Venimus Dev Diary

# Saturday 28th Dec 2019

Get user registration now includes custom questions and their answers

TODO: Displaying past and future events

# Friday 27th Dec 2019

Add support for questions when creating/updating/getting events

Removed Automapper from events controller as the code was clearer without it.

# Thursday 26th Dec 2019

API allows questions to be provided when creating/updating events

# Wednesday 25th Dec 2019

Users can now change their profile pictures

# Tuesday 24th Dec 2019

System administrators can now create and edit groups.

Added page to view event attendees

TODO: Add error handling to all update methods

# Monday 23th Dec 2019

Hide/show group actions based on user security

Validation errors when creating/updating events are now show on the UI

# Sunday 22nd Dec 2019

Added page to allow sysadmin administrators to create group administrators.

Combined two event controllers so there is now a single generic method for getting list of events.

~~TODO: Hide the button if the user doesn’t have access~~

~~TODO: Handle the parsing of errors if creating an event fails.~~

TODO: Extend the get list of events controller to return past events

# Saturday 21st Dec 2019

Added API to allow administrators to add members to a group.

# Friday 20th Dec 2019

Added validation to the two user details pages and a bit of styling. (Still looks rubbish)

Added new page to allow events to be created.

Added flag to get group view model to indicate if the user is allowed to created events.

# Thursday 19th Dec 2019

Allow a user to change their details by clicking on their profile picture.

There are now some missing tests ☹

When the user first registers, we need a test to ensure that a message is sent to slack

We need a test for the slack webhook

It would be nice if we could update the approval message in slack if a user changes their details.

# Wednesday 18th Dec 2019

Added a page to allow the user to confirm their details the first time they log in.

# Tuesday 17th Dec 2019

Combined three different group controllers into a single controller which takes query parameters instead.

# Monday 16th Dec 2019

Amended the CreateInitialData app to load in all the groups. It does this by parsing the markdown files used by the main website.

Added a new property to groups called StrapLine, this is initially the first line of the description.

Changed the home page so that it now shows the StrapLine rather than the full description. Changed the Join Now button to read “Find Out More”. Clicking this takes you to a group’s details page.

Amended the get group API so that you no longer need to be authorised in order to call it.

# Sunday 15th Dec 2019

Tidied up the code which post to and receives replies from slack. Also add an additional feedback into slack after the user has clicked approve/reject.

Couple of jobs remaining

Add a unit test which simulates the webhook call from slack. Maybe use PACT?

~~Add ApprovedBy / RejectedBy flags to the user model.~~

~~Add screen to allow the user to enter their bio and check their details.~~

# Saturday 14th Dec 2019

Spent some time playing with integrations to slack today. When a user signs up I can now post a message into slack asking an administrator to approve/reject the request. When they click a button it calls a webhook back into the API site.

For testing purposes I’ve made my local site visible using ngrok.

Had issues getting the callback to work, had to remove HttpsRedirection from startup.cs. Also marked the post method with [Consumes("application/x-www-form-urlencoded")] but I’m not sure that’s working correctly.

(There also seems to be a bug in the site related to the token expiring.)

# Friday 13th Dec 2019

Continued with the generic questions, quite a few changes – I was glad to have good test coverage 😊

Started integrating with slack. When a user wishes to join a group, we now post a message to the channel in slack. The message will contain a button for administrators to click to approve the registration. (Maybe just a link to the web site)

# Thursday 12th Dec 2019

# Started making the event registration questions more generic. Created a new Question model with the following properties:

* Code
* Caption
* QuestionType

The question type will be values such as Text, YesNo, Date etc.

Number of guests is a special type of question, with a question type of NumberOfGuests.

Some types might require additional properties, such as a poll, or the maximum number of guests.

# Wednesday 11th Dec 2019

Hooked up the amend registration details button. Decided to simplify things by combining the POST and PUT methods for CREATE/AMENDED event registration into a single PUT method.

Next step it to make the registration questions a bit more generic. Currently we ask the following three questions:

* Number of Guests
* Dietary Requirements
* Message To Organiser

It would be nice to make this more generic so that in the future the group owner can define their own questions.

# Tuesday 10th Dec 2019

Hooked up the button to allow a user to unregister from an event. Although from the websites point of view we are issuing a delete command to the endpoint (api/user/groups/{groupSlug}/Events/{eventSlug})

Internally we just update their registration record to say that they are no longer attending. This means we can tell if a user has never signed up for the event, or if they were signed up but can no longer attend.

The view members button on the MyGroups page is now hidden if the user doesn’t have access to call the API. (They must be either a system administrator or be an approved member of the group)

I’m now wondering if I need to change the design of the APIs slightly. Instead of having multiple APIs to return a list of groups, I think we need just one, but it also contains a list of available actions you can carry out on the group.

For example:

* Join Group
* View Members
* Leave Group

The API might need a new parameter to be able to specify if you only want to see groups you are/aren’t a member of.

# Monday 9th Dec 2019

Aim for today was to add support for the rendering of the markdown used in the descriptions of the groups and events.

First decision is where the conversion into HTML should take place. We can do it either in the APIs or the Frontend. I think it makes sense to do it in the frontend as it’s possible that a client might need a format other than HTML. For example, in a windows app.

I’ve included a nuget library called Markdig. This seems to be straight forward to use

var html = Markdown.ToHtml(markdown);

As we need to do this in a few places in the frontend site I’ve create a custom TagHelper called **MarkdownTagHelper**. This means that razor view we just need to write

<**markdown**>@item.Description</**markdown**>

The is library is then rendering links and text formatting (bold etc) fine. Had trouble with carriage returns at first. It turned out that the **TagHelper** was encoding them **as &#xD;&#xA;** which **MarkDig** didn’t understand. Easily fixed by **Replace("&#xD;", "\r").**

Next, I think I’ll add the ability to unregister from an event, and also change your registration details.