# Web Programming Report

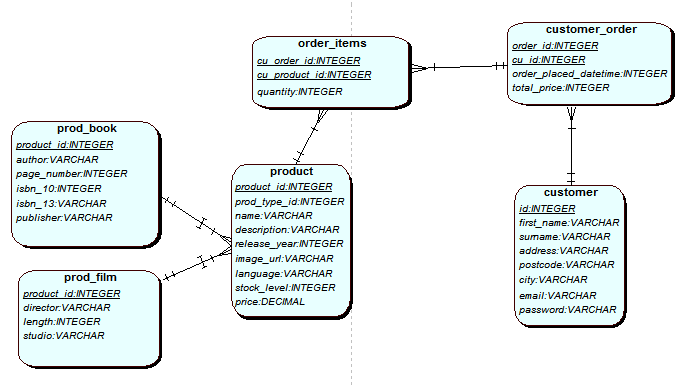
Student ID 10029052

## Introduction

This report is for the web application built for the only Web Programming assignment for the academic year 2011-2012; and will critique the system and its separate components.

## Entity-relationship model

While the assignment specification only required a conceptual E-R model, a decision was made to create a full-blown logical E-R model, as this was not that much more effort than creating a purely conceptual model, but it would also allow for exporting the model to SQL, which would save time when building the database at the later stage. Below is the final E-R model, which had undergone a few iterations after the project was started, as it was realised that some of the requirements that was kept in mind for the system had changed.



## Reusability analysis

From the start of the planning phase it had been made clear that the application should be built using open libraries and be as modular as possible – certain components that are not specific to the domain chosen (in this instance: a Sci-Fi book and film store) should be able to be reused in other similar environments with very few changes required, following the “Don’t Repeat Yourself” and “Don’t reinvent the wheel”-principles, saving time and effort which can be focused elsewhere as a result. This was kept in mind from the start of designing the database throughout the entire project.

The database design was influenced by this school of thought, and as a result, was as modular as possible. The customers table, the order table and the order items link table are very generic – not a bad thing in this instance – as they can be used as-is for any new online systems that deal with users, orders and products in any fashion.

The products table is also fairly reusable, as it branches the truly domain-specific properties to separate tables, but this would likely require some modification for most online systems to fit in easily.

As for the web application itself, it has a few generic components, though it is generally very difficult to avoid spaghetti code (mixing presentational properties and program logic) in web applications that are not based on a Model-View-Controller framework, which was not the case in this instance. During the planning phase of the project, a few generic components were identified with high potential of reusability: namely the cart, database connections, and payment validation for use with the card clearing RESTful service, and during the development, it was attempted to make these as modular as possible through the use of functions.

The execution of this, however, was more successful in some areas than others. The cart functionality is fairly compartmentalised, with separate functions to display the cart, add a product to the cart, and remove a product from the cart, though this is heavily tied in with the way the application handles sessions, and as a result, if these components were to be re-used elsewhere, they would have to be either modified significantly or used together. There is a clear potential for this to be improved upon further.

A separate “miscellaneous functions”-file was also created for the purpose of saving repetition for some common task, and contained amongst others, a wrapper function to check whether a string has the right amount of characters in it – which was used extensively for the input validation, a function to validate commonly used payment details, a function to check whether the application is being ran from inside the university, and another function extending this to download a file using the university proxy settings if you are running it off the university serve. In the true spirit of not repeating oneself, the latter two functions were reused from the Data, Schemas and Applications assignment. A lot of these functions could be used for other projects with little to no modification required.

The database connection is also partially compartmentalised to functions, though, in retrospect, there is quite a bit of duplication between the database connections file, the configuration file, and the “miscellaneous functions” file all have a slightly different way of figuring out whether the file is being ran off UWE’s server or a local server like XAMPP – in other words, plenty of repetition that could have been removed.

## Other remarks

As it stands, the entire system is not based on object-oriented principles. While the framework that was provided was not based around the notion of object-orientation, a system doesn’t necessarily need to be 100% object-oriented to benefit from it. Simply applying object-orientation to a few components, like the cart, products and users, would substantially increase maintainability and reusability. The products would especially benefit from an object-oriented approach, as the way the database is designed is basically based on inheritance. A book/film has domain-specific properties, like author vs. director, but inherits common properties such as description, title and image from a generic product.

While implementing the registration, it was discovered on the internet that using md5 to hash passwords is not an ideal solution, as md5 and most other hashing algorithms were designed for speed, and if the database gets compromised, this makes it quicker to brute-force passwords, especially as the system did not implement very strict password requirements.

To compromise for this, an existing PHP framework, PHPass (<http://openwall.com/phpass>), based on the Blowfish bcrypt hashing method was implemented, which not only made the password storage more secure, but making use of this library also saved some time to invest in other areas of the application.

## Further improvements

While the assignment specification itself did not require more advanced functionality that is standard on most other similar sites, it would have been beneficial to implement them for the experience as well as the ability to reuse these components later. Some of these improvements could be a password reset function, the ability to update your details, address/credit card storage, and the permanent storage of carts if the user is logged in – as it stands at the moment, the cart is entirely based on PHP sessions, and it would not have taken a great amount of effort to implement this, while being greatly beneficial to the users of the website.

## Learning outcomes/Conclusion

While I have dealt with PHP in many instances before – in other modules like DSA, WordPress development, personal side-projects and the like, this is the first time I have ever implemented a server-side application of this scope and complexity. While I have worked with e-commerce stores before at my part-time developer job – mostly using Magento (based on the Zend framework) – and while this obviously saves a huge amount of time, I feel it has been greatly beneficial in terms of my understanding of development to deal with the databases directly without the comfort that the abstraction of a framework provides; and these skills will also be transferrable to the next time I deal with Magento or any other programming framework.

As well, this is the first time I have designed a database entirely from scratch – and while I found it very challenging at times, and definitely has given me a new-found respect for database architects – I have come away from this project with a solid, basic understanding of logical database models and resolving many-to-many relationships, something I was entirely unaware of at the start of the project.

In conclusion, it has been an incredibly enriching experience, and I am now much more confident in my skills as a developer after completing this assignment – as well as more prepared for the exam, especially the E-R modelling part of it.