

**CAPSTONE PROJECT 1**

**DEMARK - DECENTRALIZED MARKET**

Architecture Document

Version: 1.1

CODE: DEMARK-1.0

**Mentor:** Man Nguyen Duc

**Team member:**

Ha Truong Van

Dat Ngo Ha Van

Nhan Vo Hoang Quoc

Phuong Tran Nhat

**INTERNATIONAL SCHOOL**

# **Project Information**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Project acronym** | DEMARK | | | | |
| **Project Title** | Demark - decentralized market for digital assets | | | | |
| **Start Date** | Aug 31, 2018 | | **End Date** | Dec 10, 2018 | |
| **Lead Institution** | International School, Duy Tan University | | | | |
| **Project Mentor & contact details** | Man Nguyen Duc  Email: mannd@duytan.edu.vn  Tel: 0904 235 945 | | | | |
| **Product Owner** | Ha Truong Van | | | | |
| **Scrum Master** | Phuong Tran Nhat | | | | |
| **Team members** | **Name** | **Email** | | | **Tel** |
|  | Ha Truong Van | vanha30111997@gmail.com | | | 0969356097 |
|  | Dat Ngo Ha Van | ngohavandat93ndc@gmail.com | | | 01288446176 |
|  | Nhan Vo Hoang Quoc | quocnhan810@gmail.com | | | 01674559527 |
|  | Phuong Tran Nhat | nhatphuongb1@gmail.com | | | 01692502010 |

# **Proposal Document**

|  |  |  |  |
| --- | --- | --- | --- |
| **Document Title** | Architecture Document | | |
| **Reporting Period** | October 10, 2018 | | |
| **Team Information** | **Name** | **Role** |  |
|  | Nhan Vo Hoang Quoc | Team member |  |
|  | Ha Truong Van | Product owner & team member |  |
|  | Dat Ngo Ha Van | Team member |  |
|  | Phuong Tran Nhat | Scrum master & Team member |  |
| **Date** | October 10, 2018 | **Filename** | DEMARK\_ARCHITECTURE\_1.1.docx |
| **Access** | Project and Duy Tan University Program | |  |

# **Document History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **Document History** | |  |
| **Version** | **Date** | **Comments** | **Performed by** |  |
| V1.0 | September 10, 2018 | * Draft of document | Phuong Tran Nhat |  |
| V1.1 | October 10, 2018 | * Official document | Phuong Tran Nhat |  |

# **Document Approvals**

​The following signatures are required for approval of this document.

|  |  |  |
| --- | --- | --- |
| Man Nguyen Duc  *Mentor* |  | Date: |
| Ha Truong Van  *Product Owner* |  | Date: |
| Phuong Tran Nhat  *Scrum Master* |  | Date: |

**TABLE OF CONTENTS**

[**Project Information**](#_3znysh7)

[**Proposal Document**](#_3dy6vkm)

[**Document History**](#_4d34og8)

[**Document Approvals**](#_2s8eyo1)

[**1.** **Introduction**](#_17dp8vu)

[**1.1.** **Project Overview**](#_3rdcrjn)

[**1.2.** **Constraints**](#_26in1rg)

[1.2.1. Business Constraints](#_lnxbz9)

[1.2.2. Technical Constraints](#_35nkun2)

[**2.** **Requirement**](#_1ksv4uv)

[**2.1. Functional Requirements**](#_44sinio)

[**2.2. Architectural Drivers**](#_2jxsxqh)

[2.2.1. Utility Table](#_z337ya)

[2.2.2. Quality Attributes](#_3j2qqm3)

[**3.** **Overall Architecture**](#_2bn6wsx)

[**3.1. Context Diagram**](#_qsh70q)

[**3.2. Component & Connector View (C&C View)**](#_3as4poj)

[**3.3. Sequence Diagram**](#_1pxezwc)

[**3.4. Module View**](#_49x2ik5)

[**3.5. Class Diagram**](#_2p2csry)

[**3.6. Deployment View**](#_147n2zr)

# **Introduction**

* 1. **Project Overview**

Demark is a digital asset exchange decentralized to Universities, which provides universities with the ability to collect tuition fees and other financial activities. In addition, it offers students, investors the ability to pay tuition fees electronically and the ability to communicate with other users.

* 1. **Constraints**

### **1.2.1. Business Constraints**

* The system should be completed and handed over by December 10, 2018.
* The system should allow the holder the ability to payment of tuition fees by ETH
* The system should bonus when the holder buys a lot token.

### **1.2.2. Technical Constraints**

* The system should be developed on the basis of Ethereum.
* The system should use MetaMask software to validate and secure transactions
* The system should use the web3 protocol to connect to nodes on Ethereum.
* The system should use Solidity programming language to develop the smart contracts.

1. **Requirement**

The following lists of requirements are the functional and architectural requirements of our target system. These requirements will be used for evaluating the architecture based on the quality attributes of the system. The list of the requirements is prioritized and described with the most important requirements at the top and the least important requirements at the bottom.

## **2.1. Functional Requirements**

* FR1: The Demark shall provide users with a feature that allow them to transfer tokens to other users on the system.
* FR2: The Demark shall provide users with a feature that allow them to withdraw the ETH by burning the tokens which they’re holding.
* FR3: The Demark shall provide users with a feature that allow them to buy the token from Universities.
* FR4: The Demark shall provide users with a feature that allow them to send ETH to the deposit of the token.
* FR5: The Demark shall provide users who are the representative of a university with a feature that allows them to register to submit their university’s token to the system.
* FR6: The Demark shall provide users who are the manager of Demark a feature that allow them manage Tokens that was submitted on system.

## **2.2. Architectural Drivers**

### **2.2.1. Utility Table**

There are following quality attributes drive the design of architecture. Each quality attribute scenario is ranked with importance (I) defined by the Product Owner, and the estimated level difficulty (D). Both values are based on a scale of High (H) - Medium (M) - Low (L).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Attribute** | **Concerns** | **Scenarios** | **Priority** |
| QA01 | Performance | Response Time | Transaction response time from 15s to 20s | (H, H) |
| QA02 | Performance | Performance scale | It allows up to 10,000 transactions per second | (H, H) |
| QA03 | Security | Data Change Security | History of the transaction difficult to be attacked, altered. | (H, M) |
| QA04 | Modifiability | Change functionality during runtime | Updating changes does not affect other features | (H, M) |
| QA05 | Availability | Time needed to update the software, and so on | 2 9’s (99%) = up to 87.6h / 5256.0m / 315360.0 seconds of downtime per year. |  |

### **2.2.2. Quality Attributes**

*Table 1: Quality Attributes: Availability*

|  |  |
| --- | --- |
| **Quality Attributes:** Performance | **ID:** QA01 |
| **Stimulus** | Performs the transaction. |
| **Source(s) of the stimulus** | User |
| **Relevant environmental conditions** | Unexpected blackout. |
| **Architectural elements** | The system. |
| **System response** | The transaction states. |
| **Response measure(s)** | 15-20s. |

*Table 2: Quality Attributes: Performance*

|  |  |
| --- | --- |
| **Quality Attributes**: Performance | **ID**: QA02 |
| **Stimulus** | User access to the system |
| **Source(s) of the stimulus** | User |
| **Relevant environmental conditions** | During the using process. |
| **Architectural elements** | Load capacity up to 5000 users |
| **System response** | Normal operation |
| **Response measure(s)** | Delay of not more than 1 s |

*Table 3: Quality Attributes: Security*

|  |  |
| --- | --- |
| **Quality Attributes:** Security | **ID:** QA03 |
| **Stimulus** | Attack to the system |
| **Source(s) of the stimulus** | Hacker |
| **Relevant environmental conditions** | During using process. |
| **Architectural elements** | The transaction history does not change |
| **System response** | System |
| **Response measure(s)** | The correctly transaction history |

*Table 4: Quality Attributes: Modifiability*

|  |  |
| --- | --- |
| **Quality Attributes:** Modifiability | **ID:** QA04 |
| **Stimulus** | Change the functions during runtime |
| **Source(s) of the stimulus** | Demark manager |
| **Relevant environmental conditions** | During the time using process. |
| **Architectural elements** | The system. |
| **System response** | Updating changes does not affect other features |
| **Response measure(s)** | Page Reloading time |

*Table 5: Quality Attributes: Availability*

|  |  |
| --- | --- |
| **Quality Attributes**: Avalibility | **ID**: QA05 |
| **Stimulus** | Update, maintenance and warranty the system |
| **Source(s) of the stimulus** | Administrator |
| **Relevant environmental conditions** | Time needed to update the software, and so on. |
| **Architectural elements** | The system |
| **System response** | The friendly user interface is presented |
| **Response measure(s)** | * 2 9’s (99%) = up to 87.6h / 5256.0m / 315360.0 seconds of downtime per year. |

# **Overall Architecture**

This section shows the diagrams which clearly bounds our target system and describes the architecture and interaction between components.

## **3.1. Context Diagram**

The figure below clearly delineates the boundaries of the system and the actors interacting with the system.

**There are four actors interacting with the system:**

* The student acting as a normal user of the system. He has 4 main function is get reward, withdraw, buy token and transfer.
* The University Cashier is the cashier of university who has all function of a student and deposit function.
* University Manager is the representative of the university who request to submit their university token to the system. He has all function of a Cashier and a function to request submit token. Besides, he will receive the notification of token.
* The Demark Manager is the manager of the system, who has the all function on the system and a token management function.



*Figure 1: Demark Context Diagram*

## **3.2. Component & Connector View (C&C View)**

We mainly used a C&C view to argue and reason about architectural properties, quality attribute requirements, and functional requirements that the system must adhere to.

This view type partitions the system into components that have some runtime presence such as processes, objects, data stores, and connectors or that represent pathways of communication such as information flows and access to shared storage. The diagram below shows the architecture overview including Business Service Management components and other related components. We have representations and behaviors for important components in the following sections.



*Figure 3: Demark C & C View*

**Prose**

|  |  |
| --- | --- |
| **Element** | **Responsibilities** |
| Firebase Database | The database is a component which contains information on token, user. All data the system needs. |
| Metamask | A browser addon, which is a soft wallet, helps users keep their Ethereum private key and confirm the transaction information they make. |
| Infura | A node in the Ethereum network, which provides interactive services to the Ethereum network. |
| EtherScan | An Ethereum Explorer, which provides the API portal for retrieving information on the Ethereum network. |
| Demark Web Client | The Demark Web Client is a component that manages and implements interactive functions for users and other systems that are handled on the UI. Which interactive Metamask for processing the transaction information and authoring; interacts with Infura to perform Token-related transactions such as transfer, withdraw, deposit, buy token; Interacts with EtherScan to get the token details, transaction details, and wallet details. |
| Demark Web Service | The Demark Web Service is a component that manage and perform activities related to retrieval and storage of data such as get the token list, user authorization and receive the token submit forms. It also interacts with EtherScan to get deploy the Token and get token details. |

## **3.3. Sequence Diagram**

In this figure, a sequence diagram is used to display the sequence of activities. Sequence diagrams show the workflow from a start point to the finish point detailing the many decision paths that exist in the progression of events contained in the activity. They may be used to detail situations where parallel processing may occur in the execution of some activities.



*Figure 4: Demark Sequence Diagram*

## **3.4. Module View**

This view type partitions the system into a unique non-overlapping set of hierarchically decomposable implementation units (modules). The goal is to show how the source code is decomposed, as well as the dependencies between modules. In other words, it shows the decomposition and the “uses” relation.

The elements in this view type are UML packages or UML classes or interfaces. Packages are hierarchically decomposable; hence, placement of a class or package inside another package in a diagram implicitly indicates an is-part-of relation. This view type also shows dependencies among implementation units. It tells developers what other modules must exist in order for their portion of the system to work correctly.

This view type was selected because it helps the following roles:

● The project manager, who must define work assignments, form teams, and formulate project plans and schedule, knowing which modules are more critical in terms of dependencies.

● Testers who use the modules as their unit of work to create test cases and perform the tests.

● The configuration manager who is in charge of maintaining current and past versions of the units in consistent and functional package assemblies, being able to produce a running version of the system.

● Developers, who are required to implement the elements.

● Maintainers, who are tasked with modifying the software elements.





*Figure 5: Demark Module View*

**Prose**

|  |  |
| --- | --- |
| **Element** | **Responsibilities** |
| Contracts | Init the child function and attributes of the Token. |
| Node | Components and dependencies of Nodejs |
| App.js | The main component is used to run other components in Demark Web Service |
| Controller | The Controller is responsible for controlling the application logic and acts as the coordinator between the Router and the Model. The Controller receives an input from the users via the Router, then processes the user's data with the help of Model and passes the results back to the router. |
| Model | The model is the central component of the contract. It is the application's dynamic data structure. It directly manages the data, logic, and rules of the application. |
| Router | Config the API routes HTTP requests to controllers. |
| ReactJS Module | Components and dependencies of ReactJs |
| Ethereum Client | The component handles activities that interact with the Ethereum network via Infura and Etherscan |
| Components | The component handles activities that user interact on UI. |
| CSS | The CSS components |
| JS | The JS components that using to handle in UI. |
| Store | The components to handle the data storing |
| Actions | The components to handle the action of user and the system |

## **3.5. Class Diagram**

****

*Figure 6: Class diagram*

This figure 6 describes the composition of a smart contract, component composition, function, events, and interactions between components.

**Component description:**

* Interface: Interfaces in Solidity are a lot like abstract contracts, except that their functions cannot be implemented.
* Contract: A contract in the sense of Solidity is a collection of code (its functions) and data (its state) that resides at a specific address on the Ethereum blockchain.
* Event: Events allow the convenient usage of the EVM logging facilities, which in turn can be used to “call” JavaScript callbacks in the user interface of a dapp, which listen for these events.
* Is a: Is-A relationship in Contract Oriented Programming.
* Has a: Has-A relationship in Contract Oriented Programming
* Solidity mappings are a type that has some similarities to hash tables. Solidity mappings are used to structure value types for smart contracts.



Token (interface) is the ERC20 technical standard for smart contracts on Ethereum blockchain for implementing tokens.

The ERC-20 token has the following method-related functions:

The specific wording of the function is followed by a clarification of what it does, in [brackets]

1. totalSupply () public view returns (uint256 totalSupply) [Get the total token supply]
2. balanceOf (address \_owner) public view returns (uint256 balance) [Get the account balance of another account with address *\_owner*]
3. transfer (address \_to, uint256 \_value) public returns (bool success) [Send *\_value* amount of tokens to address *\_to*]
4. transferFrom (address \_from, address \_to, uint256 \_value) public returns (bool success) [Send *\_value* amount of tokens from address *\_from* to address *\_to*]
5. approve (address \_spender, uint256 \_value) public returns (bool success) [Allow *\_spender* to withdraw from your account, multiple times, up to the *\_value* amount. If this function is called again it overwrites the current allowance with *\_value*]
6. allowance (address \_owner, address \_spender) public view returns (uint256 remaining) [Returns the amount which *\_spender* is still allowed to withdraw from *\_owner*]

**Events format:**

1. Transfer (address indexed \_from, address indexed \_to, uint256 \_value). [Triggered when tokens are transferred.]
2. Approval (address indexed \_owner, address indexed \_spender, uint256 \_value) [Triggered whenever *approve(address \_spender, uint256 \_value)* is called.]



Implement (contract) is instance of ERC20 technical standard.

UniversityToken contract is instance of Implement contract (ERC20). It has attributes:

* name: the name of the token
* decimals: the number of decimal digits of the token
* symbol: the symbol of the token
* rating: the rating of the token
* totalSupply: The total supply of the token
* creator: address of the creator
* cashier: address of the cashier
* balance: the balance of the token
* allowed: The address of the wallets that are allowed to use a balance, within licensed limits.

**The functions:**

* onlyCreator modifier function: only creator can call this function
* \_enoughTime modifier function: limit time to get reward
* BuyToken event: event when user buy token
* FundTransfer event: event when contract give eth for user
* Deposit event: event when user deposit to token
* \_getRate: get rate bonus of token when user buy amount of token equal with this rate
* \_getRegister: set time start when user buy token until they get reward
* \_buy: function support for Buy Token
* buyToken: buy token function, call when user buy token
* burn: used to burn the token and refund the corresponding amount
* reward: Bonus when holder buy multiple tokens
* getState: get the state of the token

SafeMath contract provides secure calculator libraries consist of the operator: addition, subtraction, multiplication, division, mod.

## **3.6. Deployment View**

The deployment diagram models the run-time architecture of a system. It shows the configuration of the hardware elements (nodes) and shows how software elements and artifacts are mapped onto those nodes.



*Figure 7: Demark Deployment View*

**Prose**

|  |  |
| --- | --- |
| Element | Description |
| Firebase | Store and sync data with our NoSQL cloud database. Data is synced across all clients in real-time, and remains available when your app goes offline. Version 5.9.0 https://firebase.google.com/ |
| Metamask | Metamask is an extension on browser which allows you to run dApps right in your browser without running a full Ethereum node.  Browser: Brave, Chrome, Opera, Firefox  https://metamask.io/ |
| Infura | The decentralized applications utilize Infura for scaling and infrastructure.  Version 3.0  https://infura.io/ |
| EtherScan | Etherscan is a Block Explorer and Analytics Platform for Ethereum, a decentralized smart contracts platform.  https://etherscan.io/ |