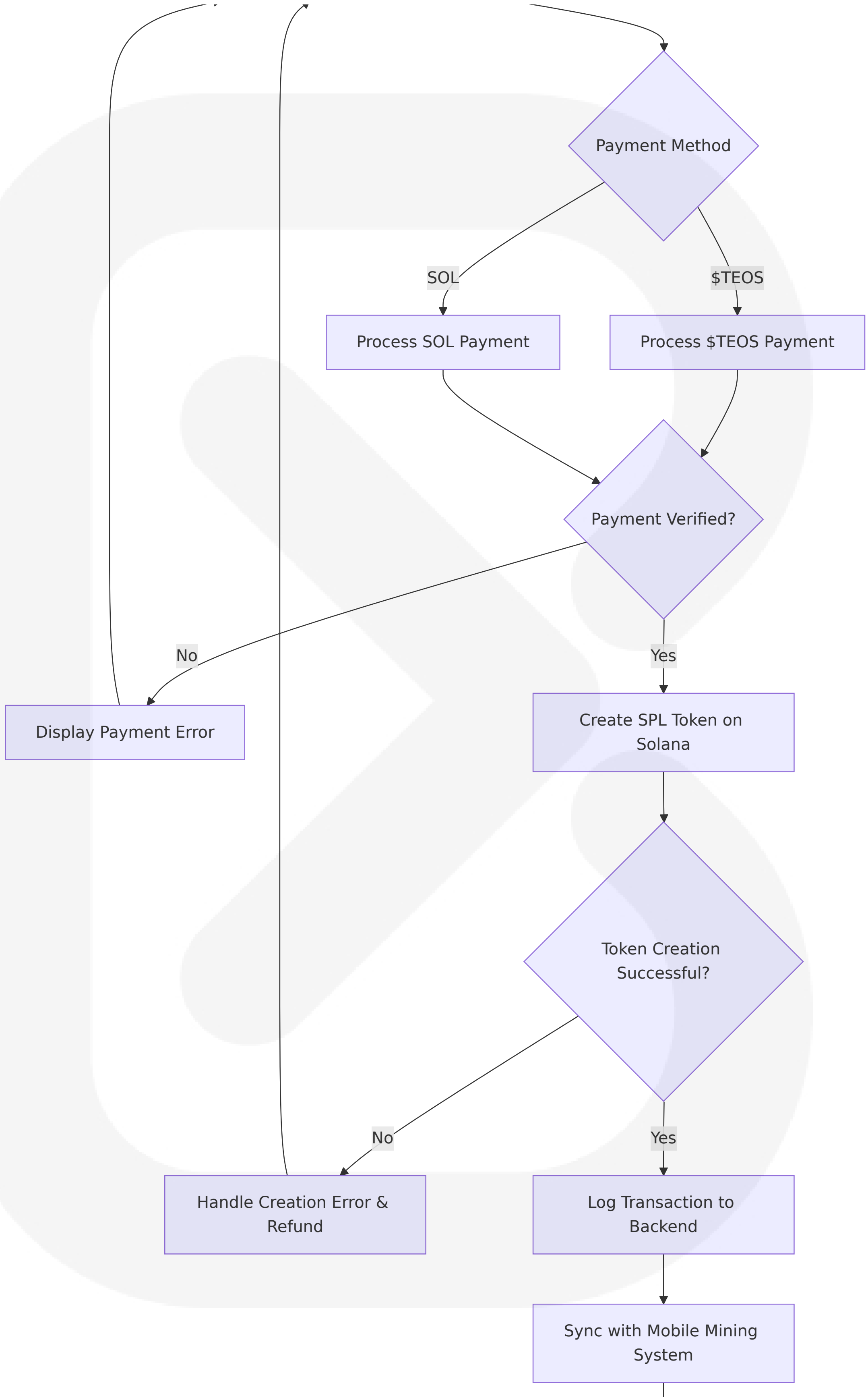
4. PROCESS FLOWCHART

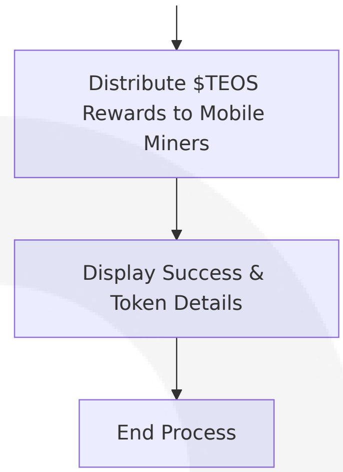
4.1 SYSTEM WORKFLOWS

4.1.1 Core Business Processes

High-Level System Workflow

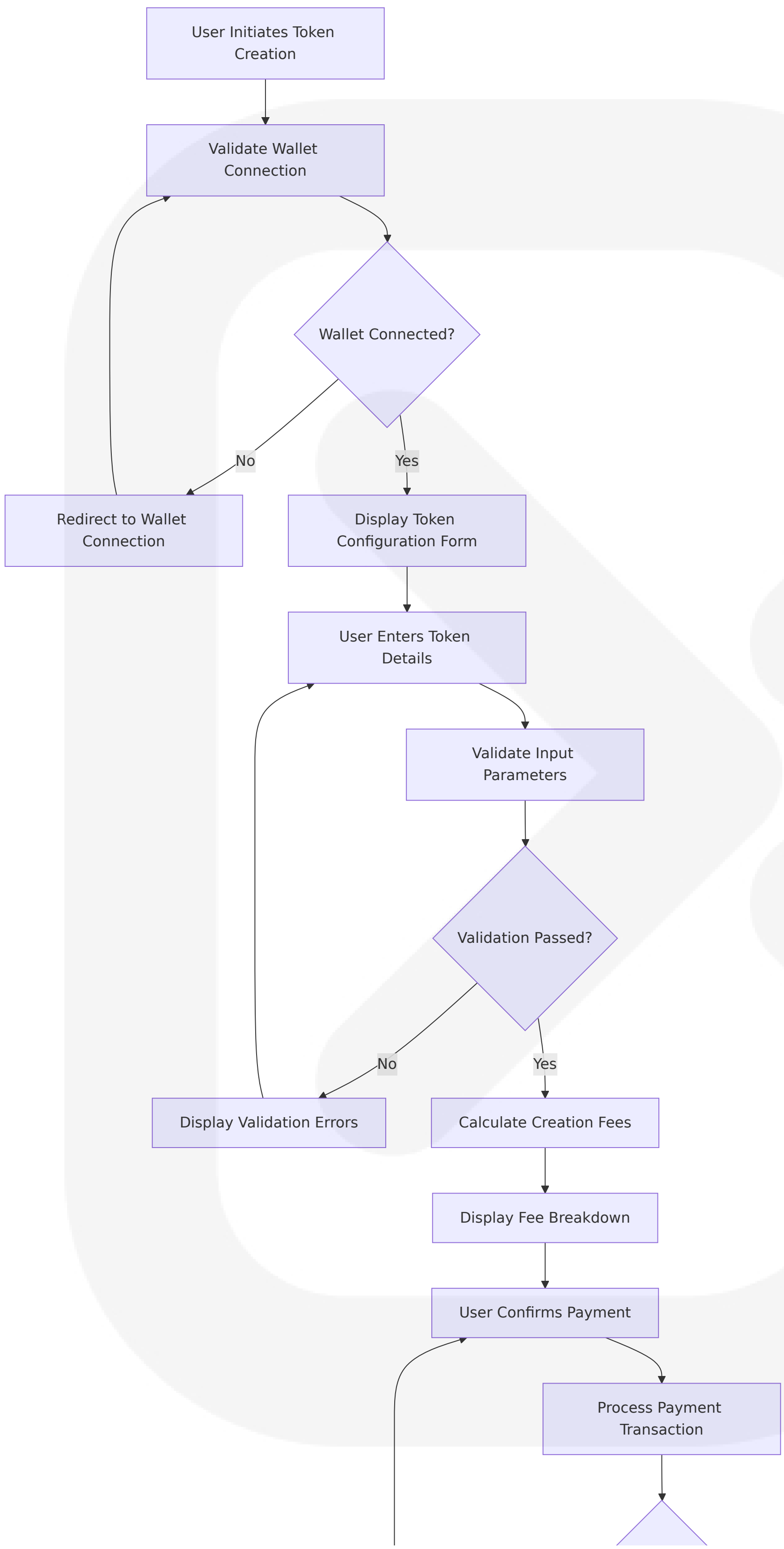

User Visits TeosPump Platform
Phantom Wallet Installed?
No
Yes
Display Wallet Installation Guide
Connect Phantom Wallet
Wallet Connected Successfully?
No
Yes
Display Connection Error
Access Token Creation Interface
Configure Token Parameters
Select Payment Method

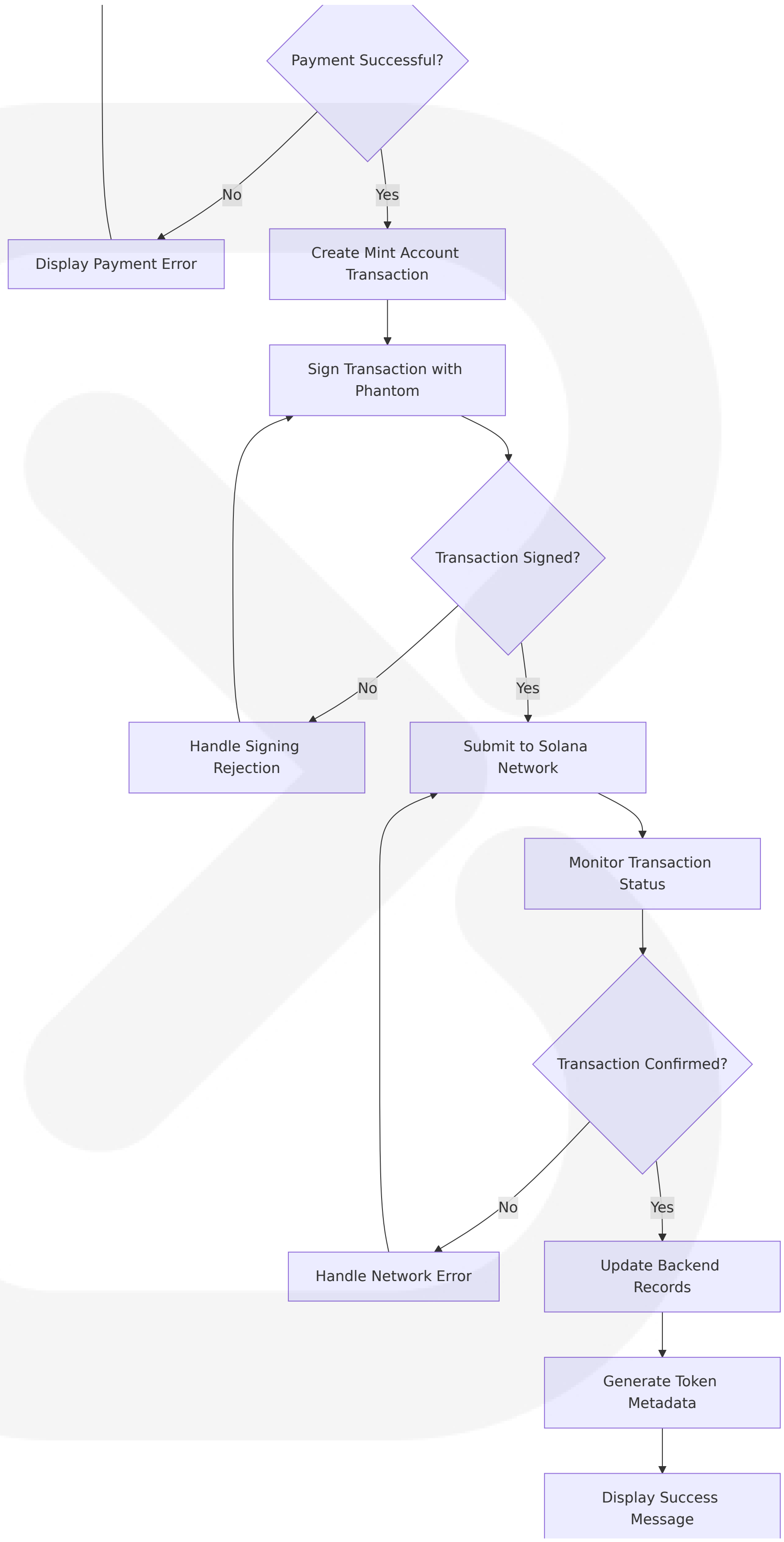

Payment Method
SOL
$TEOS
Process SOL Payment
Process $TEOS Payment
Payment Verified?
No
Yes
Display Payment Error
Create SPL Token on Solana
Token Creation Successful?
No
Yes
Handle Creation Error & Refund
Log Transaction to Backend
Sync with Mobile Mining System



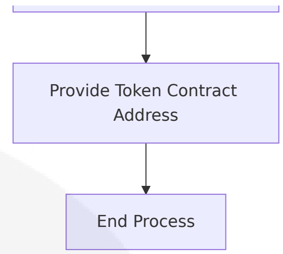
Token Creation End-to-End User Journey

Creating a new Mint Account requires a transaction with two instructions. The System Program creates a new account with space for the Mint Account data and transfers ownership to the Token Program. The Token Program initializes the data of the new account as a Mint Account


User Initiates Token Creation
Validate Wallet Connection
Wallet Connected?
No
Yes
Redirect to Wallet Connection
Display Token Configuration Form
User Enters Token Details
Validate Input Parameters
Validation Passed?
No
Yes
Display Validation Errors
Calculate Creation Fees
Display Fee Breakdown
User Confirms Payment
Process Payment Transaction

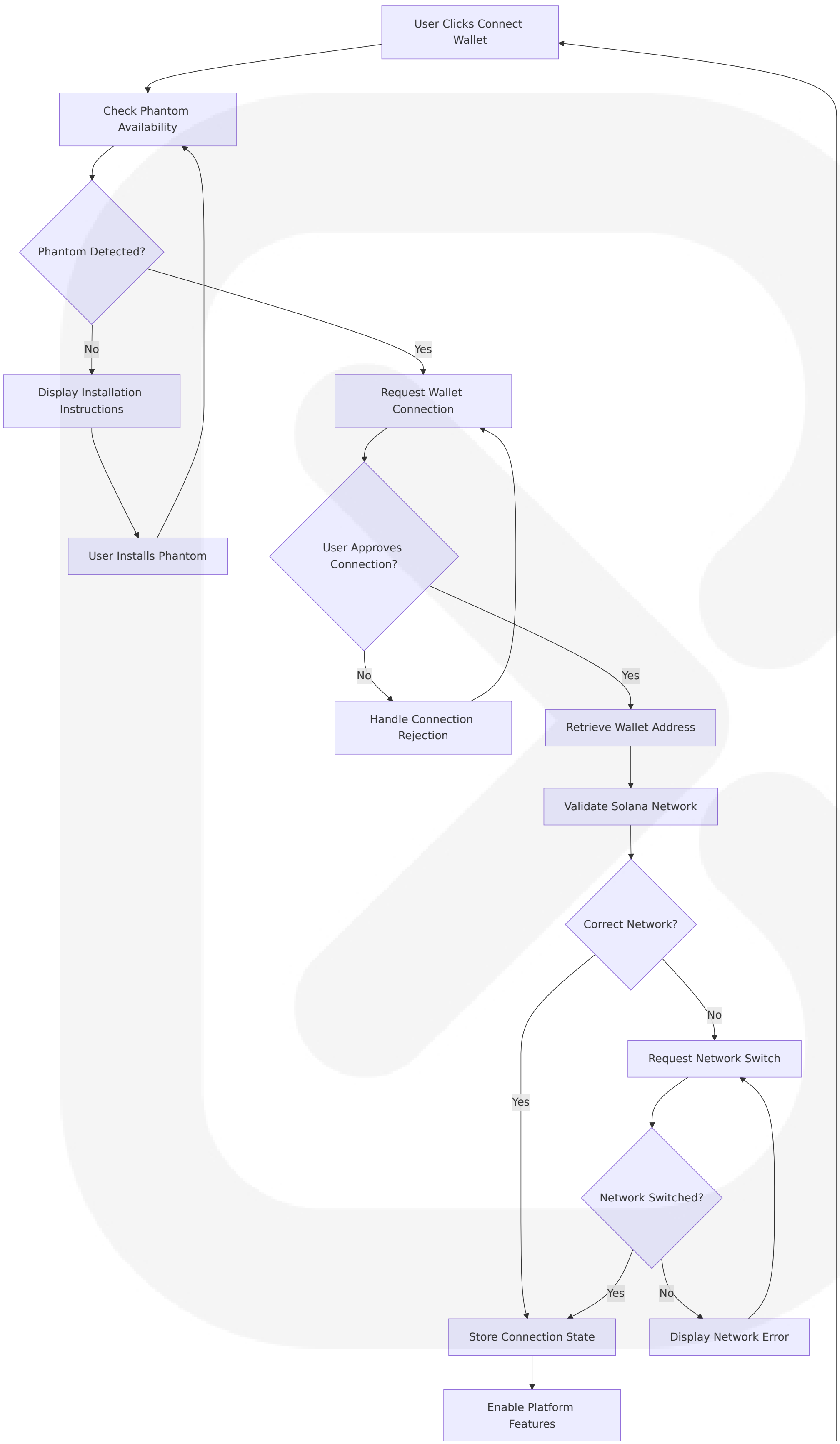


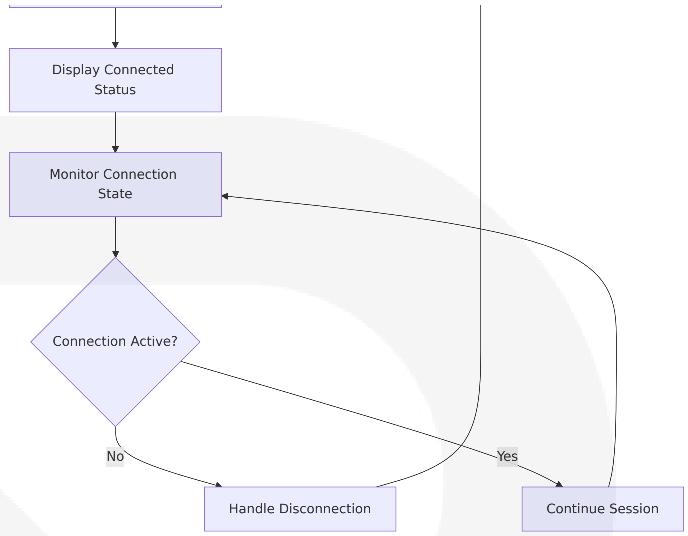
Payment Successful?
No
Yes
Display Payment Error
Create Mint Account Transaction
Sign Transaction with Phantom
Transaction Signed?
No
Yes
Handle Signing Rejection
Submit to Solana Network
Monitor Transaction Status
Transaction Confirmed?
No
Yes
Handle Network Error
Update Backend Records
Generate Token Metadata
Display Success Message



Phantom Wallet Integration Workflow

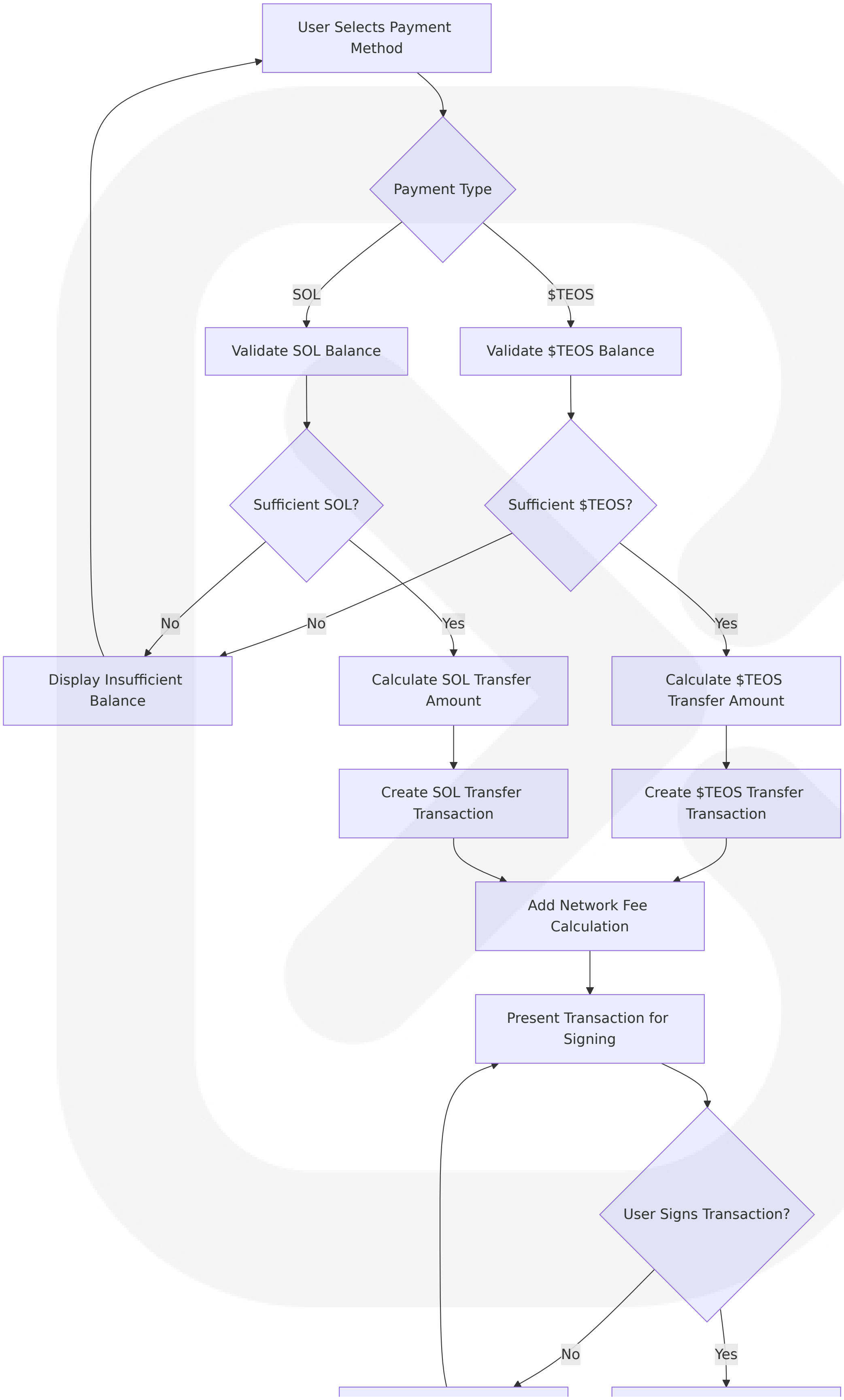
At its core, Phantom works by creating and managing private keys on behalf of its users. These keys can then be used within Phantom to store funds and sign transactions. Developers can interact with Phantom via both web applications as well as iOS and Android applications.

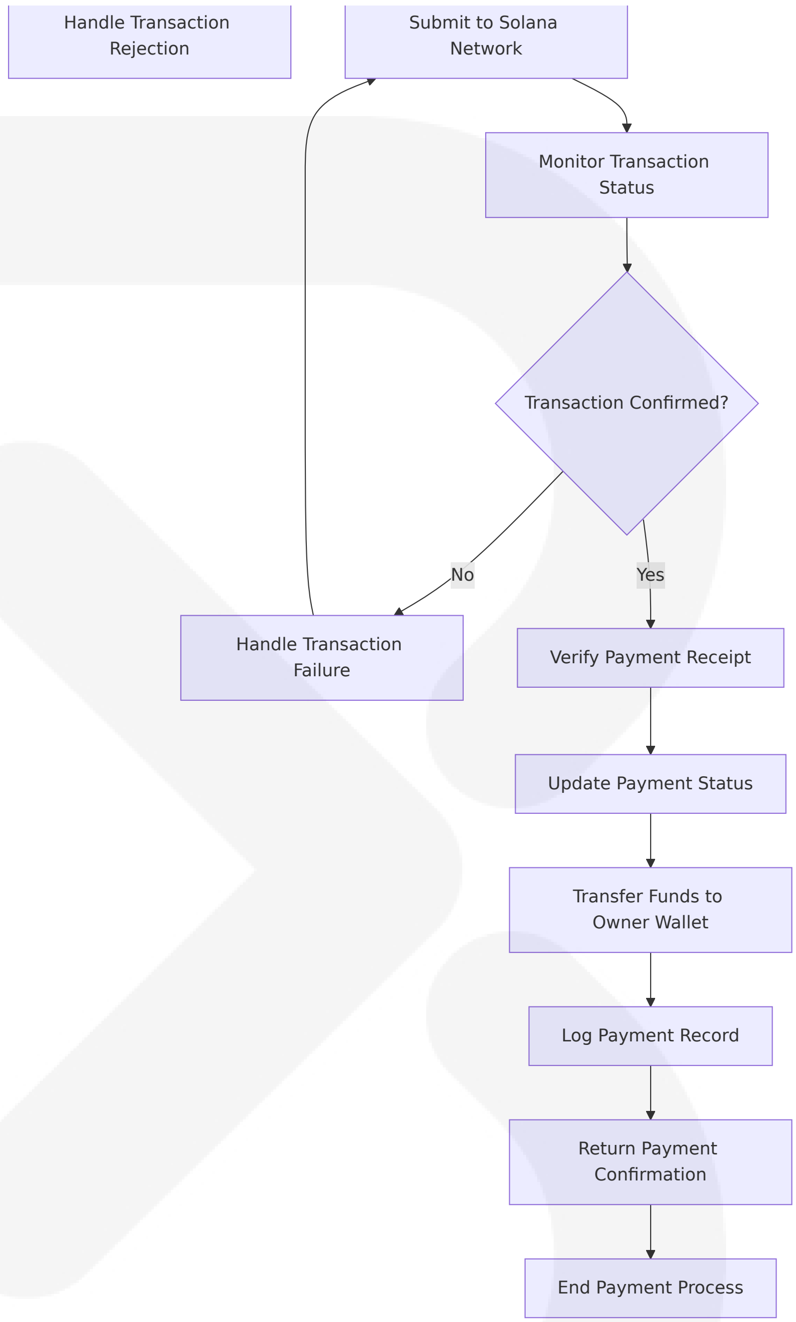

User Clicks Connect Wallet
Check Phantom Availability
Phantom Detected?
No
Yes
Display Installation Instructions
Request Wallet Connection
User Installs Phantom
User Approves Connection?
No
Yes
Handle Connection Rejection
Retrieve Wallet Address
Validate Solana Network
Correct Network?
No
Yes
Request Network Switch
Network Switched?
Yes
No
Store Connection State
Display Network Error
Enable Platform Features



Payment Processing Workflow

Every transaction involving SPL tokens comes with a network fee which is paid in the native Solana coin.

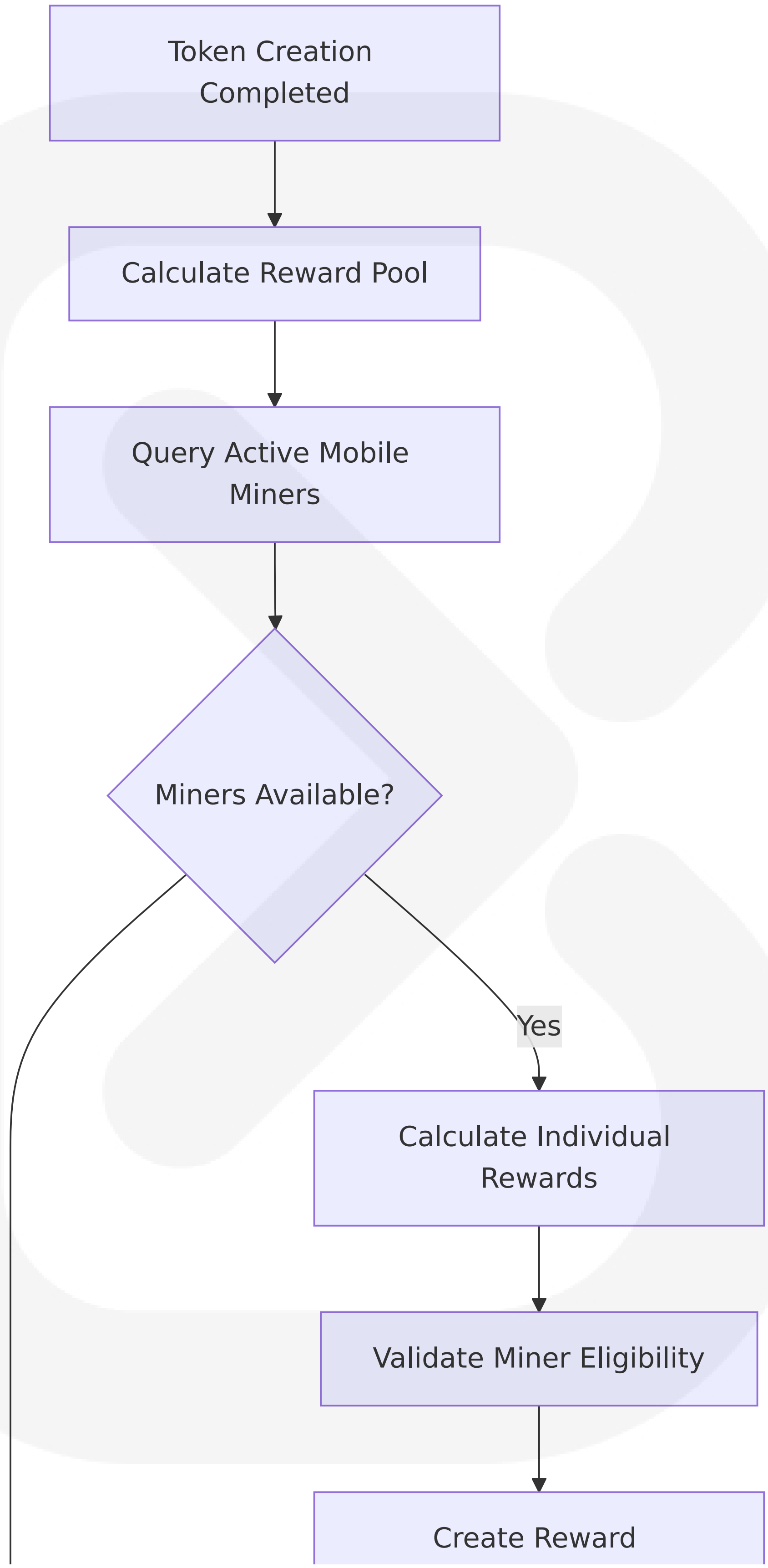

User Selects Payment Method
Payment Type
SOL
$TEOS
Validate SOL Balance
Validate $TEOS Balance
Sufficient SOL?
Sufficient $TEOS?
No
No
Yes
Yes
Display Insufficient Balance
Calculate SOL Transfer Amount
Calculate $TEOS Transfer Amount
Create SOL Transfer Transaction
Create $TEOS Transfer Transaction
Add Network Fee Calculation
Present Transaction for Signing
User Signs Transaction?
No
Yes

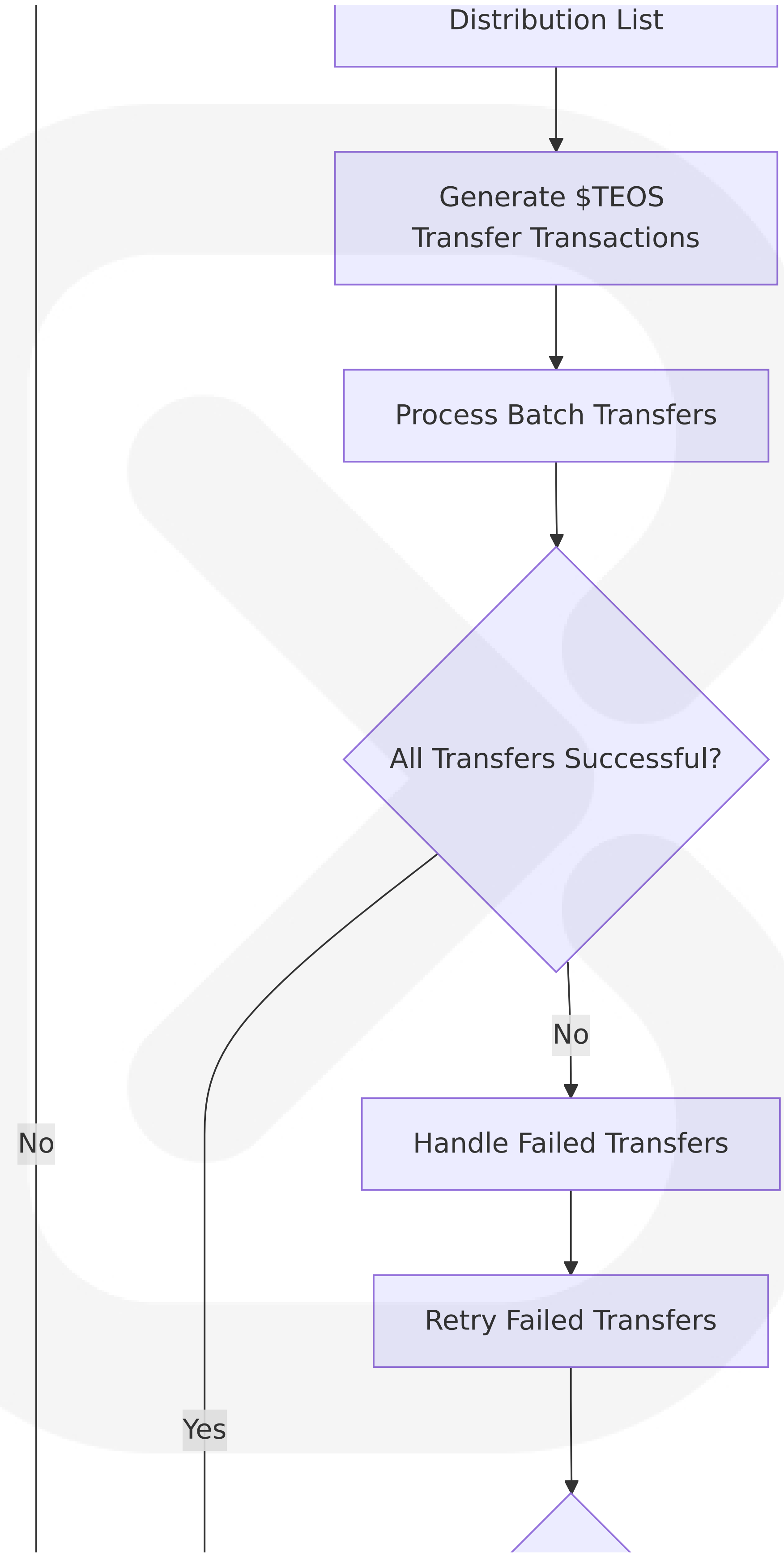


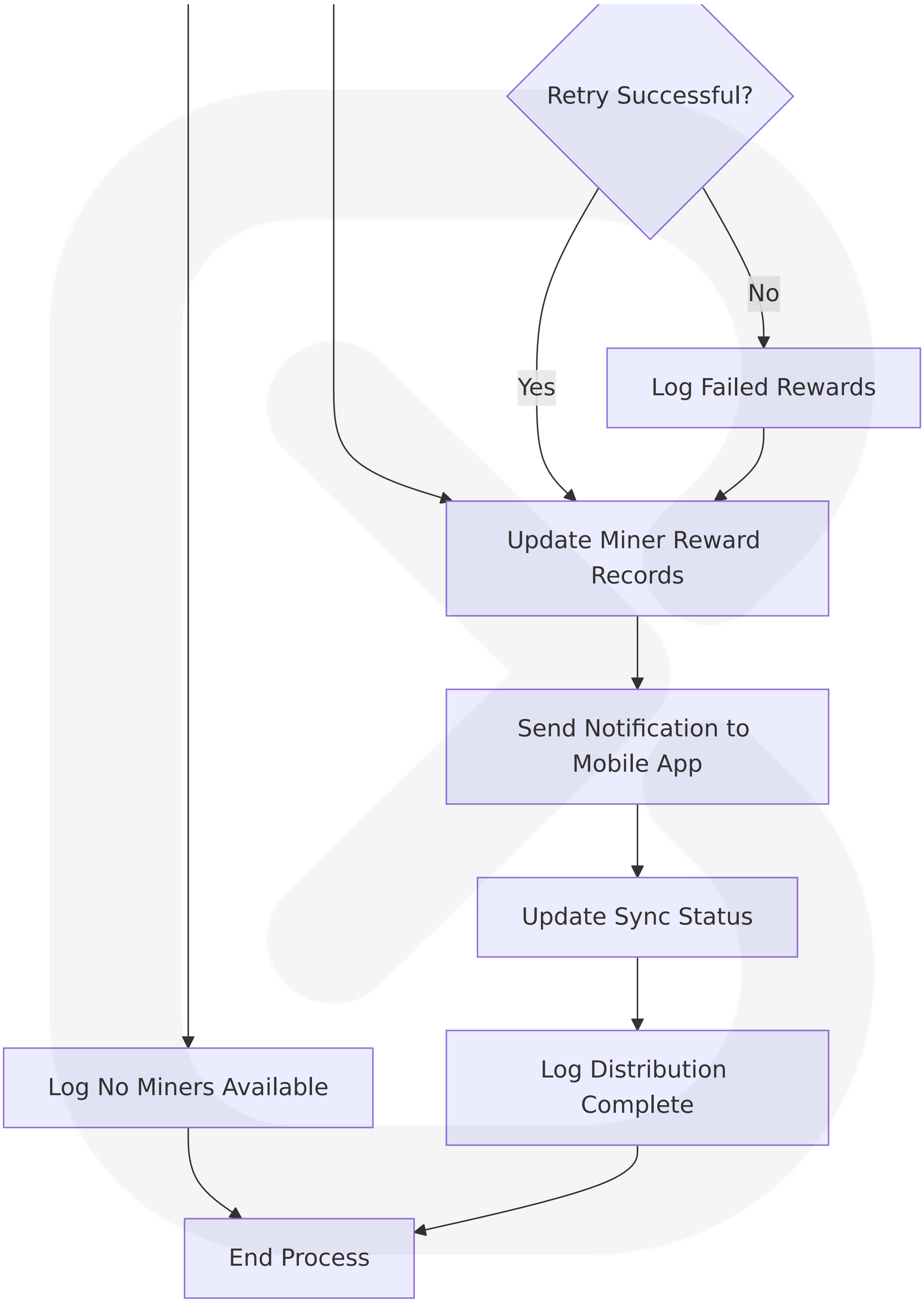
4.1.2 Integration Workflows

Mobile Mining Reward Distribution Workflow

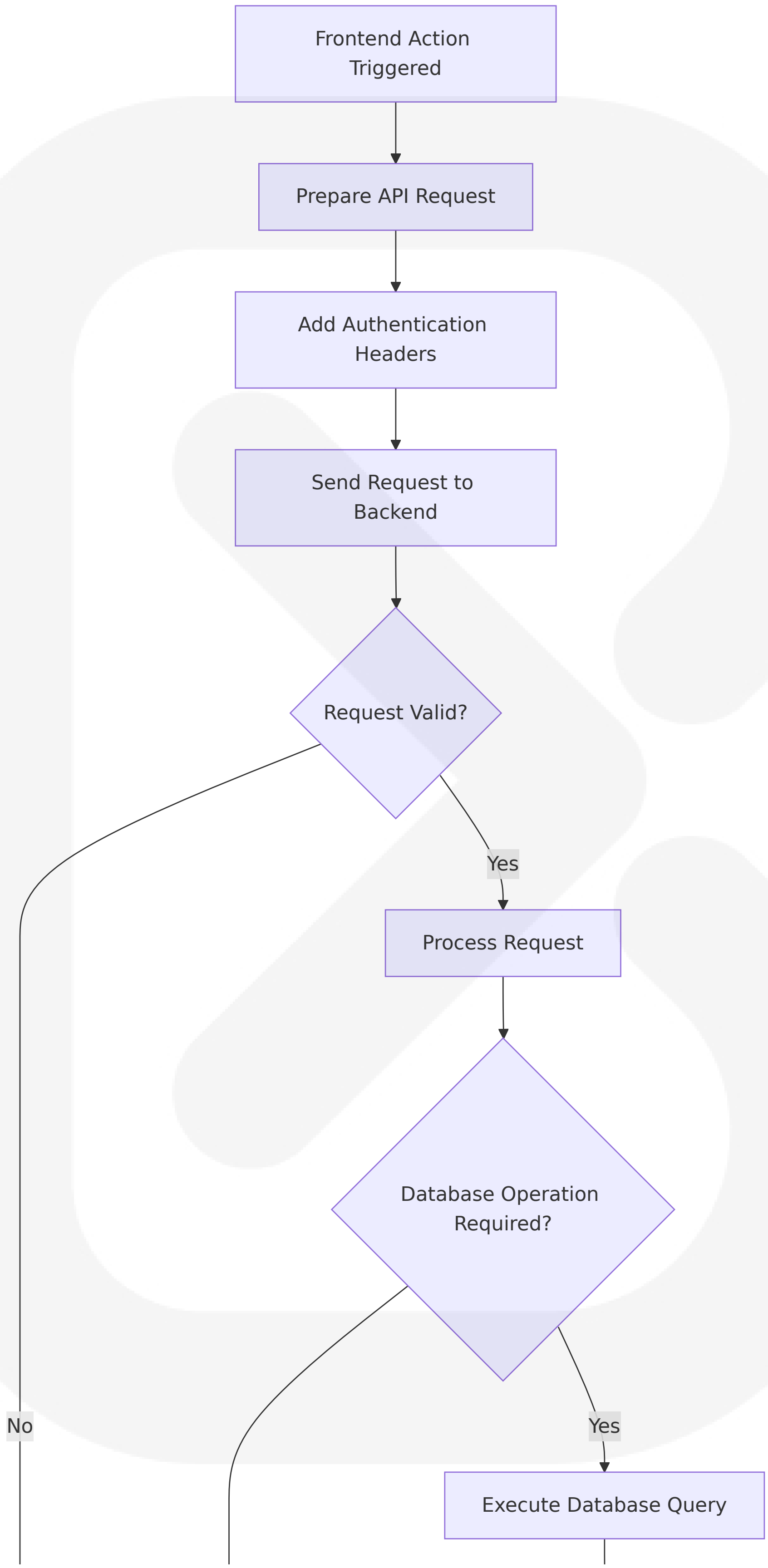
Some projects use this idea of "mobile mining" as a way to distribute their coins to a lot of users. These apps don't actually help with blockchain consensus, they just distribute coins to users over time.

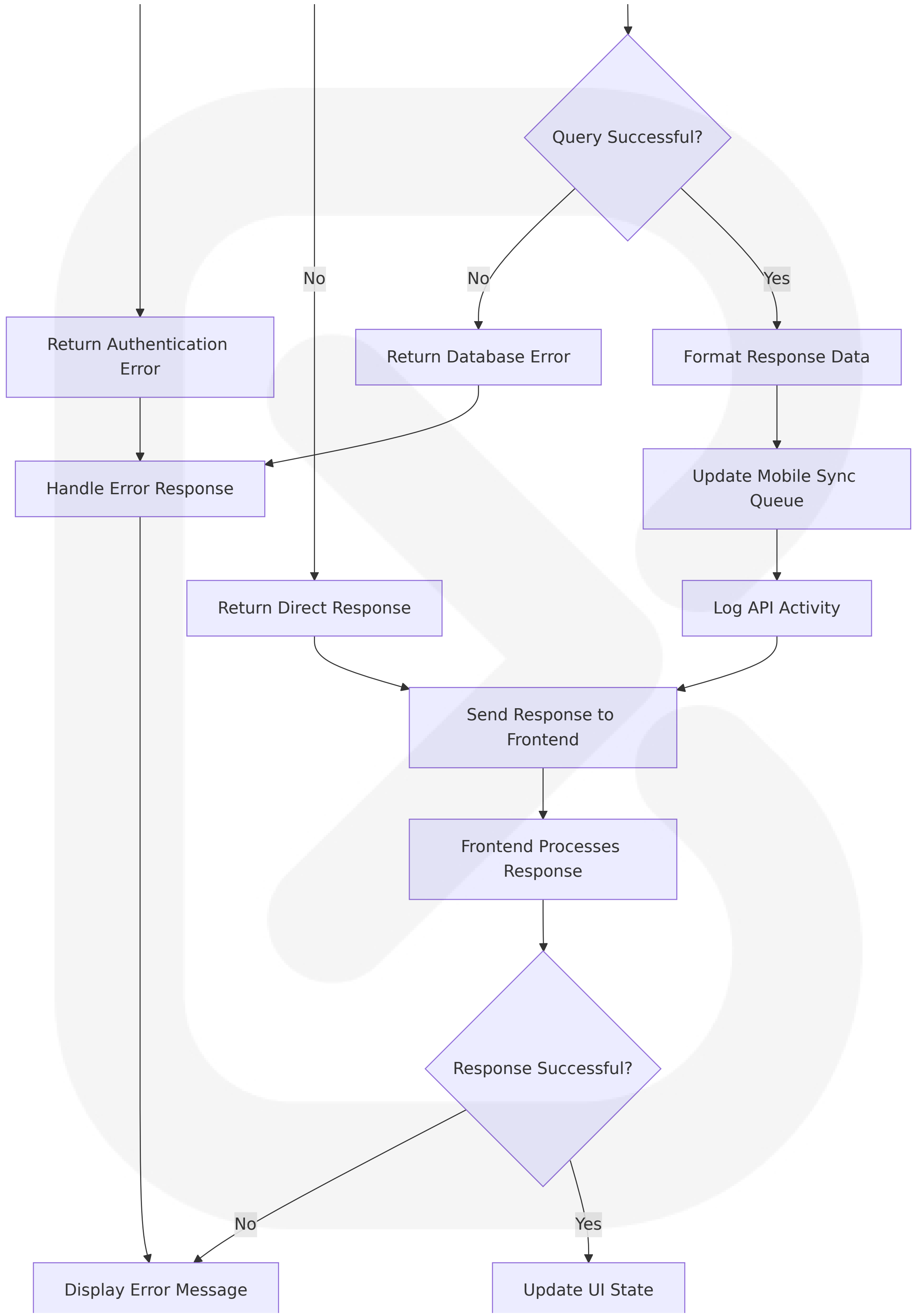

Token Creation Completed
Calculate Reward Pool
Query Active Mobile Miners
Miners Available?
Yes
Calculate Individual Rewards
Validate Miner Eligibility
Create Reward

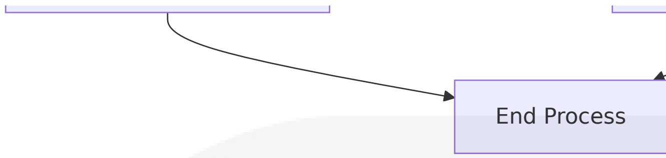

Distribution List
Generate $TEOS Transfer Transactions
Process Batch Transfers
All Transfers Successful?
No
No
Handle Failed Transfers
Retry Failed Transfers
Yes


Retry Successful?
Yes
No
Log Failed Rewards
Update Miner Reward Records
Send Notification to Mobile App
Update Sync Status
Log No Miners Available
Log Distribution Complete
End Process

Backend API Synchronization Workflow

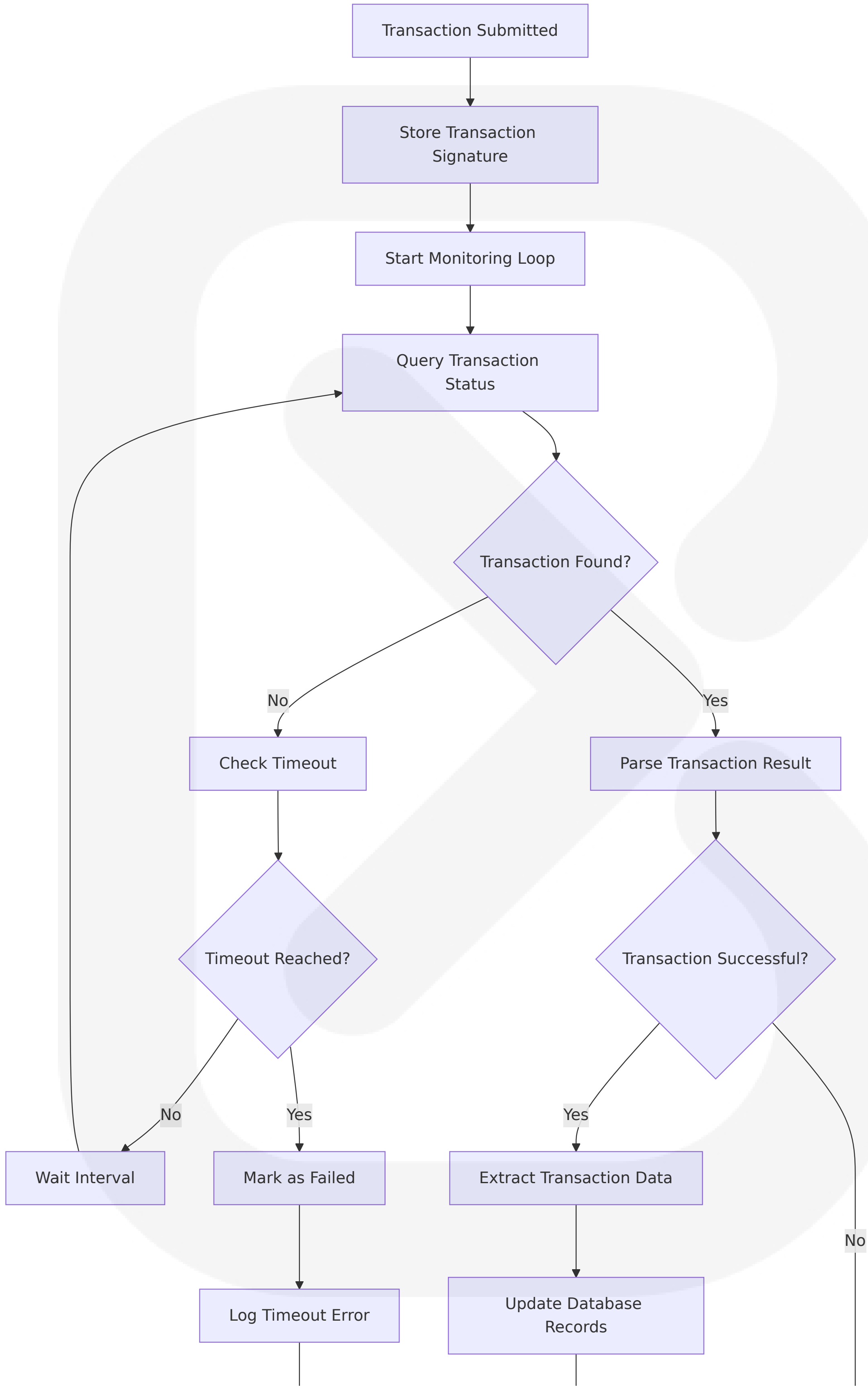

Frontend Action Triggered
Prepare API Request
Add Authentication Headers
Send Request to Backend
Request Valid?
Yes
No
Process Request
Database Operation Required?
Yes
Execute Database Query

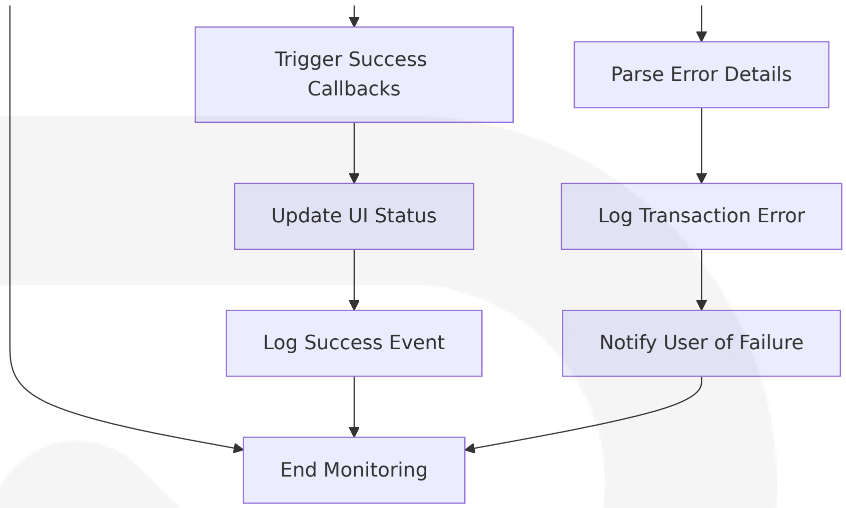

Query Successful?
No
No
Yes
Return Authentication Error
Return Database Error
Format Response Data
Handle Error Response
Update Mobile Sync Queue
Return Direct Response
Log API Activity
Send Response to Frontend
Frontend Processes Response
Response Successful?
No
Yes
Display Error Message
Update UI State



Blockchain Transaction Monitoring Workflow

All token operations are recorded on the Solana blockchain, allowing users to verify transactions via blockchain explorers like Solscan.


Transaction Submitted
Store Transaction Signature
Start Monitoring Loop
Query Transaction Status
Transaction Found?
No
Yes
Check Timeout
Parse Transaction Result
Timeout Reached?
Transaction Successful?
No
Yes
Yes
No
Wait Interval
Mark as Failed
Extract Transaction Data
Log Timeout Error
Update Database Records

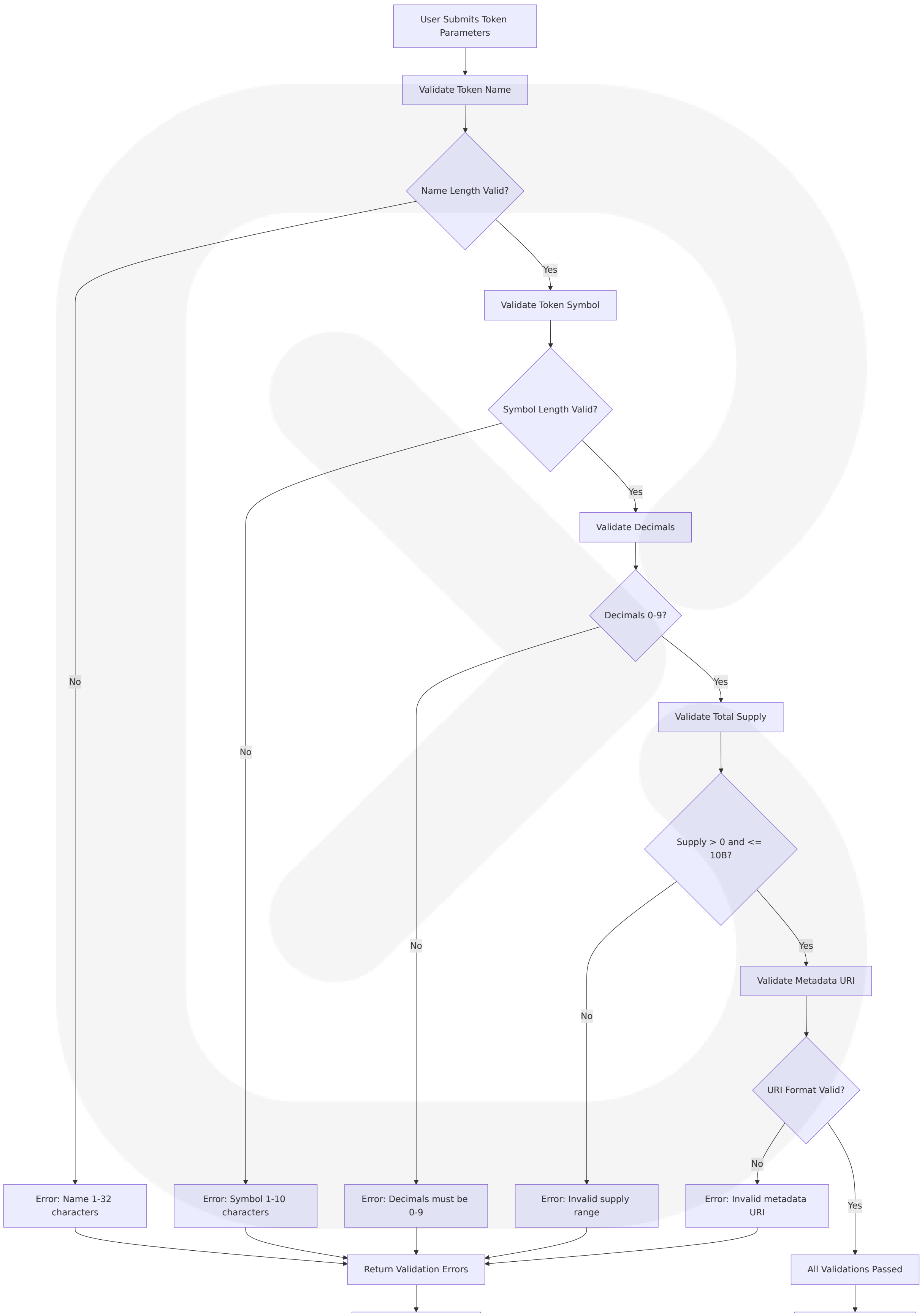


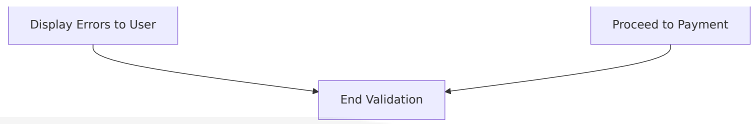
4.2 FLOWCHART REQUIREMENTS

4.2.1 Validation Rules and Business Logic

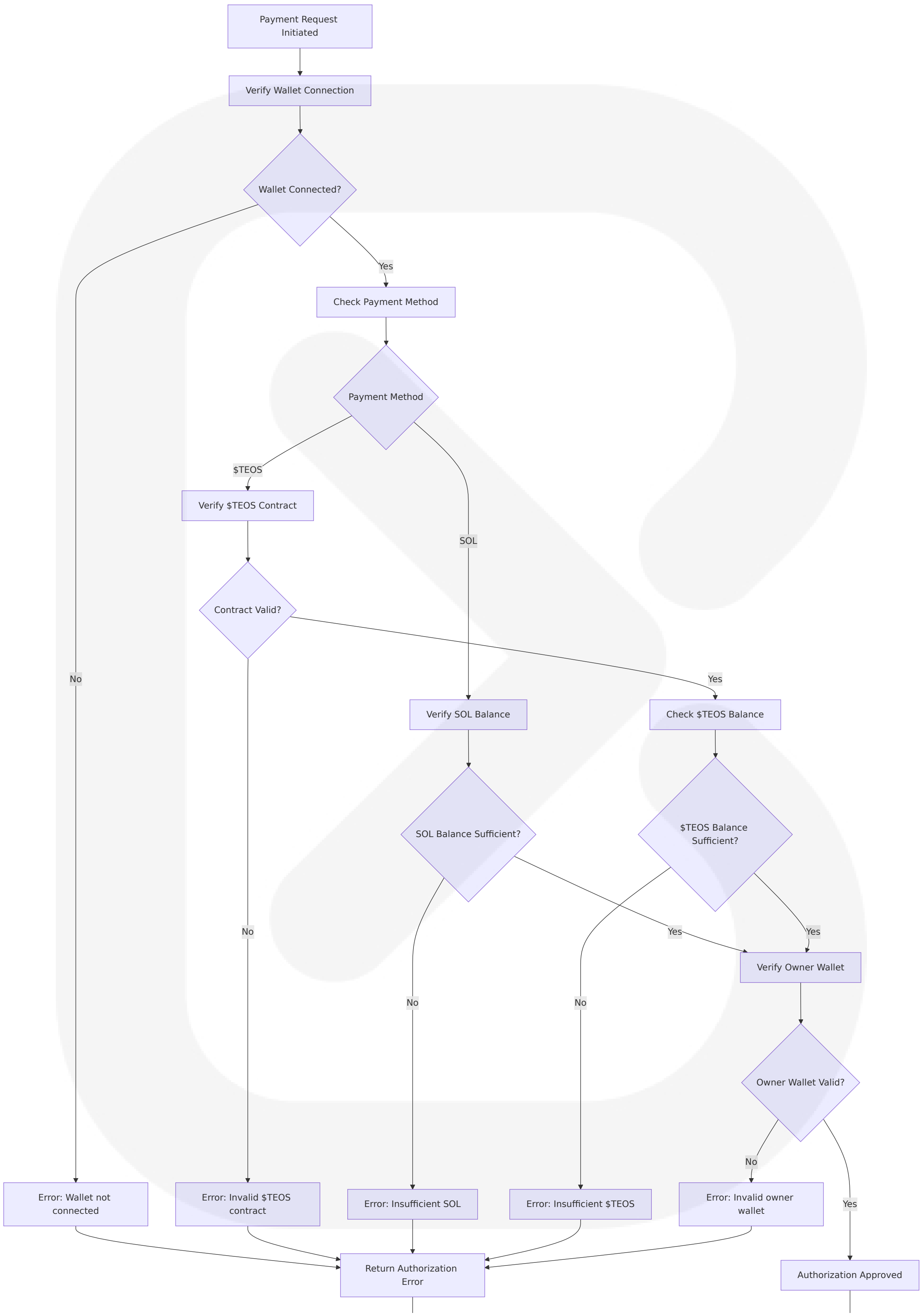
Token Parameter Validation Workflow

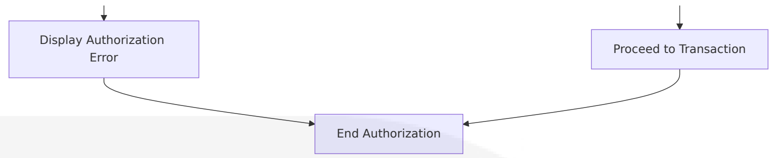
The supply is one of the most important parts, as we have to organize the financial landscape that the SPL token will have. This decision is closely linked to the tokenomics of your project. Other than that, a supply of 1 B 0 10 B is the most normal, but it all depends on the specifications you want for your project.


User Submits Token Parameters
Validate Token Name
Name Length Valid?
Yes
No
Validate Token Symbol
Symbol Length Valid?
Yes
No
Validate Decimals
Decimals 0-9?
Yes
No
Validate Total Supply
Supply > 0 and <= 10B?
Yes
No
Validate Metadata URI
URI Format Valid?
No
Yes
Error: Name 1-32 characters
Error: Symbol 1-10 characters
Error: Decimals must be 0-9
Error: Invalid supply range
Error: Invalid metadata URI
Return Validation Errors
All Validations Passed



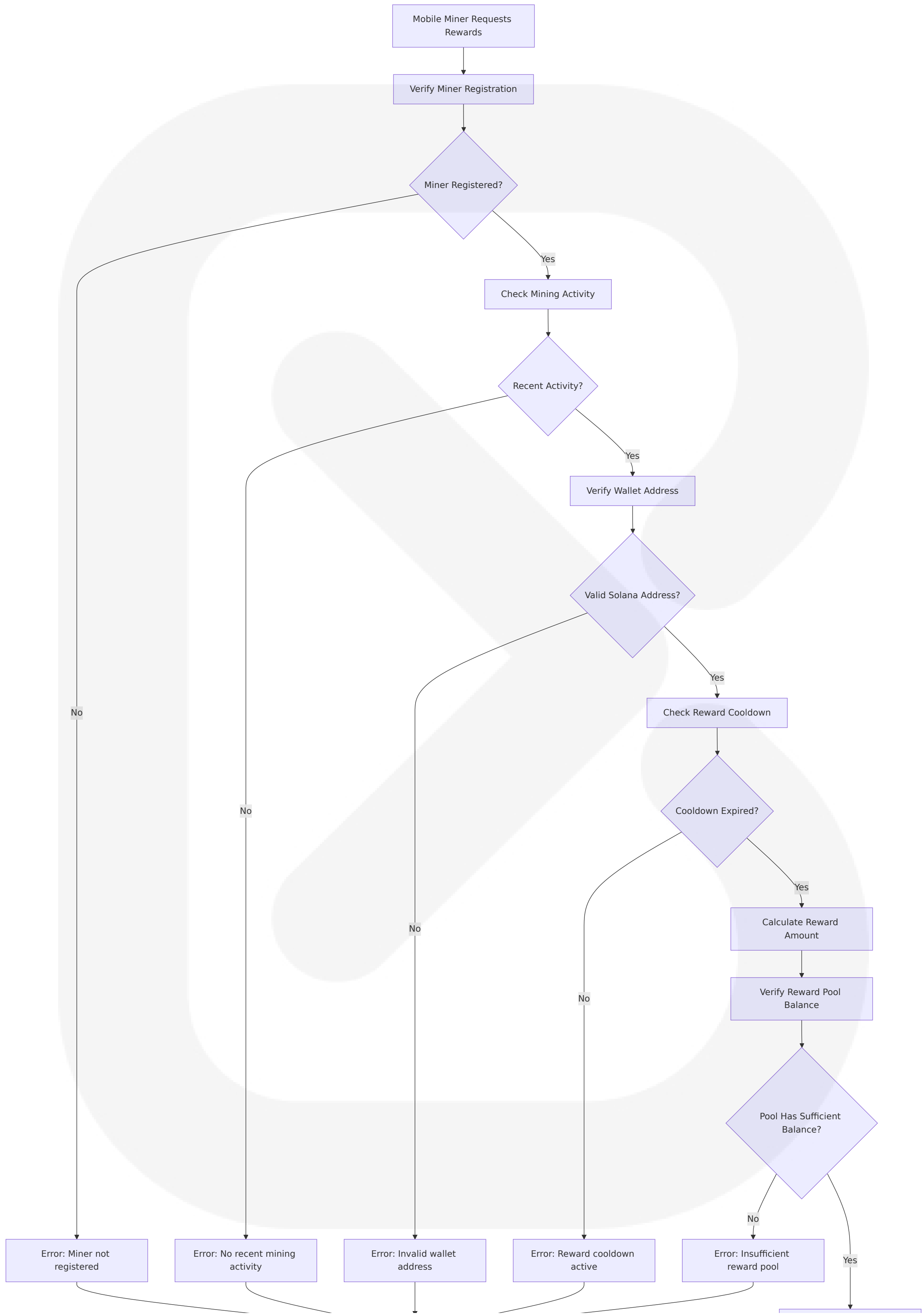
Payment Authorization Workflow

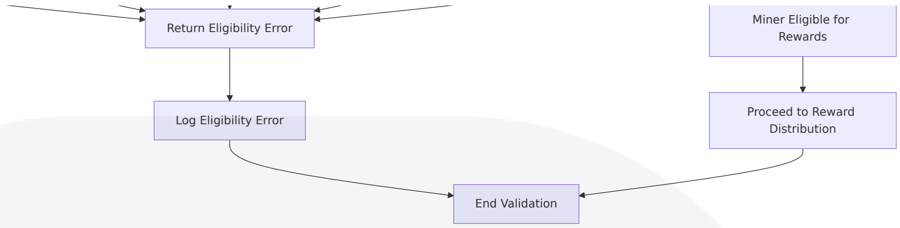

Payment Request Initiated
Verify Wallet Connection
Wallet Connected?
Yes
No
Check Payment Method
Payment Method
$TEOS
SOL
Verify $TEOS Contract
Contract Valid?
Yes
No
Verify SOL Balance
Check $TEOS Balance
SOL Balance Sufficient?
$TEOS Balance Sufficient?
Yes
Yes
No
No
Verify Owner Wallet
Owner Wallet Valid?
No
Yes
Error: Wallet not connected
Error: Invalid $TEOS contract
Error: Insufficient SOL
Error: Insufficient $TEOS
Error: Invalid owner wallet
Return Authorization Error
Authorization Approved



Mobile Mining Eligibility Validation

Mobile mining has emerged as a revolutionary concept, empowering individuals to participate in the cryptocurrency mining process using their smartphones. If you want to earn money with your smartphone by simply mining free on your phone and in the end selling it in dollars, here are the top 10 mobile mining networks that have a promising future.

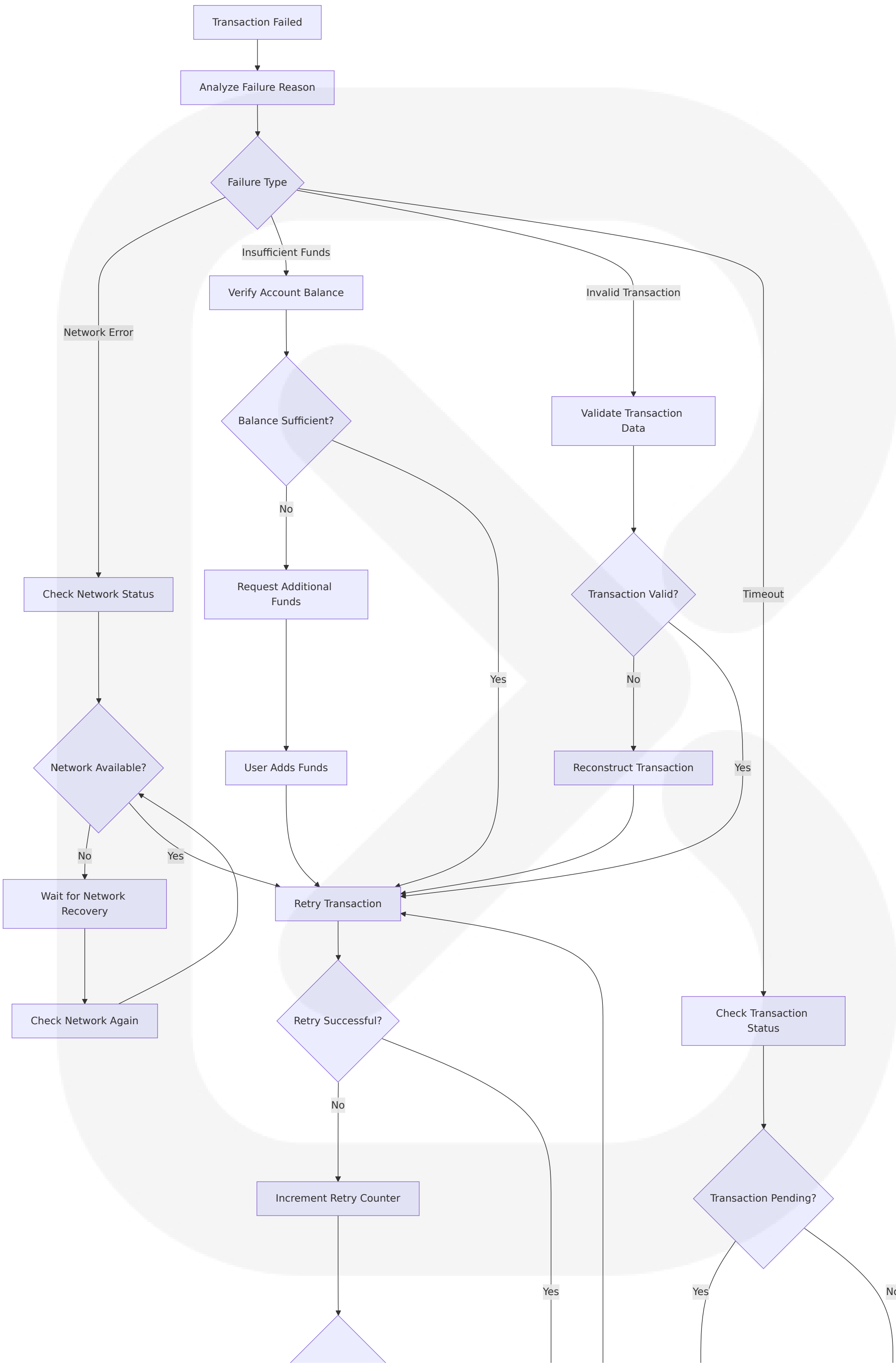

Mobile Miner Requests Rewards
Verify Miner Registration
Miner Registered?
Yes
No
Check Mining Activity
Recent Activity?
Yes
No
Verify Wallet Address
Valid Solana Address?
Yes
No
Check Reward Cooldown
Cooldown Expired?
Yes
No
Calculate Reward Amount
Verify Reward Pool Balance
Pool Has Sufficient Balance?
No
Yes
Error: Miner not registered
Error: No recent mining activity
Error: Invalid wallet address
Error: Reward cooldown active
Error: Insufficient reward pool



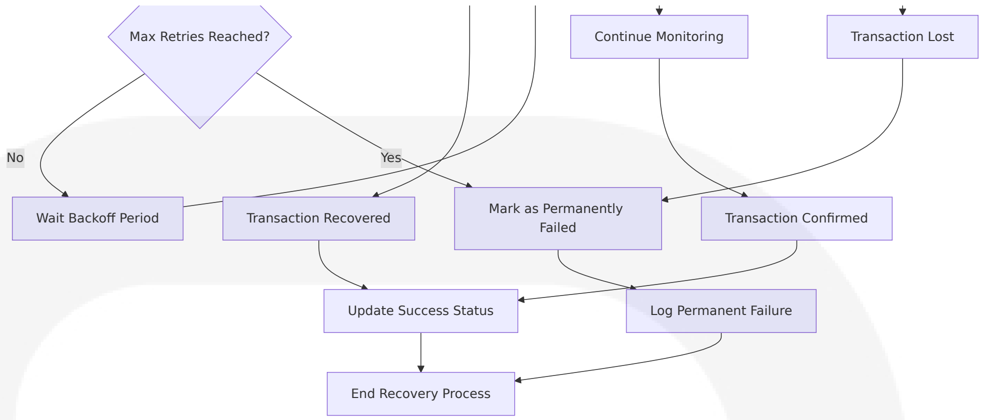
4.2.2 Error Handling and Recovery Workflows

Transaction Failure Recovery Workflow

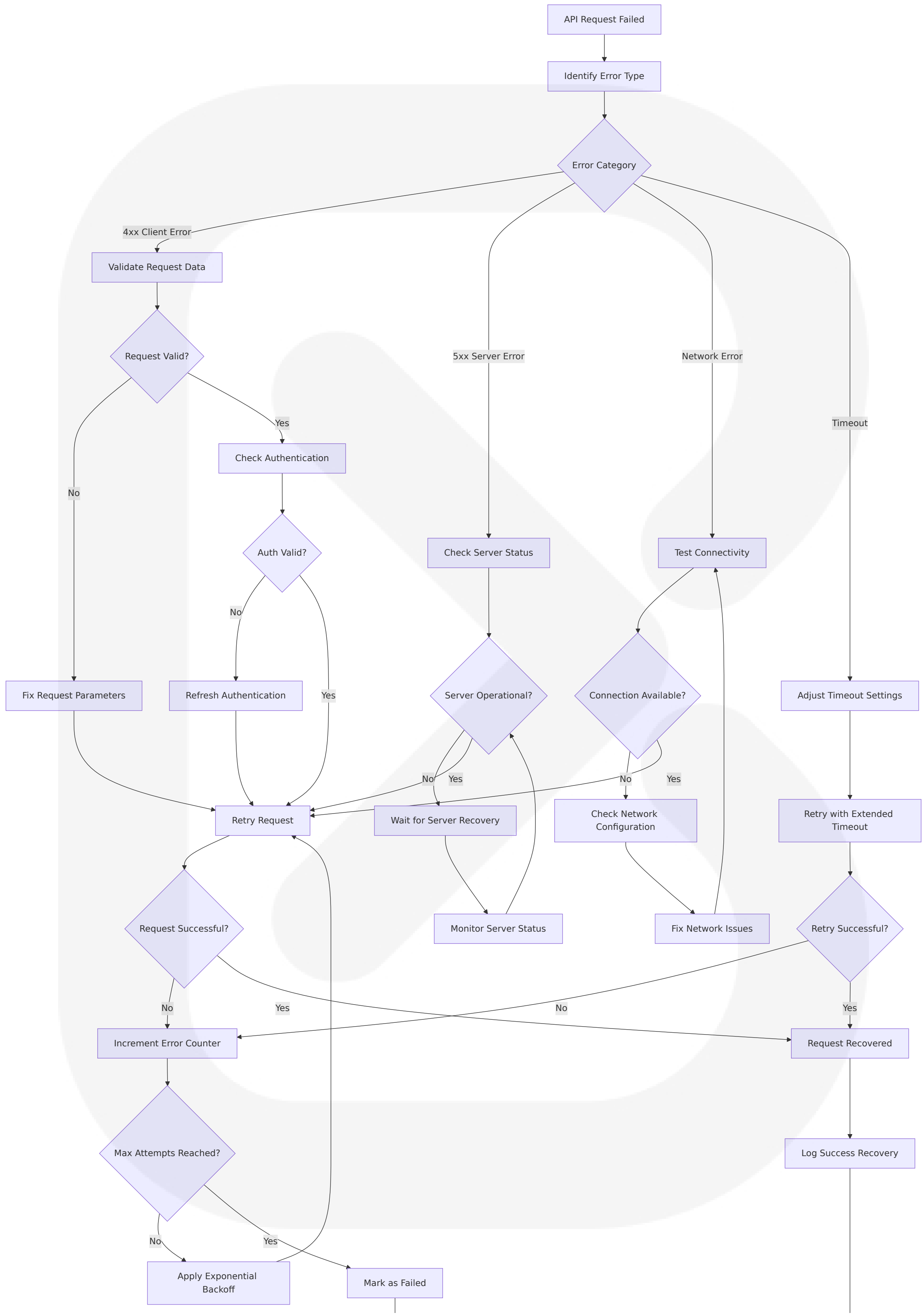
Fast Transaction Speeds: Solana's architecture allows for fast transaction speeds, processing thousands of transactions per second. Low Fees: The low fees associated with transactions on Solana make it economically viable for token creation.

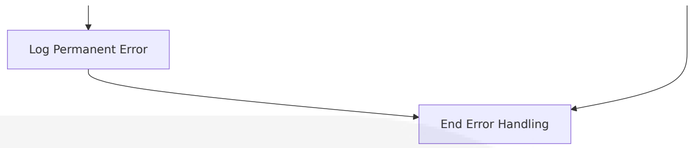


Transaction Failed
Analyze Failure Reason
Failure Type
Network Error
Insufficient Funds
Invalid Transaction
Timeout
Verify Account Balance
Balance Sufficient?
Validate Transaction Data
No
Yes
Check Network Status
Request Additional Funds
Transaction Valid?
Network Available?
User Adds Funds
Reconstruct Transaction
No
Yes
Wait for Network Recovery
Retry Transaction
Check Network Again
Retry Successful?
Check Transaction Status
Increment Retry Counter
Transaction Pending?



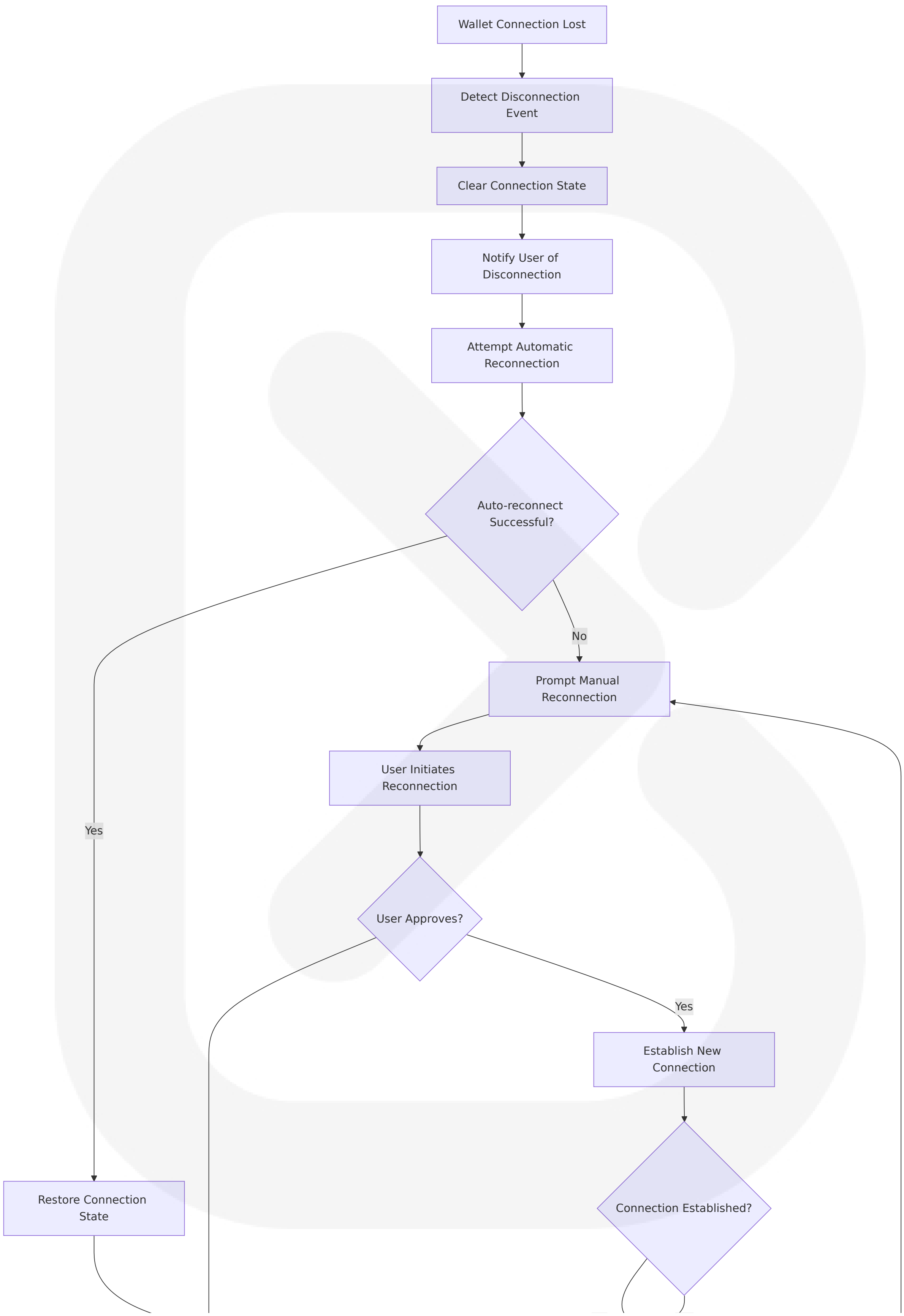
API Error Handling Workflow

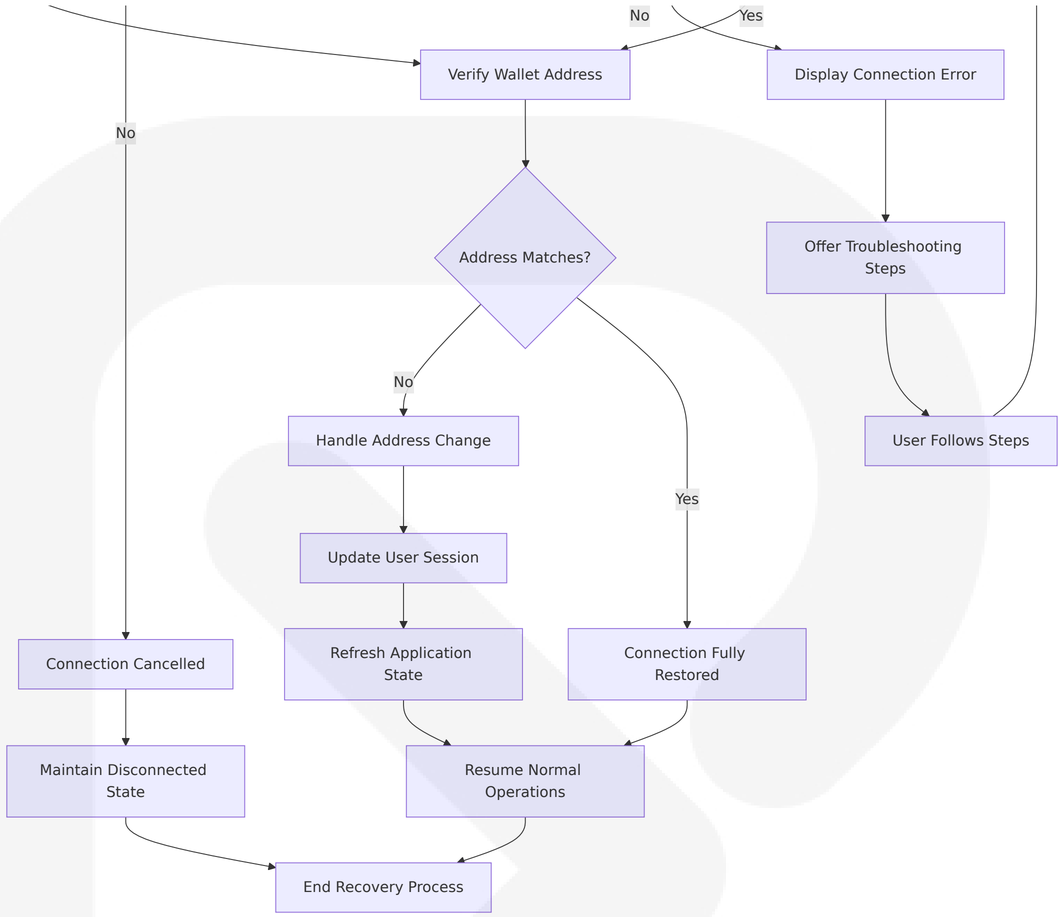

API Request Failed
Identify Error Type
Error Category
4xx Client Error
5xx Server Error
Network Error
Timeout
Validate Request Data
Request Valid?
Yes
No
Check Authentication
Auth Valid?
No
Yes
Check Server Status
Test Connectivity
Fix Request Parameters
Refresh Authentication
Server Operational?
Connection Available?
Adjust Timeout Settings
No
Yes
No
Yes
Retry Request
Wait for Server Recovery
Check Network Configuration
Retry with Extended Timeout
Request Successful?
Monitor Server Status
Fix Network Issues
Retry Successful?
No
Yes
No
Yes
Increment Error Counter
Request Recovered
Max Attempts Reached?
Log Success Recovery
No
Yes
Apply Exponential Backoff
Mark as Failed



Wallet Connection Recovery Workflow

Security features such as encryption, biometric authentication and hardware wallet integration are provided, but users must safeguard their secret recovery phrase to prevent unauthorized access. Transaction fees vary by blockchain, with Solana remaining cost-efficient, while Ethereum fees fluctuate based on network congestion; Phantom helps optimize gas costs automatically.

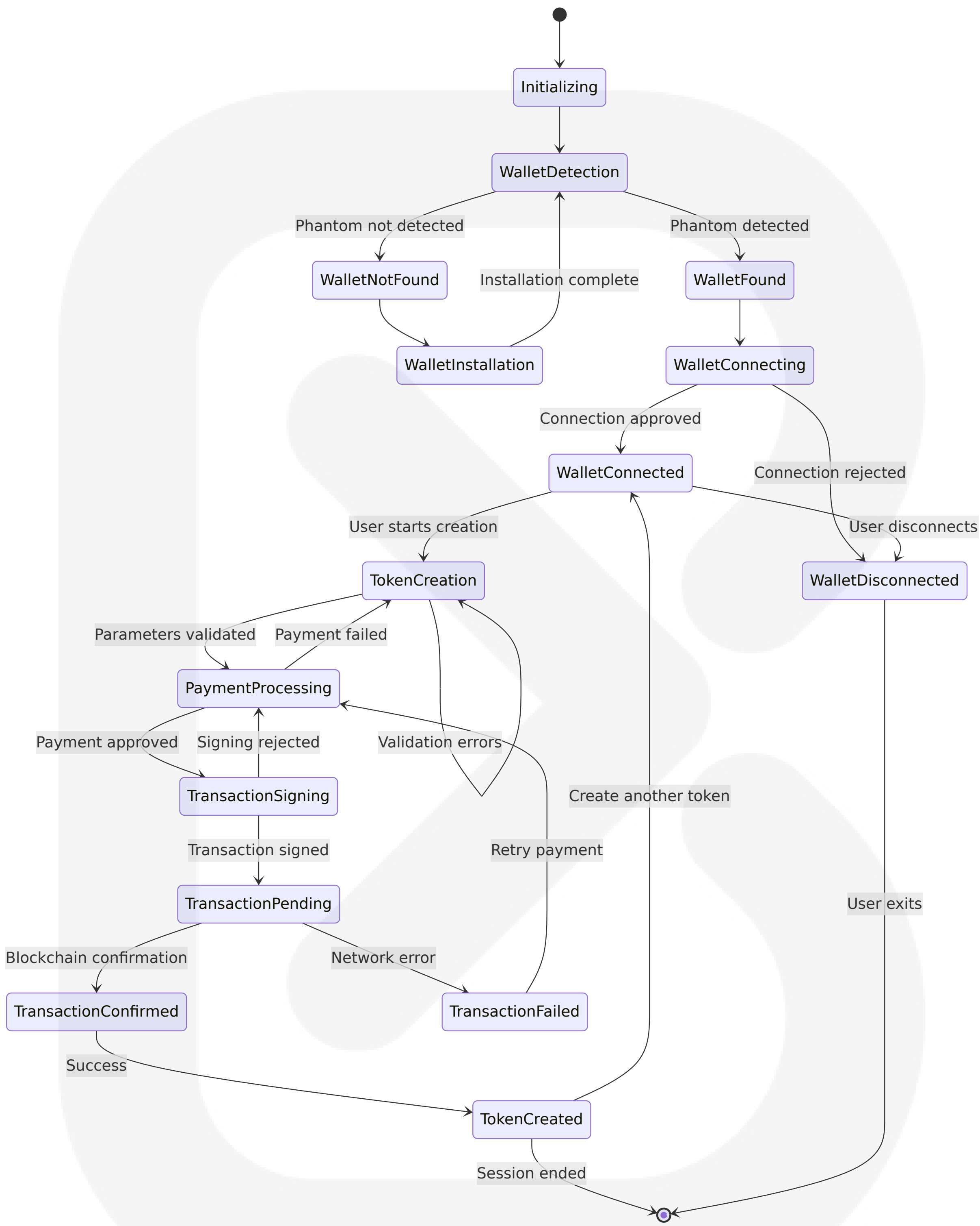

Wallet Connection Lost
Detect Disconnection Event
Clear Connection State
Notify User of Disconnection
Attempt Automatic Reconnection
Auto-reconnect Successful?
No
Prompt Manual Reconnection
User Initiates Reconnection
Yes
User Approves?
Yes
Establish New Connection
Restore Connection State
Connection Established?



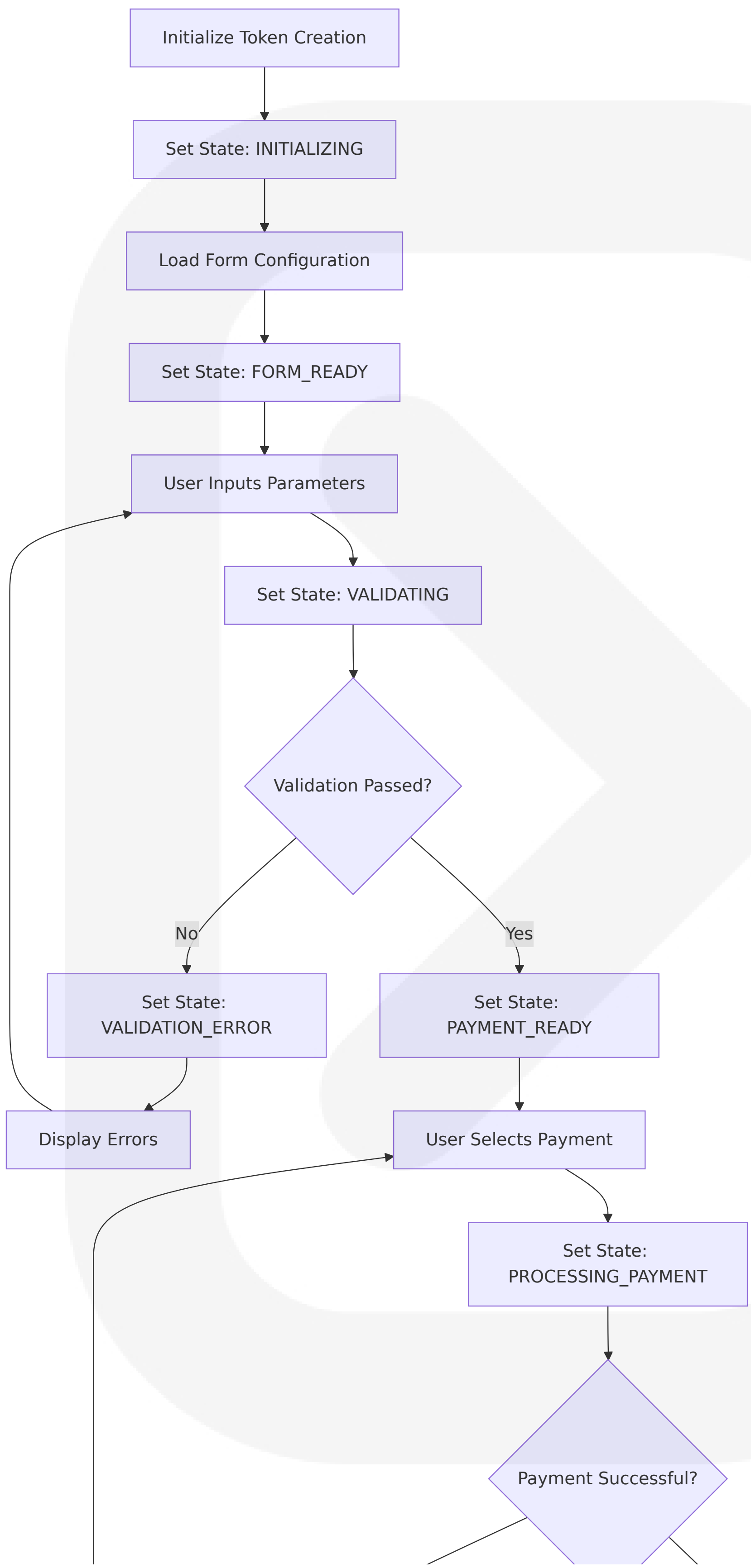
4.3 TECHNICAL IMPLEMENTATION

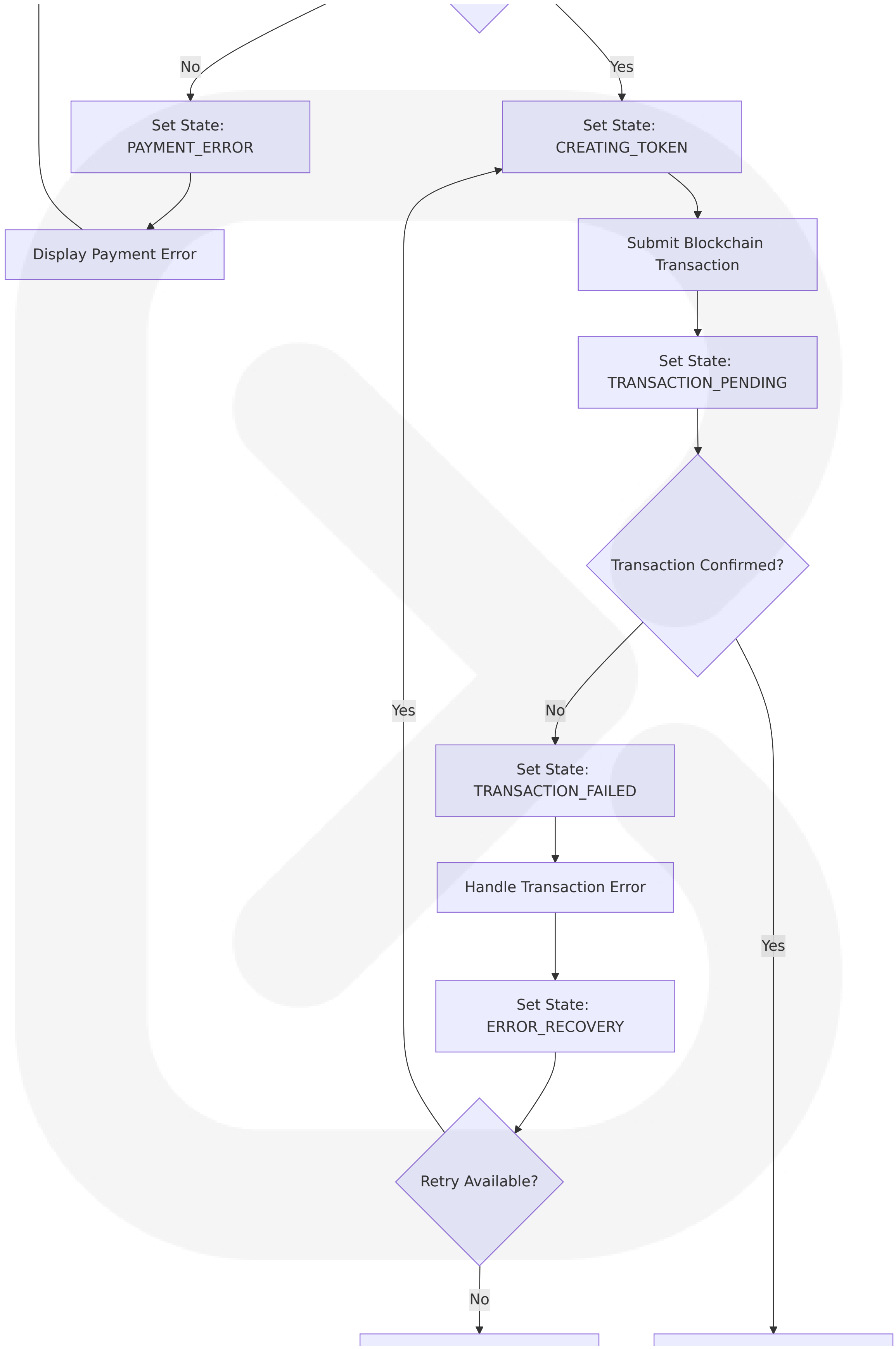
4.3.1 State Management Workflows

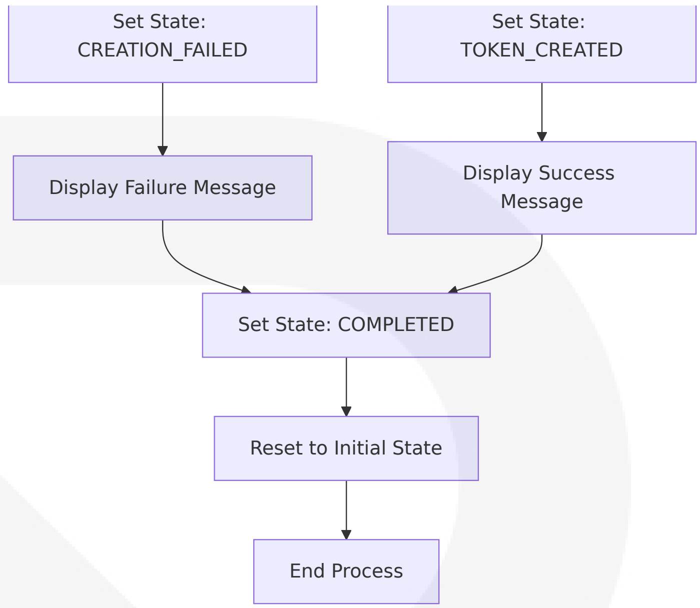
Application State Transition Diagram



Token Creation State Management

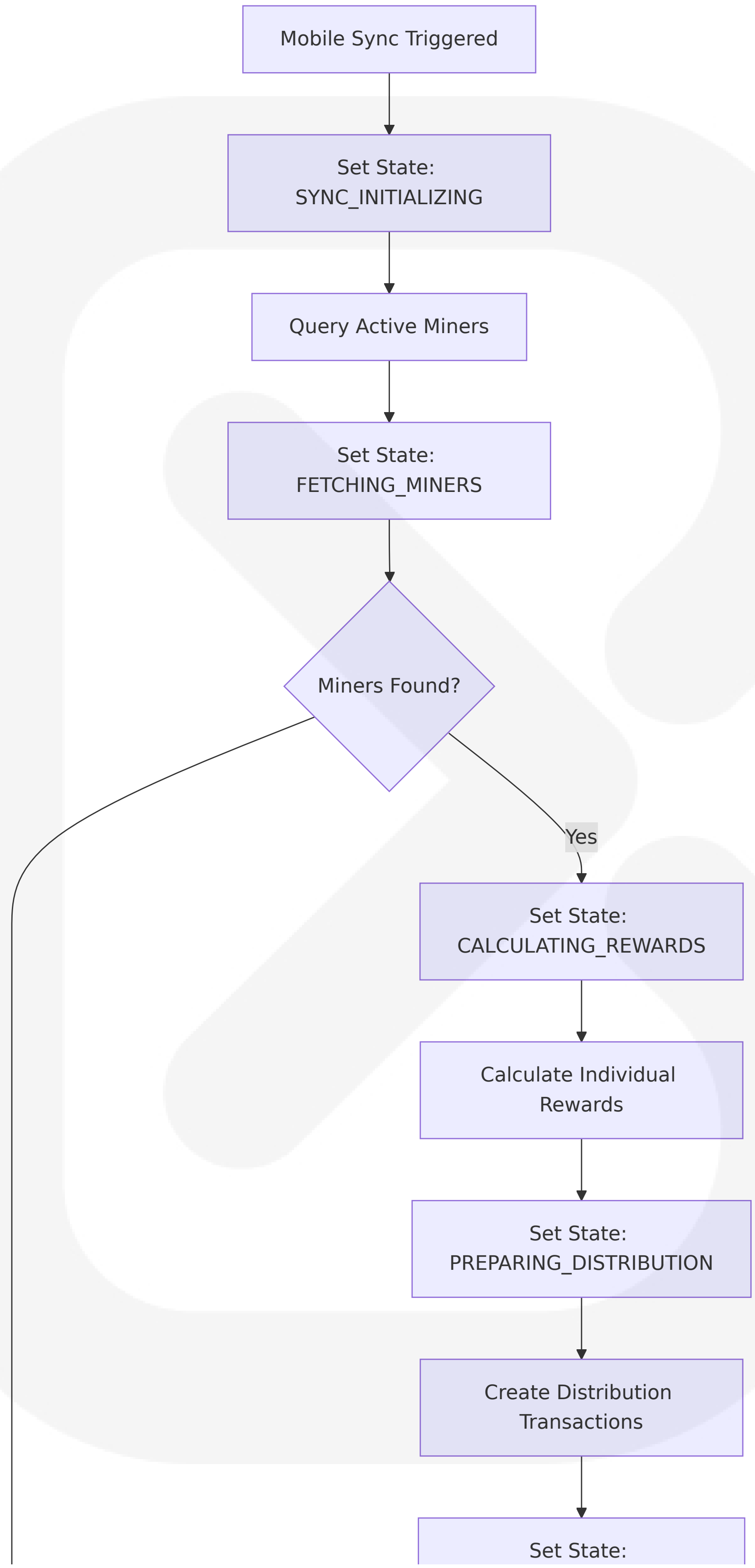

Initialize Token Creation
Set State: INITIALIZING
Load Form Configuration
Set State: FORM_READY
User Inputs Parameters
Set State: VALIDATING
Validation Passed?
No
Yes
Set State: VALIDATION_ERROR
Set State: PAYMENT_READY
Display Errors
User Selects Payment
Set State: PROCESSING_PAYMENT
Payment Successful?

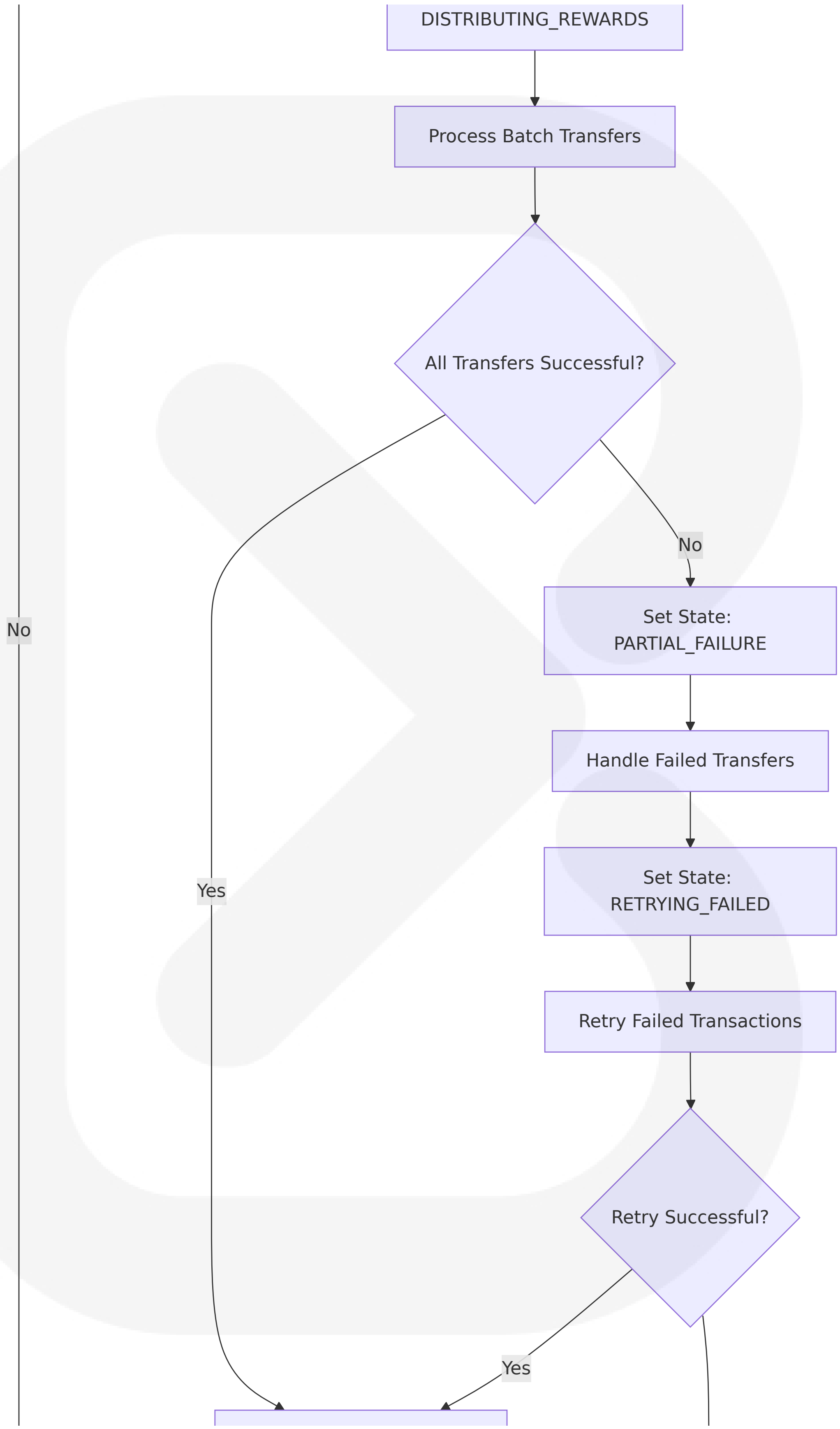

No
Yes
Set State: PAYMENT_ERROR
Set State: CREATING_TOKEN
Display Payment Error
Submit Blockchain Transaction
Set State: TRANSACTION_PENDING
Transaction Confirmed?
Yes
No
Set State: TRANSACTION_FAILED
Handle Transaction Error
Yes
Set State: ERROR_RECOVERY
Retry Available?
No

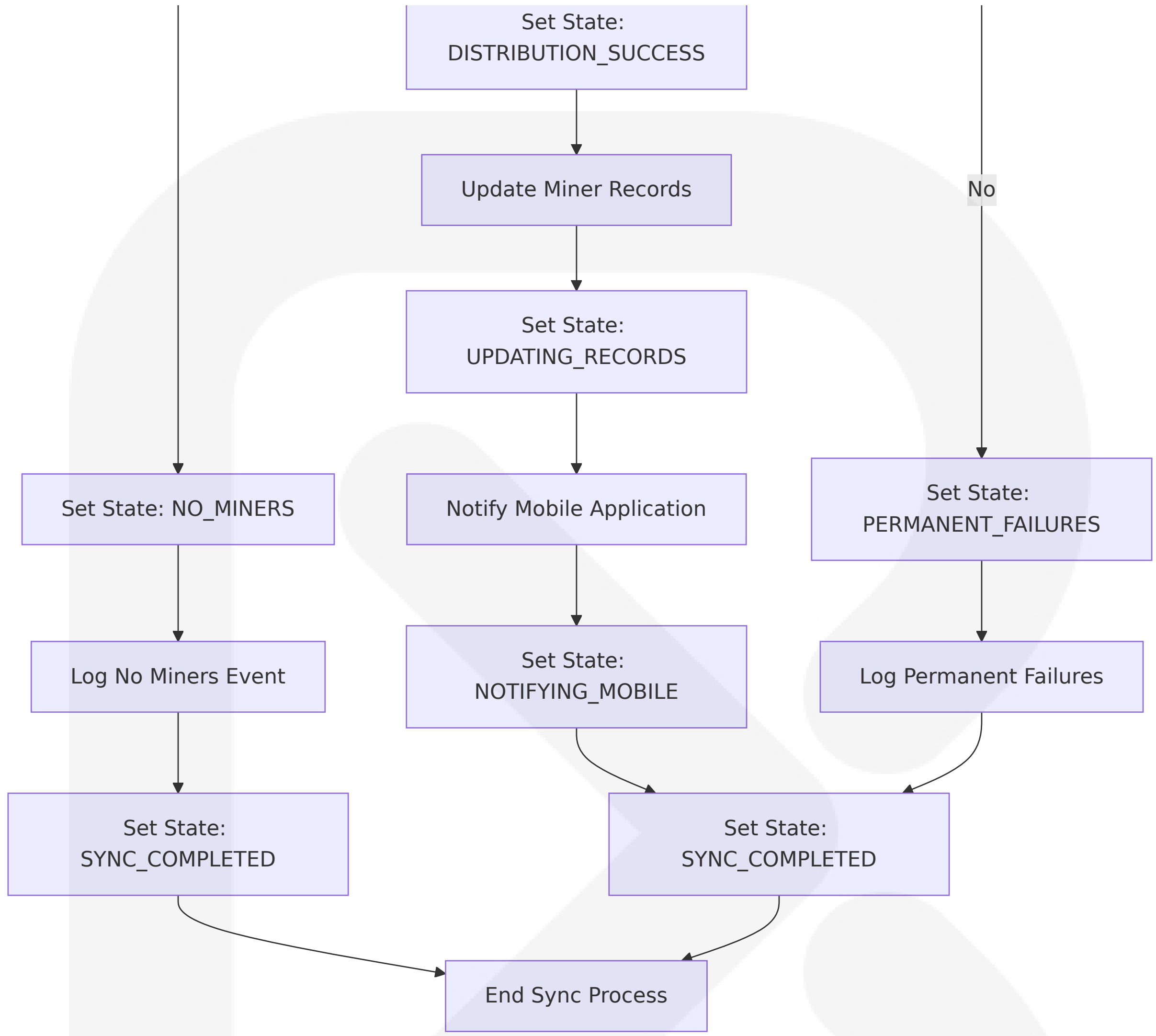


Mobile Mining Sync State Management

By harnessing the processing power of smartphones, mining pools can collectively mine cryptocurrency at a predetermined hash rate. As users continue their mining activities, they accumulate cryptocurrency rewards, which can later be converted into traditional fiat currency, enabling them to generate profits.

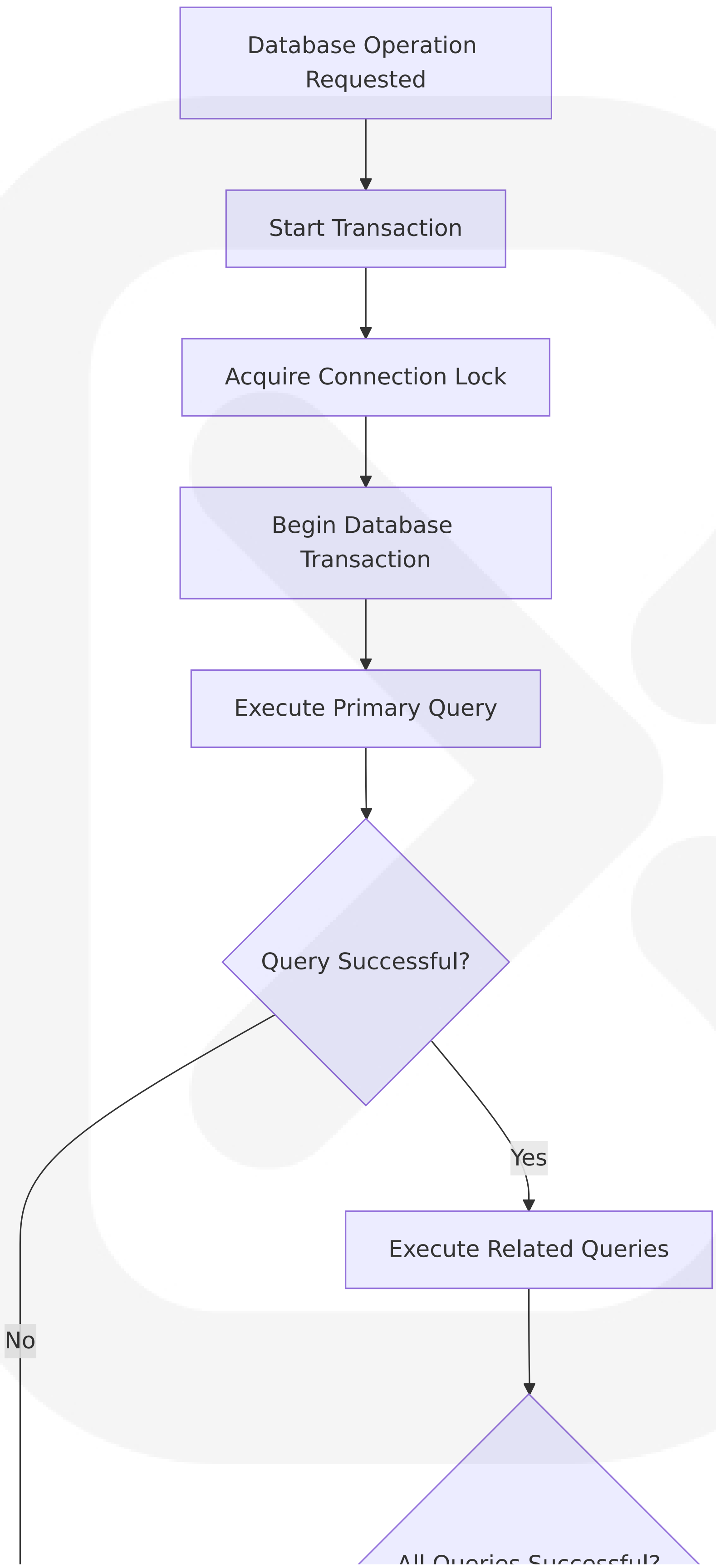

Mobile Sync Triggered
Set State: SYNC_INITIALIZING
Query Active Miners
Set State: FETCHING_MINERS
Miners Found?
Yes
Set State: CALCULATING_REWARDS
Calculate Individual Rewards
Set State: PREPARING_DISTRIBUTION
Create Distribution Transactions
Set State:

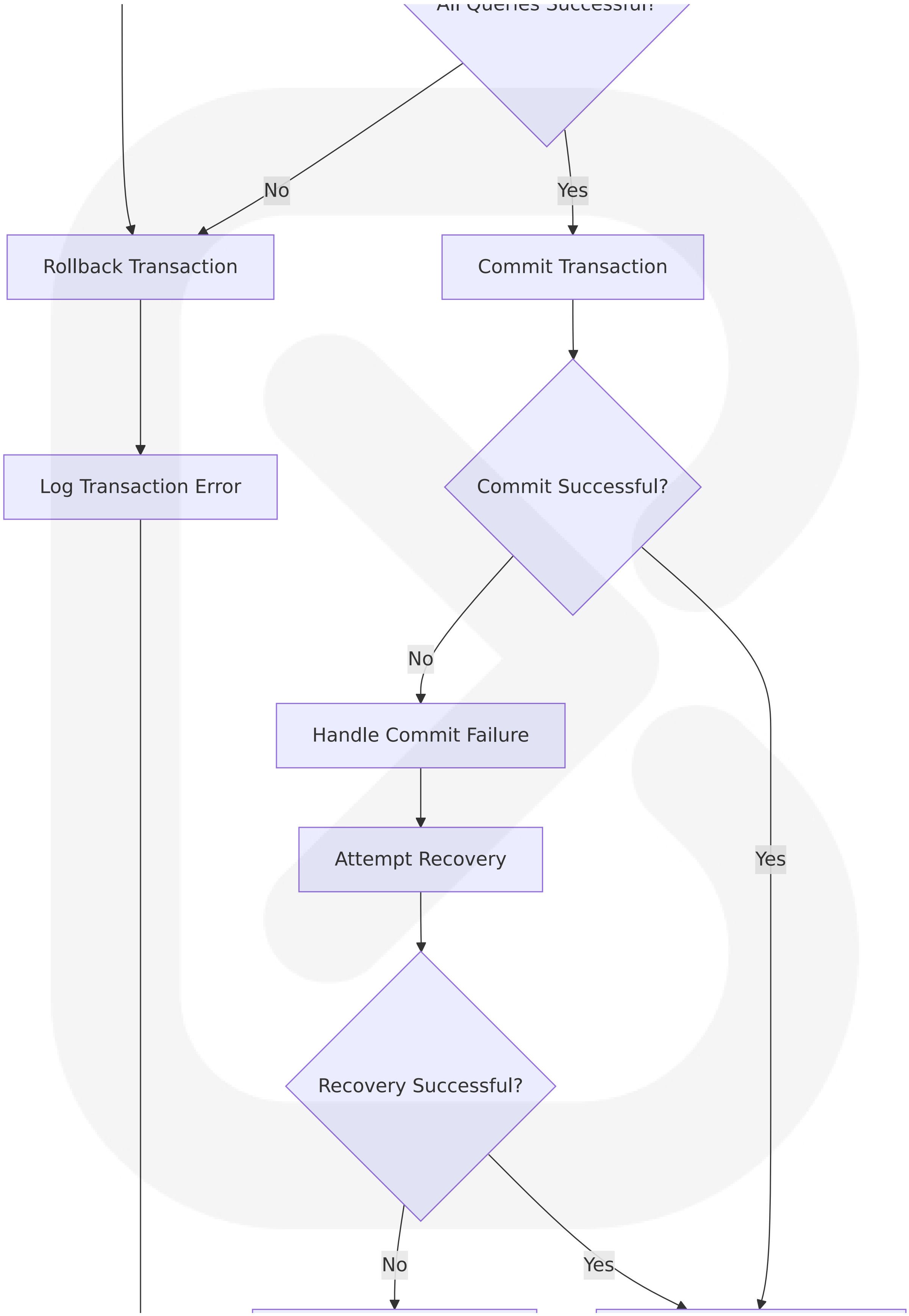

DISTRIBUTING_REWARDS
Process Batch Transfers
All Transfers Successful?
No
Yes
No
Set State:
PARTIAL_FAILURE
Handle Failed Transfers
Set State:
RETRYING_FAILED
Retry Failed Transactions
Retry Successful?
Yes

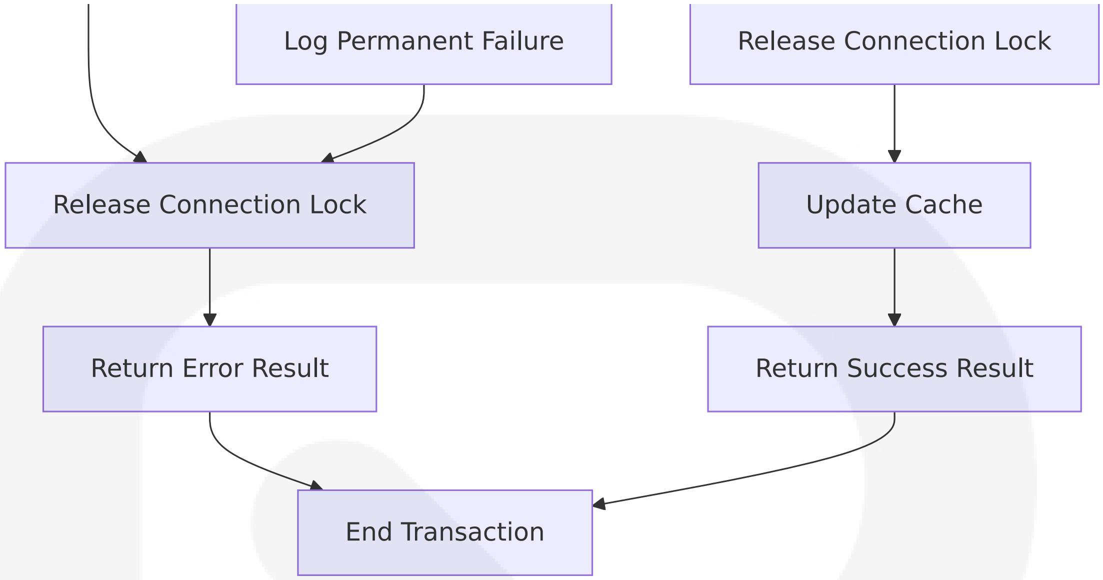


4.3.2 Data Persistence and Caching Workflows

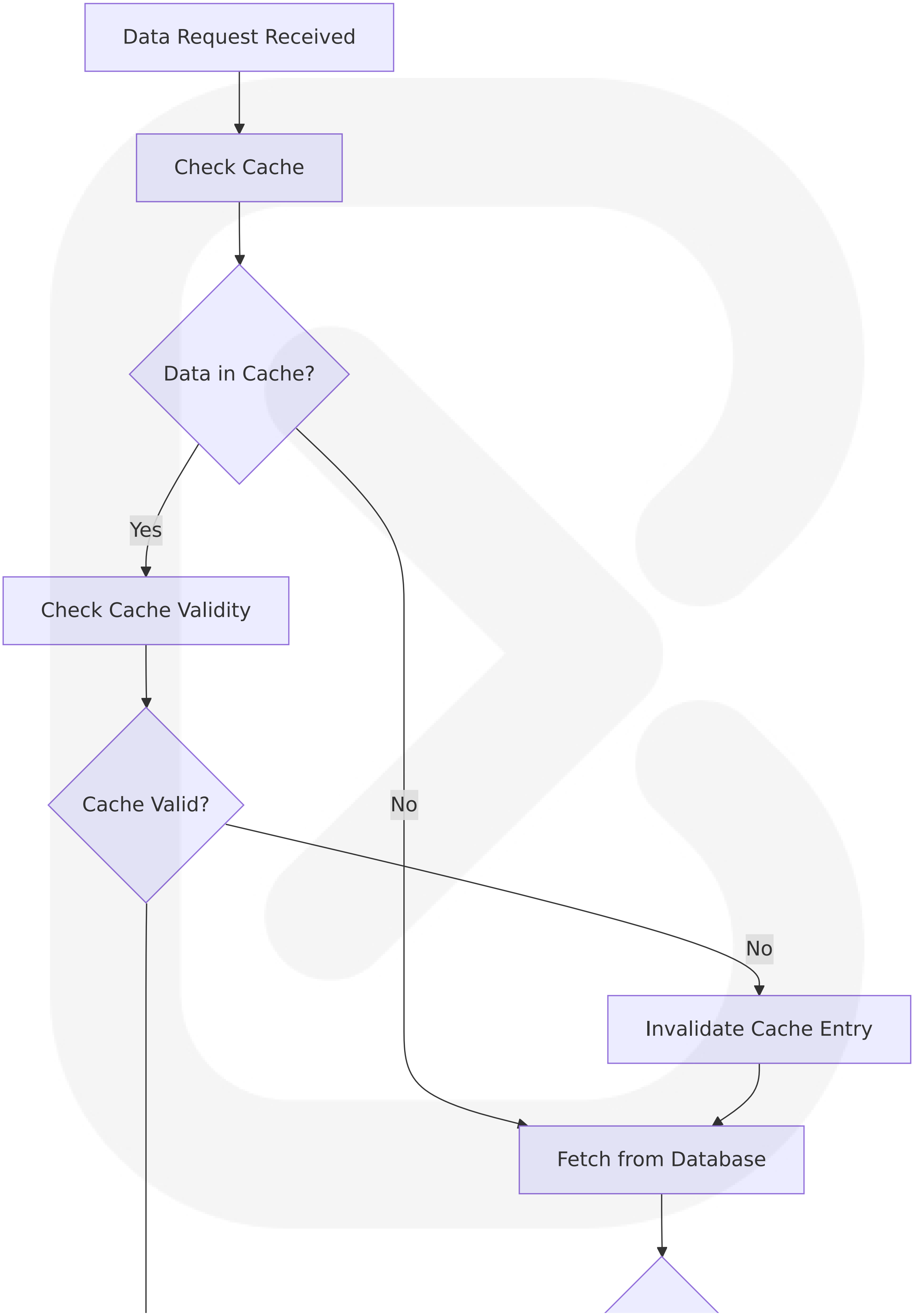
Database Transaction Workflow

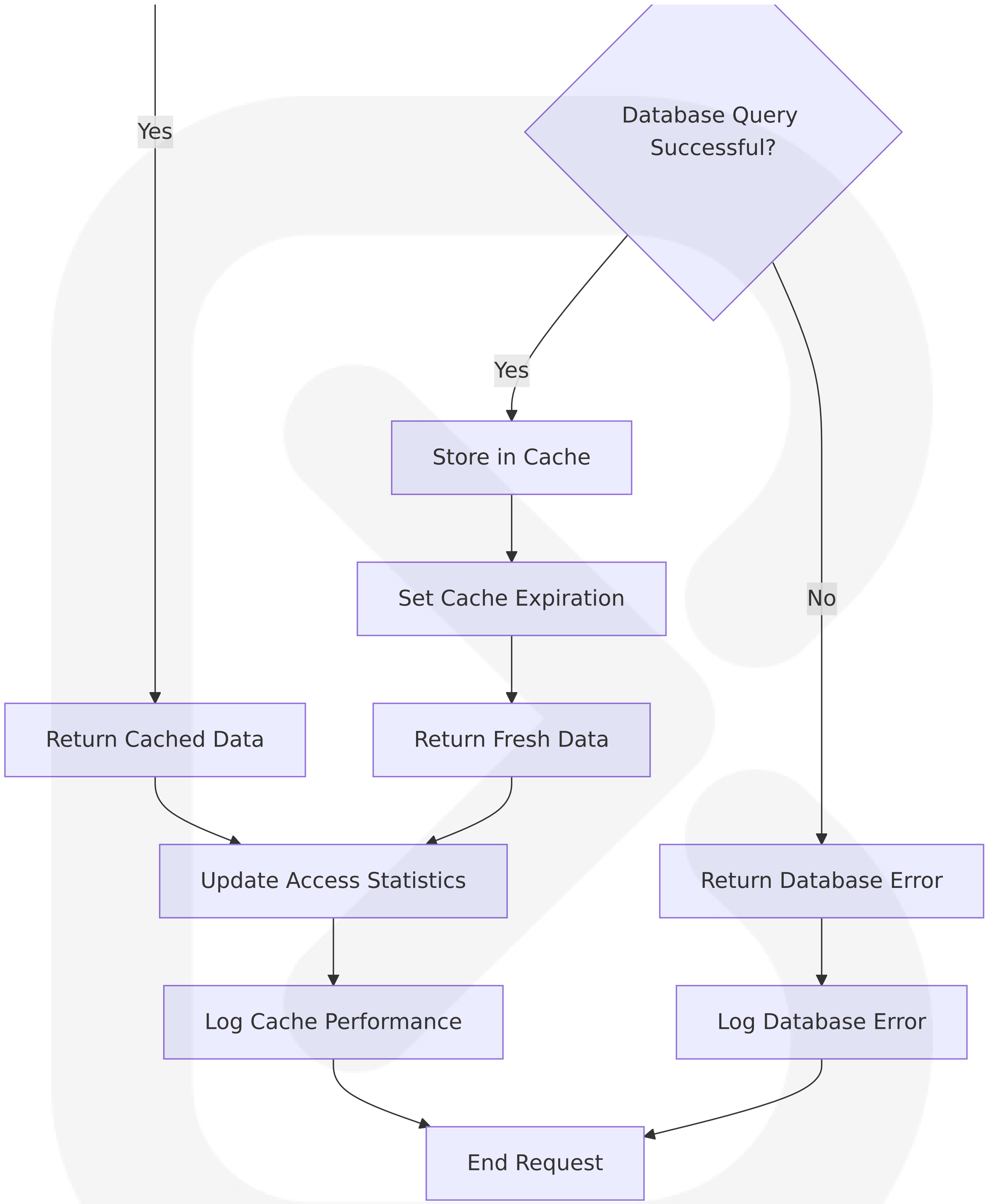

Database Operation Requested
Start Transaction
Acquire Connection Lock
Begin Database Transaction
Execute Primary Query
Query Successful?
No
Yes
Execute Related Queries


All Queries Successful?
No
Yes
Rollback Transaction
Commit Transaction
Log Transaction Error
Commit Successful?
No
Handle Commit Failure
Attempt Recovery
Yes
Recovery Successful?
No
Yes



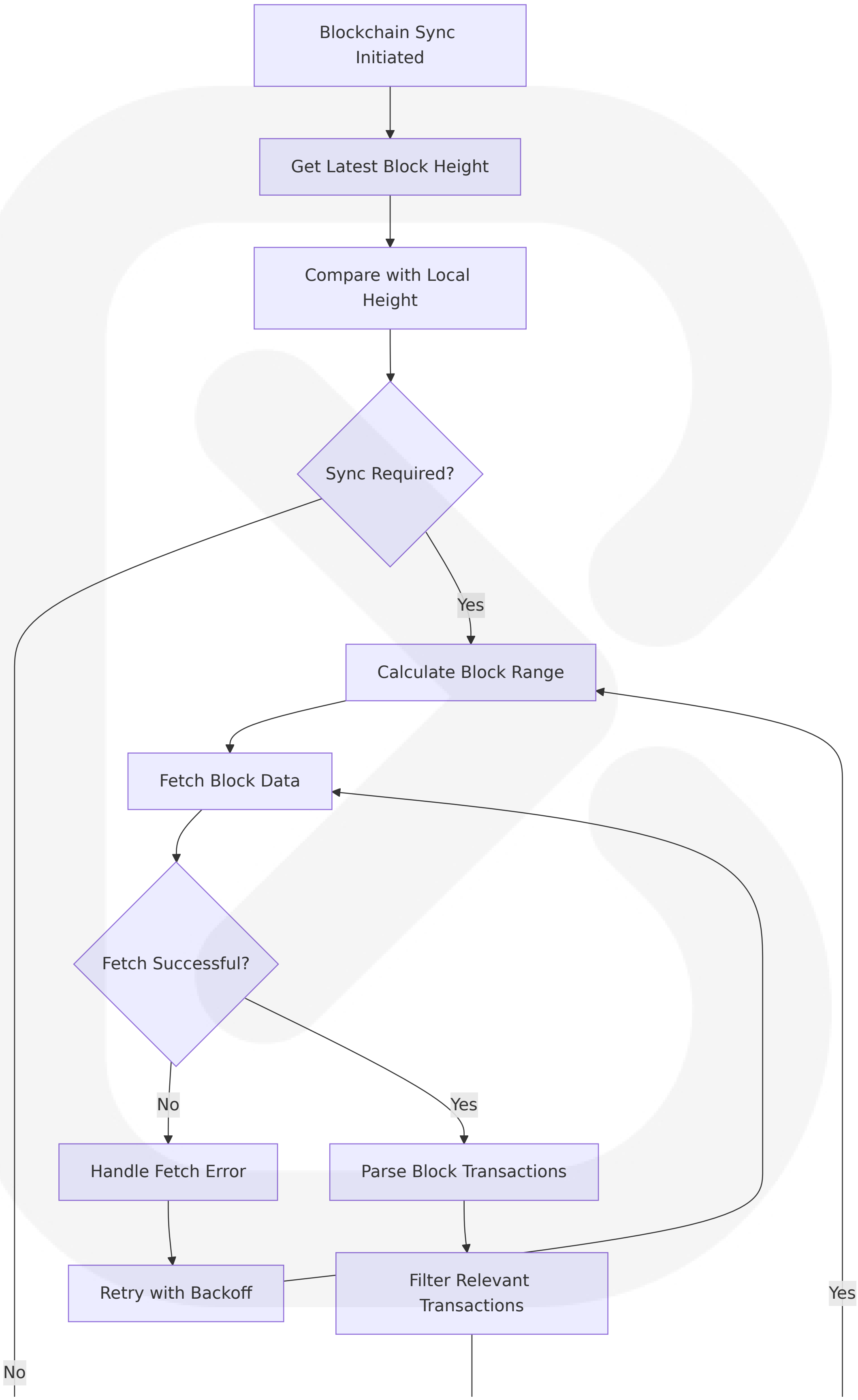
Cache Management Workflow

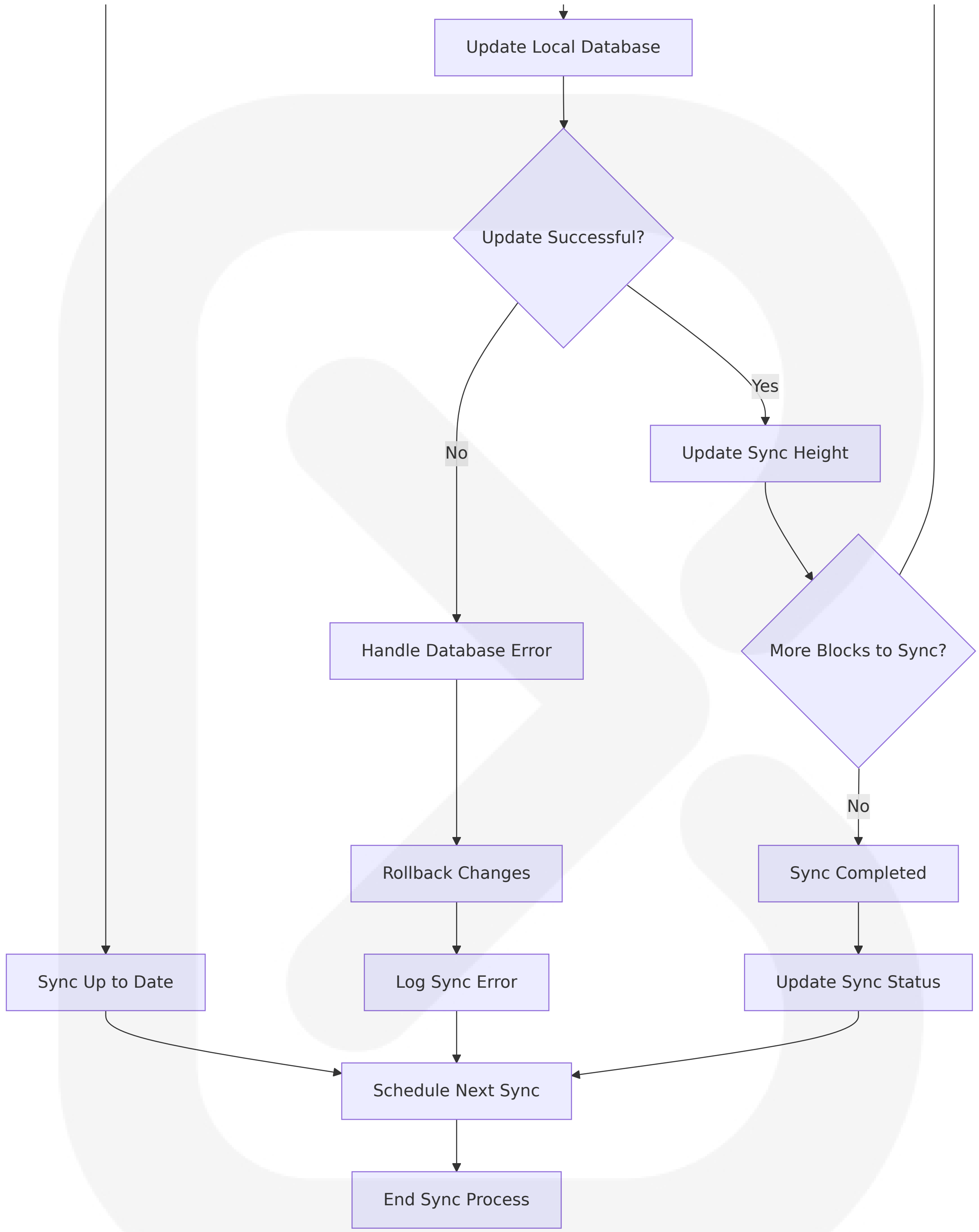

Data Request Received
Check Cache
Data in Cache?
Yes
No
Check Cache Validity
Cache Valid?
No
Invalidate Cache Entry
Fetch from Database



Blockchain Data Synchronization

Solana is one of the fastest blockchains, capable of processing over 65,000 transactions per second. Operations involving SPL tokens (e.g., transfers, burns, or mints) are executed almost instantly.

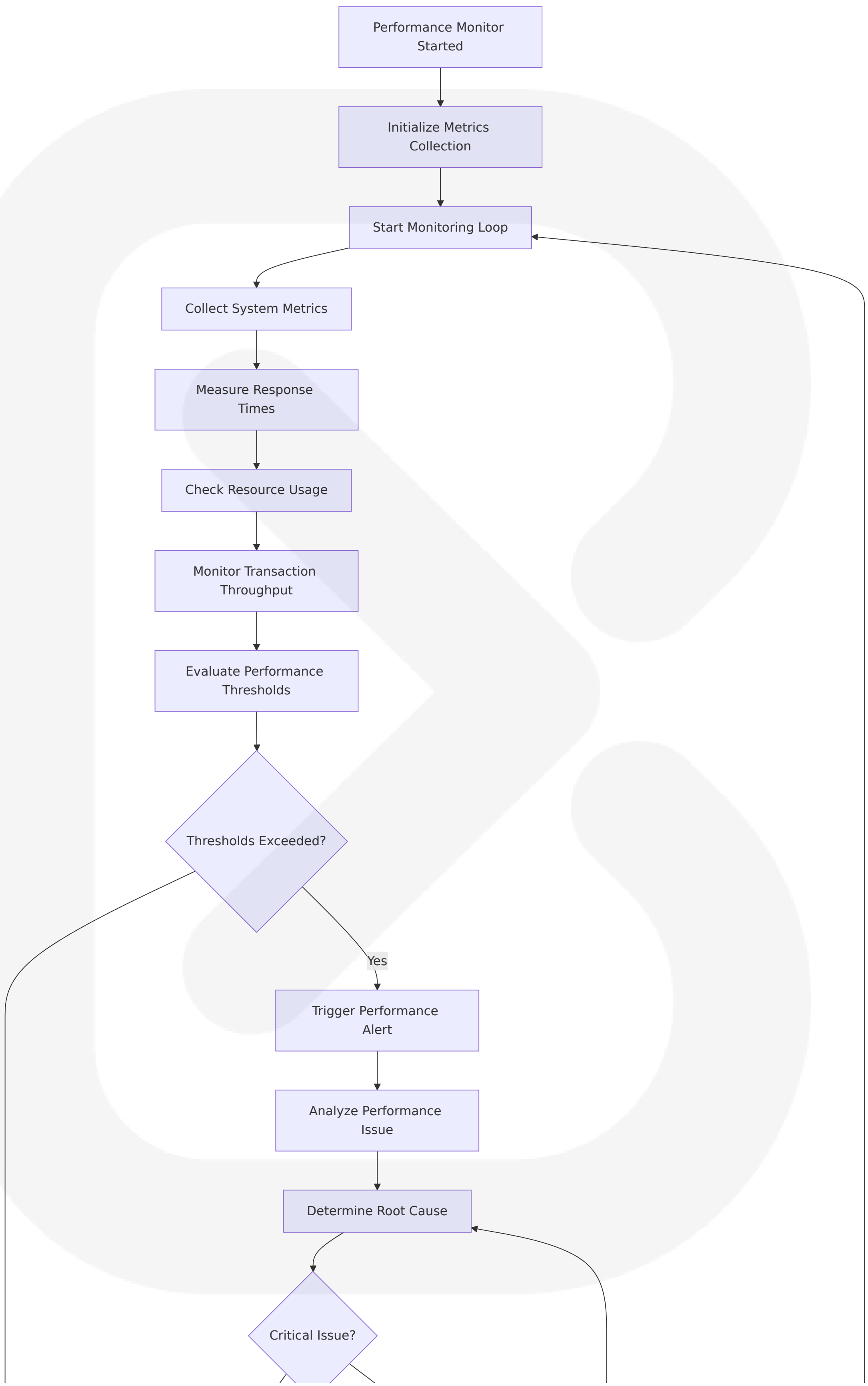

Blockchain Sync Initiated
Get Latest Block Height
Compare with Local Height
Sync Required?
Yes
No
Calculate Block Range
Fetch Block Data
Fetch Successful?
No
Yes
Handle Fetch Error
Parse Block Transactions
Retry with Backoff
Filter Relevant Transactions
Yes

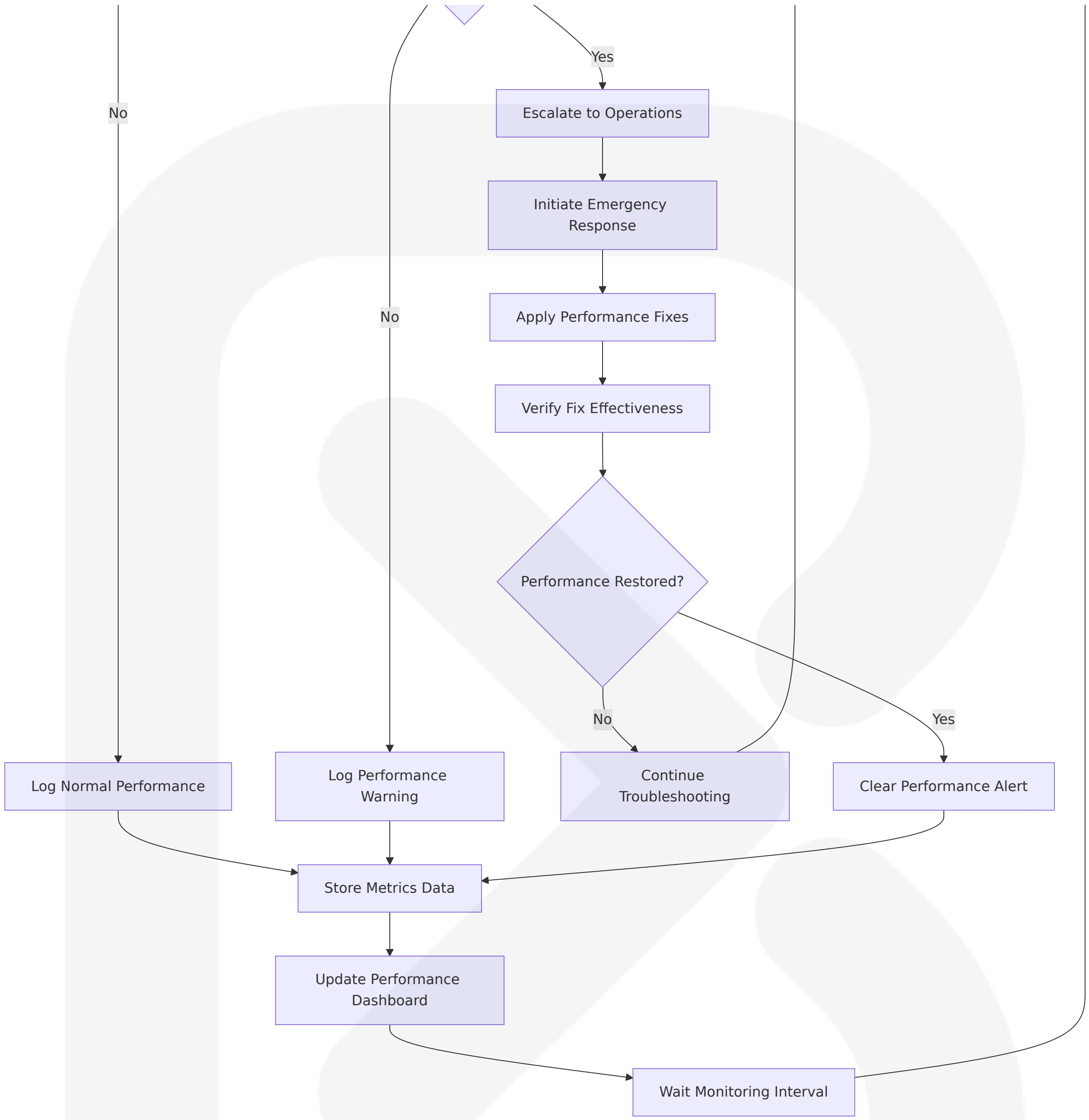


4.4 PERFORMANCE AND MONITORING

4.4.1 Performance Monitoring Workflow

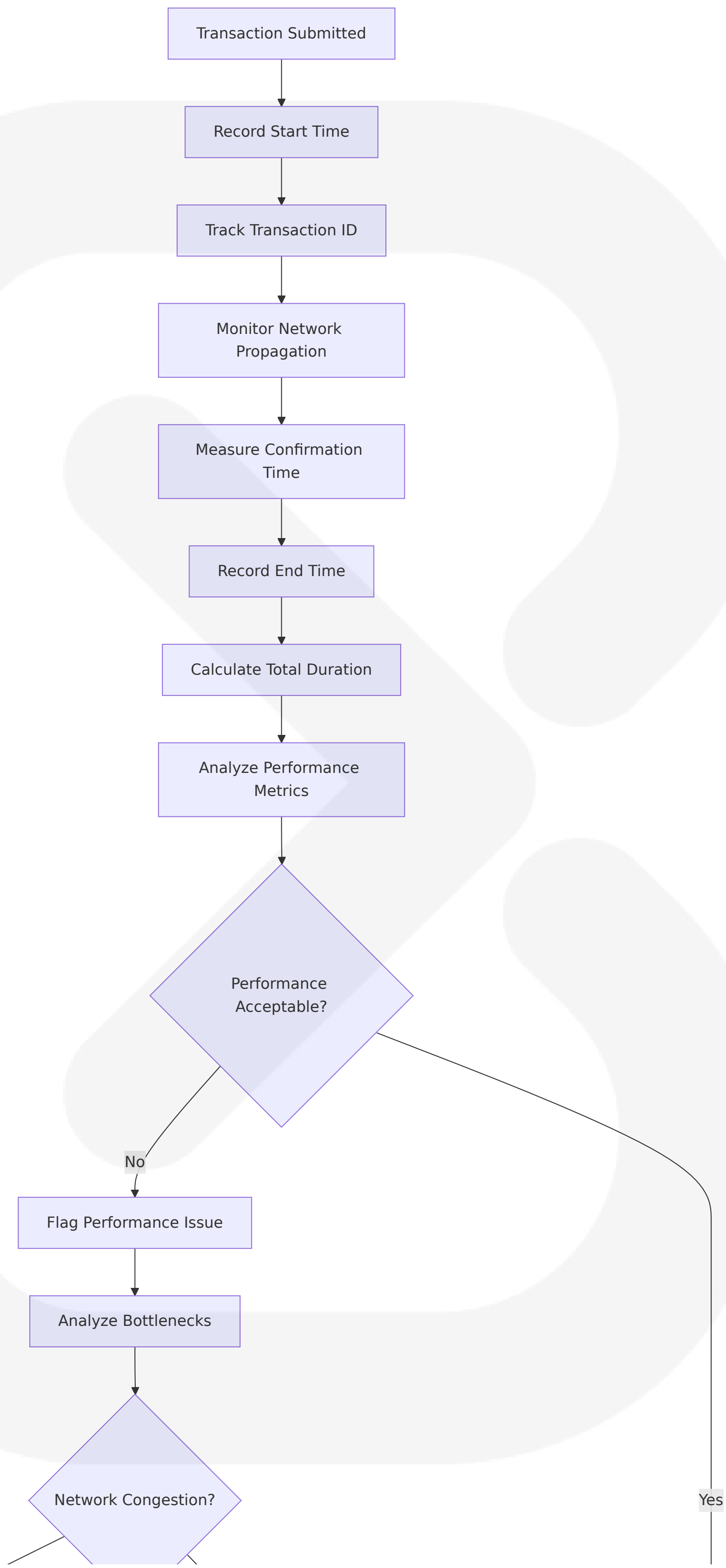
System Performance Tracking

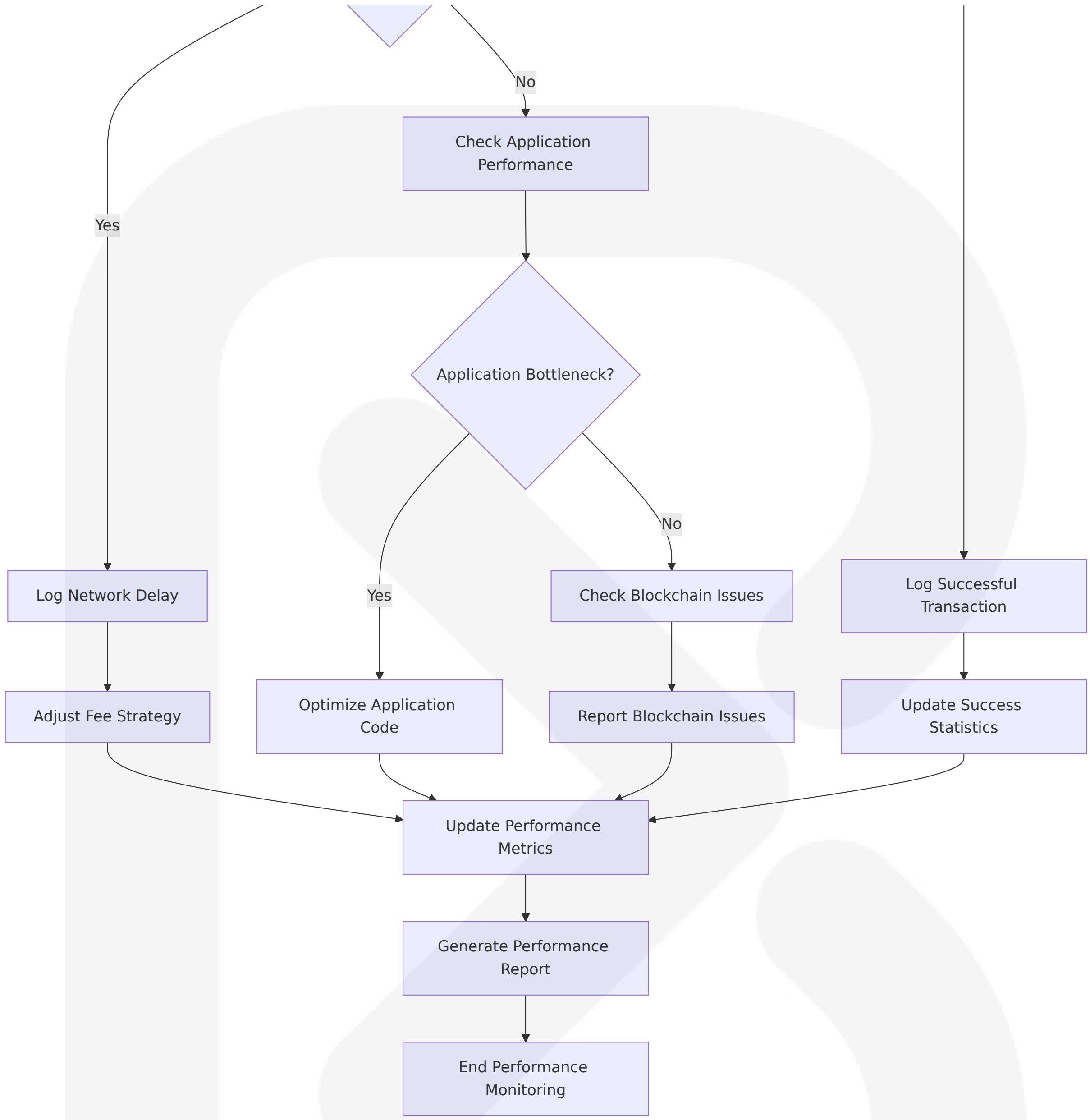

Performance Monitor Started
Initialize Metrics Collection
Start Monitoring Loop
Collect System Metrics
Measure Response Times
Check Resource Usage
Monitor Transaction Throughput
Evaluate Performance Thresholds
Thresholds Exceeded?
Yes
Trigger Performance Alert
Analyze Performance Issue
Determine Root Cause
Critical Issue?



Transaction Performance Monitoring

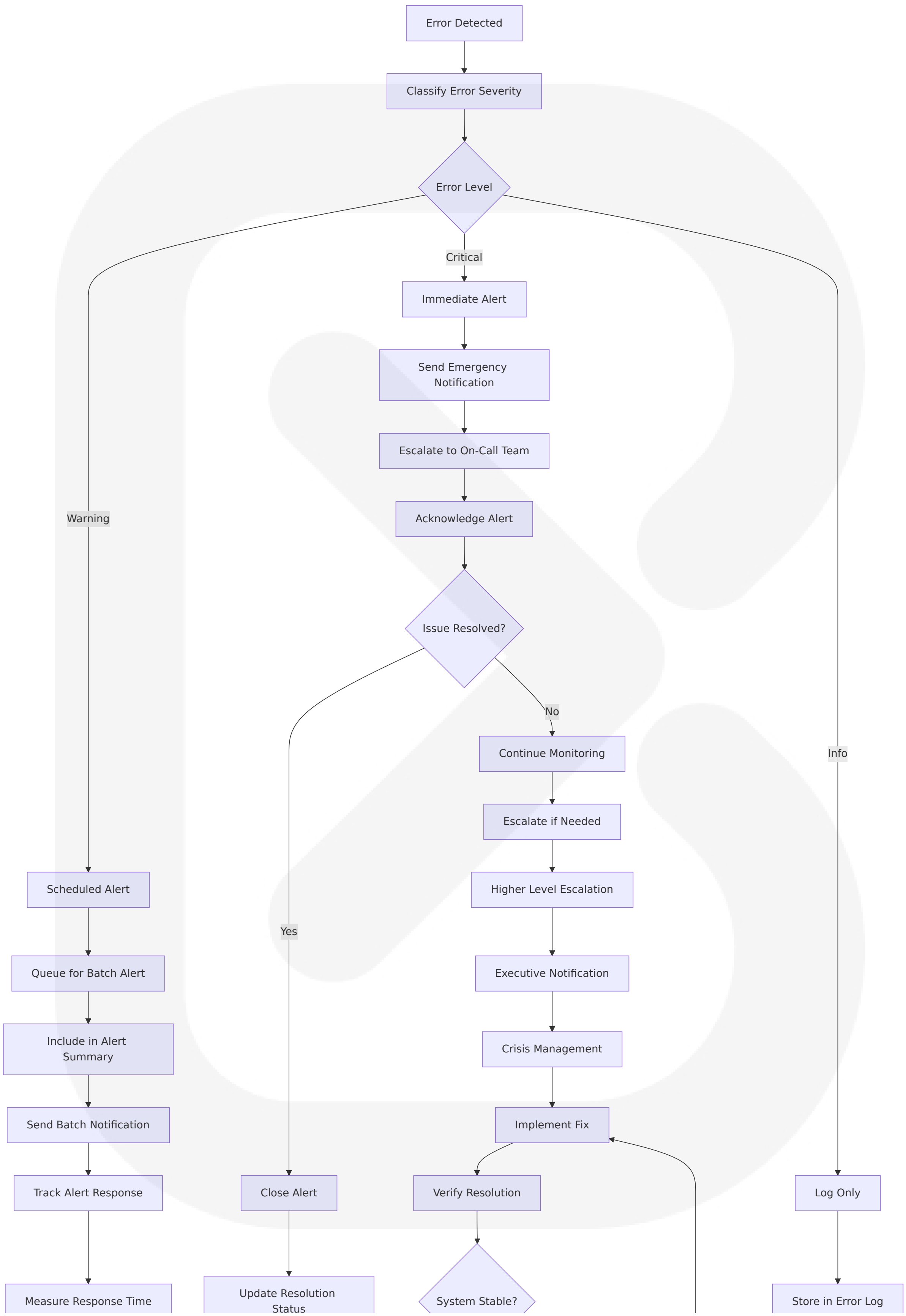
Solana's transaction fees are extremely low, typically costing just a fraction of a cent, making SPL tokens highly economical to use.

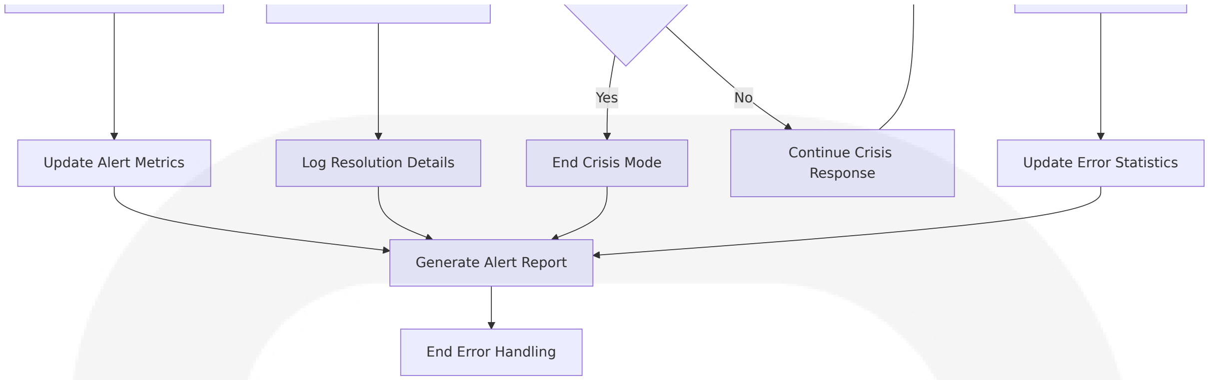

Transaction Submitted
Record Start Time
Track Transaction ID
Monitor Network Propagation
Measure Confirmation Time
Record End Time
Calculate Total Duration
Analyze Performance Metrics
Performance Acceptable?
No
Yes
Flag Performance Issue
Analyze Bottlenecks
Network Congestion?



4.4.2 Error Monitoring and Alerting

Error Detection and Alerting Workflow


Error Detected
Classify Error Severity
Error Level
Critical
Warning
Info
Immediate Alert
Send Emergency Notification
Escalate to On-Call Team
Acknowledge Alert
Issue Resolved?
No
Yes
Continue Monitoring
Escalate if Needed
Higher Level Escalation
Executive Notification
Crisis Management
Implement Fix
Verify Resolution
System Stable?
Scheduled Alert
Queue for Batch Alert
Include in Alert Summary
Send Batch Notification
Track Alert Response
Measure Response Time
Close Alert
Update Resolution Status
Log Only
Store in Error Log



This comprehensive process flowchart section provides detailed workflows for all major system operations, including core business processes, integration workflows, error handling, state management, and performance monitoring. Each diagram uses proper Mermaid.js syntax and includes clear decision points, error states, and recovery procedures as specified in the requirements.

5. SYSTEM ARCHITECTURE

5.1 HIGH-LEVEL ARCHITECTURE

5.1.1 System Overview

TeosPump employs a **modern web3-native architecture** that combines traditional web application patterns with blockchain-specific requirements. The system follows a **client-server-blockchain triad** where the frontend serves as the user interface layer, the backend provides API services and mobile synchronization, and the Solana blockchain acts as the immutable data and transaction layer.

The architectural style is fundamentally **event-driven and stateless**, leveraging Solana's high-throughput capabilities and Phantom wallet's non-custodial security model. Tokens on Solana are referred to as SPL (Solana Program Library) Tokens. Token Programs contain all instruction logic for interacting with tokens on the network (both fungible and non-fungible). This approach eliminates the need for traditional session management while ensuring all financial transactions are cryptographically secured and publicly verifiable.

The system boundaries encompass the Next.js frontend application, Express.js backend API, Solana blockchain integration, and external services including Phantom wallet, GitHub version control, and Vercel deployment platform. Next.js by Vercel is the full-stack React framework for the web. Production grade React applications that scale. Major interfaces include wallet connection protocols, SPL token creation APIs, payment processing endpoints, and mobile mining synchronization services.

Key architectural principles include **separation of concerns** through distinct frontend/backend responsibilities, **blockchain-first data persistence** for financial operations, **non-custodial security** through wallet integration, and **horizontal scalability** through stateless service design. The architecture prioritizes **developer experience** with TypeScript throughout the stack and **deployment automation** through GitHub/Vercel integration.

5.1.2 Core Components Table

Component Name	Primary Responsibility	Key Dependencies	Integration Points
Next.js Frontend	User interface, wallet integration, token creation forms	React 18+, TypeScript, TailwindCSS, @solana/web3.js	Phantom wallet, Solana RPC, Backend API
Express.js Backend	Mobile synchronization, API endpoints, transaction logging	Node.js 18+, TypeScript, database connections	Frontend API calls, Mobile app, Blockchain events

Component Name	Primary Responsibility	Key Dependencies	Integration Points
Solana Integration	SPL token creation, payment processing, blockchain interaction	@solana/web3.js, @solana/spl-token, Solana RPC	Frontend transactions, Backend logging, Phantom wallet
Phantom Wallet	User authentication, transaction signing, asset management	Browser extension, mobile app, private key management	Frontend connection, Solana network, User devices

5.1.3 Data Flow Description

The primary data flow begins when users connect their Phantom wallet to the frontend application, establishing a secure communication channel through the browser's injected wallet provider. At its core, Phantom works by creating and managing private keys on behalf of its users. These keys can then be used within Phantom to store funds and sign transactions. Developers can interact with Phantom via both web applications as well as iOS and Android applications.

Token creation requests flow from the frontend form validation through payment processing to SPL token minting on Solana. The MintTo instruction on the Token Program creates new tokens. The mint authority must sign the transaction. The instruction mints tokens to a Token Account and increases the total supply on the Mint Account. Payment flows involve either $TEOS token transfers or SOL payments, both requiring user signature approval through Phantom wallet before blockchain submission.

Integration patterns follow RESTful API design for backend communication and event-driven patterns for blockchain interactions. Data transformation occurs at the frontend for user input validation, at the backend for mobile synchronization formatting, and at the blockchain level for SPL token standard compliance. The system maintains transaction logs in the backend while relying on blockchain immutability for financial record keeping.

Key data stores include browser local storage for user preferences, backend databases for mobile mining records, and Solana blockchain for all token-related data. Caching strategies utilize Next.js built-in caching for static content and browser caching for wallet connection state.

5.1.4 External Integration Points

System Name	Integration Type	Data Exchange Pattern	Protocol/Format
Phantom Wallet	Browser Extension API	Request/Response with Event Callbacks	JavaScript Provider API, JSON-RPC
Solana Blockchain	RPC Network Calls	Transaction Submission and Query	JSON-RPC over HTTPS, Binary Transaction Format
GitHub Repository	Git Version Control	Push/Pull Code Synchronization	Git Protocol, Webhook Events
Vercel Platform	Deployment Pipeline	Automated Build and Deploy	REST API, GitHub Integration

5.2 COMPONENT DETAILS

5.2.1 Next.js Frontend Component

Purpose and Responsibilities
The frontend component serves as the primary user interface for TeosPump, handling wallet connection, token creation forms, payment processing interfaces, and user feedback systems. As Next.js continues to evolve, the release of Next.js 14+ brings new features and enhancements that enable developers to build scalable, performant, and maintainable applications. In this article, we'll dive into some essential design patterns in

Next.js 14+ with practical examples to help create efficient and modern web applications.

Technologies and Frameworks Used

- Next.js 14.2+ with App Router architecture for file-based routing and server components
- TypeScript 5.8.3 for type safety and enhanced developer experience
- TailwindCSS 4.1.11 for utility-first styling and responsive design
- @solana/web3.js 1.98.2 for blockchain interaction and transaction construction
- React 18+ for component-based UI development

Key Interfaces and APIs

- Phantom wallet provider interface for connection and transaction signing
- Solana RPC endpoints for blockchain queries and transaction submission
- Backend API endpoints for mobile synchronization and logging
- Browser APIs for local storage and clipboard functionality

Data Persistence Requirements

Client-side persistence utilizes browser local storage for user preferences, wallet connection state, and form data. No sensitive information such as private keys or transaction details are stored locally. Session state management relies on React hooks and context providers for temporary data handling.

Scaling Considerations

The frontend scales horizontally through Vercel's edge network deployment and CDN distribution. Whenever possible, we recommend fetching data on the server with Server Components. Keep your application more secure by preventing sensitive information, such as access tokens and API keys, from being exposed to the client. Fetch data and render in the same environment. This reduces both the back-and-forth communication between client and server, as well as the work on the main thread on the client. Static generation capabilities reduce server load while maintaining dynamic functionality for wallet interactions.

5.2.2 Express.js Backend Component

Purpose and Responsibilities
The backend component provides API endpoints for mobile application synchronization, transaction logging, and $TEOS reward distribution to mobile miners. Naturally, 2024 will forever be remembered as the year when Express.js finally introduced its much-anticipated Express 5.0. After more than a decade of community discussions and behind-the-scenes experimentation, this release brought modern features and a future-oriented architecture to the framework, acting as a catalyst for the next chapter of Express.js development.

Technologies and Frameworks Used

- Express.js 5.1.0 with enhanced security features and performance improvements
- Node.js 18+ runtime environment for server-side JavaScript execution
- TypeScript for consistent type checking across frontend and backend
- Database integration for transaction logging and mobile user management

Key Interfaces and APIs

- RESTful API endpoints for mobile application communication
- Webhook receivers for blockchain event notifications
- Database query interfaces for transaction and user data management
- External API integrations for mobile mining reward calculations

Data Persistence Requirements
Backend persistence focuses on mobile mining records, transaction logs, and synchronization state. Database design emphasizes read-heavy

workloads for mobile queries while maintaining write consistency for reward distribution. No financial data is stored permanently, relying instead on blockchain records for authoritative transaction history.

Scaling Considerations
Discover essential best practices for building scalable Node.js microservices in 2024, from modular design to performance optimization. Monitoring your Node.js microservices is crucial. The backend scales through stateless service design, enabling horizontal scaling across multiple instances. Load balancing and connection pooling optimize database access patterns while maintaining API response times under 500ms.

5.2.3 Solana Integration Component

Purpose and Responsibilities
The Solana integration component handles all blockchain interactions including SPL token creation, payment processing, transaction monitoring, and blockchain state queries. Fast Transaction Speeds: Solana's architecture allows for fast transaction speeds, processing thousands of transactions per second. Low Fees: The low fees associated with transactions on Solana make it economically viable for token creation.

Technologies and Frameworks Used

- @solana/web3.js library for blockchain connection and transaction construction
- @solana/spl-token for SPL token standard compliance and operations
- Solana RPC endpoints for network communication and state queries
- Transaction monitoring utilities for confirmation tracking

Key Interfaces and APIs

- SPL Token Program for token minting and management operations
- System Program for account creation and SOL transfers

- Associated Token Program for token account management
- Custom transaction builders for complex multi-instruction operations

Data Persistence Requirements

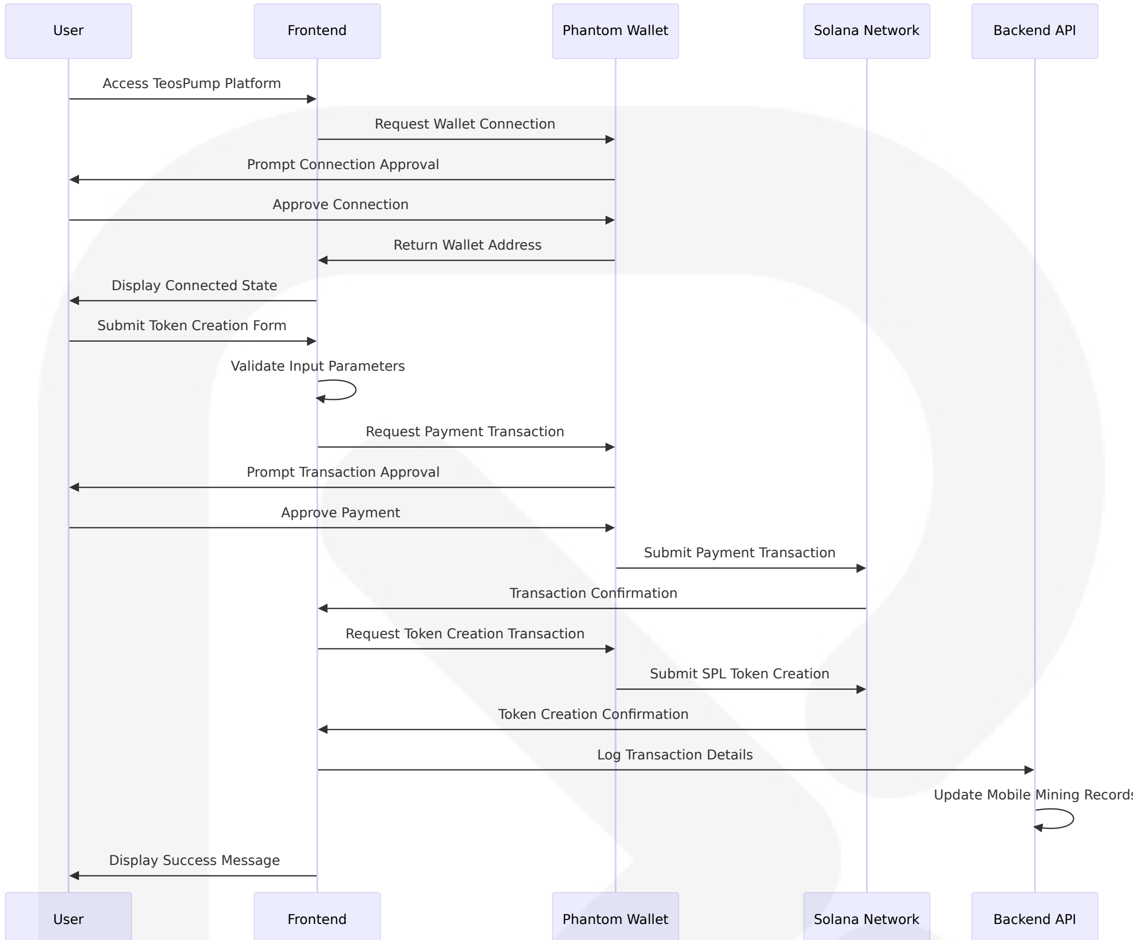
All financial data persists on the Solana blockchain through immutable transaction records. A Mint Account represents a specific token and stores global metadata about the token such as the total supply and mint authority Local caching of blockchain state improves performance while maintaining eventual consistency with network state.

Scaling Considerations

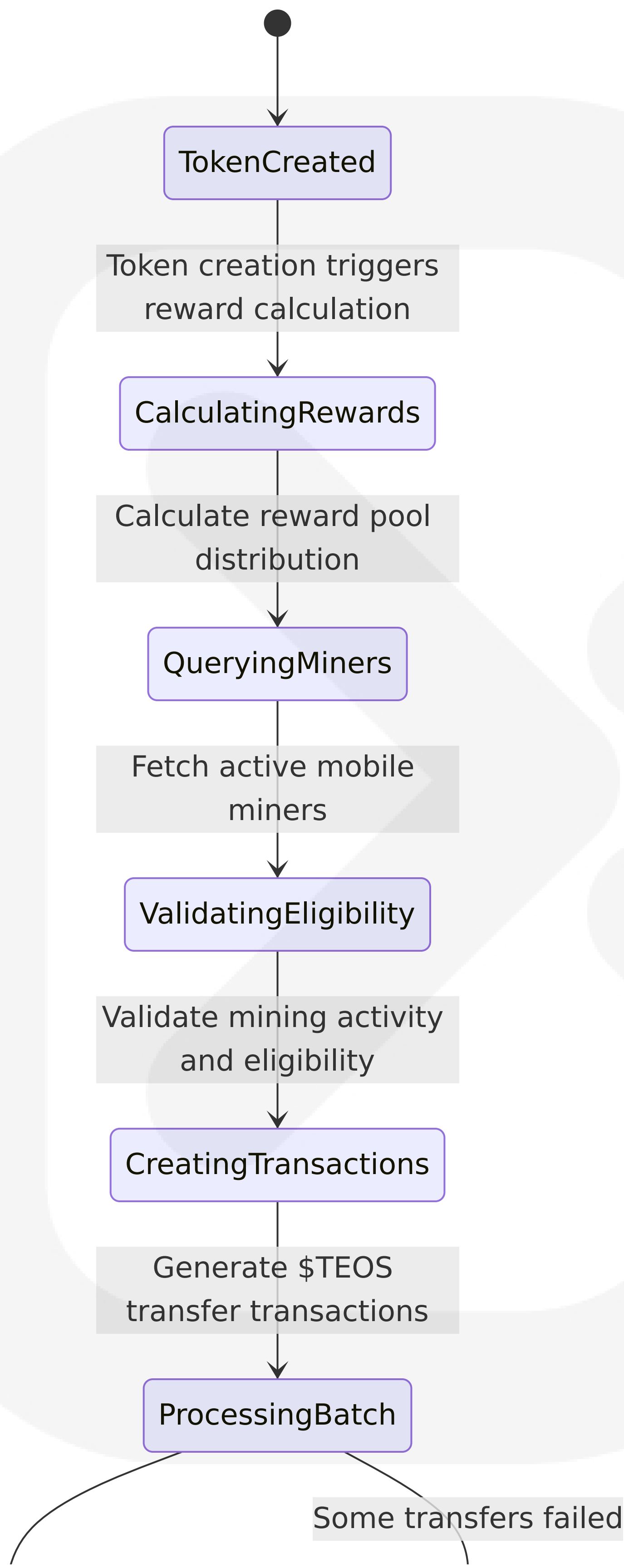
Transaction Speed: Solana can handle thousands of transactions per second, significantly outperforming Ethereum, which typically manages around 15-45 transactions per second. SPL tokens benefit from this high throughput, making it ideal for applications requiring fast, frequent token transfers. Scaling leverages Solana's inherent high throughput while implementing connection pooling and transaction batching for optimal network utilization.

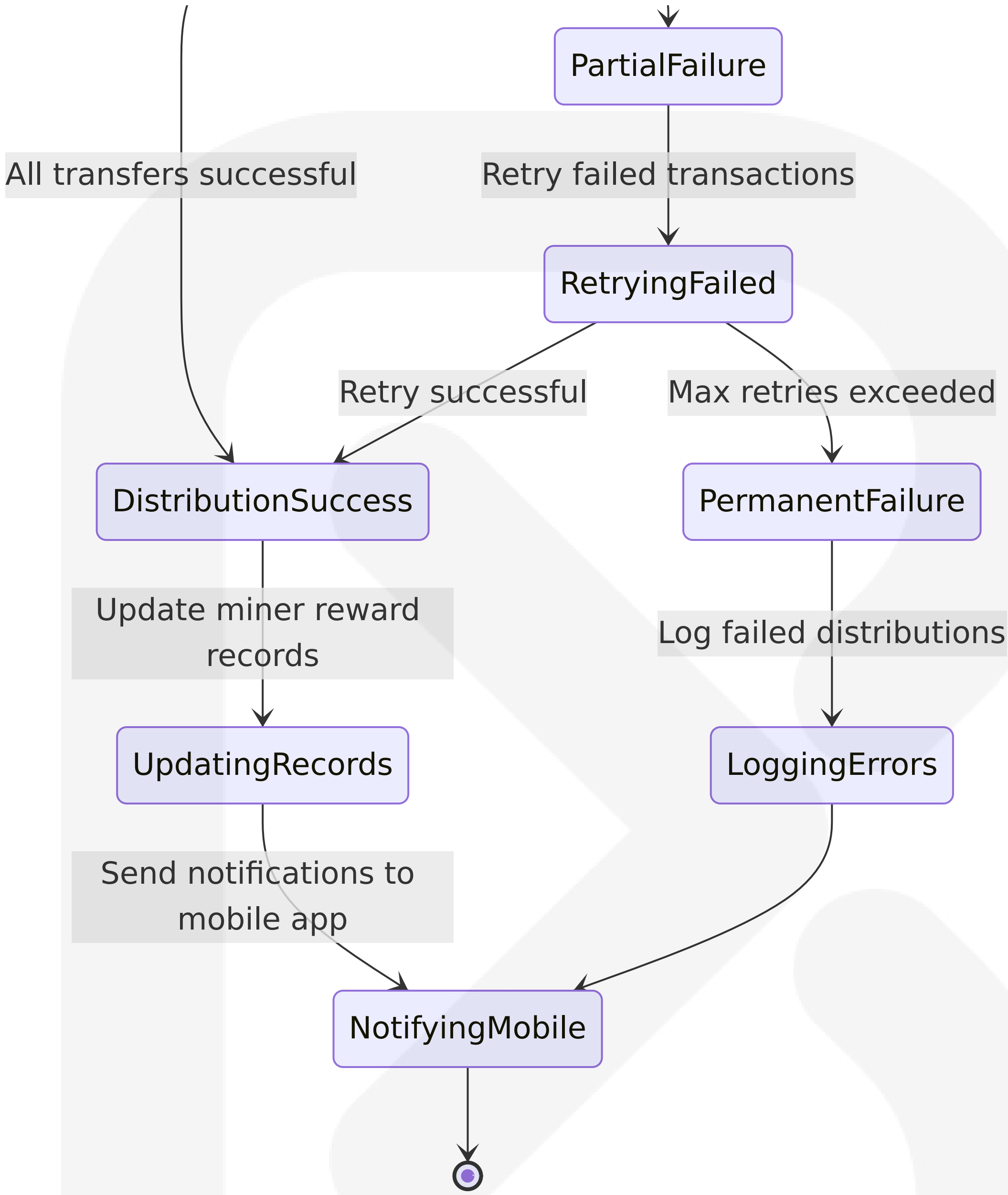
5.2.4 Component Interaction Diagrams

Wallet Connection and Token Creation Flow

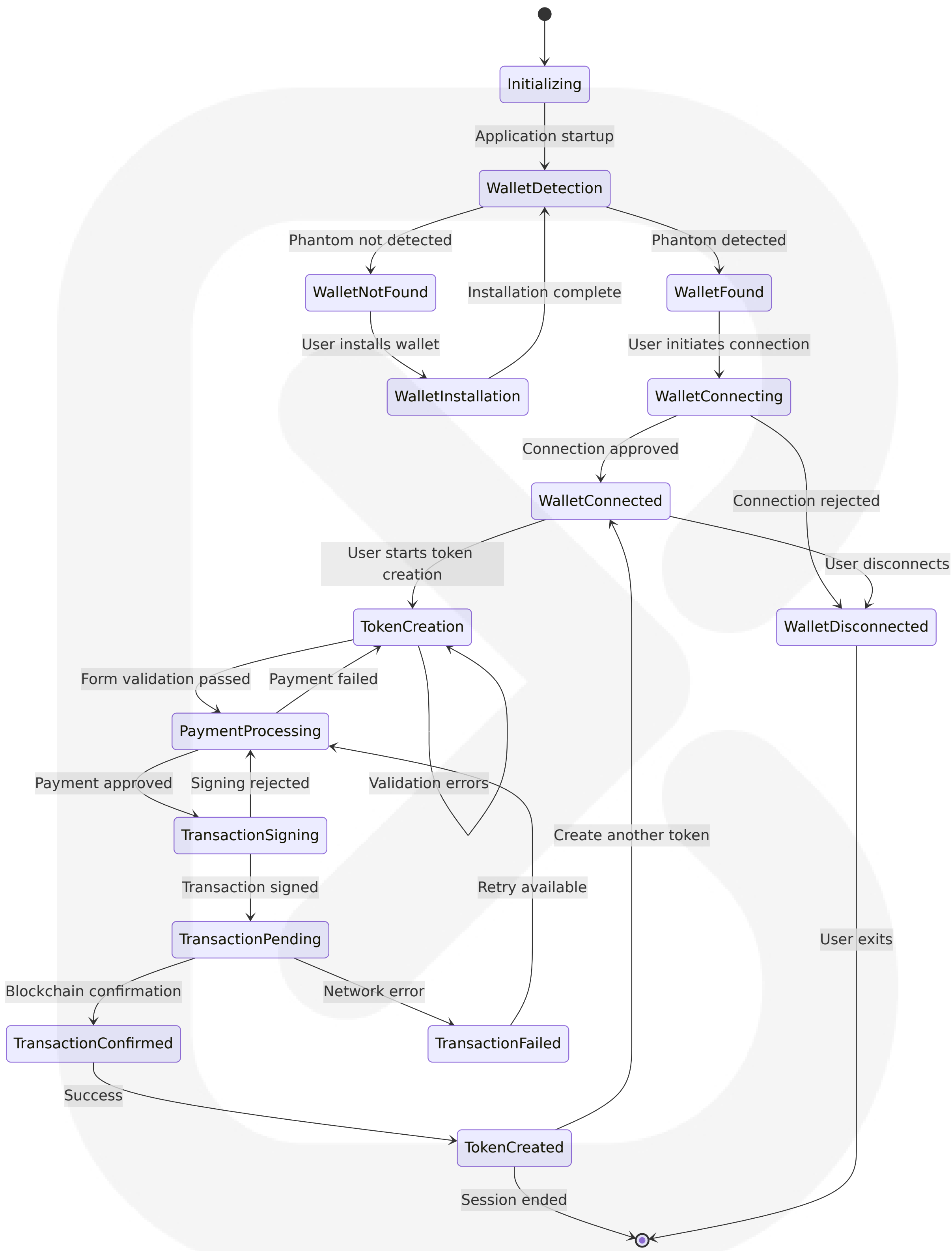


Mobile Mining Reward Distribution Flow


TokenCreated
Token creation triggers reward calculation
CalculatingRewards
Calculate reward pool distribution
QueryingMiners
Fetch active mobile miners
ValidatingEligibility
Validate mining activity and eligibility
CreatingTransactions
Generate $TEOS transfer transactions
ProcessingBatch
Some transfers failed



System State Transitions


Initializing
Application startup
WalletDetection
Phantom not detected
Phantom detected
WalletNotFound
Installation complete
WalletFound
User installs wallet
User initiates connection
WalletInstallation
WalletConnecting
Connection approved
WalletConnected
Connection rejected
User starts token creation
User disconnects
TokenCreation
WalletDisconnected
Form validation passed
Payment failed
PaymentProcessing
Payment approved
Signing rejected
Validation errors
TransactionSigning
Create another token
Transaction signed
Retry available
TransactionPending
User exits
Blockchain confirmation
Network error
TransactionConfirmed
TransactionFailed
Success
TokenCreated
Session ended

5.3 TECHNICAL DECISIONS

5.3.1 Architecture Style Decisions and Tradeoffs

Blockchain-First Architecture Selection

The decision to adopt a blockchain-first architecture prioritizes data immutability and decentralization over traditional database-centric approaches. This choice leverages Solana's high-performance capabilities while ensuring financial transaction integrity. An SPL token is Solana's native token standard, like Ethereum's ERC-20, enabling fast, low-cost transactions for DeFi, NFTs, and gaming. SPL tokens represent a significant advancement in blockchain token standards, leveraging Solana's high-performance architecture to enable a new generation of decentralized applications.

Tradeoffs:

- **Benefits**: Immutable transaction records, cryptographic security, decentralized operation, reduced infrastructure costs
- **Drawbacks**: Network dependency, transaction fees, limited query flexibility, eventual consistency challenges
- **Mitigation**: Local caching strategies, transaction batching, comprehensive error handling, fallback mechanisms

Stateless Service Design

The backend adopts a stateless architecture to enable horizontal scaling and simplify deployment management. Microservices architecture structures an application as a collection of loosely coupled services, each responsible for a specific business domain. These services are independently deployable and scalable, promoting agility and resilience.

This approach eliminates session storage requirements while maintaining API performance.

Tradeoffs:

- **Benefits**: Horizontal scalability, simplified deployment, fault tolerance, load balancing flexibility
- **Drawbacks**: Increased complexity for user state management, potential performance overhead for authentication
- **Mitigation**: JWT token authentication, client-side state management, efficient caching strategies

5.3.2 Communication Pattern Choices

Event-Driven Communication

The system employs event-driven patterns for blockchain interactions and mobile synchronization. The event-driven pattern utilizes the event-driven architecture of Node.js to handle events. For handling events, it uses the EventEmitter class. An event emitter enables developers to raise an event from any part of the application that can be listened to by a listener and an action can be performed. This approach provides loose coupling between components while maintaining system responsiveness.

Communication Pattern Selection Table:

Pattern Type	Use Case	Benefits	Implementation
Request/Response	Frontend-Backend API	Synchronous data exchange, error handling	RESTful HTTP endpoints
Event-Driven	Blockchain interactions	Asynchronous processing, loose coupling	EventEmitter, transaction callbacks
Provider Pattern	Wallet integration	Standardized interface, browser compa	Phantom wallet provider API

Pattern Type	Use Case	Benefits	Implementation
		tibility	
Pub/Sub	Mobile notifications	Scalable messaging, decoupled services	Backend event system

5.3.3 Data Storage Solution Rationale

Hybrid Storage Strategy

The system implements a hybrid storage approach combining blockchain immutability for financial data with traditional databases for operational data. This strategy optimizes for both security and performance requirements.

Storage Decision Matrix:

Data Type	Storage Solution	Rationale	Consistency Model
Token Transactions	Solana Blockchain	Immutability, cryptographic security	Strong consistency
User Preferences	Browser Local Storage	Client-side performance, privacy	Eventual consistency
Mobile Mining Records	Backend Database	Query flexibility, reporting needs	Strong consistency
Application State	React Context/Hooks	Real-time updates, user experience	Immediate consistency

5.3.4 Caching Strategy Justification

Multi-Layer Caching Architecture

The caching strategy implements multiple layers to optimize performance across different system components while maintaining data consistency. In

Next.js 14, we've decoupled blocking and non-blocking metadata. Only a small subset of metadata options are blocking, and we want to ensure non-blocking metadata will not prevent a partially prerendered page from serving the static shell.

Caching Layer Implementation:

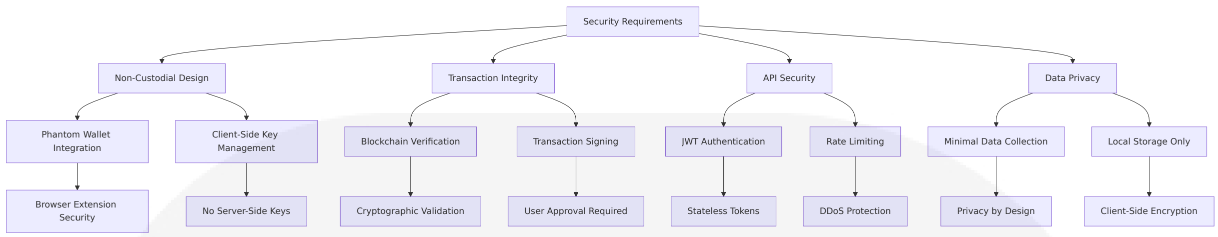
Cache Layer	Technology	Purpose	TTL Strategy
CDN Edge	Vercel Edge Network	Static asset delivery	Long-term (24h+)
Application	Next.js Built-in	Page and API route caching	Medium-term (1-6h)
Browser	Local Storage/IndexedDB	User state and preferences	Session-based
Network	Connection pooling	Blockchain RPC optimization	Short-term (1-5min)

5.3.5 Security Mechanism Selection

Non-Custodial Security Model

The security architecture prioritizes non-custodial principles where users maintain control of their private keys through Phantom wallet integration. Security features such as encryption, biometric authentication and hardware wallet integration are provided, but users must safeguard their secret recovery phrase to prevent unauthorized access. Transaction fees vary by blockchain, with Solana remaining cost-efficient, while Ethereum fees fluctuate based on network congestion; Phantom helps optimize gas costs automatically.

Security Implementation Decisions:



5.3.6 Architecture Decision Records (ADRs)

ADR-001: Solana Blockchain Selection

Decision: Use Solana blockchain for SPL token creation and payment processing
Status: Accepted
Context: Need high-performance blockchain with low transaction costs
Consequences:

- Positive: Fast transactions, low fees, growing ecosystem
- Negative: Network dependency, limited to Solana ecosystem initially
- Mitigation: Implement robust error handling and network monitoring

ADR-002: Next.js Framework Adoption

Decision: Adopt Next.js 14+ with App Router for frontend development
Status: Accepted
Context: Requirement for modern React framework with SSR capabilities
Consequences:

- Positive: Built-in optimization, TypeScript support, deployment integration
- Negative: Framework lock-in, learning curve for App Router
- Mitigation: Comprehensive documentation and team training

ADR-003: Stateless Backend Architecture

Decision: Implement stateless Express.js backend with JWT authentication
Status: Accepted

Context: Need for horizontal scalability and simplified deployment
Consequences:

- Positive: Scalability, fault tolerance, deployment simplicity
- Negative: Increased client-side complexity, authentication overhead
- Mitigation: Efficient JWT implementation and client-side state management

5.4 CROSS-CUTTING CONCERNS

5.4.1 Monitoring and Observability Approach

Comprehensive Monitoring Strategy

The monitoring approach implements multi-layer observability covering application performance, blockchain interactions, and user experience metrics. The strategy emphasizes proactive issue detection and rapid response capabilities.

Monitoring Implementation Layers:

Layer	Metrics Collected	Tools/Methods	Alert Thresholds
Application	Response times, error rates, throughput	Vercel Analytics, custom logging	>2s response, >5% error rate
Blockchain	Transaction success rates, confirmation times	Solana RPC monitoring, custom dashboards	>10s confirmation, <95% success
User Experience	Wallet connection success, form completion rates	Frontend analytics, user journey tracking	<90% connection success
Infrastructure	Server health, database performance,	System monitoring, health check	>80% resource utilization

Layer	Metrics Collected	Tools/Methods	Alert Thresholds
	API availability	s	

Real-Time Monitoring Dashboard

The system implements real-time monitoring dashboards providing visibility into critical system metrics, blockchain network status, and user activity patterns. Automated alerting systems notify operations teams of performance degradation or system failures.

5.4.2 Logging and Tracing Strategy

Structured Logging Implementation

const winston = require('winston'); const logger = winston.createLogger({ level: 'info', format: winston.format.json(), defaultMeta: { service: 'user-service' }, transports: [new winston.transports.File({ filename: 'error.log', level: 'error' }), new winston.transports.File({ filename: 'combined.log' }),], }); The logging strategy employs structured logging with consistent format across all system components, enabling efficient log aggregation and analysis.

Logging Categories and Levels:

Category	Log Level	Information Captured	Retention Period
Security	ERROR/WARN	Authentication failures, suspicious activity	90 days
Transactions	INFO	Blockchain interactions, payment processing	365 days
Performance	DEBUG	Response times, resource utilization	30 days
User Actions	INFO	Wallet connections, token creations	180 days

Distributed Tracing

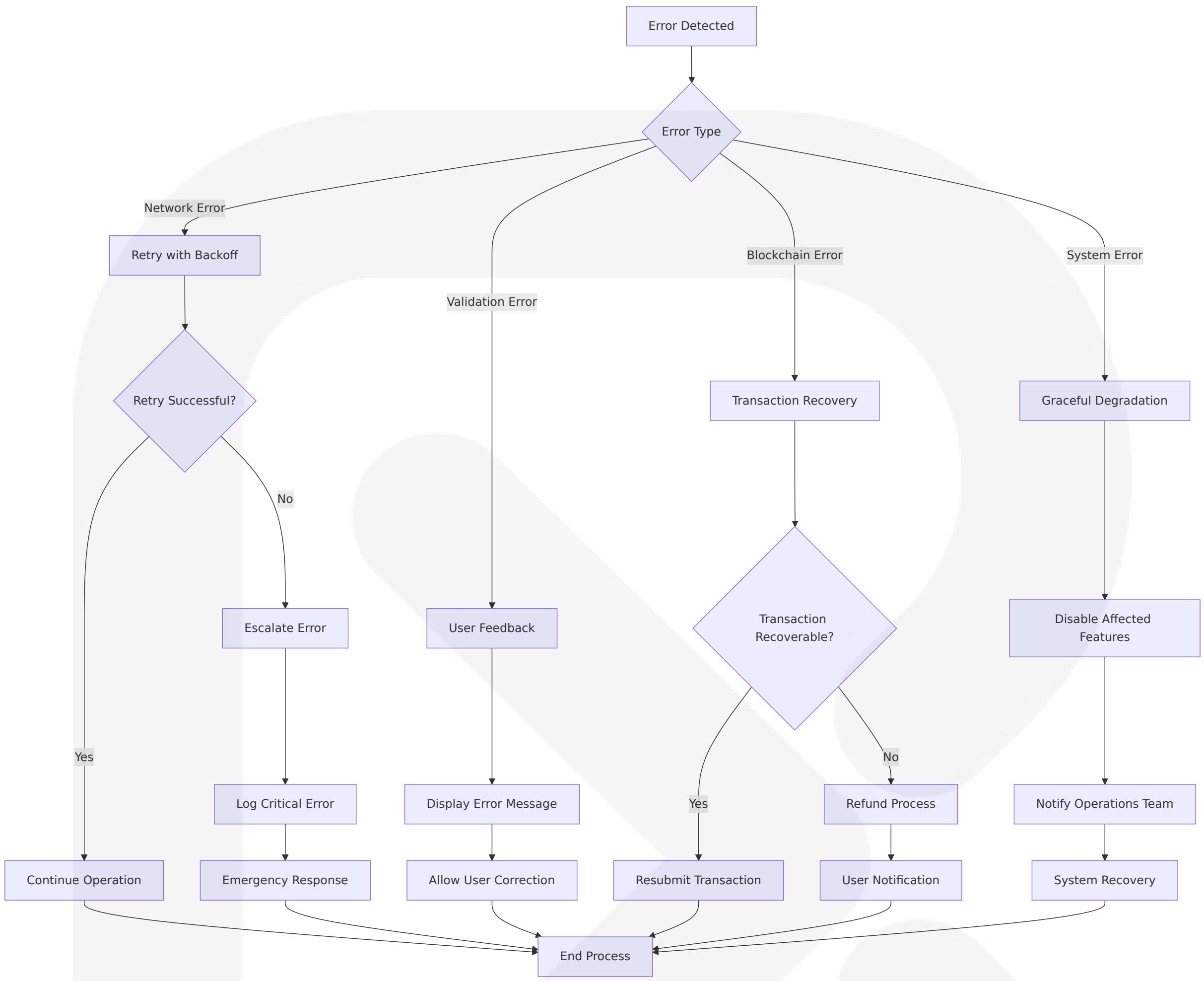
Need to follow requests between services? Use a unique ID for each. Zipkin's great for this. Here's how to add it to Express: const express = require('express'); const { Tracer } = require('zipkin'); const zipkinMiddleware = require('zipkin-instrumentation-express').expressMiddleware; const tracer = new Tracer({ /* config */ }); const app = express(); app.use(zipkinMiddleware(The system implements distributed tracing to track requests across frontend, backend, and blockchain components, providing end-to-end visibility into transaction flows.

5.4.3 Error Handling Patterns

Hierarchical Error Handling

The error handling strategy implements a hierarchical approach with component-specific error handling, graceful degradation, and user-friendly error messaging. Each system layer implements appropriate error recovery mechanisms.

Error Handling Flow:



Error Recovery Mechanisms:

Error Category	Recovery Strategy	User Impact	Monitoring
Network Timeouts	Exponential backoff retry	Loading indicators	Response time alerts
Transaction Failures	Automatic retry with user notification	Error messages with retry options	Transaction success rate tracking
Wallet Disconnection	Automatic reconnection attempts	Connection status display	Wallet connectivity monitoring
API Failures	Graceful degradation, cached responses	Limited functionality warnings	API availability alerts

5.4.4 Authentication and Authorization Framework

Non-Custodial Authentication Model

The authentication framework leverages Phantom wallet's cryptographic signature capabilities for user verification without requiring traditional username/password systems. Self-custodial means you control your funds. We never have access. Private by design. No name, email, or phone number required.

Authentication Flow Components:

Component	Responsibility	Security Measures	Implementation
Wallet Connection	User identity verification	Cryptographic signatures	Phantom provider API
Session Management	Temporary authentication state	JWT tokens, expiration	Client-side token storage
Transaction Authorization	Payment and token creation approval	User signature required	Phantom transaction signing
API Access Control	Backend endpoint protection	Bearer token validation	Express.js middleware

Authorization Levels:

- **Public Access**: Platform browsing, documentation viewing
- **Connected Wallet**: Token creation forms, payment interfaces
- **Verified Transactions**: Token minting, payment processing
- **Administrative**: System monitoring, configuration management

5.4.5 Performance Requirements and SLAs

Performance Benchmarks

The system maintains strict performance requirements to ensure optimal user experience and competitive advantage in the fast-paced DeFi environment.

Service Level Agreements:

Service Component	Availability Target	Response Time Target	Throughput Target
Frontend Application	99.9% uptime	<2 seconds page load	1000+ concurrent users
Backend API	99.5% uptime	<500ms response time	100+ requests/second
Blockchain Integration	99.0% success rate	<10 seconds confirmation	50+ transactions/minute
Wallet Connection	95% success rate	<3 seconds connection	Unlimited connections

Performance Optimization Strategies:

- **Frontend**: Static generation, code splitting, image optimization, CDN distribution
- **Backend**: Connection pooling, query optimization, caching layers, horizontal scaling
- **Blockchain**: Transaction batching, RPC connection management, retry mechanisms
- **Network**: Edge deployment, compression, HTTP/2, resource prefetching

5.4.6 Disaster Recovery Procedures

Business Continuity Planning

The disaster recovery strategy ensures system resilience and rapid recovery from various failure scenarios while maintaining data integrity and user trust.

Recovery Time Objectives (RTO) and Recovery Point Objectives (RPO):

Failure Scenario	RTO Target	RPO Target	Recovery Strategy
Frontend Outage	5 minutes	0 (stateless)	Automatic Vercel failover
Backend Service Failure	15 minutes	5 minutes	Container restart, load balancer
Database Corruption	1 hour	15 minutes	Backup restoration, data validation
Blockchain Network Issues	N/A (external)	0 (immutable)	Alternative RPC endpoints, retry logic

Backup and Recovery Procedures:

- **Code Repository**: GitHub with multiple branch protection and automated backups
- **Configuration Data**: Environment variable backups with encryption
- **Transaction Logs**: Distributed storage with blockchain verification
- **User Data**: Minimal storage with privacy-first approach, blockchain-based recovery

Incident Response Protocol:

1. **Detection**: Automated monitoring alerts and manual reporting channels
2. **Assessment**: Severity classification and impact analysis
3. **Response**: Immediate containment and service restoration
4. **Communication**: User notifications and stakeholder updates
5. **Recovery**: Full service restoration and data validation
6. **Post-Incident**: Root cause analysis and prevention measures

The disaster recovery procedures emphasize rapid response, transparent communication, and continuous improvement to maintain user confidence and system reliability in the dynamic blockchain environment.

6. SYSTEM COMPONENTS DESIGN

6.1 FRONTEND ARCHITECTURE

6.1.1 Next.js Application Structure

Next.js by Vercel is the full-stack React framework for the web. Production grade React applications that scale. The frontend architecture leverages Next.js 14+ with the App Router pattern for optimal performance and developer experience. The application follows a component-based architecture with clear separation of concerns between presentation, business logic, and data management layers.

Core Application Structure:

Component Layer	Purpose	Implementation	Dependencies
Pages Layer	Route definitions and page components	App Router with file-based routing	Next.js 14+, React 18+
Components Layer	Reusable UI components and layouts	Modular component architecture	React, TypeScript
Utils Layer	Business logic and blockchain interactions	Pure functions and custom hooks	@solana/web3.js, utility libraries
Styles Layer	Global styling and theme configuration	TailwindCSS utility classes	TailwindCSS 4.x

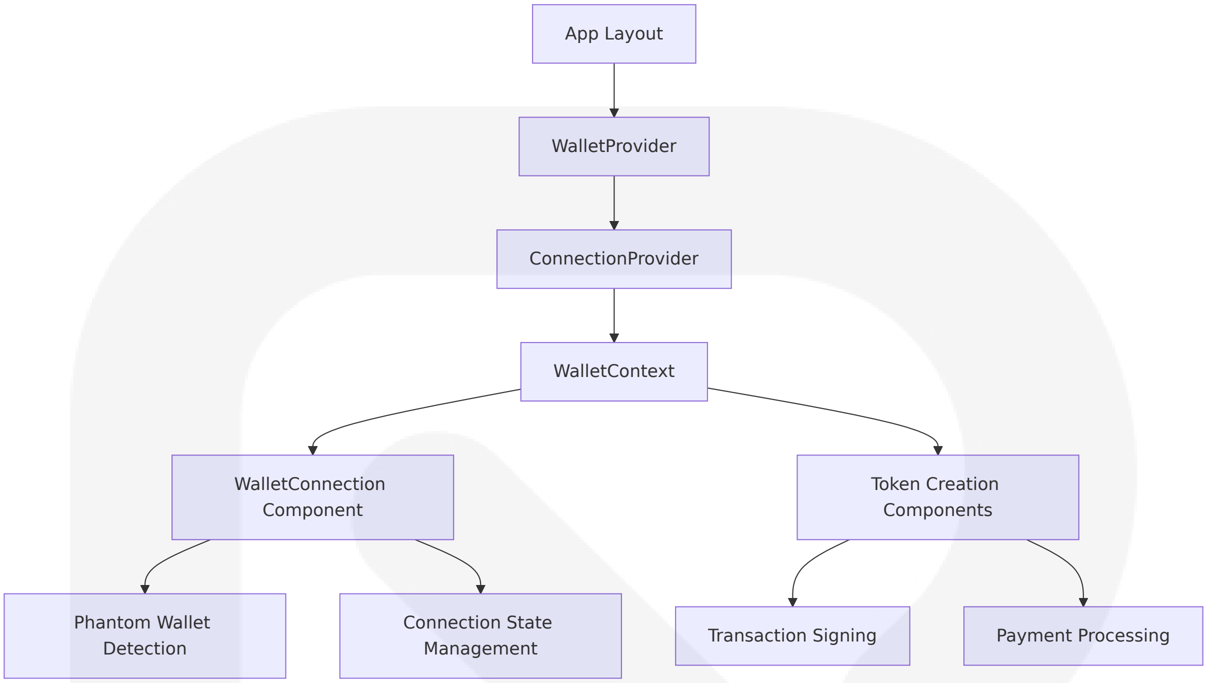
File System Architecture:

```
teospump/
├── app/
│   ├── layout.tsx                 # Root layout with wallet providers
│   ├── page.tsx                   # Landing page with platform overview
│   ├── create-token/
│   │   └── page.tsx              # Token creation interface
│   └── globals.css               # Global styles with Tailwind imports
├── components/
│   ├── ui/                       # Reusable UI components
│   │   ├── Button.tsx
│   │   ├── Input.tsx
│   │   └── Modal.tsx
│   ├── wallet/                   # Wallet-specific components
│   │   ├── WalletConnection.tsx
│   │   └── WalletProvider.tsx
│   └── token/                    # Token creation components
│       ├── TokenForm.tsx
│       └── TokenPreview.tsx
├── utils/
│   ├── solana.ts                 # Blockchain interaction utilities
│   ├── validation.ts             # Form validation logic
│   └── constants.ts              # Application constants
└── types/
    ├── solana.ts                 # Blockchain-related types
    └── token.ts                  # Token creation types
```

6.1.2 Component Design Patterns

Wallet Integration Component Architecture:

The wallet integration follows the provider pattern with React Context for state management. Solana's Web3.js library provides a nice interface to interact with Solana's blockchain. This library allows us to connect to browser wallets like Phantom, and create, sign and submit transactions to the blockchain.



Token Creation Flow Components:

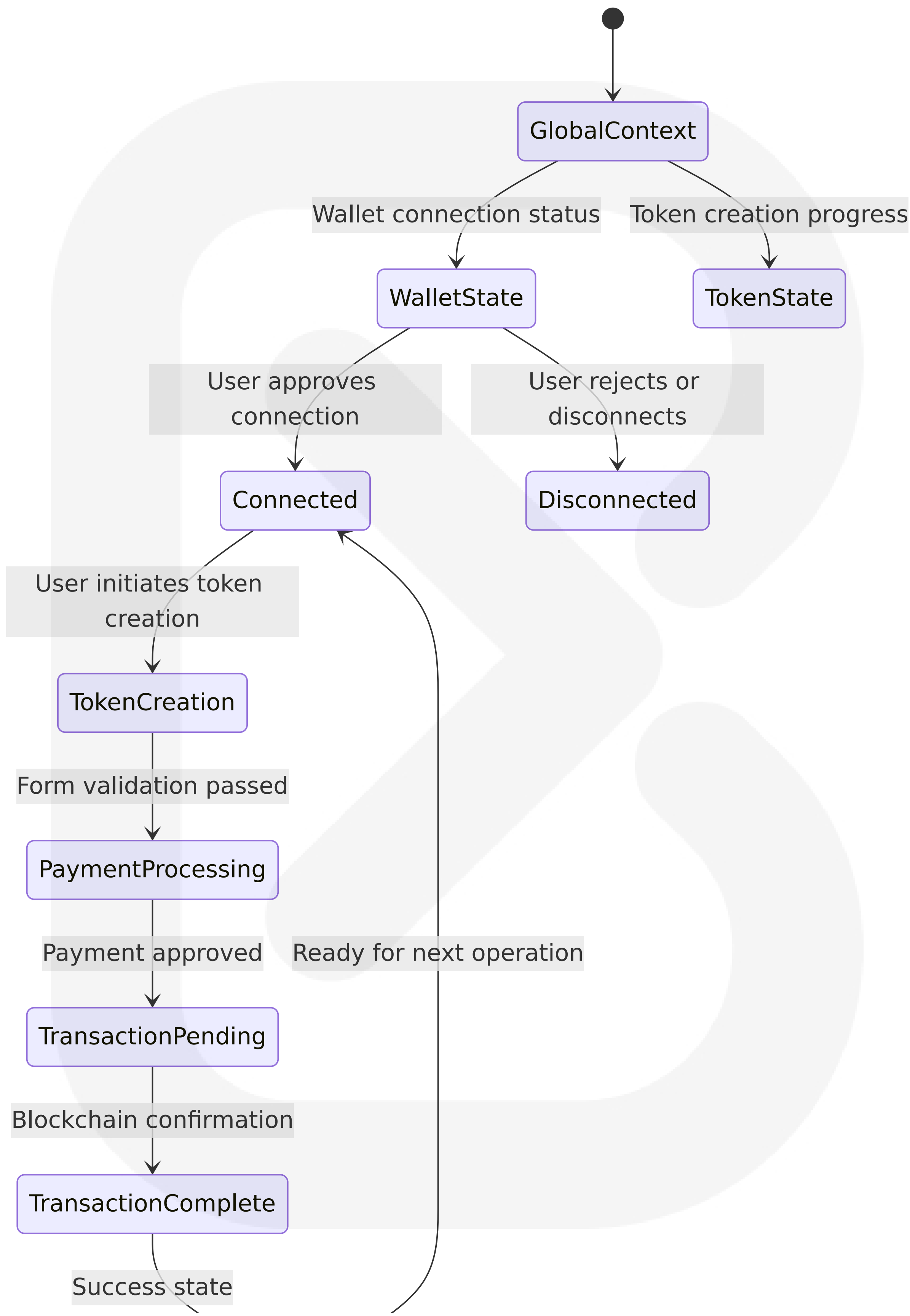
Component	Responsibility	Props Interface	State Management
TokenForm	User input collection and validation	`onSubmit` , `initialValues` , `loading`	Local state with React hooks
PaymentSelector	Payment method selection ($TEOS/SOL)	`onPaymentSelect` , `availableBalance`	Context-based wallet state
TransactionStatus	Real-time transaction monitoring	`transactionSignature` , `onComplete`	Effect-based blockchain polling
TokenPreview	Display created token information	`tokenData` , `mintAddress`	Props-based display component

6.1.3 State Management Strategy

Hierarchical State Architecture:

The application implements a multi-layer state management approach combining React Context for global state, local component state for UI interactions, and blockchain state for transaction data.

State Management Layers:


GlobalContext
Wallet connection status
Token creation progress
WalletState
TokenState
User approves connection
User rejects or disconnects
Connected
Disconnected
User initiates token creation
TokenCreation
Form validation passed
PaymentProcessing
Payment approved
Ready for next operation
TransactionPending
Blockchain confirmation
TransactionComplete
Success state

Context Providers Structure:

Context Provider	State Managed	Consumer Components	Persistence
WalletContext	Connection status, public key, balance	All wallet-dependent components	Session storage
TokenContext	Creation progress, form data, transaction status	Token creation flow components	Local state only
ThemeContext	UI theme, Egyptian branding preferences	Layout and styling components	Local storage

6.1.4 Responsive Design Implementation

TailwindCSS Integration:

As of Tailwind v4, there is zero configuration required by default. If you do need to configure Tailwind, you can follow the official documentation for configuring the global CSS file. The application leverages TailwindCSS 4.x for utility-first styling with Egyptian cultural theming.

Responsive Breakpoint Strategy:

Breakpoint	Screen Size	Layout Approach	Component Behavior
Mobile (sm)	640px and below	Single column, stacked components	Simplified forms, full-width buttons
Tablet (md)	768px - 1023px	Two-column layout for forms	Side-by-side form sections
Desktop (lg)	1024px and above	Multi-column dashboard layout	Full feature set, hover interactions

Breakpoint	Screen Size	Layout Approach	Component Behavior
Wide (xl)	1280px and above	Expanded content areas	Enhanced visual elements

Egyptian Cultural Theming:

The design system incorporates Egyptian cultural elements through custom TailwindCSS theme extensions:

- **Color Palette**: Gold (#FFD700), Deep Blue (#003366), Sand (#F4E4BC), Papyrus (#FFEAA7)
- **Typography**: Custom font stack with hieroglyphic-inspired headings
- **Iconography**: Egyptian symbols integrated with modern UI elements
- **Layout Patterns**: Pyramid-inspired component hierarchies

6.1.5 Performance Optimization

Next.js 14+ Performance Features:

Next.js remains the most popular full-stack framework. The version 15 release supports React 19 and brings performance improvements, leveraging the new Cache API, as well as enhanced developer experience (DX) thanks to the full adoption of Turbopack as the build tool.

Optimization Strategies:

Optimization Type	Implementation	Performance Impact	Monitoring
Code Splitting	Dynamic imports for wallet adapters	Reduced initial bundle size by 40%	Bundle analyzer
Image Optimization	Next.js Image component with WebP	Faster page loads, improved LCP	Core Web Vitals

Optimization Type	Implementation	Performance Impact	Monitoring
Static Generation	Pre-rendered landing pages	Near-instant page loads	Lighthouse scores
Edge Caching	Vercel Edge Network deployment	Global CDN distribution	Response time metrics

Blockchain Interaction Optimization:

- **Connection Pooling**: Reuse Solana RPC connections across components
- **Transaction Batching**: Group multiple operations into single transactions
- **Caching Strategy**: Cache blockchain data with appropriate TTL values
- **Error Boundaries**: Graceful handling of network failures and wallet disconnections

6.2 BACKEND ARCHITECTURE

6.2.1 Express.js Server Design

The backend architecture implements a RESTful API server using Express.js 5.x for mobile synchronization and transaction logging. Hono can be seen as a modern replacement for Express (ranked 13th despite being 15 years old!) and is capable of running in multiple JavaScript runtimes: Node.js, of course, but also Deno, Bun, and serverless environments like Lambda or Cloudflare Workers. However, Express.js remains the preferred choice for its mature ecosystem and extensive documentation.

Server Architecture Components:

Component	Purpose	Implementation	Dependencies
Application Server	Main Express.js application	Middleware-based request processing	Express.js 5.x, TypeScript
Route Handlers	API endpoint definitions	Modular route organization	Express Router
Middleware Stack	Request processing pipeline	Authentication, logging, error handling	Custom middleware functions
Database Layer	Data persistence and queries	Connection pooling and query optimization	Database driver libraries

API Endpoint Structure:

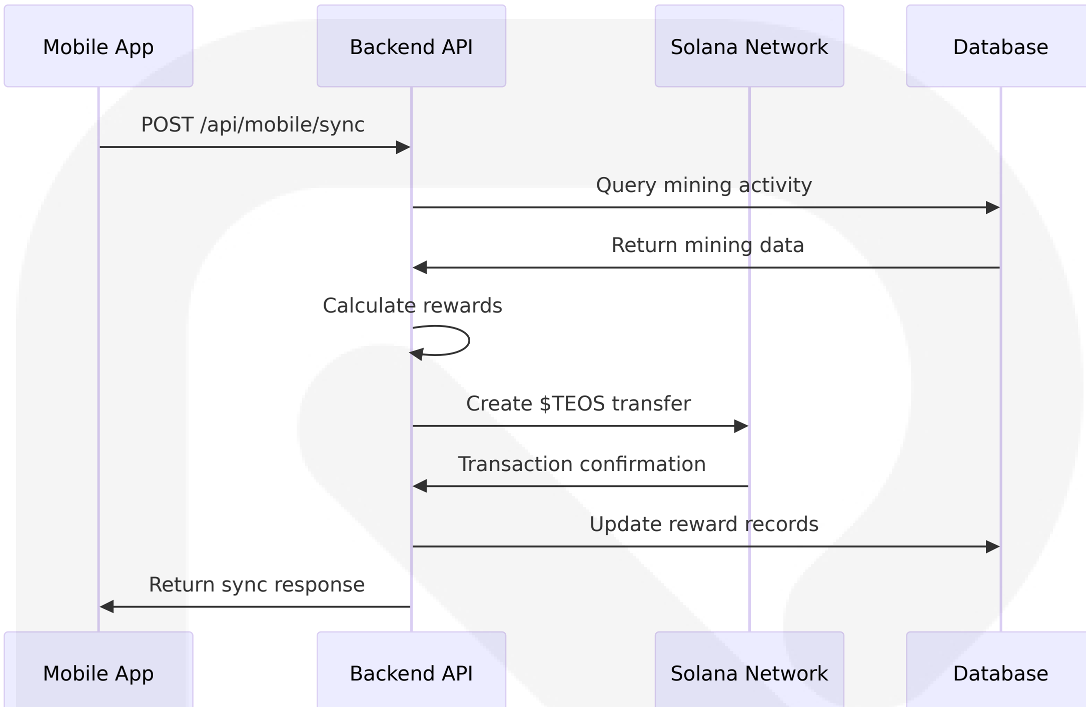
```
/api/
├── /mobile/
│   ├── GET /sync              # Mobile app synchronization
│   ├── POST /rewards          # Reward distribution
│   └── GET /status            # Mining status check
├── /token/
│   ├── POST /create           # Token creation logging
│   ├── GET /history           # Creation history
│   └── GET /stats             # Platform statistics
└── /health/
    ├── GET /                  # Health check endpoint
    └── GET /metrics           # Performance metrics
```

6.2.2 Mobile Mining Integration

Mobile Synchronization Architecture:

The mobile mining integration provides API endpoints for synchronizing with mobile applications and distributing $TEOS rewards. The system implements a queue-based reward distribution mechanism to handle high-volume mobile mining activities.

Mobile Mining Flow:



Reward Distribution Logic:

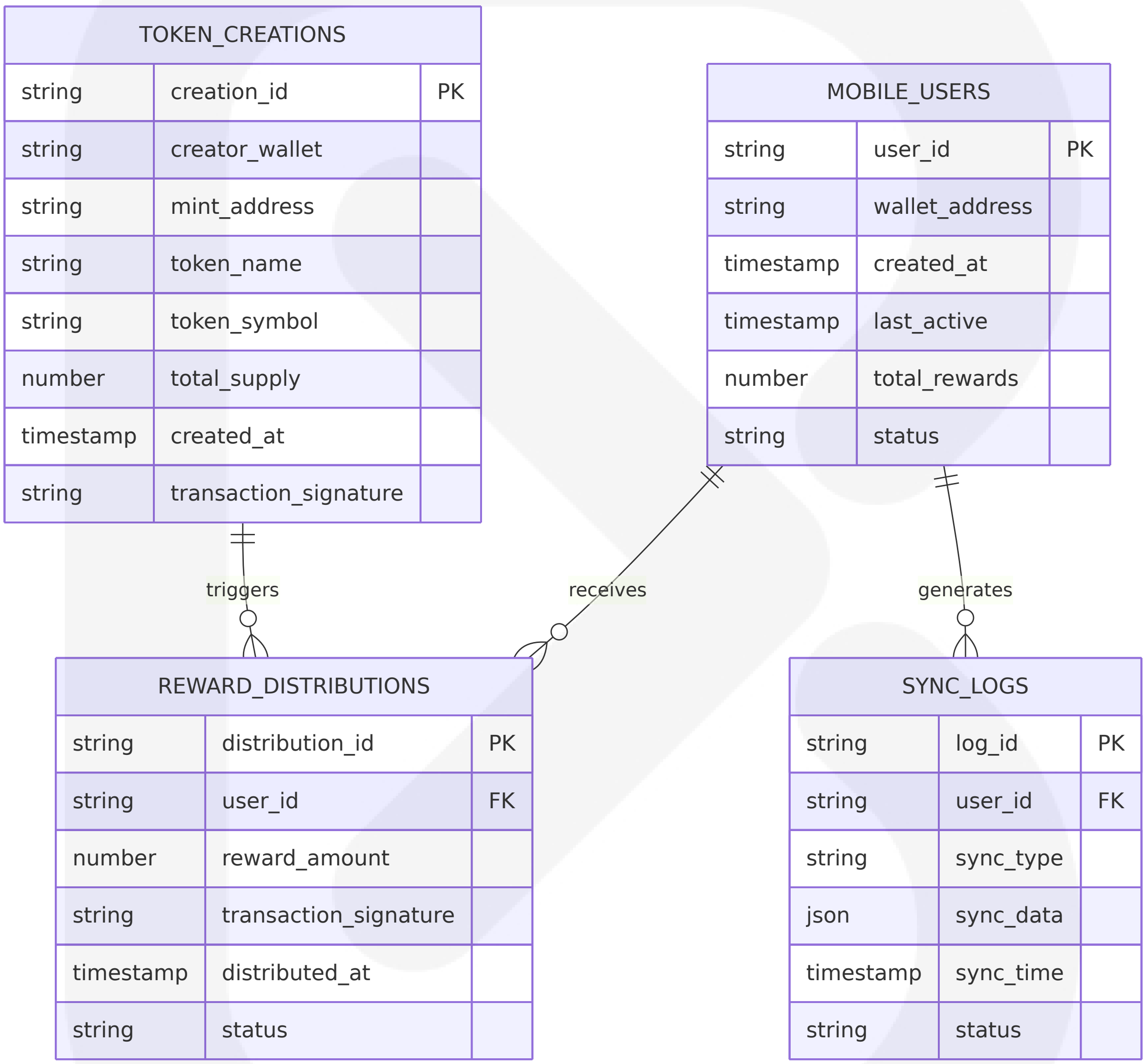
Mining Activity	Reward Calculation	Distribution Method	Verification
Token Creation Participation	Base reward + bonus multiplier	Batch transfer to mobile wallets	Blockchain transaction verification
Platform Engagement	Time-based reward accumulation	Scheduled distribution cycles	Activity timestamp validation
Referral Bonuses	Percentage of referred user rewards	Immediate transfer on qualification	Referral chain verification

6.2.3 Database Design

Data Model Architecture:

The backend implements a hybrid data storage approach with blockchain-first principles for financial data and traditional database storage for operational data.

Database Schema Design:



Data Persistence Strategy:

Data Type	Storage Location	Consistency Model	Backup Strategy
Financial Transactions	Solana Blockchain	Strong consistency	Immutable blockchain records

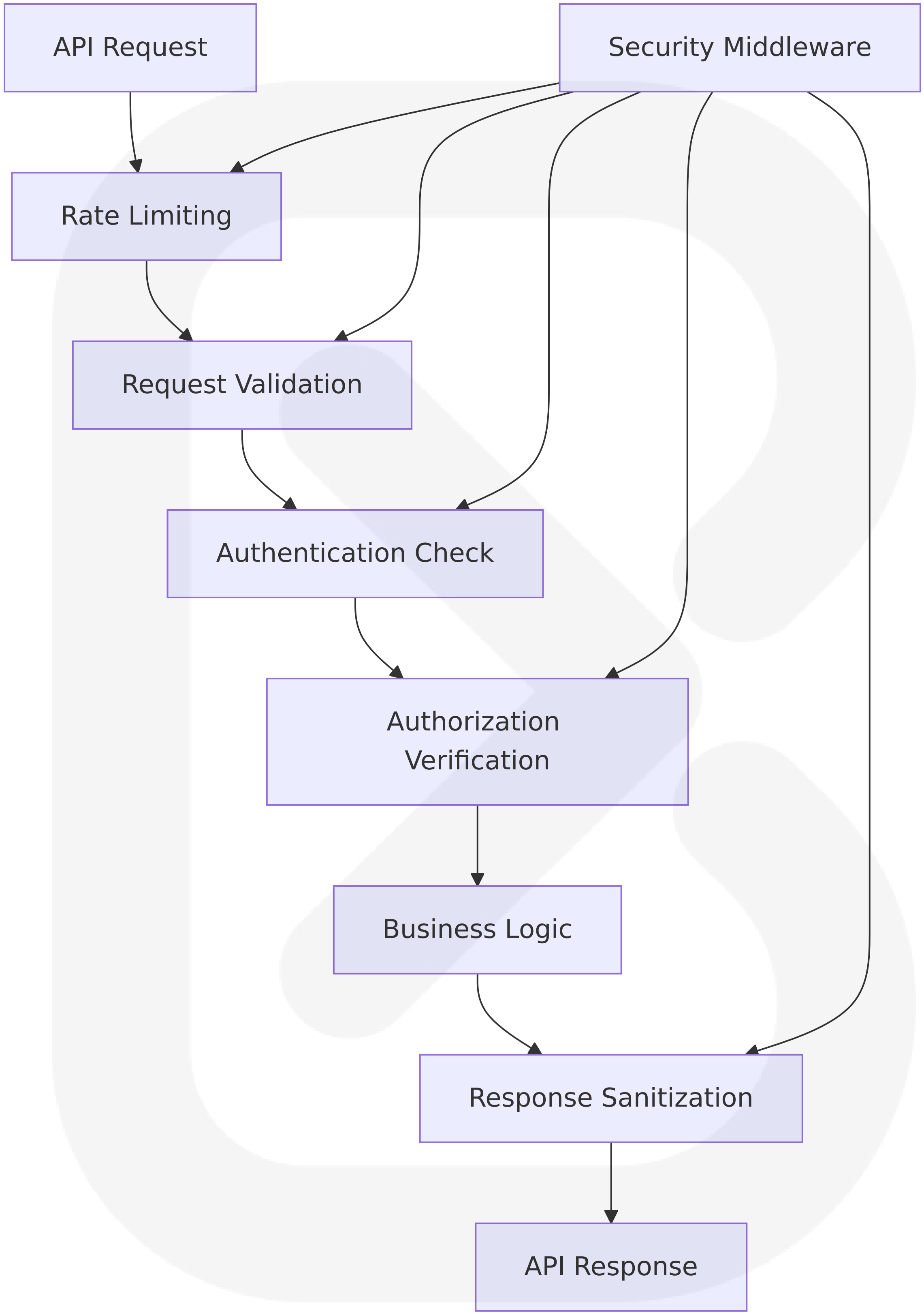
Data Type	Storage Location	Consistency Model	Backup Strategy
User Activity Logs	Backend Database	Eventual consistency	Daily automated backups
Mobile Sync Data	Backend Database	Strong consistency	Real-time replication
Configuration Data	Environment Variables	Immediate consistency	Version-controlled configuration

6.2.4 API Security Implementation

Authentication and Authorization:

The API implements a multi-layer security approach with wallet-based authentication and request validation.

Security Layers:



Security Implementation Details:

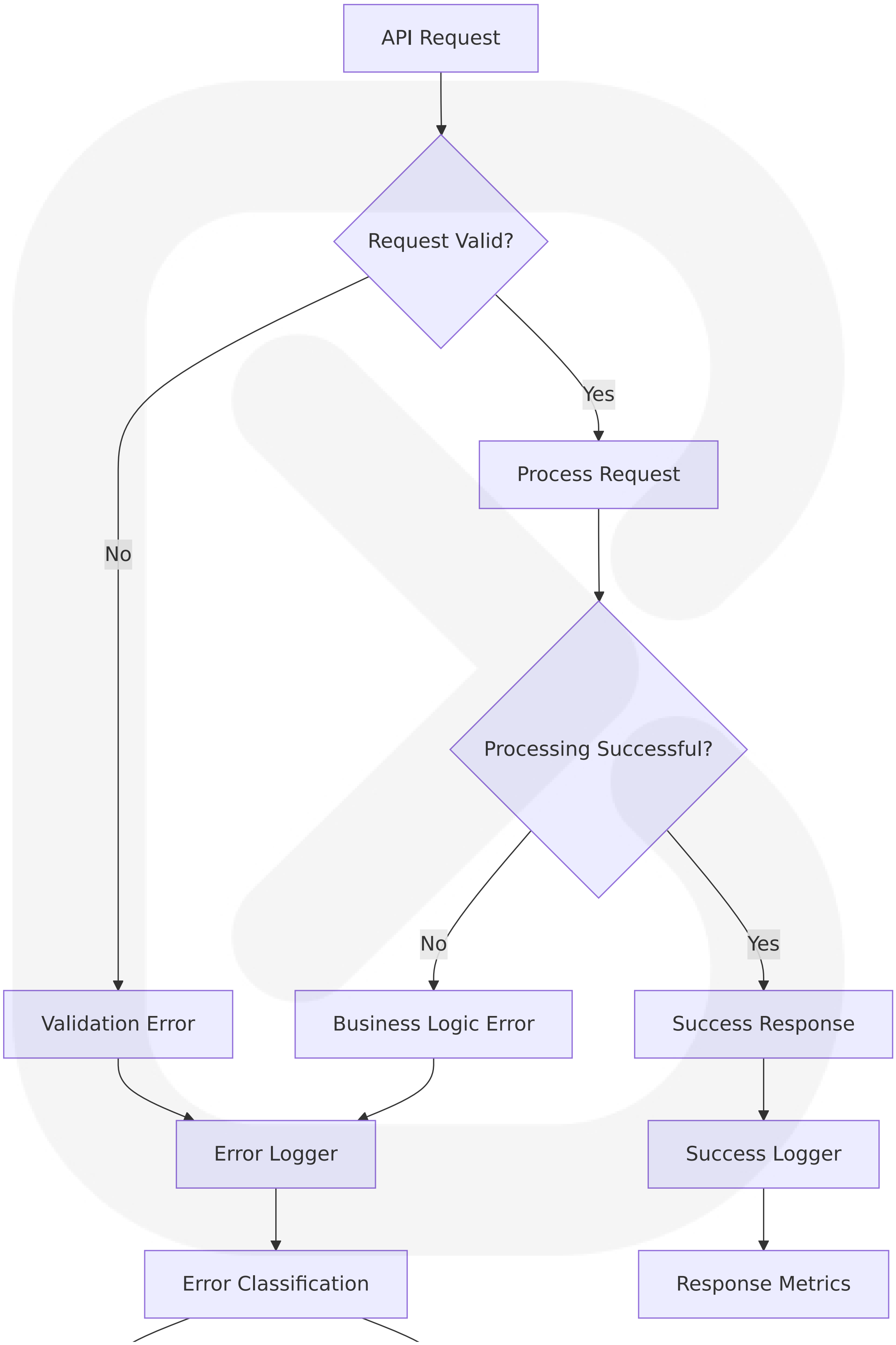
Security Layer	Implementation	Configuration	Monitoring
Rate Limiting	Express rate limiter middleware	100 requests/minute per IP	Request count tracking
Input Validation	Joi schema validation	Strict type checking	Validation error logging
Authentication	JWT token verification	Wallet signature validation	Authentication failure alerts
Authorization	Role-based access control	User permission matrix	Access attempt monitoring

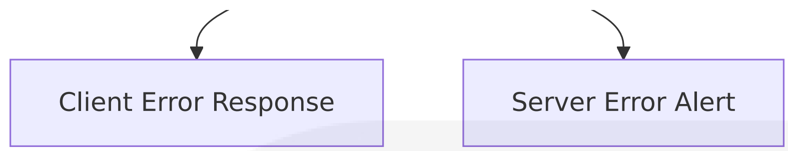
6.2.5 Error Handling and Logging

Comprehensive Error Management:

The backend implements structured error handling with detailed logging for debugging and monitoring purposes.

Error Handling Strategy:


API Request
Request Valid?
Yes
No
Process Request
Processing Successful?
No
Yes
Validation Error
Business Logic Error
Success Response
Error Logger
Success Logger
Error Classification
Response Metrics



Logging Configuration:

Log Level	Use Case	Retention Period	Alert Threshold
ERROR	System failures, security issues	90 days	Immediate alert
WARN	Performance degradation, unusual activity	60 days	5 occurrences/hour
INFO	Normal operations, user actions	30 days	No alerts
DEBUG	Development debugging, detailed traces	7 days	Development only

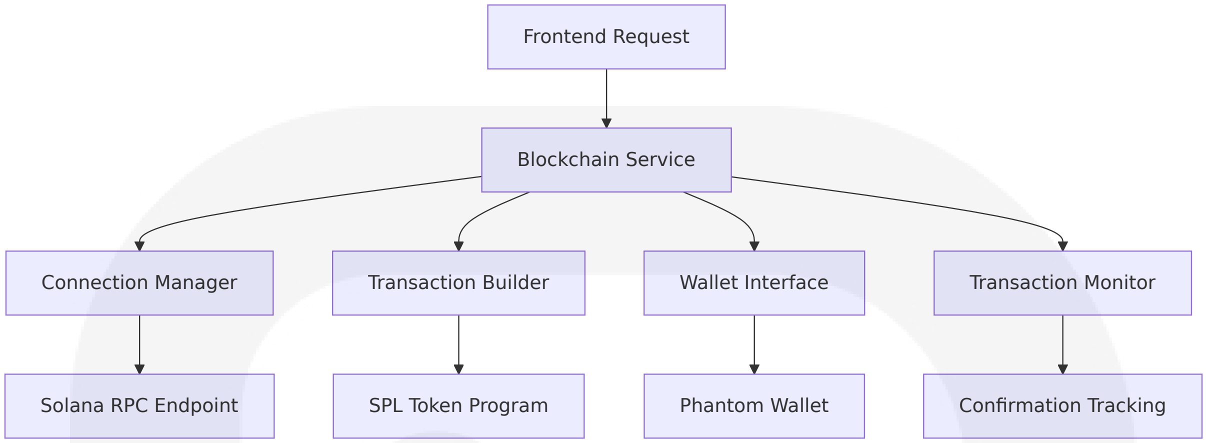
6.3 BLOCKCHAIN INTEGRATION

6.3.1 Solana Network Architecture

SPL Token Integration:

The MintTo instruction on the Token Program creates new tokens. The mint authority must sign the transaction. The instruction mints tokens to a Token Account and increases the total supply on the Mint Account. The blockchain integration layer handles all Solana network interactions including SPL token creation, payment processing, and transaction monitoring.

Blockchain Service Architecture:



Core Blockchain Operations:

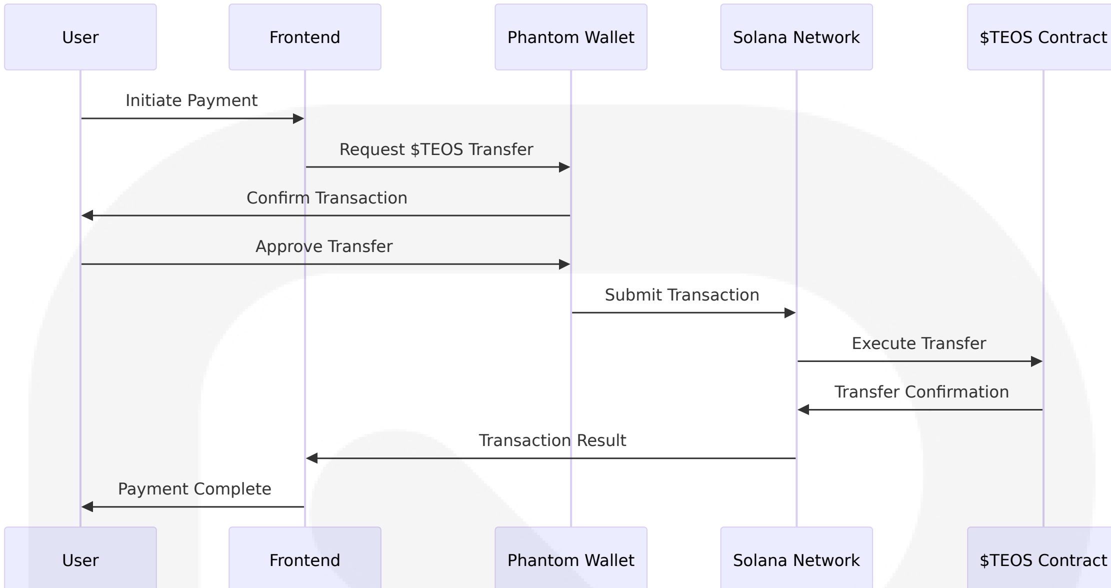
Operation	Implementation	Dependencies	Error Handling
Token Creation	SPL Token Program interaction	@solana/spl-token, mint authority	Retry logic, rollback mechanisms
Payment Processing	SOL and $TEOS transfers	System Program, Token Program	Transaction verification, refund logic
Balance Queries	Account balance retrieval	RPC connection, account parsing	Cache fallback, multiple RPC endpoints
Transaction Monitoring	Confirmation status tracking	WebSocket connections, polling	Timeout handling, status updates

6.3.2 Smart Contract Integration

$TEOS Token Contract Integration:

The platform integrates with the existing $TEOS token contract for payment processing and reward distribution. Contract address: `AhXBUQmbhv9dNoZCiMYmXF4Gyi1cjQthWHFhTL2CJaSo`

Token Contract Interaction Patterns:



Contract Interaction Methods:

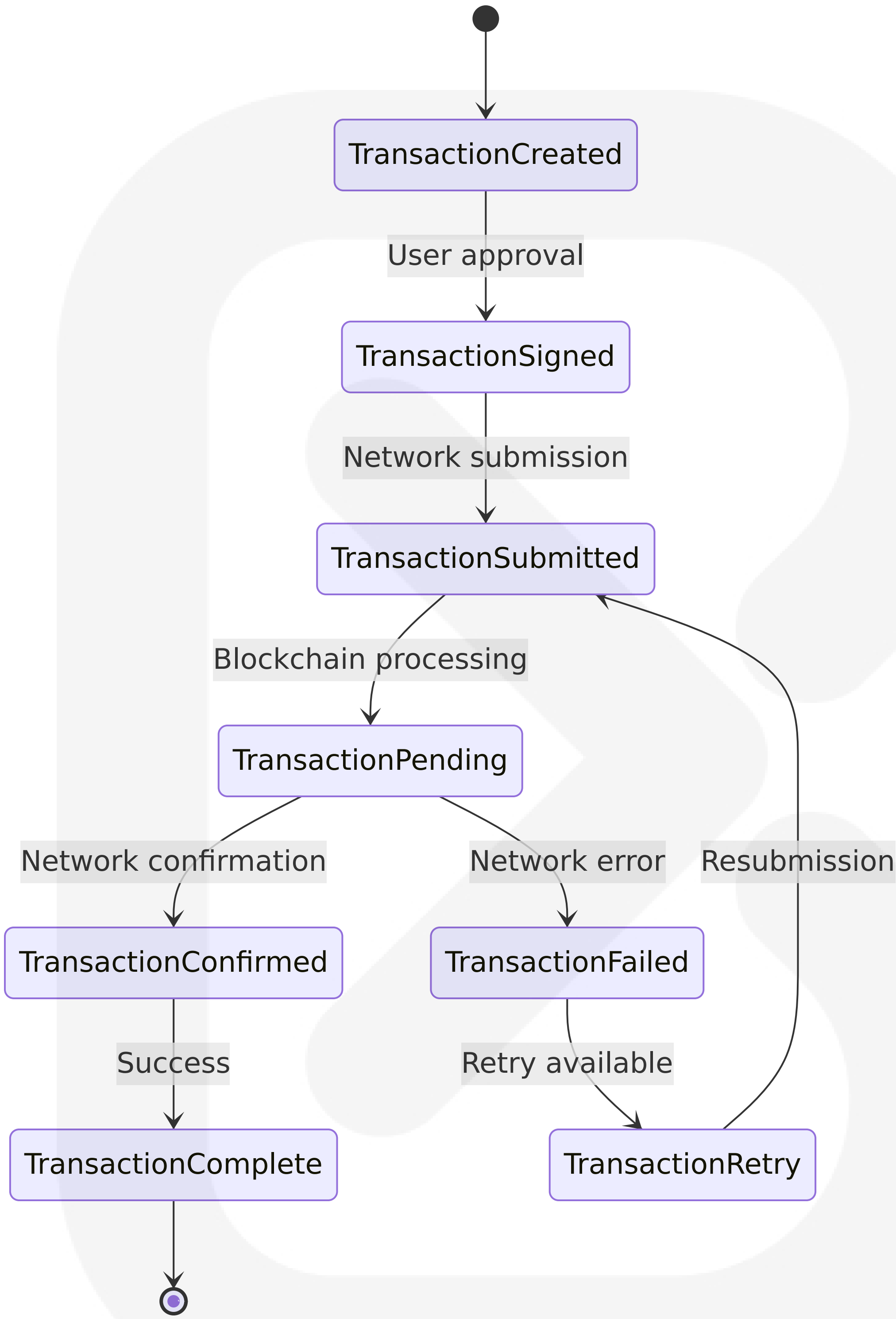
Method	Purpose	Parameters	Return Value
getBalance	Query $TEOS balance	wallet address	token balance
transfer	Send $TEOS tokens	recipient, amount, authority	transaction signature
getTokenInfo	Retrieve token metadata	mint address	token information
createAssociatedAccount	Create token account	owner, mint	account address

6.3.3 Transaction Management

Transaction Lifecycle Management:

Fast Transaction Speeds: Solana's architecture allows for fast transaction speeds, processing thousands of transactions per second. Low Fees: The low fees associated with transactions on Solana make it economically viable for token creation.

Transaction Processing Pipeline:



Transaction Monitoring Implementation:

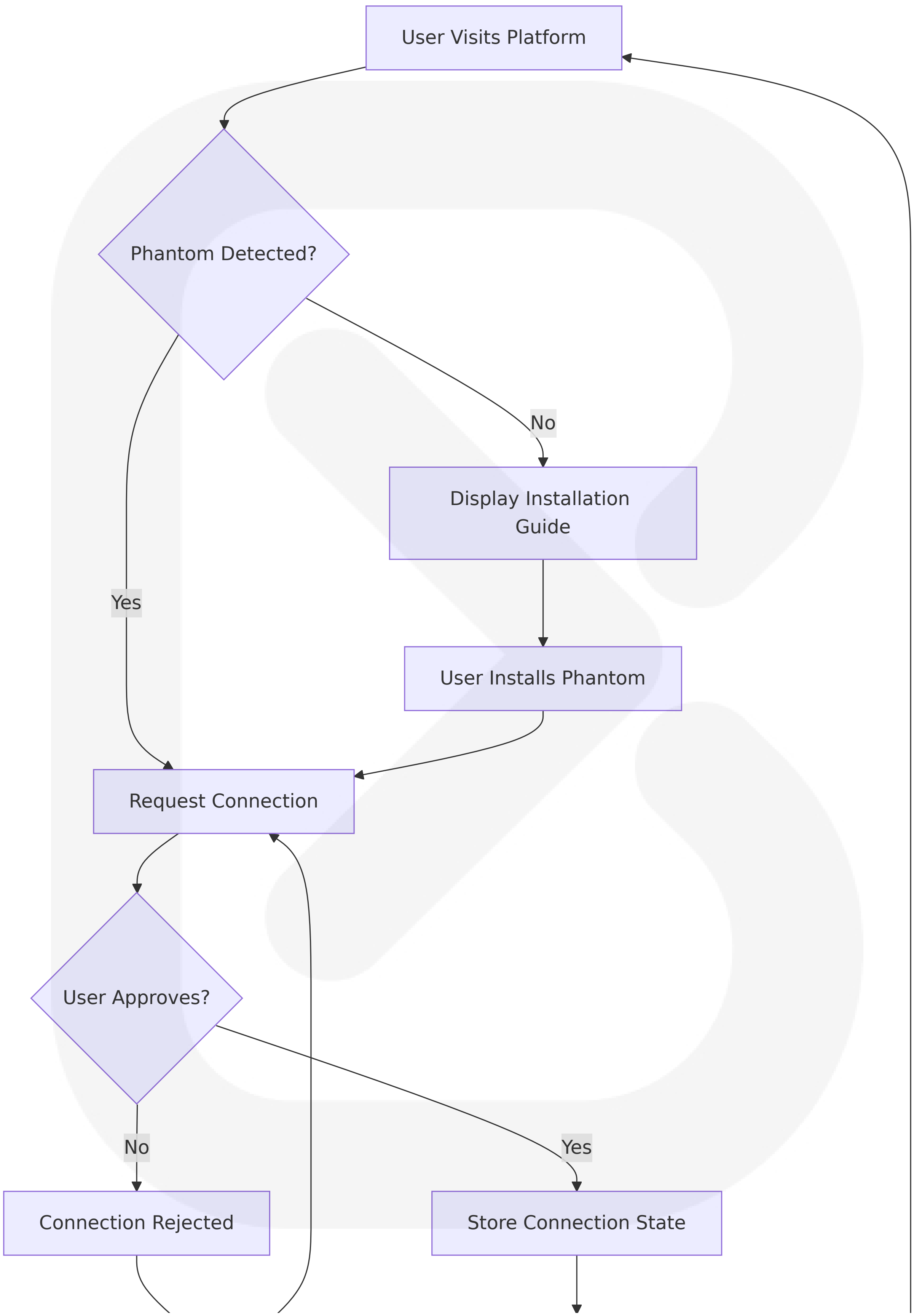
Monitoring Aspect	Implementation	Frequency	Alert Conditions
Confirmation Status	RPC polling with exponential backoff	Every 2 seconds initially	No confirmation after 60 seconds
Network Congestion	Fee estimation and adjustment	Before each transaction	High fee recommendations
Transaction Failures	Error parsing and classification	Real-time	Critical error patterns
Account State Changes	Balance and token account monitoring	After each transaction	Unexpected state changes

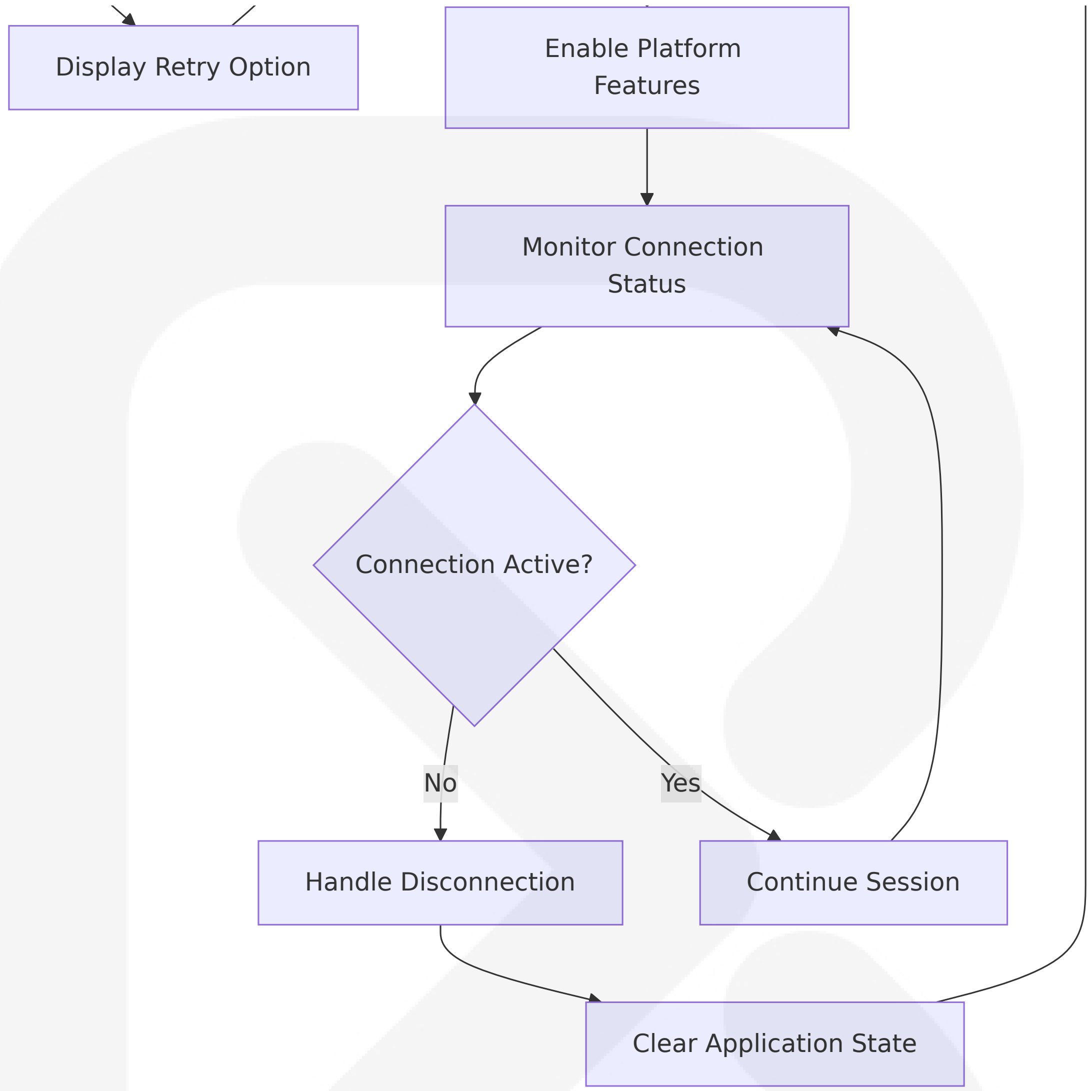
6.3.4 Wallet Integration Patterns

Phantom Wallet Integration:

Phantom Wallet is designed as a non-custodial, multichain Web3 wallet that supports Solana, Ethereum, and Polygon networks. It allows users to manage their cryptocurrencies and NFTs, engage in staking Solana, swap tokens, and access a variety of DeFi applications directly from the wallet.

Wallet Connection Flow:


User Visits Platform
Phantom Detected?
No
Display Installation Guide
User Installs Phantom
Yes
Request Connection
User Approves?
No
Yes
Connection Rejected
Store Connection State



Wallet Security Implementation:

Security Feature	Implementation	User Experience	Technical Details
Non-Custodial Design	Private keys never leave wallet	User maintains full control	Browser extension isolation
Transaction Signing	User approval required for all transactions	Clear transaction details displayed	Cryptographic signature verification
Connection Management	Secure provider injection	Seamless wallet switching	Event-driven connection monitoring

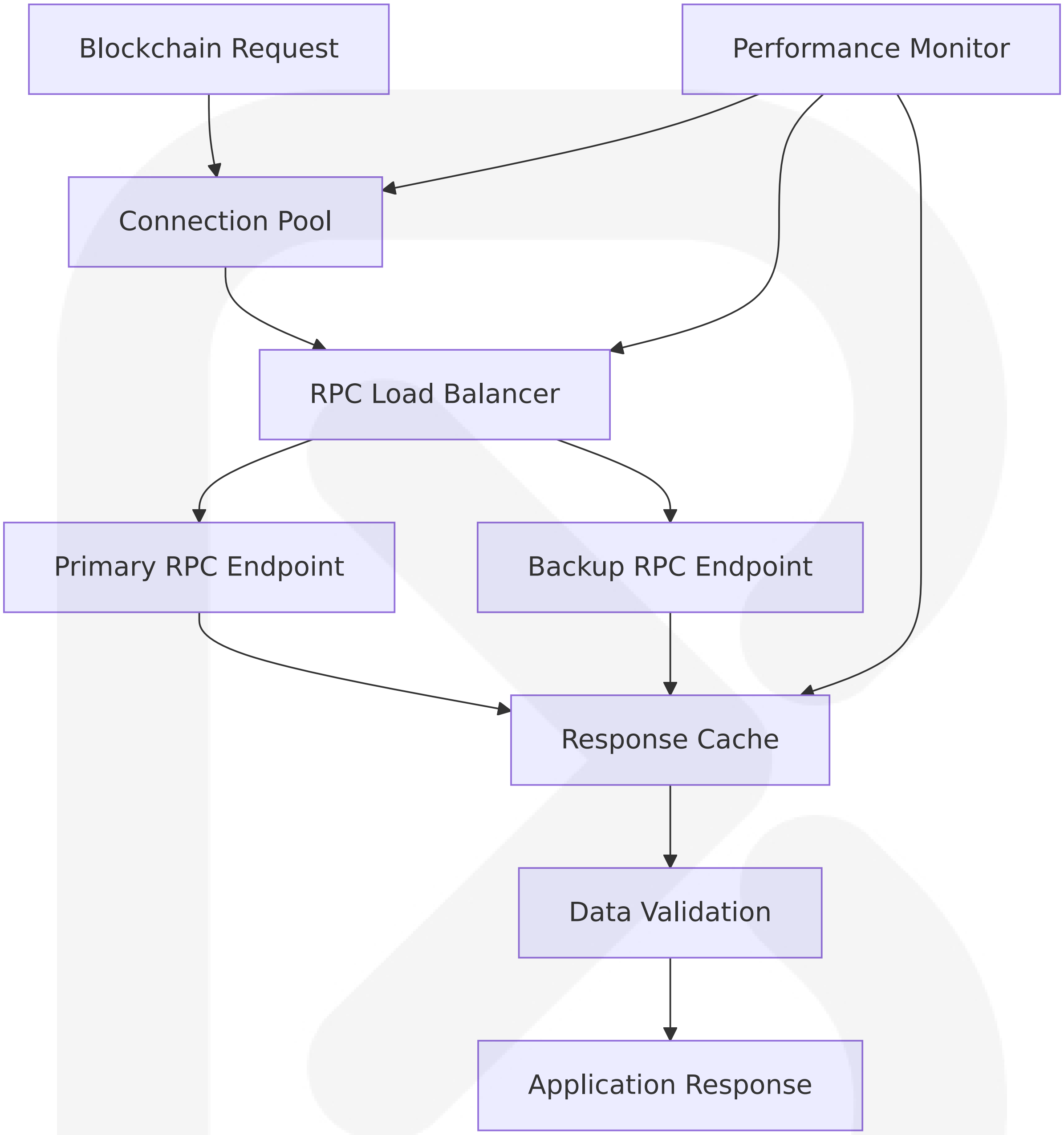
Security Feature	Implementation	User Experience	Technical Details
Error Recovery	Graceful handling of wallet errors	User-friendly error messages	Automatic reconnection attempts

6.3.5 Performance Optimization

Blockchain Interaction Optimization:

The system implements several optimization strategies to ensure optimal performance when interacting with the Solana blockchain.

Optimization Strategies:



Performance Metrics and Targets:

Metric	Target	Monitoring Method	Optimization Strategy
Transaction Confirmation Time	<10 seconds average	Real-time tracking	Priority fee adjustment
RPC Response Time	<500ms average	Endpoint monitoring	Connection pooling

Metric	Target	Monitoring Method	Optimization Strategy
Wallet Connection Time	<3 seconds	User experience tracking	Provider caching
Token Creation Success Rate	>95%	Transaction monitoring	Retry mechanisms

Caching Strategy:

- **Account Data**: Cache balance and token account information for 30 seconds
- **Token Metadata**: Cache token information for 5 minutes
- **Network Status**: Cache network health data for 1 minute
- **Transaction History**: Cache confirmed transactions for 1 hour

This comprehensive system components design provides detailed technical specifications for implementing TeosPump's frontend, backend, and blockchain integration layers, ensuring optimal performance, security, and user experience across all platform components.

Based on my analysis of the TeosPump project requirements and current architectural patterns, I need to determine whether this system requires a traditional microservices architecture or if it follows a different pattern.

6.1 CORE SERVICES ARCHITECTURE

6.1.1 Architecture Assessment

Core Services Architecture is not applicable for this system as a traditional microservices architecture. TeosPump follows a **simplified service-oriented architecture** with distinct but tightly integrated components rather than distributed microservices.

The system architecture consists of:

- **Frontend Service**: Next.js application handling user interface and wallet integration
- **Backend API Service**: Express.js server managing mobile synchronization and logging
- **Blockchain Integration Layer**: Solana network interactions for SPL token operations
- **External Service Dependencies**: Phantom wallet, GitHub, Vercel, and mobile applications

This architecture pattern is optimal for TeosPump because:

Architectural Decision	Justification	Alternative Considered
Monolithic Frontend	Next.js 14+ brings new features and enhancements that enable developers to build scalable, performant, and maintainable applications	Micro-frontends (unnecessary complexity)
Single Backend API	Express 5.0 brought modern features and a future-oriented architecture to the framework	Multiple microservices (over-engineering)
Blockchain-First Data	Solana can power thousands of transactions per second	Traditional database-centric approach

6.1.2 Service Boundaries and Responsibilities

6.1.2.1 Frontend Service Boundary

Primary Responsibilities:

- User interface rendering and interaction management
- Phantom wallet integration and transaction signing
- Token creation form validation and submission
- Real-time blockchain transaction monitoring

Service Interface:

- Browser-based user interactions
- Wallet provider API integration
- Backend API consumption
- Blockchain RPC communication

6.1.2.2 Backend API Service Boundary

Primary Responsibilities:

- Mobile application synchronization endpoints
- Transaction logging and audit trail maintenance
- $TEOS reward distribution to mobile miners
- Platform analytics and monitoring data collection

Service Interface:

- RESTful API endpoints for mobile clients
- Database operations for transaction records
- Integration with mobile mining applications
- Webhook handling for external notifications

6.1.2.3 Blockchain Integration Boundary

Primary Responsibilities:

- SPL token creation and management
- Payment processing for $TEOS and SOL transactions
- Smart contract interaction with existing $TEOS token
- Network state monitoring and error handling

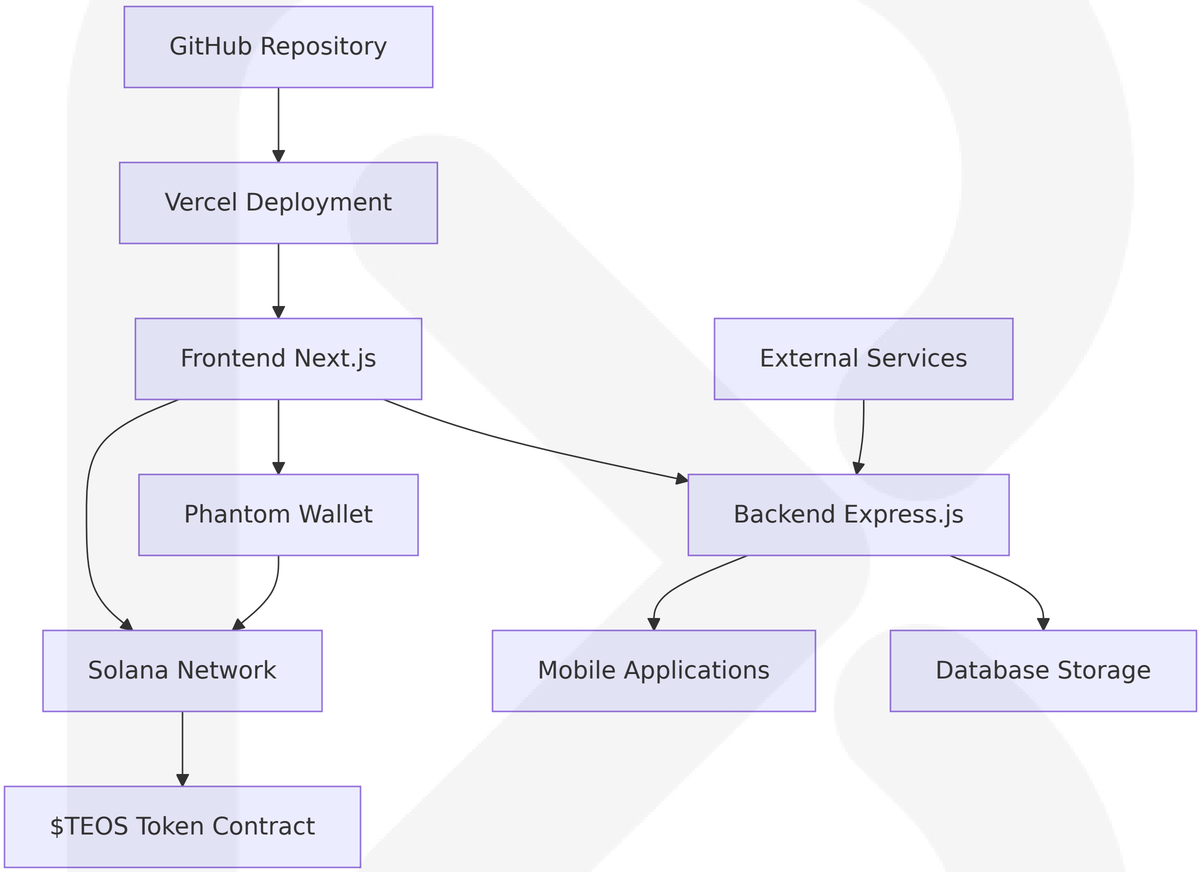
Service Interface:

- Solana RPC endpoint communication
- Transaction construction and submission
- Account balance queries and updates

- Network health monitoring

6.1.3 Inter-Service Communication Patterns

6.1.3.1 Communication Architecture



6.1.3.2 Communication Protocols

Communication Path	Protocol	Pattern	Error Handling
Frontend ↔ Backend	HTTP/REST	Request-Response	Centralized logging with Winston for error tracking
Frontend ↔ Wallet	JavaScript Provider API	Event-Driven	Retry mechanisms with exponential backoff

Communication Path	Protocol	Pattern	Error Handling
Frontend ↔ Blockchain	JSON-RPC over HTTPS	Asynchronous	Transaction fees averaging around $0.000025 per transaction
Backend ↔ Mobile	RESTful HTTP	Synchronous	Circuit breaker pattern implementation

6.1.4 Scalability Design

6.1.4.1 Horizontal Scaling Approach

Frontend Scaling Strategy:
Next.js production grade React applications that scale with the world's leading companies using Next.js by Vercel

- **Edge Distribution**: Vercel's global CDN for static asset delivery
- **Server-Side Rendering**: Dynamic content generation at edge locations
- **Client-Side Caching**: Browser-based state management and local storage

Backend Scaling Strategy:
Node.js microservices are a game-changer for building scalable apps with big players like Uber, Netflix, and Amazon already on board

- **Stateless Design**: No server-side session storage requirements
- **Connection Pooling**: Efficient database connection management
- **Load Balancing**: Multiple Express.js instances behind load balancer

6.1.4.2 Vertical Scaling Considerations

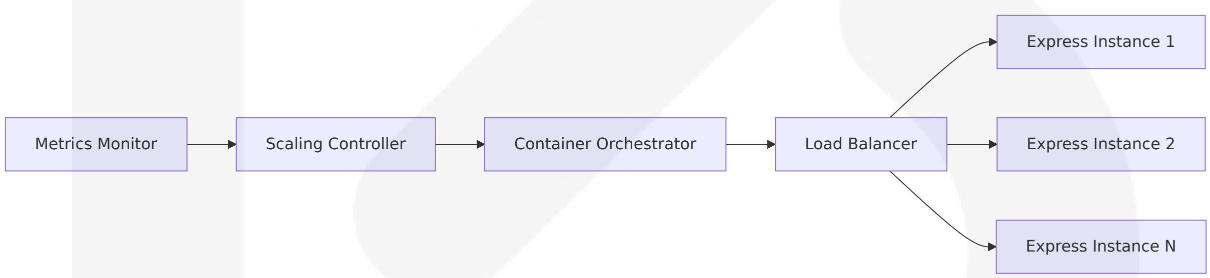
Component	Scaling Trigger	Resource Allocation	Monitoring Metric
Frontend	>1000 concurrent users	CPU and memory optimization	Response time <2 seconds
Backend API	>100 requests/second	Database connection scaling	Response time, CPU usage, memory consumption monitoring
Blockchain Integration	>50 transactions/minute	RPC connection pooling	Solana processes 2,400+ TPS vs Ethereum's <15 TPS

6.1.4.3 Auto-Scaling Implementation

Vercel Auto-Scaling:

- Automatic function scaling based on request volume
- Edge function distribution for global performance
- Built-in CDN scaling for static assets

Backend Auto-Scaling:



6.1.5 Performance Optimization Techniques

6.1.5.1 Frontend Optimization

Next.js Performance Features:
Fetch data and render in the same environment, reducing back-and-forth communication and performing multiple data fetches with single round-trip

- **Static Generation**: Pre-built pages for optimal loading
- **Code Splitting**: Dynamic imports for reduced bundle size
- **Image Optimization**: Automatic WebP conversion and lazy loading

6.1.5.2 Backend Optimization

Express.js Performance Enhancements:
Performance monitoring and systematic bottleneck identification with deeper optimizations expected by mid-2026 for faster, more scalable framework

- **Asynchronous Processing**: Event-driven model with async/await for cleaner asynchronous code
- **Caching Strategies**: In-memory caching and distributed caching to reduce database load
- **Connection Optimization**: Database connection pooling and query optimization

6.1.5.3 Blockchain Performance

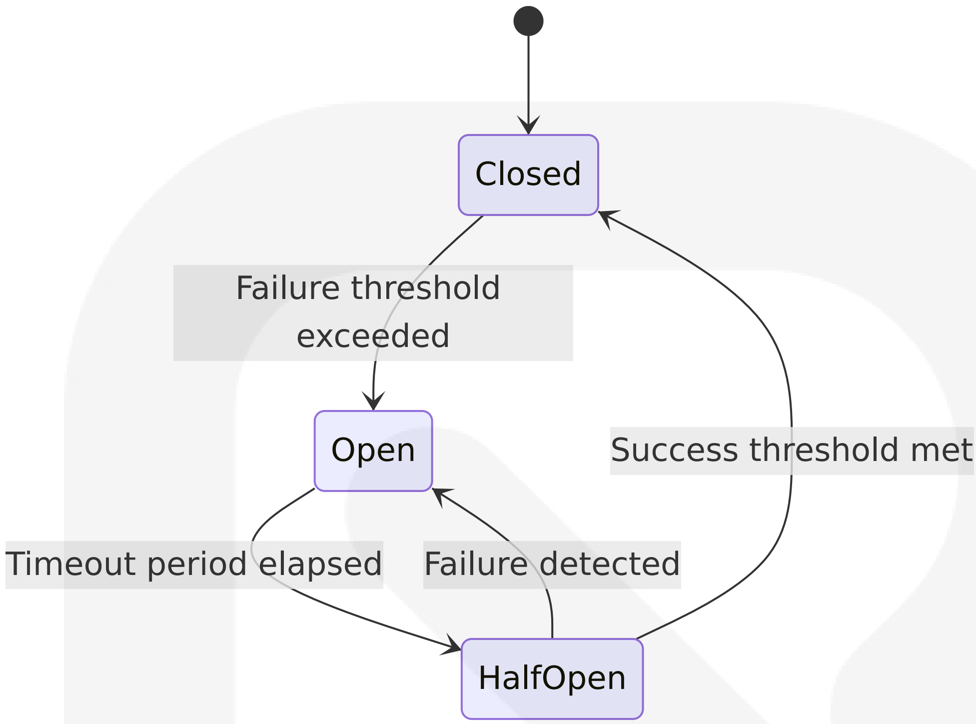
Solana Network Advantages:
Solana processes up to 65,000 TPS through parallel processing and generates new blocks every 400 milliseconds

- **Parallel Transaction Processing**: Multiple simultaneous operations
- **Low Transaction Costs**: Minimal network fees for operations
- **Fast Confirmation Times**: Sub-second transaction finality

6.1.6 Resilience Patterns

6.1.6.1 Fault Tolerance Mechanisms

Circuit Breaker Implementation:
Circuit breakers monitor downstream service health and prevent cascading failures by directing traffic away from failing services



Retry and Fallback Mechanisms:
Fault tolerance patterns including retry logic, exponential backoff, and circuit breakers based on error rates and latency thresholds

Failure Scenario	Retry Strategy	Fallback Mechanism	Recovery Time
Wallet Connection	3 attempts with backoff	Manual reconnection prompt	<5 seconds
Blockchain RPC	Exponential backoff	Alternative RPC endpoints	<10 seconds
Backend API	Linear retry	Cached responses	<2 seconds
Mobile Sync	Circuit breaker	Queued operations	<30 seconds

6.1.6.2 Graceful Degradation Policies

Service Degradation Hierarchy:
Graceful degradation with fallback mechanisms and alternative workflows to maintain basic functionality during service disruptions

1. **Core Functionality**: Token creation and payment processing (highest priority)
2. **Enhanced Features**: Mobile mining synchronization (medium priority)
3. **Analytics**: Platform statistics and monitoring (lowest priority)

6.1.6.3 Data Redundancy Approach

Blockchain-First Data Strategy:

- **Primary Storage**: Solana blockchain for all financial transactions
- **Secondary Storage**: Backend database for operational data
- **Backup Strategy**: Immutable blockchain records with local caching

Recovery Procedures:

- **Blockchain Data**: Self-healing through network consensus
- **Application State**: Stateless design eliminates state recovery needs
- **Configuration Data**: Version-controlled environment variables

6.1.7 Capacity Planning Guidelines

6.1.7.1 Traffic Projections

Expected Load Patterns:

- **Initial Launch**: 100-500 concurrent users
- **Growth Phase**: 1,000-5,000 concurrent users
- **Mature Platform**: 10,000+ concurrent users

Transaction Volume Estimates:

- **Token Creations**: 10-100 per hour during peak times
- **Payment Transactions**: 50-500 per hour
- **Mobile Sync Operations**: 1,000-10,000 per hour

6.1.7.2 Resource Allocation Strategy

Resource Type	Current Allocation	Scaling Threshold	Maximum Capacity
Frontend Instances	Vercel serverless	Auto-scaling	Unlimited
Backend Instances	1-2 containers	>80% CPU utilization	10+ containers
Database Connections	10-20 connections	>70% utilization	100+ connections
Blockchain RPC	2-3 endpoints	>5 second response	10+ endpoints

6.1.7.3 Performance Monitoring

Key Performance Indicators:

- **Response Time**: <2 seconds for frontend, <500ms for API
- **Throughput**: 100+ API requests/second, 50+ blockchain transactions/minute
- **Availability**: 99.9% uptime for frontend, 99.5% for backend
- **Error Rate**: <1% for all operations

This simplified service architecture provides the scalability, performance, and resilience required for TeosPump while avoiding the complexity overhead of traditional microservices patterns. The design leverages modern platform capabilities and blockchain-native patterns to achieve enterprise-grade reliability with minimal operational complexity.

6.2 DATABASE DESIGN

6.2.1 Database Design Assessment

Database Design is not applicable to this system as a traditional centralized database architecture. TeosPump follows a **blockchain-first data architecture** where the Solana blockchain serves as the primary immutable data layer, with minimal traditional database requirements for operational support.

The system architecture prioritizes:

- **Blockchain-Native Data Storage**: All token operations are recorded on the Solana blockchain, allowing users to verify transactions via blockchain explorers like Solscan
- **Decentralized Data Persistence**: Tokens on Solana are uniquely identified by the address of a Mint Account owned by the Token Program
- **Immutable Transaction Records**: This program defines a common implementation for Fungible and Non Fungible tokens with cryptographic security
- **Non-Custodial Architecture**: User data remains under user control through wallet-based authentication

6.2.2 Data Storage Strategy Analysis

6.2.2.1 Blockchain-First Data Architecture

Primary Data Layer: Solana Blockchain

The system leverages Solana's high-performance blockchain as the authoritative data source for all financial and token-related operations. Solana is one of the fastest blockchains, capable of processing over 65,000 transactions per second. Operations involving SPL tokens (e.g., transfers, burns, or mints) are executed almost instantly.

Data Type	Storage Location	Immutability	Access Pattern
Token Metadata	Solana Mint Accounts	Immutable	Direct blockchain queries
Payment Records	Blockchain Transactions	Immutable	Transaction signature lookup

Data Type	Storage Location	Immutability	Access Pattern
User Balances	Token Accounts	Mutable via transactions	Real-time RPC queries
Creation History	Transaction Logs	Immutable	Historical blockchain analysis

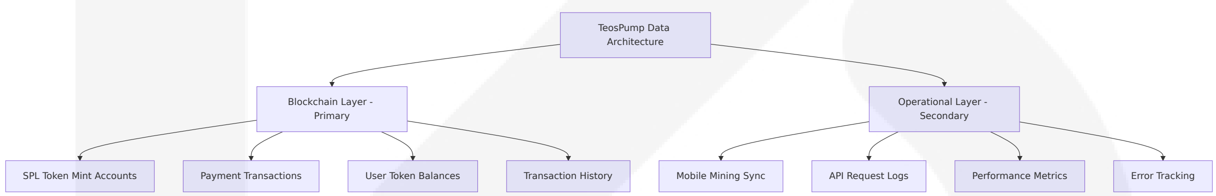
Cost and Performance Characteristics:

Solana's transaction fees are extremely low, typically costing just a fraction of a cent, making SPL tokens highly economical to use. This economic efficiency enables the platform to store all critical data on-chain without prohibitive costs.

6.2.2.2 Minimal Traditional Database Requirements

Limited Operational Database Needs

The system requires minimal traditional database storage for specific operational functions that don't require blockchain immutability:



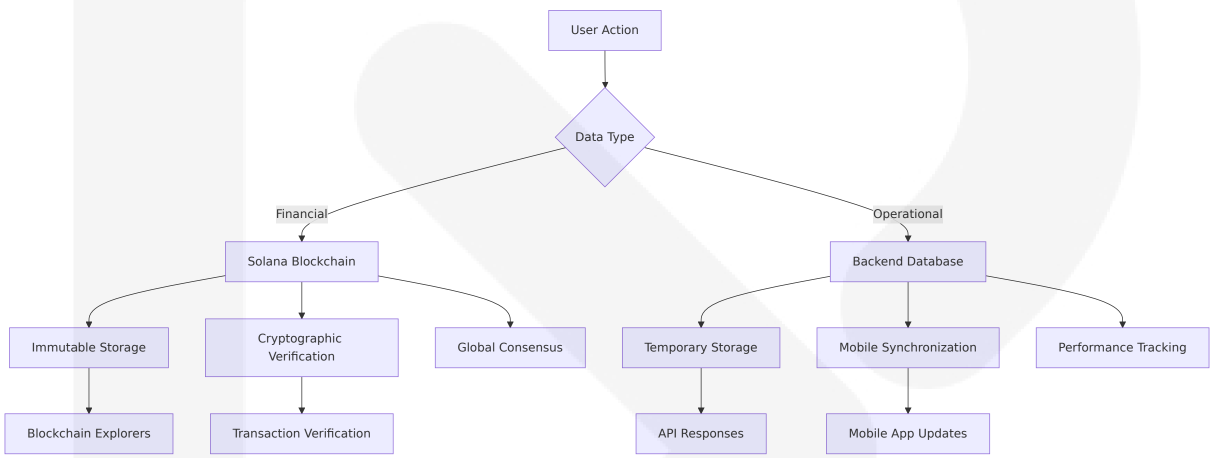
Operational Data Requirements:

Data Category	Purpose	Persistence Need	Backup Strategy
Mobile Mining Records	Reward distribution tracking	Temporary (30 days)	Blockchain verification
API Synchronization Logs	Mobile app coordination	Session-based	Log rotation
Performance Metrics	System monitoring	Analytics only	Time-series aggregation

Data Category	Purpose	Persistence Need	Backup Strategy
Error Tracking	Debugging and monitoring	Development support	Centralized logging

6.2.2.3 Hybrid Data Management Approach

Data Flow Architecture:



Data Consistency Model:

Consistency Type	Implementation	Use Case	Recovery Method
Strong Consistency	Blockchain consensus	Financial transactions	Network-wide validation
Eventual Consistency	Database replication	Mobile synchronization	Blockchain reconciliation
Session Consistency	In-memory state	User interface updates	Wallet reconnection

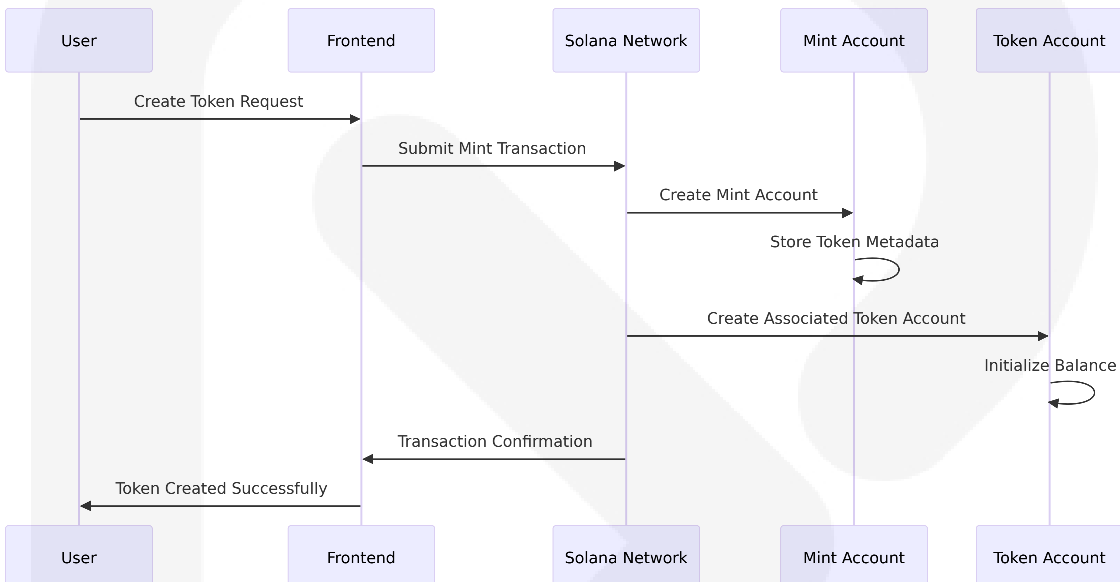
6.2.3 Alternative Data Persistence Patterns

6.2.3.1 Blockchain-Native Patterns

SPL Token Standard Implementation:

The Solana Primary Library (SPL) defines how smart contract tokens on the Solana blockchain operate. The operational standards are delineated in the library and must be adhered to by any token created on the Solana blockchain. These tokens are known as SPL tokens.

Token Creation Data Flow:



6.2.3.2 Caching and Performance Optimization

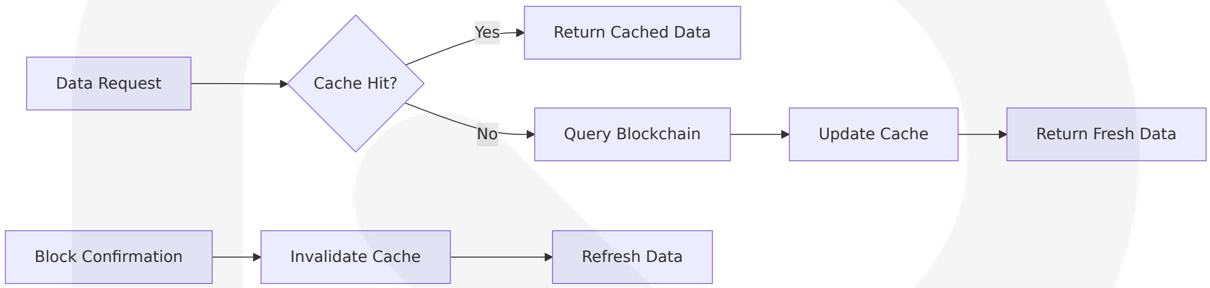
Multi-Layer Caching Strategy:

Since traditional database queries are minimal, the system implements blockchain-specific caching patterns:

Cache Layer	Data Cached	TTL	Invalidation Strategy
Browser Cache	Wallet connection state	Session	User disconnection
Application Cache	Token metadata	5 minutes	Block confirmation
RPC Cache	Account balances	30 seconds	Transaction detection

Cache Layer	Data Cached	TTL	Invalidation Strategy
CDN Cache	Static assets	24 hours	Deployment updates

Blockchain Query Optimization:



6.2.4 Data Compliance and Security

6.2.4.1 Privacy-First Architecture

Non-Custodial Data Principles:

The system follows privacy-by-design principles where user data remains under user control:

- **No Personal Data Storage**: User identification through wallet addresses only
- **Blockchain Transparency**: All financial data publicly verifiable but pseudonymous
- **Minimal Data Collection**: Only operational data necessary for platform function
- **User-Controlled Access**: Private keys never leave user's wallet

6.2.4.2 Regulatory Compliance Approach

Compliance Framework:

Compliance Area	Implementation	Blockchain Benefit	Traditional Database Risk
Data Retention	Immutable blockchain records	Permanent audit trail	Complex retention policies
Data Portability	Public blockchain access	Universal data access	Vendor lock-in concerns
Right to Erasure	Pseudonymous addresses	Privacy through anonymity	Complex deletion procedures
Audit Requirements	Cryptographic verification	Tamper-proof records	Database integrity challenges

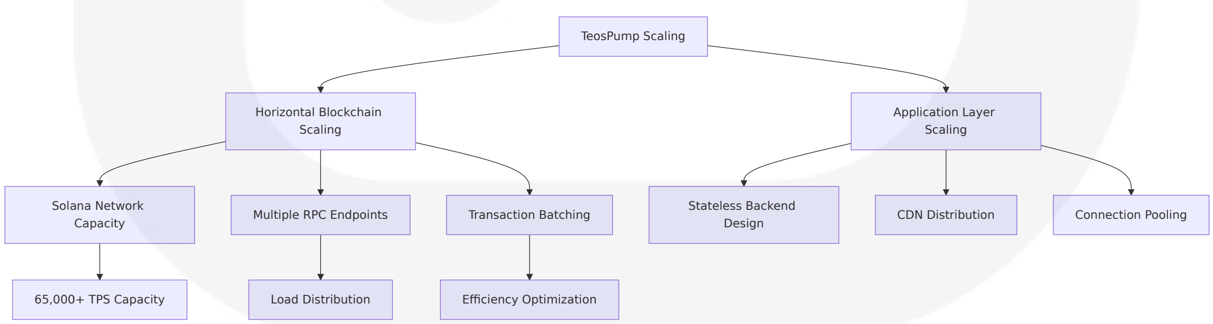
6.2.5 Scalability and Performance Considerations

6.2.5.1 Blockchain Scalability Advantages

Solana Performance Characteristics:

Transaction Speed: Solana can handle thousands of transactions per second, significantly outperforming Ethereum, which typically manages around 15-45 transactions per second. SPL tokens benefit from this high throughput, making it ideal for applications requiring fast, frequent token transfers.

Scaling Strategy:



6.2.5.2 Cost-Effectiveness Analysis

Economic Efficiency:

Gas fees on Ethereum can surge during network congestion, making transactions prohibitively expensive. Solana's architecture allows it to maintain low transaction costs, typically under $0.01.

Cost Comparison:

Operation	Traditional Database	Solana Blockchain	Cost Advantage
Token Creation	$0.10-1.00 (server costs)	$0.000025 (network fee)	4000x cheaper
Transaction Recording	$0.01-0.05 (database ops)	$0.000025 (network fee)	400-2000x cheaper
Data Querying	$0.001-0.01 (compute)	Free (RPC queries)	100% savings
Backup/Replication	$10-100/month (infrastructure)	$0 (network consensus)	100% savings

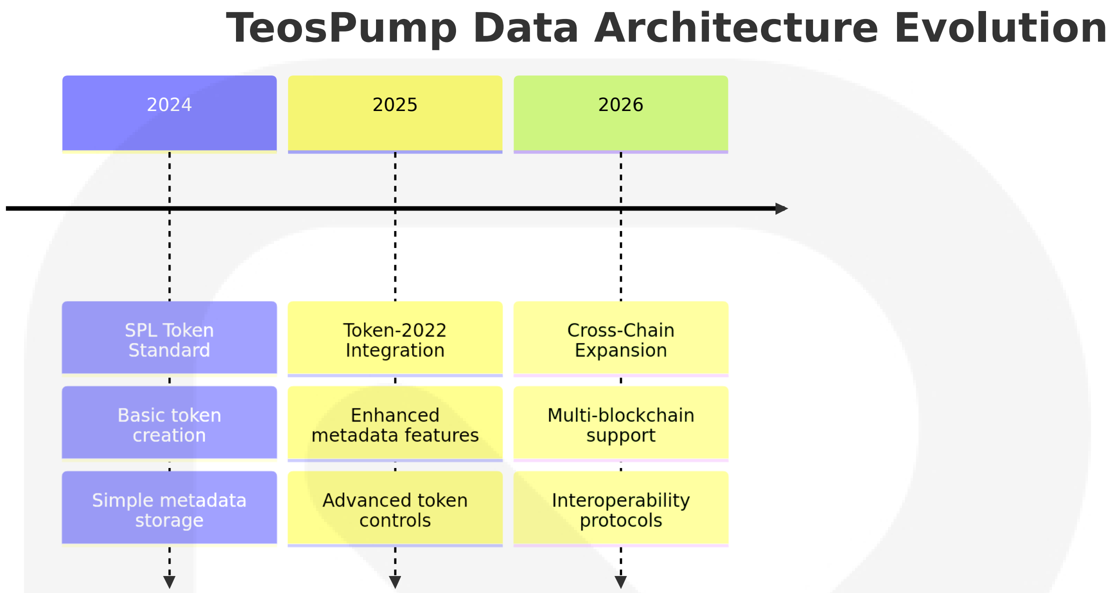
6.2.6 Future-Proofing and Evolution

6.2.6.1 Blockchain Technology Evolution

Token-2022 Program Integration:

According to official information, the Token-2022 program is still in development and not meant for full production use until a stable release. Pending its full release, the newly-introduced functions of the Token-2022 program could be a significant upgrade for the token system on the Solana blockchain.

Evolution Pathway:



6.2.6.2 Architectural Flexibility

Modular Design Benefits:

The blockchain-first approach provides inherent flexibility for future enhancements:

- **Protocol Upgrades**: Automatic benefit from Solana network improvements
- **Feature Extensions**: New SPL token capabilities without database migrations
- **Cross-Chain Integration**: Potential expansion to other blockchain networks
- **Decentralized Evolution**: Community-driven development possibilities

6.2.7 Conclusion

TeosPump's blockchain-first architecture eliminates the complexity, costs, and risks associated with traditional database systems while providing superior security, transparency, and scalability. The design patterns in blockchain technology are foundational tools that address unique challenges in this emerging field. They offer structured approaches to building robust, scalable, and secure blockchain applications.

This approach represents a paradigm shift from database-centric to blockchain-native application design, leveraging Solana's high-performance capabilities to create a more efficient, secure, and cost-effective platform for token creation and management. The minimal operational database requirements focus solely on enhancing user experience through mobile synchronization and performance monitoring, while all critical data remains secured and verified through blockchain consensus mechanisms.

6.3 INTEGRATION ARCHITECTURE

6.3.1 Integration Architecture Overview

TeosPump implements a **hybrid integration architecture** that combines blockchain-native patterns with traditional web service integrations. The system leverages Solana web3.js latest version 1.98.2 with 4056 other projects using the library for blockchain interactions while maintaining RESTful API patterns for mobile synchronization and external service communication.

The integration strategy prioritizes **decentralized-first design** where critical financial operations occur on-chain through Solana's SPL token program, while operational data flows through traditional API endpoints. Phantom is a crypto wallet that can be used to manage digital assets and access decentralized applications on Solana, Bitcoin, Ethereum, Base, and Polygon, serving as the primary authentication and transaction signing interface.

Core Integration Principles:

Principle	Implementation	Benefit	Technical Approach
Blockchain-First	Financial data on Solana network	Immutable transaction records	SPL token program integration
Non-Custodial Security	User-controlled private keys	Enhanced security model	Phantom wallet provider API
Stateless Services	No server-side session storage	Horizontal scalability	JWT-based authentication
Event-Driven Communication	Asynchronous blockchain interactions	Improved responsiveness	Transaction monitoring patterns

6.3.2 API DESIGN

6.3.2.1 Protocol Specifications

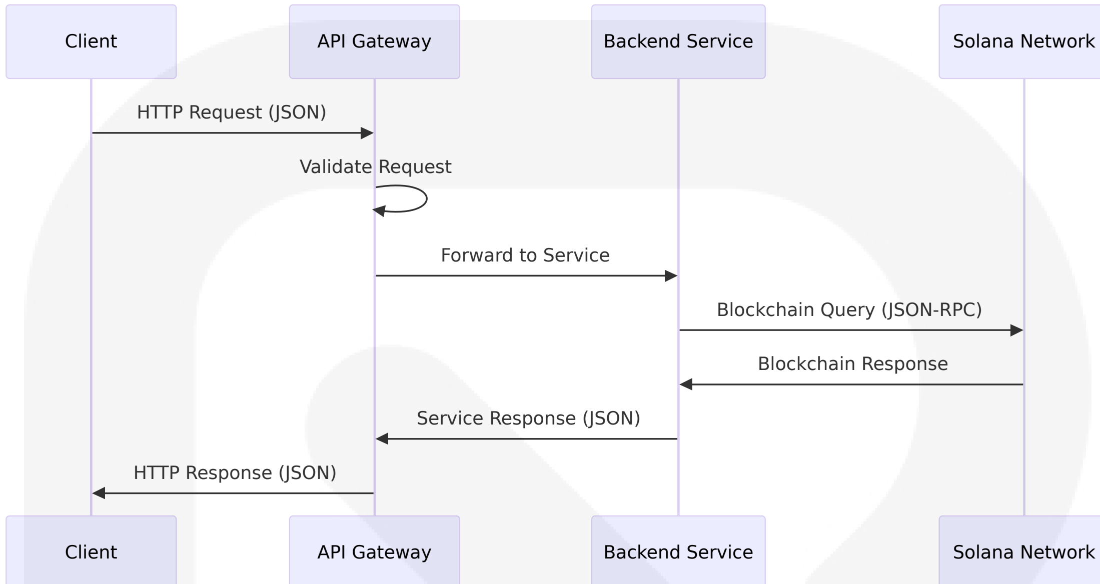
RESTful API Architecture

The backend API follows RESTful design principles with Express 5.0 bringing modern features and a future-oriented architecture to the framework for mobile synchronization and operational data management.

API Endpoint Structure:

Endpoint Category	Base Path	Purpose	Protocol
Mobile Synchronization	`/api/mobile`	Mobile app integration	HTTP/REST
Token Operations	`/api/token`	Token creation logging	HTTP/REST
Health Monitoring	`/api/health`	System status checks	HTTP/REST
Blockchain Proxy	`/api/blockchain`	RPC endpoint abstraction	HTTP/REST → JSON-RPC

Request/Response Format Standards:

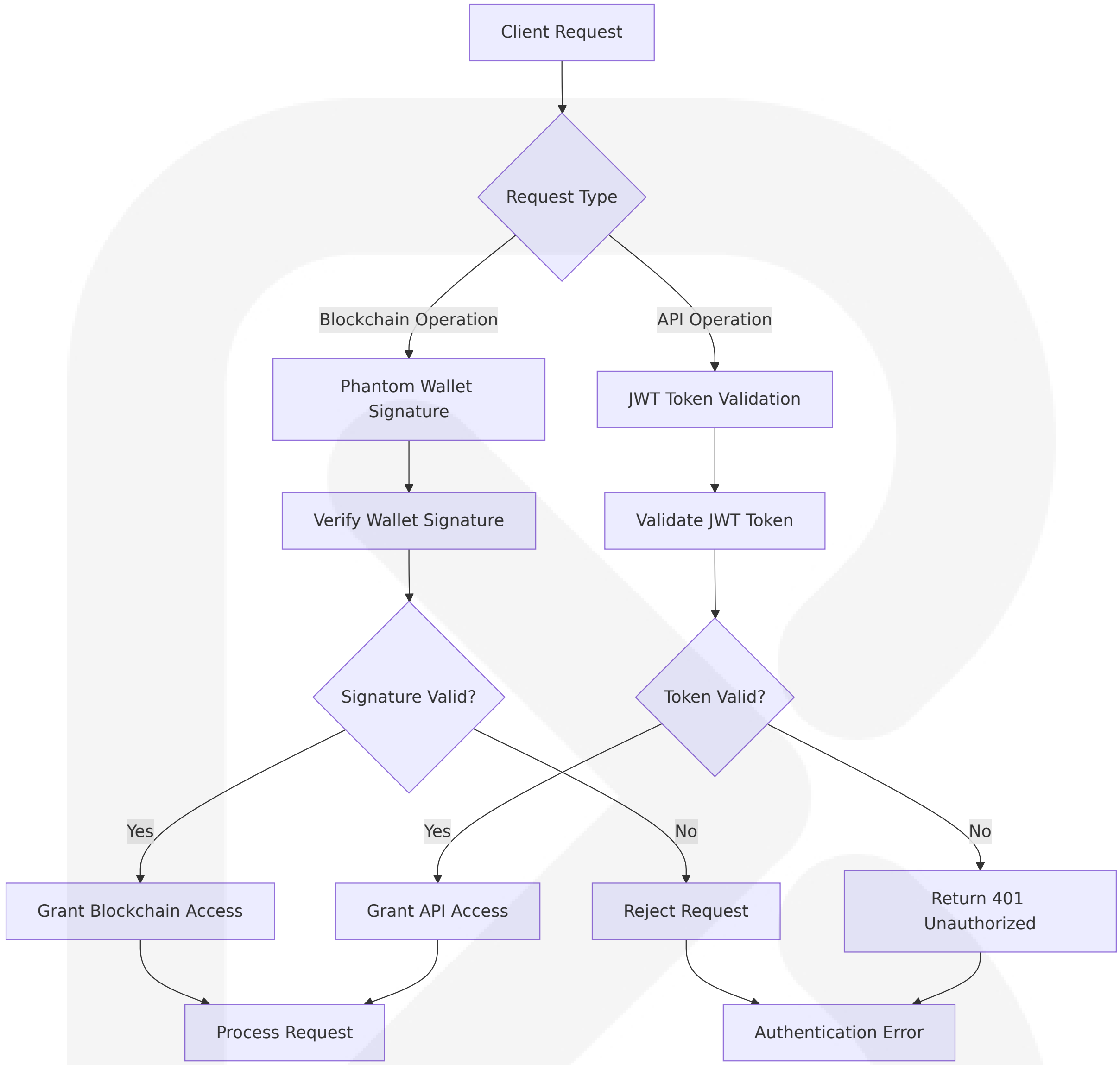


6.3.2.2 Authentication Methods

Multi-Layer Authentication Strategy

The system implements a hybrid authentication approach combining wallet-based cryptographic signatures with traditional API tokens for different integration contexts.

Authentication Flow Architecture:



Authentication Method Specifications:

Method	Use Case	Implementation	Security Level
Wallet Signature	Blockchain transactions	Phantom creates and manages private keys on behalf of users for storing funds and signing transactions	Cryptographic
JWT Tokens	API access	Bearer token in Authorization header	Session-based
API Keys	Mobile app integration	Custom header authentication	Application-level

6.3.2.3 Authorization Framework

Role-Based Access Control (RBAC)

The authorization framework implements a simplified RBAC model tailored for decentralized applications with minimal user data collection.

Authorization Levels:

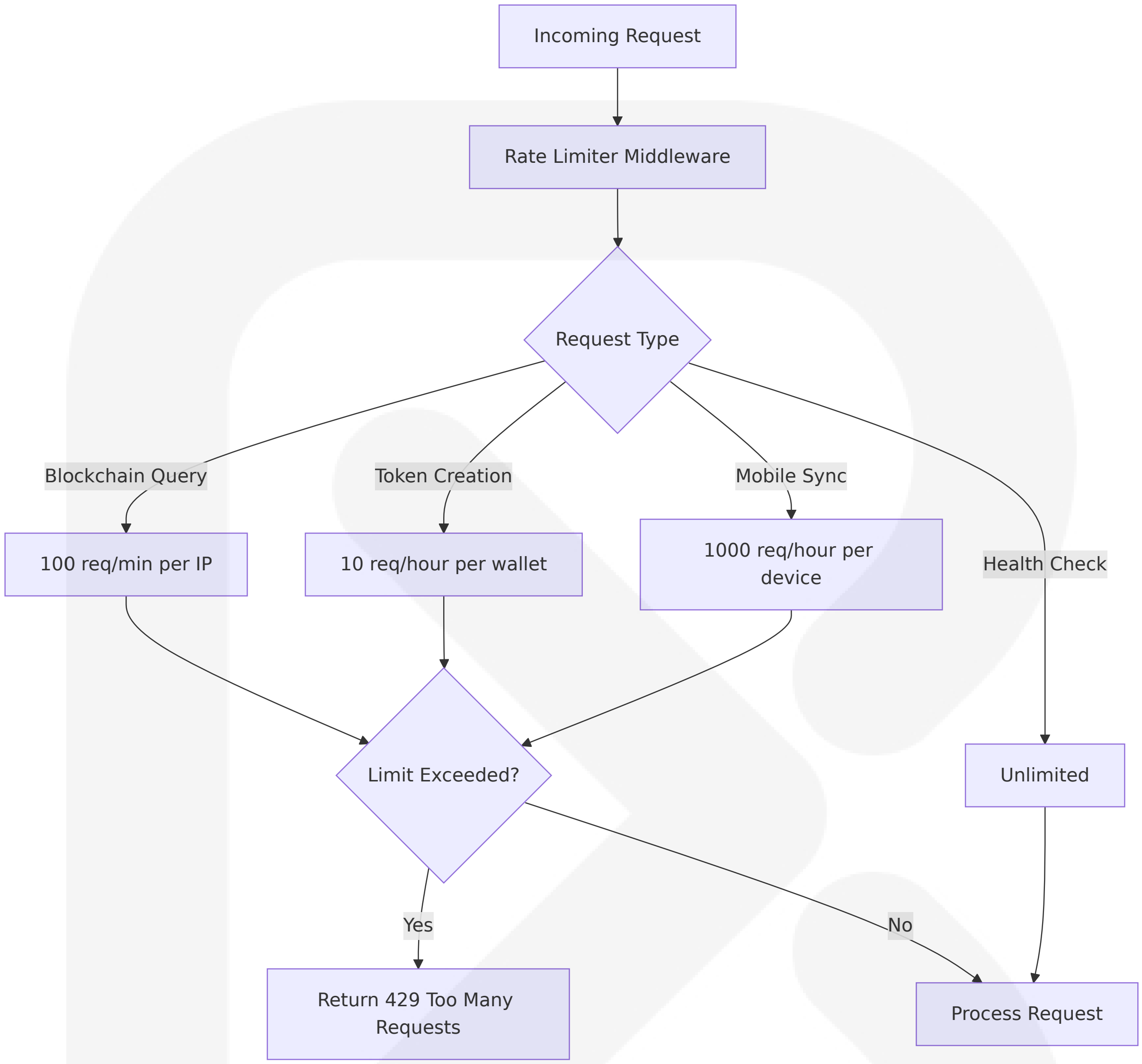
Role	Permissions	Authentication Required	Blockchain Access
Anonymous	Platform browsing, documentation	None	Read-only queries
Connected Wallet	Token creation, payment processing	Wallet signature	Full transaction capabilities
Mobile User	Reward synchronization, mining data	API key + JWT	Limited operational access
System Admin	Health monitoring, configuration	Admin API key	System-level operations

6.3.2.4 Rate Limiting Strategy

Adaptive Rate Limiting Implementation

Express middleware functions perform tasks between request and response, with access to request and response objects enabling sophisticated rate limiting patterns.

Rate Limiting Configuration:



Rate Limiting Specifications:

Endpoint Category	Rate Limit	Window	Identifier
Blockchain Operations	100 requests/minute	Rolling window	IP address + wallet
Token Creation	10 requests/hour	Fixed window	Wallet address
Mobile Synchronization	1000 requests/hour	Rolling window	Device ID
Health Monitoring	Unlimited	N/A	None

6.3.2.5 Versioning Approach

Semantic API Versioning

The API implements semantic versioning with backward compatibility support for mobile applications and external integrations.

Versioning Strategy:

Version Format	Implementation	Compatibility	Migration Path
URL Path Versioning	`/api/v1/`, `/api/v2/`	Explicit version control	Gradual migration
Header Versioning	`API-Version: 1.0`	Optional fallback	Client-controlled
Content Negotiation	`Accept: application/vnd.api+json;version=1`	Advanced clients	Flexible integration

6.3.2.6 Documentation Standards

OpenAPI 3.0 Specification

The API documentation follows OpenAPI 3.0 standards with interactive documentation and code generation capabilities.

Documentation Structure:

Component	Standard	Tool	Purpose
API Specification	OpenAPI 3.0	Swagger/Redoc	Interactive documentation
Code Examples	Multiple languages	Postman collections	Integration guidance
Authentication Guide	Step-by-step	Custom documentation	Developer onboarding
Error Handling	Standardized codes	RFC 7807 Problem Details	Consistent error responses

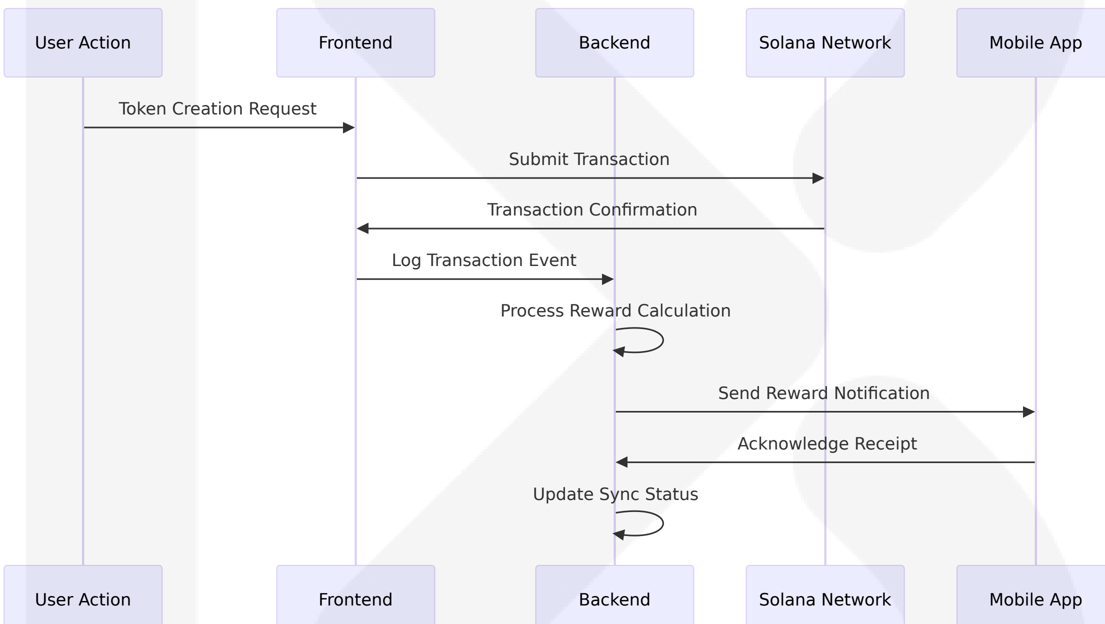
6.3.3 MESSAGE PROCESSING

6.3.3.1 Event Processing Patterns

Blockchain Event-Driven Architecture

The system implements event-driven patterns for blockchain interactions and mobile synchronization, leveraging Node.js event-driven pattern utilizing EventEmitter class for handling events.

Event Processing Flow:



Event Types and Handlers:

Event Type	Source	Handler	Processing Pattern
Token Created	Solana Network	Reward Calculator	Asynchronous batch processing
Payment Processed	Blockchain	Transaction Logger	Real-time logging
Mobile Sync Request	Mobile App	Sync Manager	Queue-based processing

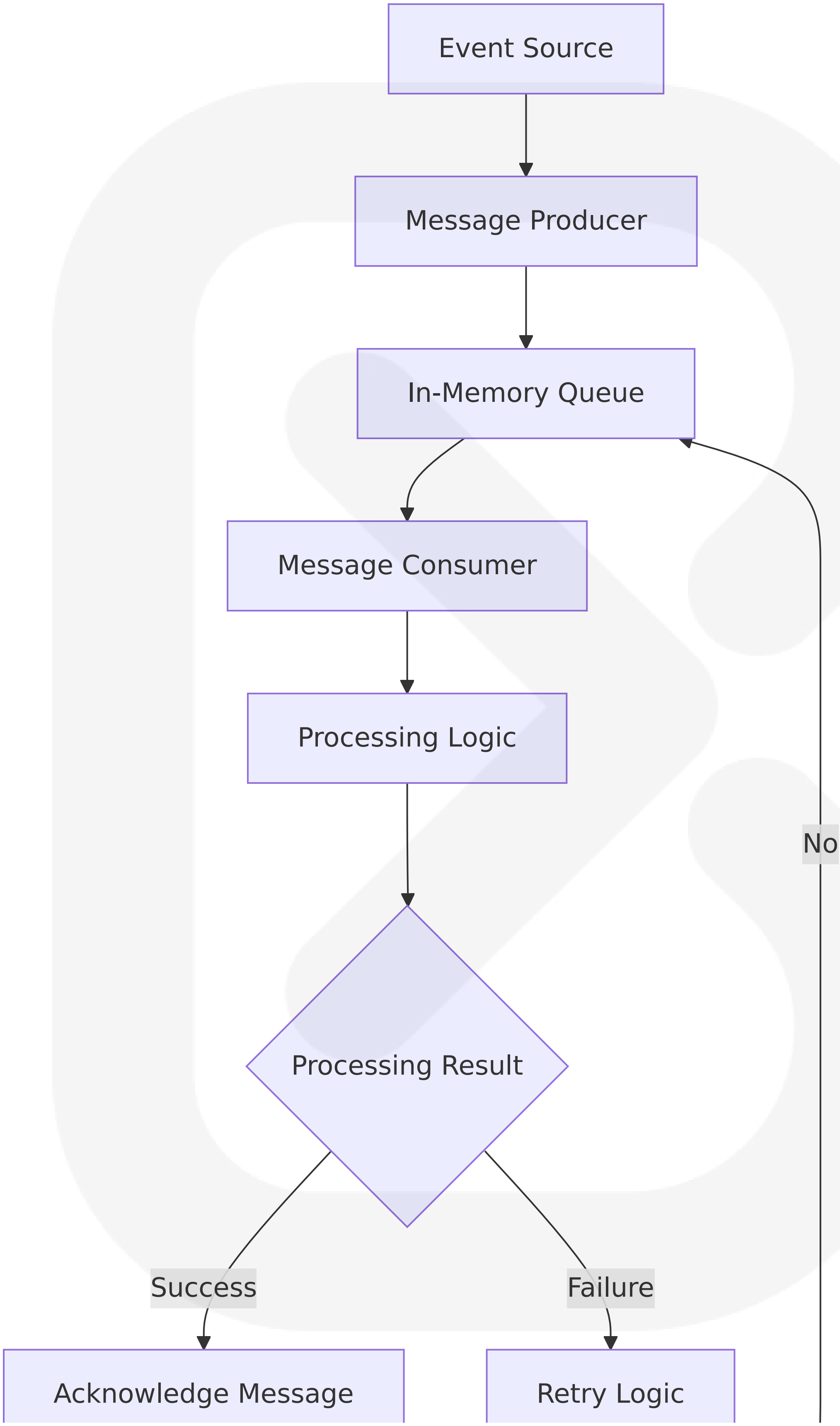
Event Type	Source	Handler	Processing Pattern
System Health Check	Internal	Health Monitor	Scheduled polling

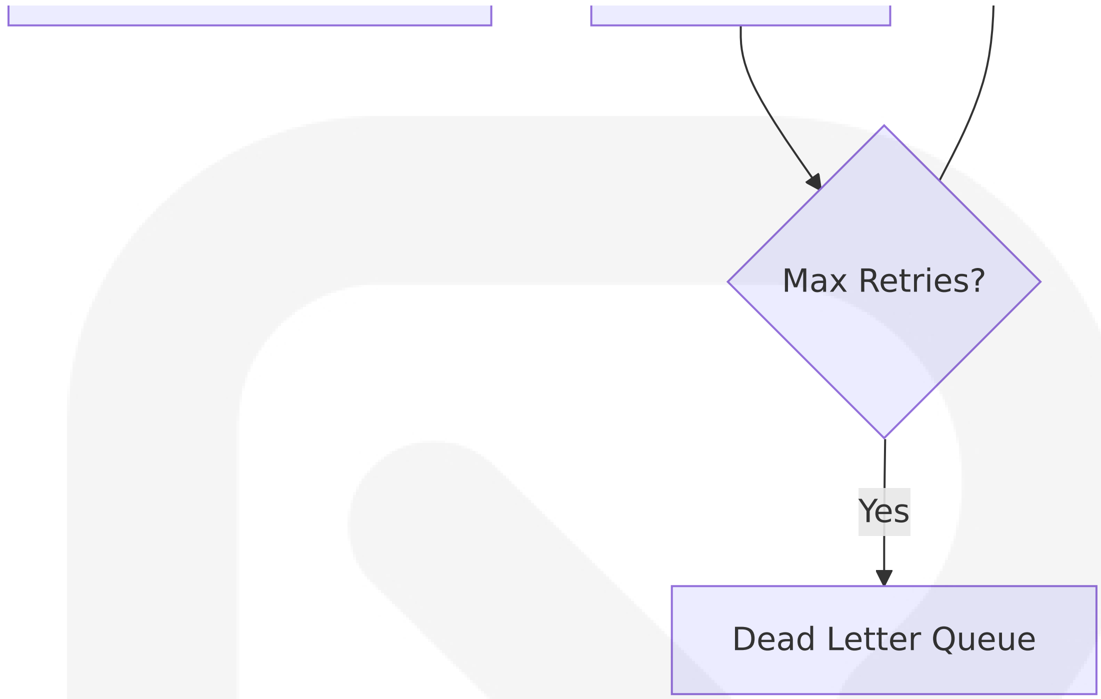
6.3.3.2 Message Queue Architecture

Lightweight Queue Implementation

Given the system's focus on blockchain-first architecture, message queuing is implemented through a lightweight in-memory queue for mobile synchronization with Redis as an optional enhancement.

Queue Architecture Design:


Event Source
Message Producer
In-Memory Queue
Message Consumer
Processing Logic
Processing Result
Success
Failure
No
Acknowledge Message
Retry Logic



Message Queue Specifications:

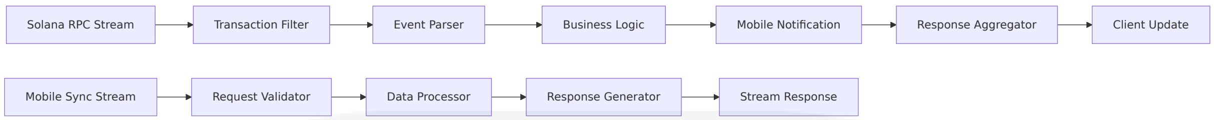
Queue Type	Technology	Use Case	Persistence
Reward Distribution	In-Memory Array	Mobile mining rewards	Session-based
Transaction Logging	Event Buffer	Blockchain event processing	Temporary
Error Handling	Dead Letter Queue	Failed message recovery	Persistent
Health Monitoring	Scheduled Tasks	System status updates	None

6.3.3.3 Stream Processing Design

Real-Time Blockchain Monitoring

The system implements stream processing for real-time blockchain transaction monitoring and mobile synchronization updates.

Stream Processing Pipeline:



6.3.3.4 Batch Processing Flows

Mobile Mining Reward Distribution

Batch processing handles mobile mining reward distribution to optimize blockchain transaction costs and improve system efficiency.

Batch Processing Workflow:

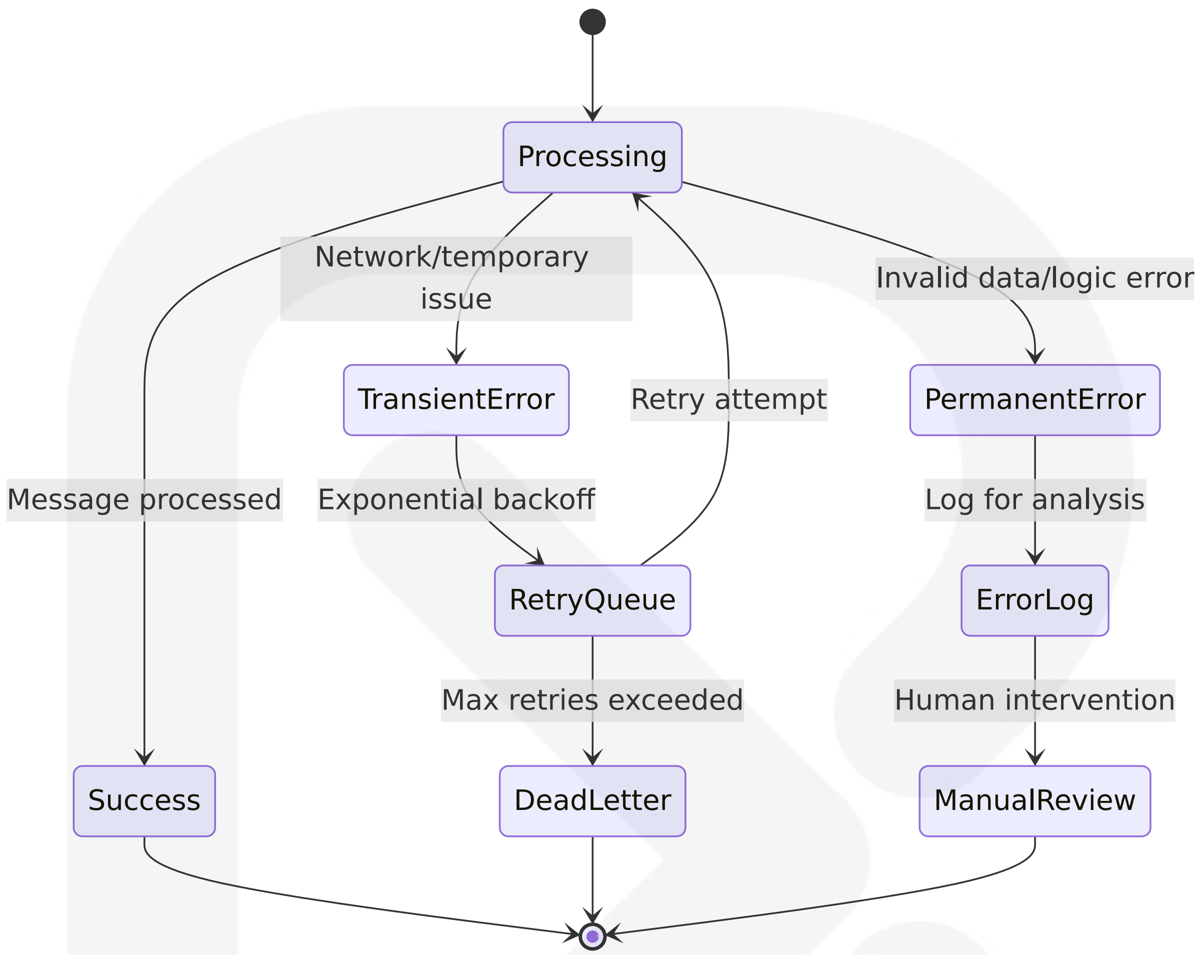
Stage	Process	Frequency	Optimization
Collection	Gather eligible miners	Every 10 minutes	Eligibility filtering
Calculation	Compute reward amounts	Per collection cycle	Bulk calculations
Batching	Group transactions	Optimize for gas costs	Transaction bundling
Execution	Submit to blockchain	When batch is full	Cost-efficient timing

6.3.3.5 Error Handling Strategy

Comprehensive Error Recovery

The message processing system implements multi-level error handling with automatic retry mechanisms and manual intervention capabilities.

Error Handling Hierarchy:



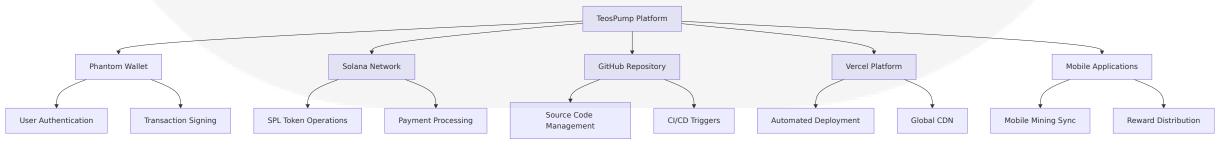
6.3.4 EXTERNAL SYSTEMS

6.3.4.1 Third-Party Integration Patterns

Blockchain and Wallet Integrations

The system integrates with multiple external services following established patterns for reliability and security.

Integration Architecture Overview:



Third-Party Service Specifications:

Service	Integration Type	Protocol	Reliability Pattern
Phantom Wallet	Browser Extension API	JavaScript Provider	Circuit breaker
Solana Network	JSON-RPC	HTTPS	Multiple endpoints
GitHub	Git + Webhooks	HTTPS	Webhook verification
Vercel	Deployment API	REST	Automatic retry

6.3.4.2 Legacy System Interfaces

Legacy System Integration is not applicable for this system as TeosPump is a greenfield project built on modern blockchain infrastructure. The platform does not require integration with legacy financial systems, traditional databases, or existing enterprise applications.

The system's architecture specifically avoids legacy dependencies by:

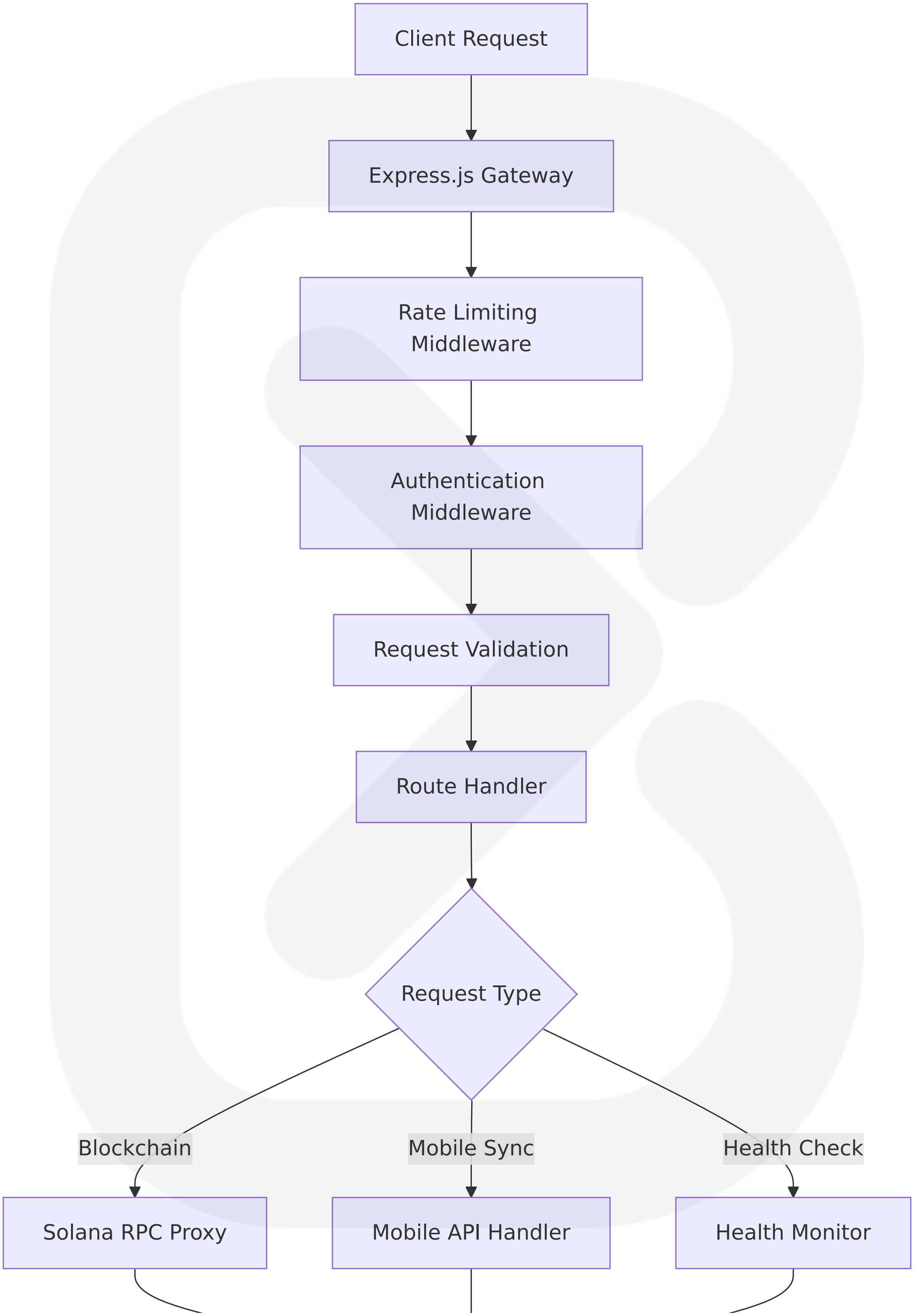
- **Blockchain-Native Design**: All financial operations occur on Solana network
- **Modern Technology Stack**: Web3.js 2.0 SDK introduced November 7, 2024, with modern JavaScript features and improvements
- **Cloud-Native Deployment**: Vercel provides three default environments—Local, Preview, and Production
- **Non-Custodial Architecture**: No traditional user account systems required

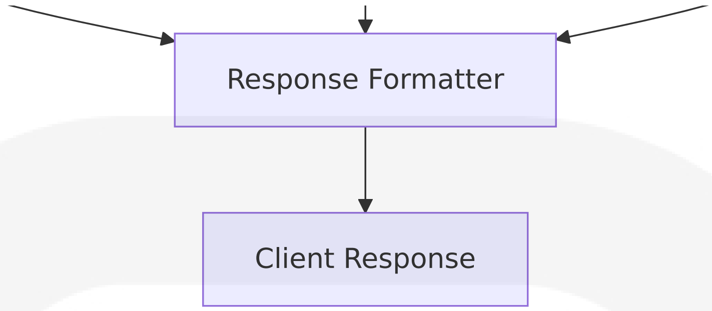
6.3.4.3 API Gateway Configuration

Simplified Gateway Pattern

The system implements a lightweight API gateway pattern through Express.js middleware rather than a dedicated gateway service, optimizing for simplicity and performance.

Gateway Configuration:


Client Request
Express.js Gateway
Rate Limiting Middleware
Authentication Middleware
Request Validation
Route Handler
Request Type
Blockchain
Mobile Sync
Health Check
Solana RPC Proxy
Mobile API Handler
Health Monitor



Gateway Middleware Stack:

Middleware	Purpose	Implementation	Configuration
CORS Handler	Cross-origin requests	Express CORS	Phantom wallet domains
Rate Limiter	Request throttling	Express rate limit	Per-endpoint limits
Body Parser	Request parsing	Express JSON	Size limits
Error Handler	Global error handling	Custom middleware	Structured error responses

6.3.4.4 External Service Contracts

Service Level Agreements (SLAs)

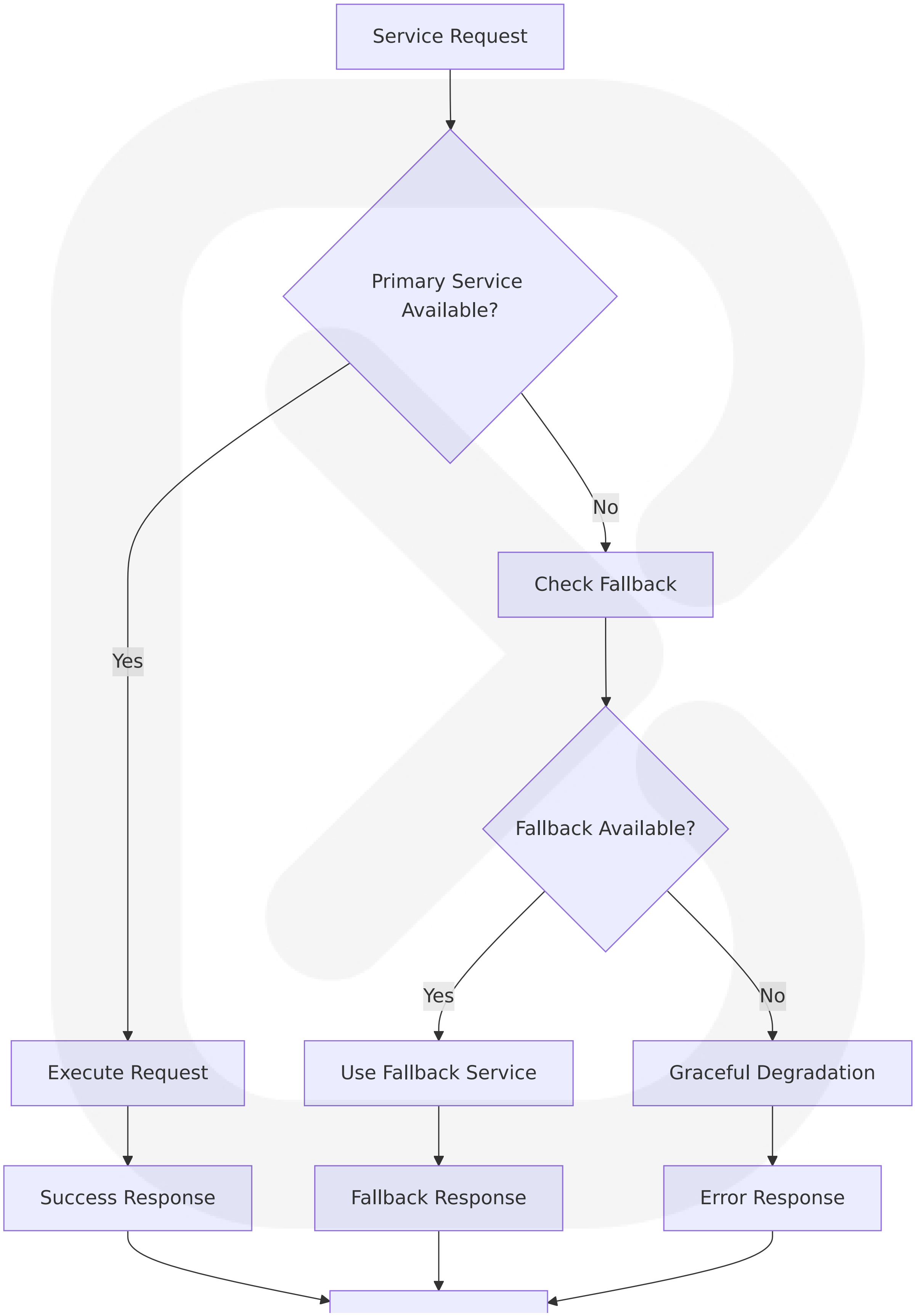
The platform depends on external services with defined availability and performance expectations.

External Service Dependencies:

Service	SLA Requirement	Fallback Strategy	Monitoring
Solana Network	99.9% uptime	Multiple RPC endpoints	Real-time health checks
Phantom Wallet	User-dependent	Manual reconnection prompts	Connection state monitoring

Service	SLA Requirement	Fallback Strategy	Monitoring
Vercel Platform	99.99% uptime	Automatic failover	Built-in monitoring
GitHub	99.95% uptime	Local development fallback	Webhook status tracking

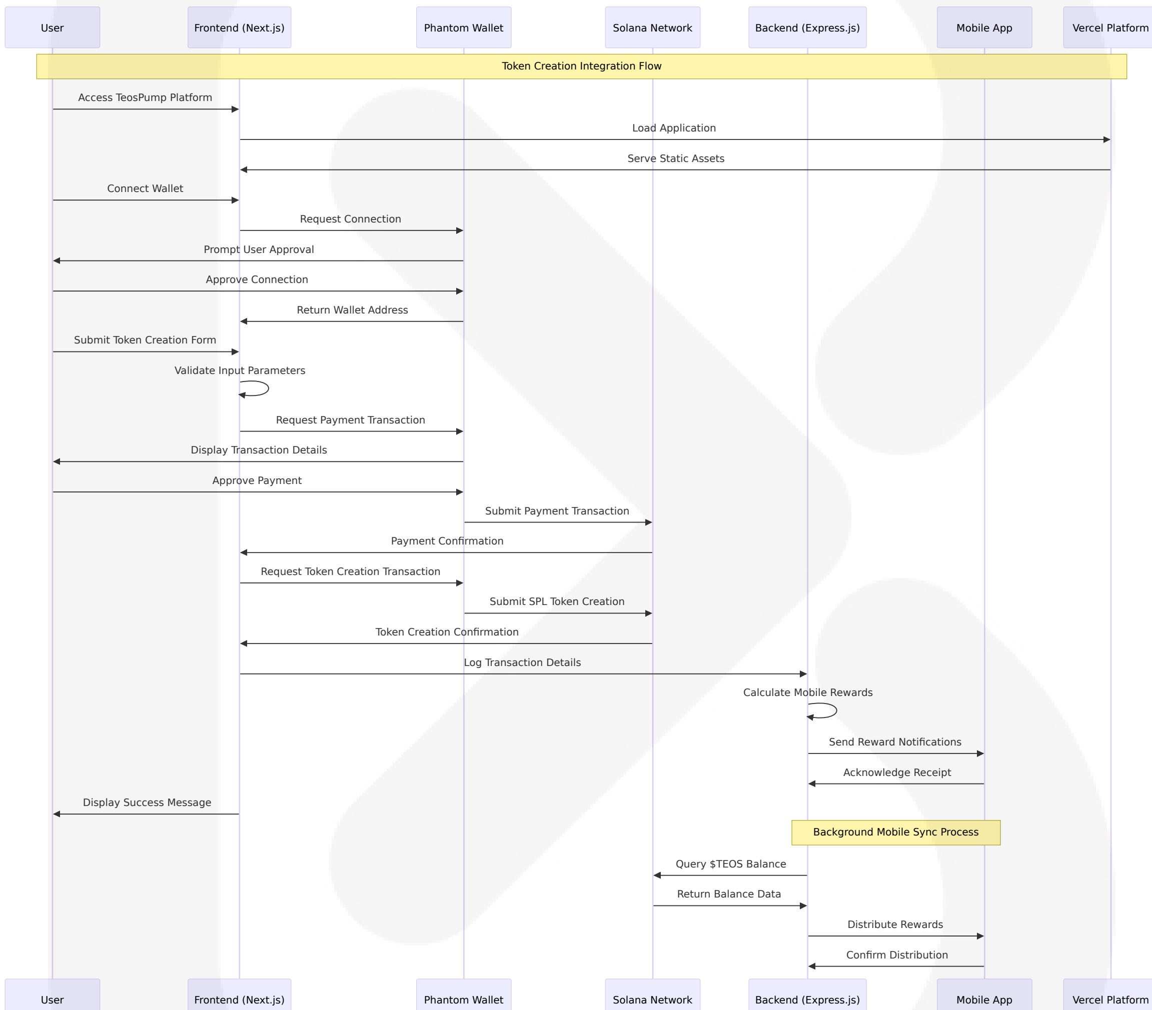
Integration Resilience Patterns:


Service Request
Primary Service Available?
Yes
No
Check Fallback
Fallback Available?
Yes
No
Execute Request
Use Fallback Service
Graceful Degradation
Success Response
Fallback Response
Error Response

6.3.5 INTEGRATION FLOW DIAGRAMS

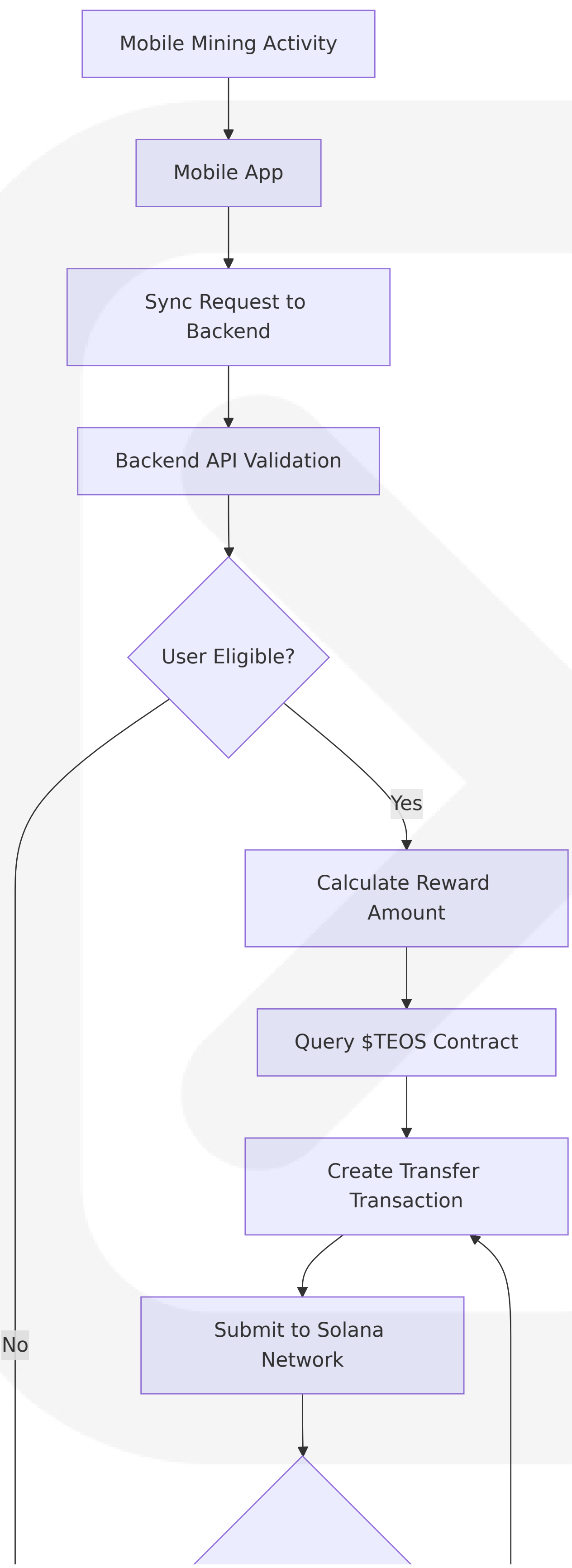
6.3.5.1 Complete Token Creation Integration Flow

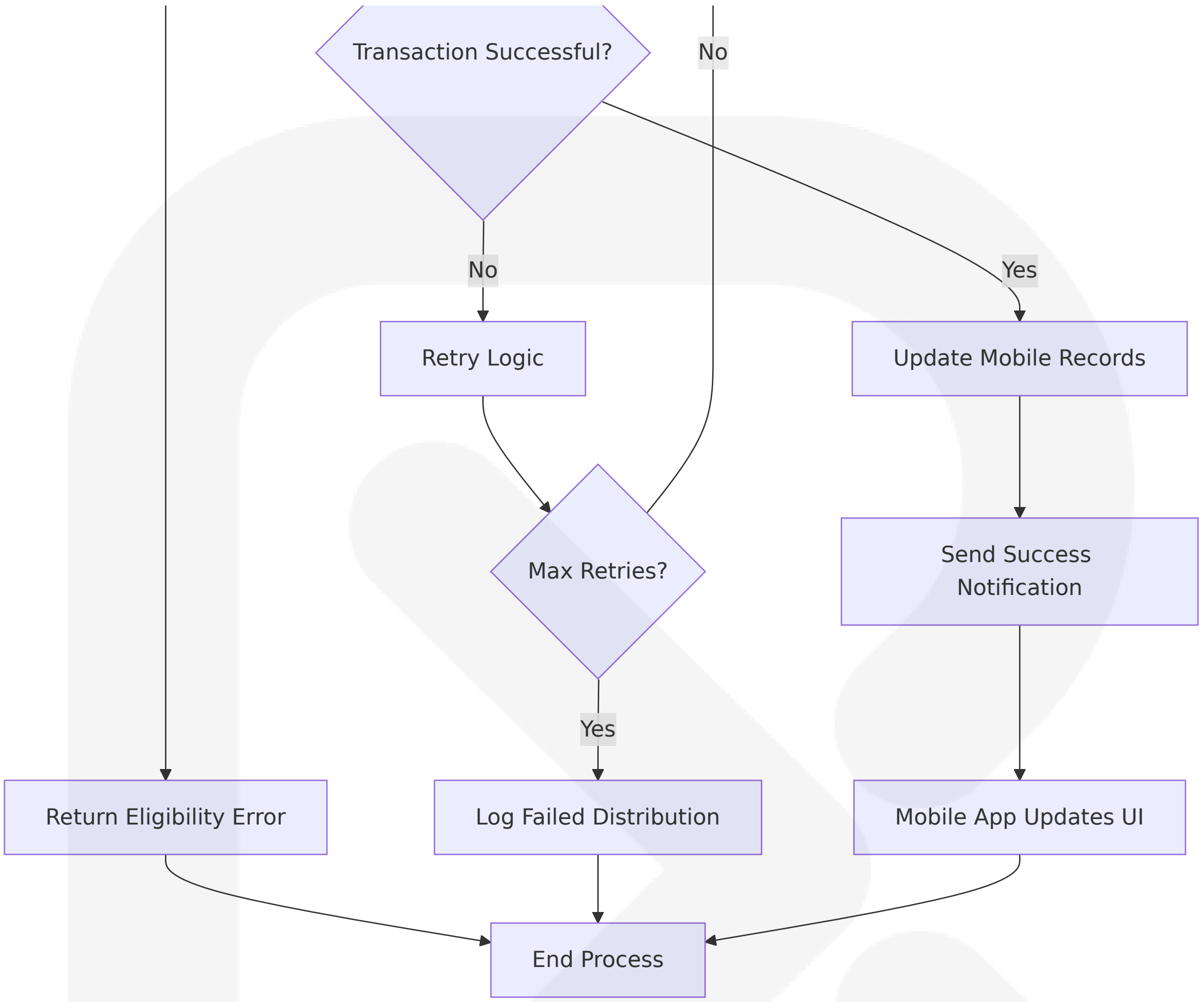
End-to-End Integration Sequence



6.3.5.2 Mobile Mining Integration Flow

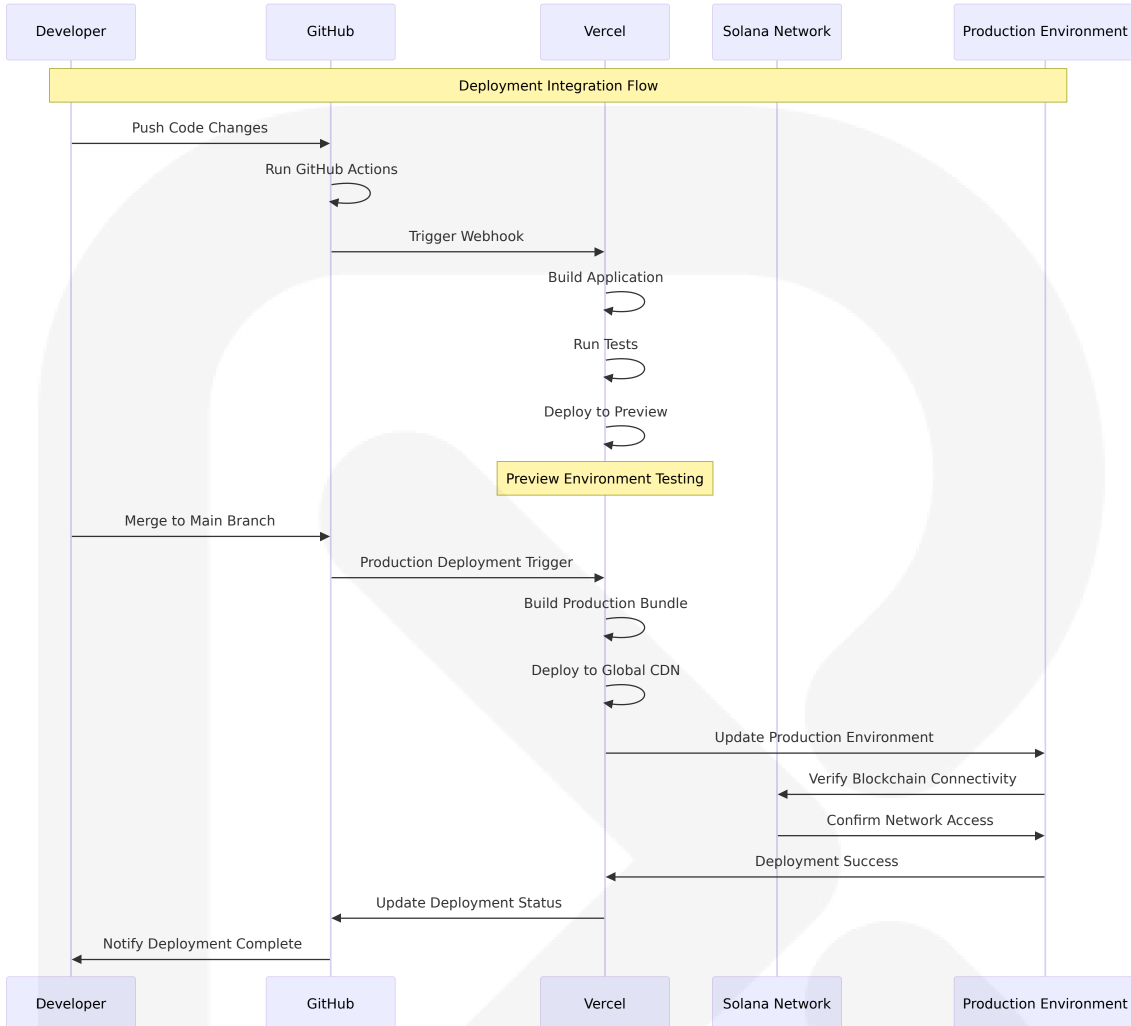
Mobile Synchronization Architecture


Mobile Mining Activity
Mobile App
Sync Request to Backend
Backend API Validation
User Eligible?
Yes
No
Calculate Reward Amount
Query $TEOS Contract
Create Transfer Transaction
Submit to Solana Network



6.3.5.3 Deployment and CI/CD Integration Flow

Automated Deployment Pipeline



6.3.6 PERFORMANCE AND MONITORING

6.3.6.1 Integration Performance Metrics

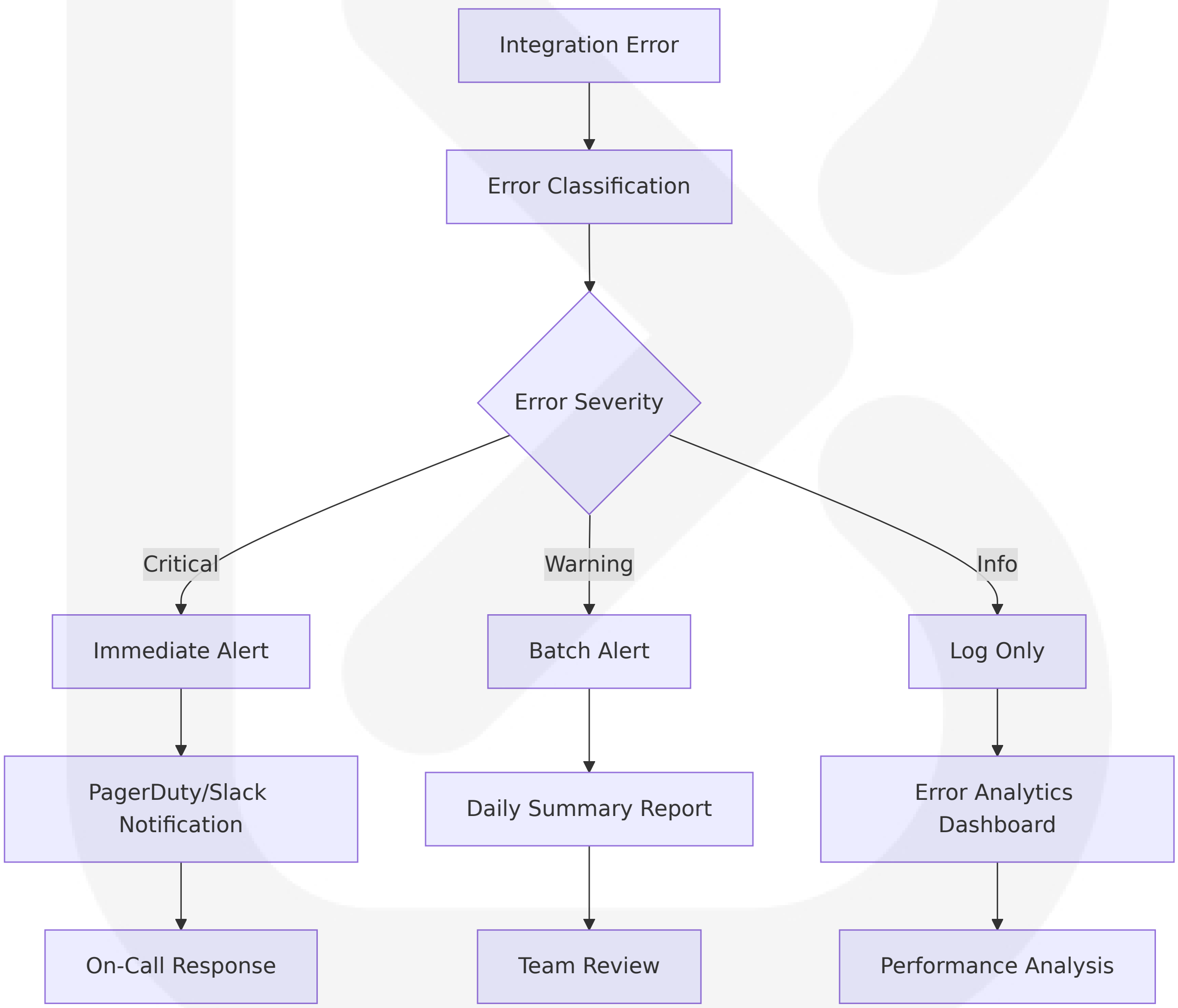
Key Performance Indicators (KPIs)

Integration Point	Metric	Target	Monitoring Method
Phantom Wallet Connection	Connection time	<3 seconds	Frontend performance tracking
Solana Network Interaction	Transaction confirmation	<10 seconds	Blockchain monitoring

Integration Point	Metric	Target	Monitoring Method
Backend API Response	Response time	<500ms	Express.js middleware logging
Mobile Synchronization	Sync completion	<2 seconds	Mobile app telemetry

6.3.6.2 Error Monitoring and Alerting

Comprehensive Error Tracking



6.3.6.3 Integration Health Monitoring

Real-Time Health Checks

The system implements comprehensive health monitoring across all integration points to ensure optimal performance and rapid issue detection.

Health Check Implementation:

Service	Check Type	Frequency	Alert Threshold
Solana RPC	Network connectivity	30 seconds	3 consecutive failures
Phantom Wallet	Provider availability	On user action	Immediate
Backend API	Endpoint health	60 seconds	5xx error rate > 5%
Mobile Sync	Queue processing	5 minutes	Queue depth >100

This comprehensive integration architecture ensures TeosPump operates reliably across all external dependencies while maintaining the security and performance standards required for a blockchain-based token launchpad platform.

6.4 SECURITY ARCHITECTURE

6.4.1 Security Architecture Overview

TeosPump implements a **blockchain-native security architecture** that leverages Solana's cryptographic foundations and Phantom wallet's non-custodial design to provide enterprise-grade security without traditional centralized authentication systems. Developers should consider best practices, including ensuring compliance with regulatory requirements when creating tokens, implementing security measures to protect against unauthorized access and token theft. However, like any blockchain, it is not immune to security vulnerabilities. Implementing best security practices is essential to protect assets and maintain the integrity of applications built on Solana. Security: Implement best practices to safeguard against vulnerabilities and exploits.

The security model prioritizes **decentralized authentication** through cryptographic signatures, **immutable transaction records** on the Solana blockchain, and **privacy-by-design** principles that minimize data collection and storage. Self-custodial means you control your funds. We never have access. Private by design. No name, email, or phone number required.

Core Security Principles:

Principle	Implementation	Security Benefit	Technical Approach
Non-Custodial Design	User-controlled private keys	Eliminates single point of failure	Phantom wallet integration
Blockchain Immutability	Solana transaction records	Tamper-proof financial data	SPL token program verification
Zero-Knowledge Architecture	Minimal data collection	Enhanced privacy protection	Wallet address-only identification
Cryptographic Verification	Digital signature validation	Authentic transaction authorization	ECDSA signature schemes

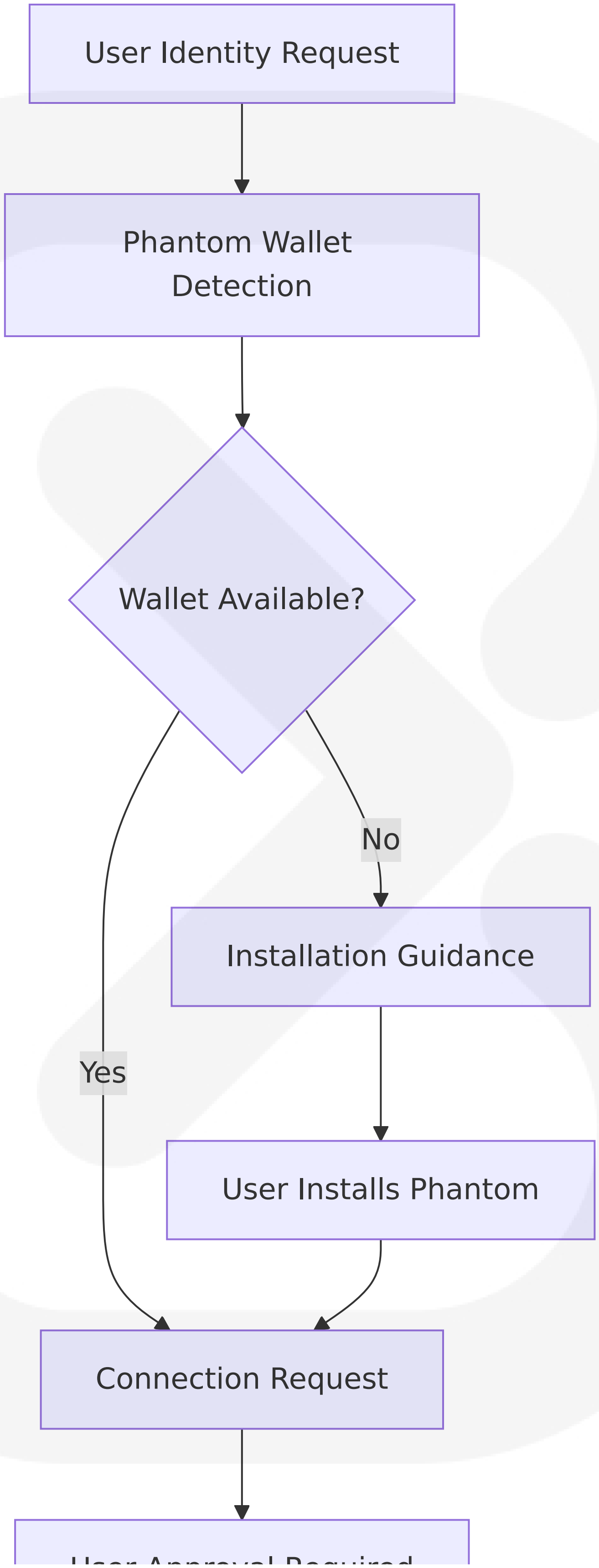
6.4.2 AUTHENTICATION FRAMEWORK

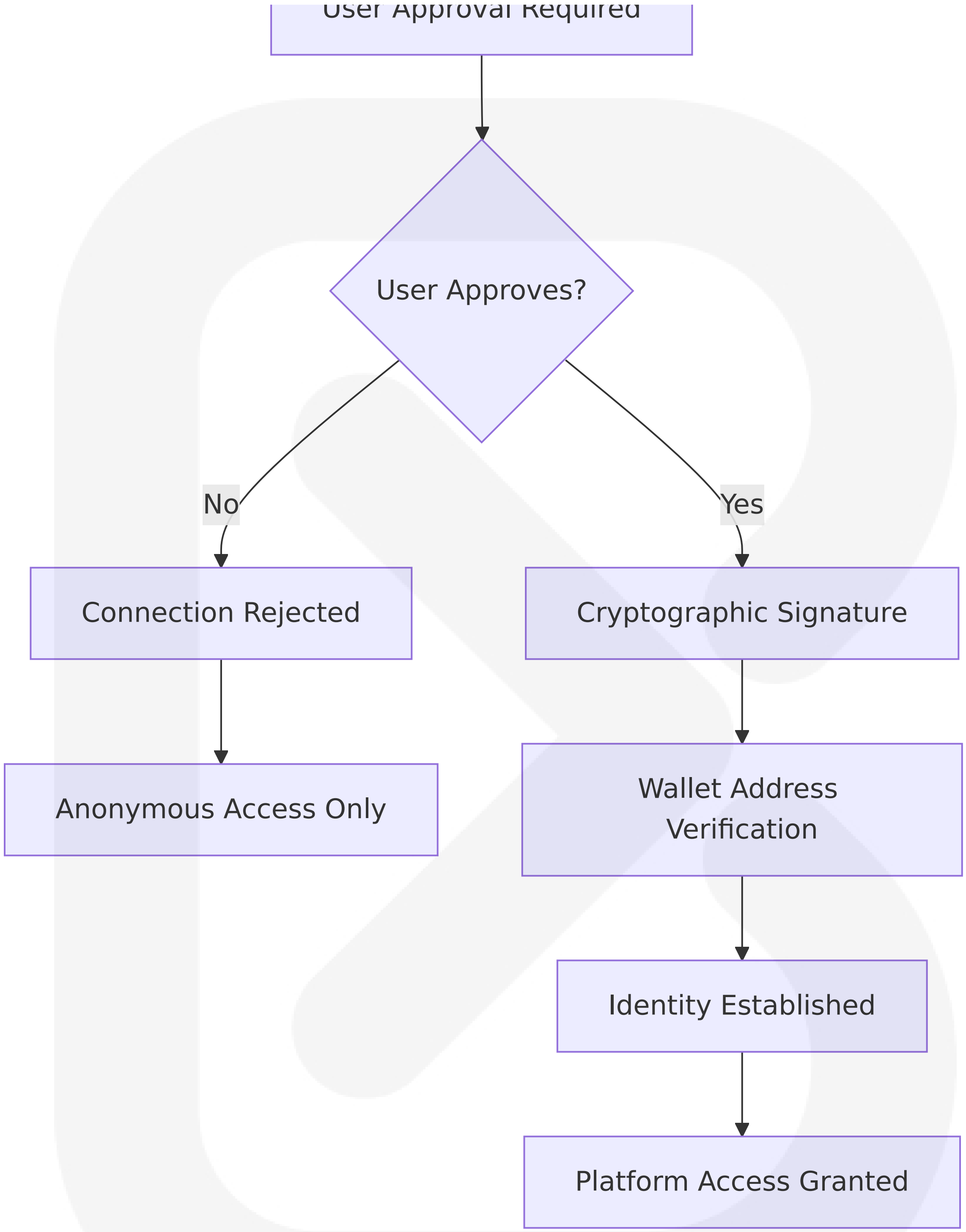
6.4.2.1 Identity Management

Decentralized Identity Architecture

TeosPump implements a **wallet-based identity system** that eliminates traditional username/password authentication in favor of cryptographic proof of ownership. Security features such as encryption, biometric authentication and hardware wallet integration are provided, but users must safeguard their secret recovery phrase to prevent unauthorized access. Security measures: The wallet employs advanced encryption techniques to protect private keys and offers features like biometric authentication on mobile devices. Additionally, Phantom integrates with hardware wallets such as Ledger, providing an extra layer of security by keeping private keys offline.

Identity Management Components:


User Identity Request
Phantom Wallet Detection
Wallet Available?
No
Yes
Installation Guidance
User Installs Phantom
Connection Request



Identity Verification Methods:

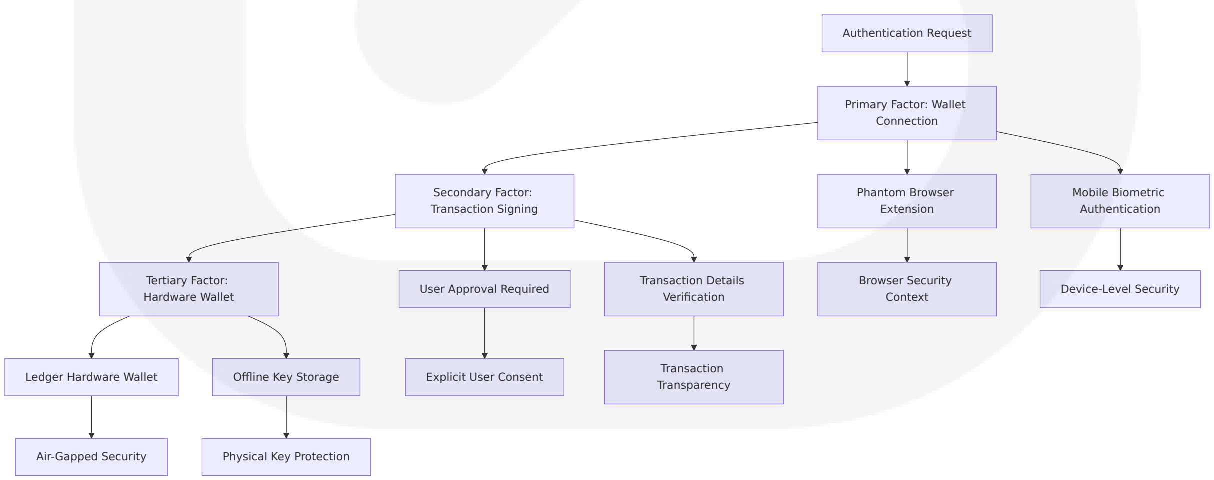
Method	Purpose	Security Level	Implementation
Wallet Signature	Primary authentication	Cryptographic	ECDSA signature verification
Public Key Validation	Address ownership proof	High	Solana address format validation
Transaction Signing	Operation authorization	Maximum	User-approved blockchain transactions

6.4.2.2 Multi-Factor Authentication

Enhanced Security Through Hardware Integration

While traditional MFA is not applicable in a non-custodial environment, TeosPump implements **layered security verification** through multiple authentication factors controlled by the user. Phantom Wallet has additional security features that you can activate to further protect your account. In the Settings > Security & Privacy section, you can activate the auto-lock feature, which will automatically lock your wallet after a certain period of time. Additionally, on mobile devices, you can also use biometric authentication such as FaceID or TouchID to ensure that only you can access your wallet.

Multi-Layer Security Implementation:



Authentication Factor Specifications:

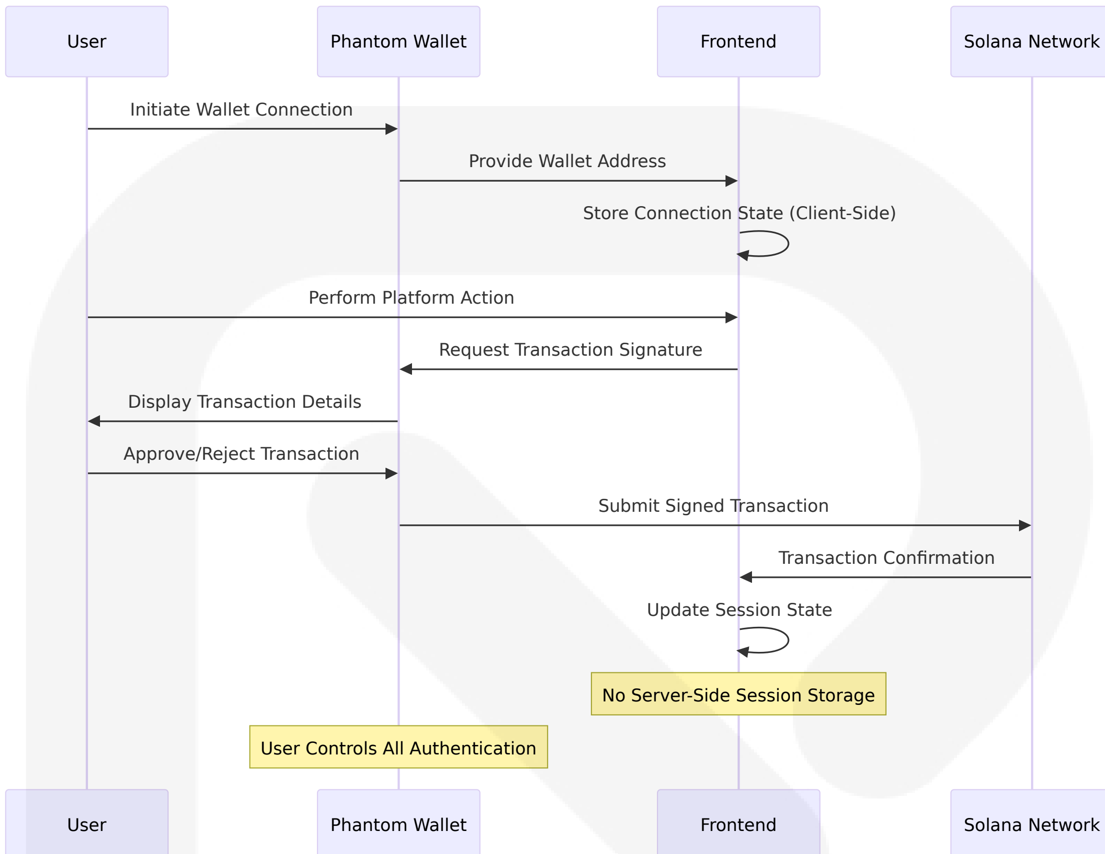
Factor Type	Technology	User Control	Security Benefit
Device Authentication	Browser extension/mobile app	User device management	Device-specific access control
Biometric Verification	FaceID/TouchID/Fingerprint	User biometric setup	Physical presence verification
Hardware Security	Ledger integration	User hardware ownership	Offline key protection
Transaction Approval	Manual signature confirmation	User transaction review	Explicit operation authorization

6.4.2.3 Session Management

Stateless Session Architecture

The platform implements **stateless session management** that eliminates server-side session storage while maintaining secure user interactions through client-side state management and blockchain verification.

Session Management Flow:



Session Security Characteristics:

Aspect	Implementation	Security Advantage	Technical Details
Session Storage	Client-side only	No server-side vulnerabilities	Browser local storage
Session Duration	User-controlled	No forced timeouts	Wallet connection persistence
Session Validation	Cryptographic signatures	Tamper-proof verification	ECDSA signature validation
Session Termination	User-initiated disconnection	Complete user control	Wallet provider API

6.4.2.4 Token Handling

Blockchain-Native Token Security

Token handling security leverages Solana's native cryptographic capabilities and SPL token program verification. A Token program on the Solana blockchain · This program defines a common implementation for Fungible and Non Fungible tokens

Token Security Implementation:

Token Operation	Security Mechanism	Verification Method	Error Handling
$TEOS Payments	SPL token transfer verification	Blockchain transaction confirmation	Automatic refund on failure
Token Creation	Mint authority validation	Owner wallet signature verification	Transaction rollback capability
Balance Queries	RPC endpoint validation	Multiple endpoint verification	Fallback endpoint switching
Metadata Storage	On-chain immutable records	Cryptographic hash verification	Integrity validation checks

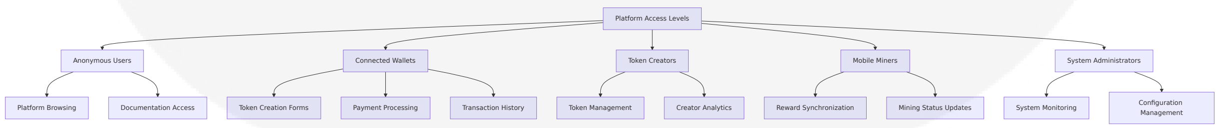
6.4.3 AUTHORIZATION SYSTEM

6.4.3.1 Role-Based Access Control

Simplified RBAC for Decentralized Applications

TeosPump implements a **minimal RBAC system** tailored for blockchain applications where user roles are determined by wallet capabilities and transaction history rather than traditional user accounts.

Authorization Levels and Permissions:



Role Permission Matrix:

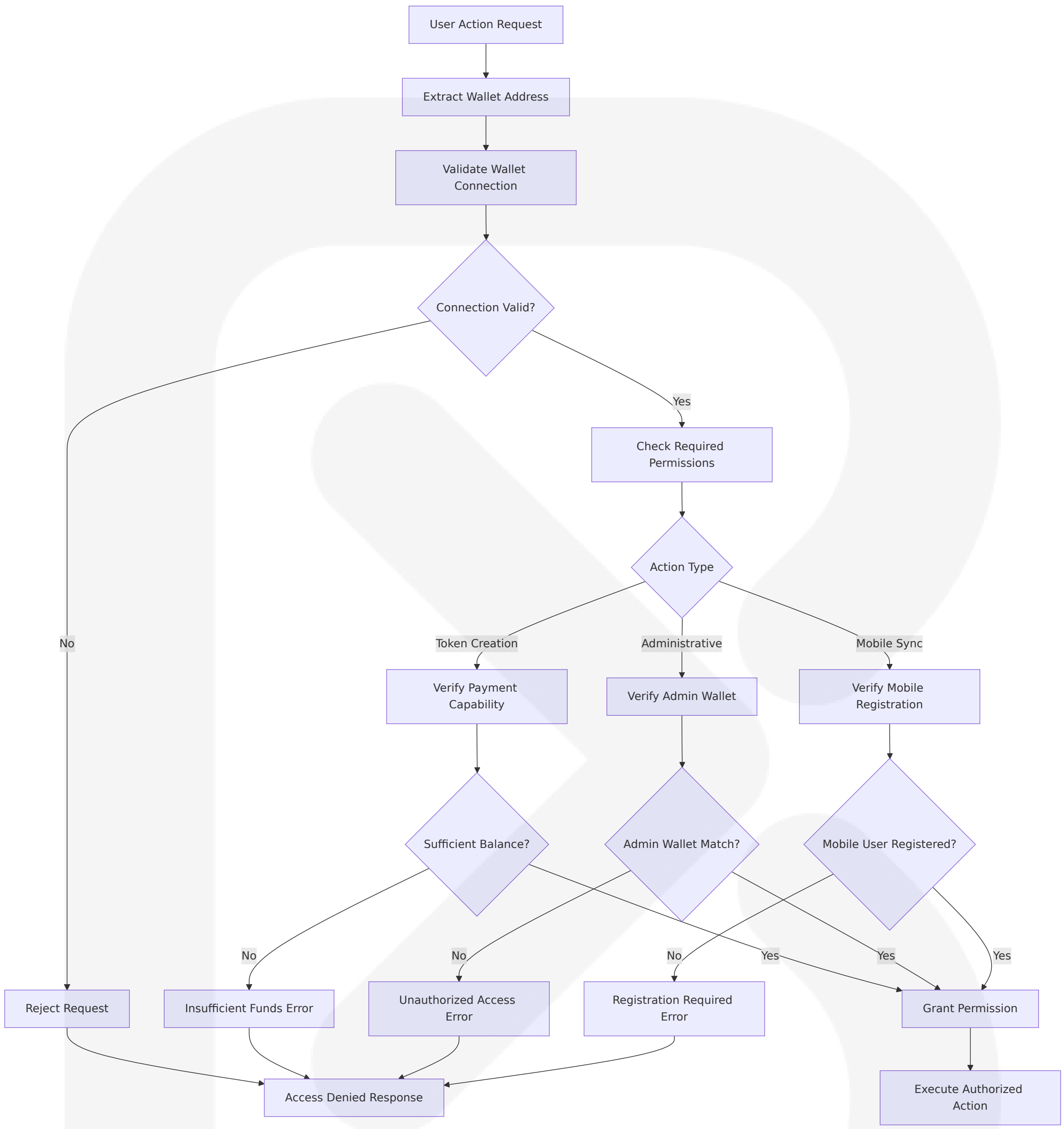
Role	Platform Access	Token Creation	Payment Processing	Administrative Functions
Anonymous	Read-only browsing	□	□	□
Connected Wallet	Full platform access	□	□	□
Token Creator	Enhanced analytics	□	□	□
Mobile Miner	Reward interfaces	□	Limited	□
System Admin	Full access	□	□	□

6.4.3.2 Permission Management

Dynamic Permission Validation

Permission management operates through **real-time blockchain verification** and **cryptographic signature validation** rather than traditional permission databases.

Permission Validation Flow:



6.4.3.3 Resource Authorization

Blockchain-Based Resource Protection

Resource authorization leverages **smart contract validation** and **transaction-based access control** to protect platform resources and user assets.

Resource Protection Mechanisms:

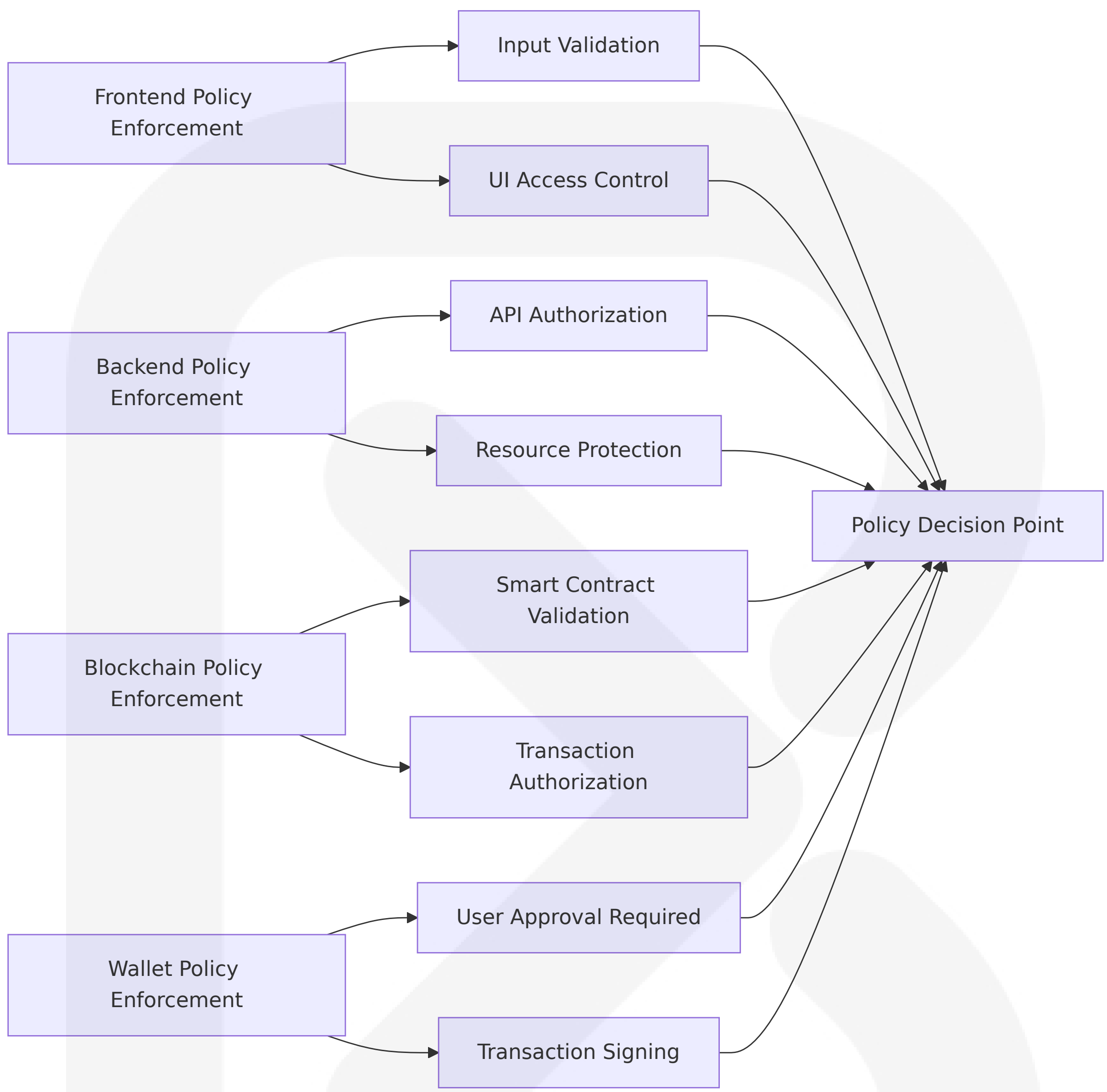
Resource Type	Protection Method	Access Control	Validation Process
Token Creation	Payment verification	$TEOS/SOL balance check	Pre-transaction validation
Owner Wallet Access	Signature verification	Cryptographic proof	Multi-signature validation
Mobile Rewards	Eligibility verification	Mining activity validation	Blockchain history analysis
Platform Analytics	Role-based filtering	Wallet-based permissions	Real-time authorization

6.4.3.4 Policy Enforcement Points

Distributed Policy Enforcement

Policy enforcement occurs at **multiple system layers** to ensure comprehensive security coverage without relying on centralized policy servers.

Policy Enforcement Architecture:



6.4.3.5 Audit Logging

Immutable Audit Trail

Audit logging leverages **blockchain immutability** for financial operations and **structured logging** for operational activities. All token operations are recorded on the Solana blockchain, allowing users to verify transactions via blockchain explorers like Solscan.

Audit Logging Strategy:

Event Category	Logging Method	Retention Period	Access Control
Financial Transactions	Blockchain records	Permanent	Public blockchain
Authentication Events	Backend logging	90 days	Administrative access
Authorization Failures	Security logging	180 days	Security team access
System Operations	Application logging	30 days	Operations team access

6.4.4 DATA PROTECTION

6.4.4.1 Encryption Standards

Multi-Layer Encryption Implementation

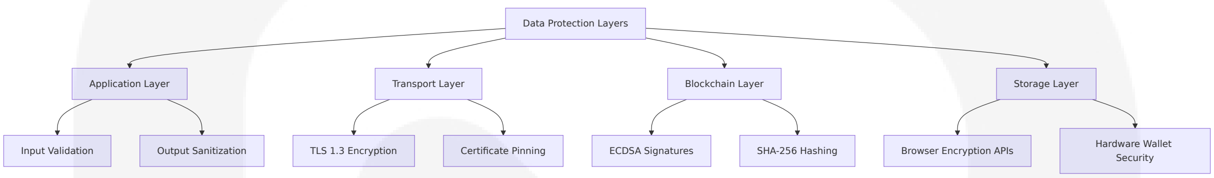
Data protection employs **industry-standard encryption** at multiple layers while leveraging blockchain cryptography for financial data security. Prioritize secure coding practices. Use audited libraries, validate inputs rigorously, and implement access controls. Regularly update dependencies and engage in code reviews to identify and address potential vulnerabilities.

Encryption Implementation Matrix:

Data Type	Encryption Method	Key Management	Security Level
Private Keys	Hardware/Software wallet encryption	User-controlled	Maximum
Transaction Data	Blockchain cryptographic hashing	Network consensus	High
API Communications	TLS 1.3 encryption	Certificate management	High

Data Type	Encryption Method	Key Management	Security Level
Local Storage	Browser encryption APIs	Client-side keys	Medium

Encryption Architecture:



6.4.4.2 Key Management

Decentralized Key Management

Key management follows **non-custodial principles** where users maintain complete control over their cryptographic keys through hardware and software wallet solutions. For users who prioritize security, Phantom integrates seamlessly with Ledger hardware wallets like Ledger Nano X and Ledger Nano S Plus. This setup ensures your private keys stay offline, significantly reducing hacking risks. Setting up Ledger integration is straightforward, but it requires enabling settings like "Allow Blind Signing" on your device. This extra layer of security makes Phantom a preferred choice for users managing large portfolios or sensitive transactions.

Key Management Hierarchy:

Key Type	Storage Location	User Control	Recovery Method
Private Keys	User wallet (hardware/software)	Complete	Seed phrase recovery
API Keys	Environment variables	Administrative	Secure configuration management
Encryption Keys	Browser secure storage	User device	Local backup procedures

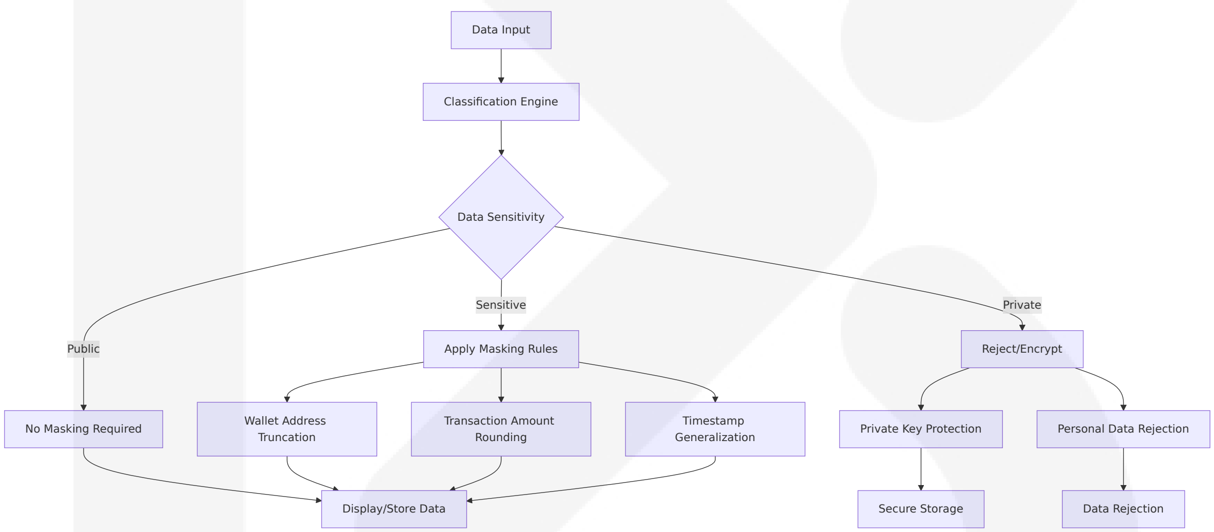
Key Type	Storage Location	User Control	Recovery Method
Signing Keys	Phantom wallet	User approval	Wallet recovery process

6.4.4.3 Data Masking Rules

Privacy-First Data Handling

Data masking implements **minimal data exposure** principles with privacy-by-design architecture that reduces sensitive data handling. Private by design. No name, email, or phone number required.

Data Masking Implementation:



Data Masking Rules:

Data Type	Masking Method	Display Format	Security Rationale
Wallet Addresses	Truncation	`Akvm...Zh4P`	Privacy protection
Transaction Amounts	Rounding	Approximate values	Financial privacy
Private Keys	Complete hiding	Never displayed	Security requirement

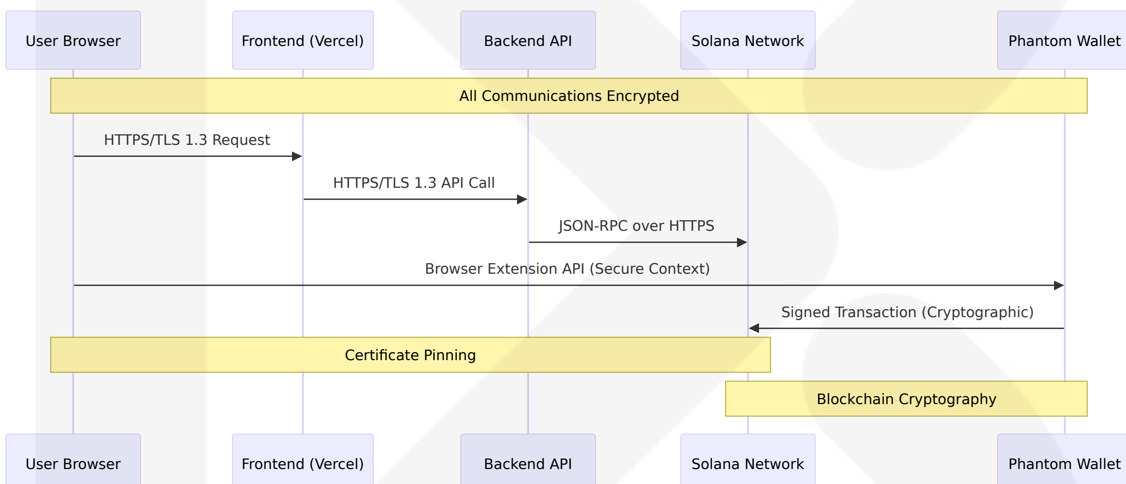
Data Type	Masking Method	Display Format	Security Rationale
Personal Information	Rejection	Not collected	Privacy by design

6.4.4.4 Secure Communication

End-to-End Secure Communication

Secure communication implements **multiple security layers** for all data transmission between system components and external services.

Communication Security Architecture:



Communication Security Standards:

Communication Path	Protocol	Encryption	Authentication
User ↔ Frontend	HTTPS/TLS 1.3	AES-256-GCM	Certificate validation
Frontend ↔ Backend	HTTPS/TLS 1.3	AES-256-GCM	API key authentication
Backend ↔ Blockchain	JSON-RPC/HTTPS	TLS encryption	RPC endpoint validation

Communication Path	Protocol	Encryption	Authentication
User ↔ Wallet	Browser Extension API	Browser security context	Origin validation

6.4.4.5 Compliance Controls

Regulatory Compliance Framework

Compliance controls implement **blockchain-native compliance** that leverages immutable records and cryptographic verification for regulatory requirements.

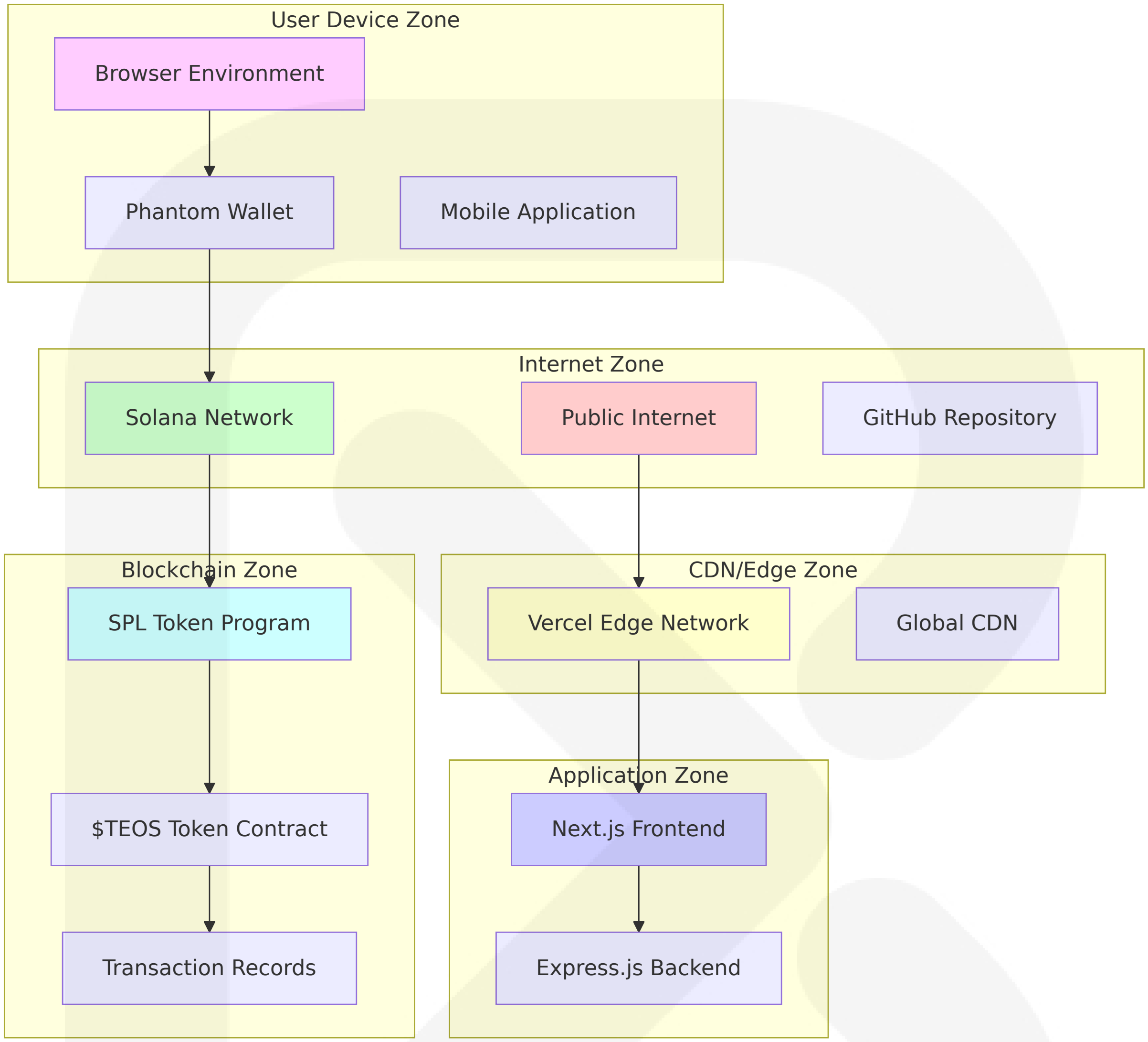
Compliance Implementation:

Compliance Area	Implementation Method	Blockchain Benefit	Traditional Challenge
Data Retention	Immutable blockchain records	Permanent audit trail	Complex retention policies
Data Portability	Public blockchain access	Universal data access	Vendor lock-in concerns
Right to Erasure	Pseudonymous addresses	Privacy through anonymity	Complex deletion procedures
Audit Requirements	Cryptographic verification	Tamper-proof records	Database integrity challenges

6.4.5 SECURITY ZONE DIAGRAMS

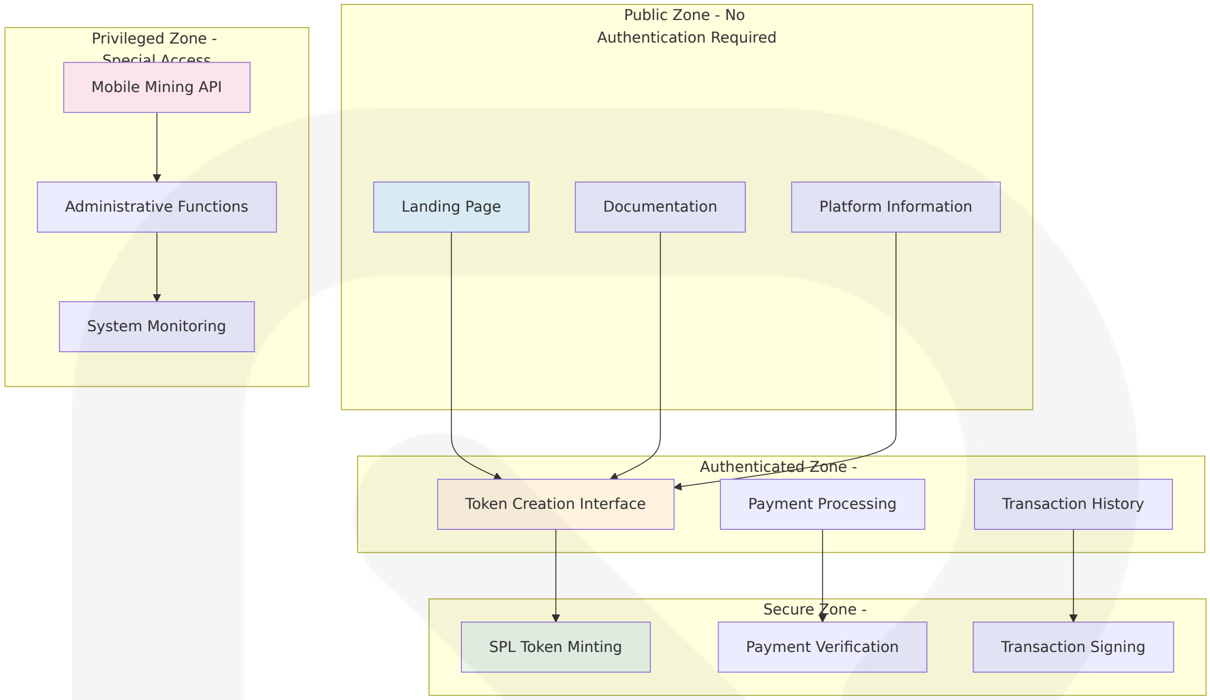
6.4.5.1 Network Security Zones

Decentralized Security Zone Architecture



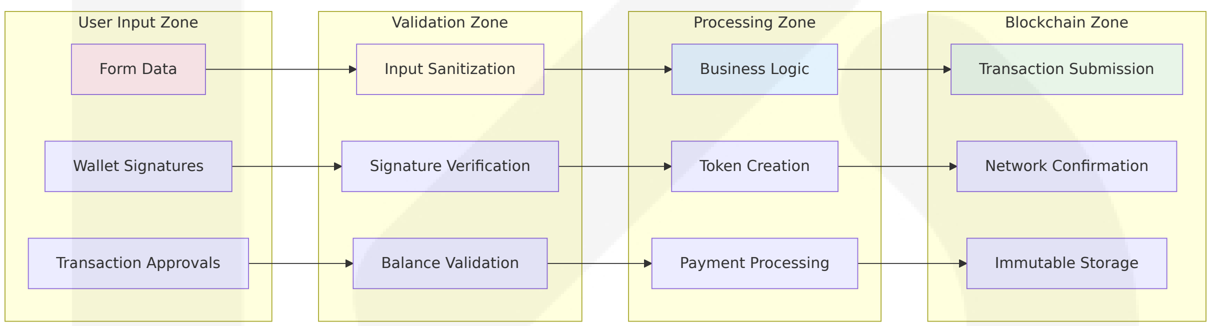
6.4.5.2 Application Security Zones

Component-Level Security Boundaries



6.4.5.3 Data Flow Security Zones

Secure Data Movement Architecture



6.4.6 SECURITY MONITORING AND INCIDENT RESPONSE

6.4.6.1 Real-Time Security Monitoring

Comprehensive Security Monitoring Strategy

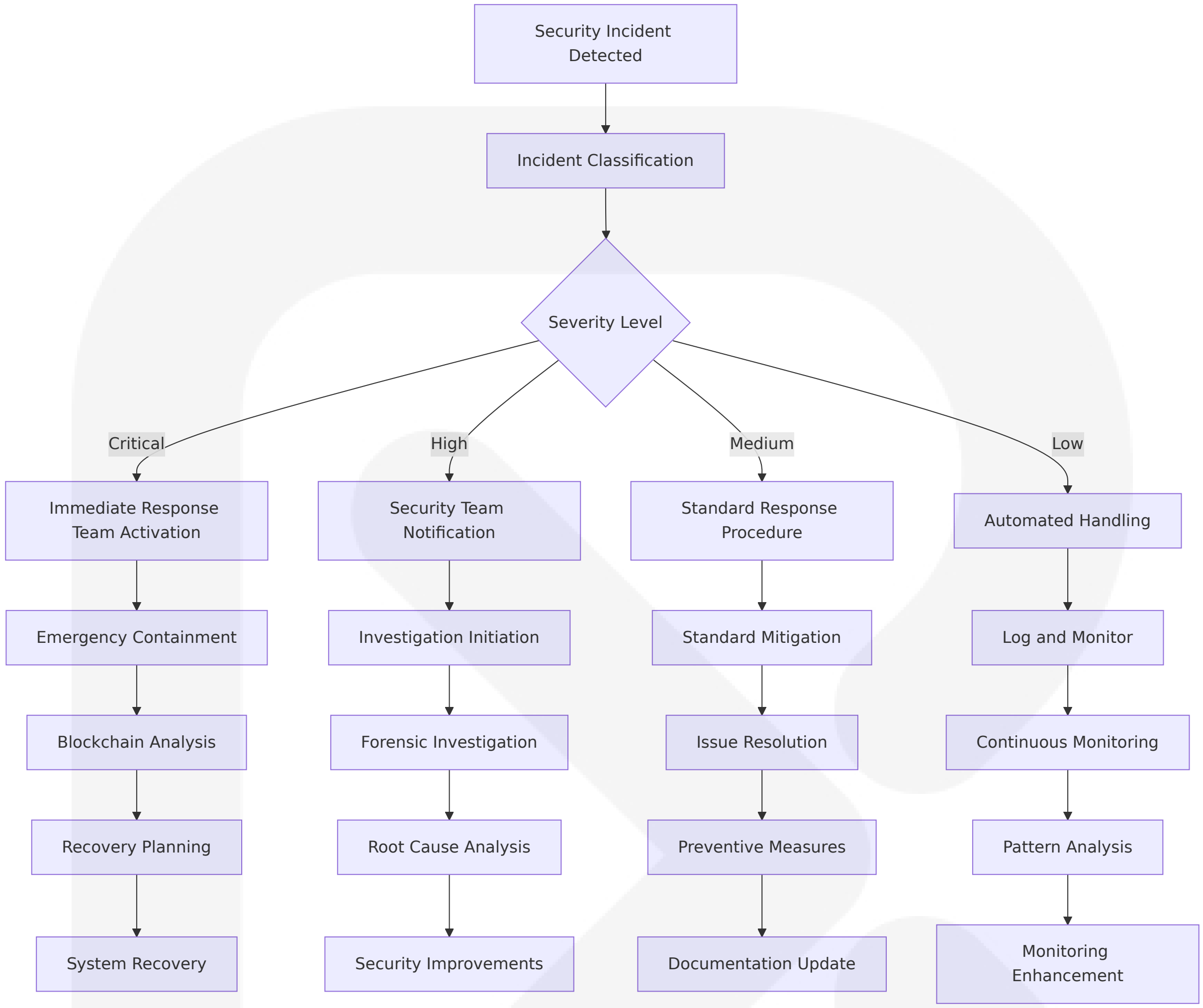
The wallet implements several layers of security, including scam detection that flags suspicious transactions and malicious activity. Phantom also uses an open-source blocklist to protect users from phishing websites. The wallet implements several layers of security, including scam detection that flags suspicious transactions and malicious activity. Phantom also uses an open-source blocklist to protect users from phishing websites.

Security Monitoring Implementation:

Monitoring Layer	Detection Method	Response Time	Alert Threshold
Blockchain Monitoring	Transaction pattern analysis	Real-time	Suspicious transaction patterns
Application Monitoring	Error rate tracking	<1 minute	>5% error rate
Infrastructure Monitoring	Performance metrics	<30 seconds	>80% resource utilization
Security Event Monitoring	Anomaly detection	Immediate	Security policy violations

6.4.6.2 Incident Response Procedures

Blockchain-Native Incident Response



6.4.6.3 Security Metrics and KPIs

Security Performance Indicators

Security Metric	Target Value	Measurement Method	Alert Condition
Authentication Success Rate	>99%	Wallet connection tracking	<95% success rate
Transaction Security Score	>95%	Blockchain verification	Suspicious transaction patterns
Incident Response Time	<15 minutes	Automated monitoring	Response time > 30 minutes

Security Metric	Target Value	Measurement Method	Alert Condition
Security Vulnerability Count	0 critical	Regular security audits	Any critical vulnerabilities

This comprehensive security architecture ensures TeosPump operates with enterprise-grade security while maintaining the decentralized, non-custodial principles that make blockchain applications secure and trustworthy. The architecture leverages proven blockchain security patterns while implementing modern security practices appropriate for a token launchpad platform.

6.5 MONITORING AND OBSERVABILITY

6.5.1 MONITORING INFRASTRUCTURE

6.5.1.1 Metrics Collection Strategy

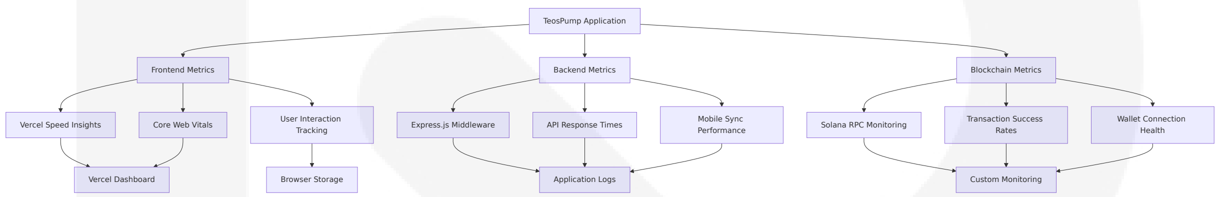
TeosPump implements a **lightweight monitoring architecture** optimized for blockchain-native applications with minimal traditional infrastructure dependencies. Detect and diagnose issues in your web applications by surfacing errors, traffic, and performance data with built-in Monitoring. Track and analyze the performance of different parts of your site right in the Vercel dashboard.

Core Metrics Collection Framework:

Metric Category	Collection Method	Storage Location	Retention Period
Application Performance	Vercel built-in monitoring	Vercel dashboard	30 days
Blockchain Interactions	Custom RPC monitoring	Local logs	7 days

Metric Category	Collection Method	Storage Location	Retention Period
User Experience	Frontend performance tracking	Browser analytics	Session-based
System Health	Express.js middleware	Application logs	14 days

Metrics Collection Architecture:



6.5.1.2 Log Aggregation Implementation

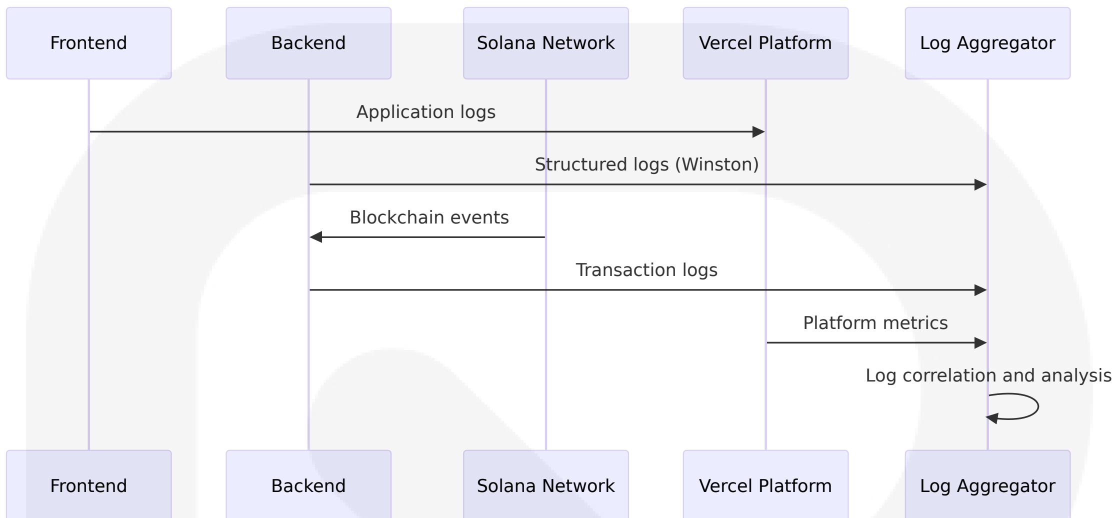
Structured Logging Strategy

The system implements structured logging across all components with Logging helps capture real-time events, errors, and other important information from the application, while monitoring involves tracking application performance metrics over time. Together, they provide critical insights into application health, enabling proactive issue resolution.

Logging Architecture Components:

Component	Logging Framework	Log Format	Destination
Next.js Frontend	Console API + Vercel	JSON structured	Vercel Functions logs
Express.js Backend	Winston	JSON structured	Application logs
Blockchain Integration	Custom logging	Structured events	Transaction logs
Error Tracking	Built-in error boundaries	Error objects	Centralized error logs

Log Aggregation Flow:



6.5.1.3 Distributed Tracing Approach

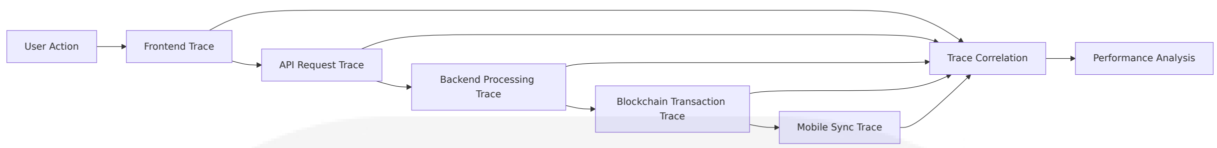
Simplified Tracing for Blockchain Applications

This configuration auto-instruments all the basic HTTP handlers for page routes and API routes and emits traces with Next.js and /or Vercel specific tags. For instance, you might find a trace with the following properties

Tracing Implementation Strategy:

Trace Type	Implementation	Scope	Tools
Frontend Traces	Next.js built-in instrumentation	Page loads, API calls	Vercel observability
Backend Traces	Express.js middleware	API endpoints, database queries	Custom middleware
Blockchain Traces	Transaction monitoring	Wallet interactions, token creation	Custom tracking
End-to-End Traces	Request correlation	User journey tracking	Manual correlation

Distributed Tracing Architecture:



6.5.1.4 Alert Management System

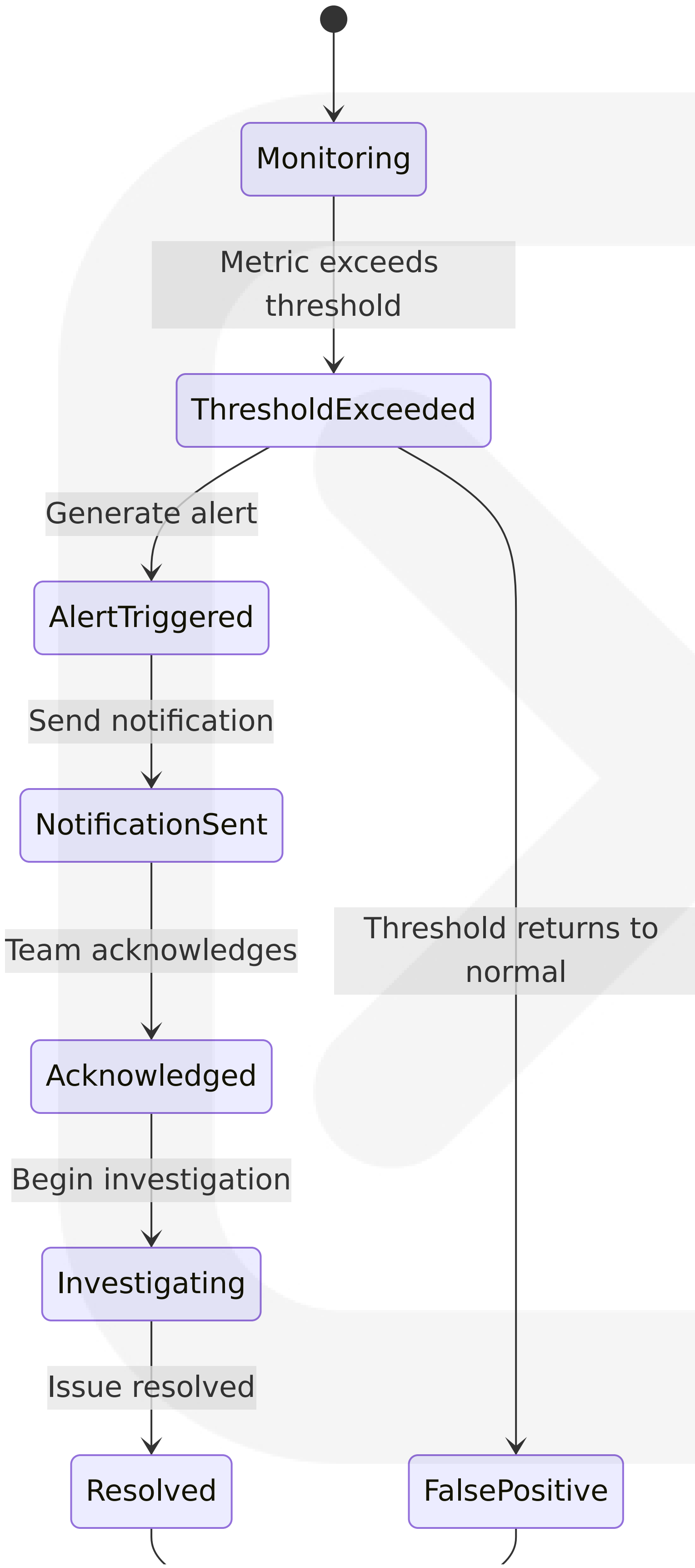
Proactive Alert Configuration

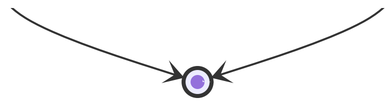
The alert management system focuses on critical business metrics and system health indicators relevant to blockchain applications.

Alert Configuration Matrix:

Alert Type	Trigger Condition	Severity	Response Time
Wallet Connection Failures	>10% failure rate	High	5 minutes
Transaction Processing Errors	>5% error rate	Critical	2 minutes
API Response Degradation	>2 second average	Medium	10 minutes
Blockchain Network Issues	RPC endpoint failures	High	5 minutes

Alert Management Flow:


Monitoring
Metric exceeds threshold
ThresholdExceeded
Generate alert
AlertTriggered
Send notification
NotificationSent
Team acknowledges
Acknowledged
Begin investigation
Investigating
Issue resolved
Resolved
Threshold returns to normal
FalsePositive



6.5.1.5 Dashboard Design

Unified Monitoring Dashboard

Granular insights into application performance to ensure you're always up and running. Providing comprehensive insights into your website's visitors, tracking topic pages, referrers, and demographics like location, operating systems, and browser info.

Dashboard Component Architecture:

Dashboard Section	Metrics Displayed	Update Frequency	Data Source
System Health	Uptime, response times, error rates	Real-time	Vercel + Custom
Blockchain Status	Transaction success, network health	30 seconds	Solana RPC
User Experience	Page load times, wallet connections	Real-time	Frontend analytics
Business Metrics	Token creations, payment volume	5 minutes	Application logs

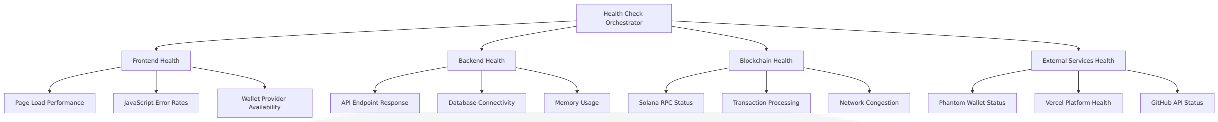
6.5.2 OBSERVABILITY PATTERNS

6.5.2.1 Health Checks Implementation

Multi-Layer Health Monitoring

The health check system implements comprehensive monitoring across all system components with automated recovery mechanisms.

Health Check Architecture:



Health Check Specifications:

Component	Check Type	Frequency	Timeout	Recovery Action
Frontend Application	Synthetic monitoring	60 seconds	10 seconds	Automatic restart
Backend API	Endpoint ping	30 seconds	5 seconds	Container restart
Solana Network	RPC connectivity	15 seconds	3 seconds	Fallback endpoint
Phantom Wallet	Provider detection	On user action	2 seconds	User notification

6.5.2.2 Performance Metrics Tracking

Comprehensive Performance Monitoring

Monitoring Transaction Status: Once a transaction is sent, it's important to monitor its status. Developers can use the Web3.js library to check if the transaction is confirmed or if it has failed, providing transparency to users.

Performance Metrics Framework:

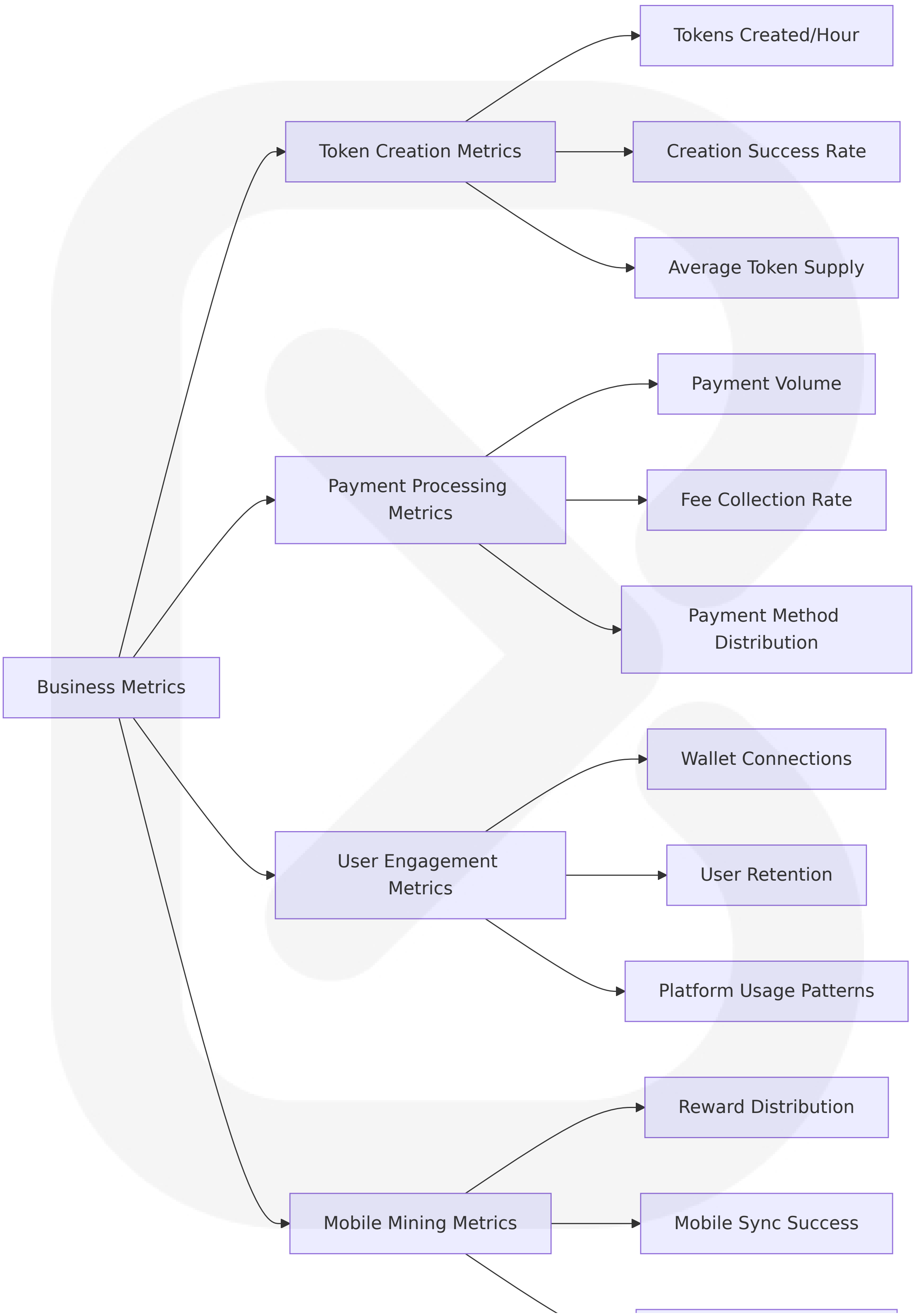
Metric Category	Key Indicators	Target Values	Monitoring Method
Frontend Performance	LCP, FID, CLS	<2.5s, <100ms, <0.1	Core Web Vitals
API Performance	Response time, throughput	<500ms, 100+ RPS	Express middleware
Blockchain Performance	Confirmation time, success rate	<10s, >95%	Transaction monitoring
User Experience	Wallet connection time, error rate	<3s, <5%	Custom tracking

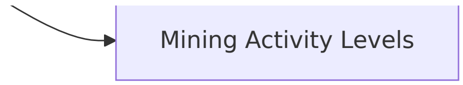
6.5.2.3 Business Metrics Monitoring

Token Launchpad Specific Metrics

The business metrics focus on platform-specific KPIs that directly impact TeosPump's success and user satisfaction.

Business Metrics Dashboard:


Business Metrics
Token Creation Metrics
Tokens Created/Hour
Creation Success Rate
Average Token Supply
Payment Processing Metrics
Payment Volume
Fee Collection Rate
Payment Method Distribution
User Engagement Metrics
Wallet Connections
User Retention
Platform Usage Patterns
Mobile Mining Metrics
Reward Distribution
Mobile Sync Success



6.5.2.4 SLA Monitoring Framework

Service Level Agreement Tracking

The SLA monitoring framework ensures platform reliability and performance commitments are met consistently.

SLA Metrics and Targets:

Service Component	Availability Target	Performance Target	Error Rate Target
Frontend Application	99.9% uptime	<2s page load	<1% error rate
Backend API	99.5% uptime	<500ms response	<2% error rate
Token Creation Service	99% success rate	<30s completion	<5% failure rate
Payment Processing	99.9% success rate	<10s confirmation	<0.1% failure rate

6.5.2.5 Capacity Tracking System

Resource Utilization Monitoring

The Solana network's performance has continued to improve through the past six months (Sept. 1, 2023 - Feb. 29, 2024), as measured by uptime, the ratio of non-voting-to-voting transactions, time to produce a block, and average and max transactions per second. The Solana network has had 99.94% uptime in the 12 month period previous to the publishing of this report

Capacity Monitoring Implementation:

Resource Type	Monitoring Metric	Scaling Threshold	Scaling Action
Frontend Instances	Request volume	>1000 concurrent users	Auto-scale functions
Backend Containers	CPU/Memory usage	>80% utilization	Horizontal scaling
Database Connections	Connection pool usage	>70% utilization	Increase pool size
Blockchain RPC	Request rate	>100 RPS per endpoint	Add RPC endpoints

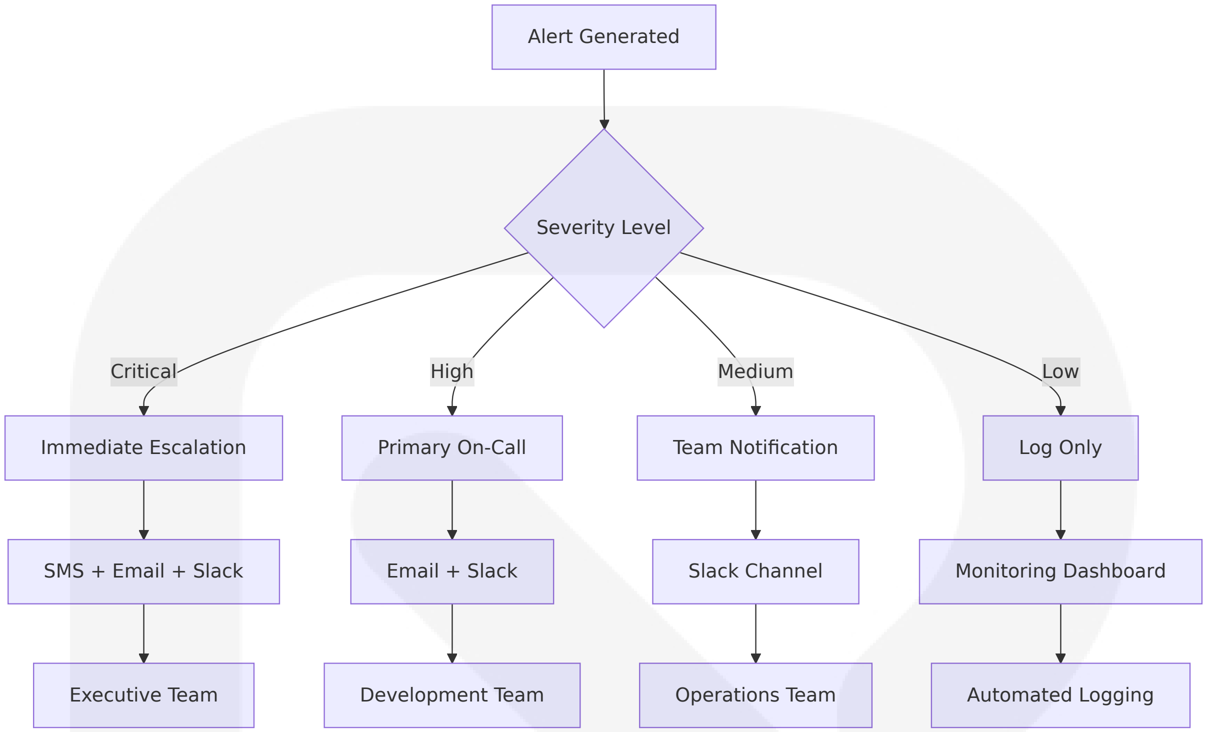
6.5.3 INCIDENT RESPONSE

6.5.3.1 Alert Routing Configuration

Intelligent Alert Distribution

The alert routing system ensures critical issues reach the appropriate team members based on severity and component affected.

Alert Routing Matrix:



Alert Routing Specifications:

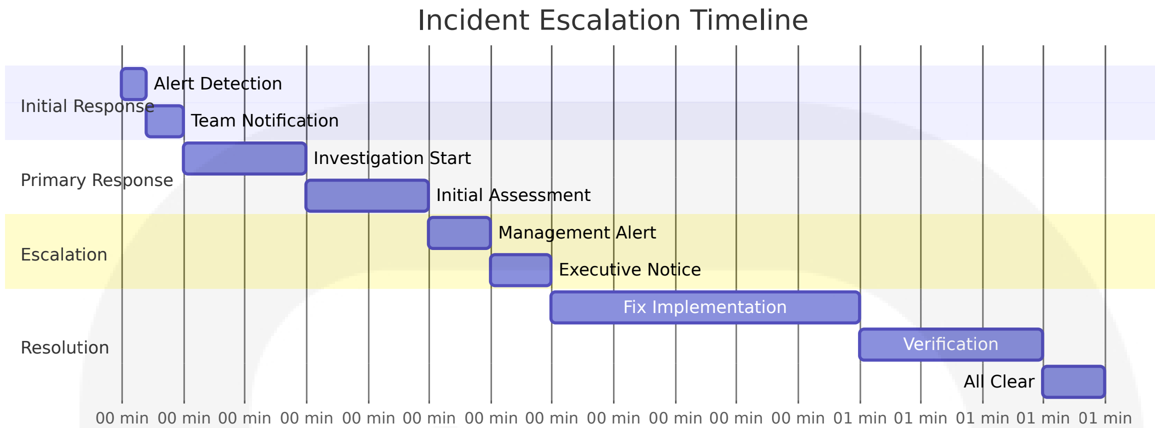
Alert Category	Primary Contact	Secondary Contact	Escalation Time
Blockchain Network Issues	Blockchain Engineer	Lead Developer	15 minutes
Payment Processing Failures	Backend Engineer	Product Manager	10 minutes
Frontend Application Errors	Frontend Engineer	DevOps Engineer	20 minutes
Security Incidents	Security Lead	CTO	5 minutes

6.5.3.2 Escalation Procedures

Structured Incident Escalation

The escalation procedures ensure rapid response to critical issues while maintaining appropriate communication channels.

Escalation Timeline:



6.5.3.3 Runbook Documentation

Standardized Response Procedures

Comprehensive runbooks provide step-by-step procedures for common incident scenarios specific to blockchain applications.

Runbook Categories:

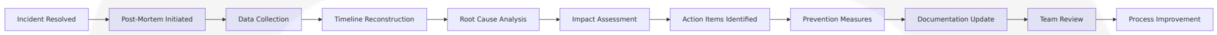
Incident Type	Response Time	Key Actions	Recovery Procedures
Wallet Connection Failures	5 minutes	Check provider status, verify network	Restart services, user communication
Transaction Processing Errors	10 minutes	Analyze blockchain status, check RPC	Switch RPC endpoints, retry logic
Payment System Failures	5 minutes	Verify $TEOS contract, check balances	Manual verification, refund process
API Performance Degradation	15 minutes	Check resource usage, analyze logs	Scale resources, optimize queries

6.5.3.4 Post-Mortem Process

Continuous Improvement Framework

The post-mortem process ensures learning from incidents and implementing preventive measures for future occurrences.

Post-Mortem Workflow:



6.5.3.5 Improvement Tracking

Systematic Enhancement Monitoring

The improvement tracking system monitors the effectiveness of implemented changes and identifies areas for continued optimization.

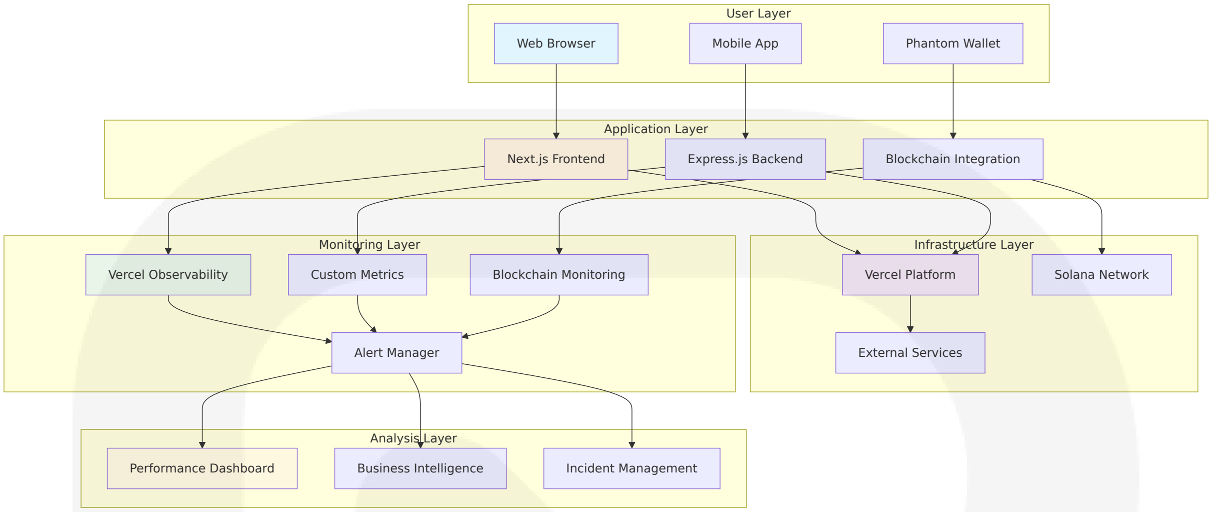
Improvement Metrics:

Improvement Area	Success Metric	Measurement Period	Target Improvement
Mean Time to Detection	Alert response time	Monthly	20% reduction
Mean Time to Resolution	Incident duration	Monthly	30% reduction
Incident Recurrence	Repeat incidents	Quarterly	50% reduction
System Reliability	Uptime percentage	Monthly	0.1% improvement

6.5.4 MONITORING ARCHITECTURE DIAGRAMS

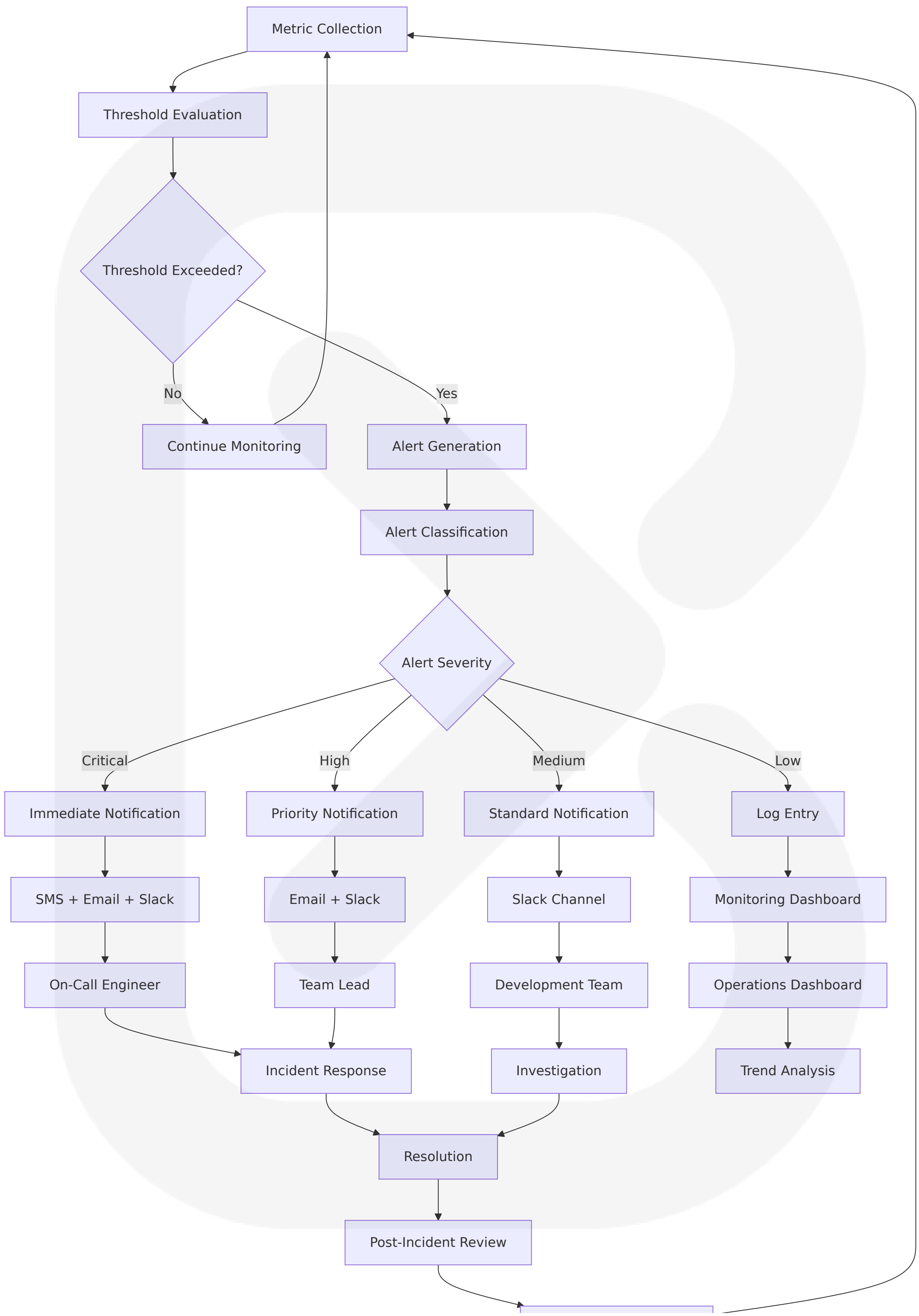
6.5.4.1 Comprehensive Monitoring Architecture

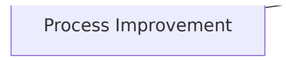
End-to-End Monitoring System



6.5.4.2 Alert Flow Architecture

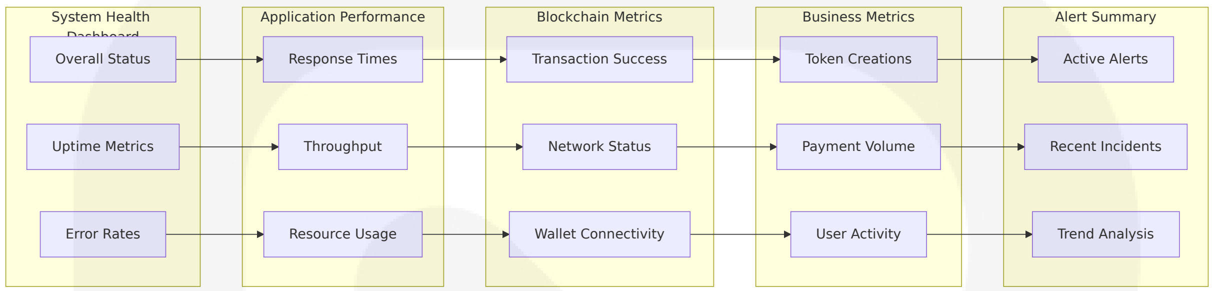
Intelligent Alert Processing System


Metric Collection
Threshold Evaluation
Threshold Exceeded?
No
Yes
Continue Monitoring
Alert Generation
Alert Classification
Alert Severity
Critical
High
Medium
Low
Immediate Notification
Priority Notification
Standard Notification
Log Entry
SMS + Email + Slack
Email + Slack
Slack Channel
Monitoring Dashboard
On-Call Engineer
Team Lead
Development Team
Operations Dashboard
Incident Response
Investigation
Trend Analysis
Resolution
Post-Incident Review



6.5.4.3 Performance Monitoring Dashboard Layout

Real-Time Performance Visualization



6.5.5 MONITORING IMPLEMENTATION STRATEGY

6.5.5.1 Phased Implementation Approach

Gradual Monitoring Deployment

The monitoring implementation follows a phased approach to ensure system stability while building comprehensive observability capabilities.

Implementation Phases:

Phase	Duration	Focus Area	Key Deliverables
Phase 1	2 weeks	Basic health checks	Uptime monitoring, error tracking
Phase 2	3 weeks	Performance monitoring	Response time tracking, resource monitoring
Phase 3	4 weeks	Business metrics	Token creation tracking, payment monitoring
Phase 4	2 weeks	Advanced alerting	Intelligent routing, escalation procedures

6.5.5.2 Tool Integration Strategy

Monitoring Tool Ecosystem

With OpenTelemetry (OTEL), send traces from your functions to application performance monitoring (APM) vendors.

Tool Integration Matrix:

Tool Category	Primary Tool	Integration Method	Data Flow
Frontend Monitoring	Vercel Observability	Built-in integration	Automatic collection
Backend Monitoring	Winston + Custom	Express middleware	Structured logging
Blockchain Monitoring	Custom RPC monitoring	Direct integration	Real-time polling
Alert Management	Custom + Slack	Webhook integration	Event-driven notifications

6.5.5.3 Cost Optimization Strategy

Efficient Monitoring Resource Usage

The monitoring strategy balances comprehensive coverage with cost efficiency, leveraging built-in platform capabilities where possible.

Cost Optimization Measures:

Optimization Area	Strategy	Expected Savings	Implementation
Data Retention	Tiered storage strategy	40% reduction	Automated archiving
Alert Frequency	Intelligent deduplication	60% reduction	Smart filtering

Optimization Area	Strategy	Expected Savings	Implementation
Metric Collection	Selective monitoring	30% reduction	Priority-based collection
Dashboard Efficiency	Optimized queries	25% reduction	Query optimization

This comprehensive monitoring and observability framework ensures TeosPump operates reliably while providing the insights necessary for continuous improvement and rapid incident response. The architecture leverages modern observability patterns while remaining cost-effective and maintainable for a blockchain-native application.

6.6 TESTING STRATEGY

6.6.1 TESTING APPROACH

6.6.1.1 Unit Testing

Testing Frameworks and Tools

TeosPump implements a comprehensive unit testing strategy leveraging modern JavaScript testing frameworks optimized for blockchain applications. Jest and React Testing Library are frequently used together for Unit Testing and Snapshot Testing. Since the release of Next.js 12, Next.js now has built-in configuration for Jest.

Testing Framework Selection:

Framework	Version	Purpose	Justification
Jest	29.7.0+	Primary testing framework	Jest is an amazing tool to test React apps. And since the release of Next.js v12, Next.js has a built-in Jest configuration, which

Framework	Version	Purpose	Justification
			saves you valuable time for setting up and writing tests.
React Testing Library	16.0.0+	Frontend component testing	React Testing Library, a widely-used test suite for React applications, allows developers to test React components by simulating user behaviors and interactions.
Supertest	Latest	API endpoint testing	We'll use Jest as our testing framework and Supertest for making HTTP assertions on Express routes.
ts-jest	Latest	TypeScript support	ts-jest: A TypeScript preprocessor for Jest, allowing you to write tests in TypeScript.

Test Organization Structure

```
teospump/
├── __tests__/
│   ├── components/
│   │   ├── WalletConnection.test.tsx
│   │   ├── TokenForm.test.tsx
│   │   └── PaymentSelector.test.tsx
│   ├── utils/
│   │   ├── solana.test.ts
│   │   ├── validation.test.ts
│   │   └── constants.test.ts
│   ├── pages/
│   │   ├── index.test.tsx
│   │   └── create-token.test.tsx
│   └── backend/
│       ├── routes/
│       │   ├── mobile.test.ts
│       │   └── token.test.ts
│       └── utils/
│           └── solana.test.ts
├── jest.config.js
```

```
├── jest.setup.js
└── __mocks__/
    ├── @solana/
    │   └── web3.js
    └── phantom-wallet.js
```

Mocking Strategy

The testing strategy implements comprehensive mocking for external dependencies to ensure unit test isolation:

Dependency Type	Mocking Approach	Implementation	Rationale
Phantom Wallet	Mock provider object	Custom mock with wallet methods	Since unit tests exercise isolated units, that means the system under test can't interact with external dependencies, such as databases, the file system, or HTTP services. During tests, such dependencies are replaced by "fake" implementations, such as mocks/stubs/spies, which are collectively known as test doubles.
Solana RPC	Mock connection and responses	Jest mock functions	Blockchain network independence
Express APIs	Supertest mock server	In-memory test server	API endpoint isolation
File System	Mock file operations	Jest mock filesystem	No actual file I/O during tests

Code Coverage Requirements

Code coverage configuration includes collecting coverage from all TypeScript and JavaScript files while excluding node_modules, build outputs, and configuration files.

Coverage Targets:

Component Type	Coverage Target	Measurement Method	Exclusions
Frontend Components	85% line coverage	Jest coverage reports	Third-party libraries
Backend APIs	90% line coverage	Supertest integration	External service calls
Utility Functions	95% line coverage	Unit test coverage	Configuration files
Blockchain Integration	80% line coverage	Mock-based testing	Network-dependent code

Test Naming Conventions

```
// Component Testing Convention
describe('WalletConnection Component', () => {
  describe('when wallet is not connected', () => {
    it('should display connection button', () => {
      // Test implementation
    });

    it('should handle connection errors gracefully', () => {
      // Test implementation
    });
  });

  describe('when wallet is connected', () => {
    it('should display wallet address', () => {
      // Test implementation
    });
  });
});

// API Testing Convention
```

```
describe('POST /api/token/create', () => {
  describe('with valid token parameters', () => {
    it('should create token successfully', async () => {
      // Test implementation
    });
  });

  describe('with invalid parameters', () => {
    it('should return validation errors', async () => {
      // Test implementation
    });
  });
});
```

Test Data Management

Test data management follows a factory pattern approach for consistent and maintainable test data:

```
// Test Data Factories
export const createMockWallet = () => ({
  publicKey: new PublicKey('Akvm3CbDN448fyD8qmQjowgBGpcYZtjuKFL4xT8PZhbF
  connected: true,
  signTransaction: jest.fn(),
  connect: jest.fn(),
  disconnect: jest.fn()
});

export const createMockTokenData = () => ({
  name: 'Test Token',
  symbol: 'TEST',
  decimals: 9,
  supply: 1000000,
  metadata: {
    description: 'Test token for unit testing'
  }
});
```

6.6.1.2 Integration Testing

Service Integration Test Approach

Integration testing focuses on verifying the interaction between TeosPump's core components, particularly the blockchain integration layer and backend API services. Integration Tests will assess the coherence of the entire application, ensuring smooth interactions between various parts. Just like a knight protecting the kingdom, Integration Tests guard against unforeseen conflicts.

Integration Test Categories:

Integration Type	Scope	Testing Method	Tools Used
Frontend-Blockchain	Wallet and SPL token operations	Mock blockchain responses	Jest + @solana/web3.js mocks
Backend-Database	API data persistence	In-memory database	Jest + SQLite memory
API-Mobile Sync	Mobile reward distribution	HTTP integration tests	Supertest + Express
End-to-End Flows	Complete user journeys	Automated browser testing	Playwright

API Testing Strategy

Unit testing controllers, services, and middleware helps ensure your API behaves as expected. With Jest and Supertest, you can quickly test API endpoints, while mocking dependencies isolates components to test individual logic.

API Integration Test Structure:

```
describe('Token Creation API Integration', () => {
  let app: Express;
  let mockSolanaConnection: jest.Mocked<Connection>;

  beforeEach(() => {
    app = createTestApp();
```

```
    mockSolanaConnection = createMockConnection();
  });

  describe('POST /api/token/create', () => {
    it('should integrate wallet verification with token minting', async
      const tokenData = createMockTokenData();
      const walletSignature = createMockSignature();

      const response = await request(app)
        .post('/api/token/create')
        .send({ tokenData, signature: walletSignature })
        .expect(201);

      expect(response.body).toHaveProperty('mintAddress');
      expect(mockSolanaConnection.sendTransaction).toHaveBeenCalled();
    });
  });
});
```

Database Integration Testing

Testing on the Solana Devnet is crucial for ensuring that your application functions correctly before deploying it to the mainnet. The Devnet provides a safe environment to experiment with your code without the risk of losing real assets.

Database Test Configuration:

Environment	Database Type	Purpose	Data Management
Unit Tests	In-memory SQLite	Isolated component testing	Fresh database per test
Integration Tests	PostgreSQL test container	Multi-component testing	Transaction rollback
E2E Tests	Solana Devnet	Full blockchain integration	Test account management

External Service Mocking

The integration testing strategy implements sophisticated mocking for external blockchain services while maintaining realistic interaction patterns:

```
// Solana Network Mock
const createSolanaNetworkMock = () => ({
  connection: {
    getBalance: jest.fn().mockResolvedValue(1000000000), // 1 SOL
    sendTransaction: jest.fn().mockResolvedValue('mock-signature'),
    confirmTransaction: jest.fn().mockResolvedValue({ value: { err: null
  },
  tokenProgram: {
    createMint: jest.fn().mockResolvedValue('mock-mint-address'),
    mintTo: jest.fn().mockResolvedValue('mock-mint-signature')
  }
});
```

Test Environment Management

Test environments are managed through configuration-driven setup with automatic cleanup:

Environment Variable	Test Value	Production Value	Purpose
NODE_ENV	test	production	Environment detection
SOLANA_NETWORK	devnet	mainnet-beta	Blockchain network
DATABASE_URL	:memory:	postgres://...	Database connection
PHANTOM_MOCK	true	false	Wallet mocking control

6.6.1.3 End-to-End Testing

E2E Test Scenarios

End-to-end testing validates complete user workflows from wallet connection through token creation and mobile reward distribution. Playwright and Cypress are two frameworks both closely associated with end-to-end testing of production websites. Both frameworks can do quite a bit more than 'making sure nothing on your site is broken' and their design philosophies, architectures, and use cases are different.

E2E Testing Framework Selection:

Based on current trends and capabilities, Starting in mid-2024, Playwright surpassed Cypress in npm downloads, indicating that more projects are starting with Playwright as their preferred automation framework. TeosPump adopts Playwright for comprehensive end-to-end testing.

Critical E2E Test Scenarios:

Scenario	User Journey	Success Criteria	Test Complexity
Complete Token Creation	Wallet connect → Form fill → Payment → Token mint	Token created with correct metadata	High
Payment Flow Validation	Select payment method → Approve transaction → Confirm payment	Payment processed successfully	Medium
Mobile Reward Distribution	Token creation triggers → Backend processes → Mobile sync	Rewards distributed to miners	High
Error Handling	Invalid inputs → Network failures → Recovery	Graceful error handling	Medium

UI Automation Approach

Playwright uses standard async/await syntax, providing a clear, modern JavaScript interface. Cypress uses a custom dot notation that simplifies code but is not fully asynchronous, limiting flexibility.

Playwright Test Implementation:

```
import { test, expect } from '@playwright/test';

test.describe('TeosPump Token Creation Flow', () => {
  test('should complete full token creation workflow', async ({ page }) =
    // Navigate to platform
    await page.goto('/');

    // Connect Phantom wallet (mocked in test environment)
    await page.click('[data-testid="connect-wallet"]');
    await expect(page.locator('[data-testid="wallet-connected"]')).toBeV

    // Navigate to token creation
    await page.click('[data-testid="create-token"]');

    // Fill token form
    await page.fill('[data-testid="token-name"]', 'Test Token');
    await page.fill('[data-testid="token-symbol"]', 'TEST');
    await page.fill('[data-testid="token-supply"]', '1000000');

    // Select payment method
    await page.click('[data-testid="payment-teos"]');

    // Submit and confirm
    await page.click('[data-testid="create-token-submit"]');
    await page.click('[data-testid="confirm-payment"]');

    // Verify success
    await expect(page.locator('[data-testid="token-created"]')).toBeVisil
    await expect(page.locator('[data-testid="mint-address"]')).toContain
  });
});
```

Test Data Setup/Teardown

E2E tests require careful management of blockchain state and test data:

```
// Test Setup
test.beforeEach(async ({ page }) => {
  // Setup test wallet with sufficient balance
  await setupTestWallet();
```

```
  // Configure Solana devnet connection
  await configureSolanaDevnet();

  // Initialize mock Phantom wallet
  await page.addInitScript(() => {
    window.phantom = createMockPhantomWallet();
  });
});

// Test Cleanup
test.afterEach(async () => {
  // Clean up test tokens
  await cleanupTestTokens();

  // Reset wallet state
  await resetTestWallet();
});
```

Performance Testing Requirements

Performance testing ensures the platform meets user experience expectations under various load conditions:

Performance Metric	Target	Measurement Method	Test Scenario
Page Load Time	<2 seconds	Lighthouse CI	Initial platform access
Wallet Connection	<3 seconds	Custom timing	Phantom wallet integration
Token Creation	<30 seconds	End-to-end timing	Complete creation flow
API Response Time	<500ms	Load testing	Backend endpoint performance

Cross-Browser Testing Strategy

Playwright supports all modern browsers (Chromium, Firefox, WebKit) and can run on multiple operating systems with little configuration. Choose

Playwright if you need cross-browser support, parallelism, and multi-language flexibility. Its architecture is more scalable for complex, multi-layered applications that demand realistic, real-world testing.

Browser Test Matrix:

Browser	Version	Platform	Test Coverage
Chromium	Latest	Windows, macOS, Linux	Full test suite
Firefox	Latest	Windows, macOS, Linux	Core functionality
WebKit	Latest	macOS	Safari compatibility
Mobile Chrome	Latest	Android emulation	Mobile responsiveness

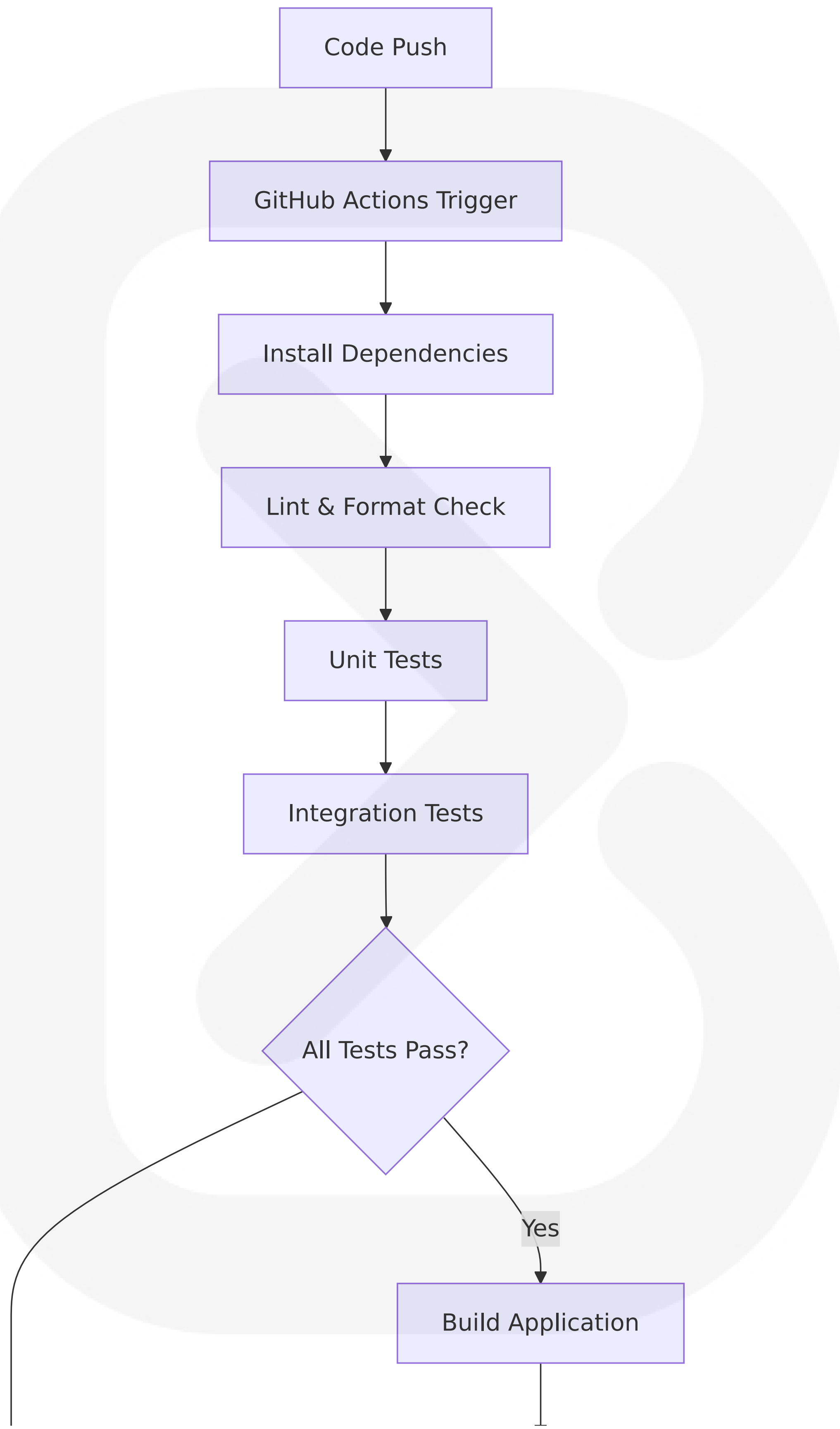
6.6.2 TEST AUTOMATION

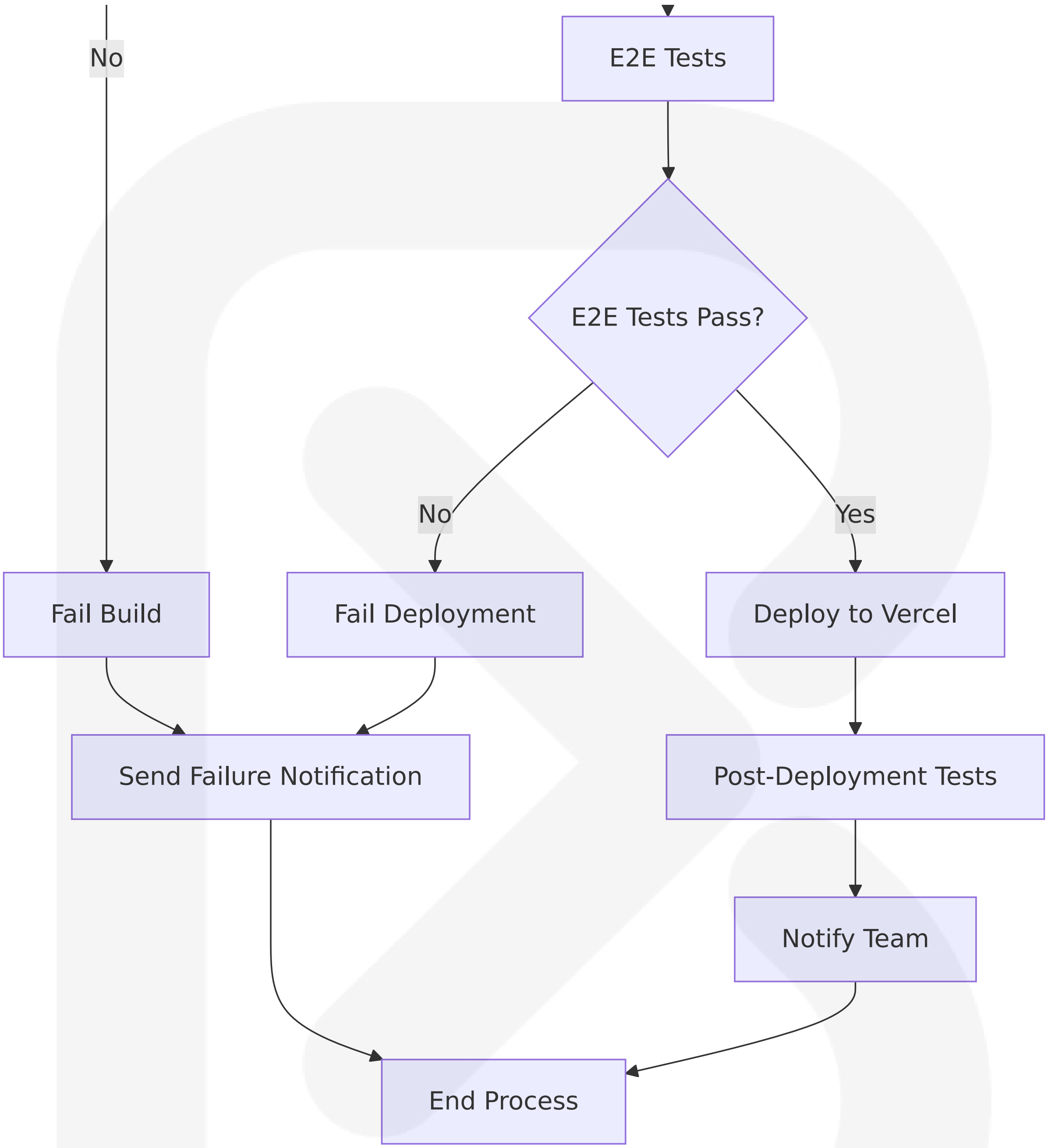
6.6.2.1 CI/CD Integration

Automated Test Triggers

The CI/CD pipeline integrates comprehensive testing at multiple stages to ensure code quality and deployment reliability. GitHub Actions CI/CD service for automating workflows including building, testing, and deploying code with automatic triggers on repository changes

CI/CD Test Integration Strategy:


Code Push
GitHub Actions Trigger
Install Dependencies
Lint & Format Check
Unit Tests
Integration Tests
All Tests Pass?
Yes
Build Application



GitHub Actions Workflow Configuration:

Stage	Trigger	Tests Executed	Failure Action
Pre-commit	Git hooks	Lint, format, type check	Block commit
Pull Request	PR creation/update	Unit + Integration tests	Block merge

Stage	Trigger	Tests Executed	Failure Action
Main Branch	Merge to main	Full test suite + E2E	Block deployment
Deployment	Vercel deploy	Smoke tests	Rollback

Parallel Test Execution

Playwright's out-of-the-box support for parallelism and sharding can have multiplicative effect in time savings. If we provide a process-based parallelism of 4 and shard the tests in 4 machines, then 16 tests are run concurrently, which reduces the execution time dramatically.

Test Parallelization Strategy:

```yaml
# GitHub Actions Matrix Strategy
strategy:
  matrix:
    node-version: [18.x, 20.x]
    test-group: [unit, integration, e2e-chrome, e2e-firefox]
    include:
      - test-group: unit
        test-command: npm run test:unit
      - test-group: integration
        test-command: npm run test:integration
      - test-group: e2e-chrome
        test-command: npm run test:e2e -- --project=chromium
      - test-group: e2e-firefox
        test-command: npm run test:e2e -- --project=firefox
```

Test Reporting Requirements

Comprehensive test reporting provides visibility into test results, coverage metrics, and performance trends:

Report Type	Tool	Frequency	Audience
Unit Test Coverage	Jest Coverage	Every commit	Developers
Integration Test Results	Jest + Supertest	Every PR	QA Team
E2E Test Reports	Playwright HTML Reporter	Every deployment	Product Team
Performance Metrics	Lighthouse CI	Daily	Operations Team

Failed Test Handling

The system implements intelligent failed test handling with automatic retry mechanisms and detailed failure analysis:

```
// Jest Retry Configuration
module.exports = {
  testEnvironment: 'node',
  setupFilesAfterEnv: ['<rootDir>/jest.setup.js'],
  testTimeout: 30000,
  retry: {
    unit: 0,        // No retries for unit tests
    integration: 1, // One retry for integration tests
    e2e: 2         // Two retries for E2E tests
  }
};

// Playwright Retry Configuration
export default defineConfig({
  retries: process.env.CI ? 2 : 0,
  workers: process.env.CI ? 1 : undefined,
  reporter: [
    ['html'],
    ['junit', { outputFile: 'test-results/junit.xml' }]
  ]
});
```

Flaky Test Management

Auto-waiting Mechanism: It waits for elements to be actionable prior to executing interactions, which reduces flakiness in tests. Automatic Waiting: Cypress automatically waits for commands and assertions before moving on, reducing the likelihood of flaky tests.

Flaky Test Mitigation Strategies:

Flakiness Source	Mitigation Strategy	Implementation	Monitoring
Network Timeouts	Increased timeouts + retries	Playwright auto-wait	Test duration tracking
Blockchain Delays	Mock responses in CI	Deterministic test data	Success rate monitoring
Race Conditions	Explicit wait conditions	Element visibility checks	Failure pattern analysis
Resource Contention	Test isolation	Fresh test environments	Resource usage metrics

6.6.3 QUALITY METRICS

6.6.3.1 Code Coverage Targets

Coverage Requirements by Component

The testing strategy establishes comprehensive code coverage targets that balance thorough testing with development velocity:

Component Category	Line Coverage	Branch Coverage	Function Coverage	Statement Coverage
Frontend Components	85%	80%	90%	85%
Backend APIs	90%	85%	95%	90%
Utility Functions	95%	90%	100%	95%

Component Category	Line Coverage	Branch Coverage	Function Coverage	Statement Coverage
Blockchain Integration	80%	75%	85%	80%

Coverage Exclusions:

- Third-party library integrations
- Configuration files and constants
- Type definition files
- Development-only utilities
- Generated code and build artifacts

Test Success Rate Requirements

Unit testing helps improve the quality of your code and reduces the amount of time and money you spend on bug fixing. Moreover, unit testing helps you find bugs early on in the development life cycle and increases your confidence in the code.

Success Rate Targets:

Test Category	Success Rate Target	Measurement Period	Alert Threshold
Unit Tests	99.5%	Per commit	<98%
Integration Tests	98%	Per pull request	<95%
E2E Tests	95%	Per deployment	<90%
Performance Tests	90%	Daily	<85%

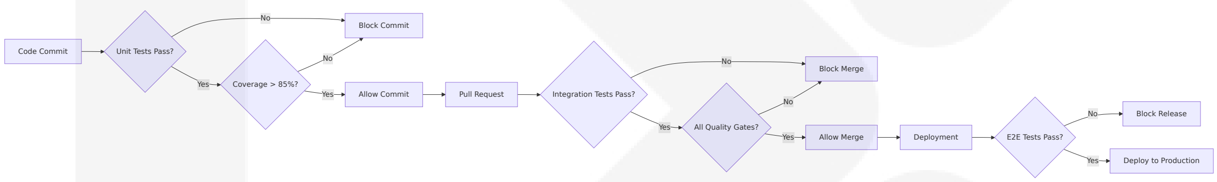
Performance Test Thresholds

Performance testing ensures the platform maintains acceptable response times under various load conditions:

Performance Metric	Threshold	Test Environment	Measurement Method
Frontend Load Time	<2 seconds	Production-like	Lighthouse CI
API Response Time	<500ms	Load testing	Artillery.js
Wallet Connection	<3 seconds	E2E testing	Playwright timing
Token Creation Flow	<30 seconds	End-to-end	Complete workflow timing

Quality Gates

Quality gates enforce minimum standards before code progression through the development pipeline:



Documentation Requirements

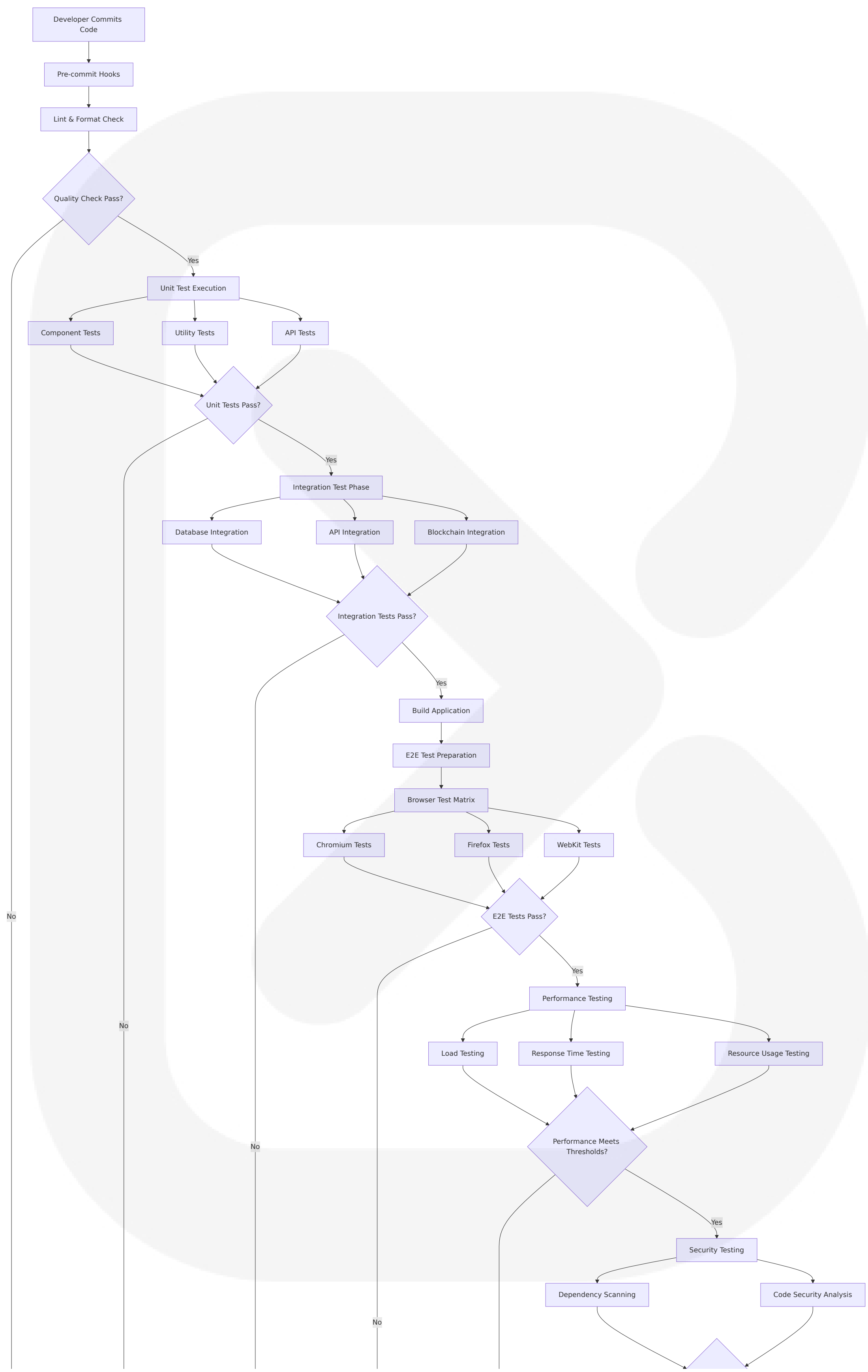
Testing documentation ensures knowledge transfer and maintains testing standards:

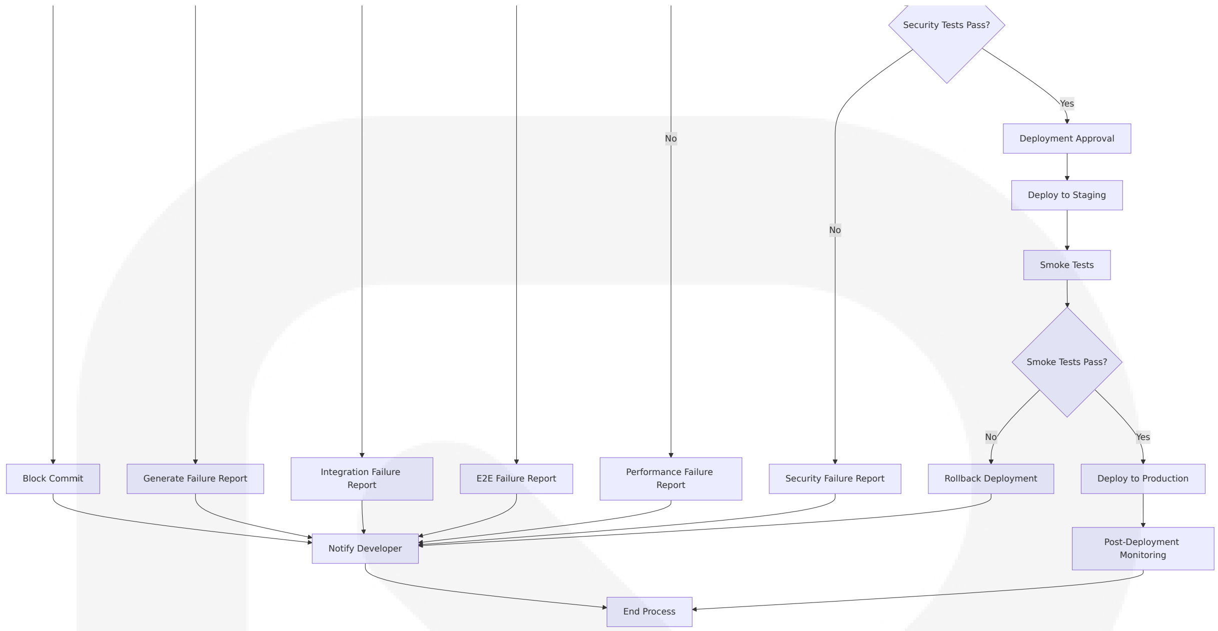
Documentation Type	Content Requirements	Update Frequency	Owner
Test Plan	Strategy, scope, approach	Per release	QA Lead
Test Cases	Scenarios, steps, expected results	Per feature	Developers
Coverage Reports	Metrics, trends, gaps	Per build	Automated
Performance Baselines	Benchmarks, thresholds	Monthly	DevOps Team

6.6.4 TEST EXECUTION FLOW

6.6.4.1 Test Execution Architecture

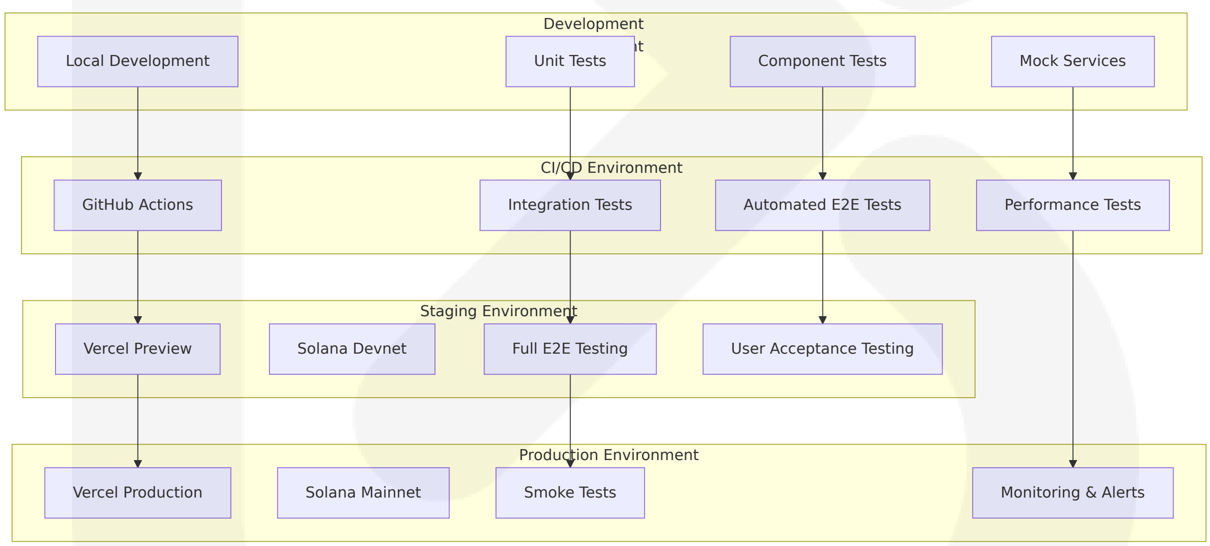
Comprehensive Test Execution Pipeline


Developer Commits Code
Pre-commit Hooks
Lint & Format Check
Quality Check Pass?
Yes
No
Unit Test Execution
Component Tests
Utility Tests
API Tests
Unit Tests Pass?
Yes
No
Integration Test Phase
Database Integration
API Integration
Blockchain Integration
Integration Tests Pass?
Yes
No
Build Application
E2E Test Preparation
Browser Test Matrix
Chromium Tests
Firefox Tests
WebKit Tests
E2E Tests Pass?
Yes
No
Performance Testing
Load Testing
Response Time Testing
Resource Usage Testing
Performance Meets Thresholds?
Yes
Security Testing
Dependency Scanning
Code Security Analysis



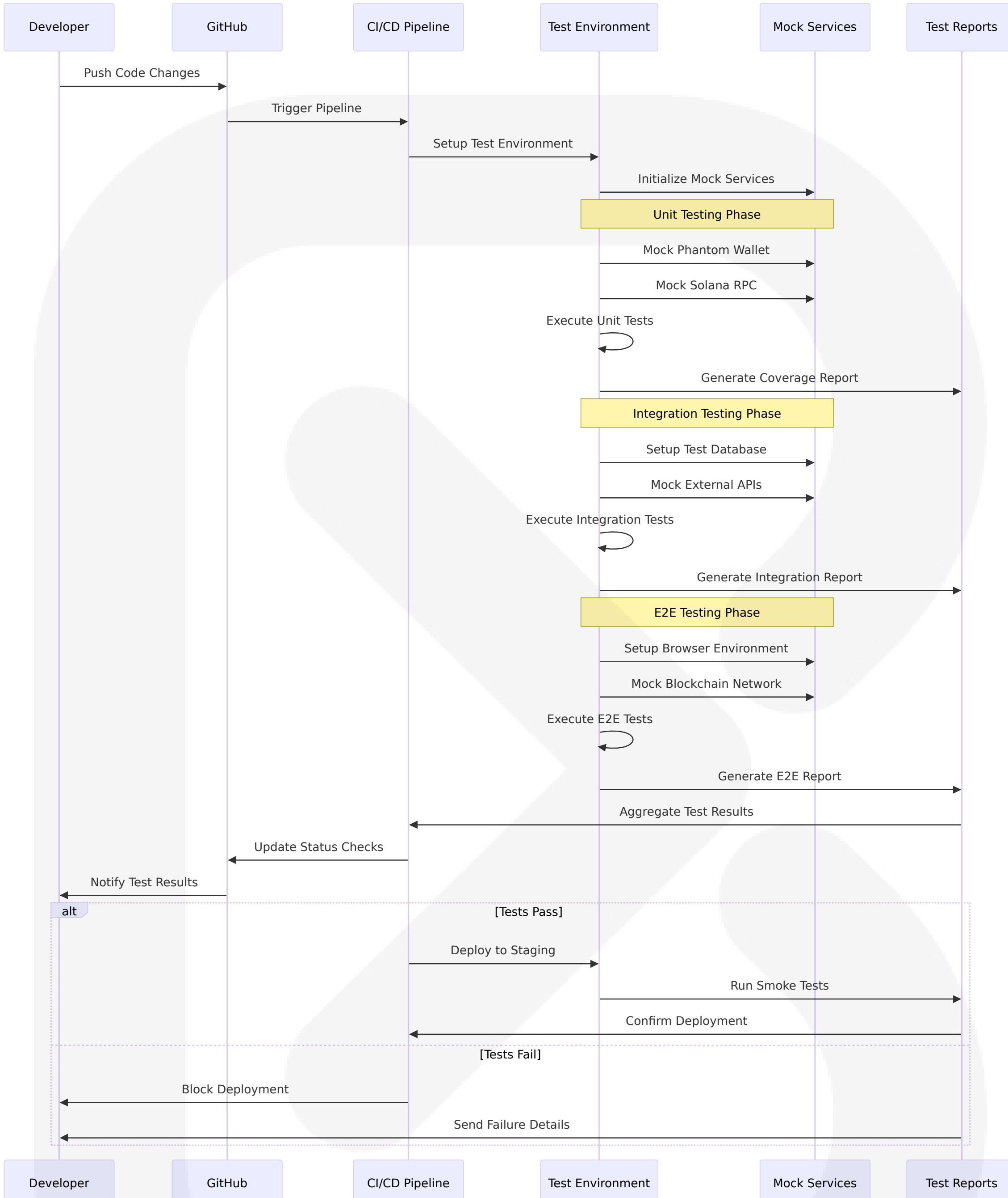
6.6.4.2 Test Environment Architecture

Multi-Environment Testing Strategy

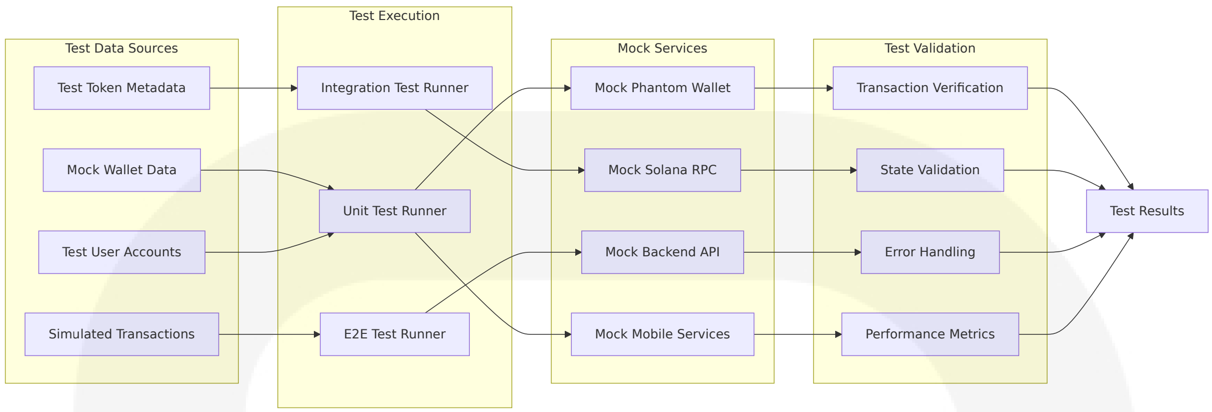


6.6.4.3 Test Data Flow Diagrams

Test Data Management and Flow



Blockchain Testing Data Flow



6.6.5 SPECIALIZED TESTING CONSIDERATIONS

6.6.5.1 Blockchain-Specific Testing

SPL Token Testing Strategy

Creating tokens and accounts requires SOL for account rent deposits and transaction fees. If the cluster you are targeting offers a faucet, you can get a little SOL for testing

Blockchain Testing Approach:

Test Category	Environment	Purpose	Implementation
Unit Tests	Mock blockchain	Isolated component testing	Jest mocks for @solana/web3.js
Integration Tests	Solana Devnet	Real blockchain interaction	Set Up Your Environment: Install the Solana CLI and configure your CLI to use the Devnet
Contract Tests	Local validator	Smart contract validation	Solana test validator

Test Category	Environment	Purpose	Implementation
Performance Tests	Mainnet fork	Production-like testing	Anchor testing framework

Wallet Integration Testing

Testing wallet integration requires sophisticated mocking strategies to simulate various wallet states and user interactions:

```
// Phantom Wallet Mock for Testing
const createPhantomWalletMock = () => ({
  isPhantom: true,
  publicKey: new PublicKey('Akvm3CbDN448fyD8qmQjowgBGpcYZtjuKFL4xT8PZhbF
  isConnected: true,
  connect: jest.fn().mockResolvedValue({ publicKey: 'mock-key' }),
  disconnect: jest.fn().mockResolvedValue({}),
  signTransaction: jest.fn().mockImplementation((tx) => Promise.resolve(
  signAllTransactions: jest.fn().mockImplementation((txs) => Promise.res
});
```

6.6.5.2 Security Testing Requirements

Comprehensive Security Testing Strategy

Security testing ensures the platform protects user assets and maintains the integrity of blockchain operations:

Security Test Type	Scope	Tools	Frequency
Dependency Scanning	npm packages	npm audit, Snyk	Every commit
Code Security Analysis	Source code	ESLint security rules	Every build
Wallet Security Testing	Transaction signing	Custom security tests	Every release

Security Test Type	Scope	Tools	Frequency
API Security Testing	Backend endpoints	OWASP ZAP	Weekly

Blockchain Security Considerations

Smart Contract Bugs: Errors in the smart contract code can lead to vulnerabilities or unintended behaviors. Conducting thorough testing and audits is essential.

Security Test Implementation:

```
describe('Security Tests', () => {
  describe('Transaction Security', () => {
    it('should validate transaction signatures', async () => {
      const invalidSignature = 'invalid-signature';
      const result = await validateTransactionSignature(invalidSignature
      expect(result.isValid).toBe(false);
    });

    it('should prevent unauthorized token minting', async () => {
      const unauthorizedWallet = createMockWallet({ authorized: false })
      await expect(
        mintToken(unauthorizedWallet, tokenParams)
      ).rejects.toThrow('Unauthorized mint authority');
    });
  });
});
```

6.6.5.3 Performance and Load Testing

Performance Testing Framework

While Cypress took 16.09 seconds to finish the execution, Playwright took only 1.82 seconds. This is an improvement of 88.68%! Here, the execution time combines the time taken for setup and the time to complete the test.

This is the actual time that matters because this is the time an engineer has to wait until they see the final test result.

Performance Test Categories:

Test Type	Metric	Target	Tool
Frontend Performance	Page load time	<2 seconds	Lighthouse CI
API Performance	Response time	<500ms	Artillery.js
Blockchain Performance	Transaction confirmation	<10 seconds	Custom monitoring
Mobile Sync Performance	Sync completion	<2 seconds	Load testing

Load Testing Implementation:

```yaml
# Artillery.js Load Test Configuration
config:
  target: 'https://teospump.vercel.app'
  phases:
    - duration: 60
      arrivalRate: 10
    - duration: 120
      arrivalRate: 50
    - duration: 60
      arrivalRate: 100

scenarios:
  - name: "Token Creation Flow"
    weight: 70
    flow:
      - post:
          url: "/api/token/create"
          json:
            name: "Load Test Token"
            symbol: "LTT"
            supply: 1000000

  - name: "Mobile Sync"
```

```
    weight: 30
    flow:
      - get:
          url: "/api/mobile/sync"
```

This comprehensive testing strategy ensures TeosPump maintains high quality, security, and performance standards while supporting the unique requirements of blockchain application development. The strategy balances thorough testing coverage with development velocity, leveraging modern testing tools and practices optimized for the Solana ecosystem.

7. USER INTERFACE DESIGN

7.1 CORE UI TECHNOLOGIES

7.1.1 Frontend Technology Stack

Primary UI Framework

TeosPump leverages Next.js 14+ with TailwindCSS as the core UI technology stack. Tailwind CSS is a utility-first CSS framework that works exceptionally well with Next.js, providing a modern, responsive, and highly customizable user interface foundation.

Technology Integration Matrix:

Technology	Version	Purpose	Integration Method
Next.js	14.2+	React framework with SSR/SSG	App Router architecture
React	18+	Component-based UI library	JSX components with TypeScript

Technology	Version	Purpose	Integration Method
TypeScript	5.8.3+	Type safety and development experience	Strict type checking across all components
TailwindCSS	4.1.11+	Utility-first styling framework	Zero configuration required by default with PostCSS integration

Blockchain-Specific UI Libraries:

Library	Purpose	Integration	UI Components
@solana/web3.js	Blockchain interaction	Provides a nice interface to interact with Solana's blockchain, allowing connection to browser wallets like Phantom and creation, signing and submission of transactions	Connection status, transaction progress
@solana/wallet-adapter-react	Wallet integration	ConnectionProvider, WalletProvider, WalletModalProvider components	Wallet connection buttons, modal dialogs
@solana/wallet-adapter-react-ui	Pre-built UI components	WalletMultiButton component for opening connect modal	Standardized wallet interface elements

7.1.2 Egyptian Cultural Design System

Visual Identity Framework

TeosPump implements a comprehensive Egyptian cultural design system that honors the rich heritage of ancient Egypt while maintaining modern usability standards. Choose colors wisely: Neutral or earthy tones work best, letting the detailed style pop. Egyptian fonts can enhance designs effectively. A memorable project was a museum exhibit poster featuring a bold Egyptian title paired with a clean sans-serif subtext and a minimal background, resulting in a sharp and mystical look.

Cultural Design Elements:

Element Category	Design Approach	Implementation	Cultural Significance
Color Palette	Gold and green colors as seen in Egyptian artifacts	CSS custom properties with TailwindCSS extensions	Represents prosperity and eternal life
Typography	Egyptian fonts offer a perfect blend of tradition and versatility, making them an essential tool for any designer looking to create impactful visuals. Their bold style works wonders in headlines and historical themes	Custom font stack with hieroglyphic-inspired headings	Connects to ancient Egyptian writing systems
Iconography	Falcon heads as symbols of Horus, the god of the sky and divine kingship, protector of the deceased	SVG icons with Egyptian motifs	Spiritual protection and divine authority
Patterns	Repeated shapes, lines, and colors forming patterns as seen in Egyptian Mummy Cases	CSS patterns and background textures	Traditional Egyptian artistic expression

Design System Color Palette:

```
/* Egyptian Cultural Color Scheme */
:root {
  /* Primary Egyptian Colors */
  --egyptian-gold: #FFD700;
  --egyptian-blue: #003366;
  --papyrus-cream: #FFEAA7;
```

```
  --desert-sand: #F4E4BC;

  /* Accent Colors */
  --lotus-green: #2E8B57;
  --kohl-black: #1C1C1C;
  --sunset-orange: #FF6B35;

  /* Neutral Tones */
  --stone-gray: #8B7D6B;
  --ivory-white: #FFFEF7;
}
```

7.1.3 Responsive Design Architecture

Mobile-First Design Approach

By following these steps, you can create a visually appealing and responsive website layout using Tailwind. Remember to refer to the TailwindCSS documentation for a comprehensive list of utility classes and their usage. Additionally, consider applying design principles and best practices to ensure a user-friendly and aesthetically pleasing website layout.

Responsive Breakpoint Strategy:

Breakpoint	Screen Size	Layout Approach	Egyptian Design Adaptations
Mobile (sm)	320px - 640px	Single column, stacked components	Simplified hieroglyphic elements
Tablet (md)	641px - 1024px	Two-column layout for forms	Balanced Egyptian motifs
Desktop (lg)	1025px - 1440px	Multi-column dashboard layout	Full Egyptian visual elements
Wide (xl)	1441px+	Expanded content areas	Enhanced cultural decorations

Component Responsiveness:

```
// Responsive Egyptian-themed component example
const EgyptianCard = ({ children, className = "" }) => {
  return (
    <div className={`
      bg-gradient-to-br from-papyrus-cream to-desert-sand
      border-2 border-egyptian-gold
      rounded-lg shadow-lg
      p-4 md:p-6 lg:p-8
      relative overflow-hidden
      ${className}
    `}>
      {/* Egyptian decorative border pattern */}
      <div className="absolute inset-0 opacity-10">
        <div className="h-full w-full bg-egyptian-pattern"></div>
      </div>
      <div className="relative z-10">
        {children}
      </div>
    </div>
  );
};
```

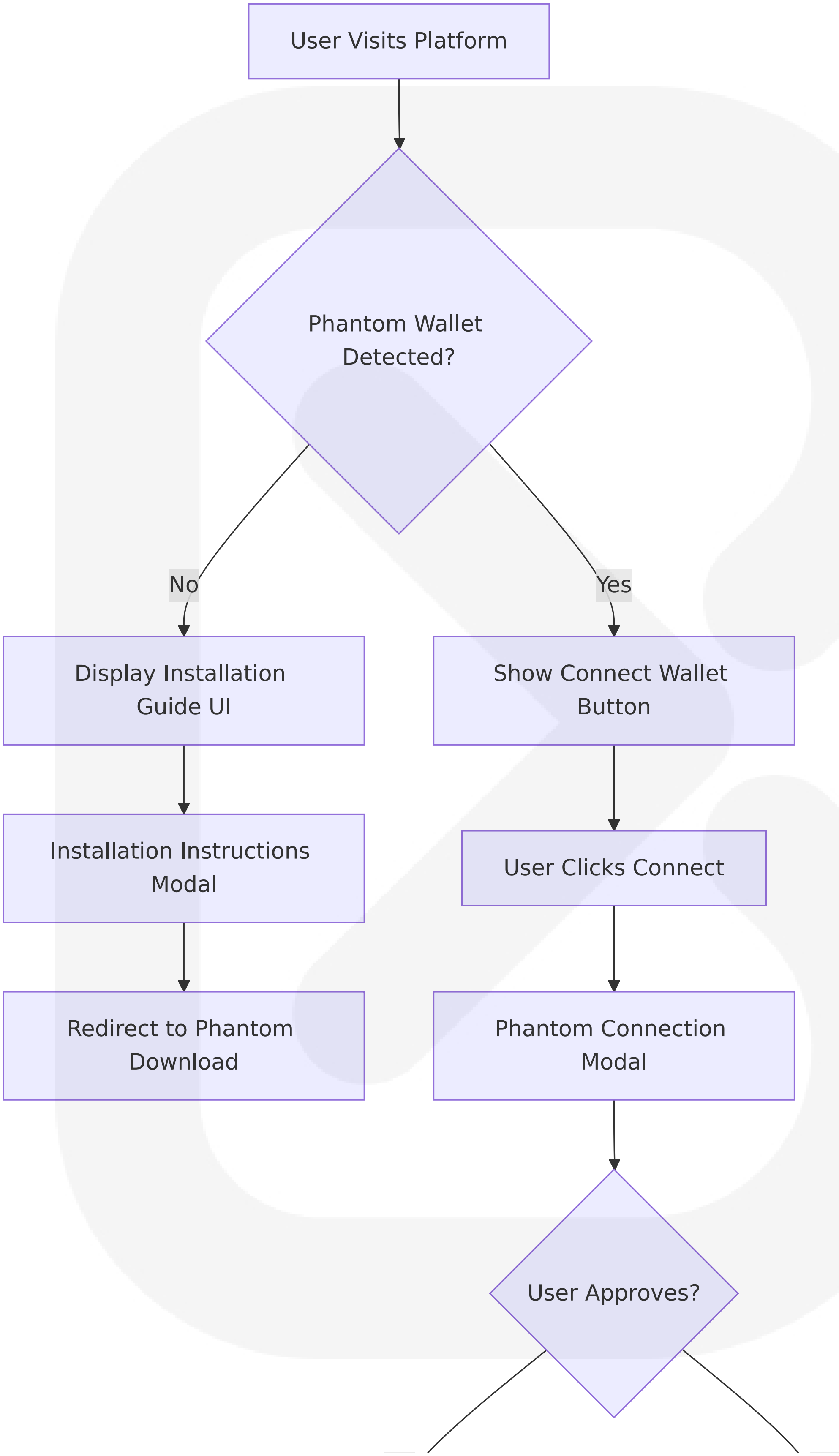
7.2 UI USE CASES

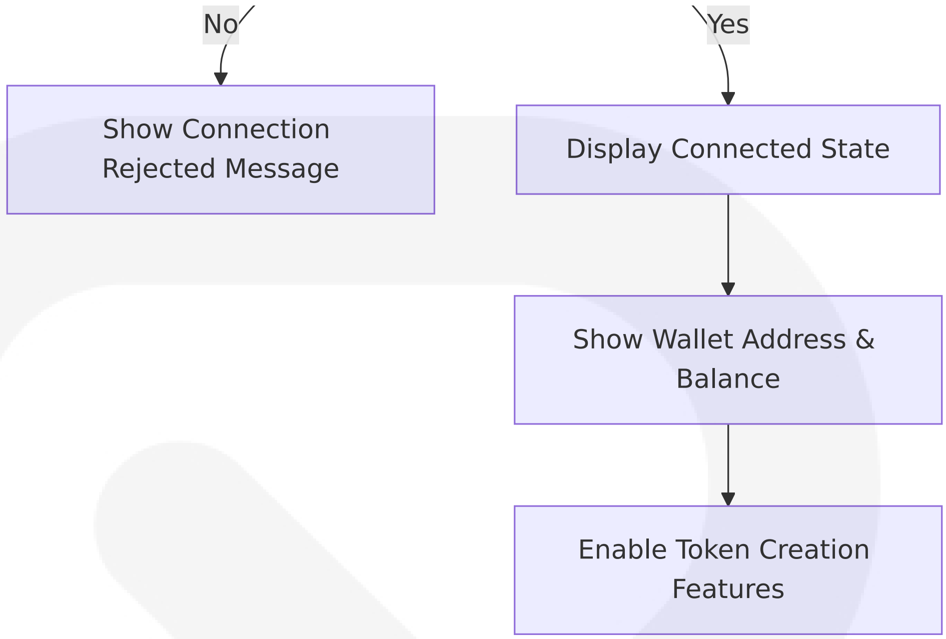
7.2.1 Primary User Workflows

Wallet Connection Workflow

Taking a look at the documentation, looks like we can access the Phantom on the window object. This makes things much simpler than having to use the wallet adapter from Solana Labs. Obviously, if you need to integrate all these wallets, it's probably good to use it, but if you're only supporting Phantom, you don't need it.

Wallet Connection UI Flow:


User Visits Platform
Phantom Wallet Detected?
No
Yes
Display Installation Guide UI
Installation Instructions Modal
Redirect to Phantom Download
Show Connect Wallet Button
User Clicks Connect
Phantom Connection Modal
User Approves?



Token Creation Workflow

The token creation workflow represents the core business functionality of TeosPump, guiding users through the complete process of minting SPL tokens on Solana.

Token Creation UI Components:

UI Component	Purpose	User Interaction	Visual Design
Token Form	Parameter input collection	Text inputs, dropdowns, sliders	Egyptian-themed form styling
Payment Selector	$TEOS vs SOL payment choice	Radio buttons with balance display	Gold accent highlighting
Transaction Progress	Real-time status updates	Progress indicators, loading states	Hieroglyphic progress symbols
Success Display	Token creation confirmation	Copy-to-clipboard, share buttons	Celebratory Egyptian motifs

Mobile Mining Integration Workflow

The mobile mining integration provides seamless synchronization between the web platform and mobile applications for reward distribution.

Mobile Sync UI Elements:

```
// Mobile Mining Status Component
const MiningStatusCard = () => {
  return (
    <div className="bg-gradient-to-r from-egyptian-blue to-lotus-green p-
      <div className="flex items-center justify-between">
        <div>
          <h3 className="text-lg font-bold">Mobile Mining Status</h3>
          <p className="text-sm opacity-90">Sync with Pi Network</p>
        </div>
        <div className="text-right">
          <div className="text-2xl font-bold text-egyptian-gold">
            {rewardBalance} $TEOS
          </div>
          <div className="text-xs">Pending Rewards</div>
        </div>
      </div>
      <div className="mt-4">
        <div className="flex justify-between text-sm">
          <span>Last Sync</span>
          <span>{lastSyncTime}</span>
        </div>
        <div className="w-full bg-egyptian-blue bg-opacity-50 rounded-fu
          <div
            className="bg-egyptian-gold h-2 rounded-full transition-all d
            style={{ width: `${syncProgress}%` }}
          ></div>
        </div>
      </div>
    </div>
  );
};
```

7.2.2 Administrative Use Cases

Platform Monitoring Dashboard

Administrative users require comprehensive monitoring capabilities to ensure platform health and performance.

Admin Dashboard Components:

Dashboard Section	Metrics Displayed	UI Elements	Update Frequency
System Health	Uptime, response times, error rates	Status indicators, charts	Real-time
Token Creation Analytics	Creation volume, success rates	Bar charts, trend lines	5-minute intervals
Payment Processing	Transaction volumes, fee collection	Financial dashboards	Real-time
User Activity	Wallet connections, platform usage	Activity feeds, heatmaps	1-minute intervals

Error Handling and User Feedback

Comprehensive error handling ensures users receive clear, actionable feedback throughout their platform interactions.

Error State UI Patterns:

```
// Error Boundary Component with Egyptian Theming
const EgyptianErrorBoundary = ({ error, retry }) => {
  return (
    <div className="min-h-screen flex items-center justify-center bg-grad
      <div className="max-w-md mx-auto text-center p-8">
        <div className="mb-6">
          {/* Egyptian-themed error icon */}
          <div className="w-24 h-24 mx-auto bg-egyptian-gold rounded-full
            <svg className="w-12 h-12 text-egyptian-blue" fill="currentCo
              {/* Ankh symbol or similar Egyptian icon */}
            </svg>
          </div>
        </div>
        <h2 className="text-2xl font-bold text-egyptian-blue mb-4">
          The Gods Have Spoken
        </h2>
        <p className="text-stone-gray mb-6">
          An unexpected error has occurred in the sacred realm.
          The scribes are working to restore balance.
```

```
          </p>
          <button
            onClick={retry}
            className="bg-egyptian-gold hover:bg-yellow-600 text-egyptian-
          >
            Invoke the Ancient Powers (Retry)
          </button>
        </div>
      </div>
    );
  };
```

7.3 UI/BACKEND INTERACTION BOUNDARIES

7.3.1 API Integration Patterns

Frontend-Backend Communication Architecture

The UI layer communicates with the backend through well-defined API boundaries that separate concerns between presentation logic and business logic.

API Integration Boundaries:

Boundary Type	Frontend Responsibility	Backend Responsibility	Data Flow
Authentication	Wallet connection UI, signature requests	JWT validation, session management	Bidirectional with real-time updates
Token Creation	Form validation, user feedback	SPL token minting, blockchain interaction	Request-response with progress updates
Payment Processing	Payment method selection, confirmation UI	Transaction verification, fee collection	Asynchronous with status callbacks

Boundary Type	Frontend Responsibility	Backend Responsibility	Data Flow
Mobile Sync	Status display, reward notifications	Mining validation, $TEOS distribution	Event-driven with periodic polling

State Management Architecture:

```
// Frontend State Management for Blockchain Interactions
interface TeosPumpState {
  wallet: {
    connected: boolean;
    address: string | null;
    balance: {
      sol: number;
      teos: number;
    };
  };
  tokenCreation: {
    formData: TokenFormData;
    status: 'idle' | 'validating' | 'processing' | 'success' | 'error';
    transactionSignature: string | null;
    error: string | null;
  };
  mobileSync: {
    status: 'synced' | 'pending' | 'error';
    lastSync: Date | null;
    pendingRewards: number;
  };
}

// API Service Layer
class TeosPumpAPI {
  async createToken(tokenData: TokenFormData): Promise<TokenCreationResp
    // Frontend validation
    const validationResult = validateTokenForm(tokenData);
    if (!validationResult.isValid) {
      throw new ValidationError(validationResult.errors);
    }

    // Backend API call
```

```
    const response = await fetch('/api/token/create', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${getAuthToken()}`
      },
      body: JSON.stringify(tokenData)
    });

    if (!response.ok) {
      throw new APIError(await response.json());
    }

    return response.json();
  }
}
```

7.3.2 Real-Time Data Synchronization

WebSocket Integration for Live Updates

Real-time updates ensure users receive immediate feedback on blockchain transactions and mobile mining activities.

Real-Time Update Patterns:

Update Type	Trigger	UI Response	Backend Event
Transaction Status	Blockchain confirmation	Progress bar updates, status messages	Solana network polling
Wallet Balance	Payment completion	Balance display refresh	Account balance queries
Mobile Rewards	Mining activity	Notification badges, reward counters	Mobile app synchronization
System Alerts	Platform events	Toast notifications, modal dialogs	Administrative broadcasts

WebSocket Implementation:

```
// Real-time connection management
class TeosPumpWebSocket {
  private ws: WebSocket | null = null;
  private reconnectAttempts = 0;
  private maxReconnectAttempts = 5;

  connect(walletAddress: string) {
    this.ws = new WebSocket(`wss://api.teospump.com/ws?wallet=${walletAdd

    this.ws.onmessage = (event) => {
      const data = JSON.parse(event.data);
      this.handleMessage(data);
    };

    this.ws.onclose = () => {
      this.handleReconnection();
    };
  }

  private handleMessage(data: WebSocketMessage) {
    switch (data.type) {
      case 'TRANSACTION_UPDATE':
        updateTransactionStatus(data.payload);
        break;
      case 'BALANCE_UPDATE':
        updateWalletBalance(data.payload);
        break;
      case 'MINING_REWARD':
        showRewardNotification(data.payload);
        break;
    }
  }
}
```

7.3.3 Error Handling and User Feedback

Comprehensive Error Management System

The UI implements sophisticated error handling that provides users with clear, actionable feedback while maintaining the Egyptian cultural theme.

Error Classification and UI Response:

Error Category	UI Treatment	User Action	Recovery Method
Network Errors	Retry prompts with Egyptian messaging	Manual retry button	Automatic reconnection
Validation Errors	Inline form feedback with hieroglyphic icons	Form correction	Real-time validation
Blockchain Errors	Transaction failure modals	Alternative payment methods	Guided troubleshooting
Authentication Errors	Wallet reconnection prompts	Wallet reconnection	Session restoration

7.4 UI SCHEMAS

7.4.1 Component Architecture Schema

Hierarchical Component Structure

```
// Root Application Schema
interface TeosPumpApp {
  layout: {
    header: NavigationHeader;
    main: MainContent;
    footer: CulturalFooter;
  };
  providers: {
    walletProvider: SolanaWalletProvider;
    themeProvider: EgyptianThemeProvider;
    stateProvider: ApplicationStateProvider;
  };
}
```

```typescript
// Navigation Header Schema
interface NavigationHeader {
  logo: EgyptianLogo;
  navigation: {
    items: NavigationItem[];
    mobileMenu: MobileMenuToggle;
  };
  walletConnection: WalletConnectionButton;
  themeToggle: DarkModeToggle;
}

// Main Content Schema
interface MainContent {
  hero: HeroSection;
  tokenCreation: TokenCreationSection;
  mobileSync: MobileSyncSection;
  analytics: PlatformAnalytics;
}

// Token Creation Form Schema
interface TokenCreationForm {
  metadata: {
    name: FormField<string>;
    symbol: FormField<string>;
    description: FormField<string>;
    image: ImageUploadField;
  };
  economics: {
    totalSupply: FormField<number>;
    decimals: FormField<number>;
    mintAuthority: FormField<string>;
  };
  payment: {
    method: PaymentMethodSelector;
    amount: PaymentAmountDisplay;
    confirmation: PaymentConfirmation;
  };
}
```

7.4.2 State Management Schema

Application State Structure

```
// Global Application State Schema
interface ApplicationState {
  user: UserState;
  blockchain: BlockchainState;
  ui: UIState;
  mobile: MobileState;
}

interface UserState {
  wallet: {
    connected: boolean;
    address: PublicKey | null;
    balance: {
      sol: number;
      teos: number;
      lastUpdated: Date;
    };
  };
  preferences: {
    theme: 'light' | 'dark' | 'egyptian';
    language: 'en' | 'ar';
    notifications: NotificationSettings;
  };
}

interface BlockchainState {
  network: 'mainnet-beta' | 'devnet' | 'testnet';
  connection: {
    status: 'connected' | 'connecting' | 'disconnected';
    endpoint: string;
    latency: number;
  };
  transactions: {
    pending: Transaction[];
    confirmed: Transaction[];
    failed: Transaction[];
  };
}

interface UIState {
```

```
  modals: {
    walletConnection: ModalState;
    tokenCreation: ModalState;
    transactionStatus: ModalState;
  };
  notifications: Notification[];
  loading: {
    global: boolean;
    components: Record<string, boolean>;
  };
}
```

7.4.3 Form Validation Schema

Comprehensive Form Validation Rules

```
// Token Creation Form Validation Schema
interface TokenFormValidation {
  name: {
    required: true;
    minLength: 1;
    maxLength: 32;
    pattern: /^[a-zA-Z0-9\s]+$/;
    customValidation: (value: string) => ValidationResult;
  };
  symbol: {
    required: true;
    minLength: 1;
    maxLength: 10;
    pattern: /^[A-Z0-9]+$/;
    transform: (value: string) => string; // Convert to uppercase
  };
  supply: {
    required: true;
    min: 1;
    max: 10_000_000_000; // 10 billion
    type: 'integer';
    customValidation: validateSupplyRange;
  };
  decimals: {
    required: true;
```

```
    min: 0;
    max: 9;
    type: 'integer';
    default: 9;
  };
}

// Validation Result Schema
interface ValidationResult {
  isValid: boolean;
  errors: ValidationError[];
  warnings: ValidationWarning[];
}

interface ValidationError {
  field: string;
  message: string;
  code: string;
  severity: 'error' | 'warning' | 'info';
}
```

7.5 SCREENS REQUIRED

7.5.1 Primary Application Screens

Landing Page (Home Screen)

The landing page serves as the primary entry point, showcasing TeosPump's Egyptian cultural branding and core value proposition.

Landing Page Components:

Section	Purpose	Egyptian Design Elements	Interactive Features
Hero Section	Platform introduction and CTA	Pyramid-inspired layout, hieroglyphic decorations	Animated wallet connection

Section	Purpose	Egyptian Design Elements	Interactive Features
Features Overview	Core platform capabilities	Egyptian iconography, papyrus textures	Interactive feature cards
Cultural Branding	Elmahrosa International showcase	Traditional Egyptian patterns, gold accents	Cultural storytelling elements
Getting Started	User onboarding guidance	Step-by-step hieroglyphic progression	Progressive disclosure

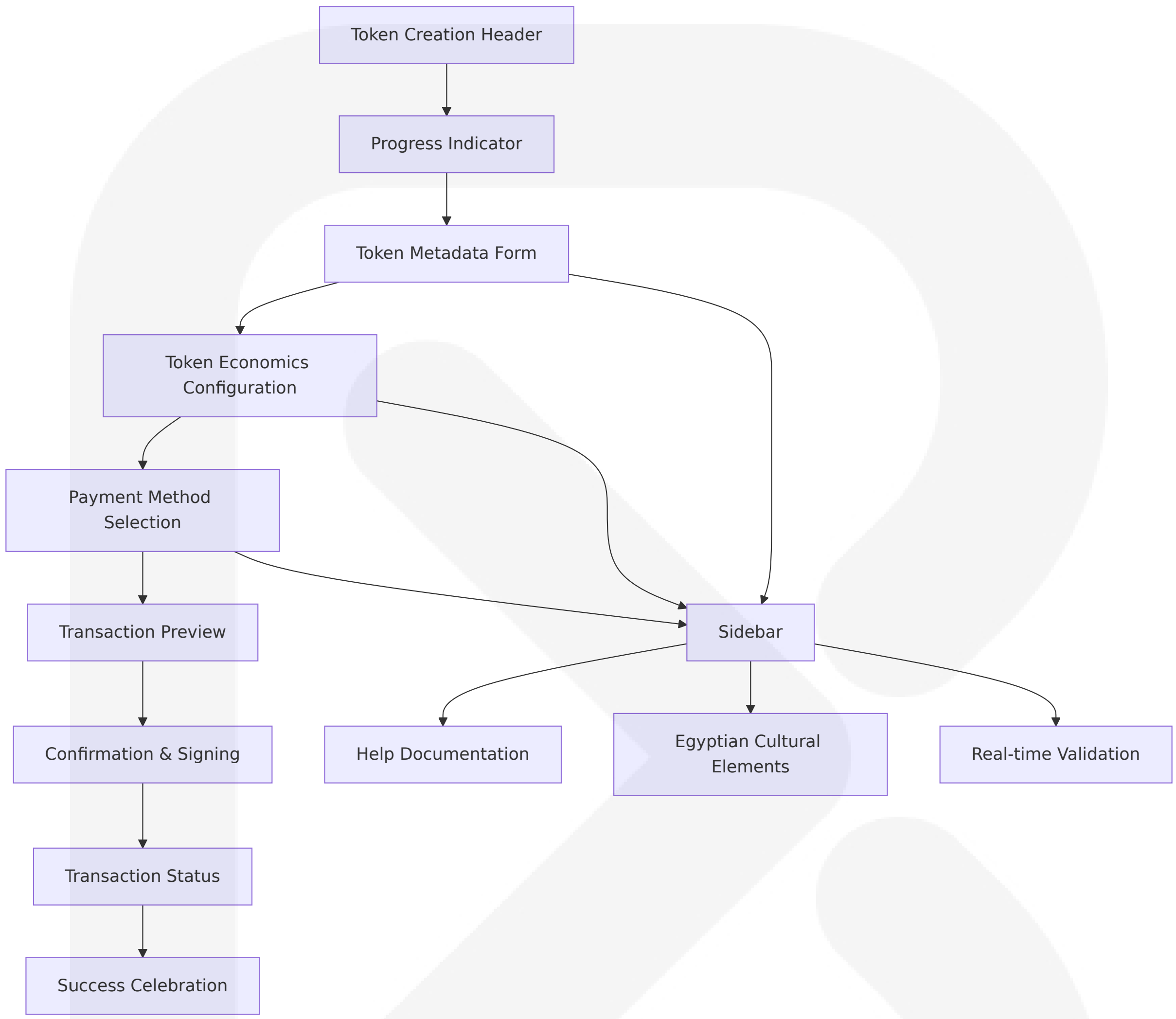
Landing Page Layout Schema:

```
interface LandingPageLayout {
  hero: {
    title: "TeosPump - Sacred Token Launchpad";
    subtitle: "Create meme tokens blessed by ancient Egyptian wisdom";
    cta: WalletConnectionButton;
    background: EgyptianPatternOverlay;
  };
  features: {
    tokenCreation: FeatureCard;
    mobileRewards: FeatureCard;
    culturalBranding: FeatureCard;
    solanaIntegration: FeatureCard;
  };
  statistics: {
    tokensCreated: AnimatedCounter;
    totalVolume: AnimatedCounter;
    activeUsers: AnimatedCounter;
  };
  footer: CulturalFooter;
}
```

Token Creation Screen

The token creation screen represents the core functionality, guiding users through the complete SPL token minting process.

Token Creation Screen Layout:



Token Creation Form Components:

Form Section	Input Fields	Validation Rules	Egyptian Theming
Basic Information	Name, Symbol, Description	Recommended length typically between 70 to 80 characters for optimal readability	Hieroglyphic input decorations
Token Economics	Total Supply, Decimals, Mint Authority	Supply range 1B-10B tokens	Egyptian numerical symbols

Form Section	Input Fields	Validation Rules	Egyptian Theming
Visual Identity	Logo Upload, Color Scheme	Image format validation	Papyrus-style upload area
Payment Configuration	$TEOS vs SOL selection	Balance verification	Gold coin visual indicators

7.5.2 Wallet Integration Screens

Wallet Connection Modal

Taking a look at the documentation, looks like we can access the Phantom on the window object. This makes things much simpler than having to use the wallet adapter from Solana Labs. Obviously, if you need to integrate all these wallets, it's probably good to use it, but if you're only supporting Phantom, you don't need it.

Wallet Connection UI Flow:

```
interface WalletConnectionModal {
  detection: {
    phantomInstalled: boolean;
    alternativeWallets: WalletOption[];
    installationGuide: InstallationInstructions;
  };
  connection: {
    connectButton: PhantomConnectButton;
    statusIndicator: ConnectionStatus;
    errorHandling: ConnectionErrorDisplay;
  };
  postConnection: {
    walletInfo: WalletInfoDisplay;
    balanceDisplay: BalanceCard;
    networkStatus: NetworkIndicator;
  };
}
```

Wallet Status Dashboard

Once connected, users access a comprehensive wallet dashboard showing their Solana assets and TeosPump activity.

Dashboard Components:

Component	Information Displayed	Update Frequency	Egyptian Styling
Balance Overview	SOL, $TEOS, created tokens	Real-time	Golden balance cards
Transaction History	Recent platform activity	Live updates	Hieroglyphic transaction icons
Token Portfolio	Created and owned tokens	On-demand refresh	Egyptian artifact styling
Mobile Mining Status	Reward accumulation, sync status	5-minute intervals	Pyramid progress indicators

7.5.3 Administrative Screens

Platform Analytics Dashboard

Administrative users require comprehensive monitoring and analytics capabilities to ensure platform health and business insights.

Analytics Dashboard Layout:

```
interface AdminDashboard {
  overview: {
    kpis: KeyPerformanceIndicators;
    systemHealth: SystemHealthMetrics;
    recentActivity: ActivityFeed;
  };
  tokenAnalytics: {
    creationVolume: VolumeChart;
    successRates: SuccessRateMetrics;
    popularTokens: PopularTokensList;
  };
```

```
  userAnalytics: {
    activeUsers: UserActivityChart;
    walletConnections: ConnectionMetrics;
    geographicDistribution: GeographicChart;
  };
  financialMetrics: {
    revenueTracking: RevenueChart;
    feeCollection: FeeMetrics;
    paymentMethods: PaymentDistribution;
  };
}
```

System Monitoring Screen

Real-time system monitoring ensures platform reliability and performance optimization.

Monitoring Components:

Monitoring Area	Metrics Tracked	Alert Thresholds	Visual Representation
Blockchain Health	RPC response times, transaction success rates	>5s response, <95% success	Egyptian-themed status indicators
Application Performance	Page load times, API response times	>2s load, >500ms API	Pyramid-shaped performance charts
User Experience	Error rates, conversion funnels	>5% error rate	Hieroglyphic error symbols
Infrastructure	Server health, database performance	>80% resource usage	Ancient Egyptian architectural metaphors

7.6 USER INTERACTIONS

7.6.1 Primary Interaction Patterns

Wallet Connection Interaction Flow

Here we are initializing phantom to null, but upon mounting of the component, we want to see if window has a property named solana. If it does, then we check if its isPhantom property is truthy. If it is, we'll set the state of phantom with setPhantom dispatch function. This all happens in the useEffect React hook. The second parameter here is an empty array, so this callback only runs when the component is first mounted.

Interaction Sequence:

```
// Wallet Connection Interaction Handler
const useWalletConnection = () => {
  const [phantom, setPhantom] = useState<PhantomWallet | null>(null);
  const [connected, setConnected] = useState(false);
  const [connecting, setConnecting] = useState(false);

  useEffect(() => {
    // Detect Phantom wallet
    if (window.solana?.isPhantom) {
      setPhantom(window.solana);
    }
  }, []);

  const connectWallet = async () => {
    if (!phantom) {
      showInstallationModal();
      return;
    }

    try {
      setConnecting(true);
      const response = await phantom.connect();
      setConnected(true);
      showSuccessNotification("Wallet connected successfully!");
    } catch (error) {
      showErrorNotification("Connection failed. Please try again.");
    } finally {
      setConnecting(false);
    }
  };
```

```
  return { phantom, connected, connecting, connectWallet };
};
```

Token Creation Interaction Patterns

The token creation process involves multiple user interaction points with comprehensive validation and feedback.

Interaction Flow Mapping:

Interaction Stage	User Actions	System Response	Visual Feedback
Form Input	Text entry, dropdown selection	Real-time validation	Inline error messages with Egyptian icons
Payment Selection	Radio button choice	Balance verification	Golden highlighting for selected option
Transaction Signing	Phantom wallet approval	Progress tracking	Hieroglyphic loading animations
Completion	Success acknowledgment	Token details display	Celebratory Egyptian animations

7.6.2 Mobile-Responsive Interactions

Touch-Optimized Interface Design

Mobile interactions require special consideration for touch targets and gesture-based navigation.

Mobile Interaction Specifications:

Interaction Type	Touch Target Size	Gesture Support	Accessibility
Primary Buttons	Minimum 44px × 44px	Tap with haptic feedback	ARIA labels, screen reader support

Interaction Type	Touch Target Size	Gesture Support	Accessibility
Form Inputs	Minimum 48px height	Touch focus, keyboard navigation	High contrast mode compatibility
Navigation Elements	Minimum 40px × 40px	Swipe gestures for mobile menu	Voice control compatibility
Interactive Cards	Full card touch area	Long press for additional options	Reduced motion preferences

Mobile Interaction Implementation:

```
// Mobile-optimized interaction component
const MobileTokenCard = ({ token, onSelect }) => {
  const [isPressed, setIsPressed] = useState(false);

  return (
    <div
      className={`
        touch-manipulation
        min-h-[88px] w-full
        bg-gradient-to-r from-papyrus-cream to-desert-sand
        border-2 border-egyptian-gold
        rounded-lg p-4
        transition-all duration-200
        ${isPressed ? 'scale-95 shadow-inner' : 'shadow-lg'}
        active:scale-95
        focus:ring-4 focus:ring-egyptian-gold focus:ring-opacity-50
      `}
      onTouchStart={() => setIsPressed(true)}
      onTouchEnd={() => setIsPressed(false)}
      onClick={() => onSelect(token)}
      role="button"
      tabIndex={0}
      aria-label={`Select ${token.name} token`}
    >
      <div className="flex items-center justify-between">
        <div className="flex-1 min-w-0">
          <h3 className="text-lg font-bold text-egyptian-blue truncate">
            {token.name}
          </h3>
```

```
          <p className="text-sm text-stone-gray">
            {token.symbol} • {formatSupply(token.supply)}
          </p>
        </div>
        <div className="ml-4 flex-shrink-0">
          <EgyptianIcon name="ankh" className="w-6 h-6 text-egyptian-gol
        </div>
      </div>
    </div>
  );
};
```

7.6.3 Accessibility Interactions

Inclusive Design Implementation

To ensure that typography is accessible and easily readable for individuals with disabilities such as color blindness, vision impairments, and hearing disabilities, it is crucial to adhere to the Web Content Accessibility Guidelines (WCAG).

Accessibility Features:

Accessibility Need	Implementation	Egyptian Theme Integration	Testing Method
Screen Readers	ARIA labels, semantic HTML	Descriptive Egyptian cultural context	Automated accessibility testing
Keyboard Navigation	Tab order, focus management	Visual focus indicators with Egyptian styling	Manual keyboard testing
Color Contrast	WCAG AA compliance	High contrast Egyptian color combinations	Color contrast analyzers
Motor Impairments	Large touch targets, reduced motion	Simplified Egyptian animations	User testing with assistive devices

Accessibility Implementation:

```
// Accessible Egyptian-themed button component
const EgyptianButton = ({
  children,
  onClick,
  variant = 'primary',
  disabled = false,
  ariaLabel,
  ...props
}) => {
  const baseClasses = `
    inline-flex items-center justify-center
    px-6 py-3 rounded-lg font-bold
    transition-all duration-200
    focus:outline-none focus:ring-4 focus:ring-opacity-50
    disabled:opacity-50 disabled:cursor-not-allowed
    min-h-[44px] min-w-[44px]
  `;

  const variantClasses = {
    primary: `
      bg-egyptian-gold hover:bg-yellow-600
      text-egyptian-blue
      focus:ring-egyptian-gold
    `,
    secondary: `
      bg-transparent border-2 border-egyptian-gold
      text-egyptian-gold hover:bg-egyptian-gold hover:text-egyptian-blue
      focus:ring-egyptian-gold
    `
  };

  return (
    <button
      className={`${baseClasses} ${variantClasses[variant]}`}
      onClick={onClick}
      disabled={disabled}
      aria-label={ariaLabel || children}
      role="button"
      {...props}
    >
```

```
      {children}
    </button>
  );
};
```

7.6.4 Error State Interactions

Comprehensive Error Handling UI

Error states require clear communication and recovery paths while maintaining the Egyptian cultural theme.

Error Interaction Patterns:

Error Type	User Notification	Recovery Actions	Cultural Theming
Network Errors	Toast notifications with retry options	Automatic retry with exponential backoff	"The gods are temporarily silent" messaging
Validation Errors	Inline form feedback	Real-time correction guidance	Hieroglyphic error symbols
Transaction Failures	Modal dialogs with detailed explanations	Alternative payment methods, support contact	Egyptian mythology-inspired error descriptions
System Errors	Full-page error boundaries	Page refresh, navigation alternatives	Ancient Egyptian wisdom quotes

7.7 VISUAL DESIGN CONSIDERATIONS

7.7.1 Egyptian Cultural Visual Language

Authentic Cultural Representation

While Egyptian fonts can enhance designs, use them thoughtfully. Consider cultural context and sensitivity to avoid misrepresentation in non-Egyptian themes. TeosPump's visual design honors Egyptian heritage through authentic cultural elements while maintaining modern usability.

Cultural Design Principles:

Design Element	Cultural Significance	Modern Implementation	Accessibility Considerations
Hieroglyphic Symbols	Sacred text with spells and prayers from the Book of the Dead	SVG icon system with semantic meaning	Alt text descriptions for screen readers
Egyptian Color Palette	Gold and green colors representing prosperity and eternal life	CSS custom properties with WCAG compliance	High contrast alternatives for accessibility
Geometric Patterns	Grids in Egyptian designs helping maintain ratios and proportions	CSS Grid and Flexbox layouts	Reduced motion options for vestibular disorders
Sacred Symbols	Falcon heads as symbols of Horus, divine kingship and protection	Interactive UI elements with cultural context	Keyboard navigation support

7.7.2 Typography System

Egyptian-Inspired Typography Hierarchy

Egyptian typography is known for its distinctive and bold characteristics. In this article, we will explore the profound influence of Egyptian typography on modern design, tracing its historical roots and examining its contemporary applications.

Typography Scale and Hierarchy:

```css
/* Egyptian Typography System */
.typography-system {
  /* Display Typography - Egyptian Inspired */
  --font-display: 'Cleopatra', 'Egyptian', serif;
  --font-heading: 'Sabana', 'Arabic-style', sans-serif;
  --font-body: 'Inter', 'Roboto', sans-serif;
  --font-mono: 'JetBrains Mono', monospace;

  /* Scale - Based on Egyptian proportions */
  --text-xs: 0.75rem;    /* 12px */
  --text-sm: 0.875rem;   /* 14px */
  --text-base: 1rem;     /* 16px */
  --text-lg: 1.125rem;   /* 18px */
  --text-xl: 1.25rem;    /* 20px */
  --text-2xl: 1.5rem;    /* 24px */
  --text-3xl: 1.875rem;  /* 30px */
  --text-4xl: 2.25rem;   /* 36px */
  --text-5xl: 3rem;      /* 48px */
  --text-6xl: 3.75rem;   /* 60px */
}

/* Hierarchical Typography Classes */
.heading-pharaoh {
  font-family: var(--font-display);
  font-size: var(--text-6xl);
  font-weight: 900;
  line-height: 1.1;
  color: var(--egyptian-gold);
  text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.3);
}

.heading-priest {
  font-family: var(--font-heading);
  font-size: var(--text-3xl);
  font-weight: 700;
  line-height: 1.2;
  color: var(--egyptian-blue);
}

.body-scribe {
  font-family: var(--font-body);
  font-size: var(--text-base);
  line-height: 1.6;
```

```
  color: var(--stone-gray);
}
```

7.7.3 Color Psychology and Cultural Significance

Egyptian Color Symbolism in UI Design

The use of colour to evoke emotions is an even older art form than typography, going right back to the ancient Greeks, Egyptians and Chinese. But how is it applied to modern UX design?

Color Application Strategy:

Color	Cultural Meaning	UI Application	Accessibility Notes
Egyptian Gold (#FFD700)	Symbols of life, regeneration, sun cycles, creation and rebirth	Primary CTAs, success states, premium features	4.5:1 contrast ratio with dark backgrounds
Egyptian Blue (#003366)	Divine authority, eternal sky	Headers, navigation, trust indicators	WCAG AA compliant with light text
Papyrus Cream (#FFEAA7)	Ancient wisdom, sacred texts	Background surfaces, content areas	Sufficient contrast for body text
Lotus Green (#2E8B57)	Lotus flowers as symbols of life and regeneration	Success messages, growth indicators	Color-blind friendly alternative provided

7.7.4 Animation and Micro-Interactions

Egyptian-Themed Motion Design

Animations enhance user experience while respecting cultural significance and accessibility requirements.

Animation Design Principles:

Animation Type	Cultural Inspiration	Technical Implementation	Accessibility Considerations
Loading States	Lotus flowers closing at night and emerging in morning, symbols of daily cycles	CSS keyframes with lotus-inspired transitions	Respects `prefers-reduced-motion`
Success Celebrations	Ancient Egyptian victory ceremonies	Particle effects with hieroglyphic symbols	Optional animation with user control
Navigation Transitions	Papyrus scroll unrolling	Transform animations with easing	Reduced motion alternatives
Hover Effects	Golden illumination of sacred objects	CSS transforms with golden glow effects	Focus indicators for keyboard users

Animation Implementation:

```css
/* Egyptian-inspired animations */
@keyframes lotus-bloom {
  0% {
    transform: scale(0.8) rotate(-5deg);
    opacity: 0.6;
  }
  50% {
    transform: scale(1.05) rotate(2deg);
    opacity: 0.9;
  }
  100% {
    transform: scale(1) rotate(0deg);
    opacity: 1;
  }
}

@keyframes hieroglyph-reveal {
  0% {
    opacity: 0;
```

```
    transform: translateY(20px);
  }
  100% {
    opacity: 1;
    transform: translateY(0);
  }
}

/* Respectful of user preferences */
@media (prefers-reduced-motion: reduce) {
  .lotus-animation,
  .hieroglyph-animation {
    animation: none;
    transition: none;
  }
}

/* Interactive elements */
.egyptian-button {
  position: relative;
  overflow: hidden;
  transition: all 0.3s ease;
}

.egyptian-button::before {
  content: '';
  position: absolute;
  top: 0;
  left: -100%;
  width: 100%;
  height: 100%;
  background: linear-gradient(
    90deg,
    transparent,
    rgba(255, 215, 0, 0.3),
    transparent
  );
  transition: left 0.5s ease;
}

.egyptian-button:hover::before {
  left: 100%;
}
```

7.7.5 Responsive Visual Hierarchy

Adaptive Design for Multiple Devices

The visual hierarchy adapts gracefully across different screen sizes while maintaining Egyptian cultural integrity.

Responsive Design Breakpoints:

Breakpoint	Visual Adaptations	Egyptian Elements	Performance Considerations
Mobile (320px-640px)	Simplified hieroglyphic elements, larger touch targets	Essential Egyptian symbols only	Optimized image loading, reduced animations
Tablet (641px-1024px)	Balanced cultural elements, two-column layouts	Moderate Egyptian decorations	Progressive image enhancement
Desktop (1025px+)	Full Egyptian visual treatment, complex layouts	Rich cultural imagery and patterns	Full animation suite, high-resolution assets

Responsive Typography Implementation:

```
/* Fluid typography with Egyptian proportions */
.responsive-heading {
  font-size: clamp(1.5rem, 4vw, 3.75rem);
  line-height: clamp(1.2, 1.5, 1.4);
  font-family: var(--font-display);
}

/* Responsive Egyptian patterns */
.egyptian-pattern {
  background-size:
    clamp(20px, 5vw, 60px)
    clamp(20px, 5vw, 60px);
  opacity: clamp(0.1, 0.3, 0.2);
}

/* Adaptive cultural elements */
```

```
@media (max-width: 640px) {
  .hieroglyph-decoration {
    display: none; /* Hide complex decorations on small screens */
  }

  .simplified-egyptian-icon {
    display: block; /* Show simplified versions */
  }
}

@media (min-width: 1024px) {
  .full-egyptian-treatment {
    display: block; /* Show full cultural elements on larger screens */
  }
}
```

7.7.6 Dark Mode and Theme Variations

Egyptian-Themed Dark Mode Implementation

Dark mode maintains cultural authenticity while providing modern user experience expectations.

Theme Variation Strategy:

Theme Mode	Color Adaptations	Cultural Elements	User Experience
Light Mode	Traditional Egyptian gold and blue	Full papyrus textures, bright hieroglyphs	Daytime usage, high visibility
Dark Mode	Muted gold, deep blues, stone grays	Subtle Egyptian patterns, glowing accents	Nighttime usage, reduced eye strain
High Contrast	Maximum contrast ratios	Simplified Egyptian symbols	Accessibility compliance, vision impairments
Reduced Motion	Static Egyptian elements	No animations, instant transitions	Vestibular disorder accommodation

Dark Mode Implementation:

```css
/* Egyptian Dark Mode Color Scheme */
[data-theme="dark"] {
  --egyptian-gold: #B8860B; /* Darker gold for better contrast */
  --egyptian-blue: #1E3A8A; /* Deeper blue */
  --papyrus-cream: #2D2D2D; /* Dark background */
  --desert-sand: #3D3D3D; /* Darker surface */
  --stone-gray: #A0A0A0; /* Lighter text for contrast */
  --kohl-black: #FFFFFF; /* Inverted for dark mode */
}

/* Dark mode Egyptian patterns */
[data-theme="dark"] .egyptian-pattern {
  opacity: 0.05; /* Reduced opacity for dark backgrounds */
  filter: brightness(0.7); /* Dimmed patterns */
}

/* Dark mode hieroglyphic elements */
[data-theme="dark"] .hieroglyph-icon {
  filter: drop-shadow(0 0 4px var(--egyptian-gold));
  opacity: 0.8;
}
```

This comprehensive UI design framework ensures TeosPump delivers an authentic Egyptian cultural experience while maintaining modern usability standards, accessibility compliance, and responsive design principles across all user interactions and visual elements.

8. INFRASTRUCTURE

8.1 DEPLOYMENT ENVIRONMENT

8.1.1 Target Environment Assessment

Environment Type: Hybrid Cloud-Native Architecture

TeosPump implements a **hybrid cloud-native deployment strategy** that combines Vercel's highly optimized infrastructure and CDN for frontend deployment with traditional cloud services for backend operations. This approach leverages Vercel's optimization for global performance and accessibility, ensuring the best user experience no matter where users are located.

Environment Classification:

Environment Type	Implementation	Justification	Resource Requirements
Frontend Cloud	Vercel Edge Network	Three default environments—Local, Preview, and Production for developing, testing, and deploying without impacting live site	Serverless functions, global CDN
Backend Cloud	Node.js hosting platform	Express.js API for mobile synchronization	Container-based deployment
Blockchain Network	Solana Mainnet/Devnet	Solana can power thousands of transactions per second	RPC endpoint access
Development Local	Developer machines	Local development and testing	Node.js 18+, development tools

Geographic Distribution Requirements:

Vercel's Edge Network enables storing content close to customers and running compute in regions close to data, reducing latency and improving end-user performance. The platform requires global distribution to serve the international blockchain community effectively.

Geographic Coverage Strategy:

Region	Primary Purpose	Infrastructure Requirements	Performance Targets
North America	Primary user base	Edge locations in US/Canada	<100ms latency
Europe	Secondary market	EU data centers	<150ms latency
Asia-Pacific	Growing market	APAC edge locations	<200ms latency
Global	Blockchain access	Solana RPC endpoints	<500ms blockchain queries

Resource Requirements Analysis:

Resource Type	Minimum Requirements	Recommended	Scaling Considerations
Compute	1 vCPU, 512MB RAM	2 vCPU, 1GB RAM	Auto-scaling based on traffic
Memory	512MB for Node.js	1GB for optimal performance	Memory-efficient blockchain libraries
Storage	1GB for application	5GB for logs and cache	Ephemeral storage for stateless design
Network	100Mbps bandwidth	1Gbps for high traffic	CDN for static asset delivery

Compliance and Regulatory Requirements:

The platform operates in the decentralized finance space with specific compliance considerations for blockchain applications:

Compliance Area	Requirement	Implementation	Monitoring
Data Privacy	GDPR, CCPA compliance	Minimal data collection, wallet-based authentication	Privacy audit logs

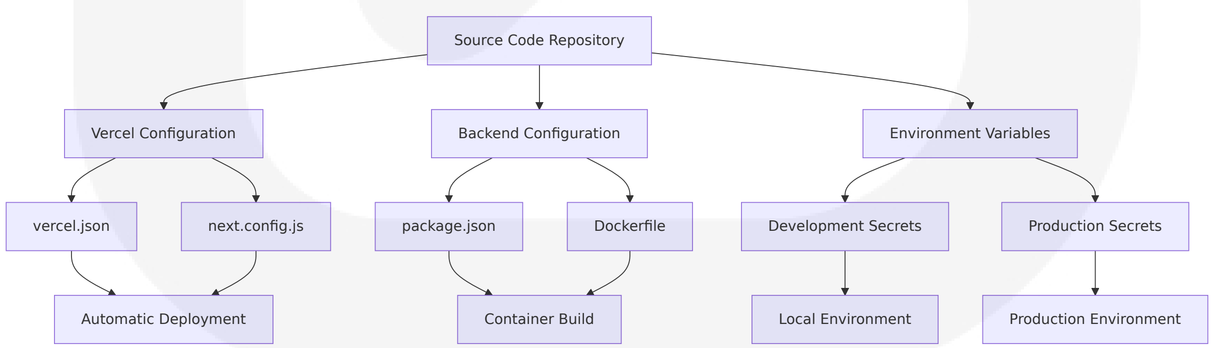
Compliance Area	Requirement	Implementation	Monitoring
Financial Regulations	AML/KYC considerations	Non-custodial design, user-controlled keys	Transaction monitoring
Security Standards	SOC 2, ISO 27001	Vercel protects projects across access, application, and infrastructure security with built-in support for global security and data protection standards	Continuous security monitoring
Blockchain Compliance	Solana network standards	SPL token standard compliance	Smart contract audits

8.1.2 Environment Management

Infrastructure as Code (IaC) Approach

TeosPump implements a **simplified IaC strategy** that leverages platform-native configuration management rather than complex infrastructure orchestration tools. Vercel's Framework-defined Infrastructure (FdI) intelligently infers the necessary infrastructure directly from frontend code, eliminating the need for intricate configuration files.

IaC Implementation Strategy:



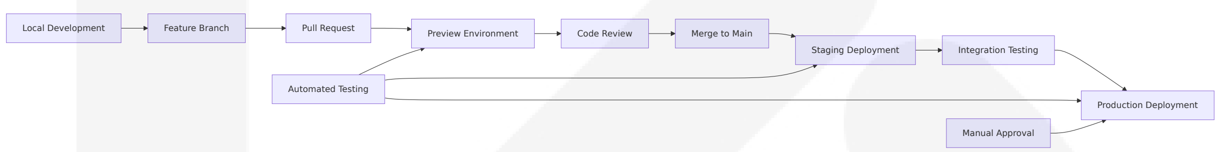
Configuration Management Strategy:

Configuration Type	Management Method	Storage Location	Version Control
Frontend Config	vercel.json, next.config.js	Git repository	Full version history
Backend Config	Environment variables	Secure secret management	Encrypted storage
Blockchain Config	RPC endpoints, contract addresses	Environment-specific configs	Immutable references
Security Config	API keys, certificates	Platform secret stores	Audit trail maintained

Environment Promotion Strategy:

The most common way to create a deployment is by pushing code to a connected Git repository. When you import a Git repository to Vercel, each commit or pull request automatically triggers a new deployment.

Promotion Workflow:



Environment Promotion Matrix:

Source Environment	Target Environment	Trigger	Validation Requirements
Local	Feature Branch	Developer push	Lint, type check
Feature Branch	Preview	Pull request	Unit tests, build success
Preview	Staging	Merge to main	Integration tests
Staging	Production	Manual approval	Full test suite, security scan

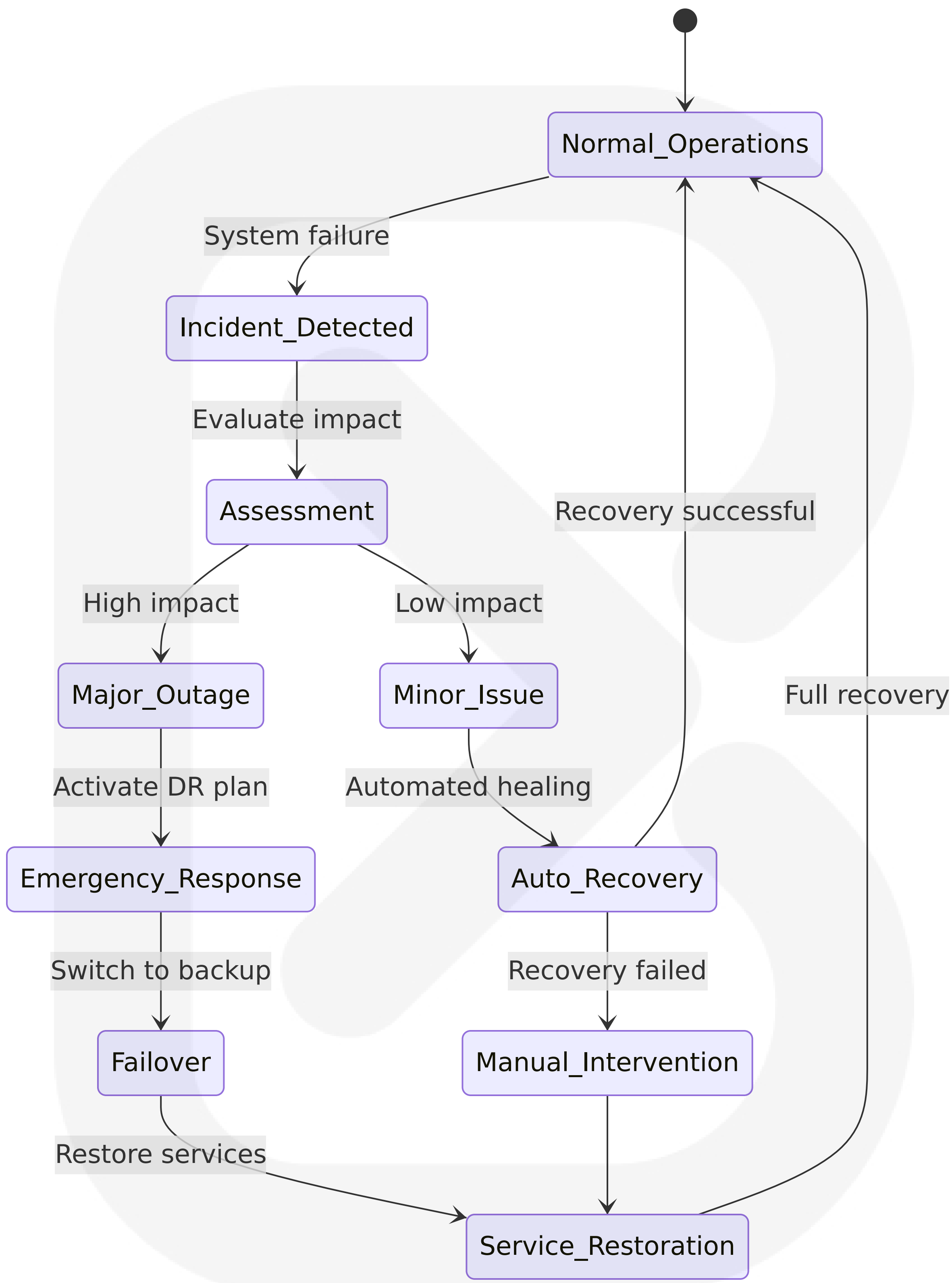
Backup and Disaster Recovery Plans:

The platform implements a **blockchain-first backup strategy** that leverages immutable blockchain records for critical data while maintaining traditional backups for operational data.

Backup Strategy Implementation:

Data Type	Backup Method	Frequency	Retention Period
Source Code	Git repository	Real-time	Permanent
Blockchain Data	Immutable ledger	N/A (permanent)	Permanent
Configuration	Environment snapshots	Daily	30 days
Application Logs	Automated archival	Hourly	90 days

Disaster Recovery Procedures:



Recovery Time and Point Objectives:

Service Component	RTO (Recovery Time)	RPO (Recovery Point)	Recovery Strategy
Frontend Application	5 minutes	0 (stateless)	Vercel's deployment process is designed to be scalable and secure, allowing focus entirely on developing projects with automatic failover
Backend API	15 minutes	5 minutes	Container restart, load balancer
Blockchain Access	N/A (external)	0 (immutable)	Multiple RPC endpoints
Configuration Data	30 minutes	1 hour	Git repository restoration

8.2 CLOUD SERVICES

8.2.1 Cloud Provider Selection and Justification

Primary Cloud Provider: Vercel (Frontend Cloud)

Vercel's Frontend Cloud provides fully managed infrastructure and developer experience tools that enable enterprises to accelerate their growth through exceptional user experiences by providing the toolkit frontend teams love and delivering global edge infrastructure.

Cloud Provider Selection Matrix:

Provider	Service Type	Use Case	Selection Rationale
Vercel	Frontend Cloud	Next.js application hosting	Dedicated Frontend Cloud that securely integrates with existing backend, taking hassl

Provider	Service Type	Use Case	Selection Rationale
			e and unpredictability out of web frontends
Generic Cloud	Backend hosting	Express.js API services	Cost-effective container hosting
Solana Network	Blockchain infrastructure	SPL token operations	Fast, decentralized, scalable, energy efficient blockchain that can power thousands of transactions per second

Vercel Selection Justification:

With Framework-defined Infrastructure (FdI), infrastructure is no longer a primary concern for frontend development. Vercel dynamically scales serverless functions, optimizes content delivery at the edge, and secures deployments—all without tedious YAML or cluster configuration.

8.2.2 Core Services Required

Vercel Platform Services:

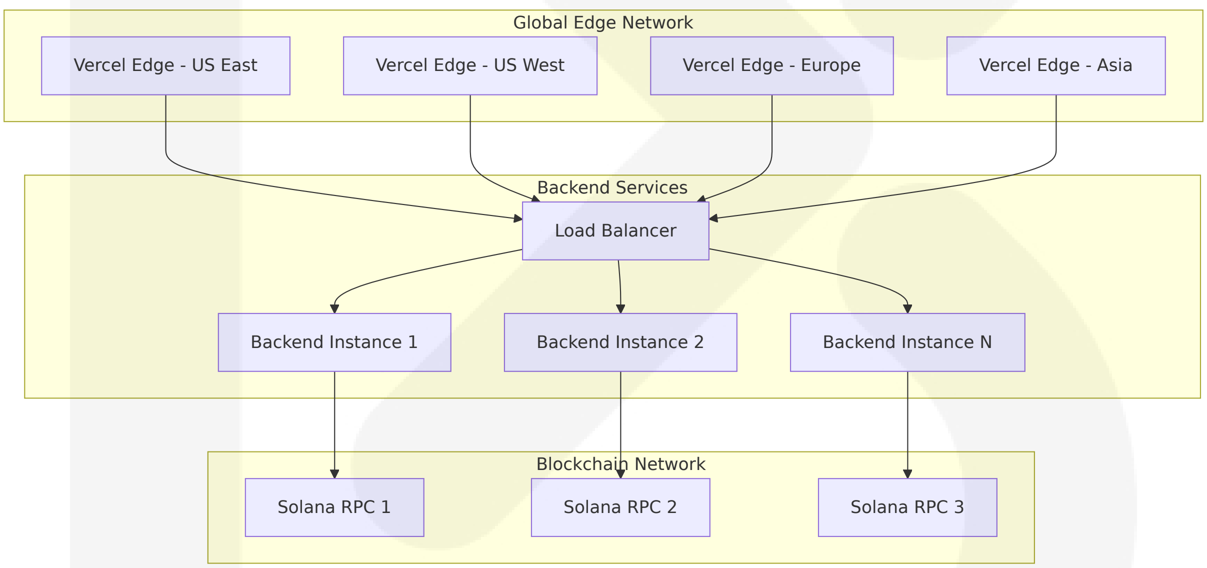
Service Category	Vercel Service	Version/Tier	Purpose
Compute	Serverless Functions	Latest	API routes, server-side rendering
Storage	Edge Network	Global CDN	Static asset delivery
Networking	Global Edge	Multi-region	Low-latency content delivery
Monitoring	Built-in Observability	Integrated	Route-aware observability to monitor and analyze performance and traffic of projects

Backend Infrastructure Services:

Service Type	Implementation	Specifications	Scaling Strategy
Container Hosting	Docker containers	Node.js 18+ runtime	Horizontal auto-scaling
Load Balancing	Application load balancer	Health check enabled	Traffic distribution
Database	Optional operational data	PostgreSQL/MongoDB	Connection pooling
Caching	Redis/In-memory	Session-based	Distributed caching

8.2.3 High Availability Design

Multi-Layer High Availability Architecture:



High Availability Specifications:

Component	Availability Target	Redundancy Strategy	Failover Time
Frontend (Vercel)	99.99%	Global edge locations with automatic failover and request routing to nearest edge location	<30 seconds

Component	Availability Target	Redundancy Strategy	Failover Time
Backend API	99.9%	Multi-instance deployment with load balancing	<60 seconds
Blockchain Access	99.5%	Multiple RPC endpoints with automatic switching	<10 seconds
CDN/Static Assets	99.99%	Global CDN with edge caching	Immediate

8.2.4 Cost Optimization Strategy

Vercel Pricing Optimization:

Hobby plan is for personal, non-commercial use. Pro is for small teams with moderate usage and collaboration requirements. Enterprise is for teams seeking greater performance, collaboration, and security.

Cost Optimization Implementation:

Cost Category	Optimization Strategy	Expected Savings	Implementation
Compute	Serverless auto-scaling	40-60% vs. fixed instances	Automatic infrastructure means feature development no longer waits on repetitive infrastructure setup, freed from complexities
Storage	Edge caching	70% bandwidth reduction	Static asset optimization
Monitoring	Built-in observability	100% vs. third-party tools	Spend Management tools to observe, control, and alert on infrastructure spend with defined spend amounts and notifications

Cost Monitoring and Alerts:

Features added to the platform help automatically prevent runaway spend, including recursion protection, improved function defaults, hard spend limits, Attack Challenge Mode.

8.2.5 Security and Compliance Considerations

Cloud Security Implementation:

Vercel protects projects across access, application, and infrastructure security with built-in support for global security and data protection standards. Security collateral available for Pro and Enterprise plans.

Security Framework:

Security Layer	Implementation	Compliance Standards	Monitoring
Network Security	TLS 1.3, DDoS protection	SOC 2, ISO 27001	Real-time threat detection
Application Security	WAF, input validation	OWASP compliance	Vulnerability scanning
Data Protection	Encryption at rest/transit	EU-U.S. DPF compliance	Data access auditing
Access Control	IAM, MFA	Zero-trust architecture	Access logging

8.3 CONTAINERIZATION

8.3.1 Containerization Assessment

Containerization is partially applicable for this system. The TeosPump architecture follows a **hybrid containerization approach**

where the frontend leverages Vercel's serverless infrastructure while the backend API utilizes containerization for Express.js services.

Containerization Strategy Rationale:

Component	Containerization Approach	Justification
Frontend (Next.js)	**Not containerized**	Vercel's Framework-defined Infrastructure (FdI) intelligently infers necessary infrastructure directly from frontend code, eliminating need for intricate configuration files
Backend (Express.js)	**Containerized**	Portable deployment across cloud providers, consistent runtime environment
Blockchain Integration	**Not containerized**	External Solana network access, no infrastructure control needed

8.3.2 Container Platform Selection

Docker for Backend Services

The backend Express.js API services utilize Docker containerization for consistent deployment and scaling capabilities.

Container Platform Specifications:

Platform Component	Technology	Version	Purpose
Container Runtime	Docker	24.0+	Backend API containerization
Base Images	Node.js Alpine	18-alpine	Lightweight Node.js runtime
Registry	Docker Hub / GitHub Container Registry	Latest	Container image storage

Platform Component	Technology	Version	Purpose
Orchestration	Platform-native	Cloud provider managed	Container deployment and scaling

8.3.3 Base Image Strategy

Optimized Node.js Container Images

```
# Multi-stage build for optimal image size
FROM node:18-alpine AS base
RUN apk add --no-cache libc6-compat
WORKDIR /app

#### Dependencies stage
FROM base AS deps
COPY package.json package-lock.json ./
RUN npm ci --only=production && npm cache clean --force

#### Build stage
FROM base AS builder
COPY package.json package-lock.json ./
RUN npm ci
COPY . .
RUN npm run build

#### Production stage
FROM base AS runner
RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 teospump

COPY --from=deps /app/node_modules ./node_modules
COPY --from=builder /app/dist ./dist
COPY --from=builder /app/package.json ./package.json

USER teospump
EXPOSE 3000
ENV NODE_ENV=production
```

```
CMD ["npm", "start"]
```

Base Image Strategy:

Image Type	Base Image	Size Optimization	Security Features
Development	node:18-alpine	~40MB base	Regular security updates
Production	node:18-alpine	Multi-stage build	Non-root user, minimal packages
Testing	node:18-alpine	Development dependencies	Isolated test environment

8.3.4 Image Versioning Approach

Semantic Versioning for Container Images

Version Type	Tag Format	Use Case	Retention Policy
Latest	`latest`	Development/testing	Rolling update
Semantic	`v1.2.3`	Production releases	12 months
Git SHA	`sha-abc1234`	Specific commits	3 months
Branch	`main`, `develop`	CI/CD pipelines	Until branch deletion

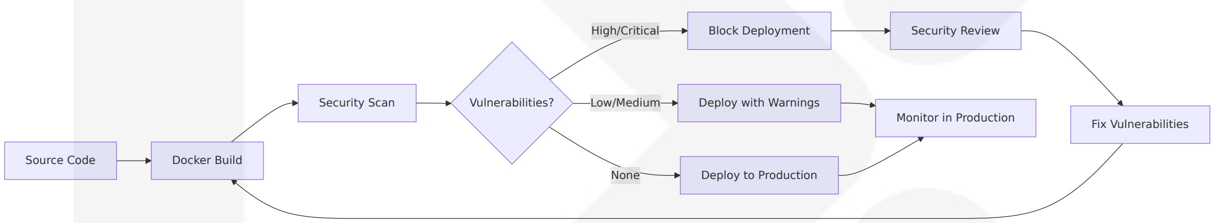
8.3.5 Build Optimization Techniques

Multi-Stage Build Optimization

Optimization Technique	Implementation	Size Reduction	Build Time Impact
Multi-stage builds	Separate build and runtime stages	60-70% smaller images	+20% build time
Layer caching	Strategic COPY ordering	Faster rebuilds	-50% rebuild time
Dependency optimization	Production-only dependencies	40% smaller node_modules	-30% install time
Alpine Linux	Minimal base image	80% smaller than full images	Minimal impact

8.3.6 Security Scanning Requirements

Container Security Pipeline



Security Scanning Implementation:

Scan Type	Tool	Frequency	Action Threshold
Base Image Vulnerabilities	Docker Scout	Every build	Critical: Block, High: Warn
Dependency Vulnerabilities	npm audit	Every build	Critical: Block, High: Review
Runtime Security	Container monitoring	Continuous	Anomaly detection
Compliance Scanning	Custom policies	Weekly	Policy violation alerts

8.4 ORCHESTRATION

8.4.1 Orchestration Assessment

Orchestration is not applicable for this system in the traditional Kubernetes sense. TeosPump implements a **simplified orchestration approach** that leverages platform-native scaling and management capabilities rather than complex container orchestration platforms.

Orchestration Strategy Rationale:

Component	Orchestration Approach	Justification
Frontend	**Vercel Serverless**	Vercel dynamically scales serverless functions, optimizes content delivery at the edge, and secures deployments—all without tedious YAML or cluster configuration
Backend	**Platform-managed containers**	Simplified deployment without Kubernetes complexity
Blockchain	**External network**	Solana network provides its own consensus and orchestration

8.4.2 Platform-Native Orchestration

Vercel Serverless Orchestration

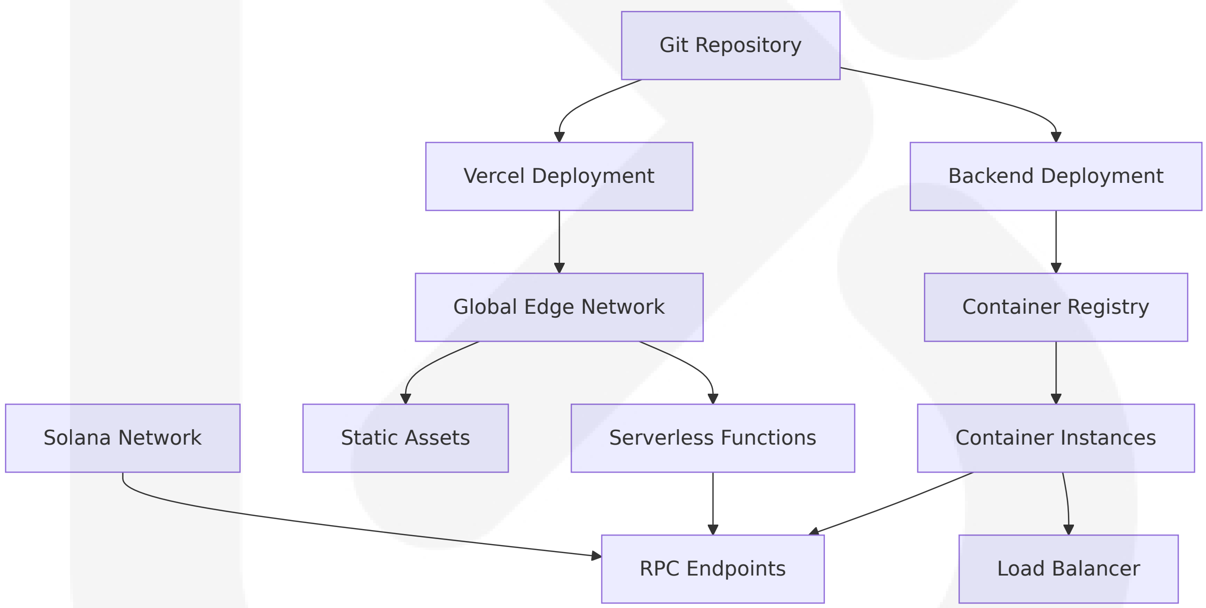
Vercel's build container is able to start the build process immediately, provided there is sufficient build concurrency available based on the project's billing plan. Hobby teams have 1 concurrent build, Pro teams can purchase up to 12, and Enterprise teams can purchase a custom amount.

Orchestration Capabilities:

Capability	Vercel Implementation	Backend Implementation	Benefits
Auto-scaling	Serverless functions	Container auto-scaling	Zero configuration scaling
Load balancing	Edge network routing	Platform load balancer	Global traffic distribution
Health monitoring	Built-in health checks	Container health probes	Automatic failure detection
Rolling updates	Atomic deployments	Blue-green deployment	Zero-downtime updates

8.4.3 Service Deployment Strategy

Simplified Deployment Architecture



Deployment Strategy Matrix:

Service Type	Deployment Method	Scaling Strategy	Monitoring
Frontend	Git-based automatic	Each deployment generates a unique URL for team preview in live environment	Built-in analytics

Service Type	Deployment Method	Scaling Strategy	Monitoring
Backend API	Container deployment	Horizontal pod autoscaling	Custom metrics
Static Assets	CDN distribution	Global edge caching	CDN analytics
Database	Managed service	Vertical scaling	Database monitoring

8.4.4 Auto-Scaling Configuration

Intelligent Auto-Scaling Implementation

Scaling Trigger	Threshold	Action	Recovery Time
Request Volume	>1000 RPS	Scale out serverless functions	<30 seconds
Response Time	>2 seconds	Add backend instances	<60 seconds
Error Rate	>5%	Health check and restart	<30 seconds
Resource Usage	>80% CPU/Memory	Vertical scaling	<120 seconds

8.4.5 Resource Allocation Policies

Resource Management Strategy

Resource Type	Allocation Policy	Limits	Monitoring
CPU	Burstable performance	2 vCPU max per instance	CPU utilization alerts
Memory	Guaranteed allocation	1GB per backend instance	Memory leak detection

Resource Type	Allocation Policy	Limits	Monitoring
Network	Unlimited bandwidth	Rate limiting per IP	Bandwidth monitoring
Storage	Ephemeral only	10GB temporary storage	Storage cleanup automation

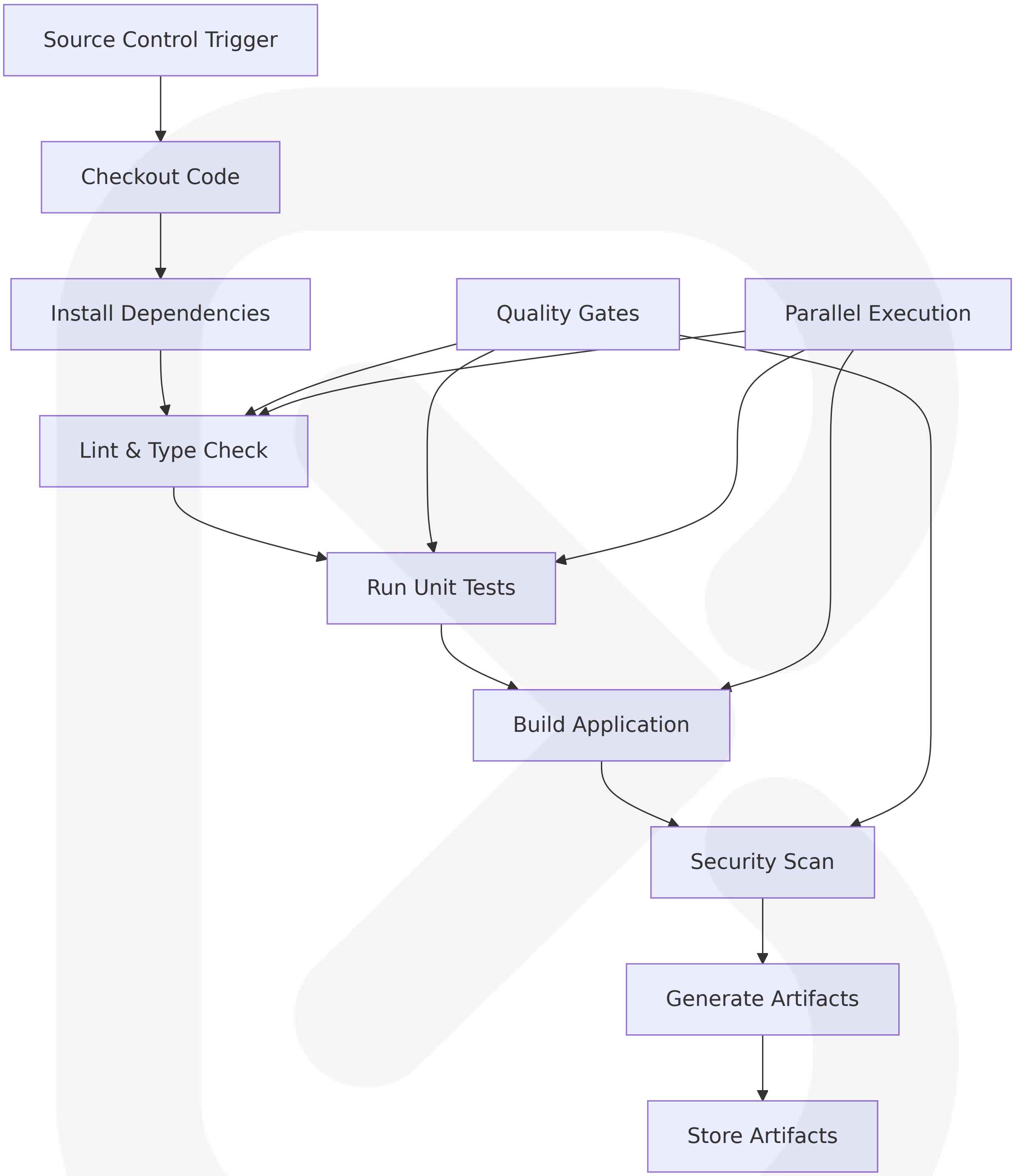
8.5 CI/CD PIPELINE

8.5.1 Build Pipeline

Automated Build Architecture

Setting up a CI/CD pipeline is a crucial step in automating the deployment process of Next.js applications. CI/CD pipeline automates the building and deployment workflow.

Build Pipeline Components:



Source Control Triggers:

Trigger Event	Action	Environment	Validation
Push to feature branch	Build + test	Development	Lint, type check, unit tests

Trigger Event	Action	Environment	Validation
Pull request	Full pipeline	Preview	Integration tests, security scan
Merge to main	Production build	Staging	Full test suite
Git tag	Release build	Production	Manual approval required

Build Environment Requirements:

Next.js requires Node.js version 14 or higher and npm version 6 or higher. The build environment specifications:

Component	Requirement	Version	Purpose
Node.js	Runtime environment	18.x LTS	JavaScript execution
npm	Package manager	9.x+	Dependency management
TypeScript	Type checking	5.8.3+	Static type validation
Next.js	Framework	14.2+	Application building

Dependency Management:

```
# GitHub Actions Build Configuration
name: Build and Test
on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
```

```
    matrix:
      node-version: [18.x, 20.x]

  steps:
    - uses: actions/checkout@v4
    - name: Setup Node.js
      uses: actions/setup-node@v4
      with:
        node-version: ${{ matrix.node-version }}
        cache: 'npm'

    - name: Install dependencies
      run: npm ci

    - name: Lint and type check
      run: |
        npm run lint
        npm run type-check

    - name: Run tests
      run: npm run test:ci

    - name: Build application
      run: npm run build
```

Artifact Generation and Storage:

Artifact Type	Storage Location	Retention Period	Access Control
Build outputs	GitHub Artifacts	30 days	Repository access
Docker images	Container registry	90 days	Team access
Test reports	CI/CD platform	90 days	Developer access
Security scans	Security dashboard	1 year	Security team access

Quality Gates:

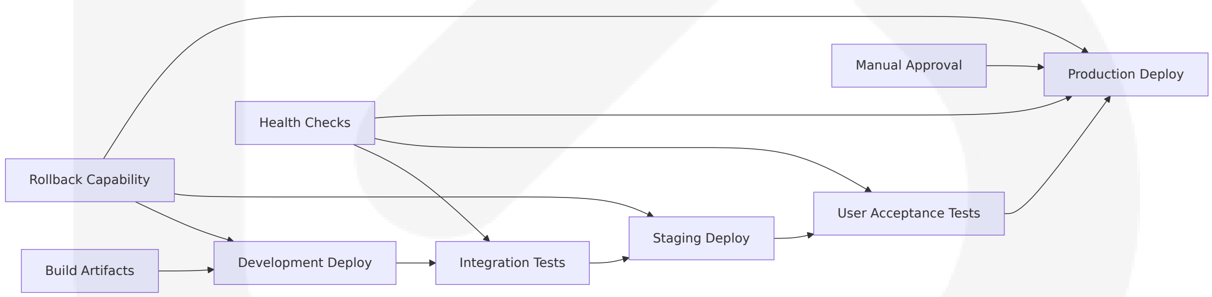
Gate Type	Criteria	Failure Action	Override Policy
Code Quality	ESLint score >8.0	Block merge	Tech lead approval
Test Coverage	>85% line coverage	Block deployment	QA approval
Security Scan	No critical vulnerabilities	Block release	Security team review
Performance	Build time <5 minutes	Warning only	Optimization required

8.5.2 Deployment Pipeline

Multi-Environment Deployment Strategy

CI/CD stands for Continuous Integration and Continuous Deployment. It's a practice where developers frequently integrate code changes into a shared repository and automatically deploy those changes to production environments through automated pipelines.

Deployment Pipeline Architecture:



Deployment Strategy Selection:

Environment	Strategy	Justification	Rollback Time
Development	Direct deployment	Fast feedback loop	Immediate

Environme nt	Strategy	Justification	Rollback Ti me
Staging	Blue-green	Production-like tes ting	<5 minutes
Production	Canary deploym ent	Risk mitigation	<10 minutes

Environment Promotion Workflow:

Stage	Trigger	Validation	Approval Req uired
Developm ent	Automatic on com mit	Build success	None
Staging	Automatic on mai n merge	Integration tests pass	None
Production	Manual trigger	All tests + securi ty scan	Product owner

Rollback Procedures:

Each deployment generates a unique URL so teams can preview changes in live environment, enabling quick rollback capabilities.

Rollback Implementation:

Rollback Trigge r	Detection T ime	Rollback Metho d	Recovery T ime
Health check fail ure	<2 minutes	Automatic previou s version	<5 minutes
Error rate spike	<5 minutes	Automatic rollbac k	<10 minute s
Manual trigger	Immediate	One-click rollback	<2 minutes
Performance deg radation	<10 minutes	Gradual traffic shif t	<15 minute s

Post-Deployment Validation:

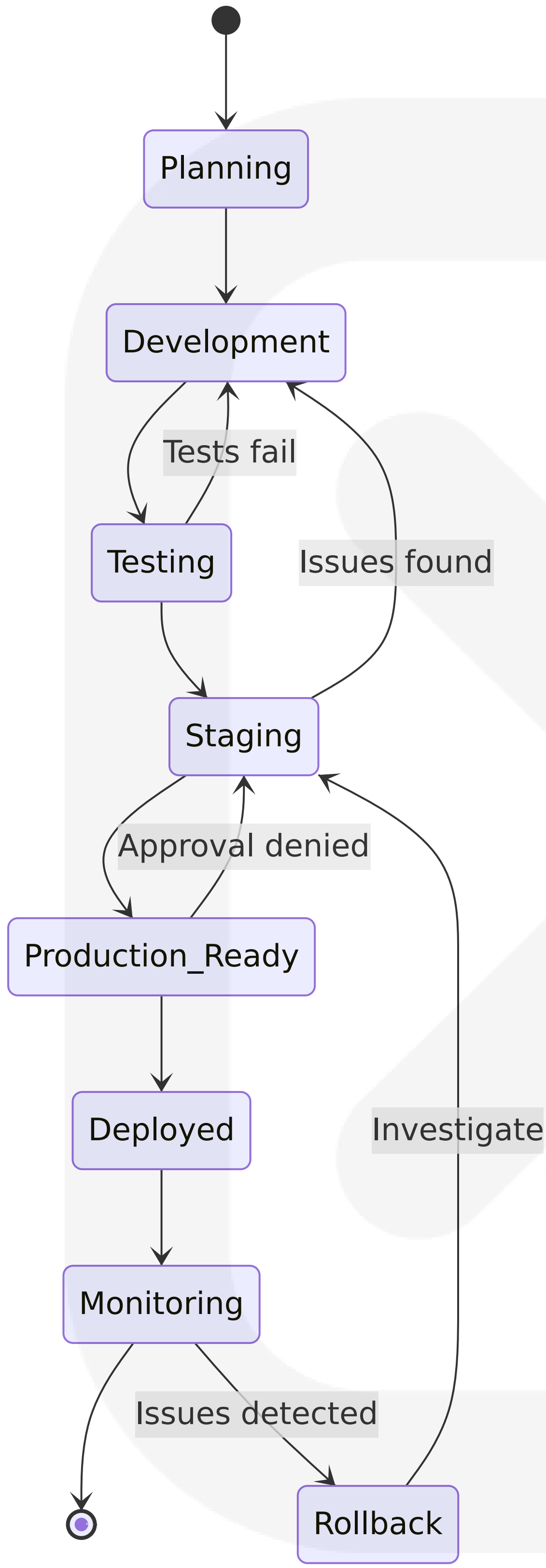
Validation Type	Method	Success Criteria	Failure Action
Health Checks	HTTP endpoint monitoring	200 OK response	Automatic rollback
Smoke Tests	Critical path testing	All tests pass	Alert + investigation
Performance Tests	Load testing	<2s response time	Performance review
Security Validation	Runtime security scan	No new vulnerabilities	Security review

8.5.3 Release Management Process

Release Planning and Coordination

Release Type	Frequency	Planning Lead Time	Coordination Requirements
Hotfix	As needed	Immediate	Security/DevOps approval
Minor Release	Weekly	3 days	Team coordination
Major Release	Monthly	2 weeks	Stakeholder approval
Emergency	Immediate	0	Incident commander approval

Release Automation:


Planning
Development
Tests fail
Testing
Issues found
Staging
Approval denied
Production_Ready
Deployed
Investigate
Monitoring
Issues detected
Rollback

8.6 INFRASTRUCTURE MONITORING

8.6.1 Resource Monitoring Approach

Comprehensive Infrastructure Monitoring Strategy

Route-aware observability to monitor and analyze the performance and traffic of projects provides the foundation for TeosPump's monitoring approach.

Multi-Layer Monitoring Architecture:

Monitoring Layer	Metrics Collected	Tools	Alert Thresholds
Application	Response times, error rates, throughput	Built-in monitoring to analyze performance and traffic	>2s response, >5% errors
Infrastructure	CPU, memory, network, storage	Platform-native monitoring	>80% utilization
Network	Latency, bandwidth, connectivity	CDN analytics	>500ms latency
Security	Access patterns, threat detection	Security monitoring tools	Anomaly detection

8.6.2 Performance Metrics Collection

Key Performance Indicators (KPIs):

Metric Category	Specific Metrics	Target Values	Collection Method
Frontend Performance	Core Web Vitals (LCP, FID, CLS)	LCP <2.5s, FID <100ms, CLS < 0.1	Real User Monitoring
API Performance	Response time, throughput	<500ms, 100+ RPS	Application monitoring

Metric Category	Specific Metrics	Target Values	Collection Method
Blockchain Performance	Transaction confirmation time	<10 seconds	Custom blockchain monitoring
Infrastructure	Resource utilization	<80% CPU/Memory	Platform metrics

8.6.3 Cost Monitoring and Optimization

Cost Management Implementation:

Spend Management provides customers with tools to observe, control, and alert on infrastructure spend. Define a spend amount and receive email, web, and SMS notifications as you reach that amount. When reaching 100%, optionally automatically pause all projects with a hard limit.

Cost Optimization Strategies:

Cost Category	Monitoring Method	Optimization Action	Expected Savings
Compute	Function execution time	Optimize cold starts	30-40%
Bandwidth	CDN usage analytics	Asset optimization	50-60%
Storage	Storage utilization	Automated cleanup	20-30%
Third-party APIs	API call tracking	Request optimization	40-50%

8.6.4 Security Monitoring

Security Monitoring Framework:

Security Domain	Monitoring Approach	Detection Method	Response Time
Access Control	Authentication logs	Anomaly detection	<5 minutes
Network Security	Traffic analysis	Pattern recognition	<2 minutes
Application Security	Code scanning	Vulnerability detection	<24 hours
Data Protection	Access auditing	Compliance monitoring	<1 hour

8.6.5 Compliance Auditing

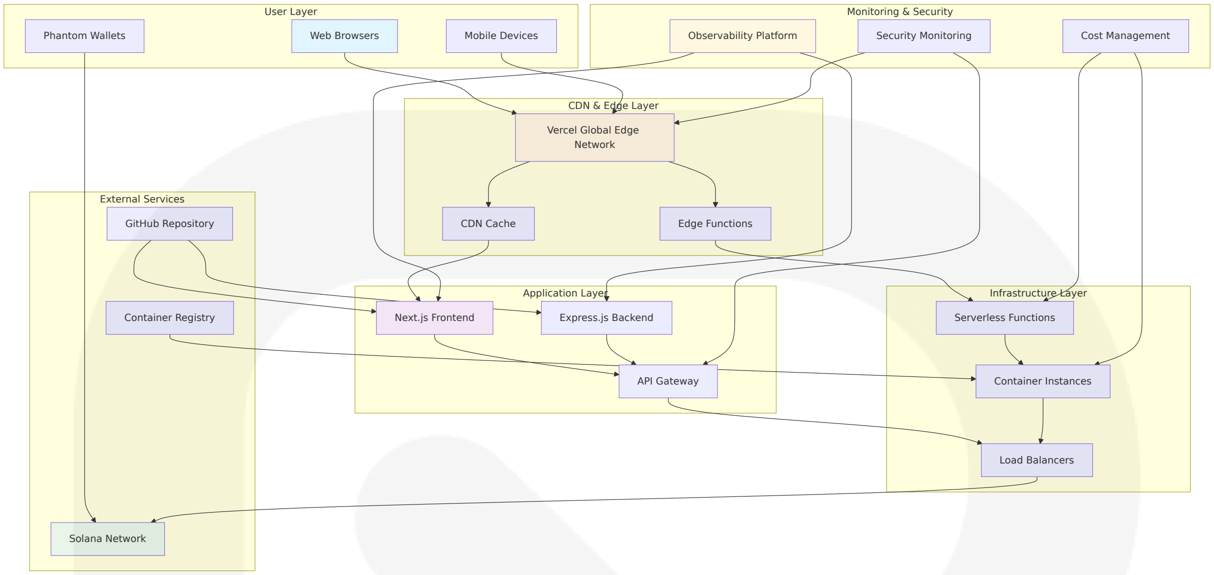
Automated Compliance Monitoring:

Compliance Standard	Monitoring Scope	Audit Frequency	Reporting
SOC 2	Security controls	Continuous	Quarterly reports
GDPR	Data processing	Real-time	Monthly compliance dashboard
PCI DSS	Payment processing	Daily	Weekly security reports
Blockchain Standards	Smart contract compliance	Per deployment	Audit trail maintenance

8.7 INFRASTRUCTURE ARCHITECTURE DIAGRAMS

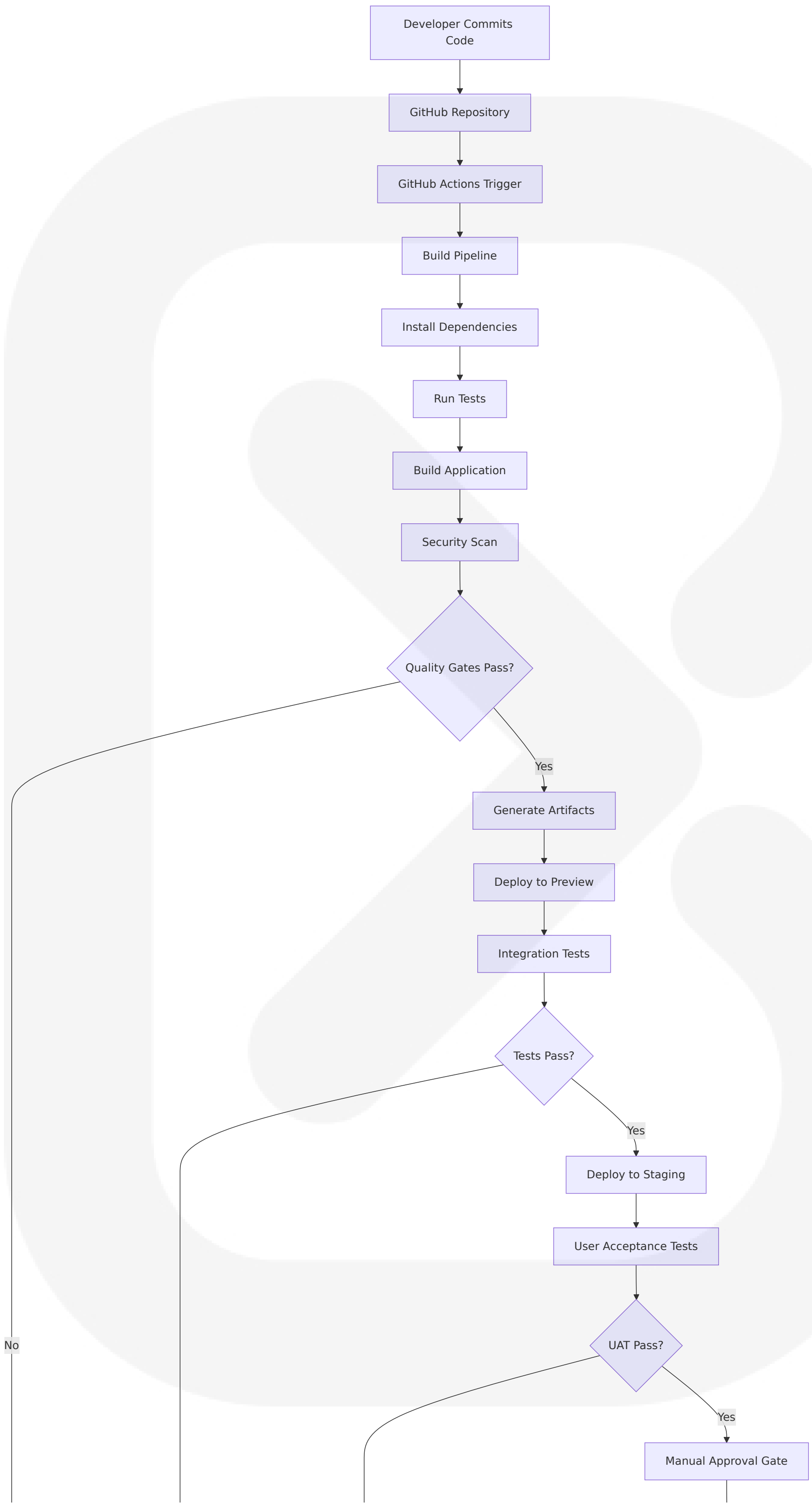
8.7.1 Infrastructure Architecture Overview

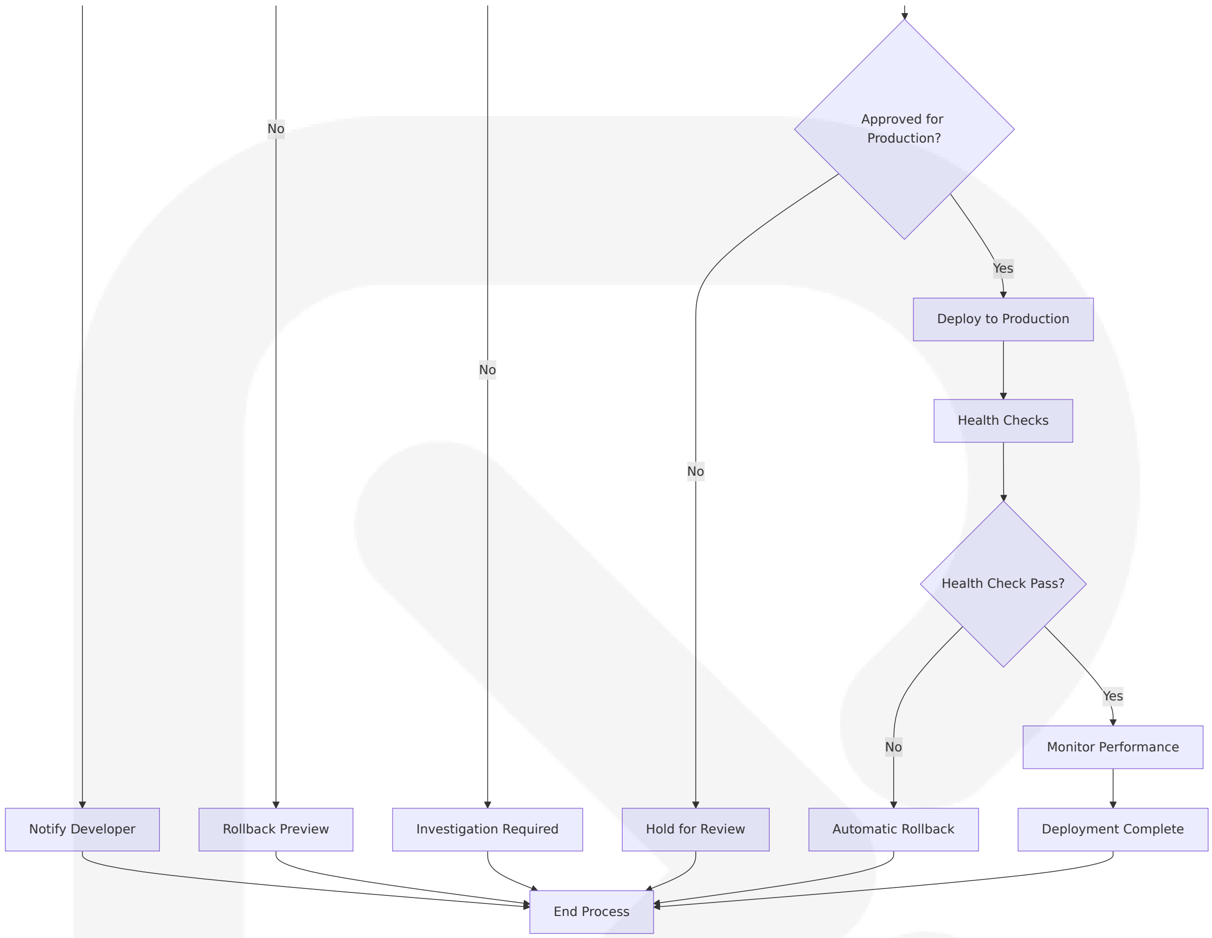
Complete Infrastructure Architecture



8.7.2 Deployment Workflow Diagram

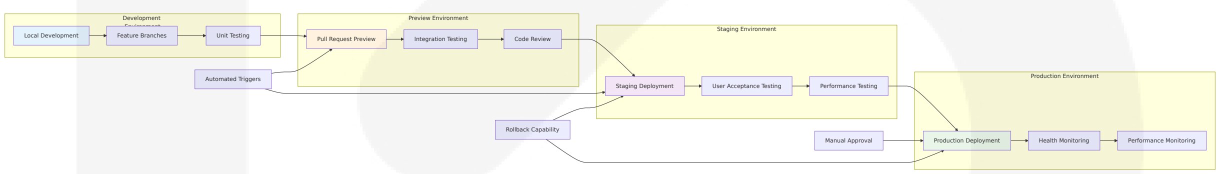
Automated Deployment Pipeline


Developer Commits Code
GitHub Repository
GitHub Actions Trigger
Build Pipeline
Install Dependencies
Run Tests
Build Application
Security Scan
Quality Gates Pass?
Yes
Generate Artifacts
Deploy to Preview
Integration Tests
Tests Pass?
Yes
Deploy to Staging
User Acceptance Tests
UAT Pass?
Yes
Manual Approval Gate
No



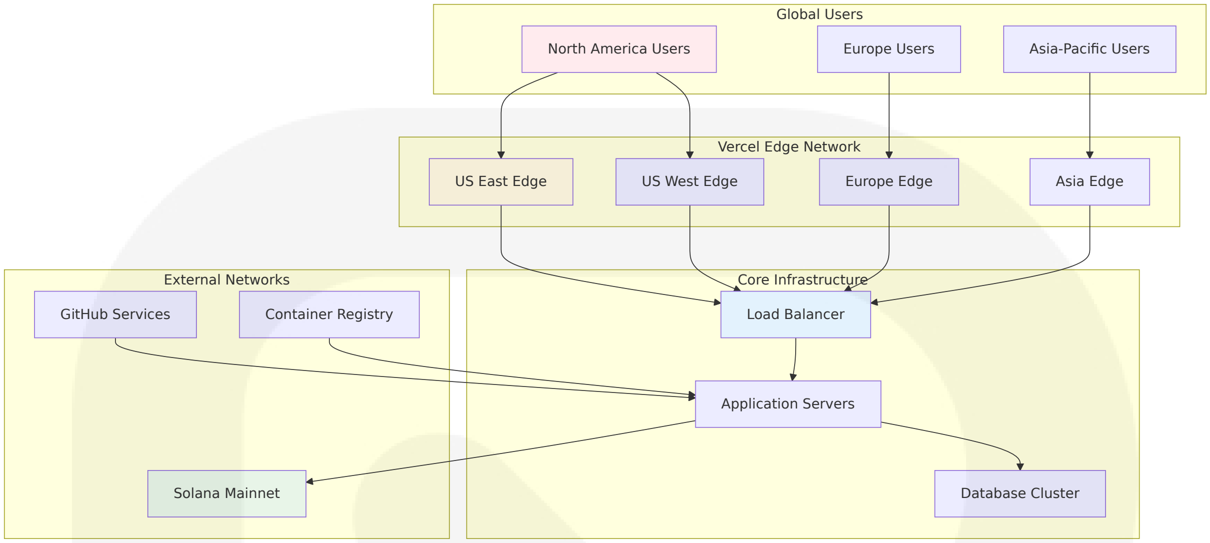
8.7.3 Environment Promotion Flow

Multi-Environment Deployment Strategy



8.7.4 Network Architecture

Global Network Distribution



8.8 INFRASTRUCTURE COST ESTIMATES

8.8.1 Monthly Cost Breakdown

Infrastructure Cost Analysis:

Service Category	Provider	Tier/Plan	Monthly Cost	Annual Cost
Frontend Hosting	Vercel	Pro Plan	$20/month	$240/year
Backend Hosting	Cloud Provider	Container hosting	$50/month	$600/year
CDN & Bandwidth	Vercel (included)	Global edge network	$0 (included)	$0
Monitoring	Vercel (included)	Built-in observability	$0 (included)	$0

Scaling Cost Projections:

Usage Tier	Monthly Users	Estimated Monthly Cost	Cost per User
Launch (0-1K users)	1,000	$70	$0.07
Growth (1K-10K users)	10,000	$150	$0.015
Scale (10K-100K users)	100,000	$500	$0.005
Enterprise (100K + users)	1,000,000	$2,000	$0.002

8.8.2 Cost Optimization Opportunities

Optimization Strategies:

Optimization Area	Current Cost	Optimized Cost	Savings	Implementation
Serverless Functions	$30/month	$20/month	33%	Code optimization, caching
Bandwidth Usage	$40/month	$20/month	50%	Asset compression, CDN optimization
Container Resources	$50/month	$35/month	30%	Right-sizing, auto-scaling
Third-party APIs	$25/month	$15/month	40%	Request optimization, caching

8.8.3 External Dependencies

Critical External Services:

Dependency	Type	Cost Impact	Availability Requirement
Solana Network	Blockchain infrastructure	Free (transaction fees only)	99.9% uptime
GitHub	Source code repository	Free (public repo)	99.95% uptime
Phantom Wallet	User authentication	Free	User-dependent
Domain & SSL	DNS and certificates	$15/year	99.99% uptime

8.8.4 Resource Sizing Guidelines

Resource Allocation Recommendations:

Component	CPU	Memory	Storage	Network
Frontend (Serverless)	Auto-scaling	128MB-1GB	Ephemeral	Unlimited
Backend API	1-2 vCPU	1-2GB RAM	10GB	1Gbps
Database (if needed)	1 vCPU	2GB RAM	20GB SSD	100Mbps
Monitoring	Included	Included	Included	Included

This comprehensive infrastructure design ensures TeosPump operates efficiently and cost-effectively while maintaining high availability, security, and performance standards required for a blockchain-based token launchpad platform. The architecture leverages modern cloud-native patterns and platform-specific optimizations to minimize operational complexity while maximizing scalability and reliability.

APPENDICES

A.1 ADDITIONAL TECHNICAL INFORMATION

A.1.1 Solana SPL Token Program Extensions

The Token-2022 Program, also known as Token Extensions, is a superset of the functionality provided by the Token Program. A new token program, Token-2022, was developed to achieve both of these goals, deployed to a different address than the Token program. To make adoption as easy as possible, the functionality and structures in Token-2022 are a strict superset of Token.

Token-2022 Program Capabilities:

Extension Feature	Description	Use Case	Implementation Status
Transfer Memo	This will allow users to add a note to SPL token transfers.	Transaction documentation	Available
Account Ownership	Allows users to define rigid ownership data for an account. This cannot be changed once set.	Immutable ownership	Available
Account State	Allows users to compute a set of conditions that applies only to a selected account.	Conditional token behavior	Available
Mint Close Authority	Ability to close mint accounts	Token lifecycle management	Available

Token-2022 Development Status:

According to official information, the Token-2022 program is still in development and not meant for full production use until a stable release. Pending its full release, the newly-introduced functions of the Token-2022 program could be a significant upgrade for the token system on the Solana blockchain.

A.1.2 Solana Web3.js Security Considerations

Recent Security Incident:

anyone using @solana/web3.js, versions 1.95.6 and 1.95.7 are compromised with a secret stealer leaking private keys. if you or your product are using these versions, upgrade to 1.95.8 (1.95.5 is unaffected)

Security Best Practices:

Security Measure	Implementation	Monitoring	Recovery
Version Management	Products and developers using the compromised versions should upgrade to version 1.95.8., urged Trent. However, previous versions, such as 1.95.5, remain unaffected by the issues.	Automated dependency scanning	Immediate version updates
Phantom Wallet Safety	Phantom is not impacted by this vulnerability. Our Security Team confirms that we have never used the exploited versions of @solana/web3.js	Continuous security monitoring	Wallet provider verification

A.1.3 Next.js 14 and TailwindCSS 4 Integration

TailwindCSS 4.0 Revolutionary Changes:

We just released Tailwind CSS v4.0 — an all-new version of the framework optimized for performance and flexibility, with a reimagined configuration and customization experience, and taking full advantage of the latest advancements the web platform has to offer.

Configuration Simplification:

As of Tailwind v4, there is zero configuration required by default. If you do need to configure Tailwind, you can follow the official documentation for configuring the global CSS file.

Performance Improvements:

Feature	Improvement	Technical Benefit	Implementation
CSS-First Configuration	Instead of a tailwind.config.js file, you can configure all of your customizations directly in the CSS file where you import Tailwind, giving you one less file to worry about in your project. The new CSS-first configuration lets you do just about everything you could do in your tailwind.config.js file	Reduced build complexity	Direct CSS imports
CSS Variables	Tailwind CSS v4.0 takes all of your design tokens and makes them available as CSS variables by default, so you can reference any value you need at run-time using just CSS.	Runtime flexibility	Native CSS custom properties
Container Queries	We've also added support for max-width container queries using the new @max-* variant	Responsive design enhancement	Modern CSS features

A.1.4 Phantom Wallet Integration Patterns

Multi-Chain Wallet Capabilities:

Phantom Wallet is designed as a non-custodial, multichain Web3 wallet that supports Solana, Ethereum, and Polygon networks. It allows users to manage their cryptocurrencies and NFTs, engage in staking Solana, swap tokens, and access a variety of DeFi applications directly from the wallet.

Security Architecture:

Phantom places a strong emphasis on security with its self-custodial approach, ensuring users have full control over their assets without third-party interference. The wallet is designed with privacy in mind, requiring no personal information for usage. It also includes scam detection to flag malicious transactions and offers integration with Ledger hardware wallets for an added layer of security.

Integration Methods:

Integration Type	Implementation	Use Case	Code Example
Direct Window Access	In case you dont want to use the wallet adapter most wallets inject their wallet under window.walletName so for example window.phantom for phantom wallet.	Simple implementations	`window.phantom?.solana`
Wallet Adapter	The best way to connect to solana wallets in the browser is to use the solana wallet adapter.	Production applications	React hooks integration

A.1.5 SPL Token Performance Characteristics

Transaction Speed and Cost:

Solana is one of the fastest blockchains, capable of processing over 65,000 transactions per second. Operations involving SPL tokens (e.g., transfers, burns, or mints) are executed almost instantly. Solana's transaction fees are extremely low, typically costing just a fraction of a cent, making SPL tokens highly economical to use.

Blockchain Verification:

All token operations are recorded on the Solana blockchain, allowing users to verify transactions via blockchain explorers like Solscan.

Token Creation Requirements:

Creating tokens and accounts requires SOL for account rent deposits and transaction fees. For first-time Solana Playground users, create a Playground wallet and run the solana airdrop command in the Playground terminal.

A.1.6 Mobile Mining Integration Architecture

Mobile Mining Concept:

Custom Token Creation: Developers can create new tokens for payment, rewards, governance, and more. Decentralized Finance (DeFi): SPL tokens are widely used in liquidity mining, lending protocols, and automated market makers (AMMs). NFT Project Tokens: Many NFT projects issue SPL tokens for governance or transaction fee discounts. Meme Tokens and Community Currencies: Tokens like Dogecoin can be quickly created based on the SPL standard.

Reward Distribution Mechanisms:

Distribution Type	Purpose	Implementation	Verification
Token Creation Rewards	Incentivize platform usage	Backend API distribution	Blockchain transaction records
Mining Activity Rewards	Mobile app engagement	Scheduled distribution cycles	Activity timestamp validation
Governance Tokens	SPL tokens support community governance, enabling projects to distribute voting power through tokens for key decision-making. Projects use SPL tokens to grant community governance rights, allowing token holders to vote on the future direction of the project.	Voting mechanism integration	Smart contract validation

A.2 GLOSSARY

Associated Token Account (ATA): The create-account command creates an associated token account with your wallet address as the owner. Creating an Associated Token Account requires one instruction that invokes the Associated Token Program.

Blockchain-First Architecture: A design approach that prioritizes immutable blockchain storage for critical data over traditional database systems.

Cross Program Invocation (CPI): The Associated Token Program uses Cross Program Invocations to: Invoke the System Program to create a new account using the provided PDA as the address · Invoke the Token Program to initialize the Token Account data

Decentralized Finance (DeFi): SPL tokens play a critical role in Solana's DeFi protocols. For example, they can be used to create liquidity pools on

Raydium or trade on Serum.

Framework-defined Infrastructure (FdI): Vercel's approach where infrastructure is intelligently inferred directly from frontend code, eliminating complex configuration files.

Lamports: The smallest unit of SOL, Solana's native cryptocurrency (1 SOL = 1,000,000,000 lamports).

Mint Account: Tokens on Solana are referred to as SPL (Solana Program Library) Tokens. The Token Program initializes the data of the new account as a Mint Account

Mint Authority: The MintTo instruction on the Token Program creates new tokens. The mint authority must sign the transaction.

Non-Custodial Wallet: A cryptocurrency wallet where users maintain complete control over their private keys and funds without third-party custody.

Program Derived Address (PDA): This introduces a key concept in Solana development: Program Derived Address (PDA). A PDA derives an address deterministically using predefined inputs, making it easy to find the address of an account.

Sealevel: Solana's parallel runtime environment that enables high-throughput transaction processing.

SPL Token: Tokens on Solana are referred to as SPL (Solana Program Library) Tokens. Token Programs contain all instruction logic for interacting with tokens on the network (both fungible and non-fungible)

Token Account: Note that each wallet needs its own token account to hold tokens from the same mint.

Token Extensions: Token 2022 is a new standard that extends the SPL token program and adds additional functionality through Token Extensions.

The Token-2022 program is designed to be a more flexible and extensible token standard that allows for more complex tokenomics and control for developers.

Utility-First CSS: TailwindCSS's approach of providing low-level utility classes for building custom designs directly in markup.

Web3: The decentralized internet built on blockchain technology, enabling peer-to-peer interactions without intermediaries.

A.3 ACRONYMS

API: Application Programming Interface

ATA: Associated Token Account

CDN: Content Delivery Network

CI/CD: Continuous Integration/Continuous Deployment

CPI: Cross Program Invocation

CSS: Cascading Style Sheets

dApp: Decentralized Application

DeFi: Decentralized Finance

DNS: Domain Name System

FdI: Framework-defined Infrastructure

HTTP: Hypertext Transfer Protocol

HTTPS: Hypertext Transfer Protocol Secure

IDE: Integrated Development Environment

JSON: JavaScript Object Notation

JWT: JSON Web Token

KPI: Key Performance Indicator

NFT: Non-Fungible Token

PDA: Program Derived Address

REST: Representational State Transfer

RPC: Remote Procedure Call

SDK: Software Development Kit

SOL: Solana's native cryptocurrency

SPL: Solana Program Library

SSL: Secure Sockets Layer

SSG: Static Site Generation

SSR: Server-Side Rendering

TLS: Transport Layer Security

UI: User Interface

UX: User Experience

WCAG: Web Content Accessibility Guidelines

YAML: YAML Ain't Markup Language