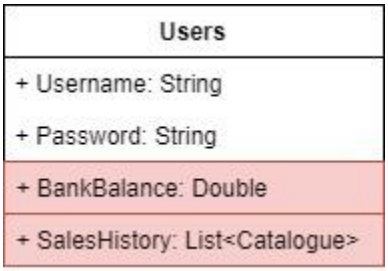
Retrospective Report

PROCESS TO ADD NEW FUNCTIONALITY TO AN EXISTING SYSTEM

The new functionality that had to be added was the shopping cart and to mock a payment system. A domain was already in place for users, so a bank balance was simply added to the domain as well as a reference for a shopping list. The following is a UML diagram with additions marked in red.



After making these changes in the domain, the database was scaffolded and all existing fields were auto updated in the database, making it an easy and effortless transition from the previous phase. The values were made nullable fields so existing records were unaffected.

DIFFICULTIES FACED WITH IMPLEMENTATION OF PHASE 2

No difficulties were faced as domain was well structured and catered for additions.

ASPECTS OF NEW FEATURES

Payment Engine: Uses a probability of 95% as a success rate. I used an easy method which generated a random number from 1 to 100. If the generated number was less than 95, payment was successful, otherwise returned a failure.

Shopping Cart: the shopping cart was made purely on the front-end. I used the same objects from the catalogue to create the cart. Because we already had these items in the scope, it made it very easy to make a reference of them on the fly.

SOFTWARE ARCHITECTURE IMPLEMENTED

The system uses a monolithic approach as the system is relatively small and does not have a sophisticated domain. This means that data can be represented in a flat tabular format. There was no need for multiple API's as the backend only consisted of a few functions.

The monolithic system however evolved to cater for a microservices approach if the website grew in the near future. The controllers communicate with a common language to pass and receive data; JSON. This makes it easy to communicate regardless if data is flowing from a variety of independent databases.

ARCHITECTURE ACCOMMODATING OR HINDERING THE IMPLEMENTATION OF PHASE 2?

The architecture was accommodating as the API calls were already set up. I just had to add new functions as per requirements.

The database was already connected to the backend and because the new features were easily adjusted into the existing database, no difficulties were faced.

ARCHITECTURE WHICH WOULD BE MOST APPROPRIATE

In the context of the assignment I think that the monolithic architecture would suffice, reason being, the system's domain is very simple and does not require multiple servers and databases to store information.

However, if the system grows to an extent where an actual payment is added, then a microservices approach would work better. You would like to have a separation of concerns for payments and users so if one is compromised, the other is still safe.