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
			User				

Sign up	Users can sign up for an account by providing their email and choosing a username and password.	A user input form on the sign up page	JSON formatted AJAX
			request containing:
			{
			"username": "example_user",
			"password": "example_password",
			"email": "example_email"
			}
				Retriving all the username, password and email.
				Then store them in the database. Also check if
				the username already existed in the data base
				or not.
					// first response (if the username already existed)
					res.json({ success: false, message: 'Username already exists.' });
					
					// second response (displaying whether
					they sign up sucessfully or fail)
					
					res.json({ success: true,
					message: 'Signup successfull'
					});
						POST
						/signup
						/home
							We need to check if the username already existed in the system. Send POST request to server, store them in param in server side and check if user name exists? After that response to client (eg. yeah ur username good to go or nah). Process this when sending POST with all the other data.

Log in	Users can log in to the website with their username and password.	Username input field
		Password input field
		On the log-in page
			JSON formatted AJAX
			request containing username
			and password:
			{
			"username": "example_user",
			"password": "example_password"
			}
				Retriving the username and password. Then
				check if they match with an existing account
				in the database.
					// first response (displaying whether
					they sign up sucessfully or fail)
					
					res.json({ success: true,
					message: 'Signup successfull'
					});
					
					// second reponse is to direct logged
					in user to the home page
						POST
						/login
							Store user's username + pass in variable. Then compare with the database using mysql Finally send ajax with json format.
							
							After loggin in, give user a cookie so we can keep track of the user's state
							(if not, they will need to login for every request they send to the server).

Log out	Users can log out of the website.	Button click and session id and
		user_id_cookie/session_token
			JSON formatted AJAX
			request containing:
			{
			"session_id": "session_id"
			}
				Router from the server side that will
				invalidate/destroy user session and redirecting
				user to login page.
					request.session.destroy(function(err) {
					if (err) {
					console.log(err);
					response.status(500).send('Logout failed.');
					} else {
					response.status(200).send('Logout successful.');
					}
					});
						POST
						/logout
							middle ware set up is the same as in login

View home page	Users can view some organisations
	and events from the home page.
		Send GET request to go to the home
		page.
		Then send GET request to parse in the
		organisation at the bottom of the home
		page following the design.
			// request for home page
			xhttp.open("GET", "/home", true);
			xhttp.send();
				Router from server recieve the request and
				respond the client with the home page. Then,
				select three organisations from the database and
				respond them in json format.
					JSON formatted AJAX response containing 3 organisation details:
					[
					{
					"id": "organisation_id",
					"name": "organisation_name",
					"description": "organisation_description",
					"photo": "organisation_photo_url"
					},
					... 3
					]
						GET
						/home	

View all organisation page	Users can view a page displaying
	all the active organisations on the website.
		Display all the organisations
		upon click the organisation page
		by the user
			// request for organisation page
			xhttp.open("GET", "/organisation", true);
			xhttp.send();
				Router from server recieve the request and
				respond the client with the organisation page.
				Then, select all organisations from the database
				and respond them in json format.
					JSON formatted AJAX response containing the all organisation details:
					[
					{
					"id": "organisation_id",
					"name": "organisation_name",
					"description": "organisation_description",
					"photo": "organisation_photo_url"
					},
					... all
					]
						GET
						/organisation	

View organisation (individual)	Users can view specific information about organisations on an organisation's page.	organisation_id	AJAX GET request xhttp.open("GET", "organisation?org_id"=orgnisation_id, true);	Router recieve GET request then proceed to respond the client with organisation page they want to visit. We also recieve some event_id that can be parse into the page to show some events of that particular organisation.	JSON formatted AJAX response containing 1 or 2 event details: [{ "id": "event_id", "name": "event_name", "description": "event_description", "time": "event_timel", "location": "event_location", "photo": "event_photo", "host": "organisation_id" }, ...]	GET	

View joined organisations	Users can view all the organisations they
	have currently joined.
		user_id	JSON formatted AJAX
			request containing:
			{
			"user_id": "user_id"
			}
				Router recieve the user_id from the request when
				the user is in /account. If then procceed to check
				which organisation has this user_id and respond
				back to the client in json format of the organisation's
				details they are in.
					JSON formatted AJAX response containing the organisation they joined:
					[
					{
					"id": "organisation_id",
					"name": "restaurant_name",
					"photo": "organisation_photo_url"
					},
					... all
					]
						POST	We can then parse in the json org to our /account and display them.

Join organisation	Users can join/sign up for organisations
	on the organisation's page. The button
	should change from "join" to "leave".
		user_id	JSON formatted AJAX
			request containing:
			{
			"user_id": "user_id"
			}
				Router recieve the user_id from the request when
				the user is in /account. If then procceed to check
				if the organisation already has this user_id. If yes,
				send response back to the client failed to join. If not,
				insert the user_id into the corresponding organisation_id
					// first response (if already joined)
					res.json({ success: false, message: 'Already Joined.' });
					
					// second response (join successfully)
					res.json({ success: true, message: Join successfull' });
						POST
						/"organisation?org_id"=orgnisation_id	

Leave organisation	Users can leave organisations they have
	joined, losing access to organisation -
	specific posts. The button should change
	from "leave" to "join".
		user_id	JSON formatted AJAX
			request containing:
			{
			"user_id": "user_id"
			}
				Router recieve the user_id from the request.
				It then procceed to check to remove the user_id from the
				table Joined_Organisation.
					res.json({ success: true, message: 'User removed from
					organisation successfully.' });
						POST
						/account	

View all event page	Users can view all the events available on
	the website (they do not have to be part of
	an organisation to view/join events)
		Display all the event upon click the
		event page by the user
			// request for event page
			xhttp.open("GET", "/event", true);
			xhttp.send();
				Router from server recieve the request and
				respond the client with the event page.
				Then, select all events from the database
				and respond them in json format.
					JSON formatted AJAX response containing all events:
					[
					{
					"id": "event_id",
					"name": "event_name",
					"description": "event_description",
					"time": "event_timel",
					"location": "event_location",
					"photo": "event_photo",
					"host": "organisation_id"
					},
					...
					]
						GET
						/event	

View event (individual)	Users can view event-specific information on an event's page.	Event ID, from the event that a user clicks on on the events page	AJAX GET request xhttp.open("GET", "organisation/event?org_id&event_id"= event_id, true);	Router from server recieve the request. Then, select the event from the organisation with the id corresponding to the event_id in the GET request from the database and respond them in json format.	JSON formatted AJAX response containing all event details: [{ "id": "event_id", "name": "event_name", "description": "event_description", "time": "event_timel", "location": "event_location", "photo": "event_photo", "host": "organisation_id" }, ...]	GET /organisation/event	

View rsvp'd events	Users can view ther upcoming events for
	events they've rsvp'd for on the events
	page
		user_id	JSON formatted AJAX
			request containing:
			{
			"user_id": "user_id"
			}
				Server recieve the user_id from the request. Then check
				which events have that user_id and respond back the
				event details that they are rspv.
					JSON formatted AJAX response containing the event they rsvp:
					[
					{
					"id": "organisation_id",
					"name": "restaurant_name",
					"photo": "organisation_photo_url"
					},
					... all
					]
						GET
						/event	

Registering for an event	Users can register for an event of
	their interest. The register button should
	be then displayed as "registered".
		Clicking on the register button	JSON formatted AJAX
			request containing:
			{
			"user_id": "user_id"
			}
				Upon clicking the register button, a GET request will be
				sent to the server. It then check if the user_id already
				registered. If not, the user_id will be inserted to the
				database. A message will be responded to the client
				saying if they register successfully or not.
					// first response (if already registered)
					res.json({ success: false, message: 'Already Registered.' });
					
					// second response (register successfully)
					res.json({ success: true, message: Registered successfully!' });
						POST
						/organisation/event	

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

Unregister event	Users can cancel their event registration. The button should change from "unregister" to "register".	Clicking on the unregister button	JSON formatted AJAX request containing:
			{
			"user_id": "user_id"
			}
				Upon clicking the unregister button, a GET request will be sent to the server. It then check if the user_id already unregistered. If not, the user_id will be deleted to the database. A message will be responded to the client saying if they unregister successfully or not.	// first response (if already unregistered)
					res.json({ success: false, message: 'Already Unregistered.' });
					
					// second response (unregister successfully)
					res.json({ success: true, message: Unregistered successfully!' });
						POST
						/organisation/event	

View all posts	Users can choose to view all posts (public and private) on the posts page ordered ordered from most recent. When user first go to New Feeds page, they will land on the page with option "All Post". The user can see all public posts from all organisations but only private posts from the organisations they are member with.	Clicking on "New Feeds" on the nav bar	// request for New Feeds page
			xhttp.open("GET", "/newfeeds", true);
			xhttp.send();
				The server will check the user_id, then procceeds to send back the public posts from all organisations. Then it will check which organisations the user is member with and respond back with the private post from these organisations.	JSON formatted AJAX response of public post details:
					[
					{
					"id": "post_id",
					"description": "post_description",
					"photo": "post_photo_url",
					"date": "post_date_and_time",
					"status": "public"
					},
					]
					
					JSON formatted AJAX response of private post details:
					[
					{
					"id": "post_id",
					"description": "post_description",
					"photo": "post_photo_url",
					"date": "post_date_and_time",
					"status": "private",
					"organisation": "organisation_id"
					},
					]
						GET
						/newfeeds	

View private posts	Users can filter posts to only view private posts from their organisation on the posts page ordered by most recent.	When choosing "Organisation" on the left bar	// request for New Feeds page
			xhttp.open("GET", "/newfeeds", true);
			xhttp.send();
				The server will check the user_id, then procceeds to send back the private posts from the organisations they have joined.	JSON formatted AJAX response of private post details:
					[
					{
					"id": "post_id",
					"description": "post_description",
					"photo": "post_photo_url",
					"date": "post_date_and_time",
					"status": "private",
					"organisation": "organisation_id"
					},
					]
						GET
						/newfeeds	

Edit user details	Users can edit and update their information from their profile. They can change their first name, last name, or email.	Input fields from the accounting setting page.	JSON formatted AJAX request containing:
			{
			"first_name": "user_first_name",
			"last_name": "user_last_name",
			"email" : "user_email",
			}
				The server will alter the user's first and last name and email as requested in the database. It also needs to check if the edited email matches any of the existing email. If yes, it will send a message indicating that edit failed (because email is unique for each user). If update succesfully, the server also send back updated user details.	// can not change to an already existing email
					res.json({ success: false, message: 'Email used!' });
					
					// second response (Update successfully)
					res.json({ success: true, message: Update successfully!' });
					
					// sending back user details
					JSON formatted AJAX response of private post details:
					[
					{
					"id": "user_id",
					"first_name": "first_name",
					"last_name": "last_name",
					"email": "email",
					},
					]
						POST
						/profile	

Load user details	Users can view their personal information on their profile page.	When going to profile page	// request for New Feeds page
			xhttp.open("GET", "/profile", true);
			xhttp.send();
				The user will retrieve and send back user information from the data base.	// sending back user details
					JSON formatted AJAX response of private post details:
					[
					{
					"id": "user_id",
					"first_name": "first_name",
					"last_name": "last_name",
					"email": "email",
					},
					]
						POST
						/profile	

Manager							
In addition to the features above.. managers have the added interactions:							

Create event	Managers can create events from their profile, with name, date, time, location, an image and description.	Input fields from the "Create Event" under "Manager Tools"	JSON formatted AJAX request containing:
			{
			"name": "event_name",
			"date": "event_date",
			"location": "event_location",
			"photo": "event_photo",
			"description": "event_description",
			"host": "organisation_id"
			}
				The server recieve the request with all the new event details, it will then check if the event existed (using event_name). If not, the server will create new row in the event table in the database and send back a sucess message. It will then send back the new event details to be parsed into the event page if a user visit.	// event already existed
					res.json({ success: false, message: 'Event already existed!' });
					
					// second response (create successfully)
					res.json({ success: true, message: Event created successfully!' });
					
					JSON formatted AJAX request containing event:
					{
					"name": "event_name",
					"date": "event_date",
					"location": "event_location",
					"photo": "event_photo",
					"description": "event_description",
					"host": "organisation_id",
					}
						POST
						/manager/event	

Update event	Managers can update event details from their profile.	Inputs field from when click on the "Manage Event" on the top right.	JSON formatted AJAX request containing:
			{
			"name": "event_name",
			"date": "event_date",
			"location": "event_location",
			"photo": "event_photo",
			"description": "event_description",
			"host": "organisation_id"
			}
				The server recieve the request with all the new event details, it will then check which of the components were changed and implement them in the database. When done, the server will send back sucess message. The manager now can see the changes with he/she click on "Update Event" on the left bar menu again.	// returning all the newly updated event details so that it can overwrite
					// the old one in "Update Event"
					
					JSON formatted AJAX request containing new event details:
					{
					"name": "event_name",
					"date": "event_date",
					"location": "event_location",
					"photo": "event_photo",
					"description": "event_description",
					"host": "organisation_id"
					}
						POST
						/manager/event	

View Created Events	Managers can view the events that have been created by other members of the organisation	When clicking on "Update Event" under "Manager Tools"	// request for "Update Event" page
			xhttp.open("GET", "/manager/event", true);
			xhttp.send();
				The server recieve the request and respond back with all the event details from the database (including the users).	JSON formatted AJAX request containing all events:
					{
					"name": "event_name",
					"date": "event_date",
					"location": "event_location",
					"photo": "event_photo",
					"description": "event_description",
					"host": "organisation_id",
					"user": "user_id"
					}
						GET	

See users who have rsvp'd	Managers can view event attendees (users who have registered for events) when updating events	When clicking on "Update Event" under "Manager Tools"	// request for "Update Event" page
			xhttp.open("GET", "/manager/organisation/event", true);
			xhttp.send();
				The server recieve the request and respond back with all the event details from the database (including the users).	JSON formatted AJAX request containing all events:
					{
					"name": "event_name",
					"date": "event_date",
					"location": "event_location",
					"photo": "event_photo",
					"description": "event_description",
					"host": "organisation_id",
					"user": "user_id"
					}
						GET	

Remove users from event	Managers can remove users from an event. Users' names will disappear from the event if successful.	Click the "remove" button to remove a specific user as desired.	JSON formatted AJAX request containing:
			{
			"user": "user_id",
			"event": "event_id"
			}
				The server will check for the user_id and the event_id in the database. If they valid, it will drop that row. The server respond with a new list of user rspv for an event (after deleteing the wanted user) to update the page.	// sending back new list of user who rsvp for that event
					JSON formatted AJAX response of private post details:
					[
					{
					"id": "user_id",
					"event": "event_id",
					},
					]
						POST	

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

View Organisation members	Managers can view all the members of an organisation	By choosing "Manage Organisation" under "Manager Tools" and sending the organisation_id	JSON formatted AJAX request containing:
			{
			"organisation": "organisaiton_id"
			}
				The server recieve the request and check the orgnisation_id in the Joined_Organisation table in the database. It then respond back all the user_id, who matches the orgnisation_id in the table.	// sending back new list of user join the organisation
					JSON formatted AJAX response of private post details:
					[
					{
					"id": "user_id",
					"organisation": "organisation_id
					},
					]
						POST	

Remove Organisation Members	Managers can remove members of an organisation. Once removed, the user should vanish from the organisation	Click the "remove" button to remove a specific user as desired.	JSON formatted AJAX request containing:
			{
			"user": "user_id",
			"organisation": "organisaiton_id"
			}
				The server will check for the user_id and the organisation_id in the database. If they valid, it will drop that row. The server respond with a new list of user of an organisation (after deleteing the wanted user) to update the page.	// sending back a new list of user who joined that organisation
					JSON formatted AJAX response of private post details:
					[
					{
					"id": "user_id",
					"organisation": "organisation_id
					},
					]
						POST	

Post updates (public/private)	Managers can make posts from their profile, with a photo, title and caption. They can choose to make them public or with the buttons on the page.	By choosing "Create Post" under "Manager Tools" and sending the organisation_id	JSON formatted AJAX request containing:
			{
			"title": "post_title",
			"date": "post_date",
			"photo": "post_photo",
			"description": "post_description",
			"status": "public/private",
			"organisation": "organisation_id"
			}
				The server recieve the request with all the post details, it will then check if the post existed (using post_title). If not, the router will create new row in the post table in the database and send back a sucess message. It will also then send back the new event details to be parsed into the newfeeds page if a user visit.	JSON formatted AJAX request containing:
					{
					"title": "post_title",
					"date": "post_date",
					"photo": "post_photo",
					"description": "post_description",
					"status": "public/private",
					"organisation": "organisation_id"
					}
						POST
						/newfeeds	

Admin							

View organisations (to edit)	Admins can view all organisations they are admins for.	When clicking on "Manage Organisation" under "Admin Tools"	// request for "Manage Organisation" page
			xhttp.open("GET", "/admin/organisation", true);
			xhttp.send();
				The server recieve the request and send back a page with all the organisation names exist in the database.	JSON formatted AJAX request containing all organisation names:
					{
					"organisation": "organisation_name"
					}
						GET
						/admin/organisation	

View organisation users	Admins can view all users in the organisation they choose	From "Manage Organisation", when click on a specific organisation, the admin can view all the users who joined that organisation.	// request for "Manage Organisation" page
			xhttp.open("POST", "/admin/organisation", true);
			xhttp.send();
			
			JSON formatted AJAX request containing all organisation names:
			{
			"organisation": "organisation_id"
			}
				The server recieve the request and send back a page with all the user_id belong to the specified organisation in the database.	JSON formatted AJAX request containing all user_id:
					{
					"user": "user_id"
					}
						POST
						/admin/organisation	

View organisation users (individual)	Admins can view the user information of organisation members	From "Manage Users", when click on a specific user (edit button), the admin can view all of that particular user's details	// request for "Manage Organisation" page
			xhttp.open("POST", "/admin/user", true);
			xhttp.send();
			
			JSON formatted AJAX request containing the user_id:
			{
			"user": "user_id"
			}
				The server recieve the request and send back with all the user's details from the database.	JSON formatted AJAX request containing all user details :
					{
					"user": "user_id",
					"first_name": "user_first_name",
					"last_name": "user_last_name",
					"email" : "user_email"
					}
						POST
						/admin/user	

Edit user information	Admins can edit user information	From "Manage Users", when click on a specific user (edit button), the admin can view all of that particular user's details	// request for "Manage Organisation" page
			xhttp.open("POST", "/admin/user", true);
			xhttp.send();
			
			JSON formatted AJAX request containing all user details :
			{
			"user": "user_id",
			"first_name": "new_user_first_name",
			"last_name": "new_user_last_name",
			"email": "new_user_email"
			}
				The server recieve the reques with all the new user details. It will overwrite the currently existing user details in the database and send back the new one if the admin go to a page that can see the user information.	JSON formatted AJAX request containing the updated user details :
					{
					"user": "user_id",
					"first_name": "user_first_name",
					"last_name": "user_last_name",
					"email" : "user_email"
					}
						POST
						/admin/user	

Delete users	Admins can delete users from the website	From "Manage Users", when click "remove" button.	JSON formatted AJAX request containing all user details :
			{
			"user": "user_id"
			}
				The server recieve the user_id that needs to be deleted. It will then go and delete that user_id in the database and send back all the users after deletion if the admin go to "Manage User" to view the all the current users after deletion.	JSON formatted AJAX request containing all user_id after deletion:
					{
					"user": "user_id"
					}
						POST
						/admin/user	

Create organisation	Admins can create new organisations with a photo, name, and description	From "Creat Organisation"	JSON formatted AJAX response containing the all organisation details:
			[
			{
			"name": "organisation_name",
			"description": "organisation_description",
			"photo": "organisation_photo_url"
			}
			]
				The server recieve the request with all the new organisation details, it will then check if the organisation existed (using organisation_name). If not, the server will create new row in the organisation table in the database and send back a sucess message. It will then send back the new organisation details (including organisation_id to be parsed into the organisation page if a user visit.	// organisation already existed
					res.json({ success: false, message: 'Organisation already existed!' });
					
					// second response (created successfully)
					res.json({ success: true, message: Organisastion created successfully!' });
					
					JSON formatted AJAX response containing the all organisation details:
					[
					{
					"id": "organisation_id",
					"name": "new_organisation_name",
					"description": "new_organisation_description",
					"photo": "new_organisation_photo_url"
					}
					]
						POST
						/admin/organisation	

Edit Organisation	Admins can edit organisation descriptions, names and images.	From "Manage Organisation"	JSON formatted AJAX response containing the new organisation details:
			[
			{
			"name": "new_organisation_name",
			"description": "new_organisation_description",
			"photo": "new_organisation_photo_url"
			}
			]
				The server recieve the reques with all the new organisation details. It will overwrite the currently existing organisation details in the database and send back the new one if the admin go to a page that can see the organisation information.	JSON formatted AJAX response containing the updated organisation details:
					[
					{
					"id": "organisation_id",
					"name": "organisation_name",
					"description": "organisation_description",
					"photo": "organisation_photo_url"
					}
					]
						POST
						/admin/organisation	

Load User Roles	Admins can view the roles of each user.	From "Manage User"	JSON formatted AJAX request containing user :
			{
			"user": "user_id",
			}
				The server recieve the user_id and then check it in the "Role_User" tabel in the database to return the corresponding role_id/role_name of that user.	JSON formatted AJAX request containing user and role:
					{
					"user": "user_id",
					"role": "role_name"
					}
						POST
						/admin/user	

Assign User Roles	Admins can change the role of each user to user, manager or admin	From "Manage User"	JSON formatted AJAX request containing user and role:
			{
			"user": "user_id",
			"role": "role_id"
			}
				The server recieve the new role_id of the user. It will then go alter the "Role_User" tabel in the database and respond with the new role_name of that user	JSON formatted AJAX request containing user and role:
					{
					"user": "user_id",
					"role": "new_role_name"
					}
						POST
						/admin/user	