- **PROJECT NAME: Crowdfunding Marketplace**
- **PHASE: Final Project**
- **DATE: 09/02/2026**
- **PREPARED BY Nugmash Bigali, Abdibay Asylbek, Tugelbayev Aidyn**
- **Team name: FutureTraders**
- **Git: https://github.com/bgl96395/Crowdfunding_Marketplace.git**

VERSION HISTORY

VERSION #	IMPLEMENTED BY	REVISION DATE	APPROVED BY	APPROVAL DATE	REASONS
1	FutureTraders	26.01.2026	FutureTraders	08.02.2026	Crutial initial documentation

ASTANA IT UNIVERSITY

1. Project Plan

1.1 DESCRIPTION/PURPOSE OF PROJECT:

1. Develop a fully functional crowdfunding application that operates on Ethereum test networks, enabling users to create and fund campaigns using blockchain technology. Also users can invest their money, and get tokens as gifts from us.
2. Demonstrate comprehensive understanding of smart contract development and ethers integration and crowd platform architecture through practical implementation.
3. Implement ERC-20 token standard for rewarding campaign contributors, showcasing tokenization concepts and blockchain-based incentive mechanisms.

1.2 SCOPE OF PROJECT:

1. **Smart Contract Development:** Create Solidity smart contracts for campaign management and ERC-20 reward token implementation with comprehensive testing.
2. **Frontend Development:** Build an intuitive web interface using HTML, CSS, and JavaScript with Ethers.js for blockchain interaction and MetaMask wallet integration.
3. **Deployment and Testing:** Deploy smart contracts to Ethereum test networks (Hardhat) and conduct thorough testing of all functionalities.
4. **Documentation:** Provide comprehensive technical documentation including architecture overview, deployment instructions, and user guides.

1.3 TIMELINE AND METHODOLOGY

DATE	MILESTONES	GOAL	DEPENDENCES	RESOURCES	OUTCOME
26/01/2026-31/01/2026 **Week 1**	Smart Contract Development, Frontend Development	Implement core contracts, Build user interface	Solidity, OpenZeppelin, Deployed contracts	Hardhat, HTML, CSS, JS, Ethers.js	Working contracts Functional UI
01/02/2026-08/02/2026 **Week 2**	Integration & Testing Documentation	Connect frontend to blockchain. Complete all deliverables	Working application and tested	Test ETH, MetaMask Github, README.md line by line - step by step	Integrated marketplace Production ready project

MILESTONE	RESPONSIBLE	STATUS
Contract Deployment	Nugmash Bigali	Completed
UI Implementation	Tugelbayev Aidyn	Completed
Testing Complete	Abdibay Asylbek	Completed
Documentation	All Team Members	Completed

2. Project Description

2.1 PROJECT STAKEHOLDER SCENARIOS

Campaign Creators: - Ability to create crowdfunding campaigns with specific goals and deadlines - Track funding progress and contributor information in real-time - Finalize campaigns and receive collected funds upon successful completion.

Backers: - Browse and discover active crowdfunding campaigns -- Receive ERC-20 reward tokens proportional to their contributions - View their contribution history and token balances - Track campaign progress and deadlines.

Development our Team: - Demonstrate mastery of blockchain development skills - Successfully complete the Blockchain 1 course final project requirements - Build portfolio-ready decentralized application.

2.2 CONSTRAINTS & RESTRICTIONS

CONSTRAINT	ISSUE/SOLUTION
Test Network Only	**Issue**: Cannot use Ethereum mainnet per project requirements. **Solution**: Deploy on Hardhat test networks with free test ETH from faucets
No Real Cryptocurrency	**Issue**: Academic project constraint prohibits real financial transactions. **Solution**: Use only test tokens with educational disclaimer and clearly labeled test environment.

Smart Contract Immutability	**Issue:** Cannot modify deployed contracts after deployment. **Solution:** Testing before deployment, version control, and potential redeployment for fixes.
MetaMask Requirement	**Issue:** Users must have MetaMask installed. **Solution:** Provide clear installation instructions and compatibility warnings.

2.3 ASSUMPTIONS & DEPENDENCIES

Assumptions

-Users have MetaMask wallet installed and configured

-Users have access to test ETH from faucets

-Users have basic understanding of blockchain and cryptocurrency concepts

-Modern web browser with JavaScript enabled is available

-Users understand this is an educational project using test tokens only

Dependencies

- Ethereum test network availability and stability
- MetaMask browser extension functionality and API compatibility
- OpenZeppelin library for secure smart contract implementation
- Hardhat development environment for compilation, testing, and deployment
- Ethers.js library for blockchain interaction
- Node.js runtime environment for development tools
- Test ETH faucet availability for user onboarding

-ERC–20 standard

3. Project Requirements

3.1 USER REQUIREMENTS

Users must be able to connect their MetaMask wallet to the application

Users must be able to create new crowdfunding campaigns with title, funding goal (in test ETH), and deadline

Users must be able to view all active campaigns with their details (progress, time remaining, contributor count)

Users must be able to contribute test ETH to any active campaign

Users must receive ERC-20 reward tokens automatically upon contribution

Users must be able to view their test ETH and reward token balances

Campaign creators must be able to finalize their campaigns after the deadline

3.2 FUNCTIONAL REQUIREMENTS

Campaign Management: - Create campaigns with customizable parameters (title, goal amount, deadline timestamp) - Store campaign data on blockchain

with immutable records - Track total contributions and number of unique contributors per campaign - Enforce campaign deadlines and prevent contributions after expiration - Allow campaign creators to finalize campaigns and withdraw funds - Emit events for all campaign-related activities for frontend tracking

Contribution System: - Accept test ETH contributions from any wallet address - Record individual contribution amounts for each contributor per campaign - Update campaign total in real-time upon contribution confirmation - Prevent contributions to finalized or expired campaigns - Return excess contributions if goal is exceeded (optional feature) - Maintain transparent contribution history on blockchain

Token Reward System: - Automatically mint ERC-20 tokens upon successful contribution - Calculate token amount proportional to contribution size (e.g., 1:1 or 10:1 ratio) - Transfer tokens to contributor's wallet address immediately - Support standard ERC-20 token operations (transfer, balance query, allowance) - Implement total supply tracking and minting controls - Provide token metadata (name, symbol, decimals)

3.3 SYSTEM REQUIREMENTS & SOFTWARE INTERFACES

System Requirements: - Node.js version 16.x or higher for development environment - Modern web browser (Chrome, Firefox, Brave, Edge) with JavaScript enabled - MetaMask browser extension version 10.x or higher - Internet connection for blockchain network access - Minimum 4GB RAM for development environment - 500MB free disk space for dependencies and build files

Software Interfaces: - MetaMask Wallet API - For wallet connection, account management, and transaction signing - Ethereum JSON-RPC API - For blockchain data queries and transaction submission - Ethers.js Library (v6.x) -

For Web3 provider instantiation and contract interaction - Hardhat Network Interface - For local testing, deployment scripts, and network configuration -

```
PS C:\Users\User\crowd_funding> npx hardhat node
Started HTTP and WebSocket JSON-RPC server at http://127.0.0.1:8545/
```

```
PS C:\Users\User\crowd_funding> npx hardhat run scripts/deploy.js --network localhost
CrowdFunding deployed to: 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512
Token deployed to: 0x5FbDB2315678afecb367f032d93F642f64180aa3
```

OpenZeppelin Contracts (v5.x) - For secure ERC-20 token implementation and access control - Ethereum Test Network RPC - Sepolia or Holesky network endpoints for deployment

3.4 USER INTERFACE REQUIREMENTS

Clean and intuitive design following modern web design principles

Responsive layout compatible with desktop (1920x1080) and mobile devices (375x667 minimum)

Clear wallet connection status indicator with connected address display

Network detection with prominent warning for incorrect networks

Campaign cards displaying:

– Campaign title and description

– Funding goal and current amount raised

– Progress bar with percentage

– Deadline with countdown timer

– Number of contributors

– Contribute button

• Form for creating new campaigns with input validation:

– Title field (required, max 100 characters)

– Goal amount field (required, positive number)

– Deadline date picker (required, future date)

• Contribution interface with:

– Amount input field with balance display

– Estimated reward tokens calculation

– Gas fee estimate

– Confirm and cancel buttons

Real-time transaction status indicators:

Pending state with loading animation

Success confirmation with transaction hash

Error messages with user-friendly explanations

Balance display section showing:

Test ETH balance

Reward token balance

Total contributions made

Error messages and user guidance for common issues:

Wrong network selected

Insufficient balance

Rejected transactions

Campaign not found

Expired campaigns

3.5 WORKFLOW AND ACTIVITIES

User Onboarding Workflow:

1. User installs MetaMask browser extension from official website

2. User creates new wallet or imports existing wallet using seed phrase

3. User secures wallet with strong password and backs up recovery phrase

4. User switches MetaMask to Ethereum test network Hardhat

5. User visits test ETH faucet and requests free test tokens

6. User navigates to Crowdfunding Marketplace application URL

7. User clicks “Connect Wallet” button on homepage

8. MetaMask popup appears requesting connection permission

9. User approves connection request in MetaMask

10. Application displays connected wallet address and balance

ASTANA IT UNIVERSITY

Hive

Description: The beehive design that makes honey harvesting easier.

Owner: 0x2546BcD3c84621e976D8185a91A922aE77ECEc30

Target: 0.00000000045 ETH

Collected: 0.000000003 ETH

Deadline: 29.03.2026

Status: Active

ETH

Donate

Show Donators

Donators:

0x2546BcD3c84621e976D8185a91A922aE77ECEc30: 3e-9 ETH

ASTANA IT UNIVERSITY

Hive

Description: The beehive design that makes honey harvesting easier.

Owner: 0x2546BcD3c84621e976D8185a91A922aE77ECEc30

Target: 0.00000000045 ETH

Collected: 0.000000003 ETH

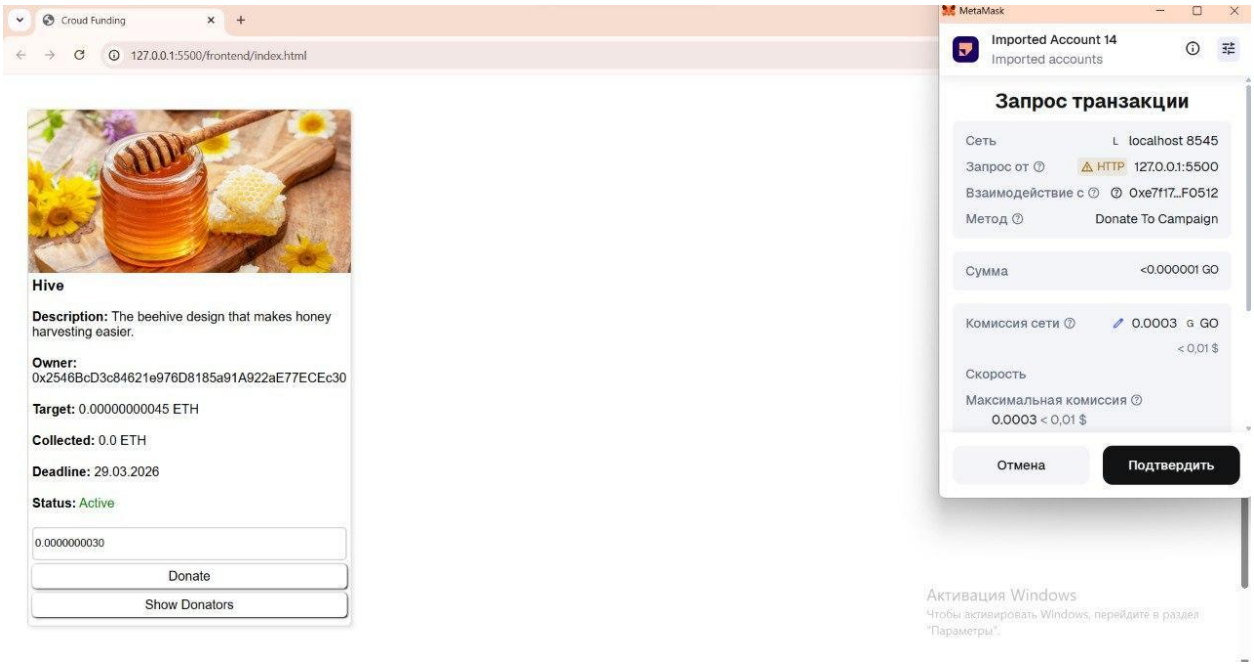
Deadline: 29.03.2026

Status: Active

ETH

Donate

Show Donators


Croud Funding
127.0.0.1:5500/frontend/index.html
Hive
Description: The beehive design that makes honey harvesting easier.
Owner: 0x2546BcD3c84621e976D8185a91A922aE77ECEc30
Target: 0.00000000045 ETH
Collected: 0.0 ETH
Deadline: 29.03.2026
Status: Active
0.0000000030
Donate
Show Donators
MetaMask
Imported Account 14
Imported accounts
Запрос транзакции
Сеть
localhost 8545
Запрос от
HTTP 127.0.0.1:5500
Взаимодействие с
0xe7f17...F0512
Метод
Donate To Campaign
Сумма
<0.000001 GO
Комиссия сети
0.0003 GO
< 0,01 $
Скорость
Максимальная комиссия
0.0003 < 0,01 $
Отмена
Подтвердить
Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

ASTANA IT UNIVERSITY

Hive

Description: The beehive design that makes honey harvesting easier.

Owner: 0x2546BcD3c84621e976D8185a91A922aE77ECEc30

Target: 0.00000000045 ETH

Collected: 0.0 ETH

Deadline: 29.03.2026

Status: Active

0.0000000030

Donate

Show Donators

Hive

Description: The beehive design that makes honey harvesting easier.

Owner: 0x2546BcD3c84621e976D8185a91A922aE77ECEc30

Target: 0.00000000045 ETH

Collected: 0.0 ETH

Deadline: 29.03.2026

Status: Active

ETH

Donate

Show Donators

Croud Funding

127.0.0.1:5500/frontend/index.html

Campaign

Title

Hive

Description

The beehive design that makes honey harvesting easier.

Image URL

https://cdn.mos.cms.futurecdn.net/zcz8f72orNC9GKgm3tbqMYj

Target amount

450000000

Deadline

29.03.2026

Create Campaign

Create your Campaign to raise funds

MetaMask

Imported Account 14

Imported accounts

Запрос транзакции

Сеть localhost 8545

Запрос от HTTP 127.0.0.1:5500

Взаимодействие с 0xe7f17...F0512

Метод Create Campaign

Комиссия сети 0.0004 GO

< 0,01 $

Скорость

Максимальная комиссия

0.0004 < 0,01 $

Одноразовый код 0

Отмена

Подтвердить

ASTANA IT UNIVERSITY

Campaign

Title

Hive

Description

The beehive design that makes honey harvesting easier.

Image URL

https://cdn.mos.cms.futurecdn.net/zcz8f72orNC9GKgm3tbqMY.jp

Target amount

450000000

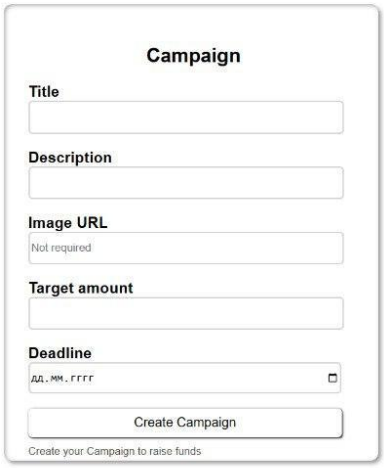
Deadline

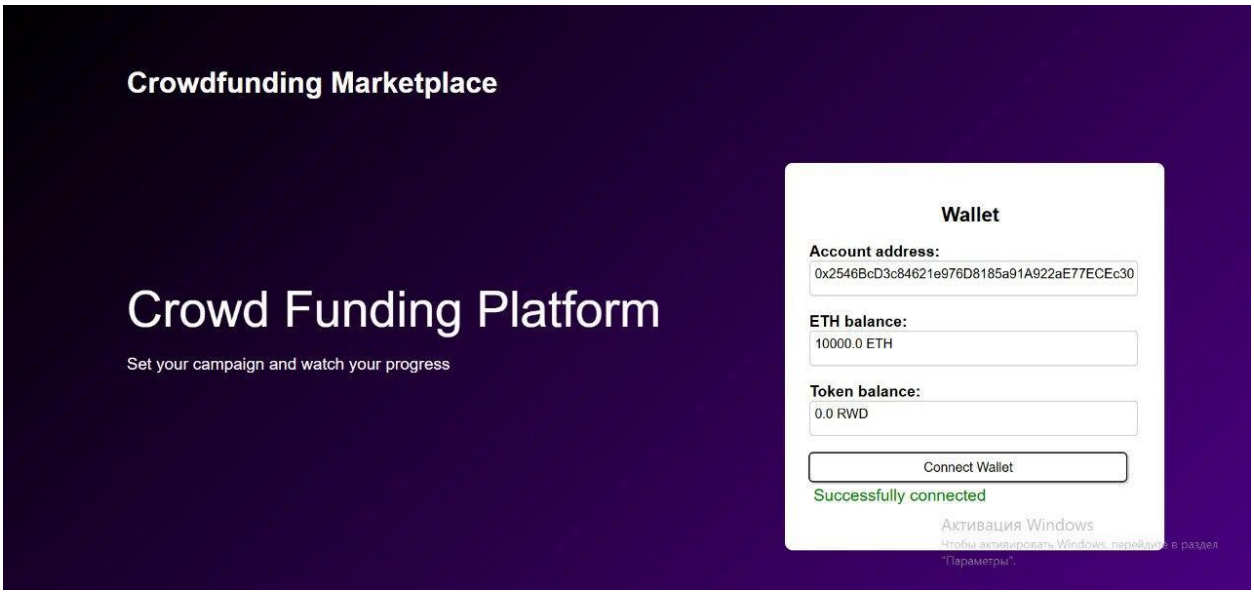
29.03.2026

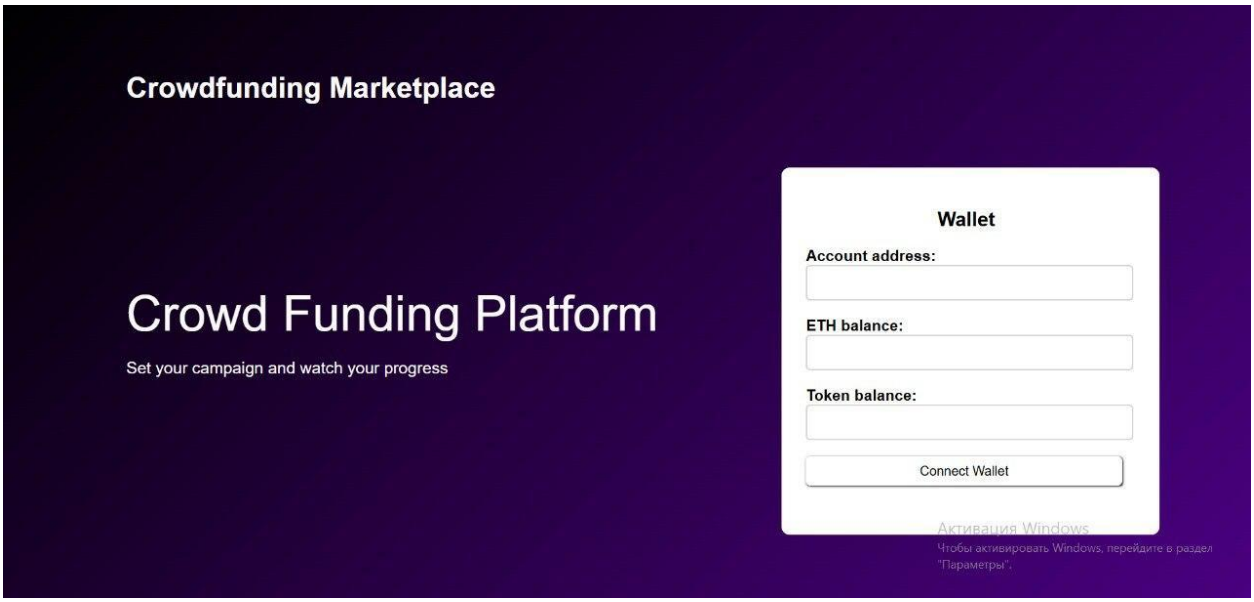
Create Campaign

Create your Campaign to raise funds

ASTANA IT UNIVERSITY







3.3 Campaign Creation Workflow:

1. Connected user clicks “Create Campaign” button in navigation
2. Create Campaign form modal/page opens
3. User fills in campaign title
4. User enters funding goal in ETH (e.g., “5”)
5. User selects campaign deadline using date picker
6. Frontend validates all inputs for completeness and correctness
7. User clicks “Create Campaign” submit button

8. Frontend calls smart contract createCampaign function

9. MetaMask popup appears showing transaction details and gas fee

10. User reviews and confirms transaction in MetaMask

11. Transaction is submitted to blockchain network

12. Frontend shows pending status with loading indicator

13. Transaction is mined and confirmed

14. Smart contract emits CampaignCreated event

15. Frontend listens for event and updates UI

16. New campaign appears in campaign list

17. User receives success notification with campaign ID

3.4 Contribution Workflow:

1. User browses list of active campaigns on homep age

2. User clicks on campaign card to view details

3. Campaign details page displays full information

4. User enters contribution amount in ETH input field

5. Frontend calculates estimated reward tokens to be received

6. Frontend displays current gas fee estimate

7. Frontend validates user has sufficient balance

8. User clicks “Contribute” button

9. Frontend calls smart contract contribute function with ETH value

10. MetaMask popup appears requesting transaction approval

11. User reviews amount, gas fee, and total in MetaMask

12. User confirms transaction in MetaMask

13. Transaction is submitted to blockchain

14. Frontend displays pending transaction status

15. Smart contract processes contribution and mints reward tokens

16. Transaction is confirmed on blockchain

17. Frontend listens for ContributionReceived and TokensMinted events

18. Campaign progress bar updates with new total

19. User’s balances update (ETH decreased, reward tokens increased)

20. User receives success notification with transaction hash link

3.6 CHANGE MANAGEMENT

All code changes tracked through Git version control system with meaningful commit messages following conventional commit standards

Regular commits to GitHub repository demonstrating incremental development progress (minimum one commit per major feature)

Branch-based development workflow: main branch for stable code, feature branches for new development

Pull requests with code review process before merging to main branch

Semantic versioning for releases (v1.0.0, v1.1.0, etc.)

Smart contract upgrades require new deployment with updated contract address due to immutability

Frontend changes deployed through updated static files on hosting platform (GitHub Pages, Vercel, Netlify)

Configuration changes (contract addresses, network settings) managed through environment files (.env)

Team collaboration through GitHub with issue tracking for bugs and feature requests

Documentation updated alongside code changes to maintain accuracy and completeness

Deployment checklist for production releases ensuring all tests pass and documentation is current

4. High-Level Tech Architecture

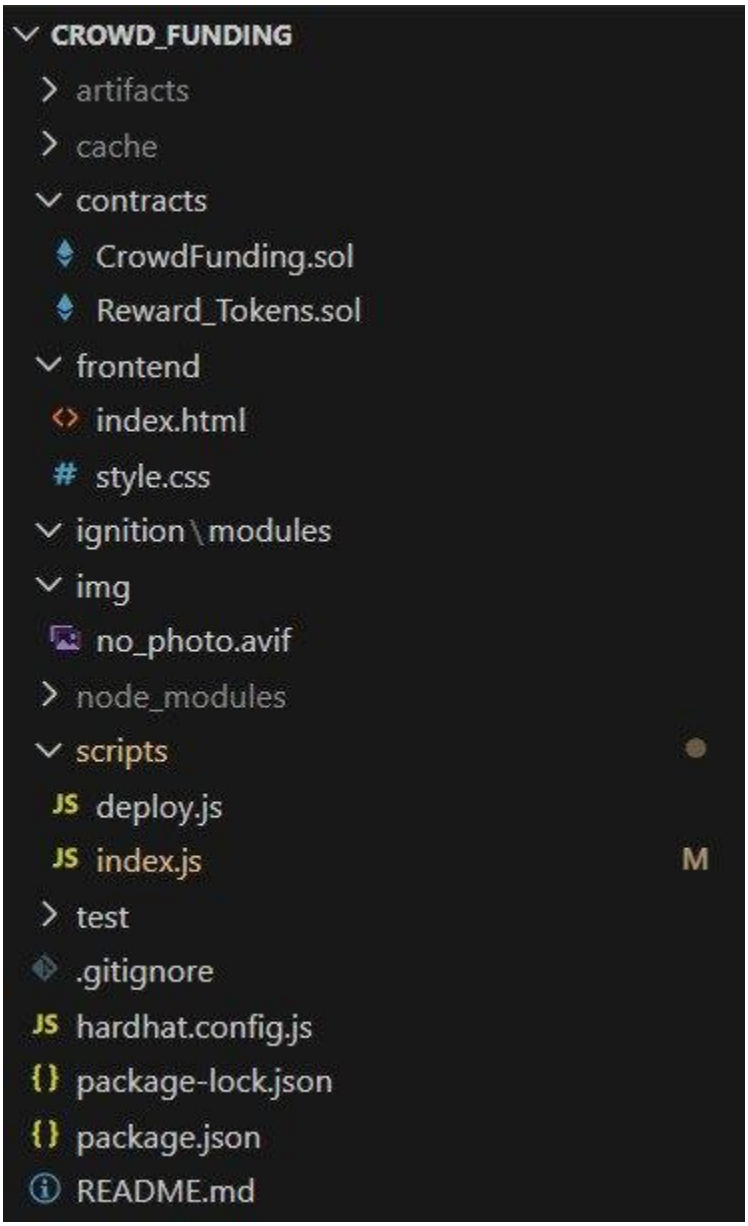
Frontend Layer: - HTML5 - Semantic markup and page structure following accessibility standards - CSS3 - Styling, layout with Flexbox/Grid, and responsive design with media queries - JavaScript - Client-side logic, DOM manipulation, asynchronous operations - Ethers.js v6.x - provider instantiation, blockchain interaction library, contract ABI handling

Blockchain Layer: - **Solidity v0.8.20+** - Smart contract programming language with latest security features - **OpenZeppelin Contracts v5.x** - Secure, audited contract libraries for ERC-20 implementation and access control.

Development Tools: - **Node.js v16+** - JavaScript runtime environment for development tools - **Hardhat** - Ethereum development environment for: - Smart contract compilation - Automated testing with Chai and Mocha - Deployment scripts - Local blockchain simulation - Contract verification - **MetaMask** - Browser wallet for: - Account management - Transaction signing - Network switching - Token management

Architecture Pattern: The application follows a decentralized application (DApp) architecture: - **Frontend (Client-Side):** Static HTML/CSS/JS files served from web hosting - **Backend (Blockchain):** Smart contracts deployed on Ethereum providing immutable business logic - **Communication Layer:** Ethers.js library bridges frontend and blockchain via JSON-RPC - **State Management:** Blockchain serves as single source of truth for all application state - **Event-Driven Updates:** Smart contract events trigger frontend UI updates - **Decentralization:** No centralized server; application runs entirely on client and blockchain.

ASTANA IT UNIVERSITY



5. Maintenance & Support

Smart Contract Maintenance:

Smart contracts are immutable once deployed; any bug fixes require new contract deployment.

Version control all contracts with tags for each deployment.

Maintain deployment history with contract addresses and network details.

Create migration strategy for transitioning to new contract versions if needed.

Frontend Maintenance:

Regular updates to dependencies.

Bug fixes and UI/UX improvements based on user feedback.

Browser compatibility testing and fixes.

Performance optimization and code refactoring.

Infrastructure Support:

Monitor Ethereum test network status.

Track test ETH faucet availability for new user onboarding.

Maintain list of working faucets with updated URLs.

Monitor MetaMask version compatibility.

Documentation:

Keep README updated with current deployment addresses.

Update user guides when UI changes.

Maintain troubleshooting section for common issues.

Document all configuration changes.

Security:

Regular security audits of smart contract code.

Monitor for new Solidity vulnerabilities and best practices.

Code reviews for all changes before deployment.

Keep emergency contacts for critical issues.

User Support:

GitHub Issues for bug reports and feature requests.

FAQ section in documentation.

Video tutorials for common tasks.

Compatibility:

Test with new MetaMask versions.

Verify compatibility with major browser updates.

Ensure mobile responsiveness maintained.

Monitor Ethereum network upgrades for breaking changes.

6. User Testing & Evaluation

TEST #1 (Smart Contract Functionality)

Objective: Verify all smart contract functions work correctly, securely, and efficiently on test network.

Artifacts: Hardhat unit test suite with 100% function coverage - Tnetwork deployment transaction receipt - Gas usage reports for optimization.

Users: Development team members - Nugmash Bigali, Abdibay Asylbek, Tugelbayev Aidyn.

Tasks:

1. Deploy contracts to local Hardhat network and verify deployment

2. Test campaign creation with valid parameters

3. Test campaign creation with invalid parameters

4. Test contribution to active campaign

5. Test contribution to expired campaign

6. Test contribution with insufficient funds

7. Verify reward tokens minted correctly proportional to contribution

8. Test campaign finalization by creator

9. Test campaign finalization by non-creator

10. Verify final balances match expected values

11. Check all events are emitted with correct parameters

12. Measure and optimize gas costs for each function

TEST #2 (MetaMask Integration)

Objective: Ensure seamless wallet connection, transaction signing, and network handling.

Artifacts: - Connected wallet addresses and network IDs - Transaction receipts and signatures - Error logs and debugging information - Screenshots of MetaMask popups.

Users: Development team members - Nugmash Bigali, Abdibai Asylbek, Tugelbayev Aidyn.

Tasks:

1. Connect MetaMask wallet to application

2. Verify correct wallet address displayed in UI

3. Test wallet disconnection and reconnection

4. Switch between different Ethereum test networks

5. Test network detection and warning messages

6. Create campaign transaction and approve in MetaMask

7. Contribute to campaign and approve transaction

8. Test transaction rejection in MetaMask

9. Verify gas fee estimates match MetaMask display

10. Test with multiple accounts in MetaMask

11. Verify token balance updates in MetaMask after contribution

12. Test adding reward token to MetaMask token list.

TEST #3 (End-to-End User Experience)

Objective: Validate complete user journey from wallet setup to successful campaign contribution.

Artifacts: - User feedback forms and notes - Screen recordings of test sessions - UI interaction heatmaps - Bug reports and improvement suggestions - Transaction history for test campaigns.

Users: Development team members - Nugmash Bigali, Abdibai Asylbek, Tugelbayev Aidyn.

Tasks:

1. First-time user setup Install MetaMask and configure wallet

2. Obtain test ETH from faucet

3. Navigate to application and explore interface

4. Connect wallet to application

5. Browse existing campaigns and understand information displayed

6. Create a new campaign with own parameters

7. Contribute to another user’s campaign

8. Verify received reward tokens in wallet

9. Check campaign progress updates in real-time

10. Test error scenarios: wrong network, insufficient balance

11. Use blockchain explorer to verify transactions

12. Provide feedback on usability and clarity

7. Sign-Offs

Signature:		Date:	08.02.26
Print Name:	Nugmash Bigali		
Title:	Team Member		
Role:	Smart Contract Developer		

Signature:		Date:	08.02.26
Print Name:	Abdibay Asylbek		
Title:	Team Member		
Role:	Integration and Testing Lead		

Signature:		Date:	08.02.26
Print Name:	Tugelbayev Aidyn		
Title:	Team Member		
Role:	Frontend Developer		