Introduction

The system will be planned following the MVC architecture. By following the MVC architecture, it will be possible to organise easily codes, facilitating the management of the code, decoupling logic, data and view. To achieve the final solution, the following frameworks will be used in the process:

- Slim Framework – A micro PHP framework to deal with server calls. Slim itself is not based on MVC, so it has to be adapted. http://www.slimframework.com/
- Twig – A template engine to render views. It supports some features that improve reuse, such as template inheritance. http://twig.sensiolabs.org/
- Doctrine 2 – A PHP library to deal with database storage. It transfers data from entities to database, regardless of database management system used. http://www.doctrine-project.org/
- jQuery – A JavaScript library to handle DOM manipulation, event handling, animation and AJAX. Functionalities http://jquery.com/

Login and logoff

Log in is the most fundamental feature of the system. Users who are not registered and are not logged cannot browse books or access their cart or wish list. To guarantee that users will be logged, the following strategy will be adopted:

- When a user visits the website, a default page will be shown requiring login details;
- After authenticating, the user will be redirected to the browse books page and will have access to the others features;
- If the user has not registered yet, there will be a link to register;
- If a user access some of the restrict access pages, the default page will be shown.

When a user authenticates to the system, a logoff option will be shown on the top right of the layout, facilitating the logoff process to the user, since it is the most design solution used in web systems.

Shopping cart

Following the tendency of others web applications, there will be a link located on the top right of the layout, next to the logoff button, so users can access their cart easily and rapidly. Just like Amazon, there will be an indicator to show how many products there is in the cart and a mouse over event to show the items added to the cart, so users can track easily which and how many items were added. This approach will be used to enhance the user experience. In addition, there will be a page of full description of their cart, indicating the quantity of each item, the price final and a link to checkout.

Wish list

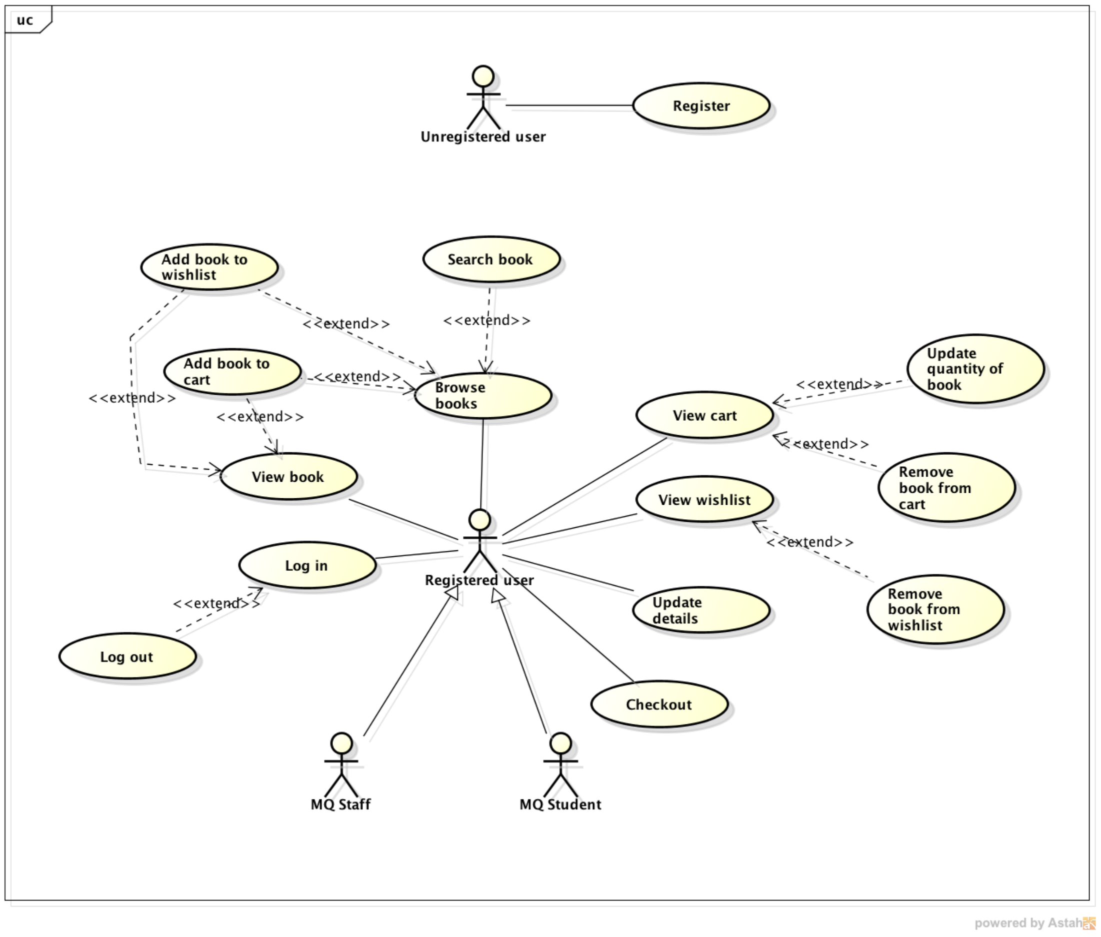
Just like the shopping cart, there will be an icon to indicate how many items there are in a user's wish list. This strategy was chosen to minimise the number of clicks to find the wish list. However, items will be not displayed, since wish lists may be a large collection of items, hindering to display them on the screen. Also, having two close features with similar behaviours may confuse users. When the user clicks on this icon, a new page will be displayed, showing the user's wish list and some others information when the price of a product is changed or it is out of stock.

Users' information

Users' information will be stored in a database after their registration and can be updated later. When a user is registering or updating, the fields will be validated on server and client side. This strategy will be adopted to enhance the user experience, so users can correct mistakes rapidly without the necessity of sending the data to the server and also to avoid data adulterated via browser tools, for example. Also, credit cards will not be stored because of the complexity of having to encrypt them, even though this is not a "real" application. In real applications, this would avoid the possibility of fraud and data stolen, minimising possible judicial problems.

Use case diagram

The following diagram shows the use cases of the system to easily understand how the features are connected and who has access to each feature.

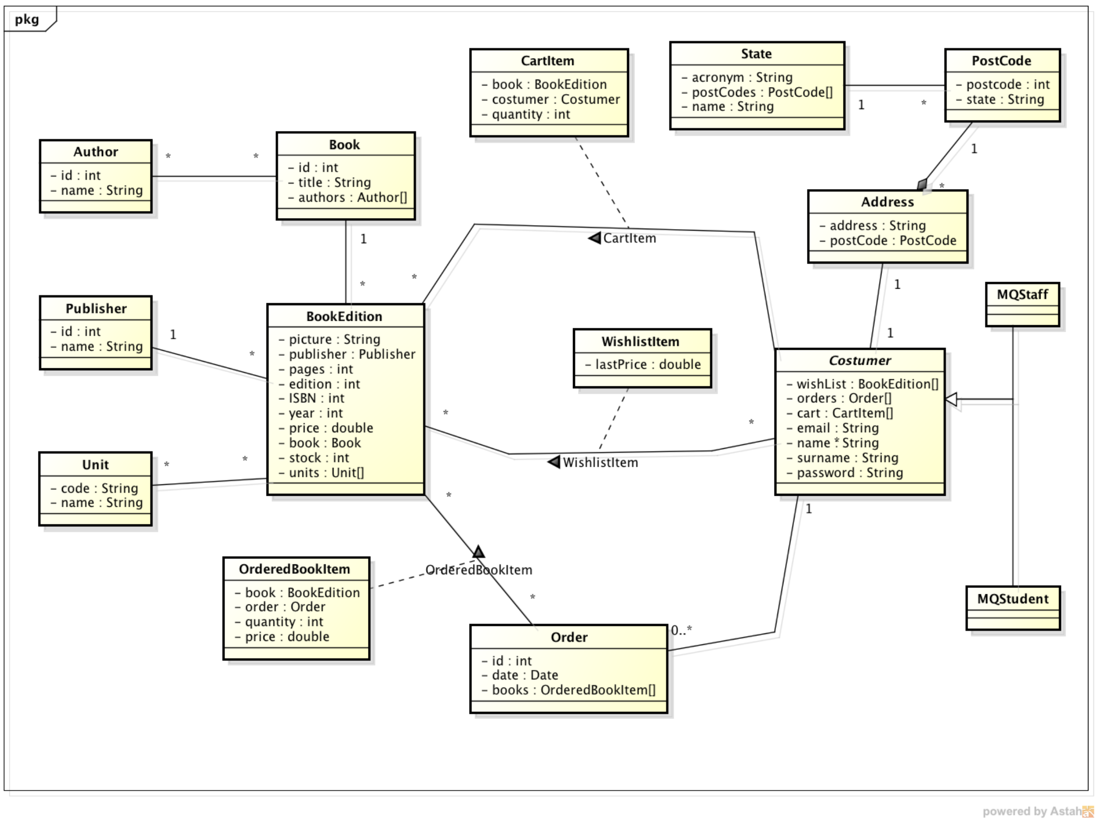


Organisation

Since the MVC architecture will be used in the system, the following sections will describe how each part – models, views and controllers – will work.

Models

The following diagram shows the relationship between the entities that composes the model part of the system. The entities storage will be based on Doctrine, a framework based on Java Hibernate that retrieves and persists data from entities to the database.



Although the diagram is self explanatory, the following decisions were taken:

Costumers, MQStaff, MQStudent

Costumer class itself is abstract, because they can be either MQStaff or MQStudent. Although it seems that there is no need to separate them, staffs and students are being treated as different classes, so the system can differentiate users from staff, being able, in the future, to add some feature for each class of costumer.

Address, postcode and state

There is a class to represent address, postcode and state to facilitate the validation of postcodes and to normalise the database, so there will be a table of postcodes and each postcode is linked to its state in the table of states.

Books, edition of books, authors, publishers and units

Since books can have many editions, they will be treated differently. This decision were taken so users can search for books by their names, for example, and see which editions are available for purchase. Thus, book class contains the name of the book and its authors and the book edition the rest of the details, such as edition number, number of pages and publisher. Also, book edition can be indicated by one (or more) unit convenor; thus, instead of having a relationship between book and unit, the relationship is between book edition and unit. In addition, authors and publishers must be registered in order to link them with books.

Order and book item

In order to be possible to save the quantity of items purchased and the price, it is necessary to have an association class, which is called “OrderedBookItem”.

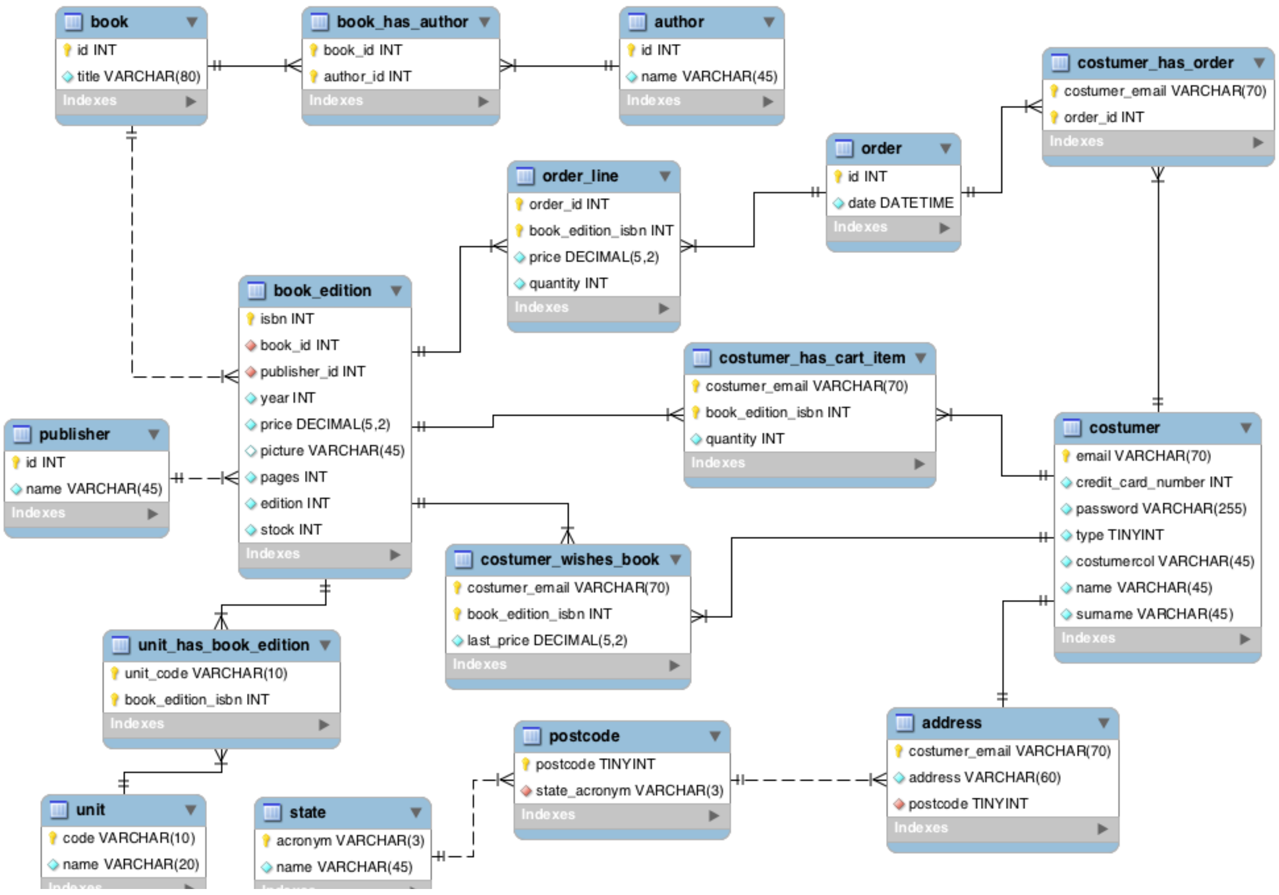
Books from cart

Since there is only one cart in the system, there is no need to have an object for it. Thus, the association class works as a cart, storing which book edition the user added and the quantity.

Books from wish list

Just like books from cart, there will be only an association class representing the wish list, since it is not possible to have more than one wish list.

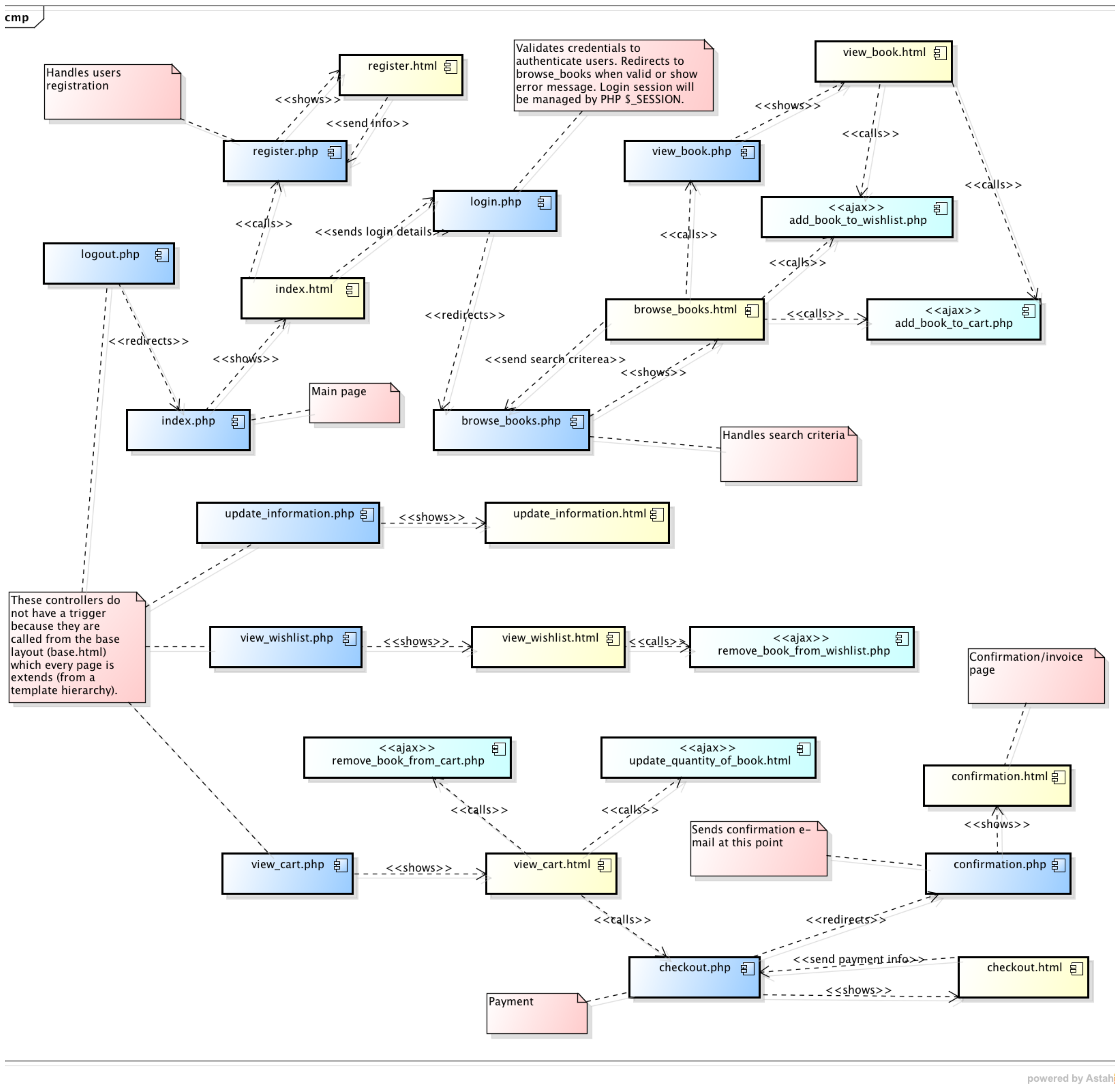
Each class has a PHP file associated to it and the following diagram represents the database scheme.



Communication between controllers and views

Since the application will be built in MVC architecture, some conventions will be used. Views (or templates) will have HTML extensions but rendered by Twig template engine. Besides HTML markup, these files contain some tags recognised by Twig to give some extra logic, such as loops, conditions and inclusion and extension of others templates. For controllers, classes will be used instead of PHP files with structured logic. There are situations which views are not necessary, for example, in AJAX calls; thus, each controller may or may not have a view.

For diagraming purposes, controllers will be represented as PHP files and views for HTML files for easier understanding.


cmp
Handles users registration
register.html
<<shows>>
<<send info>>
register.php
Validates credentials to authenticate users. Redirects to browse_books when valid or show error message. Login session will be managed by PHP $_SESSION.
view_book.html
<<shows>>
<<calls>>
view_book.php
<<calls>>
<<ajax>> add_book_to_wishlist.php
login.php
<<calls>>
<<sends login details>>
logout.php
index.html
<<calls>>
<<calls>>
<<redirects>>
browse_books.html
<<calls>>
<<ajax>> add_book_to_cart.php
<<redirects>>
<<shows>>
<<send search criterea>>
<<shows>>
Main page
index.php
browse_books.php
Handles search criteria
update_information.php
<<shows>>
update_information.html
These controllers do not have a trigger because they are called from the base layout (base.html) which every page is extends (from a template hierarchy).
view_wishlist.php
<<shows>>
view_wishlist.html
<<calls>>
<<ajax>> remove_book_from_wishlist.php
Confirmation/invoice page
<<ajax>> remove_book_from_cart.php
<<ajax>> update_quantity_of_book.html
confirmation.html
<<calls>>
<<calls>>
<<shows>>
Sends confirmation e-mail at this point
view_cart.php
<<shows>>
view_cart.html
confirmation.php
<<calls>>
<<redirects>>
<<send payment info>>
checkout.php
checkout.html
<<shows>>
Payment
powered by Astah