# **DAOforge Smart Contracts**

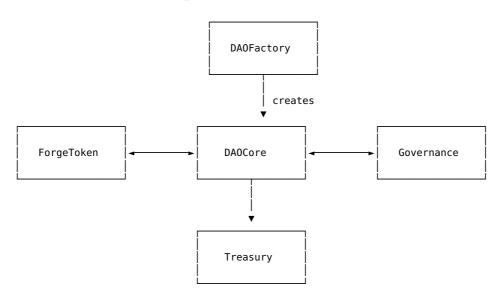
This directory contains the core smart contracts for the DAOforge platform, a comprehensive DAO tooling and governance solution. These contracts form the blockchain backend of the application and enable the creation and management of Decentralized Autonomous Organizations (DAOs).

# **Contract Architecture**

The DAOforge smart contract architecture consists of five main components:

- DAOFactory.sol Factory contract for creating new DAO instances with different templates
- DAOCore.sol Core contract that serves as the central hub for each DAO instance
- 3. **Governance.sol** Contract that handles proposal creation and voting functionality
- 4. Treasury.sol Contract that manages DAO funds and transactions
- 5. **ForgeToken.sol** ERC-20 token contract for the platform with staking capabilities

### **Contract Relationships**



# **Contract Details**

## **DAOFactory.sol**

The factory contract is responsible for creating and deploying new DAO instances. It maintains a registry of all created DAOs and supports different templates for various DAO types (investment, service, social, protocol).

Key features: - Template management for different DAO types - DAO creation with customizable parameters - Global DAO registry - Minimal proxy pattern for gasefficient deployment

#### **DAOCore.sol**

The core contract serves as the central hub for each DAO instance. It manages member information, roles, and integrations with other contracts.

Key features: - Member management with role-based access control - Integration with governance and treasury contracts - DAO metadata and configuration - Access control for various DAO operations

#### Governance.sol

The governance contract handles proposal creation, voting, and execution. It supports different voting mechanisms and delegation.

Key features: - Proposal creation and management - Voting with configurable parameters - Proposal execution - Delegation functionality - Voting analytics

## Treasury.sol

The treasury contract manages DAO funds and transactions. It supports multisignature functionality and asset tracking.

Key features: - Multi-signature transaction approval - Asset management and tracking - Budget allocation and management - Transaction execution with approval thresholds

- Financial reporting

## ForgeToken.sol

The token contract implements the ERC-20 standard with additional functionality for governance and staking.

 $Key \ features: - Standard \ ERC-20 \ functionality - Staking \ mechanisms \ for \ platform \ benefits - Tiered \ staking \ system \ with \ rewards - Governance \ weight \ multipliers - Fee \ discount \ system$ 

## **Contract Interactions**

#### 1. **DAO Creation Flow**:

- User calls DAOFactory.createDAO() with desired parameters
- $\circ\,$  Factory deploys new instances of DAOCore, Governance, and Treasury using minimal proxies
- Factory initializes contracts with proper references to each other
- Factory registers the new DAO in its registry

#### 2. Governance Flow:

- Member creates a proposal via Governance.propose()
- Members vote on the proposal via Governance.castVote()
- If proposal passes, anyone can execute it via Governance.execute()
- $\circ\,$  Executed proposals can make changes to the DAO or execute treasury transactions

## 3. Treasury Flow:

- Members create transactions via Treasury.createTransaction()
- Transactions require approvals based on the approval threshold
- Once approved, transactions are executed automatically
- Treasury tracks assets and budgets for financial reporting

## 4. Token Integration:

- Users can stake FORGE tokens for platform benefits
- Staking tier determines fee discounts and governance weight
- Staking duration affects reward multipliers
- Token can be used for governance voting weight

# **Deployment Guide**

# **Prerequisites**

- · Node.js and npm installed
- Hardhat or Truffle development environment
- Ethereum wallet with testnet/mainnet ETH for deployment

# **Deployment Steps**

#### 1. Deploy Implementation Contracts

First, deploy the implementation contracts that will be used as templates:

```
DAOCore Implementation
Governance Implementation
Treasury Implementation
```

### 2. Deploy ForgeToken

Deploy the FORGE token contract:

ForgeToken

### 3. Deploy DAOFactory

Deploy the factory contract:

DA0Factory

#### 4. Add Templates to Factory

Call addTemplate() on the factory to register the implementation contracts:

```
DAOFactory.addTemplate(
   TemplateType.INVESTMENT,
   "Investment DAO",
   "Template for investment DAOs",
   daoCore.address,
   governance.address,
   treasury.address)
```

Repeat for other template types (SERVICE, SOCIAL, PROTOCOL).

#### 5. Create a DAO

Users can now create DAOs using the factory:

```
DAOFactory.createDAO(
   TemplateType.INVESTMENT,
   "My DAO",
   forgeToken.address,
   604800, // 1 week voting period
   51, // 51% quorum
   1000000000000000000000, // 1000 tokens to create proposal
   [user1, user2], // Initial members
   [0, 0] // Member roles
)
```

# **Security Considerations**

The contracts implement several security measures:

#### 1. Access Control:

• Role-based access control for all sensitive functions

- Clear separation of concerns between contracts
- Proper authorization checks

## 2. Upgradeability:

- Minimal proxy pattern for gas-efficient deployment
- Initializable pattern to prevent re-initialization

#### 3. Safe Operations:

- SafeERC20 for token transfers
- Checks-Effects-Interactions pattern
- Input validation and bounds checking

#### 4. Governance Security:

- Timelock functionality for critical operations
- Multi-signature requirements for treasury transactions
- Quorum and threshold requirements for proposals

# **Gas Optimization**

The contracts are optimized for gas efficiency:

#### 1. Storage Optimization:

- Packed storage variables where possible
- Minimal on-chain storage for large data (using IPFS references)

#### 2. Efficient Patterns:

- Minimal proxy pattern for contract deployment
- Batch operations where appropriate
- Optimized loops and data structures

#### 3. Lazy Loading:

- Pagination for large data sets
- o On-demand computation of derived values

# **Future Improvements**

Planned enhancements for future versions:

#### 1. Advanced Voting Mechanisms:

- Quadratic voting
- Conviction voting
- Holographic consensus

## 2. Multi-Chain Support:

- Cross-chain governance
- Chain-specific optimizations

## 3. Integration Ecosystem:

- Plugin architecture
- Third-party integrations

#### 4. AI Enhancements:

- Proposal analysis
- Treasury optimization
- Governance recommendations