Blockchain Development Training (45 Days)

- **Goals:** Train students to function as junior blockchain developers. Emphasize practical skills: smart contract design, DApp architecture, multi-chain familiarity, security/auditing, and deployment**.**
- **Prerequisites:** Proficiency in a general-purpose language (e.g. JavaScript/TypeScript or Python), comfort with Git/CLI, and basic data structures/algorithms. Familiarity with web development (HTML/CSS/React) is recommended but not mandatory. We assume students know JavaScript syntax, asynchronous calls, and have basic programming experience.
- **Outcomes:** Students will be able to:

1. Explain blockchain concepts (blocks, transactions, consensus, smart contracts) and describe major platforms (Ethereum, BSC, Polygon, Solana, Hyperledger).
2. Write, test, and deploy Solidity smart contracts (ERC-20/ERC-721 tokens, simple DeFi contracts), using Hardhat or Foundry.
3. Integrate contracts with front-ends (using Ethers.js/Web3.js and MetaMask) to build full-stack DApps.
4. Optimize contracts for gas, implement security patterns (check-effects-interactions, reentrancy guards), and use audit tools (Slither, MythX, etc.).
5. Deploy contracts to public testnets (Goerli, BSC Testnet, Polygon Mumbai, Solana Devnet) and configure CI/CD pipelines.
6. Prepare a developer portfolio and technical resume, with completed projects on GitHub.
7. Answer common blockchain interview questions (cryptographic primitives, consensus, gas model) and solve coding/design tasks (e.g. implement a token, design a DEX)

Week 1: Blockchain Fundamentals & Ethereum Basics

- **Day 1:** *Intro to Blockchain.* Cover blockchain history, types (public/permissioned), basic crypto (hashes, keys, PKI). Explain blocks, transactions, P2P networks. *Lab:* Use a local blockchain simulator (Hardhat or Ganache) to send sample transactions.
- **Day 2:** *Ethereum Overview.* Network architecture, accounts vs contracts, Ether vs gas, transaction lifecycle. Launch MetaMask wallet, get ETH from Goerli faucet. *Demo:* Write and deploy a simple “Hello World” Solidity contract using Remix.

- **Day 3:** *Solidity & Smart Contracts.* Explain Solidity syntax, data types, functions. Show ERC-20 token example. *Lab:* Write a basic Storage contract (set/get an integer) and deploy on Hardhat localnet.
- **Day 4:** *Ethereum Tooling.* Install and configure Hardhat (and Foundry optionally). Learn Hardhat project structure, scripts, and test framework. Use Ethers.js to interact with contracts. *Lab:* Scaffold a Hardhat project and test the Storage contract.
- **Day 5:** *Mini-Project:* **ERC-20 Token Contract.** Students implement an ERC-20 token (using OpenZeppelin template), write unit tests in Hardhat, and deploy to Goerli. By week's end they have a working token and know how gas fees work.

Week 2: Advanced Solidity & Smart Contract Patterns

- **Day 6:** *Advanced Solidity.* Events, modifiers, inheritance. Implement access control (Ownable pattern). *Lab:* Create Pausable or Ownable modifiers.
- **Day 7:** *Common Smart Contracts.* ERC-721 NFTs vs ERC-1155. *Demo:* Mint an NFT using OpenZeppelin ERC721. *Lab:* Extend last week's token into ERC-721 or multi-token.
- **Day 8:** *Security Patterns.* Explain reentrancy, integer overflow, unchecked calls. Demonstrate check-effects-interactions and usage of OpenZeppelin's ReentrancyGuard. *Lab:* Audit and fix a vulnerable contract (e.g., simple escrow with a reentrancy bug).
- **Day 9:** *Gas Optimization.* Cover gas costs and optimization techniques (use immutable/constant, struct packing, short-circuiting). *Lab:* Refactor a contract for gas efficiency; measure gas via Hardhat tests.
- **Day 10 (Mini-Project): Decentralized Voting Contract.** Build a secure voting or auction contract. Requirements: allow users to vote (or bid) with ERC-20 token, enforce time limits, and safely handle funds. Include unit tests and a deployment script.

Week 3: Testing, Tooling & DApp Integration

- **Day 11:** *Smart Contract Testing.* In-depth Hardhat testing (Mocha/Chai, ethers.js assertions) and Foundry (forge test). Show code coverage tools. *Lab:* Write test suites for previous projects.
- **Day 12:** *Version Control & CI.* Git/GitHub best practices: forking, branching, PRs. Introduce GitHub Actions for running tests on commits. *Demo:* Set up CI pipeline to auto-test on push.

- **Day 13:** *Frontend Web3 Integration.* Intro to web3 libraries: Ethers.js vs Web3.js. Build a simple HTML/JS UI that calls a deployed contract. *Lab:* Create a React app (or plain HTML/JS) to read/write to the Voting contract via MetaMask and Ethers.js.
- **Day 14:** *DApp Architecture.* Explain full-stack DApp components: smart contracts (backend), APIs (if any), and frontend. Use a diagram to show interactions (Mermaid or equivalent). Cover state management (React context) for wallet/chain data. *Lab:* Extend frontend to handle connect/disconnect wallet, display contract events, and handle transaction states.
- **Day 15 (Mini-Project): NFT Marketplace Prototype.** Students build a simple frontend for minting and trading NFTs. Features: connect wallet, mint an ERC-721, list/sell tokens (basic mapping or order book), and view marketplace. Use a public testnet (Polygon Mumbai) for deployment.

Week 4: Oracles, DeFi Concepts & Layer-2s

- **Day 16:** *Oracles & External Data.* Explain oracles (Chainlink, Web3Oracle). *Lab:* Fetch an off-chain API (e.g. price feed) into a contract and trigger actions. (Use Chainlink testnet node or mock oracle.)
- **Day 17:** *Decentralized Finance (DeFi).* Cover DEX and lending basics (AMM model, Uniswap v2/v3, Compound). *Demo:* Walk through a simple swap contract or call Uniswap Router from code. *Lab:* Simulate a token swap via Uniswap or SushiSwap on a testnet.
- **Day 18:** *Layer-2 Solutions.* Discuss Rollups (Optimism/Arbitrum), sidechains (Polygon). Show how to bridge tokens (e.g. using Polygon PoS bridge). *Lab:* Deploy a simple contract to Polygon Mumbai and perform a bridge transaction (testnet token transfer from Ethereum).
- **Day 19:** *Other Platforms Introduction.* Quick overview of Solana and Hyperledger: when and why to use them. (Devote a section for Hyperledger basic concepts and Solana Rust development.)
    - *Solana:* Rust/Anchor framework; PoH+PoS consensus.
    - *Hyperledger Fabric:* Permissioned chains, chaincode in Go/Java/Node.
- **Day 20 (Mini-Project): Cross-Chain Token Transfer Demo.** As a fun exercise, have students issue an ERC-20 on a local chain and “bridge” it conceptually to another chain (e.g. lock tokens on Ethereum localnet and mint on a Polygon local chain) or simulate with mocked contracts. This reinforces multi-chain concepts.

Week 5: Multi-Chain Development & Enterprise Blockchains

- **Day 21:** *Binance Smart Chain (BSC).* Overview: PoSA consensus, EVM-compatible, low fees. *Lab:* Configure MetaMask for BSC Testnet; deploy an existing DApp (e.g. Voting or Token) to BSC testnet.
- **Day 22:** *Polygon.* Deep dive into Polygon PoS (Cosmos-based PoS, ~7s finality). *Lab:* Port the NFT marketplace or voting DApp to Polygon Mumbai, noting differences.
- **Day 23:** *Solana Basics.* Set up Rust toolchain, Solana CLI, and Anchor framework. *Lab:* Write a minimal Solana program (Rust) that stores a number and returns it. Deploy to Devnet. (Alternately, use Anchor examples.)
- **Day 24:** *Hyperledger Fabric.* Explain Fabric architecture (permissioned nodes, channels, chaincode). *Lab:* Use Hyperledger Fabric's test network (or simplified emulator) to deploy a Go chaincode for asset tracking. Explore writing chaincode in Node.js or Java.
- **Day 25 (Mini-Project): Enterprise Use Case on Fabric.** In teams, create a simple supply-chain chaincode (e.g. asset transfer with endorsement policy) and a Node.js client app to submit transactions. This contrasts public DApp model with permissioned networks.

Week 6: Security, Auditing, Deployment & Capstone Projects

- **Day 26:** *Security Best Practices.* Review common vulnerabilities (reentrancy, overflows, unchecked calls) and fixes. Introduce audit tools: **Slither, MythX, Manticore, Oyente**. *Lab:* Run Slither/MythX on students' contracts and fix issues.
- **Day 27:** *Gas Optimization & Best Practices.* Techniques like **const/immutable**, minimal storage writes, loop limits. Show example of packing struct variables. *Lab:* Optimize a sample contract for minimal gas and compare costs.
- **Day 28:** *Deployment and CI/CD.* Set up deployment pipelines to a testnet and (optionally) mainnet. Cover environment configs (dotenv), API keys (Infura/Alchemy), verifying contracts (Etherscan API), and releasing versions. *Lab:* Integrate a simple GitHub Action that runs hardhat test and hardhat verify on push to a deploy branch.
- **Day 29:** *Interview and Industry Prep.* Conduct mock interviews and coding quizzes. Review typical tasks: implement an ERC token, design a simple DEX system architecture, and answer interview questions (e.g. "How does consensus differ between Ethereum and Hyperledger?" or "Explain the check-effects-interactions pattern."). Reference **common questions** from industry resources. Provide tips for resume building (blockchain buzzwords, linking projects).

- **Day 30 (Capstone Submission):** Final presentations. Each team presents their capstone DApp or system. Projects include **an end-to-end DApp** (e.g. NFT marketplace with frontend, a DeFi lending app, a DAO dashboard, or an enterprise blockchain solution). All code must be on GitHub with documentation, live demo or hosted frontend, and deployment links. Grading uses the rubric below.

Hands-On Labs & Projects

Each week includes a **mini-project** (described above) that integrates the week's concepts. Students work individually or in pairs. Weekly projects progress toward larger skills:

- **Week 1 (ERC-20 Token):** Teaches Solidity syntax, OpenZeppelin contracts, testing and deployment.
- **Week 2 (Voting/Auction):** Emphasizes security (no reentrancy) and advanced Solidity.
- **Week 3 (NFT Marketplace):** Combines front-end integration, MetaMask usage, and multiple contract interactions.
- **Week 4 (Cross-Chain Demo):** Focus on bridging & oracles, reinforcing the ecosystem view.
- **Week 5 (Fabric Supply Chain):** Introduces permissioned chains and chaincode development.
- **Week 6 (Capstone):** Extended DApp from scratch. Example ideas: decentralized exchange (AMM), multi-signature wallet system, DAO/governance platform, DeFi yield optimizer, or any creative DApp (e.g. tokenized game items). Each capstone must list features (user auth, smart contracts, UI, backend if any, deployment) and be graded by functionality, code quality, and presentation.

Tools, Frameworks, Libraries & Environments

- **Languages:** Solidity (primary for EVM chains), Rust (for Solana), Go/Java/Node (for Fabric).
- **Frameworks:**
    - **Hardhat** (JS/TS framework; write, test, deploy contracts).
    - **Foundry** (Rust-based toolkit; very fast compilation/testing).
    - *[Truffle]* (older, being deprecated; mention for legacy understanding).
    - **OpenZeppelin Contracts** (reusable audited smart contract library).

    - **Remix IDE** (in-browser Solidity IDE for quick prototyping).
- **Libraries:**
    - **Ethers.js** (modern Ethereum JS SDK for frontend/backend interactions).
    - **Web3.js** (legacy JS SDK, still worth knowing).
    - **Solidity libraries:** OpenZeppelin for ERCs, SafeMath (though built-in from 0.8.x).
- **Wallets/Testnets:** MetaMask (main Ethereum wallet), other wallets (e.g. Phantom for Solana). Use Goerli or Sepolia for Ethereum, Mumbai for Polygon, BSC Testnet, and Solana Devnet for testing.
- **Node providers:** Infura, Alchemy, QuickNode – for connecting to public chains.
- **Security Tools:** Slither (static analysis), MythX (automated audit).
- **CI/CD:** GitHub Actions to auto-run tests/deployment. Docker containers (for local Fabric or Ganache).

Deployment & Security Best Practices

- **Deployment:** Use environment variables for private keys/API keys. Verify contracts on explorers (Etherscan PolygonScan). Do staging on testnet before mainnet. Use upgradeable patterns (OpenZeppelin Upgrades) if needed.
- **Security:** Follow OWASP and blockchain-specific practices:
    - Always validate inputs and check return values of external calls. Use *checks-effects-interactions* pattern to prevent reentrancy.
    - Handle integer overflow (Solidity ≥0.8 auto-checks). Avoid tx.origin for auth (use msg.sender). Use OpenZeppelin libraries (e.g. ReentrancyGuard, Ownable).
    - Restrict access: use modifiers (onlyOwner, etc.) and role-based access (OpenZeppelin AccessControl).
    - Implement circuit breakers/emergency stops if applicable.
    - Limit loop sizes to avoid DoS via gas exhaustion.
- **Gas Optimization:** Use constant/immutable for variables, pack struct fields, prefer *mapping* over large arrays, minimize state writes. Keep hot code in memory/local (check/gas costs differences).

- **Auditing Checklist:** Before finalizing a contract, run static analysis (Slither, MythX), manual code review looking for common issues (list below), and consider third-party audit for high-value contracts. Key checks from audit checklist include overflow/underflow, timestamp dependence, external call failures, delegatecalls, DoS loops, correct randomness use, etc..

Security Audit Checklist

- **Reentrancy:** Ensure no unguarded external calls; use mutex or ReentrancyGuard.
- **Integer Safety:** Rely on Solidity ≥0.8 or SafeMath. Check math logic.
- **Access Control:** Verify only authorized addresses can perform sensitive ops. Watch out for loose public functions.
- **Block/Timestamp Dependence:** Do not use timestamps for critical logic (miners can influence).
- **Unchecked Calls:** Always check return values of low-level calls (call) to external contracts.
- **Gas-Related:** Avoid unbounded loops, large storage reads in hot paths; use events for logging instead of storing unnecessary data.
- **Oracle/Randomness:** If used, ensure provable or secure sources. Beware of manipulations.
- **Upgradeability/Proxy:** If using proxies, ensure storage layout safety and avoid initializer pitfalls.
- **General:** Adhere to known best practices (e.g. OpenZeppelin style guide, OWASP).