Objective:
As a 'Backend Developer' at Jarurat Care NGO, you will play key role in working with developing a backend API using Node.js (Express) and MongoDB. You will implement various CRUD operations (Create, Read, Update, Delete), enforce role-based access control, and include input validation and authentication mechanisms.

Task Breakdown:
Set up a Node.js Application with Express:

Initialize a Node.js project using Express.js.
Set up MongoDB as the database to store and manage data.
CRUD API Implementation:

POST /create - Create a new resource and store it in the MongoDB database.
GET /all - Retrieve all records from the database.
GET /byId/:id - Retrieve a single record based on its unique ID.
PUT /update/:id - Update an existing record by its ID.
DELETE /delete/:id - Delete a record by its ID.
Role-based Access Control:

Implement two user roles: Admin and User.
Admin: Can access all routes (Create, Read, Update, Delete).
User: Can access only GET routes (/all, /byId/:id).
Create middleware to validate the role of the user and restrict access based on roles.
Field Validation:

Before storing any data in MongoDB, validate all required fields (e.g., proper data types, required fields, unique fields, etc.).
Use a validation library like Joi or express-validator to ensure data integrity.
Optional - Authentication & Token Management:

Implement user authentication by creating a JWT token that stores the user's ID and role.
Create middleware to validate the JWT token for protected routes.
Encrypt user passwords before saving them to the database (e.g., using bcrypt or argon2).
Instructions:
Create the following routes:

POST /create - Create a new entry.
GET /all - Get all entries.
GET /byId/:id - Get an entry by its ID.
PUT /update/:id - Update an entry by its ID.
DELETE /delete/:id - Delete an entry by its ID.
Role Middleware:

Implement a middleware function to check for user roles in the request, and ensure that only the admin role can access the create, update, and delete routes.
Validation Middleware:

Use middleware to validate incoming requests (e.g., data formats, required fields).
Example: Email should be in a valid format, strings should not exceed 255 characters, etc.
Optional Authentication:

Create a route for user registration and login that generates a JWT token.
The token should be sent in the Authorization header as a Bearer token.
Use middleware to verify the token on protected routes.

Deliverables:
Provide the URL link to your live website on VERCEL (where the API can be accessed)

A PDF containing:
Project Overview
Screenshots of development (code editor, postman requests, etc.)
Code snippets for key logic implementations.