# Assignment 2 – Grocery shopping system Application

## Dependencies:

### Netbeans - Webapplication

### Maven Json – 20180813 <https://mvnrepository.com/artifact/org.json/json/20180813>

### Android studio

* Gson - com.google.code.gson:gson:2.2.4 (Added via build.gradle implementation)

## Application description:

My android application is based off my JSF Grocery shopping system. There is a total of four different RESTful services, each including different operations. Each service once logged into the application requires Authentication via a Basic Auth header.

**Services:**

* API
  + User
    - GET
    - Register
      * POST
  + Product
    - {ID}
      * GET
    - ALL
      * GET
      * {Search}
        + GET
        + Store

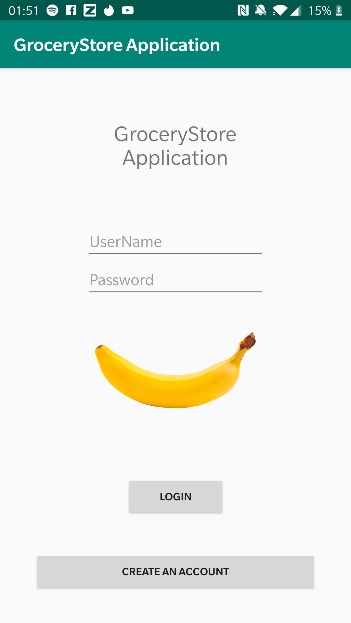
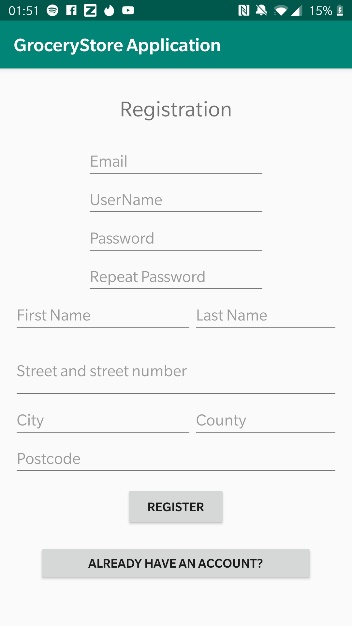
{ID}

GET

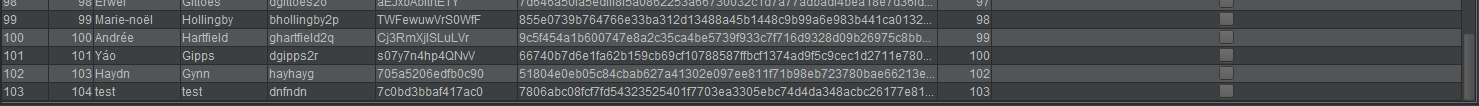
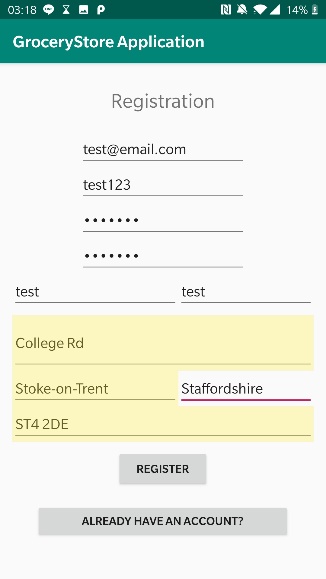
* + Order
    - Place
      * POST
  + Cart
    - Increment
      * {ID}
        + PUT
    - Decrement
      * {ID}
        + PUT
    - GET

## Login/Registration:

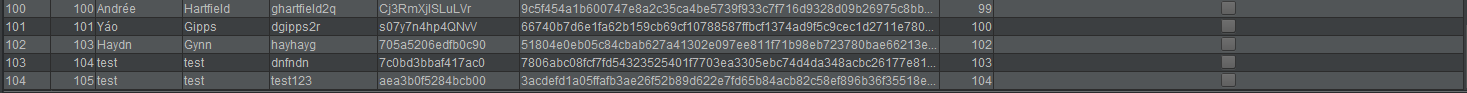
The app is launched to the login screen. Your username and password is required to login. Logging in uses the /User/Get service, it sends the Basic auth string via the header and returns the users details if correct. If incorrect an authentication error is thrown.



To register you need to fill in all of the details. The password needs to be longer than six characters, and they need to match. The username must also be unique. Clicking register will then hash the password and insert the user into the database.

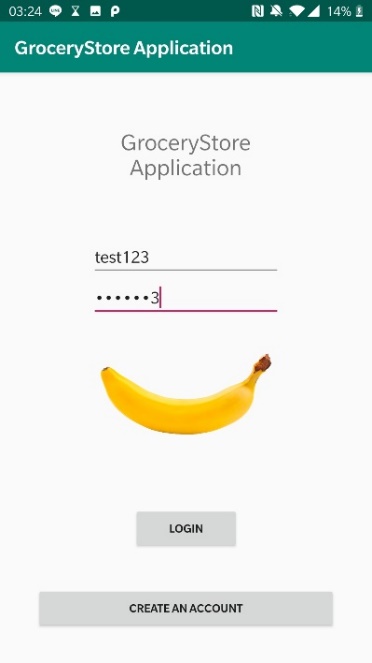


Username: test123



Password: test123

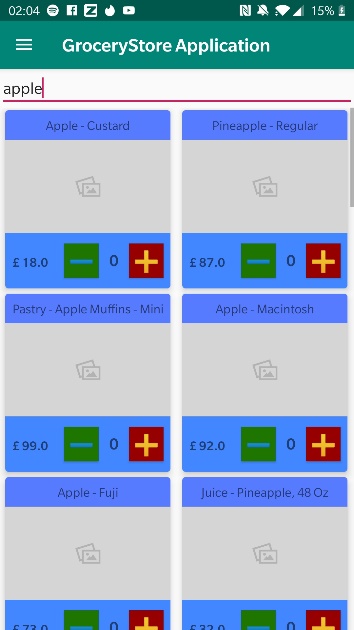
Logging in with our newly created account will take us to the products page. This will select the user from the database if they exist and the password is correct. Once logged in each subsequent request will pass the users information via a Basic auth header.



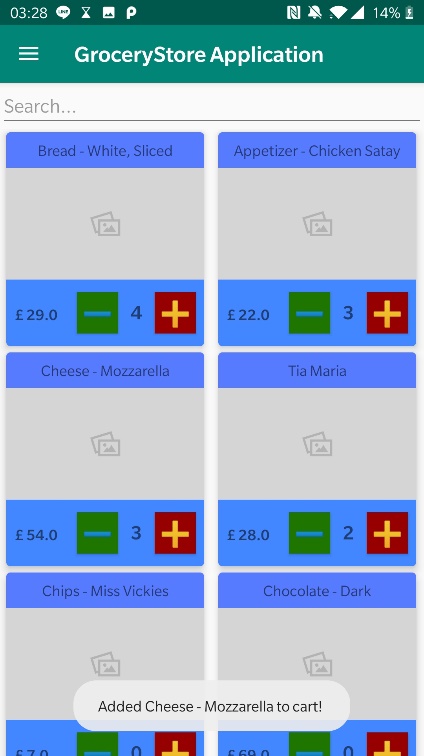
## Products page:

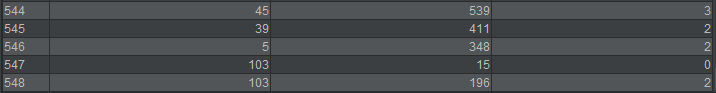
The product page allows you to search for products, and increment and decrement the amount in the cart.

When typing it triggers the onTextChanged listener. This then triggers the get products web request, the search bar text is passed as a URL parameter. The response of the request is a JsonArray of the products, this array is then rendered using a recycler view with a custom ViewAdapter and ViewHolder.



The product activity only currently returns maximum 30 products and has no page feature. You can scroll down to view all of the products on the page.

Clicking the positive or negative button will add or remove the product from the cart or increment and decrement if the product is already in the cart. This makes use of the cart service, making insert and update transactions.

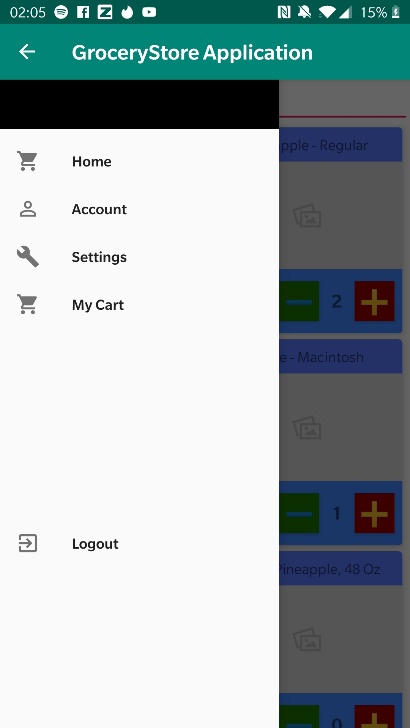
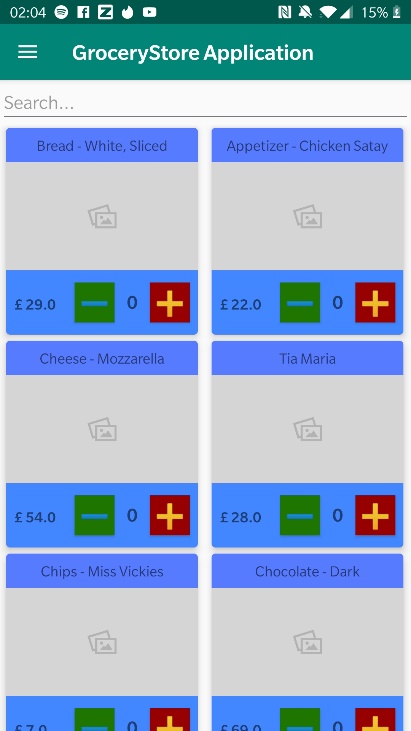
Shopping cart table



The product entry gets created and incremented if already exists

## Navigation

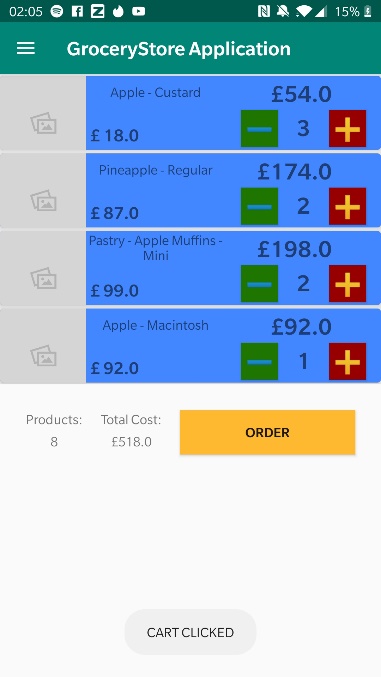
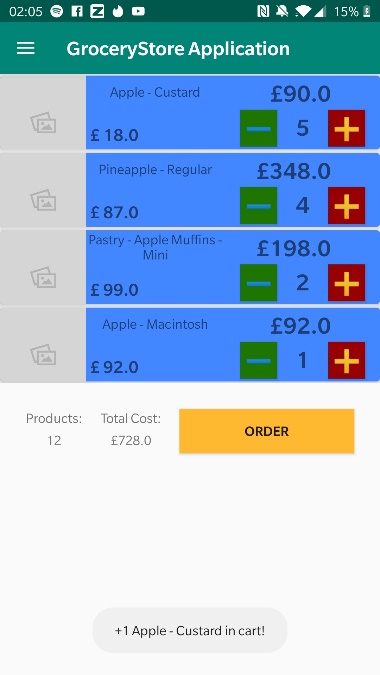
To navigate throughout my application I make use of an ActionDrawerBar.



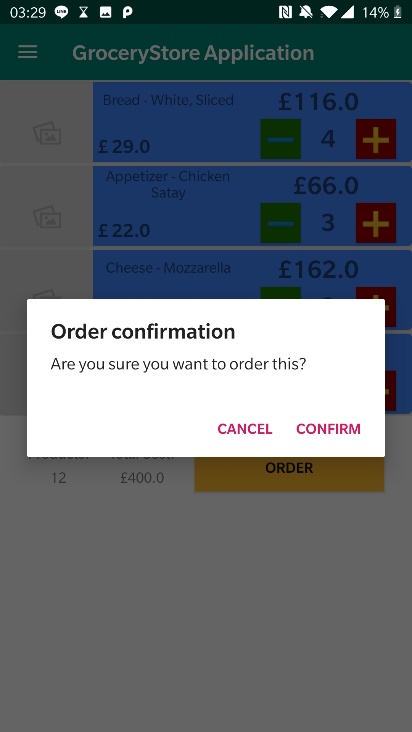
The navigation screen is opened by clicking in the top left or swiping from left to right. Here you can access the cart page by clicking on it. Clicking logout will take you back to the login screen.

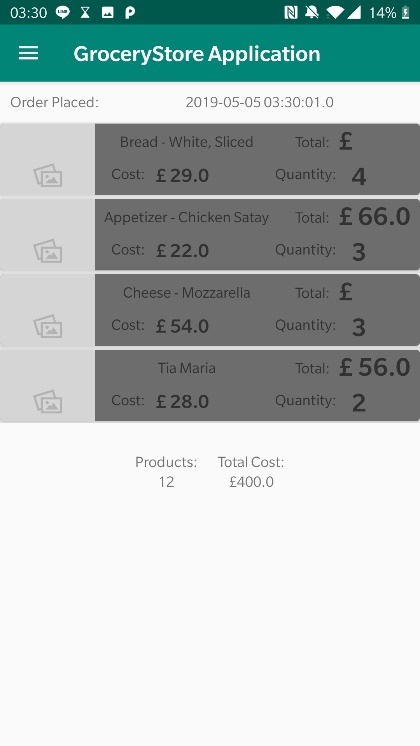
## Shopping cart:

On page load the shopping cart activity calls the get shopping cart service. This returns an array of products which also gets rendered in a recycler view using the same ViewAdapter and ViewHolder.



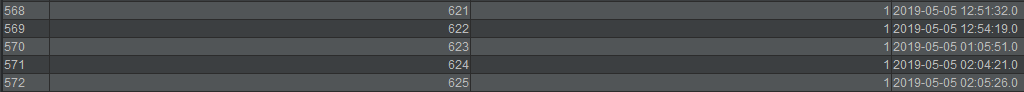
On the shopping cart page, you can still increment/decrement products from your cart.

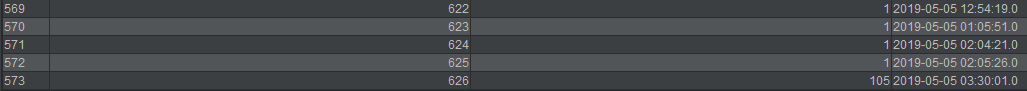
Clicking the order button will first ask for confirmation.



After confirming the order, your cart gets cleared and the order is placed. You will then get taken to a summary/receipt page of your order. The order is retrieved via web request and again rendered using previous methods. Orders cannot currently be accessed by any other method.

Order:





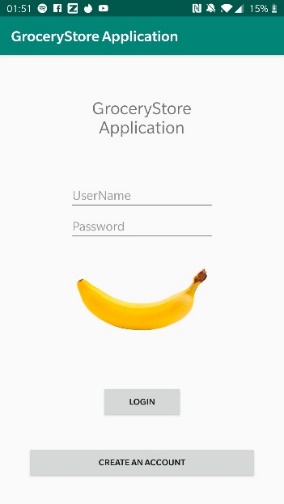
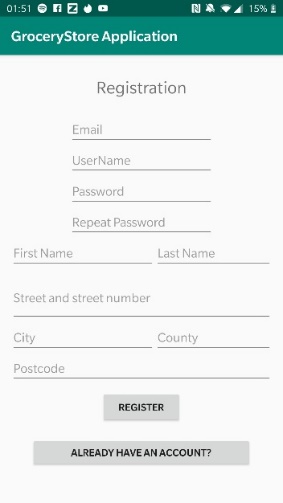
Order products

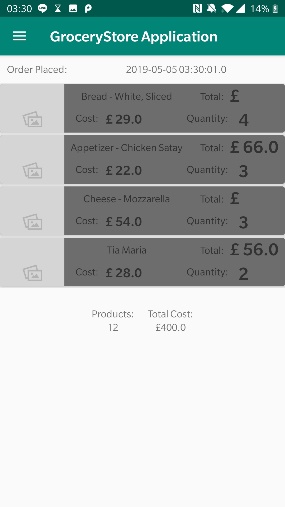


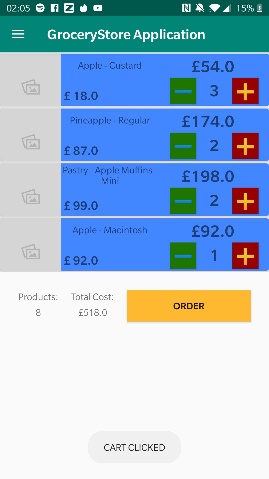
## Marking rubrics:

1.

*“>=4 screens>=4 layout xml and wide range of appropriate widgets used; Excellent use of menus; Excellent range of functionality that works flawlessly (i.e.no bugs)”*

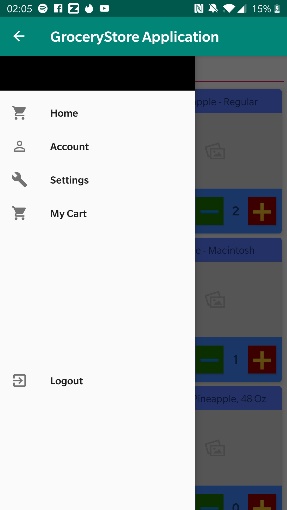
MainActivity – activity\_main.xml RegisterActivity – activity\_register.xml HomeActivity – activity\_home.xml

CartActivity – activity-cart.xml OrderActivity – activity\_order.xml Cart\_product.xml



 Product.xml

 Order\_product.xml



Features used:

* Card view
* Image
* Text
* Linear layout
* Constraint layout
* Relative layout
* Button
* Nav\_view
* Scroll view
* Recycler view

**Grade estimate: 85%**

Well linked activities working together flawlessly making use of great and simple UI design. The use of custom ViewHolders and ViewAdapters really allow me to control how certain parts of my application is rendered. User data is passed between activities to allow the application to flow effectively

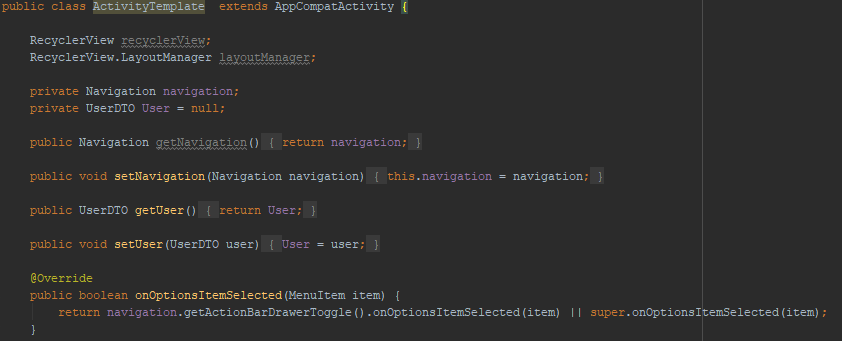
2.

*‘Excellent code structure, adhering to sound SE principles, and robust error handling’*

Effectively used an object-oriented approach to prevent code duplication and to improve readability.

Because navigation will be required on most activities, I chose an object-oriented approach to simplify the setup of navigation for each activity.

I created a template class of some common parts of each activity.

I put all of the navigation setup inside an object. This way I can simple setup the navigation by calling ‘setNavigation’



Each web service uses its own gateway, each having a onUrlReaderFinished function.

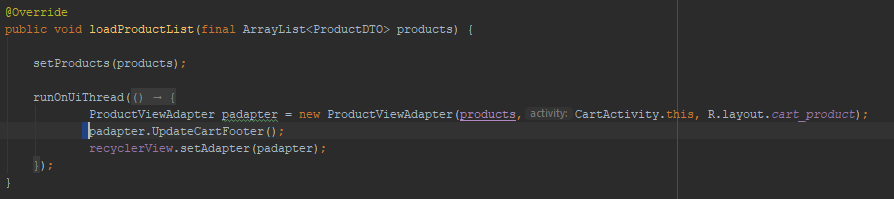
This is the GetCart Service. Each service makes use of the service controller. This handles all of the details of the web request. When executing a web request, there is some parameters you can specify.

Webrequest URL, If authentication is needed, the type of request and the authentication information If auth is specified.

The service controller uses an AsyncTask.

When rendering product arrays, I use a recycler view. I use a custom view adapter and view holder to control how it looks and how the data is displayed.

When loading the list of products I use the UIThread.



To transfer and control data throughout my application I use DTOs.

* AddressDTO
* OrderDTO
* PasswordDTO
* ProductDTO
* StoreDTO
* UserDTO

I also have a DTOConverter class, this is to convert Json to the corresponding DTO.

**Grade estimate: 90%**

I believe my application was designed efficiently, with little code duplication. Good Object oriented approaches were used. Null checks and try/catches were used correctly to allow the application to flow without any errors.

3.

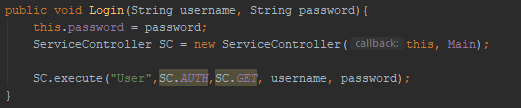
*‘Create and use a web service that retrieves, updates, deletes and inserts data records in the server-side database’*

My service controller oversees all operations involving web services.

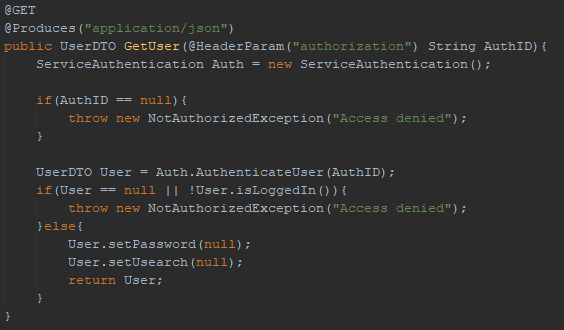
I have four main webservices, each with sub urls and functions.

**Login - Retrieves**

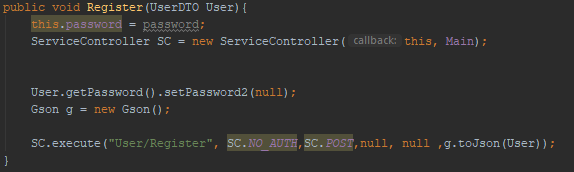
Selects the user if the credentials are correct.

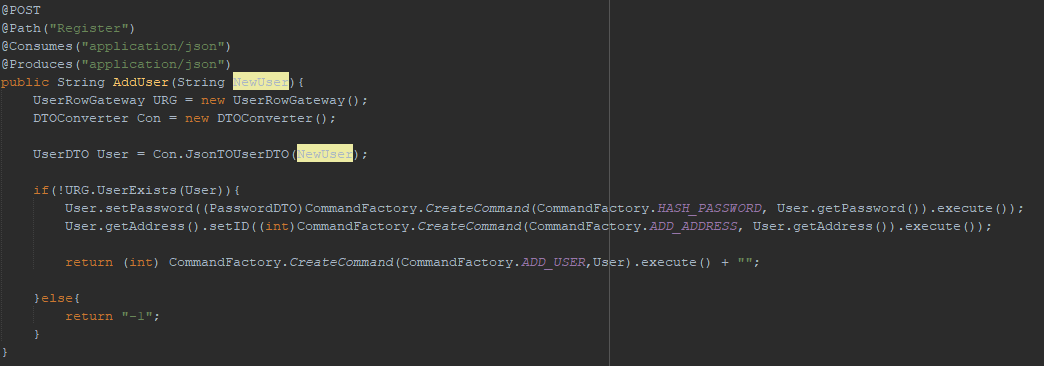
****

Webserver:

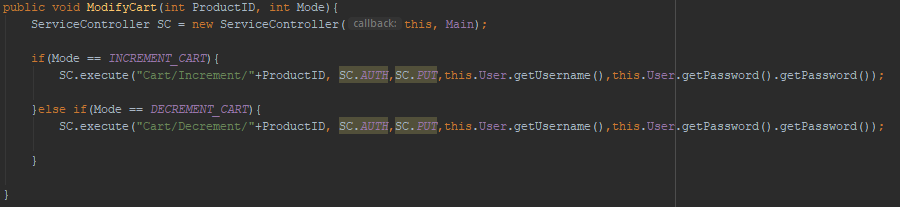


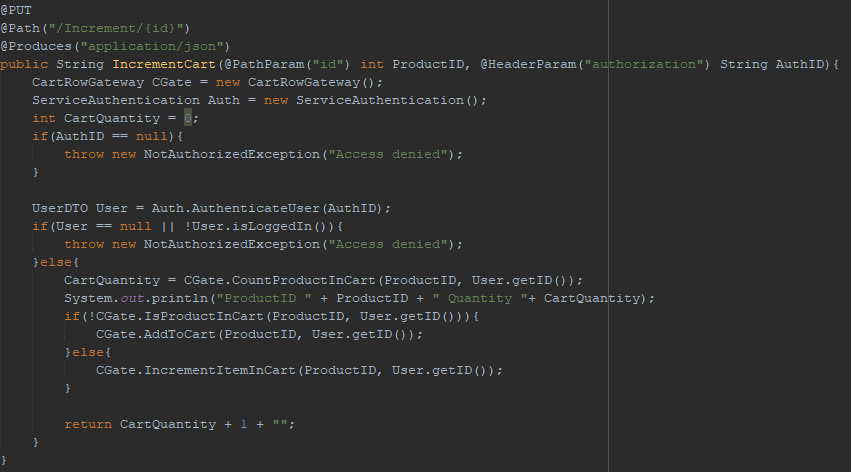
**Register – Inserts**

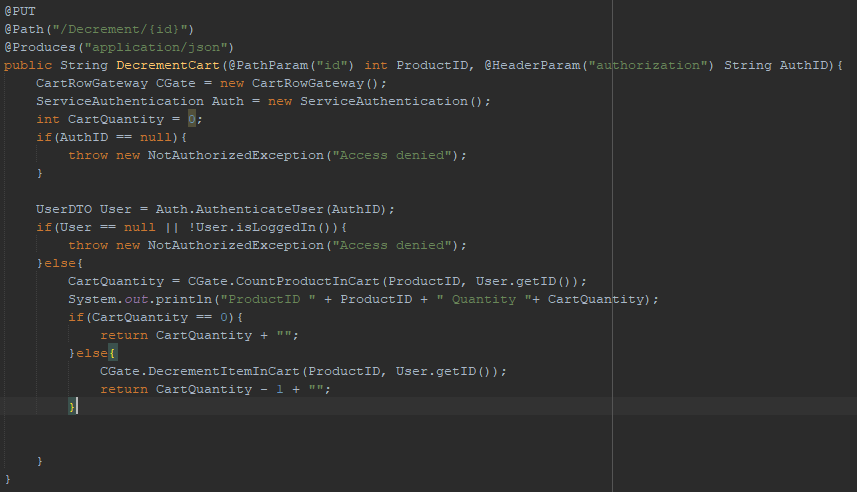
****Convert a formed UserDTO to json and send a post request off to create the user.

Web server:

**Cart – Update & Delete**

****Increment or decrement a product in cart with a given ID

Webserver:



**Grade estimate: 95%**

Lots more webservices are also used. Get cart, Count product in cart, Place order, Get product, Get products. My application works effectively with my webserver, making use of authentication and different types of requests.

## Conclusion:

The hardest part was initially setting up the capabilities for web request. I initially had some errors when my app was attempting to connect to my web server, such as data types and request types. Eventually after research and testing, I was able to fix these issues.

When sending off the web request, I had some issues forming Json out of my DTOs. In the end I opted for googles Gson Json library, this cut out a lot of time and helped my application greatly.

Once the ServiceController was setup it was very easy from there to add new web services.

Next time I think I will aim to cut down on how often im making web requests. Bandwidth can be saved by bundling up data that can be transferred together.