CST3438 Software Engineering Assignment 7

Objectives

- code a RESTful service using HTTP with JSON
- implement unit tests using Rspec
- code a Ruby client that uses the service

This assignment does not require you to render any HTML views. All input and output will be done using JSON data and REST apis.

My suggestion is do not use the rails generate scaffold command as this will generate too many files that are not needed in this assignment. The steps you are to do (not necessarily in this order)

- create a migration file
- create an active record model class
- create rspec unit tests
- configure the routes
- create the controller class and actions

For controller actions that have response data, the data is returned as JSON. There are two options to do this in rails.

option 1: put the data into a ruby hash. The rails **render** statement will produce a response body from a ruby hash.

option 2: create a **jbuilder** view template. The controller uses a **render** statement to pass data values to the template which creates the response body.

There is more about rendering JSON on page 12.

You and your partner should work as follows:

- one of you implements the customer subsystem
- the other will do the item subsystem
- You will review each other's code in assignment 8.

Background

Colorado River Jewelry (CRJ) is a jewelry store located in the southwestern United States. While CRJ does sell typical jewelry purchased from jewelry vendors, including such items as rings, necklaces, earrings, and watches, it specializes in jewelry from local artists in the southwest. Although some jewelry is manufactured jewelry purchased from vendors in the same manner as the standard jewelry is obtained, many of the artisan jewelry pieces are often unique single items purchased directly from the artisan who created the piece.

CRJ has a small but loyal clientele, and it wants to further increase customer loyalty by creating a frequent buyer award program. In this program, after every 3 purchases, a customer will receive a credit equal to 10 percent of the average of his or her last 3 recent purchases. This credit must be applied to the next (or 4^{th}) purchase.

The programming team consists of a team that will be responsible for development of a mobile app and desktop browser application.

There will be a customer app with actions that allow a customer to

- register themselves,
- view their recent purchases and whether they are entitled to an award, and
- make a purchase.

There will be a manager app that allows a manager to create and update items.

You and your partner are the second team that are responsible for the server back end application.

The lead programmer has decided that the back end application will use SOA (service oriented architecture) and be divided into 3 subsystems: an customer subsystem that will contain customer and recent purchase information, a catalog subsystem that will contains list of jewelry for sale, inventory and vendor information, and a purchase subsystem that will process invoices and purchased items.

This might seem like overkill for a simple system like this, but we will do it as an exercise in writing REST services.

Overall system design

You and your partner will design the model, controller and REST apis for the customer subsystem and the item subsystem. In the next assignment, you will work together on the order subsystem which will interact with the customer and item subsystems using their REST apis.

Think back to assignment 5 where you wrote a ruby program that used the weather service api. This assignment is about implementing a service. In assignment 9 you will write the order service that uses the customer and item services that you write now.

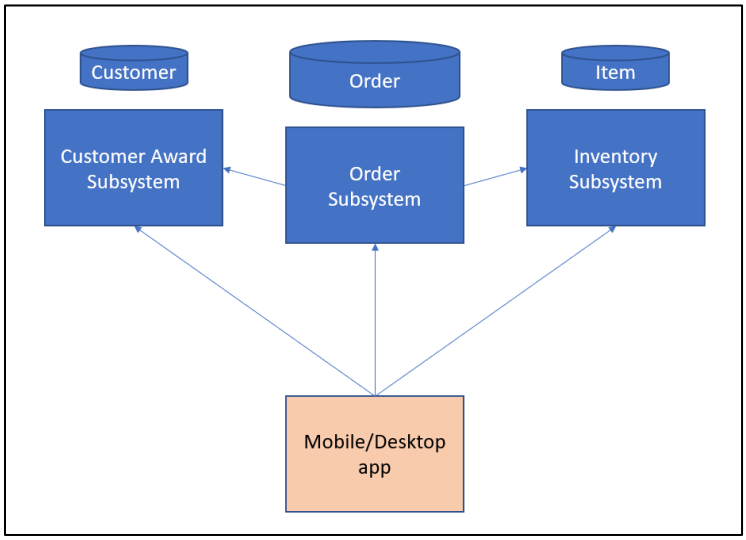

Figure 1.

Agile Philosophy

You can probably think of many requirements for this system. One example is security: customer need to authenticate themselves; we have to know if certain http requests are coming from a trusted system such as the purchase system or coming from an untrusted hacker site.

But we need to build a working prototype in the time of one iteration, so we are only going to handle the essential requirements for the first iteration. Think of this assignment as one iteration in an agile project and not the complete project. You are not going to build the complete system, just enough to demonstrate a working prototype.

Don't anticipate future requirements. Keep it simple and get a working prototype done quickly. For example, you might say instead of an award based on last 3 purchases, maybe it should be more general and allow an award to be based on the last N purchases. But this will make the prototype more complicated and we don't really know if the customer wants or needs this flexibility. So just stick to the essential requirements.

[an aside] one downside of agile, is that since there is no overall requirements document written and no overall design up front, and the project is done as a series of iterative prototypes, it might happen that in later iterations we realize we have to rewrite significant parts of the code because we did not have a design that addressed these requirements. This is called refactoring. And refactoring can impact and slow down future iterations. But a Waterfall process also has its own drawbacks. There is no perfect solution to the problem of developing high quality software. Each team has to decide whether Agile or Waterfall is best for a particular project. You can also do a hybrid; agile process where the first iterations are spent on doing an overall design and basic infrastructure. [end of aside.]

Gemfile updates

Make sure you modify the Gemfile for SQLite3 version and add gem for rspec-rails as you did in assignment 6. You do not need httparty for this assignment.

Disable CSR protection in rails (IMPORTANT! You must make this change)

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

> Rails will generate an authentication token and emit it as a hidden field in rendered HTML forms output. On input it will check for and verify this token to make sure the request data originated from a form page that came from the rails server. This is designed to minimize certain types of hacking attacks on the web site. This mechanism is designed for traditional HTML application and does not apply to application making REST JSON api calls as this data is not associated with a HTML form created by the server. Either comment or delete the statement in application_controller.rb file.

Customer service subsystem

The customer subsystem will need to keep track of customer name, email, and the dollar amounts of the last 3 purchases and to calculate any award the customer is entitled to on the next purchase. Whenever a purchase is made, a copy of the order is sent to the customer subsystem and the customer record is updated.

Customer REST api

HTTP VERB	URI	JSON formatted data
POST	/customers register a new customer	Http request contains customer data with email, lastName, firstName. Any other data is ignored. http status 201 success http response: customer fields id, email, lastName, firstName, lastOrder, lastOrder2, lastOrder3, award http status 400 error http response contains error message
GET	/customers?id=:id retrieve customer data	retrieve customer by primary key http status 200 success http response: customer fields returned as above. http status 404 if not found
GET	/customers?email=:email retrieve customer data	retrieve customer by email http status 200 success http response: customer fields returned as above. http status 404 if not found
PUT	/customers/order update customer	http request body contains order field (see page 5). http status 204 success, no content returned http status 400 error message

To return a http status code with no response body content, use the rails method

```
head :statusCode
Example:    head 204
```

Define a single Customer model class with attributes

- email
- lastName
- firstName
- lastOrder - dollar amount of last order
- lastOrder2 – dollar amount of previous to last order
- lastOrder3 – dollar amount of previous to previous to last order
- award – dollar amount of award if customer has made 3 purchases. Otherwise 0

Define constraints on the customer model to

- validate that email and customer name are not blank
- the email matches the regular expression

  `/\A([^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})\Z/i`

- email is unique.

See https://guides.rubyonrails.org/active_record_validations.html for more information on model validations. Also see Chapter 5.1 in textbook about model validations.

Define controller classes with actions for

- creating a new customer
- retrieving a customer by id or by email
- processing an order and updating customer

Order fields and processing logic

id	Integer key of order
itemId	integer key of item purchased
description	item description
customerId	integer key of customer placing the order
price	item price from catalog
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

An order will contain only 1 item. This may change in future iterations but in this first iteration we keep each order simple with a single item.

The order action of the customer controller must look up the customer record. If the order awardAmount > 0, this means that the award was redeemed. So, the last 3 order amounts and the awardAmount in the customer model are set to zero. Customer must make 3 more purchases before being entitled to next award.

If order awardAmount is 0.0, then update customer using orderAmount and setting either lastOrder, lastOrder2 or lastOrder3. Check to see if customer is now entitled to an award.

Test all the controller actions with Rspec tests

Remember to do command **`$ rails generate rspec:install`** to create the helper files needed by the rspec tests. Create your test files in the spec/requests directory.

You need to test the scenario where a customer makes 3 purchases and gets an award on the 4th purchase. Verify that the award amount is calculated correctly and that the customer record is reset correctly when the 4th order is processed.

Example

- customer makes purchases for $100, $200 and $175
- now customer has award amount of $15.83
- On the next purchase the customer now has $0.00 award.

Write a Customer client

A client is a simple ruby class that uses your customer service. Look back at assignment 5 where you wrote a ruby client that was used the weather service. You will do the same now for your customer service.

But isn't that what I did in the Rspec tests?

Not quite. Rspec does not use a real server. Rspec simulate HTTP calls and Rspec fakes it by just calling the controller class and views without going through the full network + server code path. (makes your tests run faster).

A customer client will use HTTParty to invoke the actions on the CustomerController. Your customer client program will allow a user to

- register a new customer by doing an HTTP post to /customers. user must enter last name, first name and email values.
- retrieve customer data by id or email by doing an HTTP get to /customers

To run your client, you will have to have a rails server running with your customer rails application.

I would suggest that you place the file for customer_client.rb in the same directory as Gemfile, i.e. the top directory of the application project.

The client interaction might look like this (but this is only a suggestion, you can design your own client interaction). Red shows what the user types. The customer_client program allows user to choose from registering a new customer or retrieving a customer by email or id.

```
$ ruby customer_client.rb
What do you want to do: register, email, id or quit
register
enter lastName, firstName and email for new customer
Boyce Codd bc@csumb.edu
status code 201
{"id":9,"email":"bc@csumb.edu","lastName":"Boyce","firstName":"Codd","
award":0,"lastOrder":0,"lastOrder2":0,"lastOrder3":0}

What do you want to do: register, email, id or quit
email
enter email
bc@csumb.edu
status code 200
{"id":9,"email":"bc@csumb.edu","lastName":"Boyce","firstName":"Codd","
award":0,"lastOrder":0.0,"lastOrder2":0.0,"lastOrder3":0.0}

What do you want to do: register, email, id or quit
id
enter id
9
status code 200
{"id":9,"email":"bc@csumb.edu","lastName":"Boyce","firstName":"Codd","
award":0,"lastOrder":0.0,"lastOrder2":0.0,"lastOrder3":0.0}

What do you want to do: register, email, id or quit
quit
```

Client ruby code doing HTTP requests using HTTParty

- in assignment 5 and 6 and you did get requests
- to issue other HTTP requests which require a request body.

```
base_uri 'http://localhost'

response = post '/uri',
      body: obj.to_json,
      headers:  { 'Content-Type' => 'application/json',
                  'ACCEPT' => 'application/json' }
```

Define base_uri as localhost. The rails server is running on your local machine (or Cloud 9 dev machine).

obj is a ruby hash, array or some other object that supports to_json method.

response will have methods

- response.code HTTP status code such as 200, 201, 404, etc.
- response.body

Item subsystem

The item subsystem will need to store information on each item for sale including item number, description and price and quantity in stock.

HTTP VERB	URI	JSON formatted data
POST	/items create a new item used by manager app.	Http request body contains item fields description, price and stockQty http response body contains all item fields http status 201 success http status 400 error http response contains error message
GET	/items/:id return item data	retrieve item by id http response 200. Item field returned in response body. http status 404 if not found
PUT	/items update item.	Http request body contains item fields to be updated. http status 204 success, no content returned. http status 404 if item id not found
PUT	/items/order update item	Http request contains order fields (see page 10) http status 204 success no content returned. Item stockQty is decremented. http status 404 if item id not found http status 400 if other error

Item model fields

id	Integer key of item
description	item description
price	item price from catalog
stockQty	quantity in stock

Define constraints on the item model for

- price, description, stockQty are not blank.
- price must be greater than 0
- stockQty cannot be negative and must be an integer

See https://guides.rubyonrails.org/active_record_validations.html for more information on model validations. Also see Chapter 5.1 in textbook about model validations.

Define controller classes with actions for

- create a new item
- retrieving an item by key
- update an item
- process and order

Order fields and order processing logic

id	Integer key of order
itemId	integer key of item purchased
description	item description
customerId	integer key of customer placing the order
price	item price from catalog
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

An order will contain only 1 item. This may change in future iterations but in this first iteration we keep each order simple with a single item.

The order action of the item controller must look up the item model. If stockQty is 0 item is not in stock and should return error.

Otherwise, decrement the stockQty.

Test all the controller actions with Rspec tests

Remember to do command **`$ rails generate rspec:install`** to create the helper files needed by the rspec tests. Create your test files in the spec/requests directory.

Test both good and bad cases. Item id not found. Item is not in stock (stockQty is 0 at time of purchase). Make sure that stockQty is being decremented on a successful order.

Write an Item client

Read the section about Customer client. The item client is similar but has actions for

- create a new item
- retrieve and show an item by id
- update an item

The user interaction might be as follows (but you are free to design your own interface)

```
$ ruby item_client.rb
What do you want to do: create, update, get or quit
create

enter item description
diamond ring
enter item price
1500.00
enter item stockQty
3
status code 201
{"id":8,"description":"diamond ring","price":1500.0,"stockQty":3}

What do you want to do: create, update, get or quit
update

enter id of item to update
8
enter  description
diamond ring
enter  price
1650.00
enter  stockQty
3
status code 204


What do you want to do: create, update, get or quit
get

enter id of item to lookup
8
status code 200
{"id":8,"description":"diamond ring","price":1650.0,"stockQty":3}

What do you want to do: create, update, get or quit
quit
```

More about JSON data

JSON data in http request body – used in POST or PUT call.

This is easy. All data is in the params hash used in the controller. Rails does all the work for you and puts all the key, value pairs from the HTTP request body as well as URI symbols and query parameters from the URI into the params hash. This is the same action that happens for the case of using HTML form input. So, there is no special code you have to write in the controller.

To return data in an http response body in JSON, there are two options.

Option1. Controller uses rails render method with json option to return json data.

The controller put the data to return in a hash or an array and then calls the render method as follows.

```
data = {:id=> 12, :description=>"book", :price=>123.50 }
render(json: data , status: 200 )
```

The render method will convert the hash to JSON format and put into the HTTP response body.

The :status key word is used to set http status code in the header. This option does use a view to render the output.

The http response will have a status code of 200 and a body containing the string value

```
{ "id": 12,  "description": "book", "price": 123.50 }
```

Option 2. controller uses a JBuilder view to create JSON response

The controller will put data to be rendered in one or more instance variables. The Jbuilder view will then extract data from these variables. It is important that the data be an instance variable. Local variables are not passed to the view.

This option uses a view file. The file name is related to the controller and action name. For example, if the ItemsController uses a view to return data using the "abc" view the file is `app/views/items/abc.json.jbuilder`

In the controller use the statement

```
render "abc.json.jbuilder"
```

The jbuilder file contains

```
json.extract!(@item,  :id,  :description, :price)
```

where @item is an instance variable set in the controller. The values for id, description and price are extracted from the @item object.

If you need to extract data from an array of objects, use json.array! method. Here @items is an instance variable containing the array.

```
json.array!(@items) do |item|
  json.extract!(item,  :id,  :description, :price)
end
```

Note that **`json.( . . . )`** is equivalent to **`json.extract!( . . . )`**

For other examples of what can be done in a jbuilder view see

https://github.com/rails/jbuilder

Also see https://guides.rubyonrails.org/layouts_and_rendering.html section 2.2.8 about rendering JSON in rails.

What to submit for this assignment?

A zip file containing the 4 files

1. customer_controller.rb or item_controller.rb file
    a. Your controller should have methods for each of the REST apis of the subsystem.
2. spec file containing Rspec tests
    a. There should be happy and sad tests for each of the REST apis.
3. client ruby program file that has functionality
    a. for customer: register , retrieve by email or id
    b. for item, create new item, update item, retrieve by id.
4. A PDF file containing
    a. a screen shot showing Rspec tests passing
    b. a screen shot of a client program in execution (similar to output shown on page 7 and 11

Only include the 4 files mentioned above.