8/22/2017

ExpressFoods

Use Case definitions, Data Model description and some Queries.

By

Dennis Vinther Jensen

Use Case definitions

Name	View Order
Brief description	This use case allows a user to view information about an order. A client can view only the orders created by the client. A deliverer can view all orders with the status “new” and all orders assigned to the deliverer.
Precondition	User is logged in and has the role of either client or deliverer.
Postcondition	Orders accessible to the user are displayed.
Actors	User
Initiator	This use case starts when a user accesses the View Orders Page.
Basic flow	1. SELECT ORDER: The user is presented with a list containing the orders that the user can access. For each order, the user has the option to “View Order”. If user is a deliverer, then the following extra options are available for each order: “Assign Order”, “In transit”, “Delivezred”. 2. VIEW ORDER: The user clicks “View Order” to view information for the order and is presented with the following data for the selected order: status, created datetime, delivered datetime (if delivered), ETA datetime, price, delivery address, menu items and amount. The use case ends.
Alternate flow	1. ASSIGN ORDER: At basic flow step SELECT ORDER, the user selects “Assign Order” which initiates the Manage Order use case. The use case ends. 2. IN TRANSIT: At basic flow step SELECT ORDER, the user selects “In transit” which initiates the Manage Order use case. The use case ends. 3. DELIVERED: At basic flow step SELECT ORDER, the user selects “Delivered” which initiates the Manage Order use case. The use case ends.

Name	Create Order
Brief description	This use case allows a user to create a new order in the system.
Precondition	User is logged in and has the role of client.
Postcondition	A payed for order with a delivery address and menu items are added to the system. The

	inventory reflects that the ordered menu items has been sold (they are subtracted from the inventory).
Includes	Choose Menu Items Set Delivery Address Pay for Order Update Inventory
Actors	Client
Initiator	This use case starts when a user clicks “Create new Order”.
Basic flow	1. CREATE NEW ORDER: A new order related to the current user is created in the system. 2. CHOOSE MENU ITEMS: The Choose Menu Items use case is executed. 3. SET DELIVERY ADDRESS: The Set Delivery Address use case is executed. 4. PAY FOR ORDER: The Pay for Order use case is executed. 5. ORDER STATUS: The user is redirected to the View Order Page. The use case ends.

Name	Choose Menu Items
Brief description	This use case allows a client to choose which menu items an order should contain.
Precondition	User is logged in and has the role of client.
Include	Update Inventory
Postcondition	Menu items are added to the order.
Actors	Client
Initiator	This use case is initiated by the Create Order use case.
Basic flow	1. PRESENT MENU ITEMS: The client is presented with 4 menu items (2 dishes and 2 desserts) and buttons for each menu item which can be used to add or subtract a menu item to/from the order. Below the menu items are buttons for submitting the selection and for cancelling the order. 2. ADD MENU ITEMS: The client adds menu items to the order in any combination and amount. 3. SUBMIT SELECTION: The user indicates that the menu item selection is done by submitting the selection which is saved to the order.

	4. UPDATE INVENTORY: The Update Inventory use case is executed. The use case ends.
Alternate flow	1. At basic flow step 1, the user has selected to cancel the order. The use case ends.

Name	Set delivery Address
Brief description	This use case allows a user to set the delivery address for an order.
Precondition	User is logged in and has the role of client.
Postcondition	A delivery address is defined for the order. ETA is set on the order.
Actors	Client
Initiator	This use case is initiated by the Create Order use case.
Basic flow	1. PRESENT ADDRESS FORM: The user is presented with input fields that allows for entering a street name, a street number, and a ZIP code. Buttons for submitting the address and for cancelling the order are available below the form. 2. SET ADDRESS: The user enters the desired delivery address into the form fields. 3. SUBMIT ADDRESS: The user indicates that the delivery address has been entered by clicking submit. The address is saved to the order. 4. Travel time by bike from ExpressFood to the delivery address is retrieved from the MapSystem and ETA for the order is set. The use case ends.
Alternate Flow	1. At basic flow step 1, the user has selected to cancel the order. The use case ends.

Name	Pay for Order
Brief description	This use case allows the user to pay for an order.
Precondition	User is logged in and has the role of client.
Postcondition	The order is payed for. Date and time for when the order was created (payed for) is set on the order.
Actors	Client, PaymentSystem
Initiator	This use case is initiated by the Create Order use case.
Basic flow	1. PRESENT PAYMENT FORM: The user is presented with a form for entering

	payment information. This is supplied by the Payment System Vendor. A button to cancel the order is also available. 2. SELECT PAYMENT FORM: The user selects a payment form. 3. ENTER DATA: The user enters the data required by the payment form. 4. SUBMIT PAYMENT: The user submits the payment form and thereby pays for the order. The use case ends.
Alternate Flow	1. At basic flow step 1, the user has selected to cancel the order. The use case ends.

Name	Manage Order
Brief description	This use case allows a deliverer to manager an order by assigning the order or updating the status.
Precondition	User is logged in and has the role of deliverer.
Postcondition	The order is assigned or has an updated status.
Actors	Deliverer
Initiator	Action buttons in the View Orders Page named “Assign Order”, “In Transit”, and “Delivered”.
Basic flow	1. PERFORM ACTION: one of the following actions is performed based on the action button. a. ASSIGN ORDER: The order is marked as assigned to the deliverer. The use case ends. b. IN TRANSIT: The order is marked an in transit. The use case ends. c. DELIVERED: The order is marked as delivered and the timestamp for delivery is set on the order. The use case ends.

Name	Update Inventory
Brief description	This use case allows the system to update the inventory.
Precondition	Menu item IDs and amount to be added/subtracted is required.
Postcondition	Inventory is updated to reflect the menu items in storage.
Actors	System
Initiator	This use case is initiated by either the use case Choose Menu Items or the use case Manage Menu Items.

Basic flow	1. GET INVENTORY: Menu items and the current amount is retrieved from the database. 2. UPDATE INVENTORY: The amount of menu items defined by the initiating use case is added/subtracted to/from the inventory. 3. SAVE INVENTORY: The inventory database table is updated to reflect the new calculated inventory. The use case ends.

Name	Manage Menu Items
Brief description	This use case allows a user to define which menu items are on the menu. Inventory can be updated via Extension Point.
Precondition	User is logged in and has the role of chef.
Postcondition	Menu items selected for the menu is marked as being on the menu.
Extension Points	Update Inventory
Actors	Chef
Initiator	This use case is initiated by a Chef visiting the Update Menu Items Page.
Basic flow	1. DISPLAY UPDATE FORM: The user is presented with a form in which he can select 4 menu items in all (2 mains and 2 desserts) These are available as drop-down selections and is set automatically to reflect the items in inventory (if any). Furthermore, there is a field to enter how many of the menu items are on storage. This field is also automatically filled with the amount of the menu item on storage (if any). 2. UPDATE MENU ITEMS: The user selects the menu items that should be on the menu and, if needed, defines how many are on storage. 3. SUBMIT MENU: The user submits the form by clicking submit and the menu items selected are marked as being on the menu. If storage count has also been defined, the Update Inventory use case is initiated. The use case ends.

Data Model description

To translate the inheritance from the class diagram (from user to client, chef, and deliverer) to a database, a table has been created for each one. This is to avoid repeating the first_name, last_name columns for each user type (possible more columns in the future).

Encoding

- **utf8mb4**: Support non-BMP (Basic Multilingual Plane) Unicode characters.

Tables

- **user**
  - PK is user_id
- **client**
  - PK is client_id
  - FK constraint (cascade) between **client.user_id** and **user.user_id.** This is to ensure that the related client of any deleted user is also deleted.
- **chef**
  - PK is chef_id
  - FK constraint (cascade) between **chef.user_id** and **user.user_id.** This is to ensure that the related chef of any deleted user is also deleted.
- **deliverer**
  - PK is deliverer_id
  - FK constraint (cascade) between **deliverer.user_id** and **user.user_id**. This is to ensure the related deliverer of any deleted user is also deleted.
- **delivery_address**
  - PK is delivery_address_id
  - FK constraint (restrict) between **delivery_address.client_id** and **client.client_id**. This is to avoid having delivery addresses not related to any clients.
- **client_order**
  - PK is client_order_id
  - FK constraint (restrict) between **client_order.client_id** and **client.client_id.** This is to avoid having orders with no relation to a client, due to the client being deleted.
  - FK constraint (restrict) between **client_order.deliverer_id** and **deliverer.deliverer_id.** This is to avoid having orders with no relation to a deliverer, due to the deliverer being deleted.
  - FK constraint (restrict) between **client_order.delivery_address_id** and **delivery_address.delivery_address_id**. This is to avoid an order losing its delivery address if an address is deleted.
- **menu_item**
  - PK is menu_item_id
- **order_item**
  - PK is client_order_id and menu_item_id (compound)

Compound private key is used to avoid having the same menu_item registered for the same order more than once. This will instead be registered in the amount column.
    - FK constraint (restrict) between **order_item.client_order_id** and **client_order.client_order_id**. This is to avoid having orphaned order items if an order is deleted.
    - FK constraint (restrict) between **order_item.menu_item_id** and **menu_item.menu_item_id**. This is to avoid having order items with no related menu_item, if a menu_item is deleted.
- **meal_category**
    - PK is meal_category_id
- **category_item**
    - PK is menu_item_id and meal_category_id (compound)
    - Compound PK is used to avoid having the same menu item registered for the same category more than once.
    - FK constraint (restrict) between **category_item. meal_category_id** and **meal_category. meal_category_id**. This is to avoid having orphan category items if a meal category is deleted.
    - FK constraint (restrict) between **category_item. menu_item_id** and **menu_item. menu_item_id**. This is to avoid having any category items not being related with a meal, if the meal category is deleted.
- **refrigerator**
    - PK is refrigerator_id
- **inventory**
    - PK is menu_item_id and refrigerator_id (compound)
    - Compound PK is used to avoid having the same meal registered in the same refrigerator several times.
    - FK constraint (restrict) between **inventory.menu_item_id** and **menu_item.menu_item_id**. This is to avoid having items in the refrigerator not being related with a menu item.
    - FK constraint (restrict) between **inventory.refrigerator_id** and **refrigerator.refrigerator_id**. This is to avoid having orphaned items in the refrigerator if a refrigerator is deleted.

Queries

Select all clients with name, email address, and phone number.

```
SELECT user.first_name, user.last_name, client.email_address,
client.phone_number

FROM client

INNER JOIN user

ON client.user_id = user.user_id
```

Select the order_id of all orders by clients whose first name starts with an “A”. Include same info as above.

```
SELECT user.first_name, user.last_name, client.email_address,
client.phone_number, client_order.client_order_id
FROM client
INNER JOIN user
ON client.user_id = user.user_id
INNER JOIN client_order
ON client_order.client_id = client.client_id
WHERE user.first_name LIKE 'A%'
```

As above with order items

```
SELECT user.first_name, user.last_name, client.email_address,
client.phone_number, client_order.client_order_id, menu_item.name,
order_item.amount
FROM client
INNER JOIN user
ON client.user_id = user.user_id
INNER JOIN client_order
ON client_order.client_id = client.client_id
INNER JOIN order_item
ON order_item.client_order_id = client_order.client_order_id
INNER JOIN menu_item
ON order_item.menu_item_id = menu_item.menu_item_id
WHERE user.first_name LIKE 'A%'
```

Today’s menu

```
SELECT menu_item.menu_item_id, menu_item.name, meal_category.name as category
FROM menu_item
INNER JOIN category_item
ON category_item.menu_item_id = menu_item.menu_item_id
INNER JOIN meal_category
ON category_item.meal_category_id = meal_category.meal_category_id
WHERE menu_item.on_menu = 1
```