

Ethereum Smart Contract Development

Solidity, testing, application integration and security fundamentals

FA-0597

Software Development · DevOps, Cloud & Infrastructure · Cybersecurity

Develop and test a small Ethereum application using Solidity, token standards and current development tools, with attention to security and deployment constraints.

Introduction

This developer-focused course progresses from blockchain and Ethereum architecture to Solidity contracts, token standards, testing, frontend integration and security review. Guided exercises use a local development network or an appropriate testnet; mainnet deployment requirements are discussed rather than requiring real funds.

The course is independent technical training, not a certification award or assurance of examination readiness. Ethereum's current proof-of-stake model is distinguished from historical mining. Hardhat and a selected compatible ethers.js client support the development workflow; retired Truffle, Ganache and Web3.js tooling is treated as legacy migration context.

Learning outcomes

The course teaches participants to:

- Explain Ethereum accounts, transactions, gas, the EVM and proof-of-stake network operation.
- Write and test Solidity contracts using types, storage, events, functions and access controls.
- Implement a small ERC-20 or ERC-721 example and explain the standards' different purposes.
- Connect a frontend to contracts through an ABI and a supported JavaScript client.
- Review common contract risks, compiler behaviour and the limitations of reusable libraries and audits.

- Deploy a learning application to a local network or testnet and assess additional requirements before real-world deployment.

Prerequisites

- Basic programming experience, preferably with JavaScript, and familiarity with a command line.
- An understanding of basic blockchain concepts is helpful; Solidity experience is not required.
- A compatible development environment and learning-only accounts for guided exercises.

Training outline

1. Module 1 — Blockchain and Ethereum Foundations

- Public, private and consortium networks; blocks, transactions, addresses and wallets.
- Public/private keys, hashing and basic blockchain security assumptions.
- Ethereum, ETH, the EVM, gas and smart-contract/DApp concepts.
- Ethereum's proof-of-stake consensus; proof of work and mining as historical comparison.
- Externally owned accounts, contract accounts and ecosystem tooling; guided checks.

2. Module 2 — Networks and Nodes

- Mainnet, testnets and local development networks; select an appropriate learning environment.
- Node roles, execution/consensus clients and transaction/block validation.
- Local development RPC setup versus the resource and operational needs of a full node.
- Inspect transactions and network interactions without exposing private keys or using real funds.

3. Module 3 — Solidity and EVM Programming

- Solidity and Vyper roles; Solidity source structure and compiler selection.
- EVM opcodes/assembly as an architectural overview, not a prerequisite to every contract.
- Gas estimation and trade-offs; storage layout and trie concepts for the selected Ethereum implementation.
- Compile a small contract and inspect its ABI and bytecode.

4. Module 4 — Solidity Language and Contract Structure

- Types, arrays, structs, mappings, state variables and storage considerations.
- Functions, visibility, getters, modifiers and access controls.

- Events, logging, global values and receive/fallback behaviour for the selected compiler.
- Inheritance, contract organisation, errors and low-level calls.
- Checked arithmetic and explicit unchecked blocks; coding conventions and guided tests.

5. Module 5 — Token Standards

- ERC specifications and the different purposes of ERC-20 and ERC-721.
- Implement a selected fungible-token or NFT example using reviewed library components.
- Test ownership, transfers, approvals and relevant access rules.
- Explain that a token implementation does not establish investment value, legal status or suitability.

6. Module 6 — Development Tools and Testing

- Configure a compatible Hardhat project and selected JavaScript testing tools.
- Unit tests, expected failures, events and transaction behaviour; reuse Mocha-style concepts where supported.
- Truffle/Ganache retirement and legacy-project migration considerations.
- Compiler versions, ABI/metadata, test isolation and reproducible project configuration.

7. Module 7 — Application Integration

- Use a supported ethers.js client to connect a frontend, provider and signer.
- Read contract state, request transactions, handle events and estimate gas.
- Web3.js sunset and migration concepts rather than new projects built on unsupported tooling.
- MetaMask, Remix and hosted RPC options such as Infura or Alchemy; keep accounts and credentials learning-only.

8. Module 8 — Smart Contract Security

- Reentrancy, access-control mistakes, arithmetic behaviour and external-call risks.
- Solidity security guidance, defensive design and negative testing.
- Use OpenZeppelin components with correct configuration and version review; libraries do not guarantee security.
- Upgradeable-contract concepts and their added administrative/security assumptions.
- Distinguish learning review from an independent audit or assurance of production safety.

9. Module 9 – Deployment and Review

- Deploy and interact with a selected contract on a local network or suitable testnet.
- Review RPC connectivity, frontend configuration and deployment records.
- Discuss mainnet gas, key management, testing, monitoring and independent review requirements.
- Recap the project, answer technical questions and identify remaining learning needs.