Tasks and Assignments Course 02267

Hubert Baumeister

Version 1

1 Task (group): Simple DTU Pay with a REST APItask_REST_DTU_Pay

The task is to implement a REST backend of a simplified payment system called "Simple DTU Pay" with the corresponding client that tests that backend. With Simple DTU Pay it is possible to create payments between customers and merchants by providing their respective id's and an amount. Furthermore, it should be possible to ask the system for a list of payments.

A possible way to express the payment scenario as a Cucumber scenario is:

```
Feature: Payment
Scenario: Successful Payment
  Given a customer with name "Susan"
  Given the customer is registered with Simple DTU Pay
  Given a merchant with name "Daniel"
  Given the merchant is registered with Simple DTU Pay
  When the merchant initiates a payment for 10 kr by the customer
  Then the payment is successful
```

Here is an example of the associated step definitions.

```
public class SimpleDTUPaySteps {
   private Customer customer;
   private Merchant merchant;
   private String customerId, merchantId;
   private SimpleDtuPay dtupay = new SimpleDtuPay();
   private boolean successful = false;

   @Given("a customer with name {string}")
   public void aCustomerWithName(String name) {
      customer = new Customer(name);
   }

   @Given("the customer is registered with Simple DTU Pay")
   public void theCustomerIsRegisteredWithSimpleDTUPay() {
      customerId = dtupay.register(customer);
   }

   @Given("a merchant with name {string}")
   public void aMerchantWithName(String name) {
      merchant = new Merchant(name);
   }

   @Given("the merchant is registered with Simple DTU Pay")
   public void theMerchantIsRegisteredWithSimpleDTUPay() {
      merchantId = dtupay.register(merchant);
   }


   @When("the merchant initiates a payment for {int} kr by the customer")
   public void theMerchantInitiatesAPaymentForKrByTheCustomer(Integer amount) {
      successful = dtupay.pay(amount,customerId,merchantId);
   }

   @Then("the payment is successful")
   public void thePaymentIsSuccessful() {
       assertTrue(successful);
   }
}
```

Note that the `register` and the `pay` method in class `SimpleDTUPay` uses, e.g., the JAX-RS framework to call the SimpleDTUPay REST API on the server. Hiding the REST calls in their own class helps to keep the step definitions simple and also avoids repetitions, if the same method is called several times in the tests.

```
Scenario: List of payments
  Given a customer with name "Susan", who is registered with Simple DTU Pay
  And a merchant with name "Daniel", who is registered with Simple DTU Pay
  Given a successful payment of "10" kr from the customer to the merchant
  When the manager asks for a list of payments
  Then the list contains a payments where customer "Susan" paid "10" kr to merchant "Daniel"
```

Before initiating payment, the customer and the merchant have to be registered with SimpleDTUPay. Using other customer- and merchant-id's other then for registered customers and merchants should result in a failure.

```
Scenario: Customer is not known
  Given a merchant with name "Daniel", who is registered with Simple DTU Pay
  When the merchant initiates a payment for "10" kr using customer id "non-existent-id"
  Then the payment is not successful
  And an error message is returned saying "customer with id \"non-existent-id\" is unknown"
```

There should be a similar scenario for an unknown merchant.

Also, for testing purpose, you should implement a `unregister(..)` function, taking a customer-id / merchant-id, to be able to register the same customer/merchant again in the next scenario. Note that you can use the `io.cucumber.java.After` annotation to unregister the accounts after each execution of a scenario.

Your task is to design the proper REST interface. What are the resources, e.g. merchants, customers, payments, . . . ? How can these resources be manipulated, e.g. registering (creating) merchants and customers, unregistering (deleting) merchants and customers, creating a new payment, listing all payments, and how are those realized using HTTP Verbs.

You should define the needed resources, their URI (note, these should not contain any verbs, e.g. `pay`), the HTTP verb and which function they trigger.

Have a look at the design of the REST interface for the StudentRegistration as an example.

1.1 Tips

No database is required. You can store payments in a list in a field.

You can build Simple DTU Pay based on a copy of the code from the previous task. Make sure though that you rename the projects accordingly in the `pom.xml` files and remove parts that are not used anymore.

It is important to only change small things and make sure that the resulting system is working. You notice that your steps are too big, when your application stops working and it is not immediatly clear what caused the application to malfunction. In this case, go back to the last working version and use smaller steps.