# EZCampaignManager and here is how…

## Solution Structure

The solution contains 6 projects:

1. The UI tier – **Campaign Manager**
2. The DataTier – **CMDataAccess** – This is set up to be used with Dapper\* but at the time of writing is not being used as we are getting all our data from the Mailgun API and not saving any of it to the database.
3. The business layer – **CMBusiness** which contains the services and methods.
4. A layer for globally shared entities – **CMEntities**.
5. A mailgun wrapper layer – **MailgunAPIDirect** which is a wrapper around the Mailgun API.
6. There is also a tests project, **CampaignManager.Tests**, should there ever be time to write some!

\* A high performance Micro-ORM supporting SQL Server, MySQL, Sqlite, SqlCE, Firebird etc..

### How it works…

When a user hits the site, they are greeted with the login screen.  
Login names are given to the client by us when a new account is added.

The format for the login name is ‘CM@leisure.south-norfolk.gov.uk’.  
The bit after the @ is the domain that their emails are sent from.  
This is used once they have logged in to pull back the correct campaigns for the client.

Database with the logins is on ezlive.ez-runner.com [EZCampaignManager], table [dbo].[CustomerDetails].

It uses ASP.NET Identity for authentication. You can read the basics here

<https://docs.microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity>

So if login is successful they will be redirected to the tags screen.

The code that does this is in the AccountController

public ActionResult Login(string returnUrl)

{

if (User.Identity.IsAuthenticated == true)

{

return RedirectToAction("index", "Tag");

}

ViewBag.ReturnUrl = returnUrl;

return View();

}

This directs the code to the Tag controller and the its index method.

Tags are what mailgun calls campaigns.

The index method of the TagController calls the ‘TagService’ (which is located in the ‘CMBusiness’ layer in the services folder) ‘Get’ method. This in turn calls the ‘GetTags’ method in the ‘MailgunAPIDirect’.

This calls the Mailgun API using the clients Mailgun ApiKey and returns a list of all the tags for the client.

Mailgun will only let you pull back 1000 records at a time. If there are more than a 1000 records then Mailgun will send a link to the second page of results with the first result set.

Once the first set of results is returned then a count is done to see if there were more than 1000 results. If there are it uses the next page link to get the second set of results. These are again are counted and if there are 1000, gets the next page link and uses that to get the next set of results. This carries on until the count of results is less than 1000 indicating there are no more results to be had.

The results are then displayed to the user in index.cshtml which takes the list of tags and does a foreach loop to display each row.

@if (Model.Tags.Count >= 1)

{

foreach (MailgunAPIDirect.Entities.Tag tag in Model.Tags)

{

<div class="row">

<div class="col-sm-7 campaign-manager float-left ">

<a href = "/Tag/TagStats/?id=@Url.Encode(tag.tag)&start=@Url.Encode(tag.first\_seen.ToShortDateString())">@tag.tag</a>

</div>

<div class="col-sm-2 campaign-manager float-left ">@tag.first\_seen.ToShortDateString()</div>

<div class="col-sm-2 campaign-manager float-left ">@tag.last\_seen.ToShortDateString()</div>

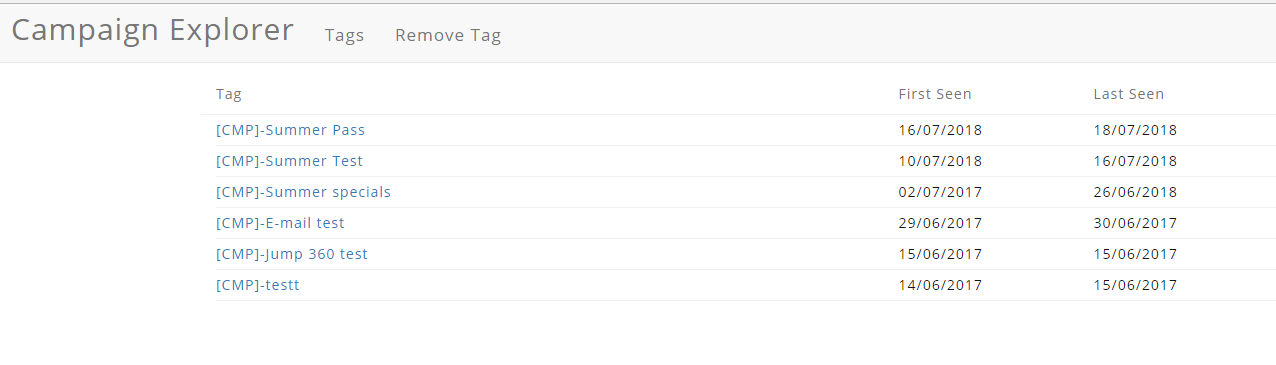
</div>

}

}

If no tags are found then a “No Campaigns Found” message is displayed to the user via a partial view \_MessagePartial.cshtml which can be used with all views to display information to the user.

If there are tags the user will see a list of them.



Notice in the above foreach loop that the href for each Tag. This is pointing at the TagsStats method on the TagController.

Domain/Tag/TagStats/?id=*thetagid*&start=dateTheTagWasFirstseen

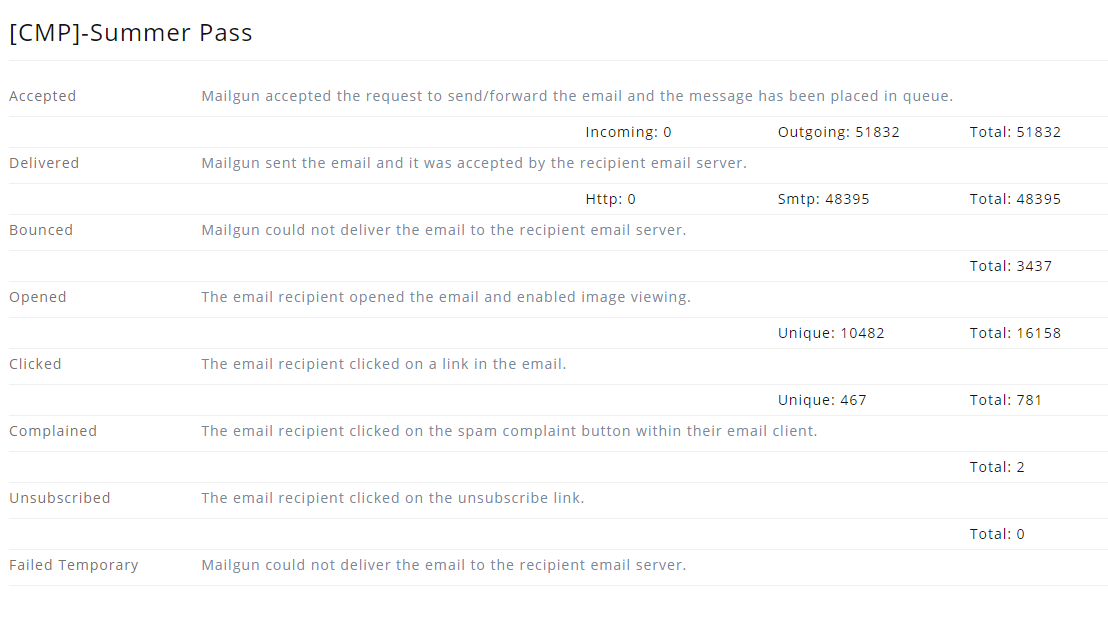
So if the title of the Tag is clicked it comes into the TagStats method with the tag id and the date it was created (or first seen, if you are Mailgun).

The TagStats method takes the id and startdate, calls the Tag Service and passes the id and startdate to the GetTagStats method. This in turn calls the GetTagStats method in the Mailgun API which returns a list of the EventStats entity.

Events are what Mailgun refer to as things like delivered, accepted, opened, etc.

Mailgun require the start date to be in Unix Time so there is a conversion method in TagMethods.cs that does the conversion.

So if we click on the first tag in the list above “Summer Pass” then we see this:

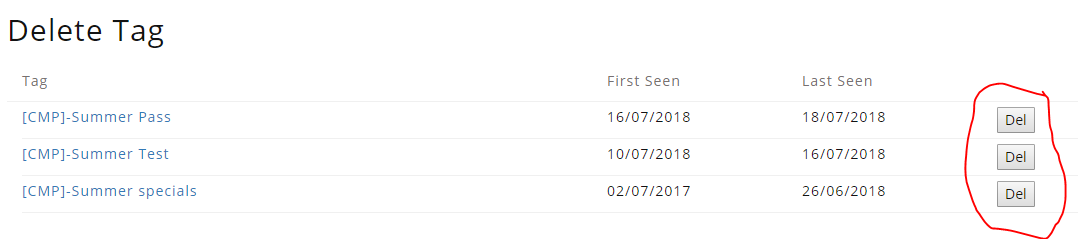


The HTML is in the TagStats.cshtml file in the Views folder.

To get back to the list of tags then the user can press the back button on the browser or the Tags link on the menu bar.



There is also a ‘Remove Tag’ link which is pointing at the Tag Controller ‘Delete’ method. This works in the same way as the index method, whereby it pulls back all the tags but this time the results are displayed in the view Delete.cshtml. This view is the same as the index view except now each tag has a ‘Delete’ button next to it.

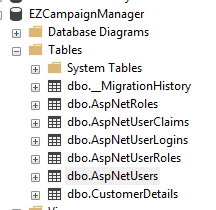


Each ‘Delete’ button in turn is pointing at the ‘DeleteTag’ method in the TagController which takes the id of the tag to delete and passes this along to the TagService which passes it along to the ‘DeleteTag’ method in the Mailgun API and the tag with that ID is removed.

### Admin and here is how…

As mentioned earlier the database with the logins is on ezlive.ez-runner.com named [EZCampaignManager].

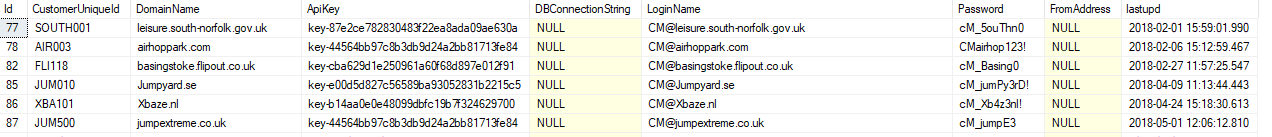
At present this consists of 5 tables



The tables prefixed with AspNet are those used by ASP.NET Identity. Currently we are only using AspNetUsers which is populated when a new client is registered.

The [CustomerDetails] table was added to store extra information about each client should it be needed in the future.

Here is what it stores…..



#### Adding a new client.

To add a new client, you need to login as the admin user.

un: [CMAdmin@Ez-Runner.com](mailto:CMAdmin@Ez-Runner.com)  
pw: cM\_Adm1n3zR

Once logged in as admin you will see two extra items on the menu bar. ‘Add Customer Details’ and ‘Register’

Hiding and showing these is done in the \_Layout.cshtml with a simple check to see who is logged in and if it is the admin user then it shows the extra items.

<div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">

<ul class="nav navbar-nav menu-item">

<li><a href="#">@Html.ActionLink("Tags", "Index", "Tag")</a></li>

<li><a href="#">@Html.ActionLink("Remove Tag", "Delete", "Tag")</a></li>

@if (HttpContext.Current.User.Identity.Name == "CMAdmin@Ez-Runner.com")

{

<li>@Html.ActionLink("Add Customer Details", "AddCustomerDetails", "Account", routeValues: null, htmlAttributes: new { @class = "nav-spacer" })</li>

<li>@Html.ActionLink("Register", "Register", "Account", routeValues: null, htmlAttributes: new { id = "registerLink", @class = "nav-spacer" })</li>

}

</ul>

<ul class="nav navbar-nav navbar-right">

<li class="login-linkelement">@Html.Partial("\_LoginPartial")</li>

</ul>

</div



‘Register’ adds the client to the [AspNetUsers] table this is necessary for ASP.NET Identity.

‘Add Customer Details’ adds the client to the [CustomerDetails] table.

## Site URL

<https://ezcampaignmanager.ez-runner.com/>

## Outstanding

### Edit existing customer's details

Currently this can only be done by editing the database details directly. There is no interface provided by the Campaign Explorer.