Divolte Workshop: Exercise 3

In the first exercise we updated the base template that is used to render all pages in the application. In this exercise we're going to update a few more templates in key places to ensure that additional events are logged.

Step 1: Generating *Preview* Events

First we're going to add a trigger so that an event is logged when a user previews a photo in the application. This could be used as a signal of interest, for example. Open up and edit: `webapp/templates/product.html`

In this file is a JavaScript function around line 23 that is executed whenever a user previews a photo. Update this function so that it reads:

```
function itemPreviewed(id) {
   divolte.signal('preview', { item_id: id })
}
```

Once you've added this, restart the web application (`./refresh`) and then preview some photos.

After waiting a minute, check the Avro data (`docker-compose run divolte show-avro`): you should see the new events turning up. One thing you might notice is that the `productId` field is empty. This is something we'll fix shortly.

Step 2: *Add to Basket* Events

We're going to continue modifying the templates. Open up and edit: `webapp/templates/add-to-basket.html`

At the end of this file is a JavaScript function called `addToBasket()` which implements this functionality. Add a new line at the *start* of this function:

```
divolte.signal('addToBasket', { item_id: id })
```

If you wish you can restart the stack and examine the data, but let's move on to handling our last event.

Step 3: *Remove from Basket* Events

To add our final event open up and edit: `webapp/templates/basket.html`

Near the top of this file is a JavaScript function called `trashItem()` which implements this functionality. Add a new line at the *start* of this function:

```
divolte.signal('removeFromBasket', { item_id: id })
```

Now's a good time to restart the stack (`./refresh`) and check that the events are being triggered. As we noted earlier, however, there's a problem: the events are being triggered but the `productId` field is empty. Let's fix that.

Step 4: Updating the Mapping

We're going to update the mapping one last time. Open up and edit: `divolte/mapping.groovy`

Look for the `EXERCISE 3: INSERT SECTION HERE` comment and add in the following:

```
section {
    when eventType().equalTo('removeFromBasket') apply {
        map eventParameters().value('item_id') onto 'productId'
        exit()
    }

    when eventType().equalTo('addToBasket') apply {
        map eventParameters().value('item_id') onto 'productId'
        exit()
    }

    when eventType().equalTo('preview') apply {
        map eventParameters().value('item_id') onto 'productId'
        exit()
    }
}
```

Restart the stack (`./refresh`) and ensure that Divolte started up properly using: `docker-compose logs -f divolte`

Browse around the site http://localhost:9011/, making sure you:

- Preview some photos from the category pages.
- Add some things to the basket.
- Remove some things from the basket.

Finally, wait a minute for the events to flush and browse the Avro data (`docker-compose run divolte show-avro`). You should now see the additional events, as well as the extra fields being filled in.

Things to Think About

- If you look carefully at the `trashItem()` implementation you'll see there's also a call to a function named

`divolte.whenCommitted()`. What do you think this does?

- What would happen if the `divolte.signal()` call was placed at the *bottom* of the `trashitem()` implementation instead of at the start?
- What other custom events would be interesting to log?