Data Plan

Group UG22

Feature/Interaction	Purpose/Description	Request Data Source	Request Sent to Server	Processing Needed/Response Data Source	Response Sent to Client	Request Type & Path	Other Notes
Events:							
Events Search (Public)	Search for events, applying filters as specified by the user, or using default filter values.	An AJAX request when the user presses the search button on the events pages	No request body. Request URL parameters: *search:* the search term *from:* start date of search, defaults to current date *to:* stop date of search *n:* the max number of events to display, defaults to 10 *branch:* the id of branch to filter by, can have multiple values. Where no default value is specified, the filter is not applied by default.	Apply filters to events in the database as specified by the filters (or the default filter values).	JSON formatted AJAX response containing an array of event details objects, as below: [{ id: 1, title: 'Event 1 Title', description: 'Description of event' date: '1/7/24', startTime: '1pm', endTime: '3pm', dayOfWeek: 'Monday', location: 'Adelaide', image_url: '/path_to_image.png' }, …]	GET Request /events/search (with optional parameters as specified)	This route is available without logging in, as it will not return any non-public events. Once users have logged in, they should make the same request to /users/events/search instead which will include private events not visible to the general public.
Events Search (Logged in)	As above	As above	As above	As above	As above	GET Request /users/events/search	Requires user to be logged in

RSVP for event	RSVP yes or no to an event	An AJAX request when the user presses the RSVP button	JSON formatted AJAX request:
			{
			eventID: {event id}
			RSVP: {'yes' or 'no'}
			}
				Add the users RSVP for the event to the 'user_event_attendance' database table. Will need to check if there is already a response from user, update this instead of adding a new entry if so.	No response body necessary, would simply send a 200 OK message if successful, or the relevant error code otherwise.	POST Request
						/users/events/rsvp
							Requires a user to be logged in, as well as a member of the branch the event is for.

Event Details	Get the details for a specific event. This includes: - Title - Description - Event details in dot point form - Date - Start time - End time - Location - A link to the event image	An AJAX request when the events details page loads	No request body, event ID is specified in URL of request	Access database entry for the event, get required data from it.	JSON formatted: { id: 3, title: 'Event 3 Title', description: 'Description of event 3', details: ["The first dot point", "The second dot point", ["nested list", "like", "this", "one", "here"], "Final dot point"], date: '3/8/24',	GET Request /events/id/{ID}/details.json	Need to check user is authenticated if the event is private (as well as that they are a member of the associated branch)

					startTime: '12pm', endTime: '5pm', location: 'Adelaide', image_url: '/not_an_image.png' };		

Create Event	Sends a POST request to create a new event.
	
	Only an authenticated branch manager can perform this action.
		An AJAX request when the submit button is pressed on the create events page.
		
		The create news button will only be dynamically shown to managers at the top of the events page. For others, it will be hidden.
			JSON formatted AJAX request:
			{
			title: 'Title here',
			description: "Description",
			details: [
			"The first dot point",
			"The second dot point",
			["nested list", "like", "this", "one", "here"],
			"Final dot point"
			],
			date: '1/7/24',
			startTime: '1pm',
			endTime: '3pm',
			dayOfWeek: 'Monday',
			location: 'Adelaide',
			image_url: '/image.png,
			}
			
			The id will be added by the server, after validation.
				Validation of data, before placing in the database.	Return 200 OK on success, or the relevant error code if it fails (such as a bad request if the data is invalid).
					
					Response body contains the ID of the new event.
						POST request to manage/event/create	Only specific branch managers will be able to perform this action. For example, a branch manager can only create an event for their specific branch.
							
							The manager who created the event will be redirected to the created page. On failure, an error message will show up on the create event page.

Edit Event	Sends a POST request to edit a specific event. Only an authenticated branch manager can perform this action. They must also be of the same branch as the event. The purpose of this is to amend errors or outdated information in events. This could include changes in dates, times or venues.	An AJAX request which is sent when the edit event button is pressed. This button is inside each event container in the event page.	As above, including only fields which have been updated.	Validation of data, before updating fields which have been changed in the existing event in the database.	No response body necessary, return 200 OK on success, or the relevant error code if it fails (such as a bad request if the data is invalid)	POST request to manage/event/edit/{ID}	Only specific branch managers will be able to perform this action. For example, a branch manager can only edit events for their specific branch. The manager who updated the event will be redirected to the updated page. On failure, an error message will show up on the update event page. The update event page will look similar to the create event page. However, the forms will already be filled using the information currently on the event page.
View RSVP Responses	Sends a POST request to create a JSON formatted list of RSVP responses. The purpose of this is to check who and how many people will be attending an event. This information can then be used to send news updates, requesting for more volunteers.	An AJAX request when a manager loads the view responses page for an event.	No request body required.	Query database to get responses for that particular event.	JSON formatted: { yes: [{name: (full name), id: (uid)}, {name: (full name), id: (uid)}, {name: (full name), id: (uid)}], no: [{name: (full name), id: (uid)}, {name: (full name), id: (uid)},	POST request to manage/event/responses/{Event ID}	Only available to branch managers (of the branch the event belongs to) and system admins.

					{name: (full name), id: (uid)}
					]
					}		

Delete Event	Sends a POST request to delete a specific event. Only an authenticated branch manager can perform this action. They must also be of the same branch as the event. The purpose of this is to remove old or cancelled events and remove clutter from the site.	An AJAX request which is sent when the delete event button is pressed. This button is inside each event container in the events page.	No request body, event ID to delete is specified in URL of request	Remove the event from the database	No response body necessary, return 200 OK on success, or the relevant error code if it fails.	POST request to manage/event/delete/{ID}	Only specific branch managers will be able to perform this action. For example, a branch manager can only delete events for their specific branch. A confirmation button will also be added to avoid accidental deletes. This would simply change the delete button to a “confirm delete?” button. This is essential as deletion will permanently remove the update. A success message will be shown on the news page on successful deletion, otherwise a failure message will be shown
News:							
News Search (Public)	Search for news updates, applying filters as specified by the user, or using default filter values. This will be similar to the events search.	An AJAX request when the user presses the search button on the news pages	No request body. Request URL parameters: *search:* the search term *from:* start date of search, defaults to current date *to:* stop date of search *n:* the max number of news updates to display, defaults to 10. *branch:* the id of branch to filter by, can have multiple values. Where no default value is specified, the filter is not applied by default.	Apply filters to news updates in the database as specified by the filters (or the default filter values).	JSON formatted AJAX response containing an array of news updates details objects, as below: [{ id: 1, title: 'Title here', description: "Description", details: ["Paragraph 1", "Paragraph 2", "Paragraph 3" "More paragraphs here."], date_posted: '1/1/24', time_posted: '00.00pm', posted_by_branch: 'branch', image_url: '/image.png, }, …]	GET request to /news/search (with optional parameters as specified before)	This route is available without logging in, as it will not return any private news updates. Once users have logged in, they should make the same request to /users/news/search instead which will include private events not visible to the general public. The search button will automatically update to accommodate for this.
News Search (Logged in)	As above	As above	As above	As above	As above	GET Request /users/news/search	Requires users to be logged in as it returns private news updates.
News Details	The purpose of this is to retrieve details for a specific news update, so that it can be displayed. These details will be stored in a JSON object. This includes:	An AJAX request when the news details page loads.	No request body, news ID is specified in URL of request	Access database entry for the required news update from the “News” table. Retrieve required entry from the database.	If authenticated, JSON formatted: { id: 1, title: 'Title here', description: "Description here", details: ["Paragraph 1",	GET request to /news/id/{ID}/details.json	Need to check whether the user is authenticated and if the news update is private. Additionally, need to verify if the news update is a part of the same branch as the user. If this is not the case, the news update will

	- Title - Description - News article as a preformatted string. - Date posted - Time posted - The branch the news was posted by. - A link to the news image.				"Paragraph 2", "Paragraph 3" "More paragraphs here."], date_posted: '1/1/24', time_posted: '00.00pm', posted_by_branch: 'branch', image_url: '/image.png, }		not be sent. Instead, the user will be redirected to the login page.

Create News Update	Sends a POST request to create a new news update.
	
	Only an authenticated branch manager can perform this action.
		An AJAX request when the submit button is pressed on the create news update page.
		
		The create news button will only be dynamically shown to managers at the top of the news page. For others, it will be hidden.
			JSON formatted AJAX request:
			{
			title: 'Title here',
			description: "Description",
			details: [
			"Content here",
			],
			image_url: '/image.png,
			}
			
			The id, branch, date and time posted will be added by the server, after validation.
				Add a new entry to the “News” database.	A 403 Forbidden message would be sent to unauthorised users. These include visitors, users and managers from other branches.
					
					No response body necessary, would simply send a 200 OK message if successful, or the relevant error code otherwise.
						POST request to manage/news/create	Only specific branch managers will be able to perform this action. For example, a branch manager can only create news articles for their specific branch.
							
							The manager who created the article will be redirected to the created page. On failure, an error message will show up on the create news page.

Edit News Update	Sends a POST request to edit a specific news update. Only an authenticated branch manager can perform this action. They must also be of the same branch as the news update. The purpose of this is to amend errors or outdated information in articles.This could include changes in dates, times or venues.	An AJAX request which is sent when the edit news update button is pressed. This button is inside each news update container in the news page.	As above.	Updates a previous entry to the “News” database, using “article_id” to find the specific entry.	As above.	POST request to manage/news/edit/{ID}	Only specific branch managers will be able to perform this action. For example, a branch manager can only edit news articles for their specific branch. The manager who updated the article will be redirected to the updated page. On failure, an error message will show up on the update news page. The update news page will look similar to the create new page. However, the forms will already be filled using the information currently on the news page. The time and date posted on the current news page will be updated to the time and date when the update was published. This will be handled by the server.
Delete News Update	Sends a POST request to delete a specific news update. Only an authenticated branch manager can perform this action. They must also be of the same branch as the news update.	An AJAX request which is sent when the delete news update button is pressed. This button is inside each news update container in the news page.	No request body, article ID to delete is specified in URL of request	First, it will need to be verified if the branch of the news update matches the branch of the manager. If this condition is not met, no further action is taken. Find news update entry in the database using its provided article ID,	A 403 Forbidden message would be sent to unauthorised users. These include visitors, users and managers from other branches. No response body necessary, would simply send a 200 OK message if successful, or the relevant error code	POST request to manage/news/delete/{ID}	Only specific branch managers will be able to perform this action. For example, a branch manager can only delete news articles for their specific branch. A confirmation button will also be added to avoid accidental deletes. This would simply change the delete button to a “confirm

	The purpose of this is to remove old articles and remove clutter from the site.			and delete it from the database.	otherwise.		delete?" button. This is essential as deletion will permanently remove the news update. A success message will be shown on the news page on successful deletion, otherwise a failure message will be shown.
Branches:							
Join Branch	Enables a user to join a branch. This allows them to see private events, public events, updates etc.	AJAX Request when user clicks 'Join Branch' button on /branches page.	{ username } Validate username is valid Check branch is public/accepting new members ?	Adds user to 'user_branch_affiliation" table in DB.	JSON formatted response indicating success or failure of the operation, along with a message. { success, message }	POST Request to /branches/join Request Body: { "username": "example_username" }	Ensure proper authentication and authorization to restrict access to only logged-in users. Validate that the branch exists and is valid. Check if the user is already a member of the branch to avoid duplicate entries.
Create Branch	Allows a system admin to create a new branch for the Volunteer Organization.	User input from a form containing branch information such as name, address, phone number, etc.	An AJAX request triggered when the admin submits the branch creation form.	Server-side validation of input data, creation of a new branch entry in the database.	JSON formatted response indicating success or failure of the operation.	POST Request to /branches/create	Ensure proper authentication and authorization to restrict access to only system admins.
Update Branch	Allows a system admin to update information about an existing branch.	User input from a form containing updated branch information.	An AJAX request triggered when the admin submits the branch update form, along with the branch ID to be updated.	Server-side validation of input data, update of the corresponding branch entry in the database.	JSON formatted response indicating success or failure of the operation.	POST Request to /branches/update/{branch_id}	Ensure proper authentication and authorization to restrict access to only system admins. The {branch_id} parameter in the path specifies which branch to update.
Delete Branch	Allows a system admin to delete an existing branch from the Volunteer Organization.	User input, possibly just the branch ID to be deleted.	An AJAX request triggered when the admin confirms deletion, along with the branch ID to be deleted.	Server-side validation of input data, deletion of the corresponding branch entry from the database.	JSON formatted response indicating success or failure of the operation.	POST Request to /branches/delete/{branch_id}	Ensure proper authentication and authorisation to restrict access to only system admins. The {branch_id} parameter in the path specifies which branch to delete. Use caution with this operation as it permanently removes data from the system.
Branch Details	Get the details for a branch This includes: - Title - Description - Location - A link to the branch image - Id - Opening hours - Email - Phone number	An AJAX request when the events details page loads	No request body, event ID is specified in URL of request	Access database entry for the event, get required data from it.	JSON formatted: { id: 1, name: 'Adelaide', location: '129 Waymouth Street, Adelaide SA 5000', openingHours: '9am-5pm', phone: '0412345678', email: 'adelaidebranch@mealmates.com', description: 'branch description.', image_url: 'image.png', },	GET Request /branch//id/{ID}/details.json	Need to check user is authenticated if the event is private
Get List of Branches	Get a list of all the branch	An AJAX request which is	No request body	Get the name and ID of all the	JSON formatted AJAX response	GET Request	Could use a different route which is used

	names, as well as the corresponding branch IDs to allow for a list of the branches to be displayed for filtering events by branch.	sent as the events page loads.		branches from the database.	containing each branches name and ID of the form:
					[
					{
					id: {branch id (int)}
					Name: {branch name (string)}
					}
					]
						/branches/summary.json	for the branches page, but can use this one to avoid sending unnecessary information (we don't need to know the branches addresses or phone number on the events page).

User:							
View Users	Allow Volunteer Organization Managers and System Admins to view a list of users which includes details such as username, email and branch affiliations	An AJAX request which is triggered when a manager or system admin visits the user management page.	AJAX request which fetches list of users	Validate request to ensure it is made by a manager or system admin. Retrieve user data from the database such as username, email and branch affiliations	JSON formatted response containing the list of users and their details. { "id": 1, "username": "example_user", "email": "user@example.com", "branch_affiliations": ["Branch 1", "Branch 2"] },	GET Request to /users	Ensure proper authentication and authorization to restrict access to only Managers and System Admins.
Update User Information	User updates their profile information (e.g. username, email etc) This could include choosing a new email, changing password or updating notification preferences.	An AJAX Request on submission of user profile information form, containing updated information	An AJAX request on submission of the profile information form, containing the updated information	Validate/sanitise new data, update user record in DB. Data to be stored in 'users' table	JSON formatted response indicating success or failure of the operation, along with a message. {success, message}	PUT Request /users/:id	Ensure proper authentication and authorization to restrict access to only the logged-in user who owns the profile.
Delete User	Allow a user to delete their account from the system.	User action, triggered by clicking a "Delete Account" button on their profile settings page.	An AJAX request triggered when the user confirms the deletion of their account.	Validate the request to ensure it is made by the logged-in user. Delete the user record from the database and any associated data.	JSON formatted response indicating success or failure of the operation, along with a message.	DELETE Request to /users/:id	Ensure proper authentication and authorization to restrict access to only the logged-in user who owns the account. Provide confirmation prompts to prevent accidental deletions.
Miscellaneous:	-						

Login System	A user must be able to authenticate with the website.
	
	Allow the website to offer a tailored experience to each user. Store personal details. Support personalised notifications etc.
	
	This also ensures there is support to access admin panels, and restrict access to normal users.
		AJAX Request when user submits login or register form.	JSON Register
			{
			username,
			password,
			email
			}
			
			JSON Login
			{
			username,
			password
			}
				Register
				Validate/sanitise inputs
				- Avoid SQL Attack
				- Ensure it meets complexity requirements
				- Ensure valid email, name etc.
				
				Create user record in DB
				
				Login
				Authenticate user
				Create/handle session
				Return success/fail
					Register
					{
					success: true/false
					}
					
					Login
					{
					success: true/false,
					token: '...'
					}
						POST request to
						
						/register
						
						/login
							Store session information in sessions

Logout process	Log out of the user’s session	AJAX Request when user presses sign out button.	No response body	Update session to remove the logged in state	Return 200 OK on success, return relevant error code otherwise	POST request to /logout	Updates an existing session, removing the logged in state from it (reverts to a visitor)