CARDANO DEVELOPMENT COURSE (Week 1-2)

Module 1: Introduction to Cardano Development.

1.0 Understanding the Cardano Blockchain and its Architecture.

1.1. Cardano eUTxO Model

1.2 Cardano DApp Architecture

1.3. Cardano Development Stacks (Aiken, Plutarch, PluTs, Helios)

1.4. Datums and Redeemers

1.5. Aiken DSL and Developer Environment setup

1.0 Understanding the Cardano Blockchain and its Architecture.

1.0.1. Brief History

Charles Hoskinson and Jeremy Wood, both former members of the Ethereum team, co-founded Cardano. In 2015, they established Input Output Hong Kong (IOHK) to develop more sustainable blockchain solutions. By adopting a peer-reviewed approach to blockchain development and introducing the Ouroboros consensus mechanism, Cardano Mainnet was launched in 2017.

1.0.2. Ouroboros

Ouroboros is a proof-of-stake (PoS) consensus mechanism that selects nodes to validate transactions based on the amount of ADA they have staked, rather than relying on computationally intensive work like proof-of-work (PoW) chains. This approach offers two key benefits: it significantly reduces energy consumption and encourages honest participation, as validators have a financial stake at risk if they act maliciously.

1.0.3. The Cardano Blockchain Architecture

Cardano is a third-generation blockchain that prioritizes scalability, security, and decentralization.The blockchain model consists of several key components and layers. Users engage with the ledger by creating transactions, which are temporarily stored in the mempool until they are added to a block. Stake pools are responsible for producing blocks and receive rewards for their participation, which they distribute to their delegators. The Cardano layered architecture consist of:

- **Cardano Settlement Layer (CSL):** This layer is responsible for handling ADA transactions and ensuring security through the Ouroboros proof-of-stake (PoS) consensus mechanism.
- **Cardano Computation Layer (CCL):** This layer supports smart contracts and DApps, allowing for programmability and automation.

- **Consensus Mechanism - Ouroboros:** Unlike Bitcoin's proof-of-work (PoW), Ouroboros uses stake delegation and validators to achieve network consensus, making Cardano more energy-efficient.
- **Native Assets:** Cardano allows users to create and manage custom tokens directly on the blockchain without the need for smart contracts, improving efficiency.

1.1 Blockchain Record Keeping Models

The major Smart Contracts Blockchain networks commonly use two record-keeping models: the Extended Unspent Transaction Output (eUTXO) model, utilized by Cardano, and the Account/Balance model, adopted by Ethereum. This section explores the fundamentals of these models and highlights their differences.

1.1.1 Understanding Cardano eUTXO Model

Cardano's Extended Unspent Transaction Output (eUTxO) model is an advancement of Bitcoin's UTxO model, offering increased flexibility, security and scalability. Each transaction consumes outputs from previous transactions and creates new outputs for future ones. Fully synchronized nodes store all unspent transaction outputs, giving rise to the term "eUTXO" (Extended Unspent Transaction Output). A user's wallet monitors the unspent transactions linked to their addresses, with the wallet balance representing the total value of these unspent outputs.

For example,

1. Jon earns 20.5 ADA through staking rewards, resulting in one eUTXO of 20.5 ADA.
2. Jon sends 2 ADA to Ron. Jon's wallet uses his eUTXO of 20.5 ADA, sending 2 ADA to Ron and receiving 18.5 ADA as a new eUTXO to his address.
3. If Ron had an eUTXO of 5 ADA before step 2, his wallet now shows a balance of 7 ADA from two eUTXOs.

1.1.2. Understanding Account/Balance Model

The Account/Balance Model maintains the balance of each account as a global state. It checks that an account's balance is sufficient to cover the transaction amount.

For example,

1. Jon gains 30 ETH through mining, recorded in the system.
2. Jon sends 7 ETH to Ron, reducing his balance to 23 ETH.
3. Ron's balance increases by 7 ETH, so if he had 3 ETH initially, he now has 10 ETH.

1.2. Overview Of The Cardano dApp Architecture

A typical Cardano DApp consists of:

1. **Frontend:** Built using frameworks like React.js or Next.js to provide an interactive UI.
2. **Backend:** Uses middleware services like Koios, Blockfrost, or custom APIs to fetch blockchain data.
3. **Off-chain:** Off-chain scripts written in Javascript or Typescript using Lucid Evolution or MeshJs. This component handles computations, data storage, and logic execution that do not need to be processed on-chain, reducing congestion and transaction costs.
4. **Smart Contracts:** On-chain scripts written in Aiken, Plutus, or other DSLs that define & enforce business logics and transaction validations.
5. **Wallet Integration:** Uses the CIP-30 standard to interact with browser wallets such as Nami, Eternl, or Lace.

1.3. Introduction To Cardano Development Stacks

Cardano supports various development stacks/languages for writing and deploying smart contracts. In this course, we'll introduce you briefly to the following programming languages:

- Plutus: Plutus is the native smart contract language for Cardano.
- Plutarch: A highly optimized eDSL for writing Plutus scripts with better control over execution costs.
- Plu-Ts: A TypeScript-based framework that allows developers familiar with JavaScript to write Plutus contracts.
- Helios: A JavaScript-friendly smart contract language designed for web developers who prefer working within JavaScript ecosystems.
- Aiken: A modern, ergonomic DSL optimized for Cardano smart contract development.

1.3.1. Introduction to Plutus Language

Plutus is a Turing complete language written in Haskell, and Plutus smart contracts are effectively Haskell programs. By using Plutus, you can be confident in the correct execution of your smart contracts. It draws from modern language research to provide a safe, full-stack programming environment based on Haskell - the leading purely functional programming language. This setup ensures that users can rely on well-established tooling and libraries without needing to learn a new, proprietary language.

Plutus Tx Compiler

The key technology that facilitates this is Plutus Tx, which acts as a compiler from Haskell to Plutus Core, the language executed on the Cardano blockchain. Provided as a GHC (Glasgow Haskell Compiler) plug-in, Plutus Tx compiles Haskell code into executable files for users' computers and Plutus Core for blockchain execution.

For more detailed information and installation guide, visit the Plutus Playground and follow the development on Github

1.3.2. Introduction to Helios Language

The Helios language is a platform agnostic functional programming language with a syntax that is notably similar to C. It features a straightforward curly braces syntax and is inspired by languages like Go and Rust. Helios aims to balance simplicity and safety in the development of decentralized applications (dApps) on the Cardano blockchain.

Tenets of Helios

• **Readability Over Writability:** The language emphasizes code readability, ensuring that code is easy to understand and maintain.
• **Easily Auditable:** Helios is designed to be easily auditable, allowing developers and auditors to quickly spot potential malicious code.
• **Opinionated:** The language provides a single, clear way to accomplish tasks, reducing ambiguity and complexity in code development.

Helios as a JavaScript/TypeScript SDK

Helios serves as a JavaScript/TypeScript SDK for the Cardano blockchain, encompassing everything needed to build dApps. This includes a simple smart contract language tailored for the Cardano ecosystem.

For more detailed information and installation guide, visit the Helios Playground and follow the documentation on the Helios website.

1.3.3. Introduction to Plu-Ts

Plu-ts is a library designed for building Cardano dApps in an efficient and developer-friendly way. It is composed of two main parts:
• plu-ts/onchain: An eDSL (embedded Domain Specific Language) that leverages TypeScript as the host language, designed to generate efficient Smart Contracts.
• plu-ts/offchain: A set of classes and functions that allow reuse of onchain types.

Design Principles

Plu-ts was designed with the following goals in mind, in order of importance:
• **Smart Contract efficiency**

- **Developer experience**
- **Reduced script size**
- **Readability**

For more information, see the Plu-ts guide book.

1.3.4. Introduction to Aiken

Aiken is a modern programming language and toolchain designed specifically for developing smart contracts on the Cardano blockchain. Influenced by modern languages like Gleam, Rust, and Elm, it prioritizes clear error messages and a seamless developer experience. Aiken focuses on creating on-chain validator scripts, requiring users to write their off-chain code for generating transactions in other languages such as Rust, Haskell, JavaScript, or Python.

For this course, we will dive deeper into the Aiken programming language.

Language Features

Aiken is a purely functional language with static typing and type inference. This means that most of the time, the compiler is smart enough to determine the type of something without requiring user annotation. Aiken allows the creation of custom types resembling records and enums but does not include higher-kinded types or type classes, aiming for simplicity. On-chain scripts are typically small in size and scope compared to other kinds of applications being developed today and do not necessitate as many features as general-purpose languages that must tackle far more complex issues. Reference

Similar to Plutus, Aiken scripts are compiled down to the untyped Plutus Core (UPLC).

Before we go further deeper, let's understand three important parameters of Cardano smart contracts namely: Datum, Redeemers, and transactioncontext.

1.4. Datums, Redeemers, and Scriptcontext.

Datum

A datum is data linked to UTXOs, representing the state of a smart contract. While the datum itself is immutable, the contract's state can evolve by consuming existing UTXOs and generating new ones. The "e" in eUTXO stands for "extended," referring to the inclusion of datums. Unlike Bitcoin's UTXO model, which lacks this feature and has limited functionality, the extended UTXO model used by Cardano offers capabilities similar to an account-based model while ensuring a more secure transaction process by eliminating global state mutations.

Redeemer

The redeemer is another piece of data provided with the transaction for script execution. The datum and redeemer intervene at two distinct moments: the datum is set when the output is created (similar to attaching a note to a wall), whereas the redeemer is provided only when spending the output (like handing over a form to an employee). Together, they play crucial roles in the functioning of smart contracts.

TransactionContext

The majority of the logic in smart contracts involves making assertions about certain properties of the TransactionContext. The TransactionContext contains valuable information, such as

- When is the transaction occurring?
- What are the inputs of the transaction?
- What are the outputs of the transaction?

All these details are encapsulated in the TransactionContext object, which is passed into the contract as the last argument. Understanding and utilizing the TransactionContext is essential for validating transactions. The TransactionContext can be visualized as an object with the following properties:

```
Transaction {
            inputs: List,
            reference_inputs: List,
            outputs: List,
            fee: Value,
            mint: MintedValue,
            certificates: List,
            withdrawals: Pairs,
            validity_range: ValidityRange,
            extra_signatories: List>,
            redeemers: Pairs<ScriptPurpose, Redeemer>,
            datums: Dict<Hash<Blake2b_256, Data>,Data>,
            id: TransactionId,
}
```

This structure highlights the importance of reading and understanding the exact details of the inputs, outputs, time, and signatories of the transaction. Simply having the datum and redeemers is not sufficient to validate a transaction; the full context provided by the TransactionContext is essential for comprehensive validation and ensuring the security and correctness of the smart contract execution.

Now that we have that out of the way, it's time for installation and the Aiken Developer Environment Setup.

1.5. Aiken DSL and Developer Environment setup

Aiken can be installed via Aikup or manually. Both methods will be demonstrated below:

Manually:

From a package manager:

```
npm install-g @aiken-lang/aiken
```

Homebrew:
`brew install aiken-lang/tap/aiken`

From Sources (All platforms):

```
cargo install aiken--version 1.1.15
```

From Nix flakes (Linux & MacOS only):

```
nix build github:aiken-lang/aiken#aiken
```

Via Aikup:

First, install Aikup: A basic cross platform utility tool to download and manage Aiken across multiple versions and for seamless upgrade. Once installed, simply run:

```
aikup
```

`aikup` alone installs the latest version available. You can install specific versions by specifying a version number.

From a package manager (NPM):

```
npm install -g @aiken-lang/aikup
```

Homebrew:

```
brew install aiken-lang/tap/aikup
```

From url (Linux & MacOs):

```
curl --proto '=https' --tlsv1.2 -LsSf https://install.aiken-lang.org | sh
```

From url (Windows):

```
powershell -c "irm https://windows.aiken-lang.org | iex"
```

However, the above guide is valid at the time of generating this material, always refer to the Aiken Language website for the most updated installation guide. Additionally, readers are required to take the eUTXO crash course found here.