**Cork Institute of Technology**

**High Level Design Document**

**Cloud Development Frameworks Assignment 2**

**Jimmy Collins – R00145569**

Contents

[Section 1 - Introduction 2](#_Toc482129406)

[Section 2 - Microservices Overview 3](#_Toc482129407)

[Cart Service 3](#_Toc482129408)

[Purpose 3](#_Toc482129409)

[Functionality Provided 3](#_Toc482129410)

[Catalogue Service 4](#_Toc482129411)

[Purpose 4](#_Toc482129412)

[Functionality Provided 4](#_Toc482129413)

[User Service 5](#_Toc482129414)

[Purpose 5](#_Toc482129415)

[Functionality Provided 5](#_Toc482129416)

[Stock Administration Service 5](#_Toc482129417)

[Purpose 5](#_Toc482129418)

[Functionality Provided 5](#_Toc482129419)

[Order Service 6](#_Toc482129420)

[Purpose 6](#_Toc482129421)

[Functionality Provided 6](#_Toc482129422)

[Front-end Service 7](#_Toc482129423)

[Purpose 7](#_Toc482129424)

[Functionality Provided 7](#_Toc482129425)

[Section 3 – Database Design 7](#_Toc482129426)

# 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 number of services, 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 LINK for viewing[[1]](#footnote-1).

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:
* Password:

**Non-Administrator**

* Username:
* Password:

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 application.

## 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 service runs on port 3003.

### Functionality Provided

The cart service provides three main API’s 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, it is also uniquely identified by a cart ID.

TODO – Example JSON?

HTTP GET

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

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

TODO: Example JSON response?

HTTP DELETE

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

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

## 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).

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

The catalogue service runs on port 3002.

### 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.

#### /deleteProduct

The ability to completely delete a product from the catalogue (e.g. if that product is no longer available for sale).

#### /deactivateProduct

The ability to deactivate a product in the catalogue (e.g. if it is out of stock). Such products will still be displayed in the front-end, but with an ‘Out of Stock’ indicator.

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.

#### /getProduct

The ability to get the details of a specific product, given its product ID. (TODO: Is this used anywhere?)

## User Service

### Purpose

The user service is responsible for user management in terms of user registration, and also 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

#### /login

User Registration – the registration user interface on the front-end calls into this functionality in order to add new users to the database. All new users are added as Customer users, but any user can be made an Administrator later (TODO: how?). 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.

#### /register

User Log-In – this logic 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, the customer ID and customer type are returned to the front-end service as a JSON response.

## Stock Administration Service

### Purpose

The purpose of this service is to administer the stock levels available in the application.

TODO

### Functionality Provided

The following functionality is available in this service.

HTTP POST

#### /incrementStock

The ability to increment 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 company.

#### /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.

#### /currentStock

The ability to get the current stock levels for each product. This is used in the administrator interface so that administrators can keep an eye on stock levels.

#### /bestSellers

Best sellers??? TODO

## Order Service

### Purpose

This service is used to process user 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.

HTTP GET

#### /order

The ability to retrieve a list of orders for a particular user. A link to view these is available in the navigation bar when a user is logged in.

#### /orderDetails

TODO

#### /allOrders

TODO

## Front-end Service

### Purpose

The front-end service is responsible for defining and managing the main user interface of the application. It is to main entry point to the application for users. In order to achieve this, it interacts with the other services.

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

TODO – More general stuff about the code base.

The front-end service runs on port 8079. The cart, catalogue, user, stock, and order service need to be running in order for the front-end to function correctly.

### Functionality Provided

TODO

#### Logging In

When a user logs in the following sequence of events happens (triggered via jQuery):

* The submit of the login form is detected.
* The code prevents moving to a new page (check this – event.preventDefault());
* A new variable is created that is created from the values in the login form (the username and password).
* This is then turned into JSON.
* Next this is posted to /login (contained in the api/users/index.js – the ‘helper’ for the users service).
* This login then posts the JSON body to the users service and checks the response code.
* If the response code is HTTP 200 the body is parsed and returned.
* If an error occurred, the status of the request is set to HTTP 500 and the error output to the console.
* A cookie is also created on the client.

# Section 3 – Database Design

TODO

1. Note that the application is still in development at time of writing this document, please ignore any issues until the final codebase is submitted before the Assignment 2 deadline (May 22nd). [↑](#footnote-ref-1)