TEESSIDE UNIVERSITY

SCHOOL OF COMPUTING

**SOFTWARE ARCHITECTURE (COM3041-N) INDIVIDUAL DOCUMENTATION**

Student Name: **Matthew Frost (L1426439)**

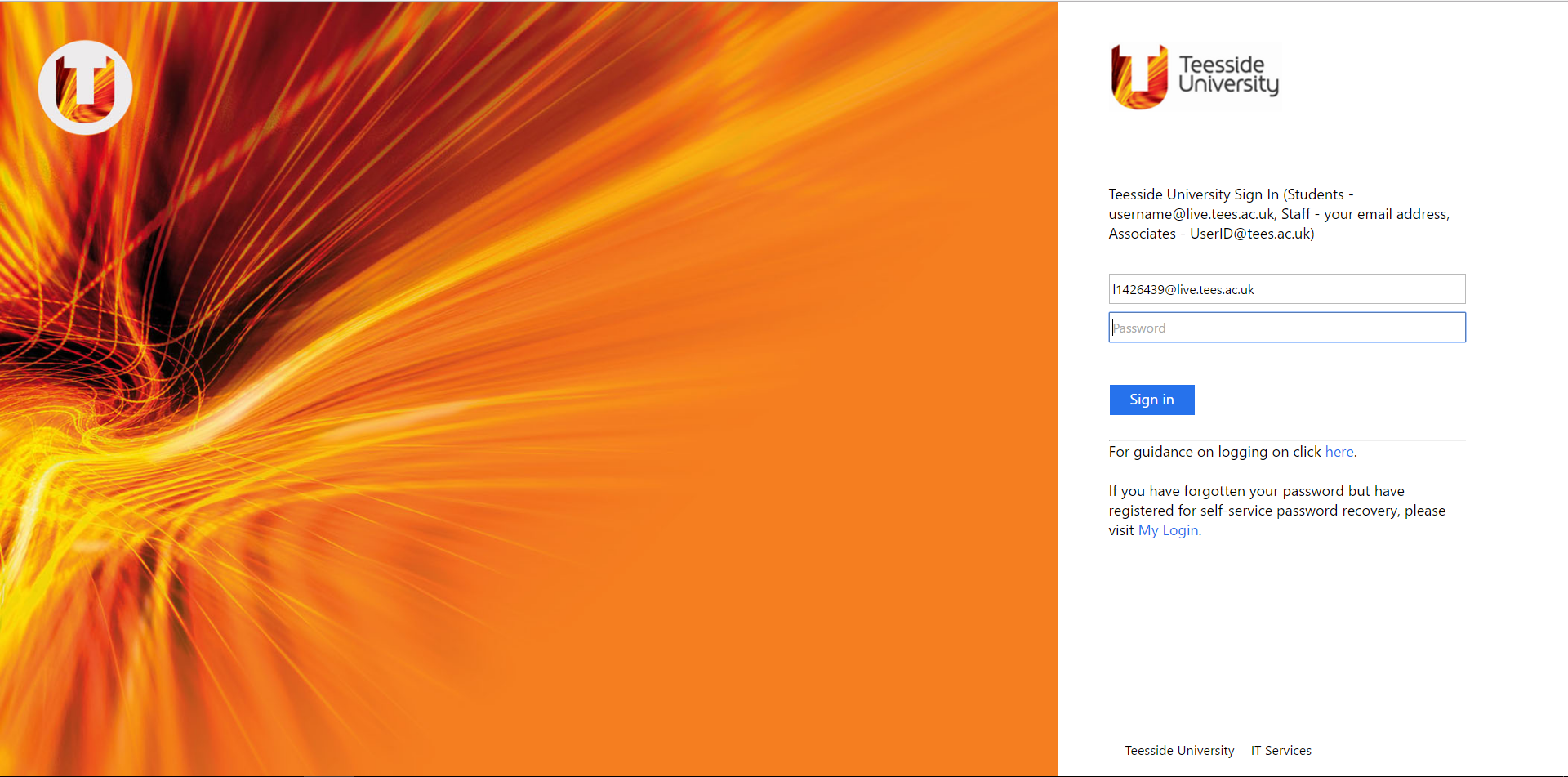
# Work Packages

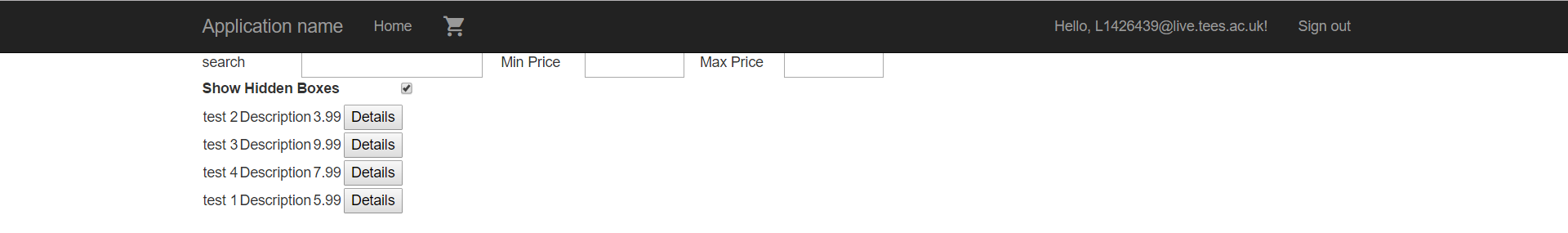
Initially I was also assigned to work on the repositories however we realised that it wouldn’t make sense for me to work on the repositories while someone else was working on the web services that used them. So the repositories were handed off to the person who was also doing the repositories, this seemed to make the work packages fairer and more suitable.

# Requirements satisfied

## Securely authenticate and authorise users

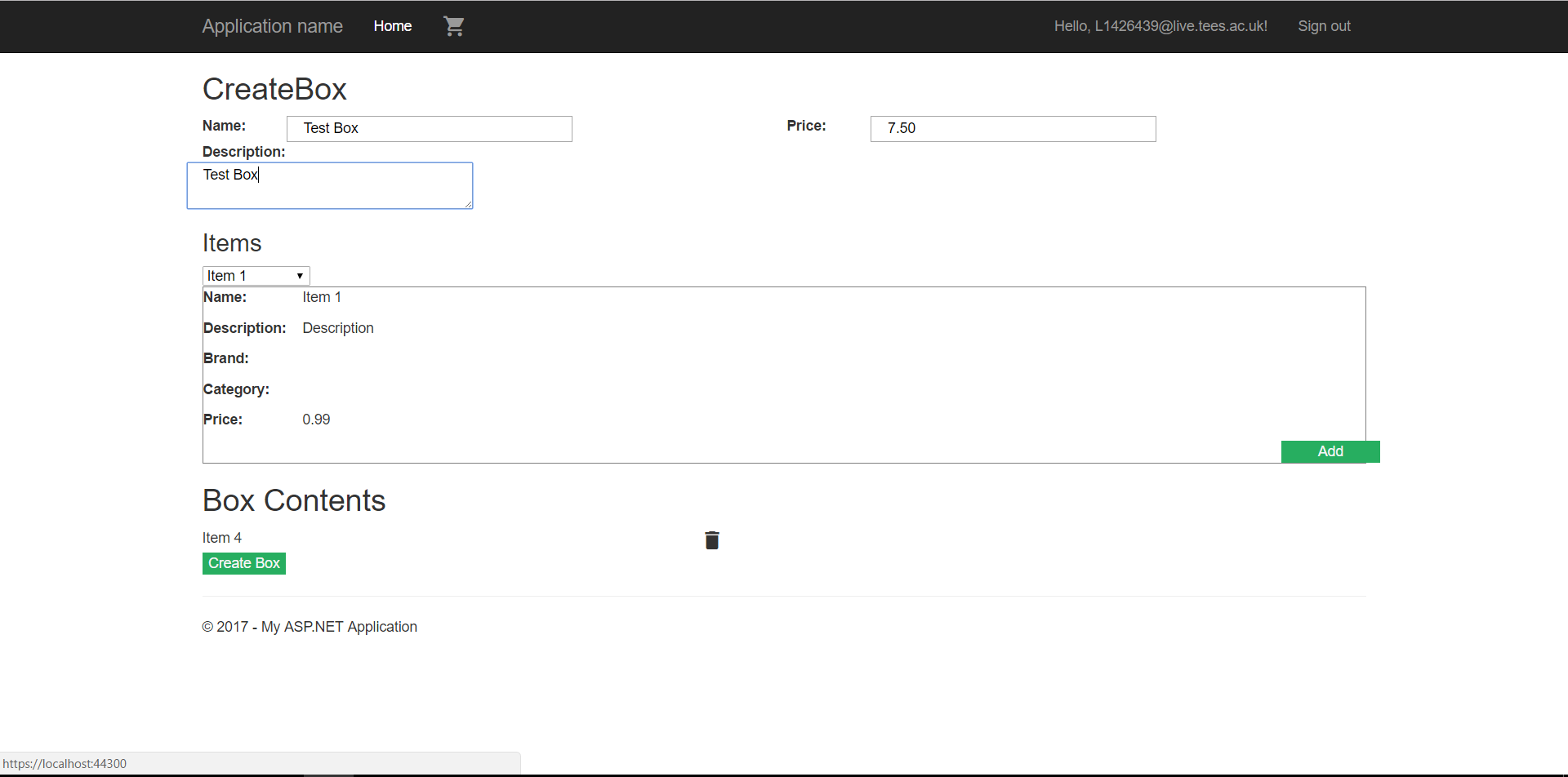
The application uses Azure Active Directory to authenticate our users. Our application requires users to log in to buy boxes and staff to log in to create and edit boxes.

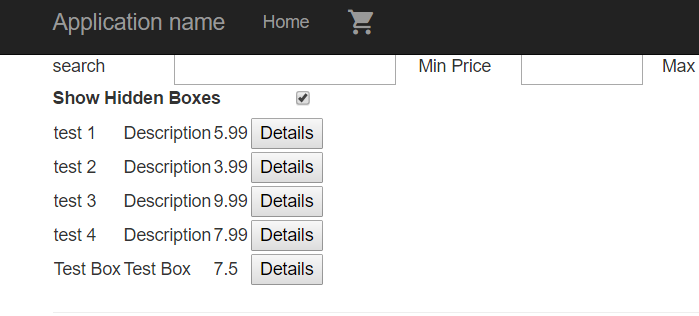




## Create a selection box design by selecting one or more items and a suitable box/wrapping from the approved providers, optionally using an existing design as an initial template.

It is possible for staff users to create boxes from scratch, add items to them, write a description and set a price. Currently boxes don’t have wrapping assigned at creation this is something that was over looked, instead customers choose the wrapping they want when they are ordering the box. Also there is no option to create a box based on a previous box, this again is a section of the requirement that was overlooked.



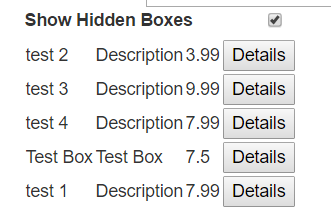


## Modify the price of a selection box design

When staff are logged in they will be able to use the edit button on the details page of any box, this allows them to change the price of a box



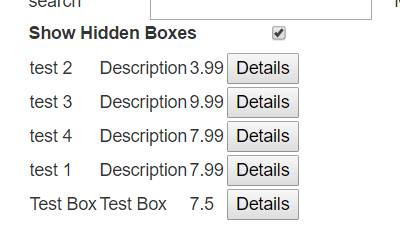


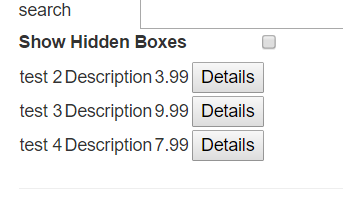


## Modify the visibility of a selection box design

The visibility of a box can also be modified by staff when using the edit button. Hiding a box means that it will not be visible for any customers logged in or not







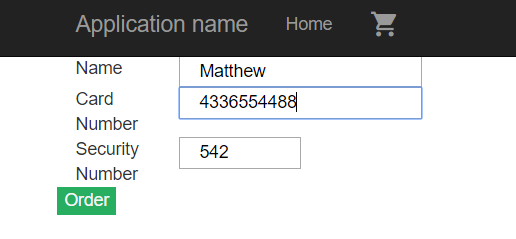
## Monitor both the availability (i.e. if it can be purchased) and visibility (i.e. if staff want it to be listed on the public-facing system) to our customers of our selection box designs

Staff have the option to hide boxes to customers so that they will not be displayed on the home page. Some boxes will still be visible view but are unable to be purchased this happens when one or more of the items are not in stock



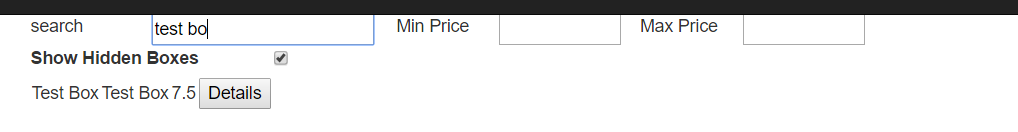
## Mock payment approval / refusal

The web application has a mock payment page which does some client side validation to make sure that all information has been entered. There should also be some validation in our orders web service, by having both client and server side validation this helps keep a solid user experience while remaining secure. If a real payment system was to be implemented, then a trusted 3rd party system would be put in place to make sure it was secure.



## Browse all visible selection boxes with suitable filtering and searching

The Index page allows the user to search for boxes by name and search for all boxes within a given price range. For the searching Knockout JS is utilised to create a seamless user experience, by using Pure Computed Observables the page can be searched and filtered without having to refresh the page.

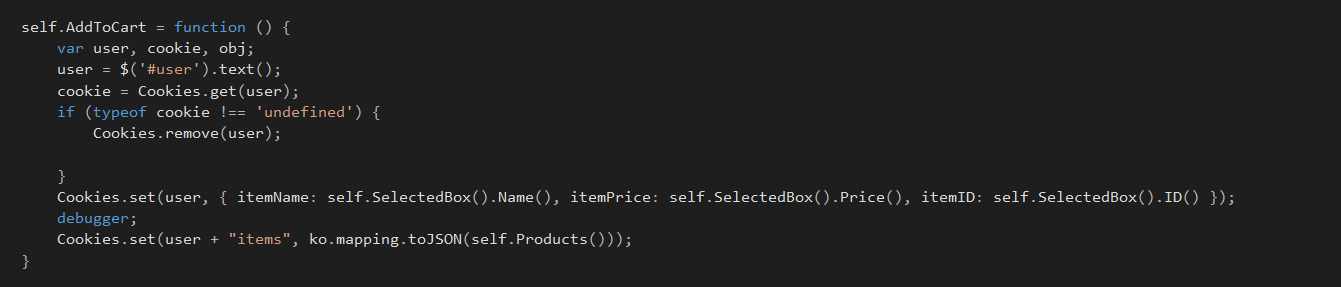




## 

## Purchase an available selection box, specifying a recipient and optional message for them

When a user wants to order a box, the selected boxes details are added to a cookie associated with the user currently logged in. Then on the cart page the selected box ID is retrieved from the cookie so that the correct box details can be retrieved from the server. Originally all details were pulled from the cookie and no call was made to the server but this allowed a malicious user to be able to modify the cost of the box from the cookie. Once on the cart page the user must enter a recipient and has the option to enter a message for the recipient.



## 

## Only take payment for a selection box if its components have been successfully acquired

Because we cache the data for items it means that when a customer views a box then the information on stock levels might not be correct. Therefore, when the customer submits an order we order each individual item from the supplier first before submitting the box order. If a negative result comes back from the orders then no payment is taken.

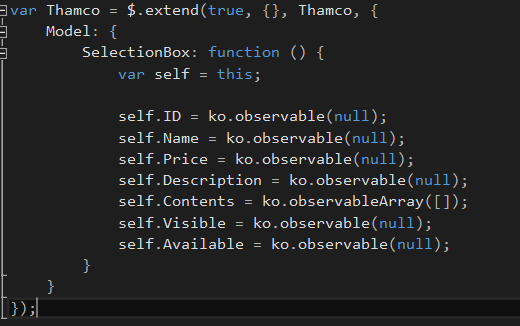


# Software Development Framework and Tools Used

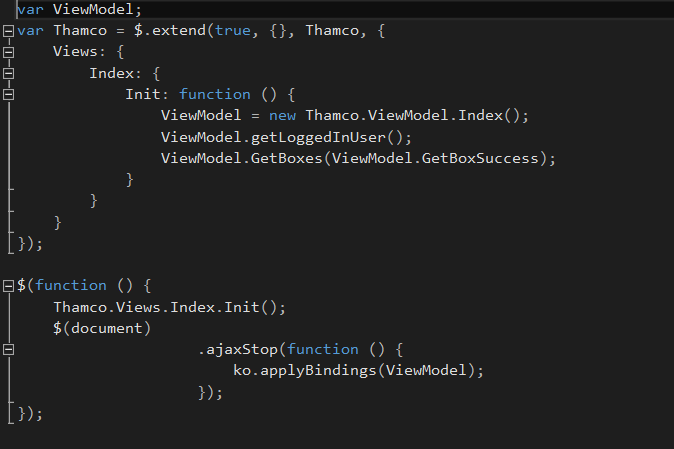
## JavaScript

### Knockout JS

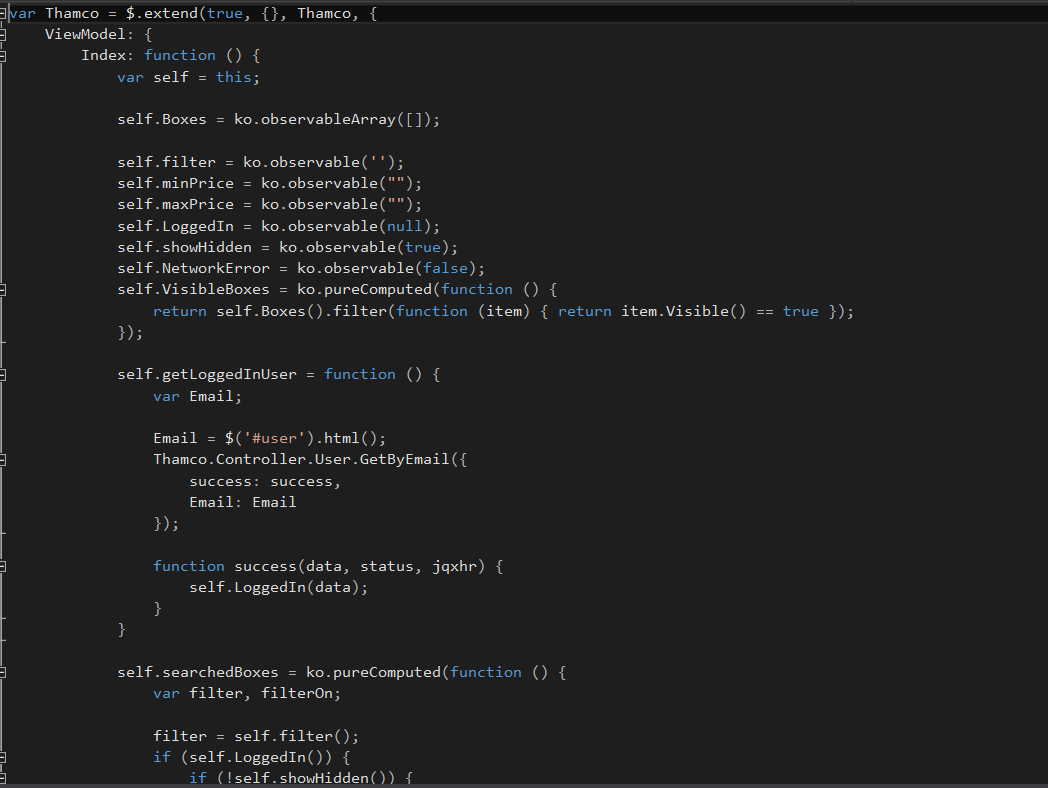
The entirety of the web application uses knockout JS to help create the Model View ViewModel (MVVM) structure. It also uses two-way databinding to seamlessly update both the view displayed to the user and the viewmodel behind it. All of the models are objects where each attribute is a knockout observable. The views are used simply to create and initialise the view models. The viewmodels is where all the logic is, it stores the data that is used to create the page, it also handles the result of clicking buttons and also the searching of a collection of items.



Model Example



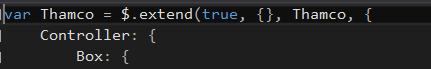
View Example



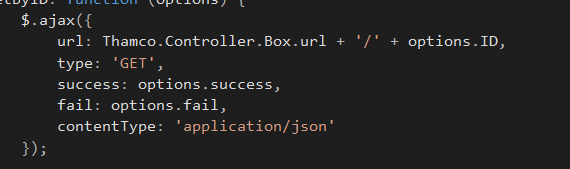
ViewModel Example

## jQuery

jQuery is a library built on JavaScript which helps with DOM manipulation and helps make the JavaScript code browser agnostic. The main use for it in my solution is to create a name space which allows code with different functionality e.g ViewModels, Models, Controllers etc. to be separated into their own file but still be called from every other file. This is done with the $.extend function which merges multiple objects into one object, so there is a global Thamco object and as files are added it is built up with extra properties. jQuery is also used in the controllers to create asynchronous JavaScript and XML (AJAX) calls. The $.ajax method takes an object with a number of properties such as the url, data and a success callback function. This is used to communicate with the web API controllers.



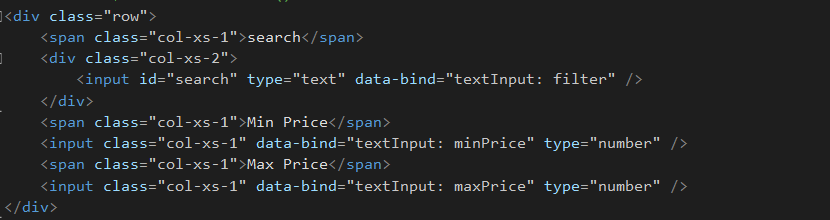
Example of the namespace being built up



Example of an AJAX call in jQuery

## Bootstrap

Bootstrap is a mobile first, front end framework for helping structure the layout of a page, it uses both CSS and Javascript. Unlike Knockout, it also has a dependency on jQuery being included. It is used in most of the application to help with the spacing and layout between textboxes and labels. Although it is a mobile first framework, the application is not designed to be used on a mobile device.

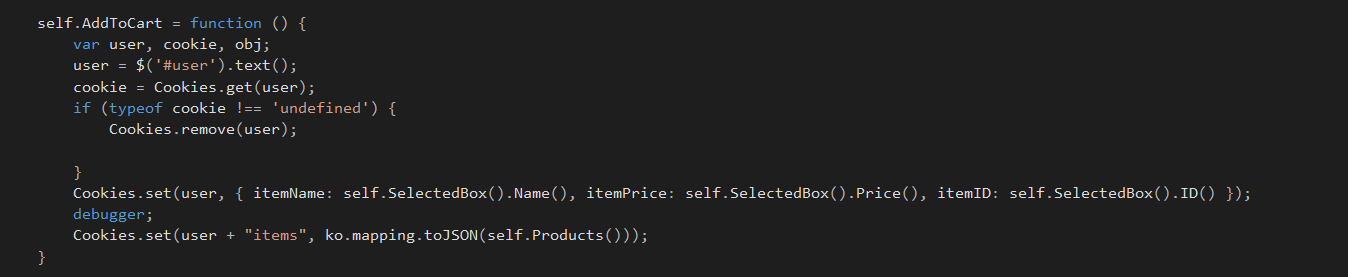
code used to create the search bar on the home page



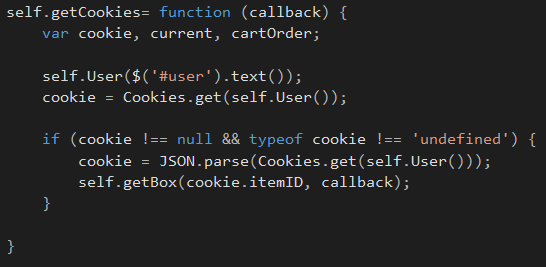
The result of using Bootstrap

### JS Cookie

JS Cookie is a small JavaScript library included to help make the usage of cookies easier. Cookies are used in this application to store the details of a box when added to the cart and also when creating the order details when going from the card to the payment page



Creating a cookie



Reading from a cookie

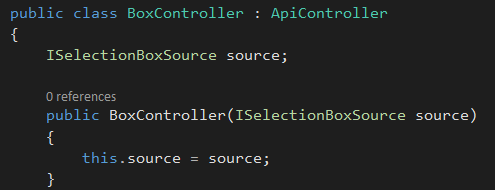
## .NET

### Simple Injector

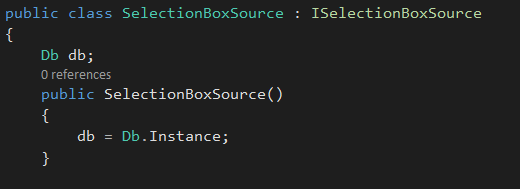
Simple injector is used for dependency injection. For this application it is used to allow the usage of mocks during development and then a seamless transition to the actual class libraries that will be used in production. Both the mocks and actual class libraries should implement the same interface and any place where the mocks would be used should be replaced with the interface class.



Configuration of Simple Injector in the Global.asax.cs file



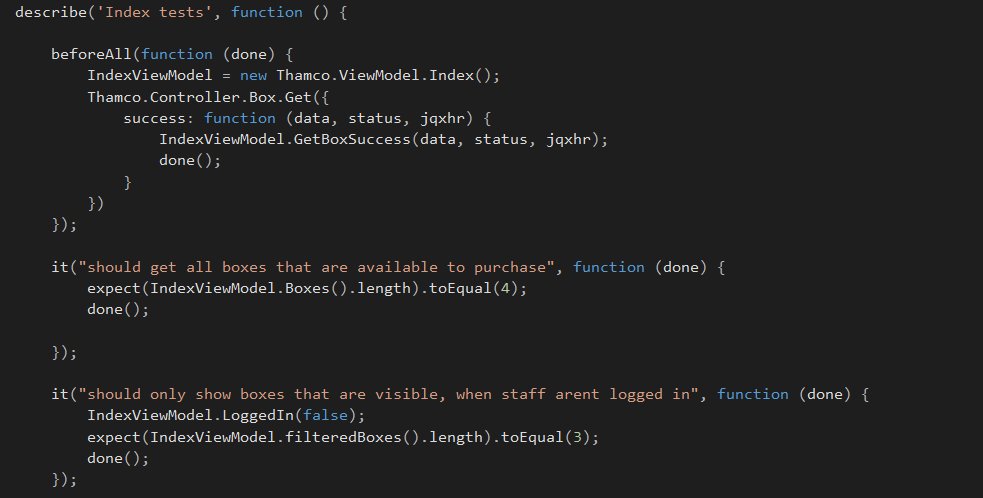
Usage of Simple Injector within a web API controller



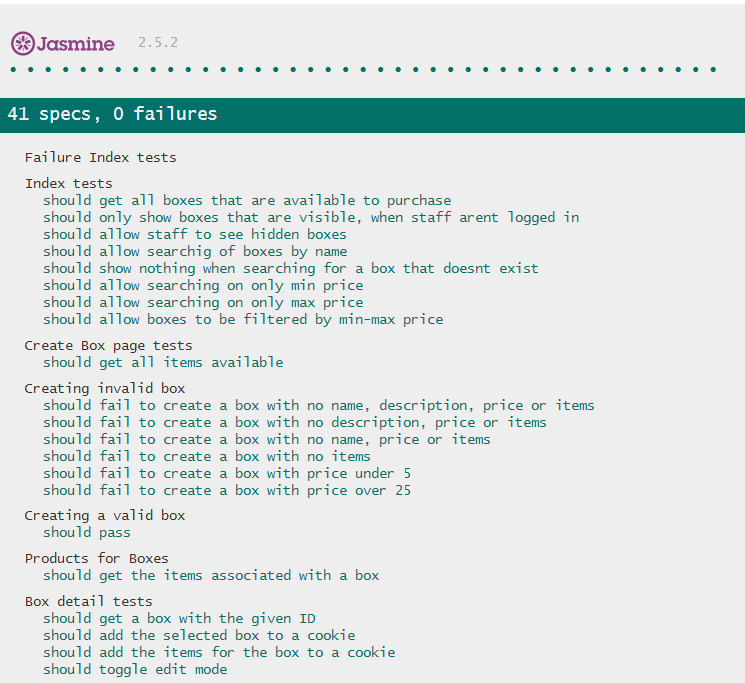
Mock class implementing the interface for dependency injection

# Testing Performed

As all of my logic is in the Knockout ViewModels I couldn’t use Visual Studio’s built in unit testing tools so I had to choose a JavaScript library that would allow me to thoroughly test the application. In the end I decided to use Jasmine as it allowed me to test my Knockout functions as well as the results of my AJAX calls. It is also the testing framework used by Knockout themselves (<http://knockoutjs.com/spec/runner.html>). An extensive number of tests are carried out such as making sure the application responds correctly when correct information is entered as well as the handling of invalid or missing data. The only current limitation is that for all tests to pass a user needs to be logged in otherwise some will fail. Navigate to /testing to view the tests running



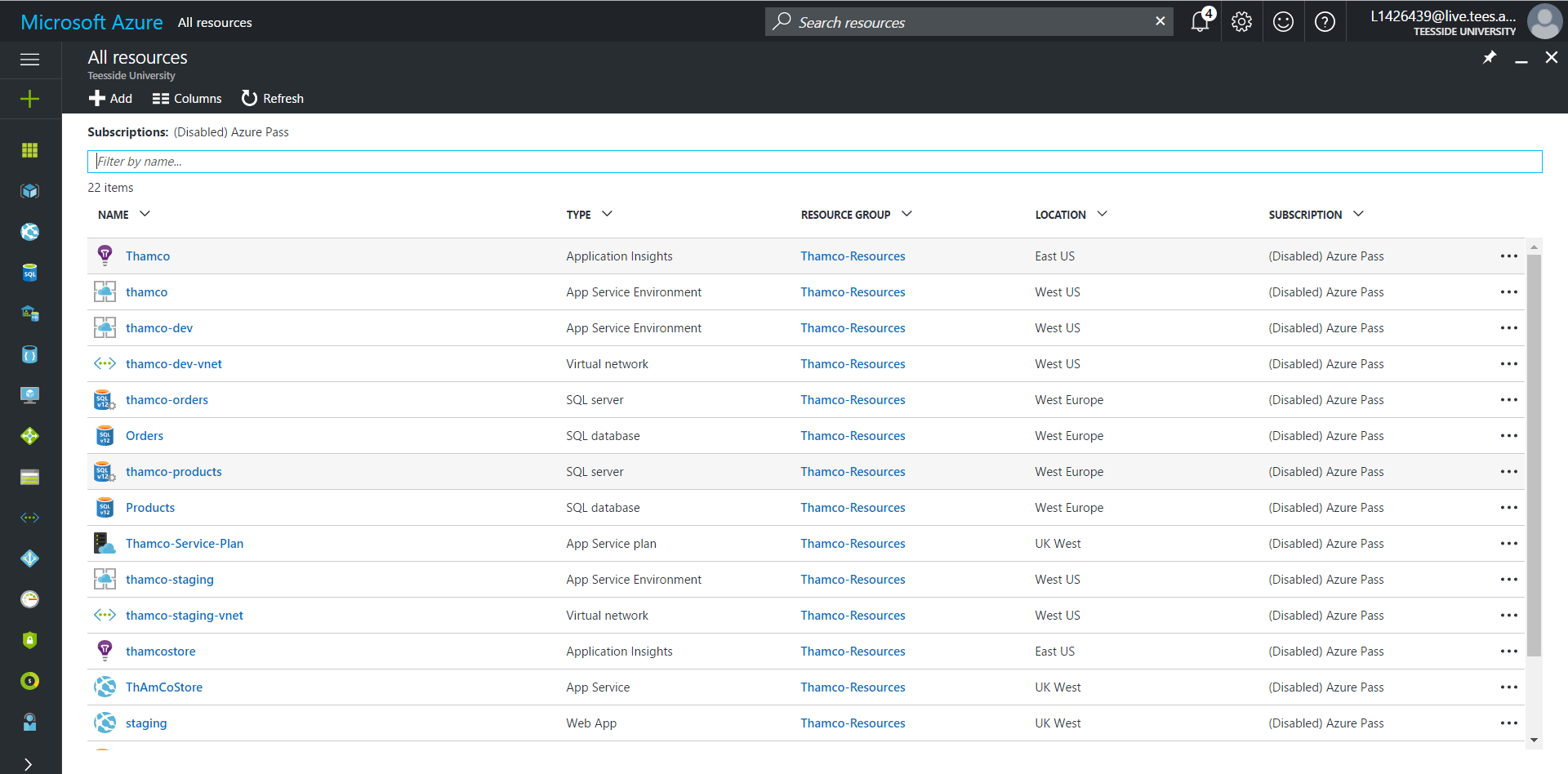
Example of code behind tests



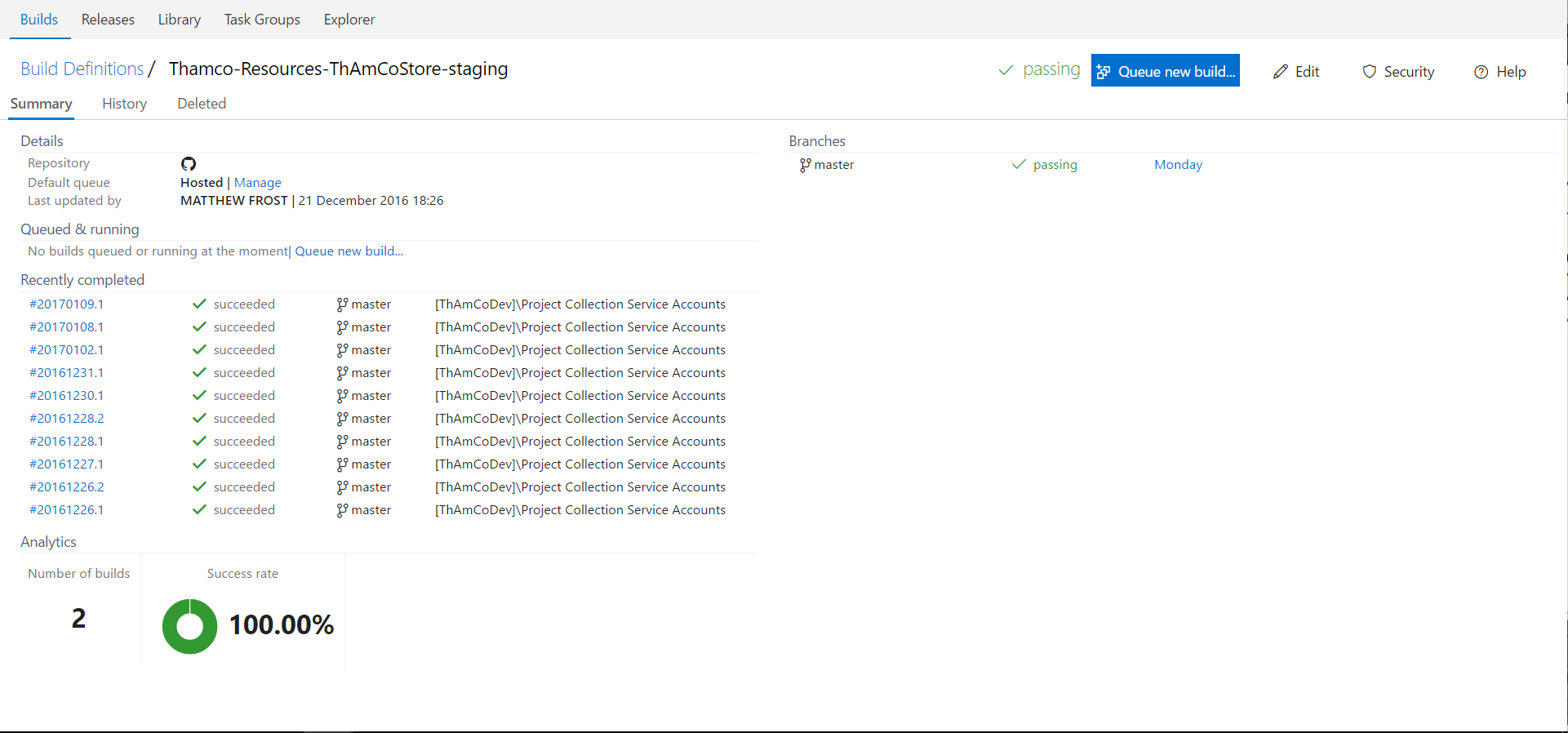
Tests being run

# Deployment

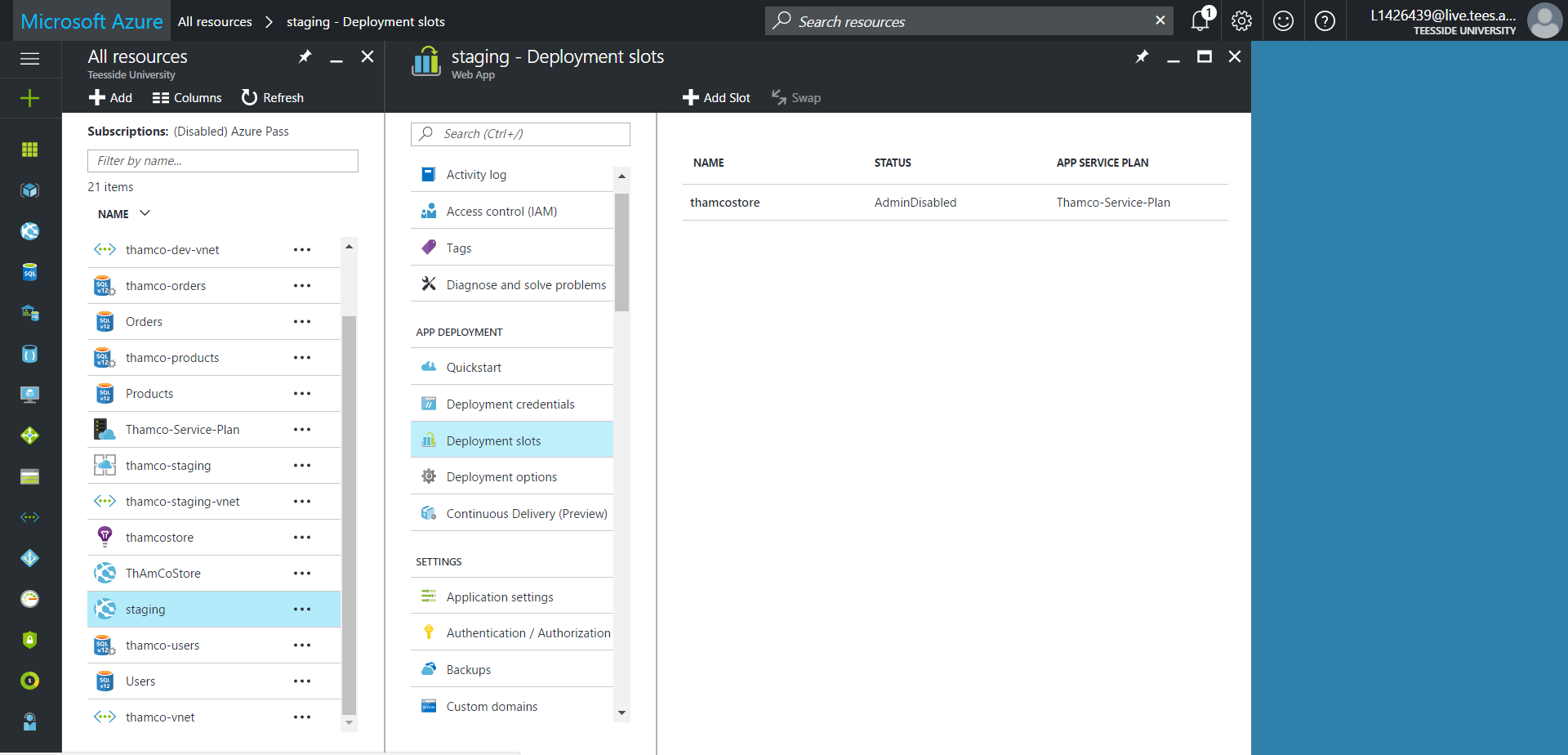
The Thamco Store Web application is hosted on Azure, there are two versions of the application a production version and a staging version. By having two different versions it means that changes can be tested and verified on the staging application before being deployed to the production version. This could help minimise downtime of the application as there would be less chance for bugs and errors to affect the production store. It would also be possible to have the staging application connect to a different database than the production one. This would allow thorough testing to be carried out as the production data would not be affected.



The web application also uses continuous integration for successful deployment to the staging environment. This allows development and local testing to be carried out on the developers machine. Then once it is pushed to the code repository it is built on the server to make sure everything compiles then it is deployed to the staging environment.



Once the deployment has been successfully built and tested on the staging environment it can be swapped with the version on thr production environment through Azure.



Production website: <http://thamcostore.azurewebsites.net/>

Staging website: <http://thamcostore-staging.azurewebsites.net/>

# References

Knockout JS - <http://knockoutjs.com/>

jQuery - <http://jquery.com/>

Bootstrap - <http://getbootstrap.com/>

JS Cookie - <https://github.com/js-cookie/js-cookie>

Simple Injector - <https://simpleinjector.org/index.html>

Jasmine - <https://jasmine.github.io/>