LaSalle College
Montréal

Final Evaluation – Part 2: 40%

Course Identification

Name of program – Code:	COMPUTER SCIENCE TECHNOLOGY – PROGRAMMING (420.BP)
Course title:	**WEB SERVER APPLICATIONS DEVELOPMENT I**
Course number:	420-DW3-AS
Group:	07392
Teacher's name:	Jean-François Parent
Duration:	Extended
Semester:	Fall 2020

Student Identification

Name: _____ Student number: _____

Date: _____ Result: _____

☐ I declare that this is an original work, and that I credited all content sources of which I am not the author (online and printed, images, graphics, films, etc.), in the required quotation and citation style for this work.

Standard of the Evaluated Competencies
Statement of the evaluated competency – Code
Develop transactional Web applications – 00SU
Evaluated elements of the competency 00SU
1. Analyze the application developement project *2. Prepare the computer development environment* *4. Program the Web interface* *5. Program the server-side application logic* *6. Program the client-side application logic* *7. Control the quality of the application* *8. Participate in the deployment of the application on the Web host* *9. Produce the documentation*

Instructions

- Students will submit a NetBeans project and a cheat sheet individually.
- It is the teacher's responsibility to identify language errors. If such errors are found, teacher may deduct up to 5% of the final grade (IPEL – Article 5.7).
- Plagiarism, attempts at plagiarism or complicity in plagiarism during a summative evaluation results in a mark of zero (0). In the case of recidivism, in the same course or in another course, the student will be given a grade of '0' for the course in question. (IPEL – Article 5.16).
- Deadlines are shared on Omnivox in the assignment box and must be respected.

- Please see the detailed rubric in the document for the breakdown of the mark for the individual work.

TOTAL: 100 POINTS

Project 3

Guidelines

Note: This is NOT a group project

You have to do this project by implementing all the following working methods:

1. All the files required in this project must be included in a **Subversion repository**. Everytime you work on the project, and the end of every day of work, you must **commit** your work to the repository (even if the code temporarily contains errors). For every commit you do, write a **commit message** which should include your **name** and **student number**, the **date** of the commit, and a **short description** of what you did during that day. For example, this is a valid commit message:

```
Bill Torvalds (1244556)        2020-09-30        Created NetBeans project and empty folders.
```

Note: You may also add more information in the commit message, like a version number, or important steps in the development of your website.

2. At the very top of **all the text files** you submit (i.e. all your **.php** files, **.css** file(s), your **PHP cheat sheet**, etc. (EXCEPT the Subversion dump file) you must write a **revision history** which should include **all the commit messages**, including your **name** and **student number**, the **date** of the commit, and a **short description** of the work done:

```
#Revision history:
#
#DEVELOPER                    DATE          COMMENTS
#Bill Torvalds (1244556)      2020-09-30    Created NetBeans project and empty folders.
#Bill Torvalds (1244556)      2020-10-01    Created buy.php file and completed about
#                                           50% of requirements for this page.
#Bill Torvalds (1244556)      2020-10-03    Created and completed purchase.php file.
#Bill Torvalds (1244556)      2020-10-05    Fixed the bug in the taxes calculation.
#Bill Torvalds (1244556)      2020-10-10    Commented debugging code. Entire project
#                                           completed at 100%.
```

3. Use a **lot of comments** in the code of **all your files**. Use only // or # (except for debugging).

4. Use **folders** for CSS, Images, JavaScript, PHP common functions, and more.

5. Use **constants** to avoid hard-coding. For example use:

define("CUSTOMER_FIRSTNAME_MAX_LENGTH", **15**); #Max length of firstname
define("FILE_PHP_COMMON", FOLDER_PHP . "**commonFunctions.php**");

Note: MAX is an abbreviation but its meaning is clear.

6. Always use **relative paths**. Never use absolute paths for files and folders. Make sure the variables containing files and folders are using the **correct letter case**. For example if a filename is PHPFunctions.php, don't create a constant with the value "phpfunctions.php" (lowercase) or your website may crash (on your server or on a Linux server).

7. Give **clear names** to your **variables** and **functions**.

8. Use **correct indentation** for the code and the curly brackets.

9. You must always use **functions** to generate the HTML code. For example, <!DOCTYPE html> should only be typed once in all the project, even if three different pages echo it. These common functions should be placed in a PHP file located in the PHP functions folder.

10. Every page of your website must **send all the HTTP headers** required to **prevent page caching**, so when users will reload the page, they will always get the latest version of your files.

11. Every page of your website should be able to display all the canadian french characters. So all the files from your website must handle **UTF-8** properly with the correct **HTML tag** and **UTF-8** HTTP header.

12. You must always send **all the network headers before** you start to echo the <!DOCTYPE html> text and the rest of the HTML code.

13. Your PHP code must **always generate valid HTML**. Use **View Page Source** in your browser to make sure all your pages contain **no malformed html** (which would be displayed in red).

14. You must **protect all the PHP pages** against HTML and JavaScript injection.

15. If an **error/exception** occurs in your code, save the details into a specific **folder** and **log file** (give them names). The log file should contain all these details: the **description** of the error, the error **code** (if available), the **date and time** when the problem happened (format should be *year/month/day hour:minute:second.**microsecond***), the name of the **PHP file**, the **line number** of the error, and the **browser version** of the client. You should also create a boolean **constant** to **show** the details in the browser, for debugging, or to **hide** the details to the users (client side) and just display a generic error.

16. When all your tests are done and conclusive, **comment** all your debugging code and set the constant of guideline 15. to **false** because the browser (client side) must **never show technical** (server side) **information** to the users.

Guidelines added since Project 1:

17. You must **protect all your PHP pages** against **SQL injection**.

18. In the entire project, there must be **only one connection string**. If we need to connect on another database, or with another MySQL user/password, we have to change only one single centralized connection string that will impact the whole website.

19. The entire website must use only the **MySQL user/account** you created in Project 2. This user account should have very limited rights on stored procedures and views only. When submitting your project **never use root** in the connection string. You can use root to debug your website, but don't forget to put back the secure MySQL user before delivering your website.

20. All the website should use a **certificate** and **private key** to encrypt the network communications. **Force** the website to use the **HTTPS** port. Submit the .key and .crt files.

Description of the project

For this project, you have to finish the website you started with Project 1 and Project 2.

All the PHP files that generate HTML code should use a common PHP functions file (like Project 1).

All the website should use a secure (HTTPS) connection. When submitting your project, also include the **.crt** and **.key** file which should contain your student number, for example, 1234567.crt and 1234567.key. When creating the files, enter your **student number** for the **Organization Name**. Submit also the **Apache configuration** file necessary to enable these 2 files. Finally, **force** all the users of your website to use the **HTTPS connection**. If a user tries to acces it with HTTP (port), redirect him/her on the secure port.

Objects:

You need to write **plural** and **singular objects/classes** to manage the data which is in the database. There are 3 tables in the database so you will need 6 classes and 6 different files in a **folder**:

- customer
- customers
- product
- products
- purchase
- purchases

Each **singular** class must follow these **requirements**:

1. Every field in the tables must have a **private** variable in the object.

2. The **constructor** of the object should **fill-in** these variables if they are passed as parameters. The parameters should be **optional**. The constructor should use the **Add** method to validate the data (see plural class section).

3. All the private variables (except the primary key) must have corresponding **public get/set functions**. For example, for the field called "firstname" in the database, you will need these functions:

   getFirstname()

   setFirstname($newFirstname)

   The set function should be used to **validate** the data. Return an appropriate **error message** if the validation fails (for example, "The firsname could not contain more than CONSTANT characters"). Return **NULL** or **FALSE** when the field is valid. These error messages are intended to be displayed **in red** near the corresponding fields when there is a validation error (Project 1). You must also retain the typed value

4. You must create 3 functions to **load, save** and **delete** the data in the database. You should use the stored procedures you created in Project 2. The **save** function should be able to save new data (**INSERT**) or existing data (**UPDATE**). You can solve that in PHP or in the stored procedure(s). You may create any other stored procedure if you need it (grant EXECUTE rights to your MySQL user if you do it). You may also add any others functions to the object if you need it.

Each **plural** class must follow these **requirements**:

1. You must use inheritance to centralize the following:
   - A **variable** used to store the **list/collection/array** of all the singular objects. For example, the **customers** object should keep a list/collection/array of many **customer** objects.

   - Functions to **add**, **remove**, **get** and **count** the objects in the list (not in regards to the database).

2. The **constructor** of the class should **fill** the **list/collection/array** with all the rows returned by the SELECT stored procedure you created in Project 2. For the purchases, load the collection only with the purchases made by the logged in user.

Your entire website must **always use these objects** to manage the customers, products and purchases.

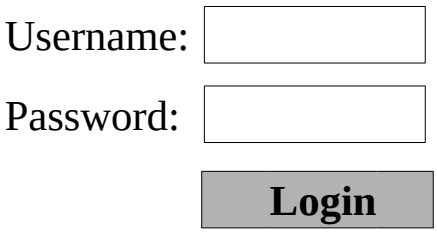
Website pages

You must not use any style attribute directly in HTML tags. All the pages should make use of at least one **.css file** and all the pages must have a **background color** or image. The design of your pages must not look too minimalistic.

The website consists of 4 (or 5) pages, which must display a **different <title>** in the browser **tab**. All the pages of the website should call a single PHP function to manage the login and the logout:

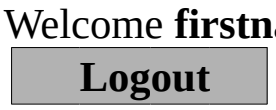
Function for login/logout

This code should be placed in a **function** located in the common functions PHP file because it must be used on all pages of your website. So on all pages of the website, if a user is not connected, you should display a **login <form>** like this:



If the user clicks the **Login** button, **validate** the **username** and **password** against the **encrypted** password in the database (in the customers table). If the username and/or password are not valid, display a general **error message** on the screen and let the user try again (for that form, no need to fill-in the username/password previously entered). If the user clicks **Register**, simply redirect him/her to the **register.php** page (explained later).

If the user enters a **valid** username/password, create a **$_SESSION** variable to store the **primary key** of this user/customer, and reload the page. When a user is logged-in, the function that usually display a **login <form>** on all pages should instead display a **logout <form>:**



The firstname and lastname should correspond to the logged-in customer, so you have to load them with a **customer->login($username, $password)** function which will load all the object properties if the username/password are correct.

If the user clicks **Logout**, it should **destroy** the **$_SESSION** variable and reload the page, thus preventing the access to many pages on the website.

Pages required for your website

- **index.php**

This page can be consulted even by a user which is not connected to your website.

This is the welcome page that displays the **logo** of the company and the description you made of it. The logo should also be included in **all the pages** of your website.

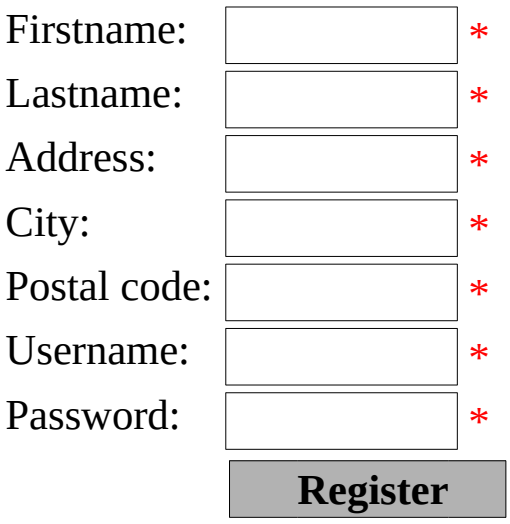
This page, and all the others, should display the same **navigation menu** to browse all the following pages: **index**.php, **buy**.php, **purchases**.php and **account**.php.

In addition of the company description, this page should contain a **link** to open your latest CHEAT SHEET which is asked since the beginning of the semester.

- **register.php**

This page can be consulted only by a user which is **not** connected to your website.

This page is used to **create a customer**. You should display a **register <form>** like this (your website should display * = required and display the red * near the required textboxes):



Note: When we type the **password**, we must **not see the letters**. Also you must **encrypt the password** before saving it in the database.

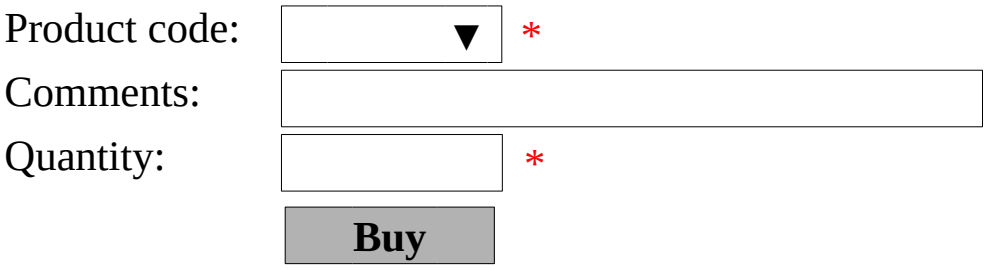
Like in Project 1, **validate** the data (but use the **get** functions to do it) and display each **error message** near the corresponding field. There is no need to write a clear message if the user tries to create an already existing username (which is UNIQUE in the database). The code may raise an error. It's correct as long as the system does not permit duplicate usernames in the table.

Note: This register.php page must NOT be displayed in the **Navigation menu** visible on every page. It can only be accessed via the **login <form>**.

- **buy.php**

This page can only be viewed if a user is connected to your system. If the user is not connected, display a message which tells the user he/she must login to the system to access this part of the website. Display also the **login <form>**.

Else, display the **logout <form>** and the following **buy <form>** used to perform a purchase (your website should display * = required and display the red * near required textboxes):



The product code dropdown should be filled with the data from the database.

The **product code** dropdown should display the **product code**, a **hyphen**, and the **description**, for example "P43helmet – Motorcycle Full Face Helmet". However when saving data, don't save the code or description but the **product_uuid** that corresponds to them.

The **comments** field is optional but must not exceed 200 characters.

The **quantity** field must be a numeric value between 1 and 99. No decimals are allowed, so a quantity of 1.3 is not valid.

If the data from some fields is not valid, you have to **write in red**, **near every corresponding field**, a clear error message telling the user how to solve the problem (for example: The last name cannot contain more than 20 characters).

When all the data is valid, you have to **save** it in the database (**purchases** table). Don't forget to specify the **UUID** of the user that is currently logged in the system for the purchases.customer foreign key.

Some fields were not mentionned in Project 2 so you have to add them to the table, stored procedures and code. Always follow good working methods (field size, etc.). Backup the new database definition before you submit project 2.

The first missing field you need to add to the table is the **subtotal** which represents the price of the product (available in product object) multiplied by the quantity. You then have to apply the local taxes of **15.2%** (the taxes rate changed since Project 1) to this subtotal to get the **taxes amount** in dollars (second missing field). Finally, add the subtotal to the taxes amount to get the **grand total** (third missing field). Save these 3 fields in the database and make sure you **always save only 2 digits** for all amounts in the database.

Example of taxes calculation
Let say a user selects **P43helmet – Motorcycle Full Face Helmet** in the dropdown.

When you POST the **buy <form>** you receive the **product_uuid**. Use that primary_key to call the **product->load()** function. Then get the price for this product. For example:

```
#Load the product
$product->load(  htmlspecialchars($_POST["product"]);

#Let say the price is 49$ in the database
$price = $product->getPrice().

#and the quantity 2
$quantity = htmlspecialchars($_POST["quantity"]); #not secure against HTML injection

#Then you multiply the price by the quantity, which gives you the subtotal:
$subtotal = 49 multiplied by 2;  #98$

#calculate the taxes amount by multiplying the subtotal with the taxes rate
$taxesAmount = $subtotal multiplied by the TAX_RATE;        #use a constant!
                                                            #This gives 14.896
#finally add the subtotal to the taxes amount
$grandTotal = 98 plus 14.896;  #this gives 112.896
```

Then instead of saving all this information in a file, **save** all these fields in the **purchases** table:

product_uuid	(the product selected in the dropdown)
customer_uuid	(the customer selected in the dropdown)
quantity	(how many items the user wanted)
price	(actual price of product. If tommorow we change product price in products table, we'll still be able to see the billed price)
comments, subtotal, taxes_amount, grandtotal	(as explained earlier)

When a purchase is complete, **redirect** automatically to the purchases page.

- **purchases.php**

This page can only be viewed if a user is connected to your website. If the user is not connected, display a message which tells the user he/she must login to the system to access this part of the website. Display also the **login <form>**.

Else, display the **logout <form>** and a **search <form>** similar to this one:

Show purchases made on this date or later: 2020-03-13

Search

When the user clicks the **Search** button, you should call the customer object, which should call the stored procedure you created in Project 2 to **filter the purchases**. Modify the stored procedure (and backup your DB) because you need to show the purchases **only for the customer** which is logged to the website. The customer_uuid which is in the $_SESSION variable should help you with this requirement.

If the user specifies a **valid date**, use it to show the purchases made **on that day or later**, or if you prefer, show purchases made on a create_datetime greater or equal to the valid date. **Sort** by the create_datetime field.

If the user specifies a date which is **not valid**, it is not mandatory to write an "error" message but you must show **all the purchases** in the database for this customer.

To fetch the data from the purchases table, use **AJAX** and the **purchases object**. The AJAX call should query a page which will generate only a **HTML <table>** with the appropriate **column headers**, **<tr>** and **<td>**. This table should show **only the purchases made by the user** which is logged in the website. You must also find a way to **delete** a purchase.

All the borders must be visible, so the table may look like this:

Delete	Product code	First name	Last name	City	Comments	Price	Qty	Subtotal	Taxes	Grand total
Delete	Phelmet4b	Ben	Masvidal	Montréal		49.99$	2	99.98$	12.05$	112.03$
Delete	Pgloves675	Justin	Legault	Québec	10 % rebate	22.49$	1	22.49$	2.71$	25.20$

Important: the dollar ($) signs must not be saved in the text file, but should only be added in the HTML <table> for display. Always display amounts with **2 decimals**.

To avoid multiple calls to the database, the stored procedure which **filters the purchases** should make use of the **JOIN** command to fetch data from the 3 tables.

The Delete button should **delete** the corresponding **purchase row** in the database (no confirmation needed). If you prefer to use another way to delete a purchase (even outside of the HTML table and AJAX call) you will get the marks for it.

- **account.php**

This page can only be viewed if a user is connected to your system. If the user is not connected, display a message which tells the user he/she must login to the system to access this part of the website. Display also the **login <form>**.

Else, display the **logout <form>** and the same **register <form>** which is in the **register.php** page. The form should be **filled-in** with the information found in the database for the current user (except the password which needs to be typed again (old password or new password, just save the new password encrypted in the database)):

Firstname:	Linus	*
Lastname:	Gates	*
Address:	1 OS Drive	*
City:	Freedom	*
Postal code:	G1B 6D2	*
Username:	Linux	*
Password:	●●●●●●	*
	Update info	

Note: When typing the **password**, we must **not see the letters**. Also you must **encrypt the password** before saving it in the database.

The validation is all the same than the register.php page except that when the data is valid, you must **UPDATE** the data in the database instead of inserting it. Even if you change the password for the user, there is no need to logout the user from the website. However, next time he/she will have to use the new password.

Optional (advanced): Instead of creating an account.php page, you can modify the register.php page to accomplish both tasks (INSERT (register) and UPDATE (Update info). If you do this, make sure not to create any bug.

Files to submit

When your website is ready, place all your files (the whole PHP project including your PHP cheat sheet) in a single compressed folder (**.zip file**). Make sure your .zip file is not corrupted by extracting its contents into a different folder. **Upload** that single verified **.zip file** on **Omnivox** to submit your project.

Don’t forget to also submit your latest **database backup** and **GRANTS** .sql files required with Project 2, and required to run your Project 3 properly.

CORRECTION GRID FOR REQUIREMENTS

Compentency : Deploy transactional Web applications – 00SU	
Elements of competencies: Analyze the application developement project (00SU.1)	
Performance criteria	**weight**
Accurate analysis of design documents (00SU.1.1)	/5
Proper identification of the tasks to be carried out (00SU.1.2)	/2
Elements of competencies: Prepare the computer development environment (00SU.2)	
Performance criteria	**weight**
Proper installation of the Web development platform and the development database management system (00SU.2.1)	/5
Proper installation of software and libraries (00SU.2.2)	/1.5
Appropriate configuration of the version control system (00SU.2.3)	/1
Proper importing of the source code (00SU.2.4)	/1
Elements of competencies: Program the Web interface (00SU.4)	
Performance criteria	**weight**
Appropriate use of markup language (00SU.4.1)	/4
Suitable creation and use of style sheets (00SU.4.2)	/2
Proper integration of images (00SU.4.3)	/0.5
Suitable creation of Web forms (00SU.4.4)	/5
Adaptation of the interface based on the display format and resolution (00SU.4.5)	/0.5
Elements of competencies: Program the server-side application logic (00SU.5)	
Performance criteria	**weight**
Proper programming or integration of authentication and authorization mechanisms(00SU.5.1)	/2
Proper programming of interactions between the Web interface and the user (00SU.5.2)	/3
Appropriate choice of clauses, operators, commands or parameters in database queries (00SU.5.3)	/4
Correct handling of database data (00SU.5.4)	/7
Appropriate use of data exchange services (00SU.5.5)	/6
Proper application of internationalization techniques (00SU.5.6)	/1
Precise application of secure programming techniques (00SU.5.7)	/4
Elements of competencies: Program the client-side application logic (00SU.6)	
Performance criteria	**weight**
Correct manipulation of DOM objects (00SU.6.1)	/2
Proper programming of asynchronous calls (00SU.6.2)	/3
Proper programming of interactions between the Web interface and the user (00SU.6.3)	/7
Systematic use of Web form data validation techniques (00SU.6.4)	/5
Web forms in compliance with usability requirements (00SU.6.5)	/1
Elements of competencies: Control the quality of the application (00SU.7)	
Performance criteria	**weight**
Precise application of test plans (00SU.7.1)	/1.5
Thorough reviews of code and security (00SU.7.2)	/4
Relevance of the corrective actions (00SU.7.3)	/2
Compliance with issue tracking and version control procedures (00SU.7.4)	/1
Compliance with design documents (00SU.7.5)	/5
Elements of competencies: Participate in the deployment of the application on the Web host (00SU.8)	
Performance criteria	**weight**
Accurate identification of the domain name (00SU.8.1)	/2.5
Appropriate configuration of the application on the Web host (00SU.8.2)	/1
Proper application of the procedure for migrating the service onto the Web host (00SU.8.3)	/3
Precise application of security measures (00SU.8.4)	/1.5
Compliance with search engine indexing requirements (00SU.8.5)	/1

Elements of competencies: Produce the documentation (00SU.9)	
Performance criteria	**weight**
Proper identification of the information to be written up (00SU.9.1)	/4
Clear record of the work carried out (00SU.9.2)	/1

CORRECTION GRID FOR LANGUAGE

Clear Communication	Clear Comm., **most of the time**	Vague Communication	Unclear Communication
- 0	- 0,5	- 1,5	- 2
(Word Choice) Use of precise and rich vocabulary	(Word Choice) Use of precise vocabulary	(Word Choice) Use of imprecise vocabulary	(Word Choice) Use of inappropriate vocabulary
- 0	- 0,5	- 1,5	- 2
(Format/Type of work) Respect of norms	(Format/Type of work) Respect of **most of the** norms	(Format/Type of work) Non-respect of the norms	(Format/Type of work) Inappropriate in relation to the required norms
- 0	- 0,5	- 1,5	- 2
(Linguistic Code) (≤2 mistakes / page)	(Linguistic Code) (3-7 mistakes/page)	(Linguistic Code) (8-10 mistakes/ page)	(Linguistic Code) (>10 mistakes/page)
- 0	- 0,5 - 2.5	- 2.5 - 3.5	- 4