Milestone 4

due at midnight on Sun Apr 3 (125 points)

For this milestone, we will implement a server-side *shopping cart* tied to a user account. The specific tasks are below. As usual, refer to my cs164pub project to examine the code that I wrote in class.

1. Reorganize your sparkdemo project (or start a new project directory if you prefer) so that:

- Within the `src` directory, there is a `util` package containing the classes that are generic functionality — perhaps specific to the spark platform or H2 database, but not tied to our e-commerce project. For example, I have:
  - `DatabaseConfig` which specifies the JDBC URL and also provides methods to connect, migrate, and process result sets.
  - `HandlebarsTemplateEngine` is a *copy* of the original class in the `spark-template-handlebars` package, but with the `handlebars` field made protected instead of private. We use this so that we can specify "template helpers" in a subclass.
  - `H2ConsoleWrapper` is completely optional, but I use it for launching the H2 database console so that it is automatically connected to the JDBC URL specified in `DatabaseConfig`.
  - `MiscTemplateHelpers` is also optional; it was just a demonstration of how to define a template helper `{{now}}` to insert the current timestamp.
- Also within the `src` directory, there is an `ecommerce` package containing the classes directly related to our project. You will mainly work here, and probably add new classes here. At this stage, I have:
  - `ProductDemo` which defines a spark server in its `main` method. This is the one that can list all the products, show details about a particular product, and contains an "Add to Cart" button.
  - `AugmentedTemplateEngine` is a subclass of our customized `HandlebarsTemplateEngine` that registers template helpers that are useful to our project.
  - `Paths` is a class for defining the URL paths and corresponding template helpers. It helps us keep the paths consistent. For now, the paths should include `/` for the home page (product list), `/product/:id` for a product detail page, and `/cart` for the shopping cart page.
- Make sure you have created a `resources/templates` directory where your HTML template files will be found by the spark engine.
- Make sure you have created the `resources/db/migration` directory where SQL migration scripts will be found by Flyway.
- My SQL migration scripts create tables for `user` and `product` with some sample data in each.

2. Add an SQL migration script to define the two tables we need to represent users' shopping carts. We think of this as a many-to-many relationship between the order and the products. (Each order can contain many products, and each product can be part of many orders.) As usual, the way that we represent many-to-many is to break it out as a separate table.

- `order` has an owner which references a user, and a status. Let's use a character or a short string for the status. For now there are probably just two statuses: `cart` means the order represents a shopping cart, and `purchased` means the user has already checked out and the order is in progress.
- `ordered_item` is the table which represents the many-to-many relationship. It has a reference to an order ID and a product ID, and then an integer `quantity` field.

3. Add another SQL migration script to insert a few sample products into your sample user's cart.

4. Make the `/cart` page display the contents of the current user's cart. Until we connect up the login/logout stuff, we don't really know who the current user is. For now, you could hard-code the ID of your sample user with something like:

```java
private static final int TEMPORARY_CURRENT_USER_ID = 1;
```

   Then after we connect up the login facility that sets a user ID in the session, we can search for occurrences of that temporary constant and replace them with the ID retrieved from the session.

5. Make the "Add to cart" button add an entry to the current user's cart before redirecting to the `/cart` page to show it. There are a few things that could happen here. If the current user doesn't have an `order` entry where status is `cart`, we need to create a new one. If current user *does* have a cart already, we then have to check whether this product is already in the cart. If it is, we just increment the quantity. If not, then we have to insert it.

Commit and push to the Git server as often as you like — it's a good way to keep backups of your work. When you have a commit candidate that you think is your final submission, **please include `#milestone4` in the first line of the commit message** — I will search for that when figuring out what to grade.