Software Design - HW3
Group 2
GitHub: github.com/yoshadmun/VolunteerActivities

Design and Development Methodology:
Our methodology for this assignment has remained Agile Development. We continued using this development methodology because the modular design and incremental development of features helped us with collaboration by outlining and working on different modules simultaneously. This also allows us to gradually incorporate our backend into our frontend, and iterative development allows for continuous improvement as we go along with the project.

Backend Technologies:

- Node.js: NPM (Node Package Manager) contains the libraries and modules which allowed us to speed up development by a lot. For example, modules like express, express-validator, and others which very much allowed us to use Express.js, auth0, and react wrapper such as @auth0/auth0-react.

- Express.js: Express was used to define routes for handling different HTTP requests. The middleware pattern in Express.js allowed us to address separating concerns, such as authentication, logging, validation, and error handling. An example is how authentication middleware was used to protect routes, and validation middleware (like express-validator) was used to validate incoming data. This modularity led to cleaner and more maintainable code. Furthermore, it was routing in Express.js that helped with designing routes in files like eventRoutes.js by separating concerns (like how routes related to events are managed in eventRoutes.js file, while routes related to volunteers are managed in volunteerRoutes.js file.

- Data Management: Data management in backend assignment is performed using CRUD (create, read, update, and delete actions) operations across different controllers.
    a. Create Operations: An example of create operation is when we use the function createEvents which takes the event data from the request body, generates, new ID, and adds it to the events array. A similar approach is used to assign events to volunteers where we run a check for existing assignment before assignment, sending notification and returning a success message with status code 200 or OK.

b. Read Operations: An example of read operations is when we use the function getAssignedEvents which finds the volunteer ID in the data array and sends the list of events assigned to the volunteer. Again, error handling is included in the function.
c. Update Operations: An example of update operations is when we use the function completeEvent which marks an event as completed for a volunteer. This is done by obtaining userId and eventId from the request body and finding the volunteer by their ID. The assigned event is moved from assignedEvents to completedEvents.
d. Delete Operations: An example of delete operations is when we use the function removeEvent which deactivates an event and also updates a volunteers' assignment accordingly. The event is found through event ID and the status is simply updated, thus removing the event from the assigned event list for all volunteers.

- Error Handling: When encountered with an error, such as if a volunteer is not found for a specific update, we return status 404 (not found) and otherwise we return status 202 and update the volunteer as needed. A 404 bad request status is also returned if, for example, a volunteer is already assigned to an event.

- Express-Validator: This was used where we define validation rules for event creation and updating, and volunteer profile creation and updating. We also implemented a middleware function to check for validation errors and send responses if any are found, for example: "Full Name is required".

- Auth0: We used useAuth0 to handle user login, logout, and authorization purposes to distinguish between Admin dashboard and User dashboard. Auth0 was also used to access the authenticated user's information based on user.sub ID, and send reminder notifications to users. This was imported from the @auth0/auth0-react library, and was also used because of the available resources regarding how it functions.

- React Hooks: Aside from Auth0, we used useEffect and useState for efficient state management and side effect handling in functional components, leading to more readable and maintainable code. useState allowed us to manage state for notification data, pagination states, loading status, matched events, assigned events, etc. Meanwhile, useEffect allowed us to fetch notification data from the

- Axios: Axios was used in the code to handle HTTP requests between the frontend React application and the backend server. It will fetch notifications for authenticated users, delete notifications when requested, other CRUD requests, and response interceptions. This was chosen for its ease to use and available resources for us to look into.

Team Contributions:

Group Member Name	**What is your contribution?**
Matin Amoozadeh	Notification controller and routes, history controller, package files, and server.
Thinh Pham	User + Admin functionalities, connecting frontend to backend, worked on additional controllers and routes.
Shadmun Talukder Shahed	Volunteer + Event assignment, connecting frontend to backend wrote the documentation.

Aside from individual contributions. We had several online team meetings, and this allowed us to collaboratively decide on the CSS global styling which allowed for consistent styling in-between pages.