Group Details		
Group Member	**Contribution**	**Discussion Notes**
1. Afra Nawar	- Decided on architecture diagram type - Suggested we go with JS for our backend based on our diagram and project's requirements - Suggested we integrate Google Maps for event locations as third-party (will not be using) - Researched and created the rough draft of our layered (N-tier) architecture	- Discussed w/ TA after meeting for help regarding Step 3 - Discussed w/ TA for diagram confirmations
2. Jennifer Nguyen	- Created spreadsheet for organizing group's decision for project - Created description of user experience and key functionalities - Created and conveyed how technology stack will interact with each other - Created final draft for our Layered Architecture Diagram - Created final draft in a Google Doc to officially turn in as Assignment 1 - Created GitHub repository for group assignment tracking and turn-ins	- Reference to Spreadsheet - Will upload everything as pdf to Github @Jenblr
3. Jonathan Hsueh	- Helped decided on front-end + back-end technologies - Decided on Agile for our methodology based on our project assignment progressions	
4. Sofia Davila	- Decided on our non-profit organization type = animal shelter - Helped create how volunteers would be sectioned to be able to match to events - Created another rough draft of our layered (N-tier) architecture diagram	

Step 1: Initial Thoughts		
Aspect	**Details**	**Answer**
Organization	Type	**Animal shelter** - The shelter will host a variety of animals that need to be rehomed. - The shelter will need a variety of volunteers that can help take care of the animals needs (e.g. bathing, feeding, grooming, clean-ups, etc) and paperwork (adoptions, animal information, etc). - The volunteers would be sectioned off into tiers for 'match-making'. The higher tier they are, the more likely they would interact with animals directly rather than simply deal with clean-ups and paper-work. - The volunteer tiers will be decided on a number of factors = animal experience history, availability, etc.
User Experience	How will users (volunteers and administrators) interact with the application?	**Administrators:** - Admin will be able to create and manage volunteer events at the animal shelter. - Admin will also be able to look-up and view every volunteer's profile and match volunteers to tasks.
		Volunteers: The volunteers will be directed to the home page where they can see the organization's mission statement (and possibly a FAQ section). Then can see company logo on top w/ task bar, and then current page from taskbar —---------- *NOT Registered:* 1. Sign up through 'Log In / Sign Up' button on top right of page 2. Complete info criteria (e.g. name, location, skills & preferences, availability) = has to be completed before signing up for any events 3. Now eligible to sign up for events *ALREADY Registered:* 1. Log in through 'Log In / Sign Up' button on top right of page 2. Directed back to home page 3. View taskbar where they can be redirected to: - Home = What the company is about - Calendar = View what events are coming up that they can sign up for - Event Sign-Up = Has section for matching the user based on their profile and event requirement(s). Also has another section for free sign-up regardless

		- Manage Profile = Can edit location, skills, preferences, availability, etc
Key Functionalities	What are the essential features the application must have?	**Login and Registration:** Users can sign up or login with email verification
		User Profile Management: Allow users to update their profiles with their location, skills, event preferences, and availability
		Event Management: Allow admin to create and manage volunteer events with each event's specific requirements - Create info about event = location, name, how many volunteers needed
		Volunteer Matching: Have a section dedicated to matching each volunteer to available events - Time availability / schedule - Experience with animal (tiered)
		Notification System: USERS: - Event Invitations = notify users when they are invited/assigned to an event - Reminders = Remind users of upcoming events they are assigned to - Updates = Inform users about changes to their assigned event - General Announcements = Read news or updates from organization ADMIN: - Notified when user signs up for an event
		Volunteer History: Each user can view their own history. Admin can view all users' history.
Technology Stack	What technologies might you use for front-end, back-end, database, and other components?	**FRONT-END:** React.js, HTML/CSS
		BACK-END: Node.js w/ Express.js
		DATABASE: PostgreSQL
		NOTIFICATIONS: WebSocket

Step 2: Development Methodology		
Aspect	**Details**	**Answer**
Methodology	Which development methodology will we use?	Agile
Reason	Why are we choosing this methodology?	- Agile is quite useful for its adaptability, meaning that we are able to build and deploy the project incrementally - This is crucial because we may need to change or add new features along the way
Project Management	How will this methodology help manage the project effectively?	- We are able to change the project based on feedback from the non-profit organization and users (volunteers) to make improvements towards our project - We can also develop the volunteer system incrementally, starting with basic functionalities such as signing up and creating events

Step 3: High-Level Design / Architecture		
Aspect	**Details**	**Answer**
Diagram	Illustrate overall structure of application	Layered Architecture Diagram
Front-end	Frameworks and libraries to be used	React.js, HTML/CSS
Back-end	Technologies for API and server-side logic	Node.js w/ Express.js
Database	Type of database and schema details	PostgreSQL
Notification	Type of website notification display for UI	WebSocket
Interactions	How components will interact with each other	**1. Front-end (React.js)** ***USER INTERFACE (UI):*** - The users (whether admin or volunteer) will interact with our application through React.js in the front-end. - The front-end will be responsible for displaying data (volunteer events, volunteer history, profile, etc), capturing user input (admin event creation, volunteer profile info, etc), and providing a dynamic and responsive user experience. ***API CALLS:*** - The React.js front-end will send HTTP requests (GET, POST, PUT, DELETE) to the back-end through RESTful APIs. - EXAMPLE = When a volunteer user logs into the application, the front-end will send a POST request tagged with the user's credentials back to the back-end API.
		2. Back-end (Node.js with Express.js) ***API Layer:*** - The Express.js framework will be responsible for incoming API requests from the front-end - Each API endpoint corresponds to a specific actions (e.g. admin creating a new event, obtaining users (volunteers') profiles, matching volunteers to events, etc) ***Server Logic Layer:*** - This layer will process data received from the front-end, determining how it will interact with the database

		- EXAMPLE = Request received to match user (volunteer) to an event = back-end will process the request based on rules (e.g. skills, preferences, availability)
		3. Database (PostgreSQL) ***Data Storage:*** - The application will utilize PostgreSQL to store structured data (e.g. user profile information, event details, volunteer history, etc) - The database schema will include tables such as 'Users', 'Events', 'VolunteerMatches', 'Notifications', etc ***Database Interaction:*** - The back-end will use Object-Relational Mapping (ORM) to handle user registration, profile management, and event creation - The back-end will also use Direct SQL Queries for volunteer matching query because volunteer matching will require joining multiple tables
	Summary of Interaction	**User Action → Front-End:** - Users will interact with React.js in the front-end, resulting in triggering action (e.g. logging in/signing up, completing profile, signing up for events, viewing notification, etc) **Front-end → Back-end:** - React.js (front-end) sends HTTP requests to Node.js (Back-end) using Express.js **Back-end → Database:** - Express.js back-end will process HTTP requests and will then interact with PostgreSQL database **Back-end → Front-end:** - After processing HTTP requests, the back-end will send a response back to the front-end, resulting in an update in the UI based on the response (e.g. event creation, user profile completion, etc). **WebSocket Notifications:** - In addition to HTTP responses, the back-end will use Websockets to send notification to the front-end for the user (e.g. when a user is assigned to an event, event reminders, etc), where React.js will update the UI to display a new notification

Further Insight into our Layered Diagram	
Aspect	**Details**
User Action → Client Layer	Our users (the volunteers and admin) will interact the UI we created from them using HTML/CSS (these files will be used to style it like the website under 'Front-end inspiration')
Client Layer → Presentation Layer	The Client Layer would then send user input to React.js, the file that will update the UI (HTML/CSS)
Presentation Layer → Service Layer	The Presentation Layer would then utilize React.js to send HTTP requests to Express.js API in our back-end for processing, or create a WebSocket connection for notifications for the user
Service Layer → Logic Layer	The Service Layer would then use Express.js API to pass the request from the Presentation Layer to the Logic Layer where Node.js lies to further process the business logic (e.g. volunteer matching) and Express.js (Middleware) (e.g. error handling, user authentication)
Logic Layer → Data Layer	The Logic Layer would then interact with the Data Layer, which could either bc the ORM part for CRUD operations (e.g. user registration) or Directly SQL for more complex tasks (e.g. volunteer matching = need different components)
Data Layer → Persistence Layer	The Data Layer would then interact with the Persistence Layer, where our relational database (PostgreSQL) lies, to store or retrieve data (e.g. fetching event data, storing new user information/updated user information, etc)

Layered Diagram of Application Structure