TEESSIDE UNIVERSITY

SCHOOL OF COMPUTING

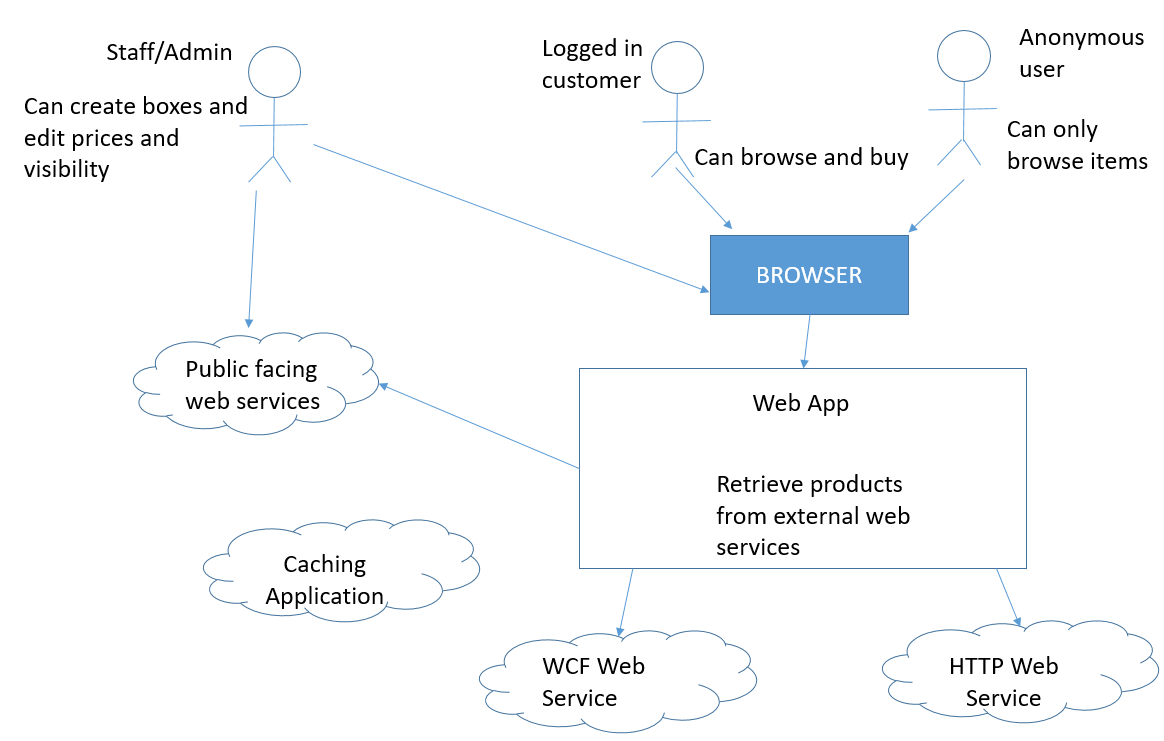
**SOFTWARE ARCHITECTURE (COM3041-N) TEAM DOCUMENTATION**

Team Name: **E**

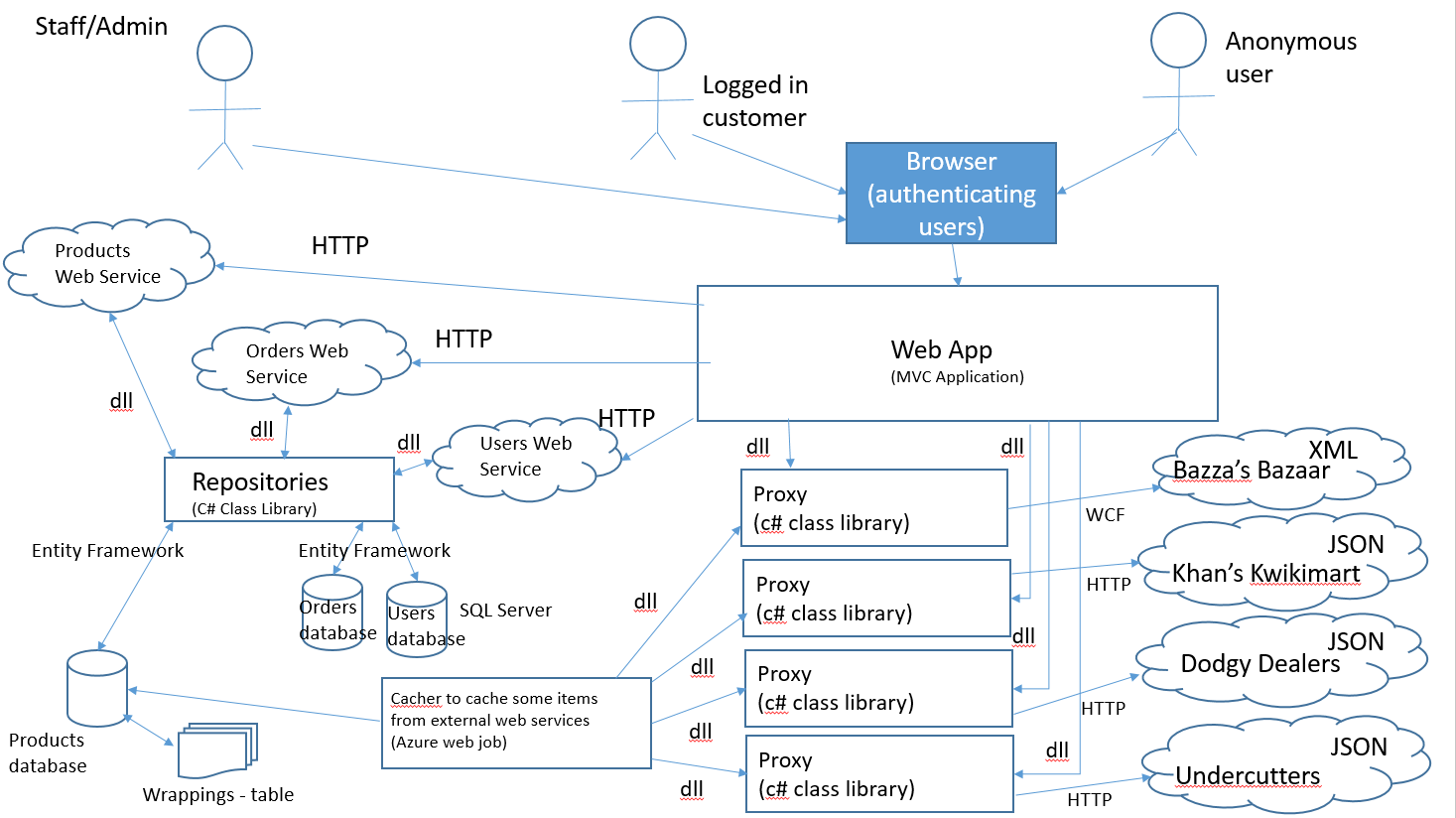
Diagrams with appropriate notes and justification to evidence that …

* *The architecture (components and their technology, interfaces, connectivity, protocols, and deployment within the infrastructure) has been documented clearly and any deviations in the implementation identified.*
* *The specifications of the components are fit-for-purpose and fit-for-use.*
* *The architectural style is reasonably clear and it appears plausible, maintainable and scalable for the future.*
* *Examples are given of decisions faced and choices taken, and the rationales for them are valid.*

Our solution uses a service orientated architecture, splitting the separate components into separate solutions that can be individually deployed and scaled when needed. Each component of the system is intended to be deployed to Azure for hosting. Since each component can be run without relying on every other component, if one component were to go down then the entire system would not be affected.



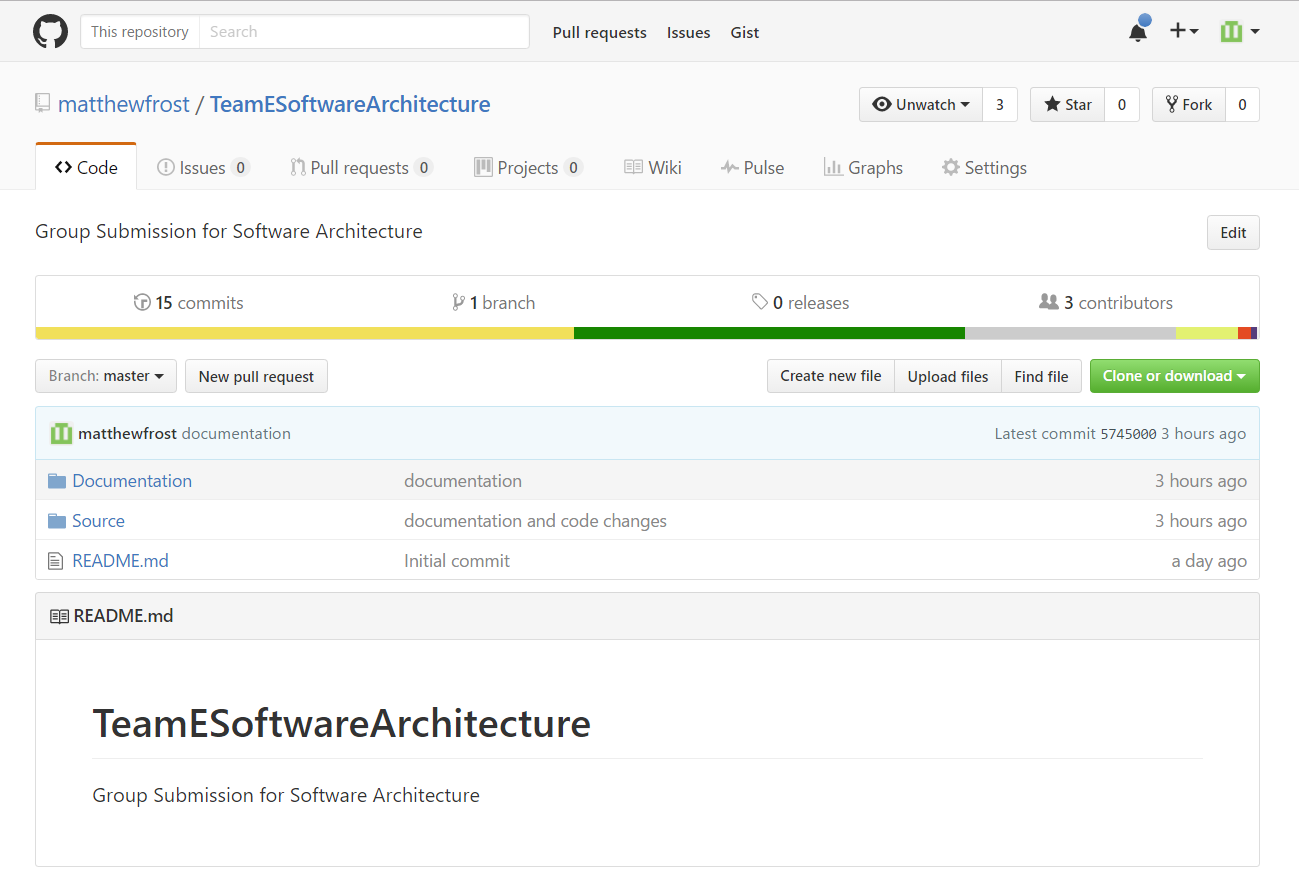
Top Level Architecture



Tier 2 Architecture

//Talk about Tier 2

During the development of the applications we decided to use Git with GitHub as our source control solution of choice. Although we could have also used something like Subversion, no one in our group were familiar with Subversion so Git was the most obvious choice. <https://github.com/matthewfrost/TeamESoftwareArchitecture>



Our repository on GitHub

The caching application was created as we found that some of the web services provided weren’t very stable and could result in a significant decrease in performance and reliability if accessed directly. The caching application is run as a web job on Azure every hour, this makes sure that any data displayed is an hour old at the most. Initially we were just going to cache the products from the web services that caused an issue, but we then realised that it would be difficult to create a relationship between boxes and the products inside them if we were not storing all of the products.

The repository pattern is implemented to provide a layer of abstraction between the data access layer and the business logic. It separates the code that maps data from the database to DTOs and the code that is used to implement business logic such as which boxes should be displayed. By having all of the logic in the repositories it keeps the web service controllers that use them, small in size. Furthermore, if any logic needs to be changed then only the repository needs to be redeployed rather than all of the applications that use the same logic.

The Proxy pattern is used to interface with the external web services such as Dodgy Dealers, Undercutters, KhansKwikiMart and Bazzas Bazaar. This allows us to take the data from those web services and convert them into a format that is suitable for our application and able to be stored in our databases.

The Web application is an MVVM application, it uses Web API controllers to interact with the necessary web services and send information back to the client side application. The application uses JavaScript and HTML as its client side languages, with Knockout JS and jQuery libraries to help display the data and modify the page. Knockout was used as it also helps make a better user experience, for example it allows for searching and filtering without having to retrieve the results from the server. We chose to make the web application essentially a “dumb” application with the majority of the business logic extracted out into other services such as the repositories and proxies. This means that minor changes to business logic will not mean a rebuild and redeployment of the application. For example, if we decided to only sell boxes with a 15% profit margin then only the repositories would be changed.