CST438 assignment 9

Objectives

- write an HTTP REST service that will create and retrieve orders.
- the order service will depend on the customer and item services from assignment 7
    - lookup customer by email and get customer id and award amount (if any) for customer
    - lookup item by id and verify item is in stock
    - send a copy of the order to both customer and item services so the services can do updates to customer and item records.
- write Rspec unit test for the order service that uses rspec ***doubles*** to avoid actual HTTP calls to the customer and item services when unit testing order service.
- Write a ruby client for the order service – similar to the clients written in assignment 7.
- use a github repository to manage code changes between you and your partner since both of you will be working on the same application

Order model fields

id	Integer key of order
itemId	integer key of item in order
description	item description from item service
customerId	integer key of customer placing the order
price	item price from item service
award	award amount applied to purchase is customer is entitled to award, otherwise 0.00
total	price – award if award is 0, then total is same as price
created_at	
updated_at	

Define a database migration and rails model class for order.

Note about rails authentication token

For security and to deter cross site request attacks, rails places a hidden field during in rendering of HTML pages and authenticates the field in subsequent post requests to verify that this page originated from the Rails server. When designing a REST JSON api for the application, this authentication will cause a problem as the JSON input does not have such a field. Typically, clients will use an appid (as used by the weatherapi) for security and control. In this iteration we will not be using such an appid for our application, but this is something that would have to done in a future iteration and should be listed in the requirements backlog.

For order, customer and item services remove (or comment out) the method call protect_from_forgery in the ApplicationController class. For other ways to solve this problem search ***rails csr authentication token.***

```
class ApplicationController < ActionController::Base
  # protect_from_forgery with: :exception
end
```

CST438 assignment 9

Order service api

HTTP VERB	URI	JSON formatted data
POST	/orders create an order	Http request body contains JSON with itemId and email of customer. Example: { “itemId” : 123456, “email” : “dw@csumb.edu” } http response body contains order fields of order created. http status 201 success http status 400 error http response contains error message
GET	/orders/:id return order data	retrieve order by id http response 200. order fields returned in http response body. http status 404 if not found
GET	/orders?customerId=nnn return array of orders	retrieve orders by customerId http status 200 . response body is JSON array of order objects. array may be empty if customerId is not found.
GET	/orders?email=nn@nnnn return array of orders	retrieve orders by customer email http status 200. response body is JSON array of order objects. array may be empty if email is not found.

Order service implementation

- create an order model and db migration file.
- create orders controller
    - creating an order will require invoking the customer service to get customerId and award amount. And using the item service to get price, description for the item.
- decide how you will render JSON output.
- create routes.rb
- use HTTParty to call the customer and item services.
- The base_url for customer service should be defined as http://localhost:8081
- The base_url for item service should be defined as http://localhost:8082

Order Rspec unit tests

The objective of a unit test is to test the controller and model in isolation. The orders controller is dependent on making calls to the customer and item services. We use Rspec doubles to stub out these calls and return fake data. That way we can run unit tests on orders controller without actually having the customer and item services running. This makes the unit tests fast, independent and repeatable. Remember the FIRST principle of testing?

- use Rspec doubles to stub out http calls.
- test all order apis. Your tests should include both happy and sad tests.

Order client

The order client will be used to do an end to end test. This is different from unit testing because we will be running an actual server (actually 3 servers). One for order, one for customer and one for item services. The end to end test is more complicated than unit testing. Since the code will be making HTTP calls is will not be fast (although in this simple application you won't be able to tell the difference), it won't be repeatable or independent because when a service does an insert or update to the database it will be committed. If you run the same test again it either won't work or will get different results.

When running unit tests, Rspec has a hook into the ActiveRecord and does a rollback at the end of each rspec test so that database changes are not made permanent. This makes each test independent and repeatable. There is no such capability when doing an end to end test because multiple servers are involved.

We could make the order client self-checking; we won't do that in this assignment. Running the order client will be done manually from the command line and the output will have to be visually inspected to verify the correctness.

- similar to assignment 7, write a ruby client that will
    - create a new order
    - retrieve an existing order by orderId, customerId, or customer email
    - register a new customer
    - lookup a customer by id or by email
    - create a new item
    - lookup an item by item id
- client source code should to the same directory as Gemfile.
- You can reuse code from customer_client and item_client in assignment 7.

What to submit for this assignment

enter the URL of your git repo in the iLearn assignment.

We will check your repo for

- **app/controllers/orders_controller.rb**
- **app/models/order.rb**
- **app/client/orders_client.rb**
- **db/migration/xxx_create_order.rb**
- **config/routes.rb**
- **spec/requests/orders_spec.rb**

Running the Order client as an End to end test of order service

running the order client will involve running 3 servers. One for order service, one for the item service and one for the customer server.

The easiest way I have found to do this is to open up 4 terminal windows in Cloud 9.

Terminal window 1 for Item server
 	
$ cd item_app_directory
$ rails server -p 8082
	Terminal window 2 for Orders server
	
	$ cd order_app_directory
	$ rails server -p 8080

Terminal window 3 for Customer server
 	
$ cd customer_app_directory
$ rails server -p 8081
	Terminal window 4 for running Order client
	
	$ cd order_client_directory
	$ ruby order_client.rb

Remember:

1. update the base_uri in all HTTParty calls
   - http://localhost:8080 is for Orders service
   - http://localhost:8081 is for the Customer service
   - http://localhost:8082 is for the Item service

   These base_uri statements appear in the code for Orders app and Order client app.
2. start the 3 servers before running the order client

New to Git? read this section.

You need to have a free github account. Go to http://github.com and register.

Read appendices A6 and A7 in textbook.

Set up your profile with SSH key (highly recommended)

at cloud9

1. **issue command from environment directory**

   ```
   dwisneski:~/environment $ cat ~/.ssh/id_rsa.pub
   ```

   to see if you have an ssh key. Copy the key if it exists.

2. Otherwise issue the command

   ```
   dwisneski:~/environment $ ssh-keygen -t rsa
   ```

   when prompted for file name enter: **`/home/ec2-user/.ssh/id_rsa`**

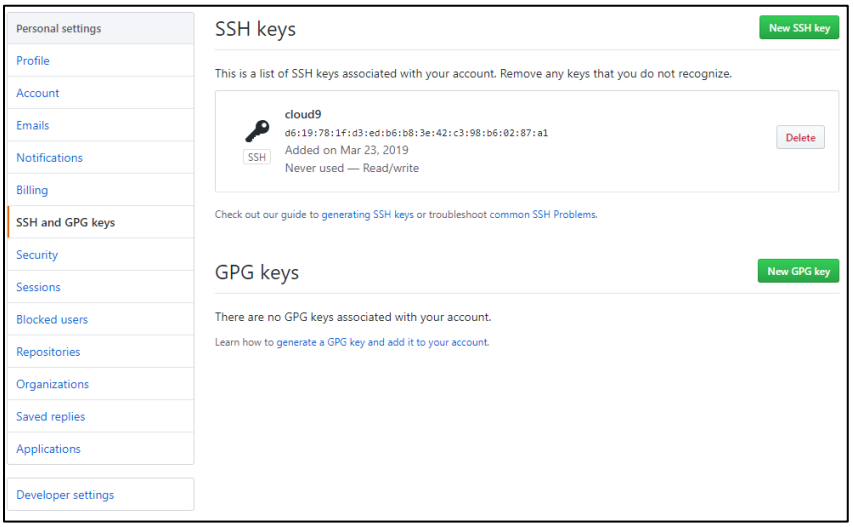
   then do

   ```
   dwisneski:~/environment $ cat ~/.ssh/id_rsa.pub
   ```

   Copy the key.

3. **at github.com and add the public key to your github profile.**

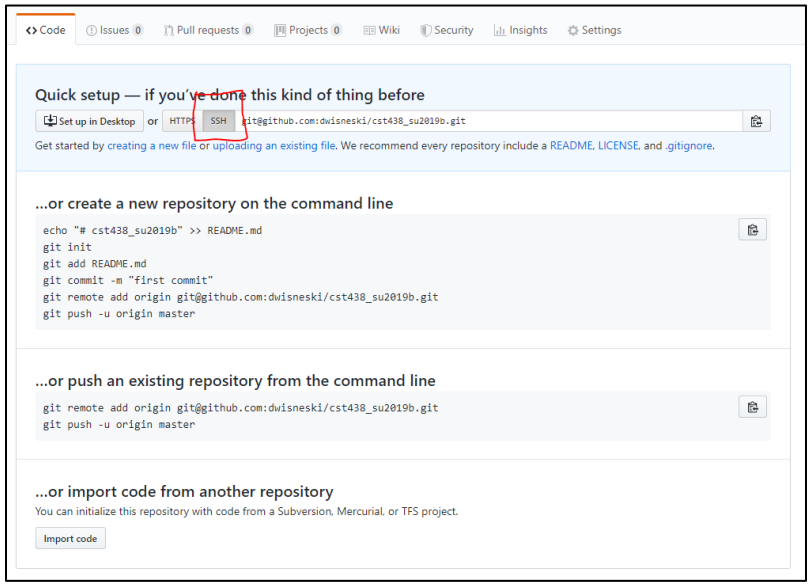
   Edit your profile settings. Select SSH and GPG keys and add New SSH key. Paste the key value.



5. **at github.com Create a new empty repository (only one partner does this)**

   CST438_su2019b_teamNN where NN is your team number.

6. **after creating repository, you will see this page about Quick Setup. Copy the SSH value and share with your partner.**



7. **Add your partner's github userid as a collaborator on the project.**

8. **on Cloud9 create a rails dev project and push it to the git server.**

```
$ rails new assignment9 # create new dev project
$ cd assignment9
$ git init # you may see message that repo has already been initialized
$ git add -A
$ git commit -m "first commit"
$ git remote add origin <ssh value from step 5>
$ git push origin master
```

9. **Your partner will do the following one time to download the project and create a local repo on cloud 9**

```
dwisneski:~/environment $ git clone <ssh value from step 5>
```

10. **Now you and your partner are both using a shared repository with a local copy on each of your dev systems.**

When working on code each developer does

1. **`git pull origin master`**
    a. to merge any recent changes made by your partner into your local repo
    b. You may get merge conflicts that will need to be resolved.
2. make your code changes
3. **`git commit -am "a commit message about reason for change"`**
4. **`git push origin master`**

Git pull downloads code from server repo and merge into local repo. You should do this before you begin making changes so that you are working from the latest version of the code.

The -am option on git commit, will first stage any modified, new or deleted files and then commit the changes to the local repo.

Git push will upload code from local repo to the server.

11. **To configure your name and email used by commit, use the commands**

```
git config global user.name  "My name"
git config global user.email  mname@csumb.edu
```