**CODELANDCS BLOCKCHAIN DEVELOPMENT SYLLABUS**

**WEEK 6**

**DAY 1**

**INTRODUCTION TO ETHERS JS**

Ethers.js is a powerful library for **interacting with the Ethereum blockchain.** It is a popular open-source library that offers an extensive set of tools for developers to interact with Ethereum-based applications. Ethers.js is written in JavaScript and can be used in both **Node.js and browser environments.** In this hour lecture, we will explore the different features of Ethers.js and how they can be used in various use cases.

Firstly, Ethers.js offers an easy-to-use interface for creating and managing Ethereum wallets. **A wallet is a collection of private keys** that are used to sign transactions on the Ethereum network. Ethers.js provides a simple way to create a new wallet or import an existing one from a private key, mnemonic phrase, or JSON file. This makes it easy for developers to manage their private keys securely without worrying about the low-level details.

Secondly, Ethers.js provides an interface for interacting with Ethereum smart contracts. Smart contracts are self-executing contracts that are stored on the Ethereum blockchain. They can be used to automate a wide range of processes, including token sales, decentralized exchanges, and even voting systems. Ethers.js provides a simple way to deploy a new smart contract or interact with an existing one. It also supports complex contract interactions, including calling functions with multiple arguments and handling events emitted by the contract.

Thirdly, Ethers.js provides a powerful set of tools for working with Ethereum transactions. Transactions are the fundamental building blocks of the Ethereum network, and they are used to transfer ether (the native cryptocurrency of Ethereum) or interact with smart contracts. Ethers.js provides an easy-to-use **interface for creating and sending transactions, estimating gas costs,** and **checking transaction status.** It also supports more advanced transaction features, such as nonce management and gas price optimization.

Fourthly, Ethers.js supports the development of decentralized applications (dApps) on Ethereum. dApps are applications that run on a decentralized network, rather than a centralized server. Ethers.js provides an easy-to-use interface for interacting with Web3-enabled browsers, such as MetaMask. This makes it easy for developers to create dApps that can be accessed by users directly from their browser.

Finally, Ethers.js offers a variety of utilities and helpers that make it easy for developers to work with Ethereum. For example, it includes a set of utility functions for working with Ethereum addresses and private keys. It also includes a built-in unit converter for working with ether denominations, as well as a JSON-RPC client for interacting with Ethereum nodes directly.

In conclusion, Ethers.js is a powerful library for interacting with the Ethereum blockchain. It offers a wide range of features, including wallet management, smart contract interactions, transaction management, dApp development, and more. With its easy-to-use interface and comprehensive set of tools, Ethers.js is a popular choice for developers looking to build Ethereum-based applications.

**NODE PACKAGE MANAGER**

One important tool used in the development of smart contracts is the Node Package Manager (NPM).

Node Package Manager (NPM) is a **package manager for the JavaScript programming language.** It is used to install and manage open-source packages, libraries, and dependencies. It is an essential tool for JavaScript developers, providing access to a vast repository of packages and modules that can be used to build and deploy decentralised applications.

**NPM and Smart Contract Development**

NPM is not just a package manager for traditional JavaScript development. It is also an important tool for smart contract development. Many smart contract platforms, such as Ethereum, use JavaScript for their smart contract development. This means that developers can **use NPM to manage the dependencies** and packages required for smart contract development.

**Smart contracts often require the use of external libraries**, such as Web3.js, ethers js to interact with the blockchain. These libraries can be installed and managed using NPM. NPM also allows for the development of custom packages that can be used across multiple smart contracts. This can help to reduce development time and improve code quality.

NPM also **provides version control**, which is critical for smart contract development. Smart contracts are immutable, meaning that once they are deployed, they cannot be changed. This means that developers need to be sure that they are using the correct version of each package and library. NPM allows developers to specify the version of each package they are using, ensuring that the smart contract will work correctly when deployed.

**Best Practices for NPM and Smart Contract Development:**

There are several best practices that developers should follow when using NPM for smart contract development. These include:

**Locking Package Versions:** As mentioned earlier, smart contracts are immutable. This means that developers must ensure that they are using the correct version of each package. To do this, developers should use NPM's package-lock.json file to lock the versions of packages they are using.

**Using Trusted Packages:** Developers should only use packages from trusted sources to ensure the security and reliability of their smart contracts. They should also regularly update their packages to ensure that they are not using packages with known vulnerabilities.

**Using Semantic Versioning:** Semantic Versioning is a standard way of versioning packages. It uses three numbers, separated by dots, to represent the version number (e.g., 1.2.3). The first number represents a major version, the second number represents a minor version, and the third number represents a patch version. Developers should use Semantic Versioning when creating their own packages to make it clear which changes are breaking changes.

**Conclusion:**

In conclusion, NPM is an essential tool for smart contract development. It provides access to a vast repository of packages and libraries, allowing developers to build and deploy applications more efficiently. It also provides version control, which is critical for smart contract development. Developers should follow best practices, such as locking package versions, using trusted packages, and using Semantic Versioning, to ensure the security and reliability of their smart contracts.

**DEPLOYING A CONTRACT USING ETHERS JS**

Deploying a contract on the Ethereum blockchain using ethers.js involves several steps. In todays, we will go through the process of deploying a smart contract using ethers.js using a simple Storage contract.

We would be making use of the **solidity compiler** and the **ethers js** library which we would be installing from NPM.

Factors to watch out for

* Make sure the **solidity compiler** version to be installed from NPM is the same declared in the pragma statement.
* Install the ethers js library to help us communicate with the Ethereum blockchain
* Create a script that would be compiling the solidity code.
* We would download the Ganache Software (EVM Compatible) which is a fake blockchain to help connect to a block chain and simulate calls to and from our contract.