

A Self-Service Ordering System and Method

Technical Field

[0001] The application relates to the field of intelligent catering technology, particularly to a self-service ordering system and method.

Background

[0002] With the development of technology, self-service ordering systems have been gradually applied in various restaurants. Generally, customers can communicate with cloud servers through mobile phones or ordering machines to obtain ordering pages for placing orders. The cloud server stores various food item descriptions, prices, meal combinations, and flavor settings from merchants.

[0003] The self-service ordering systems in existing technology are not yet perfect. For example, when the network is slow, problems such as order submission jittering (i.e., customers clicking to submit the same order multiple times), lack of printer management, and slow page loading can easily occur. These problems not only reduce the customer's ordering experience but also decrease ordering efficiency.

Summary of the Invention

[0004] The embodiments of the application aim to provide a self-service ordering system and method to solve the problem of imperfect self-service ordering systems in existing technology.

[0005] To solve the above technical problems, the embodiments of the application provide the following technical solutions:

[0006] In one aspect, the application provides a self-service ordering system. The system comprises: an ordering terminal, configured to obtain user-selected food items to generate an order request and send the order request to a cloud server; the cloud server, configured to receive the order request, determine whether the order request meets preset order generation conditions, and when the order request meets the preset order generation conditions, generate an order based on the order request and send the order to a printing server, wherein the order generation conditions comprises: a current system state is in an orderable state and the order request has not yet generated a corresponding order; the printing server, configured to generate multiple food item printing tasks according to attributes of each food item in the order, and send each food item printing task to a first printer corresponding to its food item attributes; multiple first printers, configured to execute printing operations, according to the food item printing tasks sent by the printing server.

- [0007] Optionally, when the current system state is in a closing state, a message indicating that the system is in a closing state is returned to the ordering terminal;
when the current system state is in the orderable state and the order request has already generated the corresponding order, the state of the already generated order is obtained and returned to the ordering terminal.
- [0008] Optionally, the printing server is further configured to send a failed food item printing task to a preset second printer for printing, when the food item printing task has failed to print.
- [0009] Optionally, the printing server is deployed on a Raspberry Pi and uses ESC/POS commands to communicate with the first printer and the second printer.
- [0010] Optionally, the cloud server is further configured to monitor a query frequency of each food item, and when the query frequency for a food item reaches a preset frequency threshold, perform high-speed caching of interface elements required to display that food item.
- [0011] Optionally, the cloud server is further configured to compares a food item information in the order request with the food item information on unpaid orders, when the order request meets the order generation conditions; and generate a collision order based on the order request, when the food item information in the order request is the same as the food item information on any unpaid order, wherein the collision order is settled at a preset collision discount price;
when the food item information in the order request is not the same as the food item information on any unpaid order, the cloud server is further configured to generate a regular order based on the order request.
- [0012] In another aspect, the application provides a self-service ordering method. The method comprises: obtaining user-selected food items to generate an order request and sending the order request to a cloud server, by an ordering terminal;
receiving the order request, and determining whether the order request meets preset order generation conditions, and when the order request meets preset order generation conditions, generating an order based on the order request and sending the order to a printing server, by the cloud server; wherein the order generation conditions comprises a current system state is in an orderable state and the order request has not yet generated a corresponding order;
generating multiple food item printing tasks according to attributes of each food item in the order, and sending each food item printing task to a first printer corresponding to its food item attributes, by the printing server;
wherein printing operations are executed by each first printer prints according to the food item printing tasks sent by the printing server.

- [0013] Optionally, the method further comprises sending the failed food item printing task to a preset second printer for printing by the printing server, when the printing server detects that a food item printing task has failed to print.
- [0014] Optionally, the method further comprises monitoring a query frequency of each food item, and when the query frequency for a food item reaches a preset frequency threshold, performing high-speed caching of interface elements required to display that food item, by the cloud server.
- [0015] Optionally, the method further comprises comparing a food item information in the order request with the food item information on unpaid orders by the cloud server, when the order request meets the order generation conditions; and generating a collision order based on the order request by the cloud server, when the food item information in the order request is the same as the food item information on any unpaid order; wherein the collision order is settled at a preset collision discount price; generating a regular order based on the order request by the cloud server, when the food item information in the order request is not the same as the food item information on any unpaid order.
- [0016] The beneficial effects of the embodiments of the application is that: unlike the prior art, the embodiments of the application provide a self-service ordering system that includes: an ordering terminal, configured to obtain user-selected food items to generate an order request and send it to a cloud server; a cloud server, configured to receive the order request, determine whether the order request meets preset order generation conditions, and if so, generate an order based on the order request and send the order to a printing server, wherein the order generation conditions include that the current system state is in an orderable state and the order request has not yet generated an order; a printing server, configured to generate multiple food item printing tasks according to the attributes of each food item in the order, and send each food item printing task to a first printer corresponding to its food item attributes; multiple first printers, configured to print according to the food item printing tasks sent by the printing server. By adopting the system of the application, the jittering problem caused by slow networks is resolved, and printers are uniformly managed, enabling different food items to be printed to different production departments. This not only improves the customer's ordering experience but also enhances production efficiency.

Description of the Drawings

- [0017] One or more embodiments are illustrated by way of example in the corresponding figures of the accompanying drawings. These exemplary illustrations do not constitute limitations on the embodiments. Elements in the drawings with the same reference

numerals represent similar elements unless otherwise stated. The drawings are not to scale unless specifically noted.

[0018] Figure 1 is a structural diagram of a self-service ordering system according to an embodiment of the application;

[0019] Figure 2 is a flow chart of a self-service ordering method according to an embodiment of the application.

Detailed Description of the Embodiments

[0020] To make the purpose, technical solutions, and advantages of the application's embodiments clearer, the technical solutions in the embodiments of the application will be described clearly and completely in conjunction with the accompanying drawings. Obviously, the described embodiments are only a part of the embodiments of the application, not all of them. Based on the embodiments in the application, all other embodiments obtained by those skilled in the art without creative effort are within the scope of protection of the application.

[0021] Moreover, the technical features involved in the various embodiments of the application described below can be combined with each other as long as they do not conflict with each other.

[0022] It should be noted that the steps shown in the flowcharts of the accompanying drawings can be executed in a computer system, such as a set of computer-executable instructions. Although a logical sequence is shown in the flowcharts, in some cases, the steps shown or described can be executed in an order different from that described here.

[0023] Embodiment 1

[0024] Please refer to Figure 1, which shows the structure diagram of a self-service ordering system according to the embodiment. The system includes an ordering terminal 10, a cloud server 20, a printing server 30, and multiple first printers 40.

[0025] The ordering terminal 10 can be various electronic devices used for ordering, such as mobile phones, tablets, or ordering machines. When the ordering terminal 10 is a mobile device like a phone or tablet, users can scan a QR code on the dining table or provided by the merchant to enter the self-service ordering program. When the ordering terminal 10 is the merchant's ordering machine, users can enter the self-service ordering program by selecting a table number and entering the number of diners.

[0026] In one embodiment, the ordering homepage of the self-service ordering program supports switching between Traditional Chinese, Simplified Chinese, and English, and allows selection of dine-in or takeaway. The left side of the ordering homepage shows menu categories, while the right side displays detailed food item descriptions for the

corresponding category, including à la carte items or set meals. The food item description shows the food image on the left and the item name and price on the right. If an item is sold out, it displays "Sold Out"; if inventory quantity is set, it shows how many portions are left. If there are required options for set meals or individual items, the bottom right corner displays "Select Options", otherwise it shows a "+" button for customers to quickly add to the cart (the "+" button increases quantity, the "-" button decreases quantity). If there are flavor options or set meal choices, clicking "Select Options" or the left image enters the detailed selection mode for the item, such as multiple optional or required flavors, multiple optional or required side dishes, increasing item quantity, etc. After completing the selection, the item can be added to the cart. For new orders, the number of diners needs to be entered. In the shopping cart, customers can increase or decrease item quantities or remove items as needed. After confirmation, click "Submit Order" to generate a new order or add to an existing order. In one embodiment, before submitting the order, the total amount of the current order is displayed (if the total amount exceeds 1000 yuan, there will be a second prompt indicating that the current consumption amount has exceeded 1000 yuan). After confirmation, the user's selected items are configured to generate an order request sent to the cloud server 20. The order request can be for a first-time order or an additional order, and may include user ID, food item information, table number, number of diners, order time, etc. The food item information may include item name, price, flavor, etc.

[0027] The cloud server 20 is deployed in the cloud and stores various food item information. It establishes communication connections with the ordering terminal 10, printing server 30, and ordering management platform (not shown in the figure). The cloud server 20 interacts with the ordering terminal 10 to support self-service ordering, payment, and other functions; it interacts with the printing server 30 to send orders for print allocation; and it interacts with the ordering management platform to support food item management, order management, cashier operations, account management, and other functions.

[0028] After receiving an order request from the ordering terminal 10, the cloud server 20 checks the system state and the order request to determine whether the order request meets preset order generation conditions. If so, it generates an order based on the order request and sends the order to the printing server 30. The order generation conditions include that the current system state is in an orderable state and the order request has not yet generated a corresponding order. Preferably, the cloud server 10 generates an order number based on the order time in the order request.

[0029] In one embodiment, the self-service ordering system supports a clearing function. System state includes closing state and orderable state. Merchants can set daily clearing

times in the ordering management platform. When the clearing time is reached, the system enters closing state, conducting daily business analysis and generating multiple closing reports. To ensure data integrity, when the system is in closing state, customer ordering is not supported, and a message indicating the system is in closing state is returned to the ordering terminal 10.

[0030] When the network is slow, customers may click to submit the same order multiple times due to lack of response. To prevent jittering, the cloud server 20 also determines whether an order request has already generated an order. Specifically, it can match the order time, user ID, and food item information in the order request with the corresponding information of generated orders. If there's a match, it determines that the order request has already generated an order, and the cloud server 20 does not generate a new order. Preferably, the cloud server 20 queries the state of the already generated order and returns it to the ordering terminal 10, which generates corresponding prompt messages based on the state. The state of the generated order includes in production, paying, paid, cancelled, etc.

[0031] The printing server 30 is deployed on a Raspberry Pi, running a self-developed central printing dispatch service that uses low-level ESC/POS commands, TCP/IP protocol, and asynchronous communication to communicate with printers. The small data packet size of ESC/POS commands allows printers to respond quickly and print receipts. To improve merchant production efficiency and ensure accuracy, the printing server 30 generates multiple food item printing tasks based on the attributes of each item in the order and sends each printing task to the corresponding first printer 40 based on the item's attributes. For example, beverages are printed to the bar, grilled items are printed to the Western kitchen, etc.

[0032] The first printers 40 can be thermal receipt printers, deployed in different locations according to food item attributes, and executes printing operation in accordance with the food item printing tasks sent by the printing server 30.

[0033] In one embodiment, the self-service ordering system also includes backup printers (i.e., second printers). When the printing server 30 detects a failed food item printing task (e.g., network failure, open cover, paper jam, etc. of the first printer 40), it transfers the failed printing task to a designated backup printer. Multiple levels of backup printers can be set up, searching for the next level if one fails, until all levels have been exhausted.

[0034] For frequently queried food item information, Redis high-speed caching is used in the cloud server 30 to improve data query and read times and enhance user experience. In one embodiment, the cloud server 30 monitors the query frequency of each food item in real-time. When the query frequency reaches a preset threshold, it caches various interface elements for displaying the food item. These interface elements

include food images, videos, text descriptions, etc. In other embodiments, the cloud server 30 pre-caches various interface elements needed to display historically popular items with high query frequencies.

[0035] In one embodiment, the self-service ordering system provided by the application also supports a meal clash function. Specifically, after the cloud server 20 receives an order request from the ordering terminal 10, it checks the system state and the order request. When the order request meets the order generation conditions, it compares the food item information in the order request with the food item information on unpaid orders. If the food item information in the order request matches the food item information on any unpaid order, it generates a meal clash order based on the order request, which is settled at a preset discounted price for meal clashes; otherwise, it generates a regular order based on the order request.

[0036] It can be understood that the self-service ordering system provided by the application also includes an ordering management platform. Through this platform, merchants can manage the provided food items, including management of item images, text descriptions, prices, and dish updates, set meal combinations, flavor settings, listing times, merchant quantities, etc. The text descriptions of food items can include Traditional Chinese, Simplified Chinese, and English versions. Additionally, the ordering management platform supports ordering, table management, cashier operations, printing, and other functions. In table management, the real-time state of each table and the number of bills to be settled can be displayed in different colors, and additional orders can be placed or orders can be cancelled. When cancelling an order, double verification is required. When using the ordering management platform for cashier operations, the cashier can enter the corresponding checkout interface by scanning the QR code on the customer's food record slip or entering the table number. After entering the checkout interface, the order enters a locked state. When an order is locked, quick additional orders for certain miscellaneous items (such as egg tarts, plastic bags, etc.) are supported, but self-service ordering, additional ordering, order cancellation, etc. cannot be performed to ensure data integrity. In one embodiment, the checkout interface supports various discount methods, combined payment methods, different point redemptions, food coupon redemptions, etc., and displays the remaining amount in real-time. After payment is completed, if cash is used, the cash drawer opens immediately, and a payment receipt is printed in real-time. Payment receipts can also be reprinted, with "Reprint" marked on the ticket.

[0037] The self-service ordering system provided by this embodiment of the application includes an ordering terminal for obtaining user-selected food items to generate an order request and send it to a cloud server; a cloud server for receiving the order request, determining whether the order request meets preset order generation

conditions, and if so, generating an order based on the order request and sending the order to a printing server, where the order generation conditions include that the current system state is in an orderable state and the order request has not yet generated an order; a printing server for generating multiple food item printing tasks according to the attributes of each food item in the order, and sending each food item printing task to a first printer corresponding to its food item attributes; multiple first printers for printing according to the food item printing tasks sent by the printing server. By adopting this system, the jittering problem caused by slow networks is resolved, and printers are uniformly managed, enabling different food items to be printed to different production departments. This not only improves the customer's ordering experience but also enhances production efficiency.

[0038] Embodiment 2

[0039] According to the application example, a flow chart for self-service ordering is provided, as shown in Figure 2, which is a flow chart of a self-service ordering method according to the application example, specifically including:

[0040] Step S201: The ordering terminal obtains the user's selected food items to generate an order request and sends it to the cloud server.

[0041] As mentioned above, the order request can be for a first-time order or an additional order, and may include user ID, food item information, table number, number of diners, ordering time, etc. The food item information may include the name of the dish, price, flavor, etc.

[0042] Step S202: After receiving the order request, the cloud server determines whether the order request meets the preset order generation conditions. If so, it generates an order based on the order request and sends the order to the printing server.

[0043] The order generation conditions include that the current system state is in an orderable state and the order request has not yet generated a corresponding order. System state includes machine closing state and orderable state. When the system is in the machine closing state, it does not support customer ordering and returns a message to the ordering terminal indicating that the system is in the machine closing state.

[0044] To prevent jitter, the cloud server also determines whether an order has already been generated for the order request. Specifically, it can match the ordering time, user ID, and food item information in the order request with the corresponding information of generated orders. If the match is successful, it is determined that an order has already been generated for this order request, and the cloud server will not generate a new order. Preferably, the cloud server 20 queries the state of the already generated order and returns the state of the already generated order to the ordering terminal. The ordering terminal generates corresponding prompt information based on the state of

the already generated order. The state of the already generated order includes in production, in payment, paid, cancelled, etc.

[0045] Step S203: The printing server generates several food item printing tasks according to the attributes of each food item in the order, and sends each food item printing task to the first printer corresponding to its food item attributes.

[0046] The printing server is deployed on a Raspberry Pi, running a self-developed central printing dispatch service, using low-level ESC/POS commands, TCP/IP protocol, and asynchronous communication to communicate with printers. The small data packet size of ESC/POS commands allows printers to respond quickly and print receipts. To improve the merchant's production efficiency and ensure accurate production, the printing server generates several food item printing tasks according to the attributes of each food item in the order and sends each food item printing task to the first printer corresponding to its food item attributes. For example, sending beverage prints to the bar area, grilled items to the Western kitchen, etc.

[0047] In one embodiment, backup printers (i.e., second printers) can also be set up. When the printing server detects that a food item printing task has failed (such as network failure, front cover open, paper jam, etc. of the first printer), it transfers the failed food item printing task to the designated backup printer. Multiple levels of backup printers can be set up. When a first-level backup printer fails, it looks for a second-level one; when the second-level backup printer fails, it continues to look for a third-level backup printer, until all levels are exhausted before terminating.

[0048] Step S204: Each first printer executes printing operations according to the food item printing task sent by the printing server.

[0049] For food item information with high query frequency, to improve data query and read time and enhance user experience, Redis high-speed caching is used in the cloud server. In one embodiment, the cloud server monitors the query frequency of each food item. When the query frequency reaches a preset frequency threshold, it caches the interface elements for displaying the food item for high-speed access. The interface elements include food item images, videos, text descriptions, etc. In other embodiments, the cloud server pre-caches various interface elements needed to display historically popular food items with high query frequencies.

[0050] In one embodiment, the self-service ordering system provided in the application also supports a meal collision function. Specifically, after the cloud server receives an order request from the ordering terminal, it checks the system state and the order request. When the order request meets the order generation conditions, it compares the food item information in the order request with the food item information on unpaid orders. If the food item information in the order request is the same as the food item information on any unpaid order, it generates a collision order based on the order

request, and the collision order is settled at a preset discounted price for collision orders; otherwise, it generates a regular order based on the order request.

[0051] Through the description of the above implementation methods, those skilled in the art can clearly understand that each implementation method can be realized by means of software plus a general hardware platform, and of course, it can also be realized through hardware. Based on this understanding, the essential part of the above technical solutions or the part that contributes to the related technology can be embodied in the form of a software product. The computer software product can be stored in a computer-readable storage medium, such as ROM/RAM, magnetic disk, optical disc, etc., including several instructions to enable a computer device (which can be a personal computer, server, or network device, etc.) to execute the methods described in each embodiment or parts of the embodiments.

[0052] Finally, it should be noted that: the above embodiments are only used to illustrate the technical solutions of the application, not to limit them; under the idea of the application, the technical features between the above embodiments or different embodiments can also be combined, steps can be implemented in any order, and there are many other changes in different aspects of the application as described above. For the sake of brevity, they are not provided in detail; although the application has been described in detail with reference to the preceding embodiments, those skilled in the art should understand: they can still modify the technical solutions recorded in the preceding embodiments, or replace part of the technical features equally; and these modifications or replacements do not make the essence of the corresponding technical solutions deviate from the scope of the technical solutions of the embodiments of the application.

Claims

1. A self-service ordering system, characterized in that the system comprises:
an ordering terminal, configured to obtain user-selected food items to generate an order request and send the order request to a cloud server;
the cloud server, configured to receive the order request, determine whether the order request meets preset order generation conditions, and when the order request meets the preset order generation conditions, generate an order based on the order request and send the order to a printing server, wherein the order generation conditions comprises: a current system state is in an orderable state and the order request has not yet generated a corresponding order;
the printing server, configured to generate multiple food item printing tasks according to attributes of each food item in the order, and send each food item printing task to a first printer corresponding to its food item attributes;
multiple first printers, configured to execute printing operations, according to the food item printing tasks sent by the printing server.
2. The system according to claim 1, characterized in that,
when the current system state is in a closing state, a message indicating that the system is in a closing state is returned to the ordering terminal;
when the current system state is in the orderable state and the order request has already generated the corresponding order, the state of the already generated order is obtained and returned to the ordering terminal.
3. The system according to claim 1, characterized in that,
the printing server is further configured to send a failed food item printing task to a preset second printer for printing, when the food item printing task has failed to print.
4. The system according to claim 3, characterized in that, the printing server is deployed on a Raspberry Pi and uses ESC/POS commands to communicate with the first printer and the second printer.
5. The system according to claim 1, characterized in that,
the cloud server is further configured to monitor a query frequency of each food item, and when the query frequency for a food item reaches a preset frequency threshold, perform high-speed caching of interface elements required to display that food item.
6. The system according to any one of claims 1 to 5, characterized in that,
the cloud server is further configured to compares a food item information in the order request with the food item information on unpaid orders, when the order request meets the order generation conditions; and generate a collision order based on the order request,

when the food item information in the order request is the same as the food item information on any unpaid order, wherein the collision order is settled at a preset collision discount price;

when the food item information in the order request is not the same as the food item information on any unpaid order, the cloud server is further configured to generate a regular order based on the order request.

7. A self-service ordering method, characterized in that the method comprises:
obtaining user-selected food items to generate an order request and sending the order request to a cloud server, by an ordering terminal;
receiving the order request, and determining whether the order request meets preset order generation conditions, and when the order request meets preset order generation conditions, generating an order based on the order request and sending the order to a printing server, by the cloud server; wherein the order generation conditions comprises a current system state is in an orderable state and the order request has not yet generated a corresponding order;
generating multiple food item printing tasks according to attributes of each food item in the order, and sending each food item printing task to a first printer corresponding to its food item attributes, by the printing server;
wherein printing operations are executed by each first printer prints according to the food item printing tasks sent by the printing server.

8. The method according to claim 7, characterized in that the method further comprises:
sending the failed food item printing task to a preset second printer for printing by the printing server, when the printing server detects that a food item printing task has failed to print.

9. The method according to claim 7, characterized in that the method further comprises:
monitoring a query frequency of each food item, and when the query frequency for a food item reaches a preset frequency threshold, performing high-speed caching of interface elements required to display that food item, by the cloud server.

10. The method according to any one of claims 7 to 9, characterized in that the method further comprises:
comparing a food item information in the order request with the food item information on unpaid orders by the cloud server, when the order request meets the order generation conditions; and
generating a collision order based on the order request by the cloud server, when the food item information in the order request is the same as the food item information on any unpaid order; wherein the collision order is settled at a preset collision discount price;

generating a regular order based on the order request by the cloud server, when the food item information in the order request is not the same as the food item information on any unpaid order.