Additional Specs for Phases 4

In phase 4 you will have to create models for `Shift`, `Job`, `ShiftJob`, `PayGrade` and `PayGradeRate`, as well as modify `Store`, `Assignment` and `Employee` and write unit tests for these models. To help you build these models and tests, here are some specs for each of the models:

Shifts must:

1. have all proper relationships specified
2. values which are the proper data type and within proper ranges
   [Note: there is currently an open issue posted on github with the newest version of `validates_timeliness` gem; some nonsense strings resolve to "`2000-01-01 00:00:00 UTC` which would make it appear to be a valid time. For now you simply need to ensure that end_time is after start_time and that dates are either today or sometime in the future for new shifts.]
3. be added to only current assignments, not past assignments
4. can only be deleted if the status of the shift is pending
5. have a method called 'report_completed?' which returns true or false depending on whether or not there are any jobs associated with that particular shift
6. *new* shifts should have a callback with automatically sets the end time to three hours after the start time, if an end time is not already set
7. must have a method called 'duration' which calculates the length of the shift in terms of hours (*note that rounding rules specified in the phase 1 narrative must be enforced*[1])
8. have the following scopes:
   a) 'completed' -- which returns all shifts in the system that have at least one job associated with them
   b) 'incomplete' -- which returns all shifts in the system that have do not have at least one job associated with them
   c) 'for_store' -- which returns all shifts that are associated with a given store (parameter: store_id)
   d) 'for_employee' -- which returns all shifts that are associated with a given employee (parameter: employee_id)
   e) 'past' -- which returns all shifts which have a date in the past
   f) 'upcoming' -- which returns all shifts which have a date in the present or future
   g) 'pending' -- which returns all shifts which have a status of pending
   h) 'started' -- which returns all shifts which have a status of started
   i) 'finished' -- which returns all shifts which have a status of finished
   j) 'for_next_days' -- which returns all the upcoming shifts in the next 'x' days (parameter: x)

k) 'for_past_days' -- which returns all the past shifts in the previous 'x' days (parameter: x)
l) 'chronological' -- which orders values chronologically in ascending order based on date and start time
m) 'by_store' -- which orders values by store name
n) 'by_employee' -- which orders values by employee's last, first names

Jobs must:

1. have all proper relationships specified
2. have a name
3. have the following scopes:
   a) 'active' -- returns only active jobs
   b) 'inactive' -- returns all inactive jobs
   c) 'alphabetical' -- orders results alphabetically
4. can only be deleted if the job has never been recorded as being done by an employee during a shift
5. have methods 'make_active' and 'make_inactive' which make the object active or inactive, respectively

Shift Jobs must:

1. have all proper relationships specified
2. have a required connection to both shift and job
3. shift and job must be active in the system
4. have an alphabetical scope based on the name of the job
5. can only be created for shifts that are finished

PayGrades must:

1. have all proper relationships specified
2. have a level (i.e., its name)
3. have the following scopes:
   a) 'active' -- returns only active records
   b) 'inactive' -- returns all inactive records
   c) 'alphabetical' -- orders levels of pay alphabetically
4. can be made inactive but never deleted *(but not automatically inactive if not deleted)*
5. have methods 'make_active' and 'make_inactive' which make the object active or inactive, respectively

PayGradeRates must:

1. have all proper relationships specified
2. have a rate that is a number greater than zero
3. cannot be destroyed
4. have the following scopes:
    a) 'current' -- returns only those rates where end date is not known
    b) 'for date' -- returns all pay grade rates for a particular date (parameter: date)
    c) 'chronological' -- orders results chronologically by start date
    d) 'for_pay_grade' -- returns all pay grade rates for a given pay grade object (parameter: pay_grade)

Stores (in addition to previous requirements) must:

1. have an appropriate relationship with new entities
2. can never be deleted, only made inactive
3. have methods 'make_active' and 'make_inactive' which make the object active or inactive, respectively

Employees (in addition to previous requirements) must:

1. have an appropriate relationship with new entities
2. can only be deleted if the employee has never worked a shift. If the employee can be deleted, their assignment (if it exists) should also be deleted along with all future shifts. If the employee can't be deleted, the employee should be made inactive, their current assignment terminated and all future shifts should be deleted.
3. has a 'role?' method which verifies the role an employee has in the system
4. has a class method called 'authenticate' which takes a username and password and attempt to authenticate this combination as demonstrated in class
5. has a method called 'current_pay_grade' which returns the pay grade level (a string value) for that employee's pay grade at their current assignment, or returns nil if there is no current assignment
6. ~~called 'pay_grade_on' which takes a parameter `date` and returns a pay grade employee's pay grade for their assignment on that date, or returns nil if there is no assignment on that date~~
7. has a method called 'current_pay_rate' which returns the pay rate (a float value) reflecting the employee's pay rate at their current assignment, or returns nil if there is no current assignment

8. ~~called 'pay_rate_on' which takes a parameter `date` and returns a pay rate value reflecting the employee's pay rate for their assignment on that date, or returns nil if there is no assignment on that date~~
9. have methods 'make_active' and 'make_inactive' which make the object active or inactive, respectively
10. must all have usernames and those usernames should be unique (case-insensitive)
11. must have valid passwords that follow rules set forth in class *(encrypted in the database, at least four characters, and the like)*

Assignments (in addition to previous requirements) must:

1. have an appropriate relationship with new entities
2. when assignments are terminated *(end date set to now or in the past)*, any pending shifts associated with the assignment are deleted.
3. may be terminated but never destroyed if there are non-pending shifts that are associated with that assignment. If there are no non-pending shifts, the assignment may be destroyed along with any associated pending shifts.
4. require a pay grade and the associated pay grade must be active in the system
5. have the following additional scopes:
    a) 'for_role' -- returns all assignments for employees of a particular role (parameter: role)
    b) 'for date' -- returns all assignments for a particular date (parameter: date)
    c) 'for_pay_grade' -- returns all assignments for a given pay grade object (parameter: pay_grade)

[1] As a reminder: "One thing that complicates matters is the requirement that for payroll purposes, shift starts need to be rounded down to the nearest 15-minute interval while end times need to be rounded up to the next 15-minute interval. For example, if someone starts at 10:12am, the start time has to be adjusted to 10:00am and if the shift ends at 2:47pm, then the time rounds to 3:00pm. In the database Alex and Mark want to track the actual times the shift starts and ends, but when calculating payroll, they need to round the times to meet requirements. Thankfully, their dad has already created a gem for you called time_date_helpers that will be helpful with the rounding issues and is available at GitHub at https://github.com/profh/time_date_helpers." The README file with this repository has lots of helpful information to get you started as well as a test suite to help you see the gem in action.