Additional Specs for Phase 3

In phase 3, you will have to create models for `Item`, `ItemPrice`, `Employee`, `User`, and `OrderItem` and write unit tests for so that all models have 100 percent test coverage. In addition, you will need to modify the previous models so they conform to the final ERD given in this phase. This will require you to write migrations which change the database schema; do ***not*** simply edit existing migrations to accommodate these changes. To make sure you can build these models and tests properly, here are some specs for each of these models:

Items must:

1. have all proper relationships specified.
2. have values which are the proper data type and within proper ranges. This includes:
    a) a weight greater than zero.
    b) units per item being an integer greater than zero.
3. must be either in the category of 'breads', 'muffins', or 'pastries', with those categories represented as an enum, with the default being 'breads'.
3. have a name and all names must be unique in the system *(watch lower/upper cases – case differences alone shouldn't make it unique).*
4. have the following scopes:
    a) 'active' -- returns only active items.
    b) 'inactive' -- returns all inactive items.
    c) 'alphabetical' -- orders results alphabetically.
    d) 'breads' -- returns all items for the breads category.
    e) 'muffins' -- returns all items for the muffins category.
    f) 'pastries' -- returns all items for the pastries category.
5. have the following methods:
    a) 'current_price' -- which returns the current price of the item. If no price has been set, then should return `nil`, not zero.
    b) 'price_on_date' -- which returns the price of the item on a particular date. If no price has been set for that date, then should return `nil`, not zero.
    c) 'make_active' -- which flips an active boolean from inactive to active.
    d) 'make_inactive' -- which flips an active boolean from active to inactive.
6. can only be deleted if the item has never been shipped to a customer. If removed, any unshipped and unpaid orders items should also be removed.

Item Prices must:

1. have all proper relationships specified.
2. have values which are the proper data type and within proper ranges; most notably, a price can be zero (free) or higher, but it cannot be a negative number.
3. have the following scopes:
    a) 'current' -- returns only current prices.
    b) 'for_date' -- returns all prices for a specified date *(parameter: date).*
    c) 'for_item' -- returns all prices for a specified item *(parameter: item_id).*
    d) 'chronological' -- orders results chronologically with most recent price changes at the top of the list.
4. when a new item price is created, the date is automatically set to the next day (prices can't change midday so the earliest it could kick in would be the next day).
5. when a new item price is created, the end date of the previous price is automatically set to today.
6. restrict item_ids to items which exist and are active in the system.
7. not be deleted for any reason.

Order Items must:

1. have all proper relationships specified.
2. have values which are the proper data type and within proper ranges. This includes:
    a) a quantity greater than zero.
    b) a shipping date which can be blank, but otherwise is a legitimate date.
3. have the following scopes:
    a) 'shipped' -- returns only order items that have been shipped.
    b) 'unshipped' -- returns only order items that have not been shipped.
4. have an instance method called 'subtotal' that finds the subtotal (quantity * price) of a given order item. The method takes a date as an optional parameter (if none given, then should default to current date) so subtotals of past order items are calculated correctly. The date passed must be either the current date or a past date; if the date is in the future the method should return nil.
5. restrict item_ids to items which exist and are active in the system.

Employees must:

1. have all proper relationships specified.
2. have a social security number that is a 9 digit number, but the user may input values with dashes, spaces or other common formats – e.g., 999-99-9999; 999 99 9999; 999999999 are all acceptable. Social security numbers are unique to each person and should be unique in the system.
3. has a date hired which is either today or some day in the past (forecasting future hires is not allowed).
4. if a date terminated exists, it must be a legitimate date that is after the date hired as well as today or some day in the past (again, no forecasting future terminations before they happen).
5. must automatically deactivate the user associated with the employee if the employee is made inactive and the change must be saved when complete.
6. have the following scopes:
   a) 'active' -- returns only active users.
   b) 'inactive' -- returns all inactive users.
   d) 'alphabetical' -- orders results alphabetically by last, first name.
7. have the following methods:
   a) 'make_active' -- which flips an active boolean from inactive to active.
   b) 'make_inactive' -- which flips an active boolean from active to inactive.
   c) 'name' -- which returns the employee last name, first name.
   d) 'proper_name' -- which returns the name as "first_name last_name".
8. not be deleted for any reason.

Users must:

1. have all proper relationships specified.
2. have the role of either 'customer', 'baker', 'shipper', or 'manager', with those roles represented as an enum, with the default being 'customer'.
3. have usernames that are unique in the system *(and case insensitive).*
4. have passwords that are at least 4 characters in length.
5. must be properly confirmed (password_confirmation must match password) before saving.
6. have the following scopes:
   a) 'active' -- returns only active users.
   b) 'inactive' -- returns all inactive users.
   c) 'employees' -- returns all non-customer users.
   d) 'alphabetical' -- orders results alphabetically by username.
   e) 'customer_role' -- returns all users who are customers.

   f) 'manager_role' -- returns all users who are managers.
   g) 'baker_role' -- returns all users who are bakers.
   h) 'shipper_role' – returns all users who are shippers.
7. have the following methods:
   b) 'make_active' -- which flips an active boolean from inactive to active.
   c)'make_inactive' -- which flips an active boolean from active to inactive.
8. not be deleted for any reason.

Customers must (in addition to previous requirements):
1. not be deleted for any reason.
2. have a relationship to the User class.
3. have a method called 'billing_address' which returns the active billing address for a customer.
4. make a customer's user inactive if the customer is made inactive and the change must be saved when complete.

Addresses must (in addition to previous requirements):
1. may be destroyed only if there are no orders associated with the address and it is not an active billing address. In either case, the system must give the user appropriate feedback.

Orders must (in addition to previous requirements):
1. have a relationship to OrderItem.
2. have a method called 'is_editable?' which returns false if there are no unshipped order items associated with the order; return true otherwise
3. have a class method called 'not_shipped' which returns all the distinct orders that have one or more order items that remain unshipped; similar to a scope, what is returned is an `ActiveRecordRelation` and not an array of objects per se.
4. have an instance method called 'unshipped_items' which returns all the items from a particular order that have not yet shipped.
5. have an instance method called 'total_weight' which calculates the total weight of all the items in the order.
6. have an instance method called 'shipping_costs' which calculates the shipping costs for this particular order. More on shipping costs calculations can be found in the Shipping service tests.
7. have a method called 'credit_card_type' which returns the credit card code (AMEX, VISA, MC, etc.) of the card being used in the order. If there is no valid code, this particular method should return "N/A".

8. be able to accept `credit_card_number`, `expiration_year` and `expiration_month` as properties of an order (with necessary getter and setter methods) and verify the card's type (VISA, MC, etc.) and its expiration date. These properties are not saved in the database.
9. separate validations (and appropriate error messages) for invalid credit card numbers and invalid expiration dates.
10. the payment receipt needs to be modified so that it is a base64 encoded string of the format "order: <the order id>; amount_paid: <grand total>; received: <the order date>; card: <card type> ****<last 4 digits>; billing zip: <billing zip code>".
11. orders should be destroyable only if no item(s) associated that order have been shipped; otherwise they must be kept in the system. If an order is destroyed, all the order items must be destroyed as well.
12. have a callback exists that automatically sets the order date to today when a new order is created. In addition, disable previous date validation as this field is now handled by the callback.

Please note: all foreign keys in all models are required and appropriate steps should always taken to make sure they are present.