Resources / Assignments (/COMP9321/18s1/resources/15672) / Week 3 (/COMP9321/18s1/resources/15673) / Assignment 1: Building RESTful APIs

Assignment 1: Building RESTful APIs

Deadline: Week 5 Thursday (29th March) 23:59.

Total 10 marks allocated.

Individual assignment.

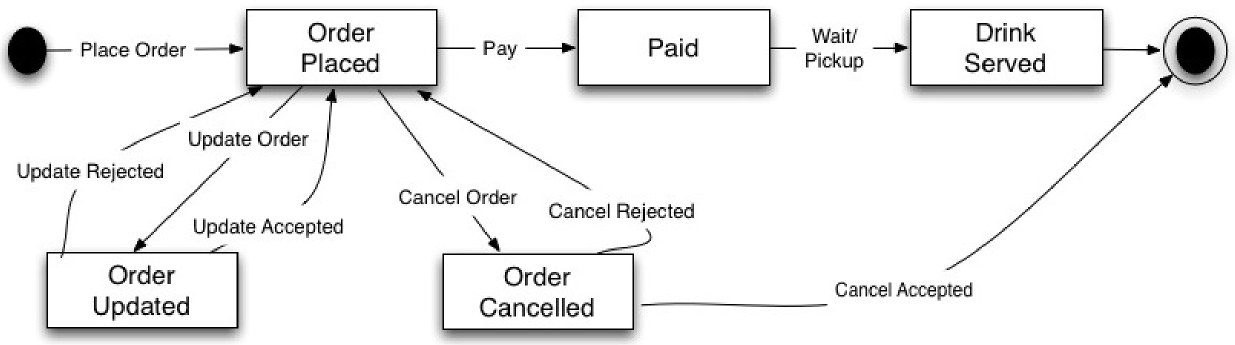
Late penalty: 20 percent of the final mark per day

Business Scenario

In this assignment, you are to implement a set of REST-based APIs that effectively manage a coffee order processing logic. The scenario is based on the article published by Jim Webber and others in How to GET a Cup of Coffee (http://www.infoq.com/articles/webber-rest-workflow) . ***DO* read** the article through as it will help you understand the tasks required in the assignment better. However, this assignment specification defines a slightly different set of requirements, so the article should only be considered as background reading - **not as the specification** . An important concept to grasp in the article is relating resources through link/next elements to implement the coffee order workflow (i.e., the next possible steps).

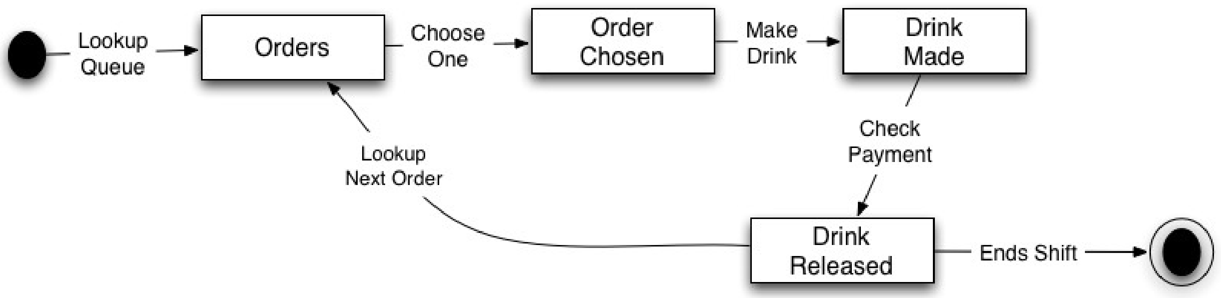
First, let us imagine that we have a coffee shop whose operations are based on the following two workflows (modelled as state machines).

Cashier/customer's Workflow:



Imagine a cashier at the coffee shop interacting with a customer. In the workflow above, the process depicts a scenario where the cashier creates an order on behalf of the customer, takes payment for the order, and then the customer waits to pickup the coffee when it is ready. Between placing and paying for the order, the cashier (on behalf of the customer) can amend it (for example, asking for skimmed milk to be used, or adding an extra shot). It is also possible to cancel the order completely during the period (i.e., between placing and paying for the order).

Barista's workflow:

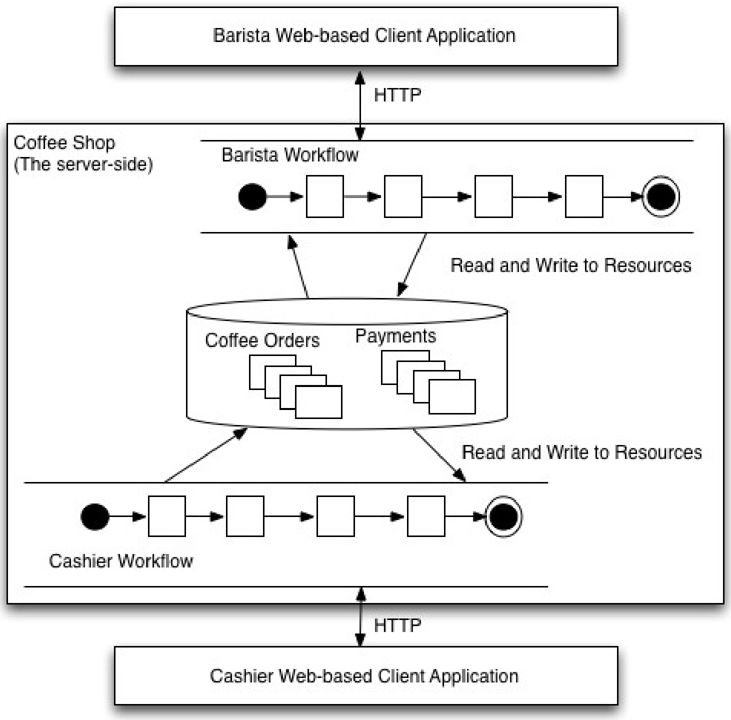


The barista has his or her own workflow. This workflow is not visible to the customer or the cashier (i.e., imagine a dividing screen between the coffee shop reception area and the coffee making area). Each barista works in shifts. When he or she checks in to work, a screen is displayed with current outstanding orders. The barista looks for the next order to be made, picks up an order, prepares the drink, and releases the drink to the reception area.

Our baristas are very cautious people, so they don't want to release the drinks if the payment hasn't been made. Before releasing each drink, the barista checks if payment is made for the order, and only then the drink is released. If not - the order's status will not move forward.

API Implementation

System Overview


Barista Web-based Client Application
HTTP
Coffee Shop (The server-side)
Barista Workflow
Read and Write to Resources
Coffee Orders
Payments
Read and Write to Resources
Cashier Workflow
HTTP
Cashier Web-based Client Application

The diagram above shows how the server-side system and their two client applications are structured. The picture aims to show how the clients might interact with the resources, to give you more context about the APIs you are building.

It is important to note that you are not building the Barista or Cashier applications for this assignment. There is no client-side implementation involved here at all. You will be designing and implementing the RESTful APIs that expose Coffee Order and Payment as resources. Of course, the APIs will eventually be used by the client applications, but that is out of scope. The two workflows introduced earlier explain how the clients will interact with your APIs.

To summarize the **main functionalities** indicated by the workflows:

增 • Creating Order
查 • Getting Order Details
改 • Amending Order (before payment and barista's status update)
删 • Cancelling Order (before payment)
增 • Creating a Payment
查 • Getting Payment Details for an Order

The following describes the API you should implement.

Resources and Their URI Patterns:

There are two types of resources to be managed: Coffee Orders and Payments for the orders. The article mentioned earlier (How to GET a Cup of Coffee (http://www.infoq.com/articles/webber-rest-workflow)) shows some examples of how these resources might be represented. But it is up to you to design these resources and their URI patterns. Indeed, this will be your first task. Go through the scenario and make sure that you have every piece of data needed to service the scenario is covered.

Some suggested information for each resource:

- **Order** : type of coffee, cost, additions (e.g., skim milk, extra shot)
- **Payment** : payment type (cash or card), payment amount, card details (if card)

Please be noted that:

1. Things like 'additions' are obviously optional data. So some attributes are 'required', some are 'optional'
2. You may need a 'status flag' to manage the current status of an order (or payment as well?). What status should you have to service the scenario effectively?
3. Card validation is not required, but you may want to do some minimal checks on the values, e.g., all digits?. This idea actually applies to all data - should you trust what is being given to you (i.e., the API) or should you be checking the input diligently?

Managing Resources:

It is important that all HTTP methods implemented by you satisfy the safety and idempotency properties of REST operations. It is also important that the design of the API allow stateless communications. It is part of your design task to choose an appropriate HTTP method to implement the following management operations.

On orders:

- An order is created. The new order ID and the total cost of the order (i.e., the amount to be paid for the order) should be included in the response to this operation. Note that the total cost is always given from the server whenever an order resource is created or updated.
- The detail of an order can be retrieved anytime. The response of this operation must contain the latest representation of the order.
- The detail of multiple orders can be retreived anytime by status. The response of this must contain the list of all orders matching the status given.
- An order is updated (when possible). Note that both Barista and Cashier can update, but different parts of the resource will be affected. An update could be implemented by PUT or PATCH depending on the update needs. The response of HTTP PUT must contain the updated representation of the resource (e.g., new total cost). The response of PATCH is not defined - i.e., you may return an appropriate status code, but not necessarily any body content.
- An order is cancelled (when possible). The response of this operation may return an appropriate status code, but not necessarily any body content. Note: cancelling an order is different from completing an order.

An important note on doing a PUT on orders; You should make sure that you are doing proper checking before you allow a PUT operation (e.g., can this order be updated now?). The same idea applies to PATCH as well. You probably will be using PATCH to update the status of an order. You should be doing proper checks for a PATCH to be allowed (e.g., can this order status be set to 'released'?).

On payments:

- a payment is `created'. Similar to the scenario in the article `How to get a cup of coffee', the location of a new payment resource is given from the API to the client. This means the client only updates the payment with the given URI to create it, so to speak. This also means, of course, issuing this operation multiple times into the same URI does not result in multiple payments.
- The detail of a payment can be retrieved anytime. The response of this operation must contain the latest representation of the payment.
- In the given workflow, we do not encounter a scenario where payment is updated or deleted.

Linking resources:

- So, when is the client given the URI of a payment? Probably the most appropriate moment is when the order is created. For example, the response of HTTP POST on order should contain a link to new payment resource so that the cashier can take the payment. Here, the idea is to reveal a relationship/link between resources to the client as it becomes available.

Persisting resources:

Ideally, you would like to store these data in a persisting data source like a database. In this assignment, you are allowed to implement the solution with **in-memory data structures** (any of your favorite Python data structure that will suite the purpose would do.). **Do NOT attempt to provide a database-backed solution here** . We do not expect it and it may create a situation that your solution doesn't run for marking.

Authorisation and Authentication:

In this first assignment, you do not have to implement the API security aspect of the scenario. Obviously, There are certain operations that require checking of basic security properties (e.g., only Barista should be allowed to update the status of the order to 'released', or only Cashier can create an order). We will ignore the security aspect and just focus on the API's core operational functions.

RESTful APIs are stateless:

Your implementation of APIs should show that your APIs are stateless (i.e., you do not store any specific information about a particular client) and all communications are self-contained with the messages.

API documentation:

You should document all of your endpoints and their sample input and output. This is important for the tutors to mark your assignment correctly.

The documentation should be written in a single HTML file and will be submitted along with your source code.

Submission and Marking scheme

The marking environment setup is basically the same as the lab setup. We will use PyCharm to import and run your solution. You must make sure that it runs correctly that way before making a submission. We will not be tweaking any of the configuration. Your PyCharm project should simply run after import (make sure to use virtual environment). More details to be released in due course.

Submission is through WebCMS3. A detailed instruction and marking guide will be given soon.

Resource created 3 days ago (Monday 12 March 2018, 08:50:12 AM), last modified about an hour ago (Thursday 15 March 2018, 01:44:13 PM).

Comments

(/COMP9321/18s1/forums/search?forum_choice=resource/15676)

(/COMP9321/18s1/forums/resource/15676)

Add a comment

There are no comments yet.