An Asynchronous-Driven Node.js Based Intermediary-free Direct Deal Distribution Platform Converged with Cloud Service

SongYeon Lee[1] and JongHo Paik[2*]
[1] Department of Computer Science, Seoul Women's University
Seoul, Republic of Korea
[e-mail: sylee8466@swu.ac.kr]
[2] Department of Software Convergence, Seoul Women's University
Seoul, Republic of Korea
[e-mail: paikjh@swu.ac.kr]
*Corresponding author: JongHo Paik

Received February 27, 2019; revised April 26, 2019; accepted June 13, 2019; published August 31, 2019

Abstract

In this paper, a design and implementation for direct deal distribution platform is proposed to bypass the complex traditional distribution structure of agricultural market, as one of the fields where distribution patterns have changed. In the case of domestic agricultural distribution, demand and supply are unstable since the sales market is excessively concentrated in the designated wholesale market. Besides sales must go through multiple stages of distribution leading to problems in freshness and stability of agricultural products and downward pressure on profit margins for producers.

To solve the above mentioned issues, we propose a cloud service convergence direct deal distribution platform based on asynchronous-driven Node.js. The proposed platform can facilitate a variety of direct trading functions and also access to visualization information related to agricultural products, which may increase user confidence at an intermediary-free direct transactions platform. First, we describe the requirements of intermediary-free direct transactions of agricultural products and transaction entities. Next the database structure and transaction functions are designed and then implemented according to those requirements. Finally, an API based cloud convergence service structure is designed to provide the analyzed information to ensure a trustworthy system.

Keywords: Transaction platform, Node.js, cloud service, MVC pattern, server platform

A preliminary version of this paper was presented at ICONI 2018, and was selected as an outstanding paper.
This work was supported by a sabbatical year and research grant from Seoul Women's University(2019).
This manuscript is an addition based on the first author's master's thesis from Seoul Women's University.

http://doi.org/10.3837/tiis.2019.08.022 **ISSN : 1976-7277**

1. Introduction

As we enter the fourth industrial revolution, the distribution industry is undergoing a fundamental change. The sources of value and competitiveness are changing from "goods service transaction intermediation" to "knowledge and information about production consumption" [1]. Among other signs of this trend, the shift to online trading is making progress in the agricultural market. In the domestic agricultural distribution sector, demand and supply for the domestic agricultural distribution are unstable since the sales market is excessively concentrated in the designated wholesale market. Besides sales must go through multiple stages of distribution leading to problems in freshness and stability of agricultural products and downward pressure on profit margins for producers.

In response to this trend, a direct deal distribution platform is needed to directly connect producers and consumers and provide new value-producing services [2]. The features and transaction functions of each transaction entities must be defined. In a new form of direct deal distribution platform, there is an operating entity that manages the platform but does not generate revenue by engaging in transactions. By this reason, the database design must facilitate simple transactions between a producer and a consumer, and an intuitive user interface without distractions such as advertising. In order to reflect the above requirements, it is necessary to provide a function to directly confirm all sales and purchases, and related transactions.

Whenever developing a new distribution platform, however, it is essentially needed to verify stable service functions and to reduce additional costs due to modification and implementation of existing service functions. Recently, there is a growing interest in how to utilize a service platform that is already stable in providing commercial services when developing a new distribution platform. In this paper, we propose a design and implementation for cloud service convergence direct deal distribution platform based on Node.js with the characteristic of low cost and easy development. Particularly, Node.js is known to be faster I/O response than PHP/Apache server. And also Node.js can use npm (node package manager) which provides easy access to a vast set of open source library modules.

This paper is organized into five sections. In addition to this introduction, Section 2 explains the related works of the proposed platform. Section 3 presents the design methodology of the proposed platform. And section 4 explains the implementation method of the proposed platform based on Node.js. Finally conclusion is drawn in section 5.

2. Related Works

2.1 Node.js based server platform

Node.js is an asynchronous event-based runtime that runs in JavaScript, and is designed for scalable network application [3]. Since Node.js's runtime operates in a non-blocking I/O method, it does not happen that the process is deadlocked by processing multiple processes at the same time. According to the characteristic of JavaScript, the asynchronous I/O method can register a callback to each function and call a function as an event which can allocate a function to a variable. It is very effective for development of scalable systems [4, 5, 6].

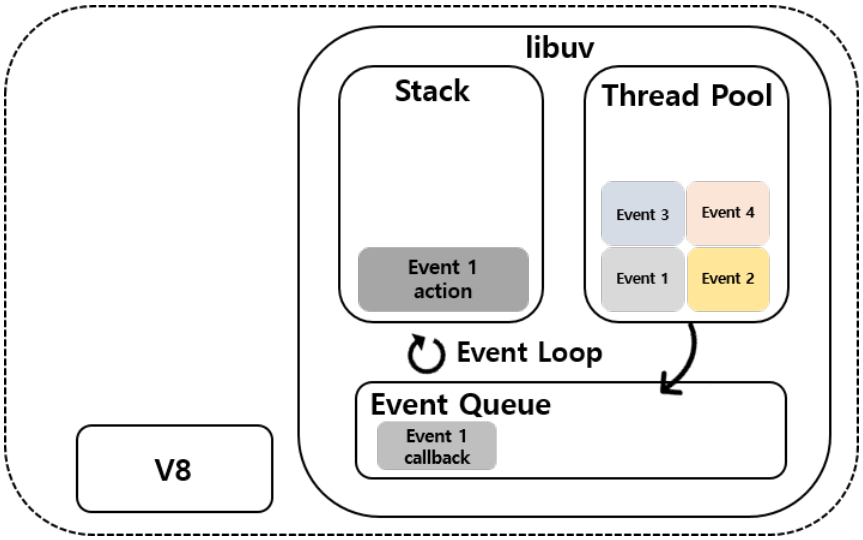

Fig. 1. Configuration and operating sequence of each component in Node.js

Node.js is designed to use multiple processes in a single thread to handle multiple tasks, so it is not necessary to add threads to implement asynchronous I/O operation. **Fig. 1** shows the configuration and operating sequence of each component for asynchronous operation in Node.js. Asynchronous I/O can be implemented through JavaScript compilation via the `V8` engine and through the `thread pool` and event loop of the `libuv` library. Owing to this asynchronous operation, Node.js is suitable for servers where I/O operations are frequently performed, rather than high CPU processing.

One aspect that makes Node.js extensibile is the use of the npm module which provides easy acces to a vast set of open source library modules. For another reason, existing server platforms such as Apache, nginx, and IIS are required to build a special web server engine. However, in the case of Node.js, there is no need to build a extra server engine because the web server is built in it. That is why Node.js is accessible and also scalable when constructing server platform.

In Node.js, both MongoDB(NoSQL) and MySQL(SQL) are commonly used as databases. In the case of MongoDB, unlike existing relational databases, there is no concept of table or column. Instead, a certain set of data in the form of document in a collection is stored in correspondence with a specified field. Unlike MySQL, it does not define attribute information for fields in a collection that acts as a table in the database, thus making it difficult to perform transactions or JOIN functions, but it is often used for scalability and ease of use. In addition, there is no restriction on the format of the data to be stored, and it is possible to store it in the JSON format. Therefore, MongoDB is suitable for servers that require some extent of consistency of data, but do not need the strict definition of relational database format.

2.2 Cloud based convergence service

In recent years, cloud computing services have been introduced in a variety of industries and public sectors, in order to find a survival strategy through cost reduction and to create new value by providing new services. In particular, various industries find that it reduces the maintenance burden on the IT infrastructure and the large initial investment cost as well.

Cloud services not only save cost but also increase the agility of the enterprise by enabling rapid creation of information services with pre-built professional services or service components available on demand. As a result, research on cloud services and converged service structures has been conducted in various fields [7, 8].

In order to utilize the convergence cloud service from an existing system, an application programming interface (API), a technology for client-to-agent connection, is required. In the cloud environment, the agent processor is connected to the existing platform through API to enable the provision of new services. The cloud agent is a processor in a virtual space, that the user need not worry about, but executes tasks just like ones own computer.

The cloud agent may respond with data representing visualizations for delivery to another module (closer to the user) which generates the actual visualization.

3. Design of cloud service convergence direct deal distribution platform based on Node.js

3.1 Architecture of the proposed platform

The Architecture of the proposed platform server is shown in **Fig. 2**. The platform is implemented in Node.js, which is an event-based asynchronous-driven server platform that can support many user connections at the same time. The proposed platform consists of cost-effective Node.js based on OMS system, which can share both social data analysis and cost-predictable platform. In the Node.js based server, MongoDB, which has high scalability and high processing performance, is used as a NoSQL type database. However, in the proposed platform, Oracle Database is used for secure transaction data including personal information. In this regard, although Node.js with a high speed of I / O processing is asynchronous, it requires a synchronous processing when interworking with the database especially when using Oracle Database.

The direct deal distribution platform basically performs the function of ordering the products registered by the producer and the basic order management system required when making transactions.

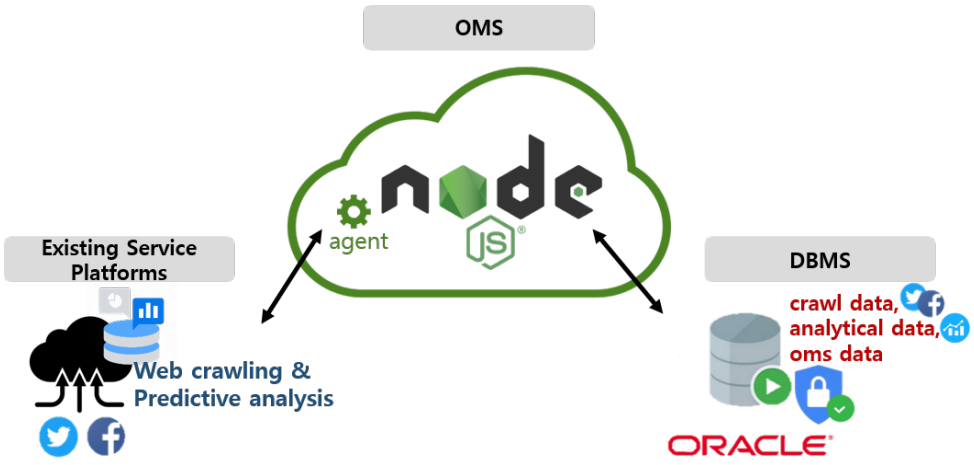

Fig. 2. The Architecture of the Proposed Direct Deal Distribution Platform based Node.js Server

In addition, it provides information about forecasted prices and demand for agricultural products and analyzes data based on direct transaction details, and obtains data on the quality or preference of the product based on the SNS analysis data. These visualization of analyzed data is performed using the agent provided by the existing service platform.

The implemented platform is adapted to the MVC model which is a software design pattern that is often used when developing server client system. Especially, the Model is run on the Oracle Database, View is provided by EJS/html, and Controller consists of Node.js's routing module. For the view configuration, we use the EJS template engine to dynamically display the data processed by the server. If server only shows static display, there is no need to use the template engine that makes up the view, however, in this platform due to various transactions the client side have to be organized with template engine.

As the size of the server grows, the number of routes that must be provided increases, and the increased route pages can lead to management complexity. Particularly, it is effective to separate and manage the functions to be provided through the routing module because the function to be provided differs according to the transaction subject in the direct deal distribution platform. Therefore, the implemented direct deal distribution platform provides the routing function for each page using the "express" web framework module of Node.js. A routing page is commonly used only for a main page that provides overall information of agricultural products using API agent before the transaction entities access their respective pages, and all other routing pages are separated by transaction entities.

3.2 MVC Model

In the MVC pattern, the user interface and the business logic are separated from each other, so that the maintenance of each component can be performed without affecting each other's role. Generally, an application in the Model area is responsible for data control.

Therefore, database storage and data query processing are independent of Controller and View. View takes the data received from model's query and displays it to the client-side browser. At this time, rather than the view requesting a query to the model directly if the controller requests a query to the model, the result of the request is transmitted to the View and finally data is displayed to the user. That is, the Controller acts as an intermediary between the data (Model) and the user interface (View). The MVC model is illustrated in **Fig. 3**, which consists of Oracle DB for Model, EJS and html for View, and Node.js server for Controller.

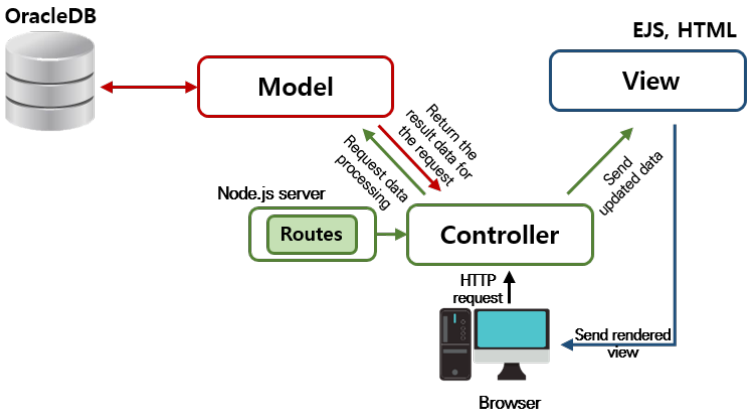

Fig. 3. Relationship between Model, View, Controller

First, when a user makes an HTTP request for data inquiry and the browser transmits it, the controller routes the HTTP request according to the condition of the router defined at the Node.js server. Next, according to the routing condition, the controller requests data processing from the model, receives the result from the model, and transmits the result to the view. Finally, view transmits the rendered view to the user's browser by reflecting the received data.

In other words, Oracle Database, acting as a model, independently performs the task of processing the data requested by the controller and maintaining the integrity of the data regardless of the state change of the controller and the view. Therefore, controller and EJS (view) and Oracle model all can be independently modified without affecting each other.

3.2.1 Model : Design of Database Structure

The data model designed in this paper is a logical data model, and the data model of the proposed direct transaction platform is designed to be applied to Oracle Database.

Table 1. Requirements for direct deal distribution between producer and consumer

Functional Classification	Function Details	Detail Requirements
Member Information	Definition of required membership information by transaction entity	The company information and the personal information of the person in charge should be collected
Product Information	Definition of information about products registered by the producer	Detailed transaction should be specified so that consumer can make purchasing decisions
Transaction Information	Definition of cart and order information	Information of the product to be ordered and the trading partner's membership information should be included
	Define order/shipping / check information for ordered goods	Checking products should be made based on shipping information

The data model should reflect the basic functions related to the characteristics of the direct transactions.

The requirements are defined in **Table 1**. First, the basic functions comprise member information collection, product specification and transaction function. The producer and the consumer can apply for the subscription based on their business information to use the platform and once approved by the administrator, the platform is available.

The producer directly registers the goods to be sold on the platform, and the consumer decides the purchase based on the registered goods. The information about the goods to be traded shall be specified in detail to provide everything the purchaser needs to make a decision. When the transaction occurs, information about the goods to be traded must be included with the transaction information so that accurate transaction details are provided to both the producer and the consumer.

The shipping information about the ordered product is stored according to the standard specified with the product registration, and the consumer can request the inspection item only for the goods that have been shipped.

3.2.2 View : Design of EJS/HTML based Bootstrap View

Node.js can use different template engine modules to display the rendered screen to the client. Through the template engine, HTML-based static code can be rendered with JavaScript. The template engine that can be used in Node.js varies, and the template engine applied in this case is EJS is often used with Node.js and can be combined with HTML syntax in one file. With EJS, dynamic code can be implemented using syntax such as variable declarations, loops, and conditional statements that could not be implemented in HTML alone.

In terms of screen composition, Bootstrap is a framework that helps to simplify development using libraries in frontend development. It provides a template based on HTML and CSS, and is composed of a grid-based responsive web so it is also optimized for mobile screen [9].

3.2.3 Controller : Design of page routing according to transaction entities

The transaction entities of the designed direct deal distribution platform are divided into the producer and consumer. Therefore, the routing module is divided into the modules used the producer and consumer in common those used by the consumer, and those used by the producer. Basically, the routing module branches the URI or HTTP request through the user's web browser, with the help of the “Express” module. **Fig. 4** shows the sequence diagram for routing the client’s request to the server**.**

As shown in **Fig. 4**, when the server starts for the first time, configuration information such as various database information, routing information, and server port information is loaded into the main server file. When the URI-based HTTP request is sent by the client's browser, the main server searches the registered routing information according to the Route Loader and forwards the request processing command to the corresponding routing module. Upon completion of the processing, the routing module delivers the result view to the client through the “response” object or redirects to a specific URL.

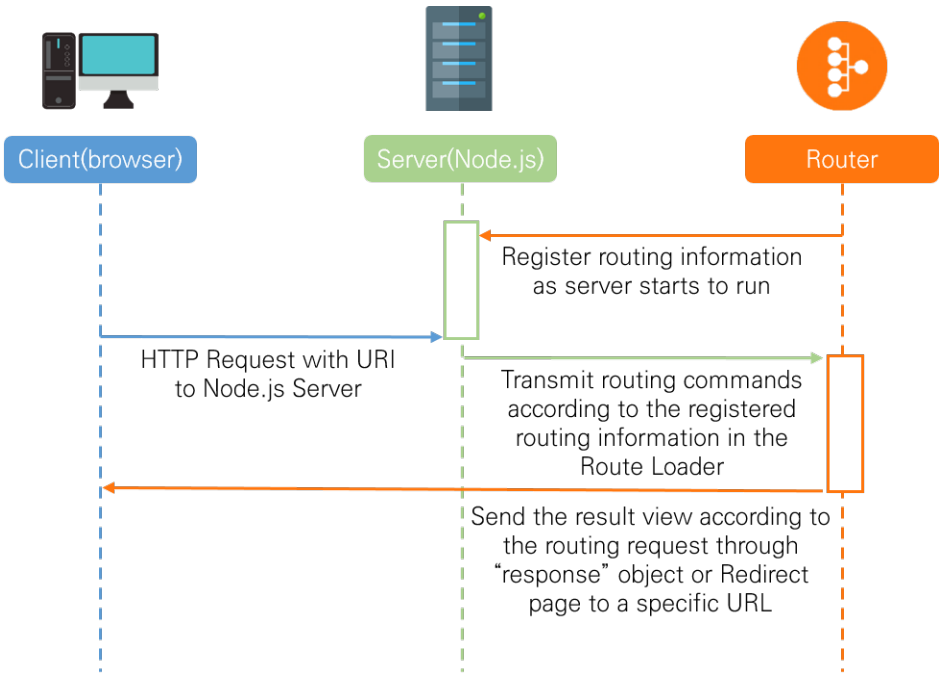

Fig. 4. Routing sequence diagram according to the request of the client

4. Implementation of cloud service convergence direct deal distribution platform based on Node.js

The direct deal distribution platform is developed in the Windows 10 Pro environment and the version of Node.js is v8.11.4 LTS with npm version 5.6.0. To link the database to the Node.js server, Oracle Database 11g Express Edition Release 11.2.0.2.0 version was used. The circulation data analysed with gathered agricultural data which are displayed on the main screen of the direct deal distribution platform is provided from RSN Co.,Ltd.

4.1 Model : Oracle DBMS

In order to secure transaction data and personal information from clients, the platform must be linked with the Oracle database. To link the Oracle DB with node.js, the "`node-oracledb`" module has to be installed in the Node.js server. **Fig. 5** shows the detailed database specification to securely access the Oracle DB.

The structure of the database implemented based on the requirements defined in the Model design is shown in **Fig. 6**. The database is composed of a total of five tables: the member information (T_MEMBER), the product information (T_PRODUCT), the order header information (T_ORDER_HEAD), the shopping cart information (T_BASKET), and the order detail, or product itemization (T_ORDER_DETAIL).

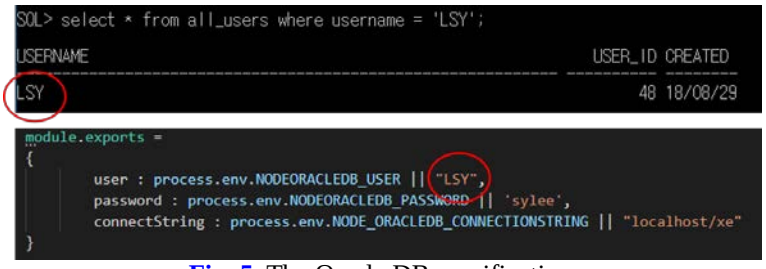

Fig. 5. The Oracle DB specification

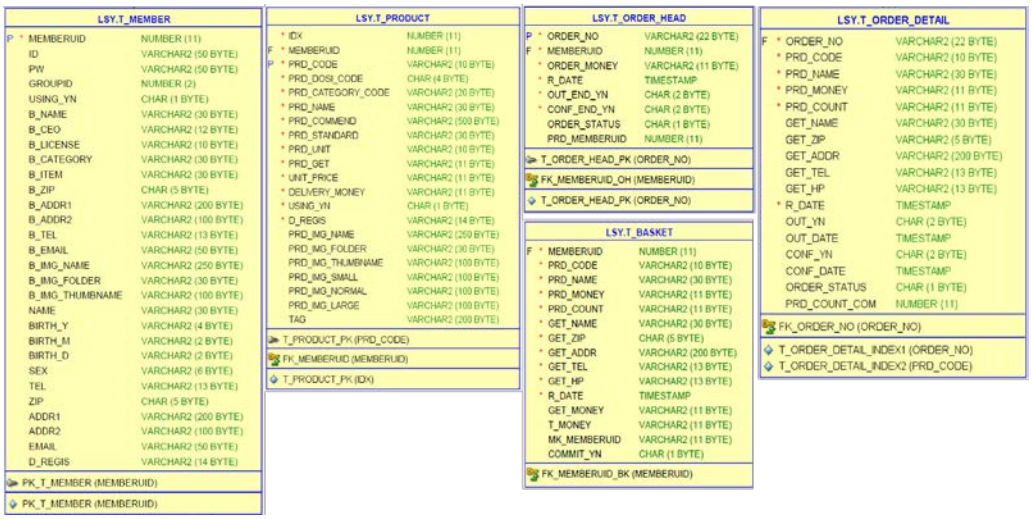

Fig. 6. The structure of implemented database

4.2 View: HTML5 & EJS

In order for the controller (Node.js's routing module) to render HTML data, the EJS template is used. When the data received through the `post` method data is rendered through the EJS template, the array values received from the arbitrary array can be put into the posts array through `for` and `forEach` statement. The screen thus rendered is shown in **Fig. 7**.

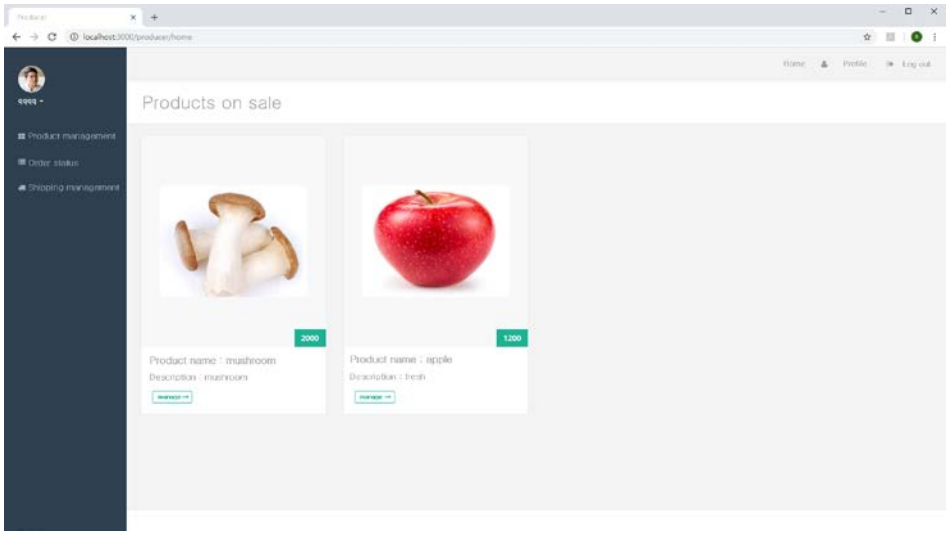

Fig. 7. The main screen of producer

When sending data from the client to the server in the `post` method, if the names of the dynamically generated tags are duplicated, only the last-set value is transmitted as the post value. To resolve this problem, set the name of the tag to array and send it. The array basically joins the data with commas but if the client input contains "," the value entered by the client may cause an error when splitting on the server. Therefore, before transmission, you must call the "`getArray()`" function and add a "/" mark as a delimiter to make it exactly split.

4.3 Controller: Node.js Server

The routing structure of the platform as implemented using the Express module of Node.js is shown in **Fig. 8**. The display of the visualization module and access to the corresponding page by transaction entity takes place in the main page. In the main screen, sign in and register are done in different routing pages for the producer and the consumer, and the producer's or consumer's main screen after sign in is different according to the assigned routing page.

On the producer routing page, the main functions provided are order management functions such as product registration, order status inquiry, and shipping registration. Then the consumer routing page provides functions such as product ordering, shopping cart registration, order status view, and product inspection management.

The detailed functions and data definitions of the routing page are shown in **Fig. 9** and **Fig. 10**, respectively. **Fig. 9** shows the functions and data definitions in the producer merchandise Management menu. With the product information entered by the producer, information such as the production area and the product classification is stored in the database in advance with the code chosen by the producer, and the stored data can be utilized when analyzing ecommerce statistics.

The functions and data definitions in the order status inquiry menu are shown in **Fig. 10**. Database tables used for order registration and query are T_ORDER_HEAD and T_ORDER_DETAIL. The order number is generated at the time the consumer makes an order decision and the order detail is stored together with the product information (product code, product name, product price, order quantity, destination information).

Fig. 8. The structure of a routing page

At this time, the initial shipment and inspection status for the order issue are combined. The membership number stored at the member information table is referenced in order to identify the customer placing the order and the producer of the product, and the corresponding member number acts as an identifier at the time of order inquiry.

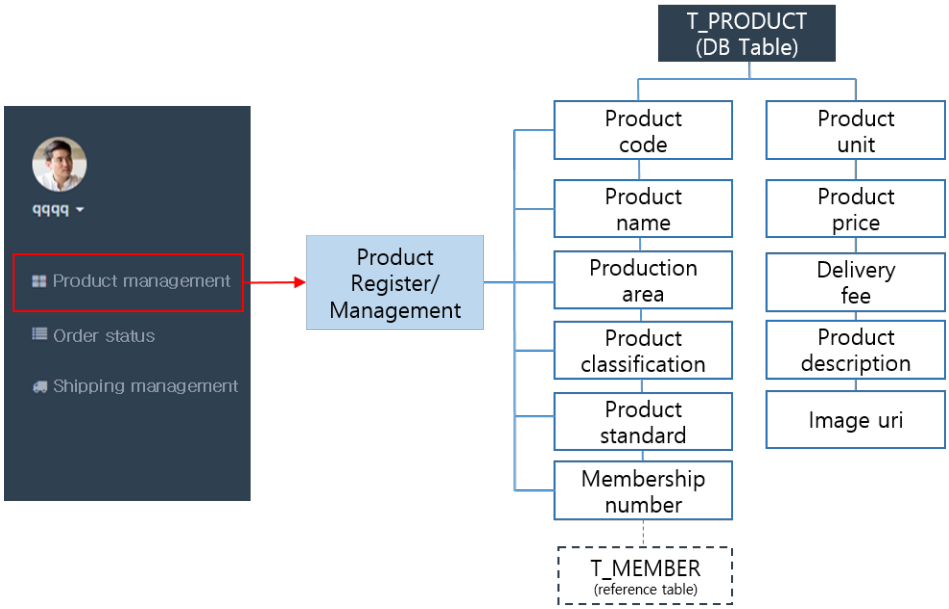

Fig. 9. The detailed functions and data definitions of product register and management

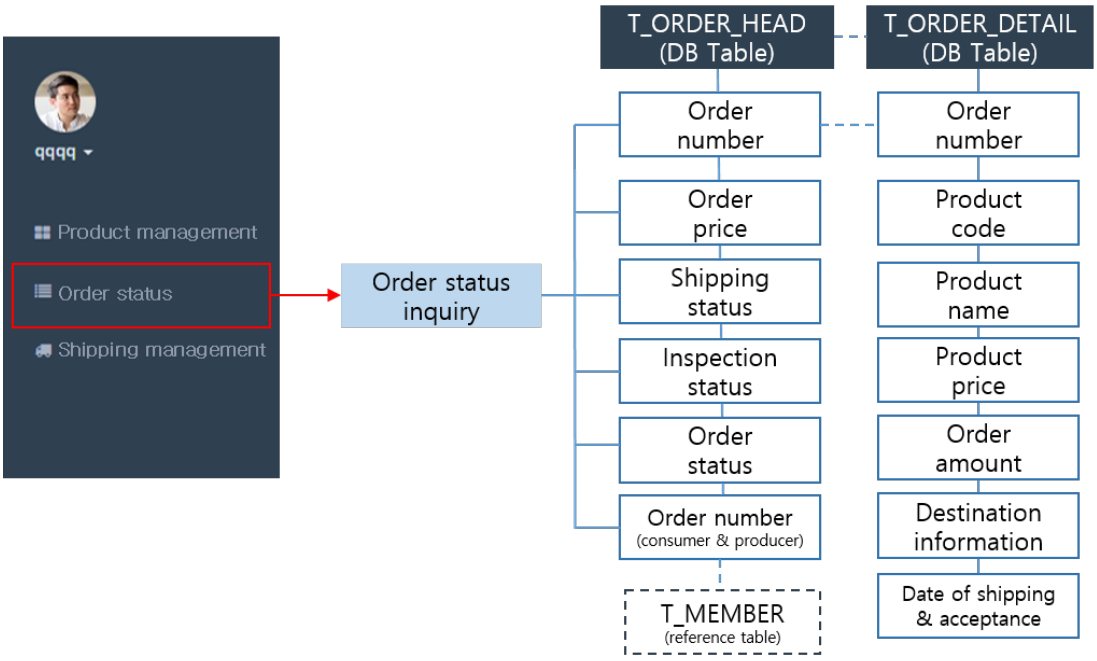

Fig. 10. The detailed functions and data definitions of order status inquiry

On the consumer's order status page, the order information for the goods can be displayed. If the goods have not yet been shipped, the order can be canceled and withdrawn. The order is withdrawn by updating the order status item of T_ORDER_HEAD and the order item is retained, but with a "deleted" status to maintain a record of the order. On the screen, only the valid order items are displayed according to the order status.

When an order event occurs according to the order number generation rule, the order number registered in the order table must be executed after the inquiry about the stored order item. However, there is a problem due to the asynchronous logic of Node.js, namely that the inquiry about the order item and the registration of the order item may not proceed sequentially, especially when there are several items to register.

Therefore, when Oracle DB or other database systems must be used in node.js, query execution such as `select, insert, delete, update` may not be done in sequence as the developer does not intend according to the asynchronous processing logic of node.js. Therefore, it is necessary to apply the async method to the function related to the db to perform the sequential query and each query has to be declared as a function when a user's post request requires multiple query execution. **Fig. 11** shows the flowchart of multiple query execution process in the Node.js server.

After declaring a function for each query execution, call the function in the specified order that they should be executed in `async.waterfall` by putting a db connection object and a callback as parameters [10]. In this case, db connection and release also have to be included in the `async.waterfall` module.

In the proposed platform, the order function in OMS requires multiple query execution. In order to perform the OMS function every item of order data, shipment or inspection data must be managed by sequence number made with fixed rules.

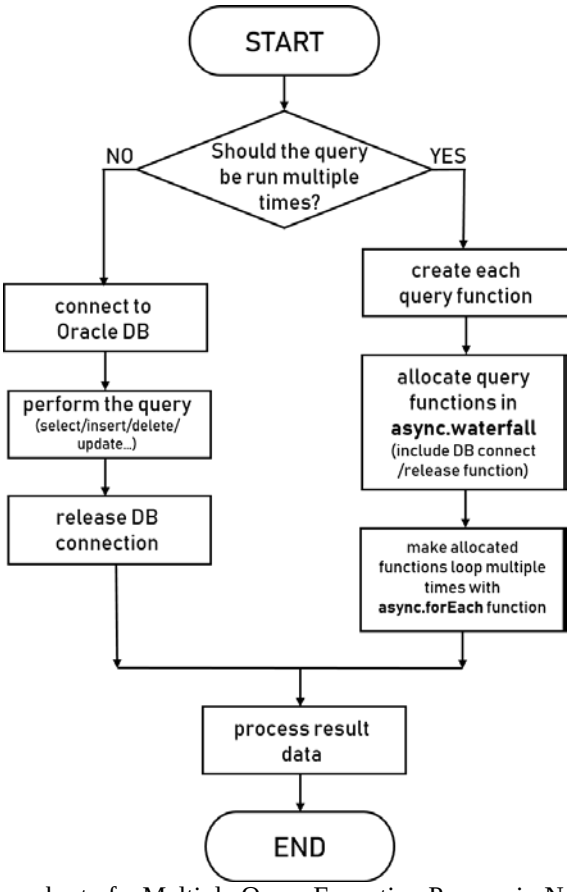

Fig. 11. The Flowchart of a Multiple Query Execution Process in Node.js Platform

In the shopping cart function, especially, a wholesaler can add multiple items registered by the producer. When the wholesaler orders all items that were in the shopping cart, the order procedure such as insert and update must be done in the regular sequence with the generated order number.

The order number must be generated using the selection result from the database and then query statements such as select, insert and update have to be executed in sequence.

4.4 Agent-based API in cloud service convergence environment

Fig. 12 shows the detailed configuration of the agent for applying the analysis service to the cloud service convergence direct deal distribution platform proposed in this paper. A data collection and analysis server exists outside the Node.js server. After performing a series of operations on the server, the visualization result is displayed in the form of API on the platform of the Node.js server [11]. In the analysis server, collected and analyzed data on SNS data related to agricultural products, distribution articles, etc., are stored in a seperate database, and then processed by the second data processing. Once the secondary processing of the data is completed, the collected data and processed data are stored again in the database, and the analysis data is derived from the real-time analysis platform based on the data.

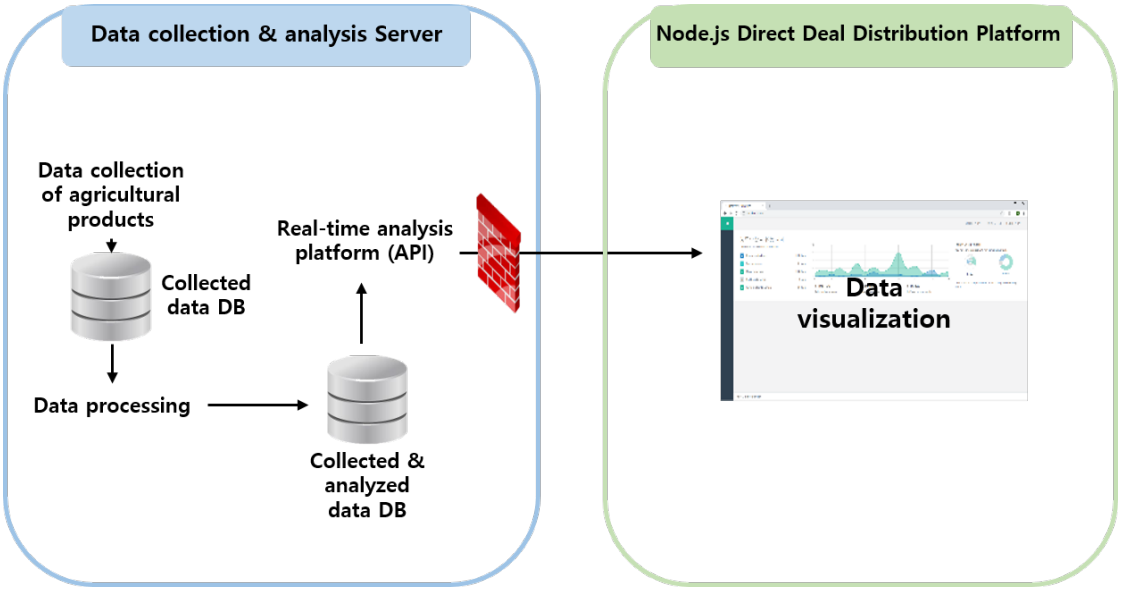

Fig. 12. Detailed configuration of agent for applying analysis service

Analyzed and processed data is called in API format, and when the corresponding API is called in the main page of the Node.js server, the API engine displays the visualization of the transmitted data. At this time, the visualization module is provided together with the analysis data so that the analyzed data can be displayed in the correct form.The main page of the Node.js direct transaction platform with the agent API provided is shown in **Fig. 13** [12].

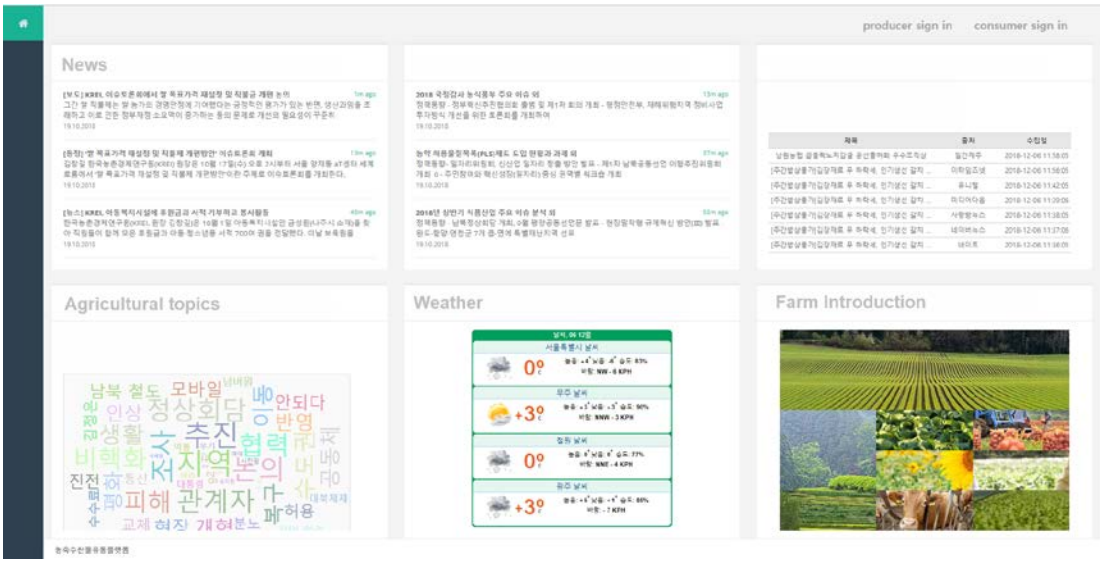

Fig. 13. Node.js direct deal platform main screen with agent API applied

5. Conclusion

In this paper, we proposed the design and implementation of cloud service convergence direct deal distribution platform based on Node.js for agricultural products. Agricultural distribution is under pressure to change by the trend of distribution 4.0, especially in the direction of sharing knowledge and information about production and consumption. In the domestic agricultural market, due to its concentration on the traditional wholesale market, demand and supply are unstable and the price competitiveness declines, therefore the distribution structure needs to be improved. To improve the distribution structure, we proposed direct transaction for agricultural products. At this time, in order to provide a competitive direct deal distribution platform, a variety of transaction information services related to agriculture are provided to the user along with the direct transaction function for user confidence.

In order to implement such a cloud convergence direct deal distribution platform, a Node.js based server platform is designed and implemented based on the MVC pattern, and the requirements of direct transactions of agricultural products are defined. And also, a new structure of database and user interface design is required for implementation of intermediary-free direct transactions.

We described the design of the platform with Oracle Database as the Model, EJS as the View and Express module as the Controller. Based on this routing structure, the screens of the Node.js platform are presented as implementation results. In addition, we analyzed and designed the API based cloud convergence service structure for the related distribution information service which can provide trustworthiness to the users in the direct transactions of agricultural products. We then presented the structure and screen of the platform applying the API.

The cloud-based direct deal distribution platform presented in this paper enables direct trade between a producer and a consumer, facilitating provision of agricultural products at a reasonable price without going through a complex distribution structure. In addition, it can increase the interest and confidence in the use of the platform by providing the trading partner with a wide range of information about transactions or products in the related field. If a new distribution service is provided as a Node.js based server platform to which the agent API is applied, it is expected to be applicable to the development of a platform that is easy to maintain and has a verified cloud-based service integrated with the platform.

References

[1] E-Mart Distribution Industry Research Institute, 2018 distribution industry outlook, 2017. 11.
[2] Sang Tae Kim, “A Study on the Design and Development of a Mobile Convergence Real-Time Distribution Platform,” *Ph.D Thesis Yeungnam University*, 2017.
[3] Node.js. https://nodejs.org.
[4] Ioannis K. Chaniotis, Kyriakos-Ioannis D. Kyriakou, Nikolaos D. Tselikas, “Is Node.js a viable option for building modern web applications? A performance evaluation study,” *Journal Computing Archive*, Vol.97 Issue 10, pp.1023-1044, 2015. Article (CrossRef Link).
[5] Magnus Madsen, Frank Tip, Ondřej Lhoták, “Static analysis of event-driven Node.js JavaScript applications,” in *Proc. of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp.505-519, 2015.
Article (CrossRef Link).

[6] Daniele Bonetta, Luca Salucci, Stefan Marr, "GEMs : Shared-Memory Parallel Programming for Node.js," in *Proc. of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 531-547, 2016. Article (CrossRef Link).
[7] Kwang Kyu Seo, "Analysis of Technical Factors for Multidisplinary Cloud Service Model based on Use Case," *Society of Korea Industrial and Systems Engineering*, vol. 10, no. 10, pp. 545-550, 2012. Article (CrossRef Link).
[8] Jeong-Yeop Kim and Eun-Ju Kim, "Public Sector Cloud Computing Adoption Policy and Status," *Communications of the Korean Institute of Information Scientists and Engineers*, Vol. 32, No. 2, pp. 32~39, 2014. Article (CrossRef Link).
[9] Bootstrap. https://getbootstrap.com/.
[10] Node.js. https://github.com/nodejs
[11] RSN Co.,Ltd, "Local SW commercialization social data API specification," 2018.
[12] Song Yeon Lee, "Design and Implementation of Cloud Service Convergence Direct Deal Distribution Platform Based on Node.js," *Master's thesis, Seoul Women's University*, 2019.

SongYeon Lee received the B.S and M.S. degrees in Computer Science from Seoul Women's University, Korea, in 2017 and 2019 respectively. She is currently working toward the Ph.D. degree in the Department of Computer Science from Seoul Women's University. Her research interests are in the areas of web-based platform design, IoMT platform design and recommendation system design based on machine learning as well.

JongHo Paik received the B.S., M.S., and Ph.D. degrees in the school of Electrical and Electronic Engineering from Chung-Ang University, Seoul, Korea, in 1994, 1997, and 2007, respectively. He was a Director with Advanced Mobile Research Center at Korea Electronics Technology Institute (KETI) by 2011. Since 2011, he is currently an associate professor in the department of Software Convergence, Seoul Women's University, Seoul. His research interests are in the areas of web-based communication, software testing, wireless/wired communications system design, video communications system design and system architecture for realizing advanced digital communications system and for advanced mobile broadcasting networks as well.