**MSc. Software Engineering**

**High Level Design Document**

**Cloud Development Frameworks Assignment 2**

**Jimmy Collins – R00145569**

Contents

[Section 1 - Introduction 2](#_Toc482642601)

[Section 2 - Microservices Overview 3](#_Toc482642602)

[Cart Service 3](#_Toc482642603)

[Purpose 3](#_Toc482642604)

[Functionality Provided 3](#_Toc482642605)

[Catalogue Service 5](#_Toc482642606)

[Purpose 5](#_Toc482642607)

[Functionality Provided 5](#_Toc482642608)

[User Service 7](#_Toc482642609)

[Purpose 7](#_Toc482642610)

[Functionality Provided 7](#_Toc482642611)

[Stock Administration Service 8](#_Toc482642612)

[Purpose 8](#_Toc482642613)

[Functionality Provided 8](#_Toc482642614)

[Order Service 9](#_Toc482642615)

[Purpose 9](#_Toc482642616)

[Functionality Provided 9](#_Toc482642617)

[Front-end Service 11](#_Toc482642618)

[Purpose 11](#_Toc482642619)

[Functionality Provided 11](#_Toc482642620)

[Section 3 – Database Design 12](#_Toc482642621)

[Products Table 12](#_Toc482642622)

[Users Table 13](#_Toc482642623)

[Orders Table 14](#_Toc482642624)

[OrderDetails Table 15](#_Toc482642625)

# Section 1 - Introduction

This document outlines the design of the ‘*Super Carz Inc.*’ site that that I decided to implement for my final project. The code base is based on the Assignment 1 code base, with the necessary improvements and additions required to make the application functionally complete, and some user interface changes to improve the aesthetics of the site.

The main purpose of the site I have implemented is to allow users to view and purchase high-end cars. It is comprised of a collection of services written in Node.js and using the Express[[1]](#footnote-1) framework, each of which is outlined in-depth in Section 2 of this document.

The application is currently hosted on IBM Bluemix and is available at <https://shop-jc.mybluemix.net/> for viewing[[2]](#footnote-2).

To use the site at the above link, the following *Administrator* and *Non-Administrator* accounts can be used. The Administrator account can access functions like Stock Management etc.

**Administrator**

* Username: jim
* Password: jim

**Non-Administrator**

* Username: joe
* Password: joe

Note that I have also compiled a developer journal for this project that outlines my efforts in the development and deployment of the project.

This will be submitted along with the full codebase before the May 22nd deadline.

# Section 2 - Microservices Overview

This section outlines the operation of each of the services that comprise the ‘*Super Carz*’ application.

There are six in total:

* Cart Service
* Catalogue Service
* User Service
* Stock Service
* Order Service
* Front-end Service

## Cart Service

### Purpose

The purpose of the cart service is to provide an in-memory storage area for items that a customer has added to their cart on the site which they may wish to purchase later when they use the checkout functionality.

The cart is mainly used by the front-end service to provide this functionality to users.

### Functionality Provided

The cart service provides three API’s, details of which are outlined below.

**HTTP POST**

#### /add

This API provides the ability to add a specified quantity of an item to a customer’s cart. The item is identified by a unique product ID. If an item with this product ID already exists in the cart, then the items will be combined. As well as each item being identified by a product ID in the cart, the cart itself is also uniquely identified by a cart ID.

#### Example JSON Payload



**HTTP GET**

#### /cart/:custId/items/

This API provides the ability to retrieve the current contents of a specific customer’s cart as JSON.

#### Example JSON Response



**HTTP DELETE**

#### /cart/:custId/items/:id

This API provides the ability to delete an item from the customer’s cart.

For example, calling into /cart/5/items/1 would delete item 1 in the cart of the customer with the customer ID of 5.

## Catalogue Service

### Purpose

The catalogue service handles interactions with the backend MySQL database where details of the available products are stored (*high-end cars in the example of the site that I have developed based on the Assignment 1 Codebase*).

The functionality within the catalogue service is used by both customer facing features and some of the administrator-only features.

### Functionality Provided

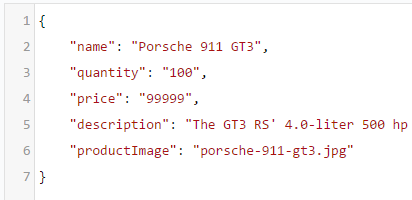
The following functionality is provided within the catalogue service.

**HTTP POST**

#### /newProduct

This API provides the ability to add new products to the catalogue. This area from which this functionality is available in the front-end service is restricted to administrator users.

#### Example JSON Payload



#### /deleteProduct

This completely deletes a product from the product catalog. It is an irreversible action that deletes the relevant row from the products table in the database.

#### /deactivateProduct

This provides the ability to deactivate a product in the product catalog. This does not delete a product, but makes it inactive in the products table in the database such that it will no longer appear for sale on the site. This function is only accessible from the Stock Management user interface in the Administration Panel, and can be reversed using the reactivateProduct API (see below).

#### /reactivateProduct

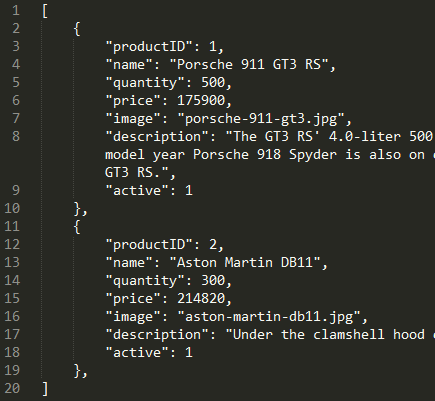
This is the reverse of the above, and can be used to reactivate a product on the site and make it available for sale again. This is also only accessible from the Stock Management user interface in the Administration Panel.

**HTTP GET**

#### /getProducts

This API provides the ability to get all the active products in the catalog. This is used by the front-end service to show the available products to users.

#### Example JSON Response



## User Service

### Purpose

The user service is responsible for user management with regards to user registration and handling requests to login to the application.

### Functionality Provided

This service provides the following API’s that are used by the front-end service for user registration and user login validation.

**HTTP POST**

#### /register

The registration user interface on the front-end calls into this API in order to add new users to the users table in the database. All new users are added as *Customer* users, but any user can be made an *Administrator* later in the database by changing the usertype.

The registration functionality will return errors in the event of a user already existing in the database (based on the username), or if any MySQL errors occur.

#### /login

This API is invoked from the front-end service and is responsible for validating that user credentials are correct when a user attempts to login to the application. If the credentials are not valid, then an error is returned to the front-end service which displays a message to the user.

If successful, a number of cookies will be created on the client which are used by the application for different purposes.

## Stock Administration Service

### Purpose

The purpose of this service is to provide a set of API’s that can be used (via the front-end service user interface) to administer stock levels of products within the application.

### Functionality Provided

The following functionality is available in this service.

**HTTP POST**

#### /incrementStock

The ability to increment the stock levels of a particular item from the catalogue. This would be used for example in the event of new stock being purchased by the business.

#### Example JSON Payload



#### /decrementStock

The ability to decrement stock levels for a particular item from the catalogue. This would be used in the event of a customer making a purchase of this item, or stock levels being reduced for some other reason.

#### Example JSON Payload

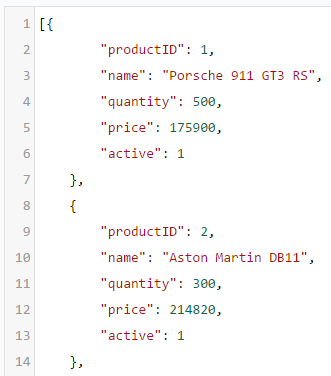


**HTTP GET**

#### /currentStock

This provides the ability to get the current stock levels for each product. This is used in the Stock Management user interface in the administration panel to show an overview of the stock levels and the value of the stock for each item in the catalogue.

#### Example Response



## Order Service

### Purpose

This service is used to process user orders. It is quite simplistic currently in its implementation in that it does not handle payments, just the persisting of orders in the MySQL database, and the retrieval of information about these orders.

### Functionality Provided

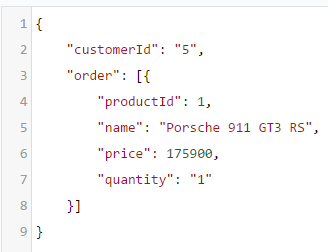
The following functionality is available in this service.

**HTTP POST**

#### /order

The ability to place an order for a customer. In the database a record of this order is persisted. When an order is placed by a customer, the stock level for the particular items on order will be decremented automatically.

#### Example JSON Payload

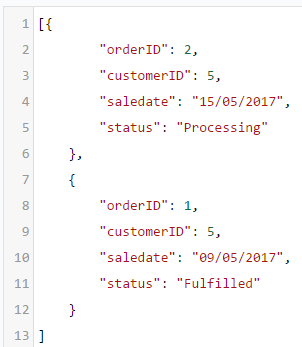


**HTTP GET**

#### /order

The ability to retrieve a list of orders for a particular user. Users can view their historical orders via a link in the navigation bar when the user is logged in to the application.

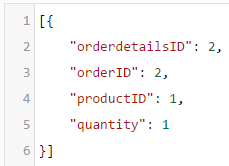
#### Example JSON Response



#### /orderDetails

This is used to view the low level details of a particular order. This data is accessible to both users and administrators via an option in the user interface.

#### Example JSON Response



#### /allOrders

This API will return all historical orders for all customers in every state. It is a function that is only accessible to Administrator users from the Administration panel. The JSON response is similar to that of the /order API.

## Front-end Service

### Purpose

The front-end service is responsible for defining the main user interface of the application, and handling all events related to it. In order to achieve this, it interacts with each of the other services to perform the necessary functions. It is to main entry point to the application for users.

Within the front-end source tree, the api folder contains wrappers that enable the front-end to talk to the other services.

The cart, catalogue, user, stock, and order service need to be up and running and accessible to the front-end service in order for it to function correctly.

### Functionality Provided

The front-end services provides the user interface definition, and integrates with the other services in order to provide the application functionality.

In contrast to the previous services discussed in this document which don’t have user interfaces and just provide API’s, the front-end service contains the mark-up that defines the entire user interface of the application.

The user interface is defined in public/index.html.

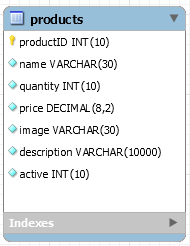
Most of the user interface styling is accomplished via bootstrap[[3]](#footnote-3), but there is a separate style.css in the same directory which defines some custom CSS.

# Section 3 – Database Design

The database design is quite simple currently, containing just 4 tables that handle the persisting of the required data.

## Products Table

The products table is used to store the items that are available for sale on the site.

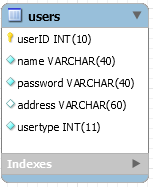


#### Column Definitions

* productID – This is the unique ID of a particular product in the products table. It is the primary key.
* name – The name of the product as it should appear on the user interface.
* quantity – The quantity of the product currently in stock (used for stock administration).
* price – The price of the product as it should appear on the user interface.
* image – The name of the image related to this product. These are stored in the front-end service source tree under public/images.
* description - The description of the product as it should appear on the user interface.
* active – A binary flag that indicates wheatear the product is active in the catalogue or not. If this is set to 0, then the product will not be available for purchase in the user interface.

## Users Table

The users table stores the users of the site (both customers and employees).

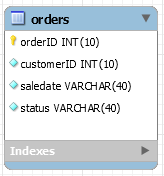


#### Column Definitions

* userID – The user’s unique ID. It is the primary key.
* name – The username the user will use to log in.
* password – The password the user will use to log in.
* address – The user’s address.
* usertype – This is used to determine if the user is an administrator or not.

## Orders Table

The orders table is where user orders are persisted. This table stores the high level data, with the actual products etc. being purchased being stored in the orderdetails table (see later below).

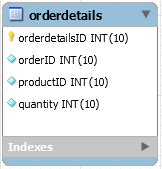


#### Column Definitions

* orderID - Unique ID for the order. It is the primary key.
* customerID – The ID of the customer making the purchase (essentially the userID from the users table).
* saledate – The date the sale was made in the format (DD/MM/YYYY).
* status – a string representing the current status of the order, e.g. ‘*Processing*’.

## OrderDetails Table

This stores the low level details (i.e. the product and the quantity) related to a user order. If an order contains multiple products, each one will be a separate row in this table.



#### Column Definitions

* orderDetailsID – The unique ID for the details of this order. It is the primary key.
* orderID – The ID of the order that these details relate to in the orders table.
* productID – The ID of the product being ordered in the products table.
* quantity – The requested quantity of the product being ordered.

1. https://expressjs.com/ [↑](#footnote-ref-1)
2. Note that the application is still in development at time of writing this document, please ignore any issues/downtime until the final codebase is submitted before the Assignment 2 deadline (May 22nd). [↑](#footnote-ref-2)
3. <http://getbootstrap.com/> [↑](#footnote-ref-3)