ECommerce API Documentation

By Juliana Reider and Andrew Rohrer

Getting Started

The current version of the API lives at **https://localhost:8081 and in AWS at**

Versions

Version	**Date**	**Changes**
Version 1	10/25/2018	Initial deployment
Version 2	10/26/2018	Change layout and add CreditCard/Order API
Version 3	12/02/2018	HATEOAS and client-side added

Common Flows Through our UI Client

Search -> Login -> Add items to Cart -> Place order -> View orders (click “My Orders”) -> View order status -> View order total -> cancel Order

Login-> Search -> Add items to Cart -> Place order -> View orders (click “My Orders”) -> View order status -> Cancel order

Login -> View my Orders -> search Items -> Add Item to cart -> Order Item -> Search Items -> Add Items to cart -> place Order -> View Order status

Endpoints

Verb	**Endpoint**	**What It Does/ Domain info Returned**	**Links Returned (HATEOAS)**	**Path Parameters**	**Method Params (Place in the body of request)**
Partners Resource					
GET	/partners/{partnerid}	Returns partner Representation based on unique identifier.	Delete partner link	- partnerId: String	N/A
POST	/partners	Register & create partner profile. Returns PartnerRepresentation	Delete partner links	N/A	partnerRequest: PartnerRequest
PUT	/partners/{partnerId} /newProduct	Adds product to partner & Add product to Market place	Delete partner link	partnerID: String	productRequest: ProductRequest
DELETE	/partners/{partnerId}	Deletes Partner by ID	N/A	String id	Status OK
Product Resource					
GET	/products/{productId}	Returns product Representation based on unique identifier.	Buy link Search Link	productId: String	N/A
GET	/products/searchres	Returns product	Buy link (for each)	Searchterm:	N/A

	ults/{searchterm}	Representations that match the given search term	Search Link	String	
POST	/products	Creates the new product in the database and returns Product Representation based on that new product	N/A	N/A	ProductRequest: ProductRequest
Credit Card Resource					
GET	/creditcard/customercards?customerID=3	Get all credit cards for a specified customer in the query string	N/A	customerID:String	N/A
GET	/creditcard/customercard?ccNo=<creditcardnumber>	Get specific credit card information	N/A	ccNo:String	N/A
DELETE	/creditcard/creditcard?ccNo= <creditcardnumber>	Delete specific credit card	N/A	ccNo:String	N/A
POST	/creditcard/newcreditcard	Enter new credit card into system	N/A	N/A	ccNum:String ccHolder:String ccExpirationDate:String ccSecurityCode ccCUstomerNo
Order Resource					
GET	/order/orderService/	Get all orders for a customer	Cancel Order Link	customerID:	N/A

	orders		Order Status Link Search Links	String	
GET	/order/orderService/order	Get information for a specific order	Cancel Order Link Order Status Link Search Link	orderID: String	N/A
GET	/order/orderService/order/fulfillmentAcknowledgeFulfillment{orderId}	Get fulfilment acknowledgement of a specific order	N/A	orderId:String	N/A
GET	/order/orderService/order/status?orderID=3	Get status of an order	Cancel Order Link Order Status Link Search Link	orderID:String	N/A
DELETE	/order/orderService/order/cancelledorder?orderID=3	Cancel an order	N/A	orderID:String	N/A
POST	/order/orderService/order/neworder	Place a new order	Cancel Order Link Order Status Link Search Link	N/A	orderNo: String productsOnOrder: Arraylist of product/quantity of product customerID: String
Customer Resource					
GET	/customers	Get all customers	Delete Customer Links Search Link	N/A	N/A
GET	/customers/{customerId}	Get a customer by ID	Delete Customer Link Search Link	String: id	N/A

POST	/customers	Add new customer	Delete Customer Link Search Link	N/A	CustomerRequest: customerRequest
POST	/customerAuthentication	Login for customer by their username and password	**My Orders Link** *(access customers list of orders)* Delete Customer Link	N/A	CustomerRequest: customerRequest
DELETE	/customers/{customerId}	Deletes customer by ID. Returns status OK	N/A	String: id	N/A

Samples:

Sample code

This is sample javascript code you might have client-side to harness our Restful API and achieve HATEOAS:

```
/**************************************
* POST METHOD : User authentication
**************************************/
   $("#loginbtn").on("click", function(){
      $.ajax({
         url: customer_Authentication_URI,
         type:"POST",
         data:
            JSON.stringify({
                  userName: $("#username").val(),
                  password: $("#userpassword").val(),
            "}),
         contentType:"application/json; charset=utf-8",
```

```
        accept: "application/json",
        dataType:"json",
        success: function(data, status){
                console.log("This is the status: " + status);
                console.log(data);
           if(data == null){
              alert("Incorrect username or password");
           } else {
              isSignedIn = true;
              hideLoginModal();
              alert("Login Success!");
              var parsedResponse = data;
              var id = parsedResponse.id;
              signedInCustomerNo = id;
              var myOrderURL = data.link[0].url;  //grab link from returned data
              console.log("CustomerID: " + id);
              console.log("URI: " + myOrderURL);
              // insert returned URL into menu (links user to their own orders)
              $("#customerorders").attr('custordurl', myOrderURL); //Stored so user to can access their personal orders
           }
        }
     });
  });
```

Sample code

This is sample Java code you might have in your backend to harness our API:

```
/**************************************
* GET METHOD : Get an existing Partner
```

```
**************************************/
 //Providers Set up
List<Object> providers = new ArrayList<Object>();
JacksonJsonProvider provider = new JacksonJsonProvider();
provider.addUntouchable(Response.class);
providers.add(provider);

System.out.println("GET METHOD .....Get partner with id 17");
WebClient getClient = WebClient.create("http://localhost:8081", providers);

 //Configuring the CXF logging intercepter for the outgoing message
WebClient.getConfig(getClient).getOutInterceptors().add(new LoggingOutInterceptor());
//Configuring the CXF logging intercepter for the incoming response
WebClient.getConfig(getClient).getInInterceptors().add(new LoggingInInterceptor());

// set Accept and ContentType headers
// set path with Partner ID = 17
getClient = getClient.accept("application/json").type("application/json").path("/partnerservice/partners/17");

//The following lines are to show how to log messages without the CXF interceptors
String getRequestURI = getClient.getCurrentURI().toString();
System.out.println("Client GET METHOD Request URI:  " + getRequestURI);
String getRequestHeaders = getClient.getHeaders().toString();
System.out.println("Client GET METHOD Request Headers:  " + getRequestHeaders);

//to see as raw XML/json
String response = getClient.get(String.class);
 System.out.println("GET METHOD Response: ...." + response);

/************************************
* END
```

**********/

Sample Request

URI:
http://localhost:8081/partnerservice/partners/1

Verb:
POST

Headers Example:
Accept:application/xml
Content-Type:application/xml

XML Body:

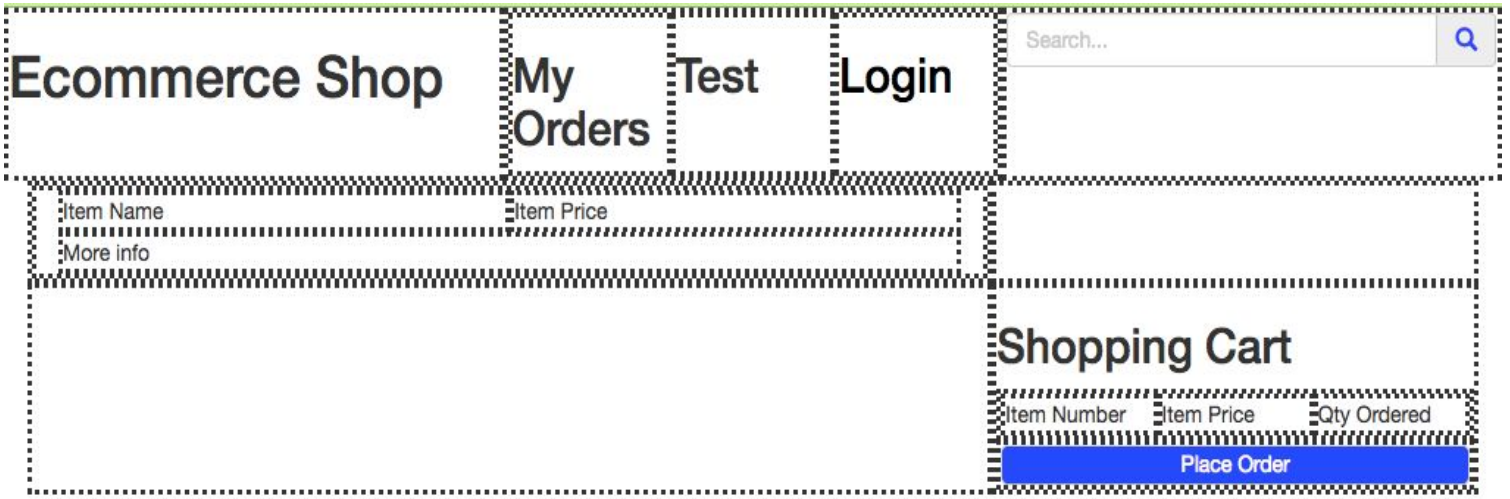
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<partnerRequest>
        <id>0</id>
        <companyName>FredsFlowers</companyName>
        <userName>fred</userName>
        <password>flowers</password>
</partnerRequest>
```

Notes on Status Codes:

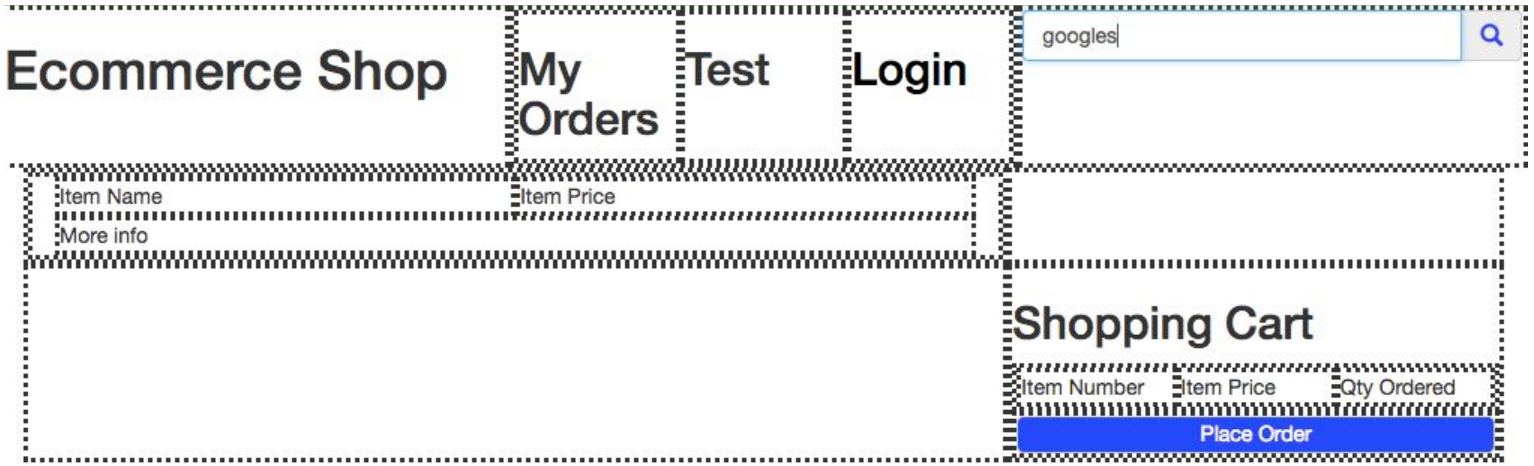
- If you get a HTTP 400 ; Bad Request Status code response, it's most likely that you tried to access something that does not exist as an api or in our database.

Sample UI:

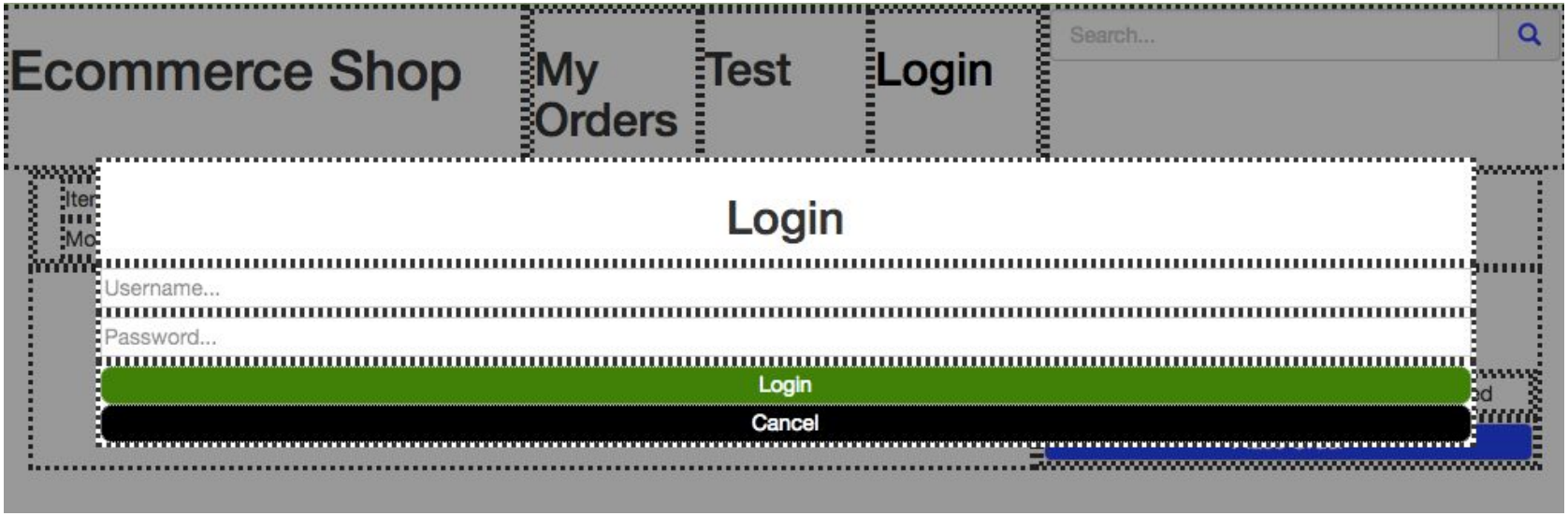
Opening Page:



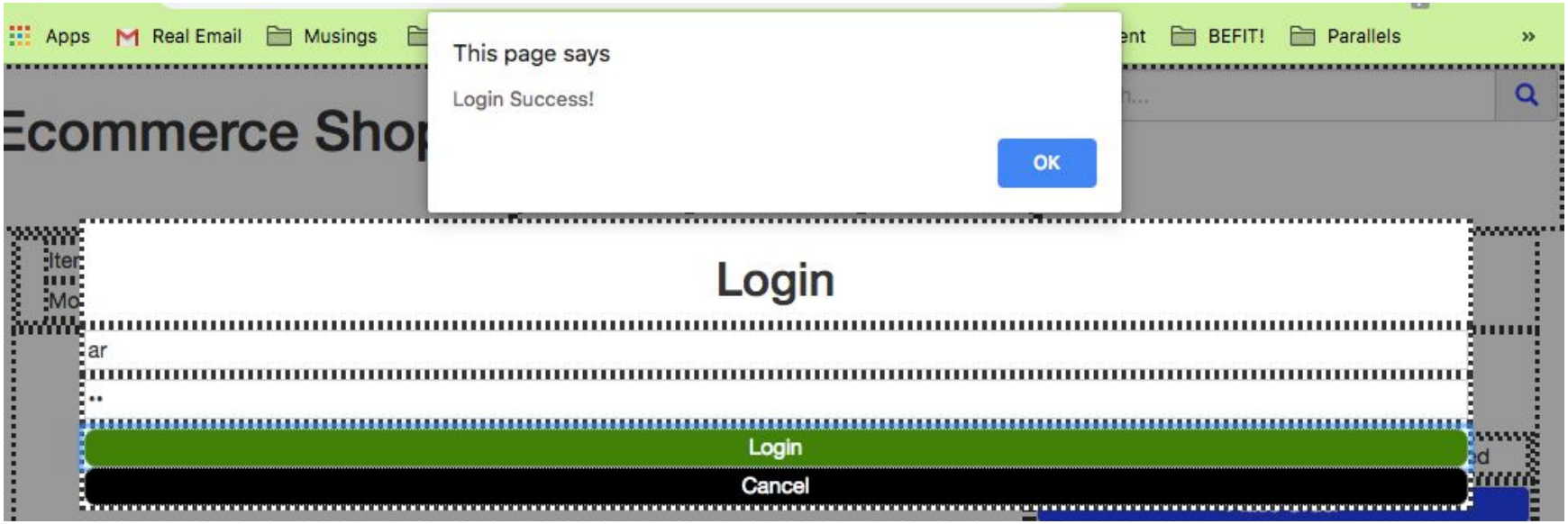
Search Functionality:



Login model:



Successful Login:



My Orders Page: