CS387. Take-home assignment Feb-4

This assignment is to be done in pairs. Submit the two exercise as two separate zip files of the form rollnumber1-rollnumber2-ex1-feb4.zip, and rollnumber1-rollnumber2-ex2-feb4.zip

Due: Feb 8, 11:59PM.

Exercise 1. The Badlu Online store in Django and Postgres

A database backed order-processing system for Badlu's chai adda. Assume there are only three products for now: idli, chai and samosa.

The user interaction is as follows:

User:	Open localhost:8000/login method type: 'GET'
App:	Returns an empty login form with username and password
User:	Fills form and submits it. URL: localhost:8000/login, method type 'POST'. (Use http, not https)
App:	If user in database and validated correctly, return a redirect to localhost:8000/orders See: https://www.tutorialspoint.com/django/django_page_redirection.htm Browser goes to localhost:8000/orders with a 'GET' method App returns an order-view-create page, which contains the last 5 orders (at most), one on each line, along with a form that has fields for the quantities to be ordered of each item.
User:	Fills form and hits submit. URL: localhost:8000/orders, method_type: 'POST'. The parameters will automatically be encoded in the body of the request as 'samosa=10&idli=0&chai=10'.
App:	Inserts order into order history in the database, and returns the order-view-create page, with the order history updated, and the form fields emptied. A status line at the bottom should be in green to convey acknowledgment, or in red to convey error.

Now for some implementation details.

1. The django app should be called orders
2. The database contains users and order history. You can pre-populate the list of users
3. At this point we don't want to use Django's built-in users table. We'll just create our own. Create table addaUsers with (loginName (primary key), password) in postgres

Create table orderItem (orderId (fk), item, itemQuantity)
Create table public.order (orderId(pk), loginName, dateTime)
I used the fully qualified name public.order instead of 'order' because the *order* is a reserved word.
Use postgres serial sequence to auto-increment orderId.
See: https://www.postgresqltutorial.com/postgresql-serial/

Use the "`insert into order… returning id`' form of the insert query (specific to postgres), so that the order id is returned in the confirmation message to the user.

4. You must use the session mechanism in django (see See 'https://docs.djangoproject.com/en/3.0/topics/http/sessions/'. This is to maintain continuity between the time the login screen is shown and subsequently the order is submitted.
5. For this exercise, I am not too fussy about what the UI looks like (for this exercise!). But for the sake of automated testing, the form fields should have the names 'idli', 'samosa', 'chai', and there should be a <div id="status"> somewhere in the form that begins with "Success" or "Error".
6. Zip up the project directory and upload as rollnumber1-rollnumber2-ex1-feb4.zip

Exercise 2. Periodic Table. Django Layout, Styling and dynamic UI.

Overview. Display the periodic table, one cell of which has been done in the class. There should be a row of checkbox components at the bottom, one for each *category* of element ("lanthanide", "diatomic nonmetal" etc). By default, all these checkboxes are checked. If a category is unchecked, cells in that category are left blank (visually)

Specifically,

1. The data source is https://github.com/Bowserinator/Periodic-Table-JSON/blob/master/PeriodicTableJSON.json . This contains all the information you could want and more, including the x/y position on a grid.
2. This is an exercise in css grids, a fairly intuitive layout option in CSS. See; https://www.freecodecamp.org/news/learn-css-grid-in-5-minutes-f582e87b1228/
In particular, the container will need a "display: grid", "grid-template-columns" and "grid-template-rows" properties, and the child cells will use grid-row and grid-column for placement.
3. Create a "PeriodicTable" React component as the outside React container, with a "Cell" component for each element. The PeriodicTable's state is a dictionary, that maps

category → boolean values. This remembers which categories are checked and which are not.

4. When a checkbutton is toggled, the associated hander calls a setState on the PeriodicTable, and the table re-renders itself without those cells whose categories are not activated.
5. Important: do not directly update the state. The following is wrong.
   ```
   state[category] = false;  setState(checked);   // BAD
   ```
   Instead, make a copy of the state (dictionary) before modifying it and then call setState, as follows.

   ```
   newstate  = {...state}; // "Spread operator" to clone the object.
   newstate[category] = false;
   setState(newstate) // CORRECT
   ```
6. Create a directory called "rollnumber1-rollnumber2-ex2-feb4"
   a. From codesandbox, File->Export To Zip, and move the zip file to the above directory.
   b. Create a file called `url.txt` in the above directory, and simply copy the URL of your codesandbox project into it. This way, we have a backup of your work, and a direct URL for testing.
7. Zip up the folder into rollnumber1-rollnumber2-ex2-feb4.zip and submit to moodle. Make sure this zip file contains both the url.txt as well the zip file from codesandbox.