Bytus Token Project Guide

Table of Contents

Introduction

This guide provides comprehensive documentation for the Bytus Token (BYTS) project, an ERC20 token implementation on the Ethereum blockchain with enhanced security features and administrative capabilities.

Project Overview

The Bytus Token is a feature-rich ERC20 token developed for the Ethereum blockchain, offering standard token functionality with important additional features for improved security and administration.

Key Features:

- ERC20 compliant token implementation
- Emergency pause/unpause functionality
- Token burning capabilities
- Custom decimals (3 instead of standard 18)
- Administrative controls
- Web-based dashboard for management

Project Structure:

Copy

```
bytus-token/
├── contracts/              # Solidity contract files
│   └── BytusToken.sol
├── scripts/                # Deployment scripts
│   └── deploy.js
├── hardhat.config.js       # Hardhat configuration
├── index.html              # Admin dashboard HTML file
└── node_modules/           # Dependencies
```

Setup and Installation

Prerequisites

- Node.js (v14 or above)
- npm (Node Package Manager)
- Git (optional for cloning the repository)

Installation Steps

1. **Create project directory and initialize:**

   bash Copy

   ```bash
   mkdir bytus-token
   cd bytus-token
   npm init -y
   ```

2. **Install required dependencies:**

   bash Copy

   ```bash
   npm install --save-dev hardhat @nomicfoundation/hardhat-toolbox
   npm install --save-dev @openzeppelin/contracts
   ```

3. **Initialize Hardhat project:**

   bash Copy

   ```bash
   npx hardhat init
   ```

   Select "Create a JavaScript project" when prompted.

4. **Replace default files with Bytus Token files:**
   - Replace `contracts/Lock.sol` with `contracts/BytusToken.sol`

- Replace default deploy script with custom `scripts/deploy.js`
- Add the `index.html` dashboard file to project root

5. **Update Hardhat configuration:**

```javascript
require("@nomicfoundation/hardhat-toolbox");

module.exports = {
  solidity: "0.8.20",
  networks: {
    localhost: {
      url: "http://127.0.0.1:8545"
    }
  }
};
```

Contract Details

The BytusToken contract utilizes OpenZeppelin contracts as a foundation for security and reliability.

Contract Inheritance

- **ERC20**: Provides base token functionality
- **ERC20Burnable**: Adds token burning capability
- **Pausable**: Enables pausing all token transfers
- **Ownable**: Restricts administrative functions

Custom Features

- **3 decimal places**: Instead of standard 18
- **approveAndCall function**: Facilitates contract interactions
- **Pause mechanism**: For emergency situations

Key Functions

- **constructor**: Sets token name, symbol, and initial supply
- **decimals**: Returns token's decimal places (3)
- **pause/unpause**: Freezes/unfreezes token transfers
- **approveAndCall**: One-step approval and contract interaction

- **_beforeTokenTransfer**: Ensures transfers only happen when not paused

Deployment Guide

Local Deployment

1. **Start a local Hardhat node:**

   bash Copy

   ```
   npx hardhat node
   ```

   This creates a local blockchain with 20 test accounts pre-funded with 10,000 ETH.

2. **Deploy the token contract:**

   bash Copy

   ```
   npx hardhat run scripts/deploy.js --network localhost
   ```

   This deploys the BytusToken with:
   - Initial supply: 66,000,000 tokens
   - Name: "Bytus Token"
   - Symbol: "BYTS"

3. **Verify deployment:**
   - The script will output the contract address
   - Default first-deployment address: 0x5FbDB2315678afecb367f032d93F642f64180aa3
   - Save this address for connecting to the dashboard

Production Deployment

1. **Update hardhat.config.js with network details:**

javascript Copy

```
require("@nomicfoundation/hardhat-toolbox");
require("dotenv").config();

module.exports = {
  solidity: "0.8.20",
  networks: {
    localhost: {
      url: "http://127.0.0.1:8545"
    },
    goerli: {
      url: `https://goerli.infura.io/v3/${process.env.INFURA_API_KEY}`,
      accounts: [process.env.PRIVATE_KEY]
    }
  }
};
```

2. **Create a .env file with credentials:**

Copy

```
INFURA_API_KEY=your_infura_api_key
PRIVATE_KEY=your_wallet_private_key
```

3. **Deploy to selected network:**

bash Copy

```
npx hardhat run scripts/deploy.js --network goerli
```

MetaMask Configuration

Local Development Setup

1. **Add Hardhat Network to MetaMask:**
   - Open MetaMask → Network dropdown → "Add Network"
   - Select "Add a network manually"
   - Enter these details:
     - Network Name: Hardhat Local
     - New RPC URL: http://127.0.0.1:8545
     - Chain ID: 31337
     - Currency Symbol: ETH

    - Block Explorer URL: (leave blank)
  - Click "Save"

2. **Import Test Accounts:**
  - In MetaMask, click on account icon → "Import Account"
  - Paste the private key of a test account (shown when starting Hardhat node)
  - Click "Import"
  - The account should show 10,000 ETH

Requirements

- MetaMask must be connected to the Hardhat network
- Hardhat node must be running
- Imported private key must match exactly as shown in Hardhat console

Admin Dashboard

The admin dashboard provides a user interface for interacting with the BytusToken contract.

Dashboard Setup

1. **Start a local web server: Using Node.js http-server:**

   bash Copy

   ```
   npm install -g http-server
   http-server -p 8080
   ```

   Using Python's server:

   bash Copy

   ```
   python -m http.server 8080
   ```

2. **Access the Dashboard:**
  - Open browser and go to http://localhost:8080
  - Connect MetaMask wallet to Hardhat network
  - Enter contract address in "Admin Controls" section
  - Click "Set" to connect

Adding HTTPS for Development

1. **Install local-ssl-proxy:**

bash Copy

```
npm install -g local-ssl-proxy
```

2. **Run HTTP server:**

bash Copy

```
http-server -p 8080
```

3. **Start SSL proxy:**

bash Copy

```
local-ssl-proxy --source 8443 --target 8080
```

4. **Access secure dashboard:** https://localhost:8443

Dashboard Features

1. **Connection Management:**
   - Auto-connects to contract when wallet is connected
   - Displays connection status and user information
2. **Token Operations:**
   - Transfer tokens to any address
   - Approve addresses to spend tokens
   - Transfer tokens on behalf of others (if approved)
   - Burn tokens from your account or other accounts
3. **Administrative Controls:**
   - Pause/unpause contract functionality (owner only)
   - View contract status and information

Testing Procedures

Contract Function Testing

1. **Information Verification:**
   - Token name: "Bytus Token"
   - Token symbol: "BYTS"
   - Decimals: 3

- Total supply: 66,000,000

2. **Transfer Testing:**
   - Use "Transfer Tokens" panel to send tokens
   - Import a second account to receive tokens
   - Verify balance updates
3. **Approval Testing:**
   - Use "Approve" panel to authorize spending
   - Enter address and approval amount
   - Use "Check Allowance" to verify
4. **TransferFrom Testing:**
   - Import approved spender account
   - Use "Transfer From" panel to send tokens
   - Verify balance updates and allowance reduction
5. **Burn Testing:**
   - Use "Burn Tokens" panel to reduce token balance
   - Verify reduction in balance and total supply
6. **Admin Function Testing:**
   - Switch to owner account
   - Test "Pause Contract" button
   - Verify transfers fail when paused
   - Test "Unpause Contract" button
   - Verify transfers work after unpausing

Testing Tips

- Use multiple accounts for different roles
- Test both positive and negative scenarios
- Verify event emissions in transaction history
- Test dashboard responsiveness on different screen sizes

Security Considerations

Contract Security

1. **OpenZeppelin Security:**

- Uses audited OpenZeppelin libraries
- Follows established security patterns

2. **Emergency Pause:**
   - Contract can be paused by owner in emergencies
   - All transfers and approvals blocked when paused
3. **Access Control:**
   - Administrative functions restricted to owner
   - Standard functions follow proper access control

Development Security

1. **Local Development:**
   - Hardhat provides secure isolated environment
   - Private keys not exposed to internet
2. **HTTPS in Development:**
   - Local-ssl-proxy adds security layer
   - Helps identify potential secure context issues

Production Security

1. **Private Key Management:**
   - Never expose private keys in code or repositories
   - Use environment variables or secure key management
2. **Security Audit:**
   - Consider professional audit before mainnet deployment
   - Follow Ethereum smart contract security best practices
3. **SSL for Production:**
   - Obtain proper SSL certificates for production
   - Consider Let's Encrypt for free, trusted certificates

Troubleshooting

Common Issues and Solutions

1. **MetaMask Connection Problems:**
   - Ensure Hardhat node is running

- Verify correct network is selected in MetaMask
- Check that the imported private key matches a Hardhat account

2. **Contract Deployment Failures:**
   - Verify network configuration in hardhat.config.js
   - Ensure enough ETH for gas fees in deployment account
   - Check for syntax errors in contract code
3. **Dashboard Connection Issues:**
   - Verify contract address is correct
   - Ensure MetaMask is connected to proper network
   - Check browser console for JavaScript errors
4. **Transaction Failures:**
   - Check if contract is paused (transfers will fail)
   - Verify sufficient token balance for operations
   - Ensure proper approvals are in place for transferFrom
   - Check gas limit and price settings in MetaMask