Homework 2

Due: Friday April 26, end of day

Submission instructions:

- Go to the admin console of web2py (URL: /admin), choose your application, and click on "Pack All".
    - Do *not* select other options such as "compile".
    - Please check and recheck that you are clicking on Pack All for your homework solution, and not for some other application.
- Save the resulting .w2p file.
- Submit the .w2p file to this CrowdGrader assignment: https://peer.crowdgrader.com/crowdgrader/venues/view_venue/4300

Grading instructions:

- You will download submissions in the .w2p format.
- Again, go to the /admin console of web2py.
- Select "Upload and install packed application" on the right, give it an arbitrary name *without dashes in it,* such as review1, and select the .w2p file for upload.
- Click on Install.
- The application (e.g., review1) will appear on the left. Click on it to open it.
- Grade the application following the provided grading rubric.

The Assignment

You have to construct a site for keeping track of products that are in stock. The site has two main pages:

- A "home" page that lists products that have been "starred".
- A "list" page where you can see and search the whole list of products, increment or decrement their quantity, and edit the products.

Products

Each product has the following attributes:

- Name
- Description
- Quantity in stock
- Quantity sold
- Sales Price (to customer)
- Cost (to us the store owners)
- Starred (boolean)
- User who created it

The List Page

The list page must be generated with an SQLFORM.grid, and for each product (and so, for each row) it should display:

- Name
- Quantity in stock
- Quantity sold
- Sales Price
- Cost
- Profit, computed as the product of quantity_sold * (sales_price - cost)
- A star. For the star icon, you should use Font Awesome 4.7 (4.6.3 also fine). The master branch includes 4.6.3 already.
    - The star is black and empty if the product is “unstarred” (false), and it is full and golden if the product is “starred” (true). Note that Font Awesome provides both full and outline only stars.
    - **Clicking** on the icon **if the user is logged in** should toggle the status (use a signed URL to cause the effect, similarly to how immediate delete was implemented). Any logged in user can star/unstar a product. Clicking while not logged in has no effect (i.e., the star is not a link in that case).
- **If the user is logged in,** there should be two additional buttons. These buttons need to be implemented as buttons with signed URLs, that have effect immediately.
    - A +1 button, which increments the quantity (in stock) by one.
    - A -1 button, which decrements the quantity (in stock) by one, and increments by one the number of products sold. The button should be enabled only if the quantity in stock is positive; otherwise it should be grayed out.
- **If the user is logged in and is the user who created the product,** there should also be two additional buttons:
    - **A button to edit the item** (you can use a button with a pencil icon). This leads to a separate product edit page, where the user can edit the product. When the edit is done, you should redirect back to the listing page.
    - **A button to delete the item** (you can use the trash icon). This acts immediately, via a signed URL.

Note: We are unfortunately not working in javascript yet. So when you click on a star, or on a [+1] button, etc, you have to implement this as following a link, that is, a GET request with a signed URL. The receiving controller has to take the action, updating the database, and then redirects back to the list page. This causes an ugly reloading of the page whenever one clicks on star, +1, -1, …. This is ugly, but there is no way out right now. In a model-view-controller framework, unless we use javascript, the only way to display a change is to display a new page. It’s even uglier because, if you are on page 2 of a long list, clicking on (say) star would cause the list to reload showing page 1, unless you implement some way of keeping track of the state,

which I do not require. Please put up with it, and we will have cleaner implementations once we move to javascript.

Sample line, with abbreviated names for column headers, and not including icons for Edit and Delete buttons:

Name	Stock	Sold	Price	Cost	Profit					
Yellow Socks	3	120	$20	$9	$1,320	☆	[+1]	[-1]	[Edit]	[Delete]

Creating Products. The list page should also let users create new products. To do so, you can either use the create feature of SQLFORM.grid, or you can provide a separate product creation controller and page. When creating a product, these (and only these) are the fields that can be filled in:

- Name
- Description
- Quantity in Stock
- Sales Price
- Cost

Creating and editing products: validation
When a product is created or edited, you must check that:

- Sales price >= 0
- Cost >= 0
- Quantity in stock >= 0

Please have a look at the IS_FLOAT_IN_RANGE and IS_INT_IN_RANGE validators. You can use 1e10 as upper bound for these.

The Home Page

The home page should display, in rectangles, all the starred products. For each product there should be the product name and the sales price. That's it.

Suggested Implementation Strategy

Implementation Steps

To implement the solution, I suggest that you follow the steps in the order below. Note: I have not implemented the solution, so this is my best guess.

1. Start from the master branch of the class code; this gives you a clean slate to start the implementation. The branch already comes with Font Awesome and stupid.css.
2. Define the database table for the products.
3. Create the SQLFORM.grid page, initially focusing on the attributes you can display directly from the database: sales price, cost, quantity in stock, name, …
4. Add some way to create new products (a new add controller may be the simplest). If you use a separate controller, set create=False in the SQLFORM.grid.
5. Then, add the button to star/unstar.
6. Add the buttons to +1 and -1 the quantity.
7. Add to the grid a field
8. Implement the button to edit a product.
9. Implement the button to delete a product.
10. Implement the display of the profit.
11. Implement the home page. The simplest way is to create a fixed-width div that is used to display a product, and just generate a bunch of these divs side by side, so that the browser will change line when it wants. All I am looking here is for you to be able to retrieve from the db all the starred products, and display them. We will return to this customer facing page better in the next homework.

To implement the button to star/unstar, here is a hint. Look at the implementation of the rightback method in Lecture 6. Then:

1. In a links field, create a star that is either outline or full according to the star status of the record.
2. Have the star link (just like the rightback button) to a toggle_star controller.
3. The toggle_star controller reads the product, toggles the state of the star in the db, and returns with a redirect to the page that implements the SQLFORM.grid.

Database Access

Here are some methods for working with the database that may help you.

To add 1 to the quantity in stock of an item with id k, you can do as follows (see this):

```
p = db.product(k)
if p is None:
```

```
    redirect(...)
p.quantity = p.quantity + 1 # This makes the change in memory.
p.update_record() # This writes the change to the db.
```

What you have to do in the toggle_star controller is quite similar.