

**DBMS MINI PROJECT**

NSS MANAGEMENT SYSTEM

**Team no:** 2

**Member 1:** NIKAASH T K (2022103035)

**Member 2:** ANAND KARTHIKEYAN S (2022103305) **Degree:** B.E CSE

**Semester:** 4

**Batch:** P

**Subject code:** CS6106

**Subject name:** Database Management Systems **Submitted on:** 23/05/2024

**Submitted to:** Dr.S.Renugadevi, Department of Computer Science and Engineering, CEG, Anna University.

**PROBLEM STATEMENT:**

To develop a comprehensive database management system for NSS units within a specific college (institution) to enhance organizational efficiency and foster community engagement.

**GENERAL INTRODUCTION:**

The National Service Scheme (NSS) is a student-centred program aimed at fostering social responsibility and community engagement among young individuals. Within educational institutions, NSS units play a crucial role in coordinating various social service activities and events. This project is a comprehensive database management system for managing the operations of NSS units within a specific college.

**PROJECT ABSTRACT:**

This project aims to streamline the management of NSS activities by designing and implementing a robust database system. The system will facilitate the management of records and details of the organization as well as efficient coordination of NSS activities. By providing a centralized platform for managing volunteer data and activities, the database system seeks to optimize resource allocation, improve communication, and strengthen the impact of NSS efforts within the college community.

The manual maintenance of records in an organization requires a lot of manual work and a lot of time. It is also difficult to maintain hard copies of records. Due to maintenance limitations tracking historical data is not possible beyond an extent. And the retrieval of data is also not very effective. Even when data is stored in file systems the retrieval methods are deliberate. And thus, using a database to store all the data related to an organization which is NSS is an outstanding and apt solution. In a database, it is possible to retrieve data easily and effectively, modifications can be done effortlessly, and we can keep track of old data in archive for future reference. A perfect user interface for this DBMS project makes all the functionalities easier and even more convenient.

This DBMS will facilitate efficient organization, coordination, and tracking of volunteer activities, events, and participation within NSS units. Key requirements include registration and management of volunteers, assignment and management of roles within NSS units, tracking of volunteer activities, events, and participation, managing program officers overseeing NSS units, and integrating active alumni into NSS activities and events.

**TECHNOLOGIES USED:**

**Frontend Technologies:** HTML, CSS, JavaScript, React.js

**Backend Technologies:** Node.js, Express.js,

**Database:** PostgreSQL *(cloud database provided by Neon)*

This web application is built by frontend and backend technologies which are used by most professional developers of this time. *(Source: StackOverflow Developer Survey 2023)*

**ENTITIES:**

1. Unit (Unit No, P.O. ID, General Secretory, Joint Secretory,                 No of Volunteers, Manuals conducted)

2. Volunteers (Volunteer ID, Unit No,Name,DOB, Gender, Email ID, Joining Date, Year of Study, Password)

3. Program Officers (P.O. ID, Name, Designation, Department, Email ID)

4. Alumni (Alumni ID, Name, Unit No, Year of Passing, Email ID)

5. Manuals (Manual ID, Unit No*,* Theme, Description, Date, Duration, Location, Lead Organizer ID)

**OTHER TABLES:**

1. Contact (Contact ID, Phone no) #multivalued attribute

2. Participations (VolunteerID, Manual ID) #relationship

3. Volunteer Roles (Volunteer ID, Role ID) #relationship

4. Admin (Admin ID, Name, Password) #Overall admin

5. Roles (Role ID, Role Title) #role reference

**CONSTRAINTS:**

**Foreign key constraints:**

1. Unit (Unit No) — >Volunteer (Unit No), Manuals (Unit No), Alumni (Unit No)

2. Volunteers (Volunteer ID) — > Participation (Volunteer ID), Volunteer Roles (Volunteer ID), Manuals (Lead Organizer ID)

3. Program Officers (P.O ID) — > Unit (P.O ID)

**Unique Constraints:**

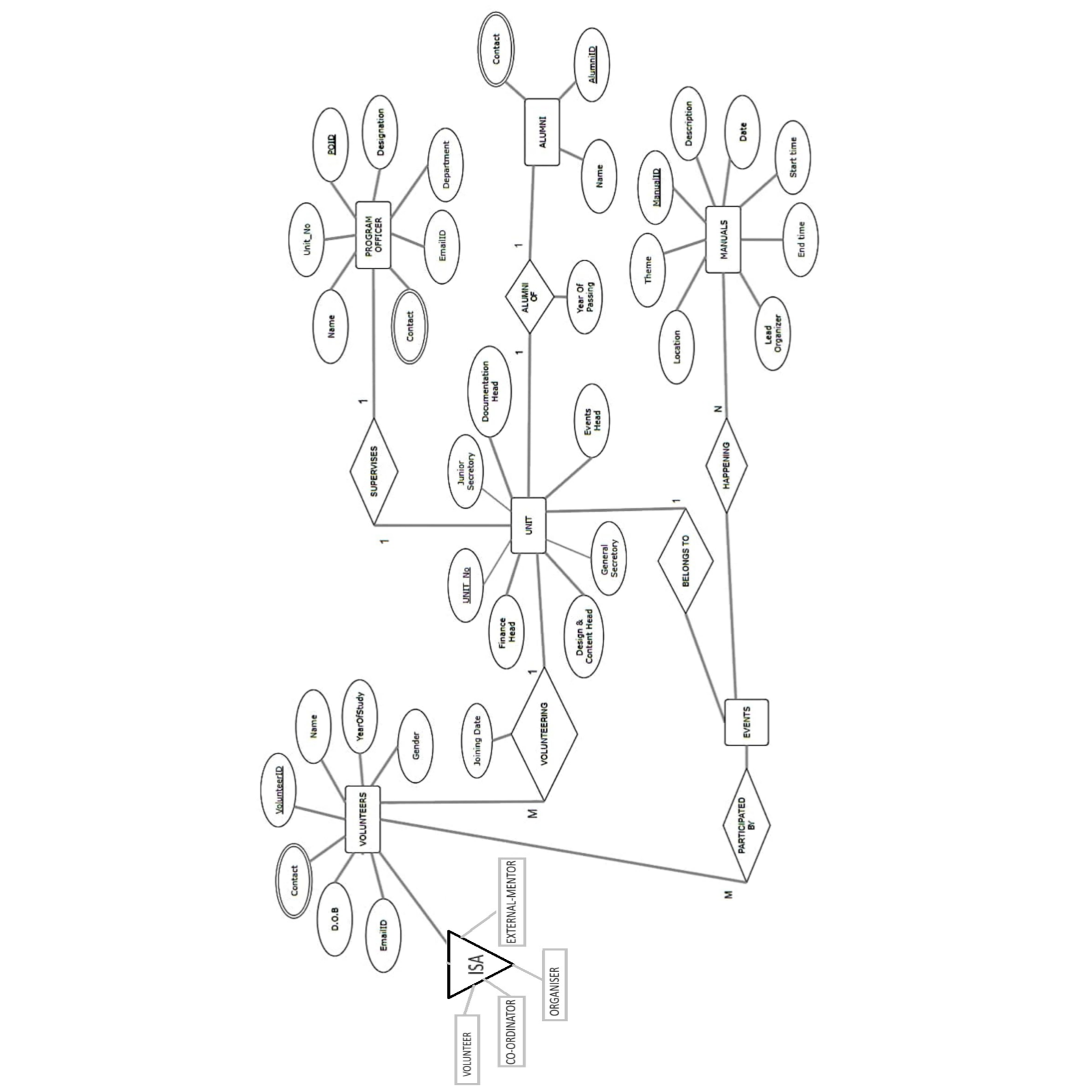
1. Volunteer (Email ID)

 2. Program Officers (Email ID)

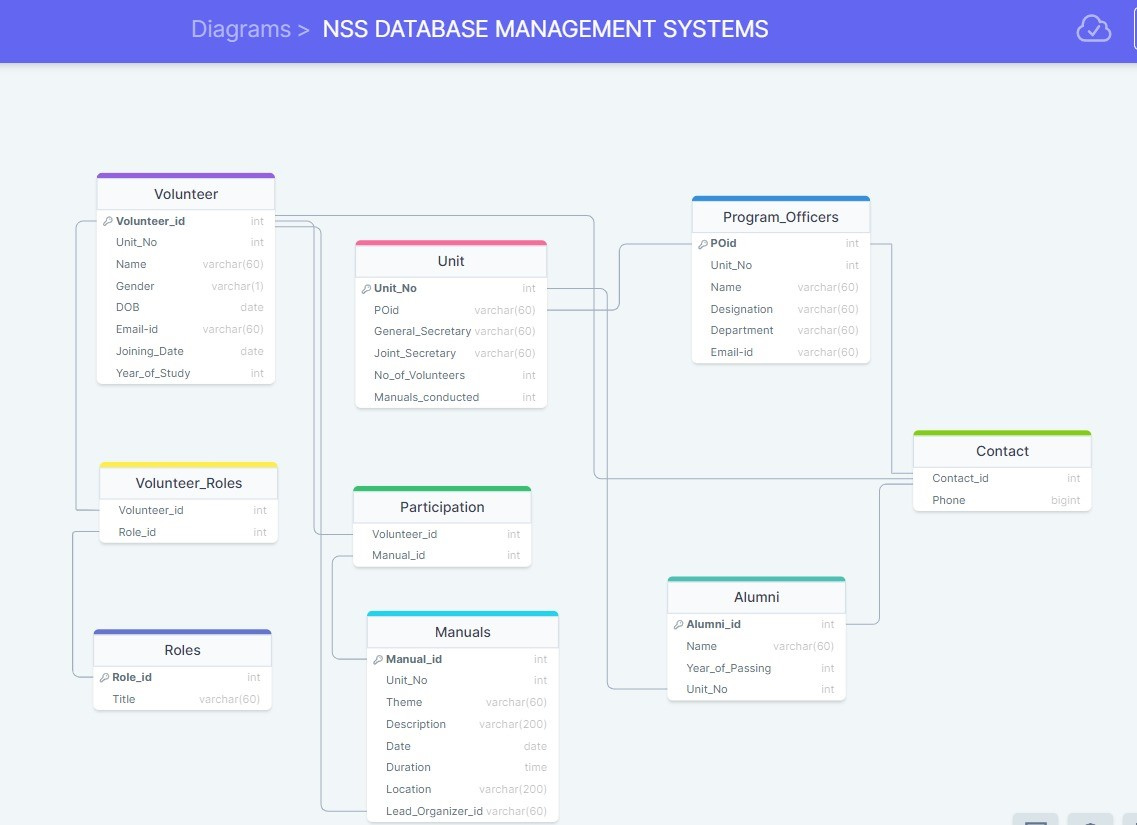
3. Alumni (Email ID)

4. Participations (VolunteerID, Manual ID)

5. Volunteer Roles (Volunteer ID, Role ID)

**ER DIAGRAM:**

**RELATIONAL DATABASE MODEL:**



**DATABASE ORGANIZATION:**

**Functions:**

**1)** CREATE OR REPLACE FUNCTION move\_to\_alumni()

RETURNS TRIGGER AS $$

DECLARE

years\_out INT;

BEGIN

years\_out := EXTRACT(YEAR FROM CURRENT\_DATE);

IF NEW."current\_year" > 4 THEN

INSERT INTO alumni ("alumni\_id", "year\_of\_passing", "name", "unit\_no","email")

VALUES (NEW."volunteer\_id", years\_out, NEW."name", NEW."unit\_no",NEW."email");

delete from volunteers where volunteer\_id = new.volunteer\_id;

END IF;

RETURN NULL;

END;

$$ LANGUAGE plpgsql;

**2)** CREATE OR REPLACE FUNCTION update\_no\_of\_volunteers()

RETURNS TRIGGER AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

UPDATE unit SET no\_of\_volunteers = no\_of\_volunteers + 1 WHERE unit\_no = NEW.unit\_no;

ELSIF (TG\_OP = 'DELETE') THEN

UPDATE unit SET no\_of\_volunteers = no\_of\_volunteers - 1 WHERE unit\_no = OLD.unit\_no;

ELSIF (TG\_OP = 'UPDATE') THEN

IF (NEW.unit\_no <> OLD.unit\_no) THEN

UPDATE unit SET no\_of\_volunteers = no\_of\_volunteers + 1 WHERE unit\_no = NEW.unit\_no;

UPDATE unit SET no\_of\_volunteers = no\_of\_volunteers - 1 WHERE unit\_no = OLD.unit\_no;

END IF;

END IF;

RETURN NULL;

END;

$$ LANGUAGE plpgsql;

**3)** CREATE OR REPLACE FUNCTION assign\_unit\_no() RETURNS TRIGGER AS $$

DECLARE

max\_unit\_no\_m INT;

max\_unit\_no\_f INT;

new\_volunteerid INT;

year\_of\_study INT;

year\_dob INT;

BEGIN

SELECT unit\_no INTO max\_unit\_no\_m FROM volunteers WHERE volunteer\_id = (SELECT MAX(volunteer\_id) FROM volunteers WHERE gender = 'M');

SELECT unit\_no INTO max\_unit\_no\_f FROM volunteers WHERE volunteer\_id = (SELECT MAX(volunteer\_id) FROM volunteers WHERE gender = 'F');

SELECT MAX(volunteer\_id) + 1 INTO new\_volunteerid FROM volunteers;

IF NEW.gender = 'M' AND ((max\_unit\_no\_m IS NULL) OR (max\_unit\_no\_m = 3)) THEN

NEW.unit\_no := 1;

ELSIF NEW.gender = 'M' THEN

NEW.unit\_no := max\_unit\_no\_m + 1;

ELSIF NEW.gender = 'F' AND ((max\_unit\_no\_f) IS NULL OR (max\_unit\_no\_f = 3)) THEN

NEW.unit\_no := 1;

ELSIF NEW.gender = 'F' THEN

NEW.unit\_no := max\_unit\_no\_f + 1;

END IF;

NEW.joining\_date := TO\_CHAR(CURRENT\_DATE, 'YYYY-MM-DD');

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

**4)** CREATE OR REPLACE FUNCTION role\_id()

RETURNS TRIGGER AS $$

begin

insert into volunteer\_roles(volunteer\_id,role\_id) values (NEW.volunteer\_id , 1);

RETURN NULL;

END;

$$ LANGUAGE plpgsql;

**5)** CREATE OR REPLACE FUNCTION volunteer\_remove()

RETURNS TRIGGER AS $$

BEGIN

DELETE FROM volunteers where volunteer\_id=NEW.alumni\_id;

RETURN NULL;

END;

$$ LANGUAGE plpgsql;

**6)** CREATE OR REPLACE FUNCTION remove\_roleid()

RETURNS TRIGGER AS $$

begin

delete from volunteer\_roles where volunteer\_id=old.volunteer\_id;

delete from participations where volunteer\_id=old.volunteer\_id;

RETURN OLD;

END;

$$ LANGUAGE plpgsql;

**7)** CREATE OR REPLACE FUNCTION manualid\_insert()

RETURNS TRIGGER AS $$

declare

t\_mid int;

BEGIN

select max(manual\_id)+1 into t\_mid from manuals ;

NEW.manual\_id := t\_mid;

RETURN NEW;

END;

$$

LANGUAGE plpgsql;

**8)** CREATE OR REPLACE FUNCTION update\_manuals\_conducted()

RETURNS TRIGGER AS $$

BEGIN

UPDATE unit

SET manuals\_conducted = manuals\_conducted + 1

WHERE unit\_no = NEW.unit\_no;

RETURN NEW;

END;

$$

LANGUAGE plpgsql;

**9)** CREATE OR REPLACE FUNCTION manual\_delete\_after()

RETURNS TRIGGER AS $$

BEGIN

delete from participations where manual\_id= OLD.manual\_id;

update unit set manuals\_conducted= manuals\_conducted-1

where unit\_no = ( select unit\_no from manuals where manual\_id = OLD.manual\_id);

RETURN OLD;

END;

$$

LANGUAGE plpgsql;

**10)** CREATE OR REPLACE FUNCTION check\_current\_year\_before\_insert()

RETURNS TRIGGER AS $$

BEGIN

IF TG\_OP = 'INSERT' AND NEW.current\_year >= 5 THEN

RAISE EXCEPTION 'current\_year must be less than 4 during insert';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

**Triggers:**

**1)** CREATE or replace TRIGGER move\_to\_alumni

AFTER UPDATE ON volunteers

FOR EACH ROW

EXECUTE FUNCTION move\_to\_alumni();

**2)** CREATE or replace TRIGGER update\_no\_of\_members\_trigger

AFTER INSERT OR DELETE OR UPDATE ON volunteers

FOR EACH ROW

EXECUTE FUNCTION update\_no\_of\_volunteers();

**3)** CREATE OR REPLACE TRIGGER assign\_unit\_no

BEFORE INSERT ON volunteers

FOR EACH ROW

EXECUTE FUNCTION assign\_unit\_no();

**4)** CREATE or replace TRIGGER role\_id\_insert

AFTER insert ON volunteers

FOR EACH ROW

EXECUTE FUNCTION role\_id();

**5)** create or replace trigger volunteer\_remove

after insert ON alumni

FOR EACH ROW

EXECUTE FUNCTION volunteer\_remove();

**6)** CREATE or replace TRIGGER remove\_roleid

before delete ON volunteers

FOR EACH ROW

EXECUTE FUNCTION remove\_roleid();

**7)** CREATE OR replace TRIGGER manualid\_insert

BEFORE INSERT ON manuals

FOR EACH ROW

EXECUTE FUNCTION manualid\_insert();

**8)** CREATE OR replace TRIGGER update\_manuals\_conducted\_trigger

AFTER INSERT ON manuals

FOR EACH ROW

EXECUTE FUNCTION update\_manuals\_conducted();

**9)** CREATE OR replace TRIGGER manual\_delete\_after

after delete ON manuals

FOR EACH ROW

EXECUTE FUNCTION manual\_delete\_after();

**10)** CREATE or replace TRIGGER check\_current\_year\_trigger

BEFORE INSERT ON volunteers

FOR EACH ROW

EXECUTE FUNCTION check\_current\_year\_before\_insert();

**Cursor:**

**1)**

CREATE OR REPLACE FUNCTION fetch\_manuals\_for\_unit(unit\_no\_param INT)

RETURNS SETOF manuals AS $$

DECLARE

manual\_record manuals%ROWTYPE;

manual\_cursor CURSOR FOR

SELECT \* FROM manuals WHERE unit\_no = unit\_no\_param ORDER BY manual\_id ASC;

BEGIN

FOR manual\_record IN manual\_cursor LOOP

RETURN NEXT manual\_record;

END LOOP;

RETURN;

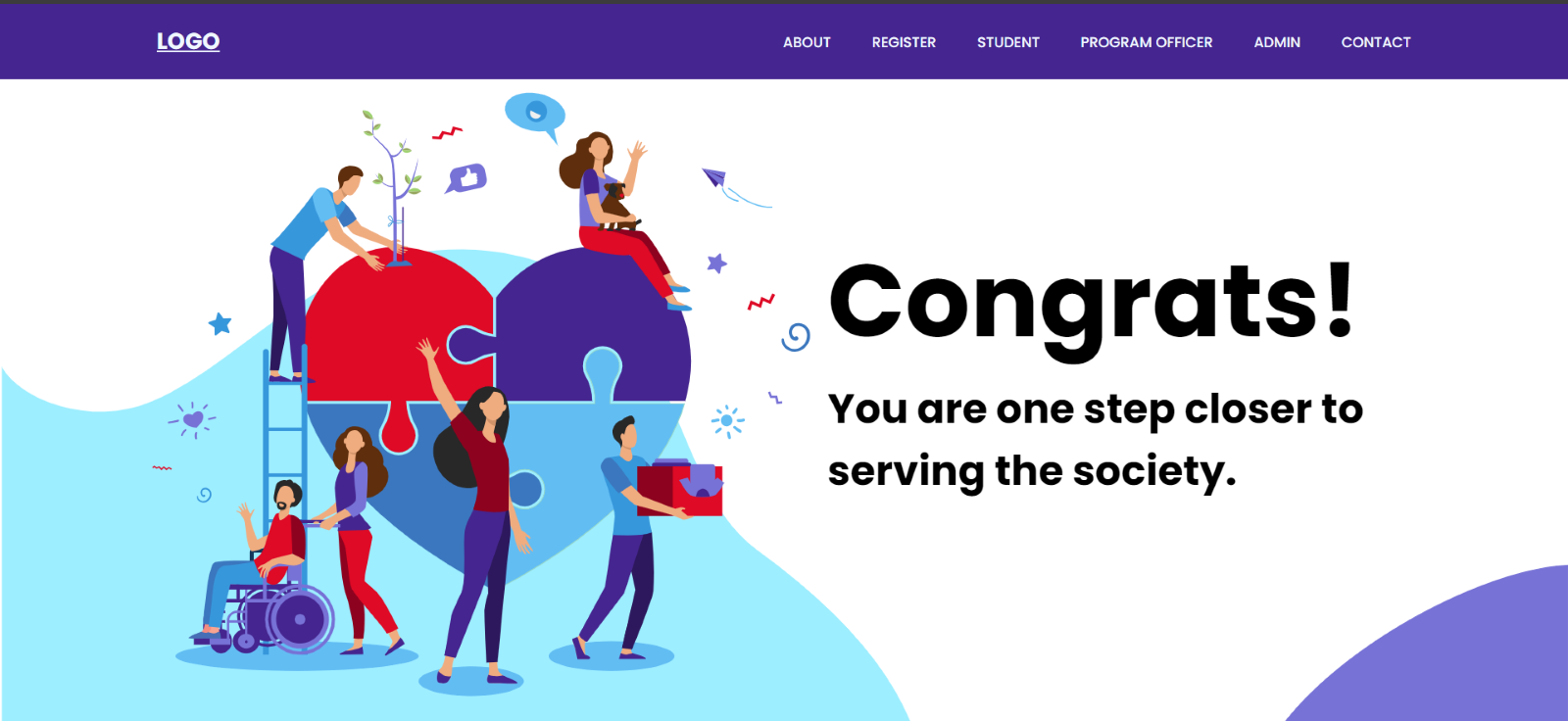
END;

$$ LANGUAGE plpgsql;

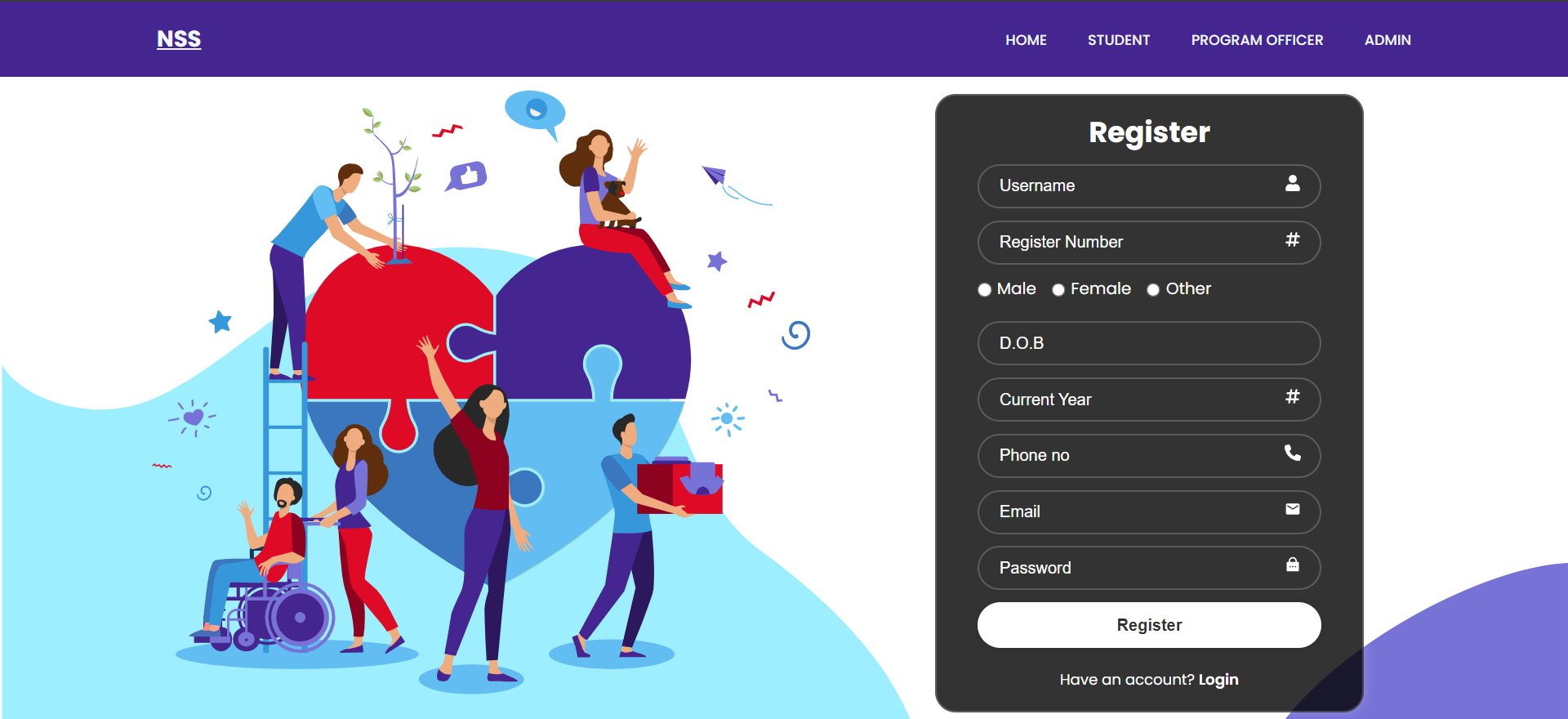
SELECT \* FROM fetch\_manuals\_for\_unit(1);

**IMPLEMENTATION:**

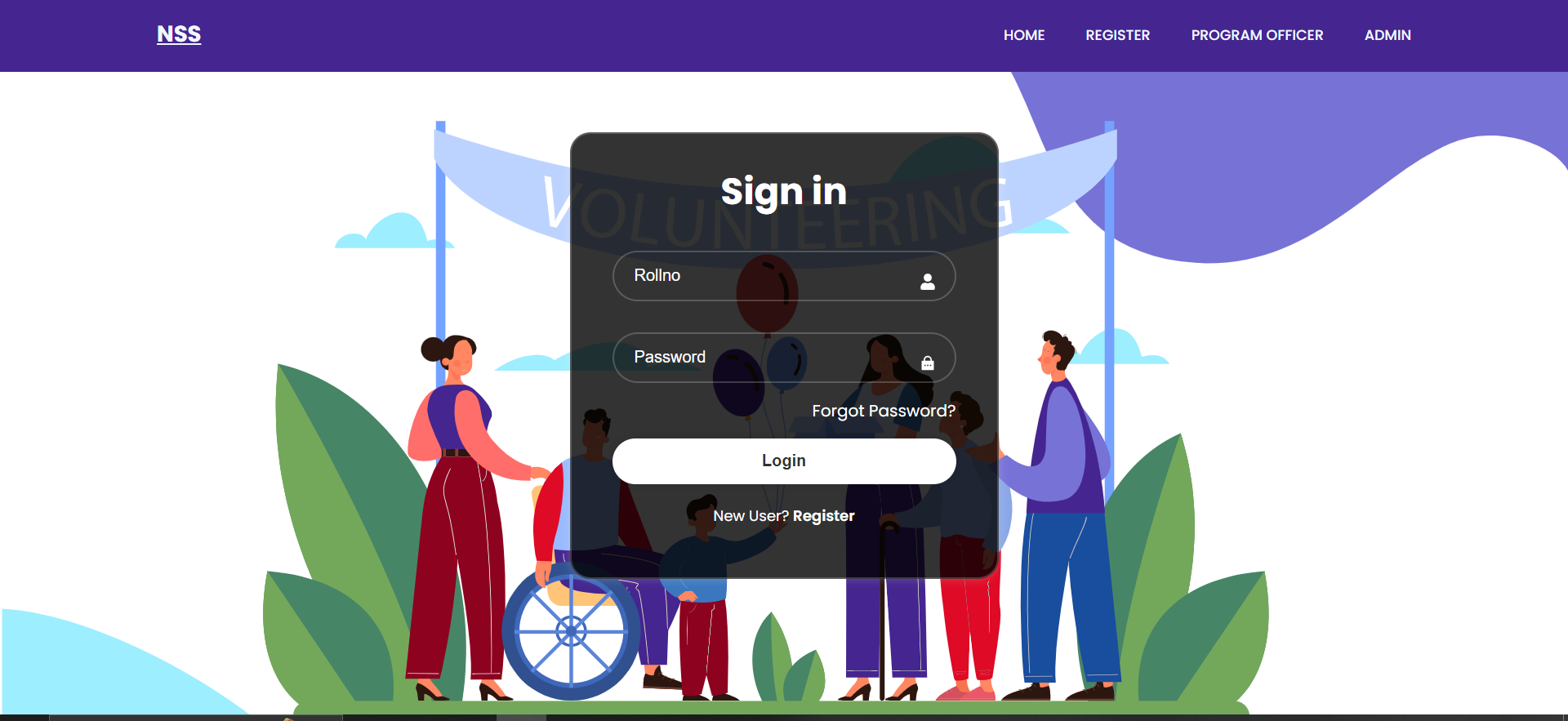
**Home Page:**



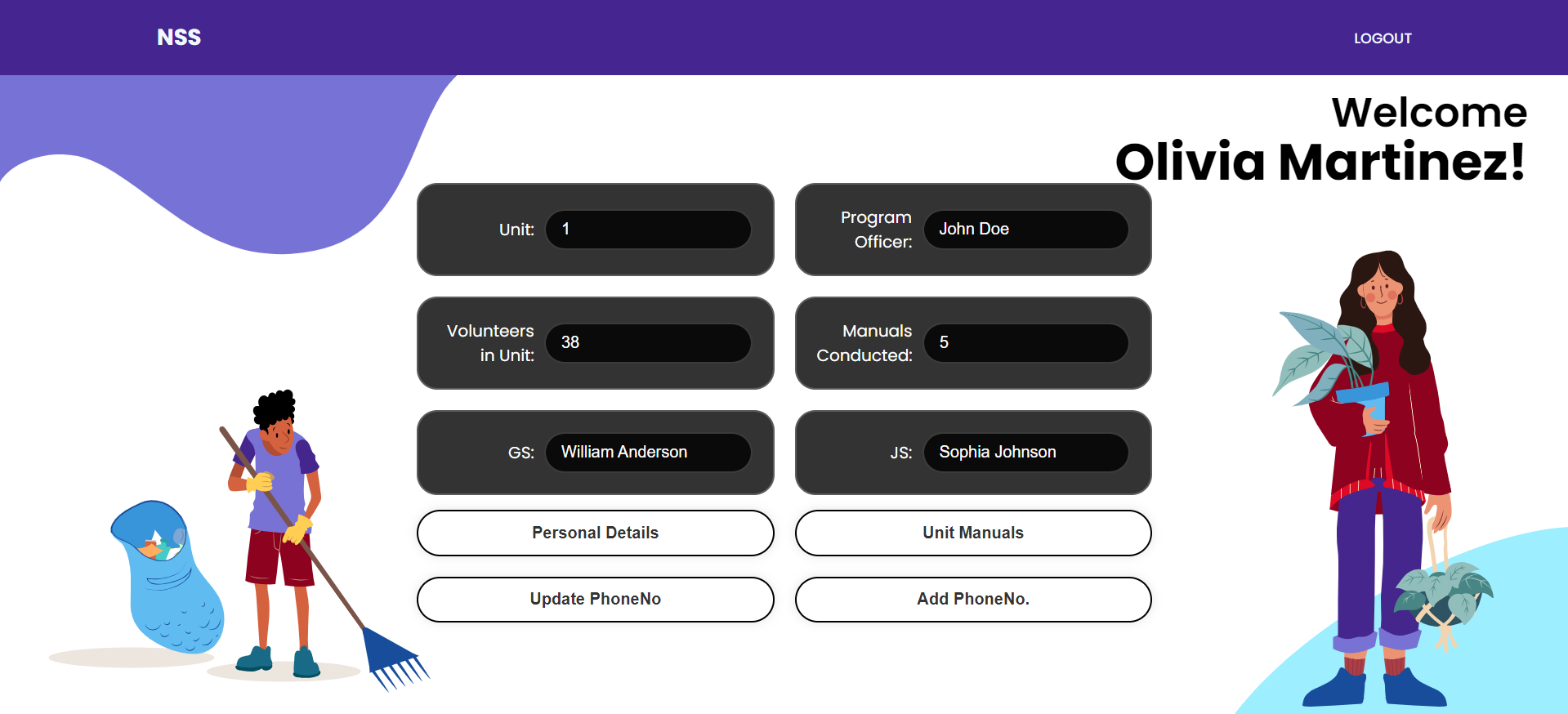
**Register Page:**



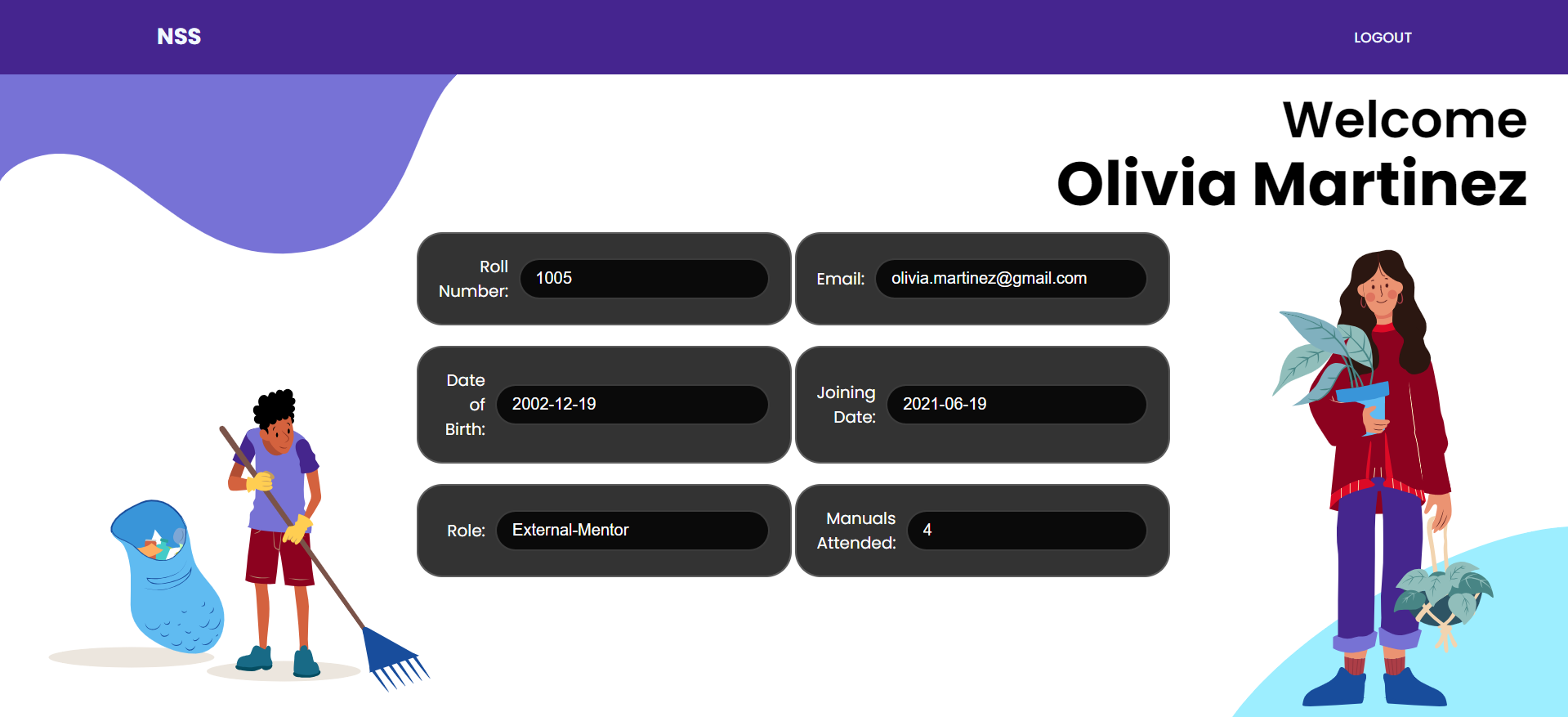
**Login Page:**



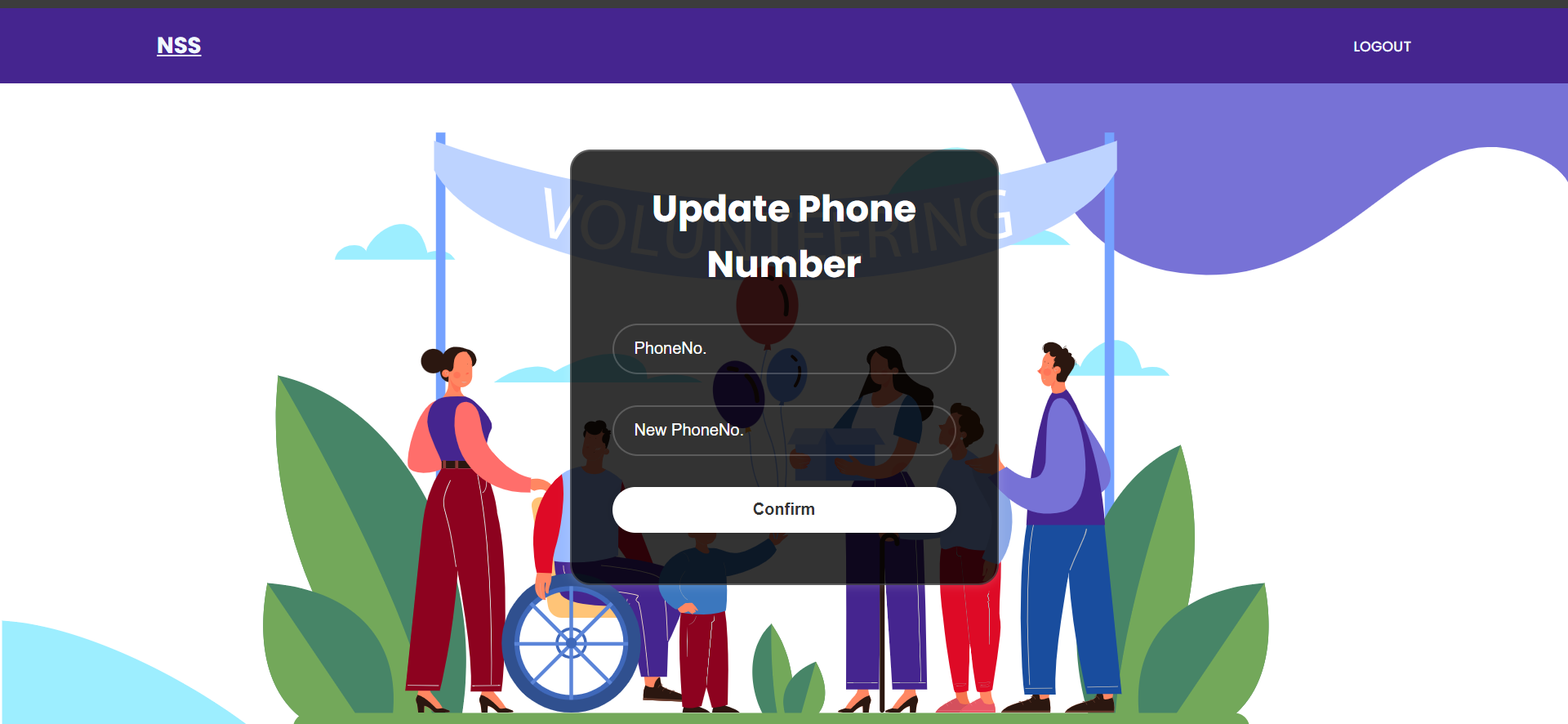
**Login Response Page:**



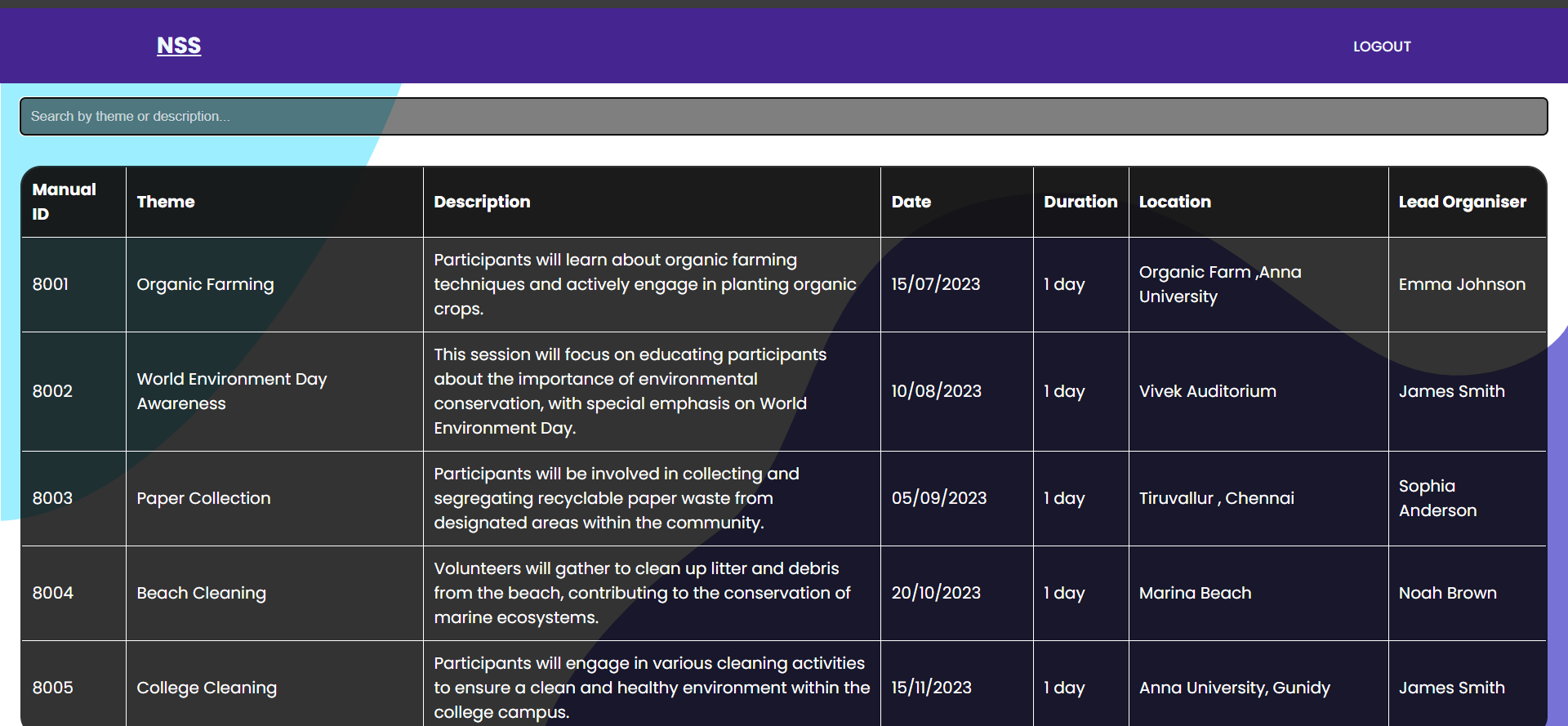
**Personal Details :**

****

**Update/Add phone number :**



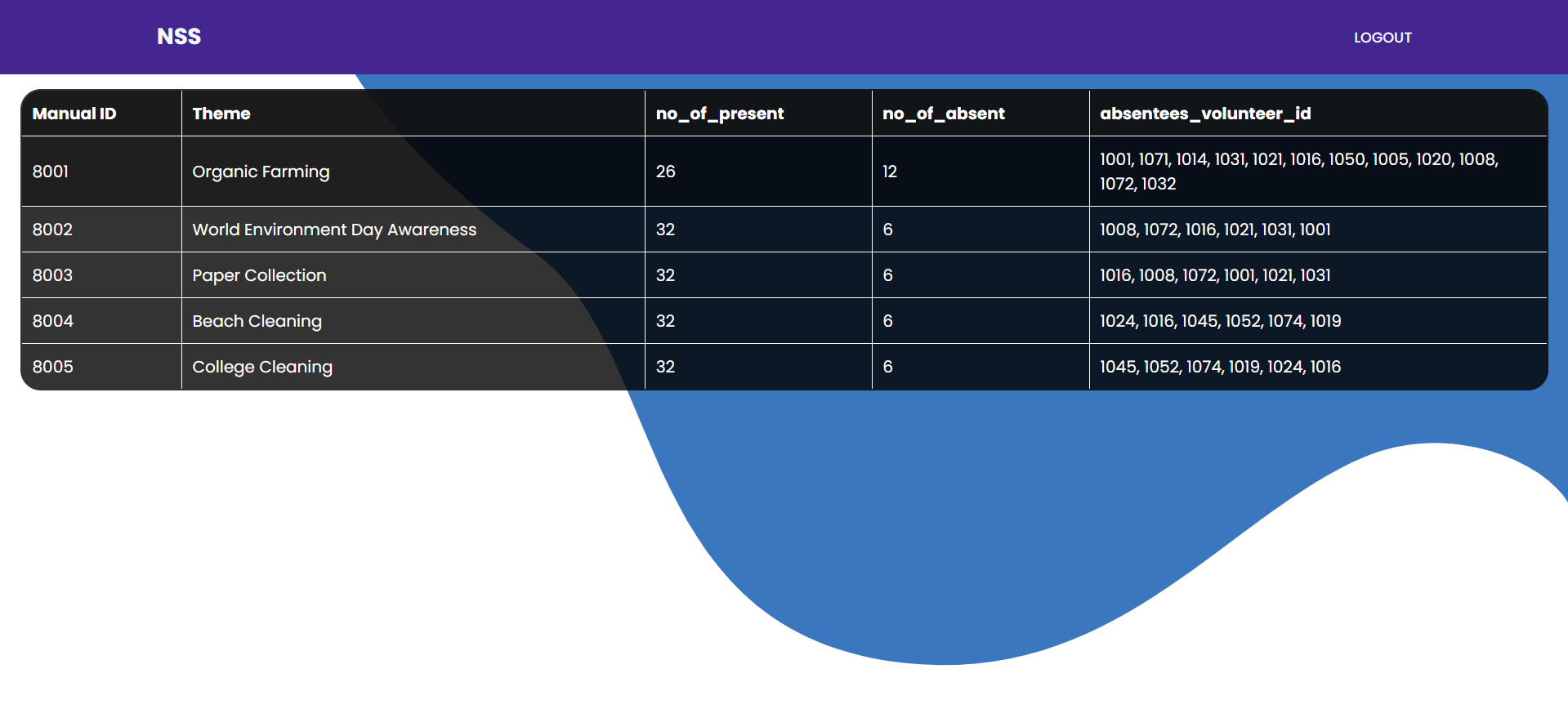
**Manual Details:**



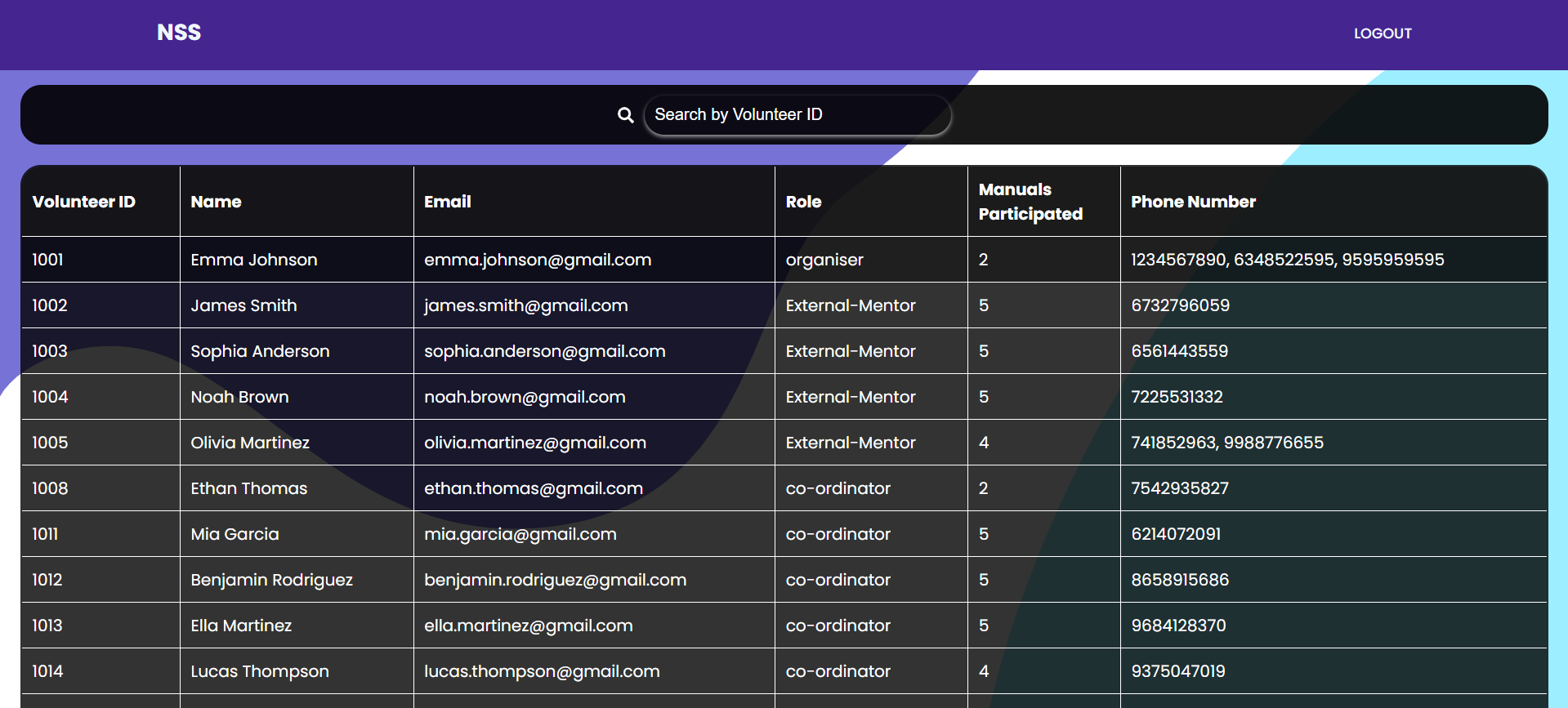
**PO Login Response :**

****

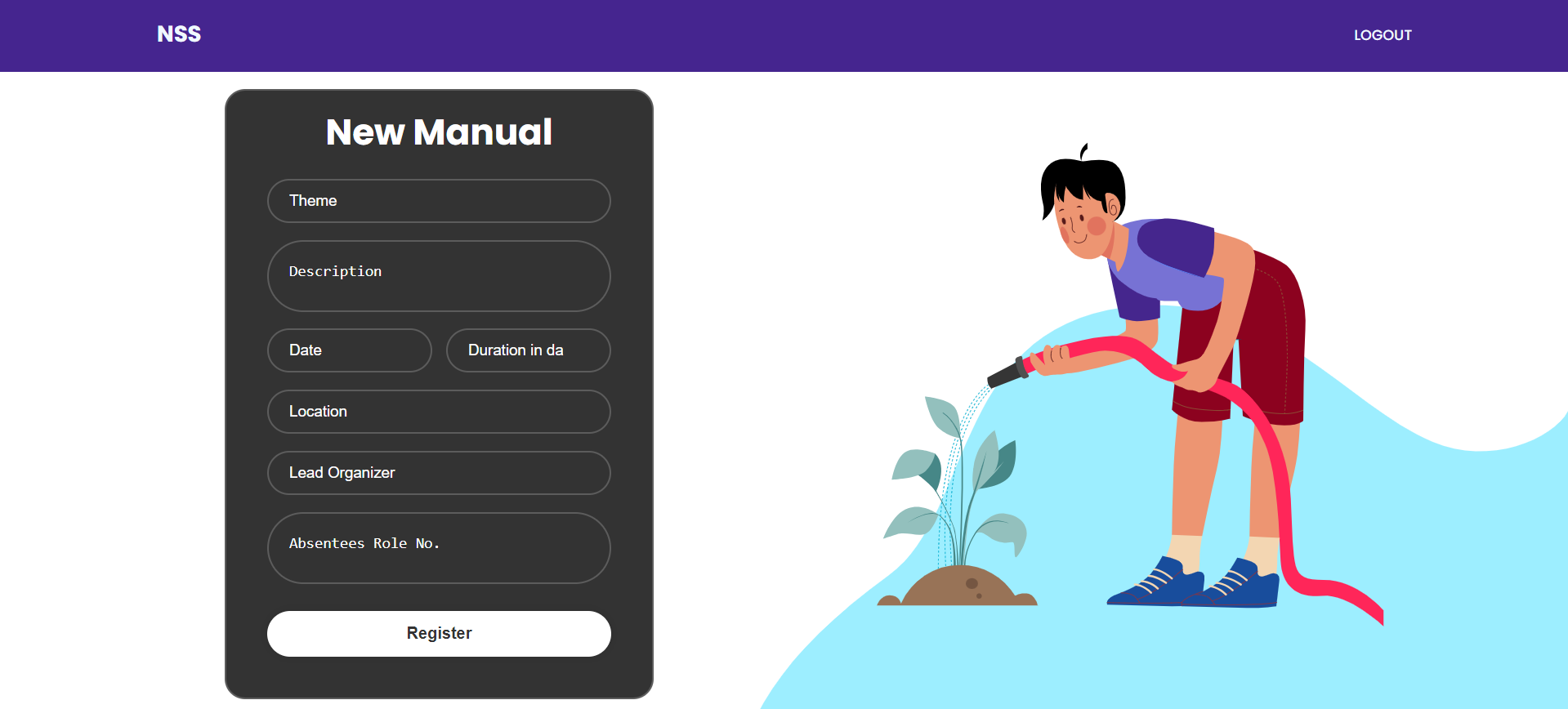
**Attendance Details:**



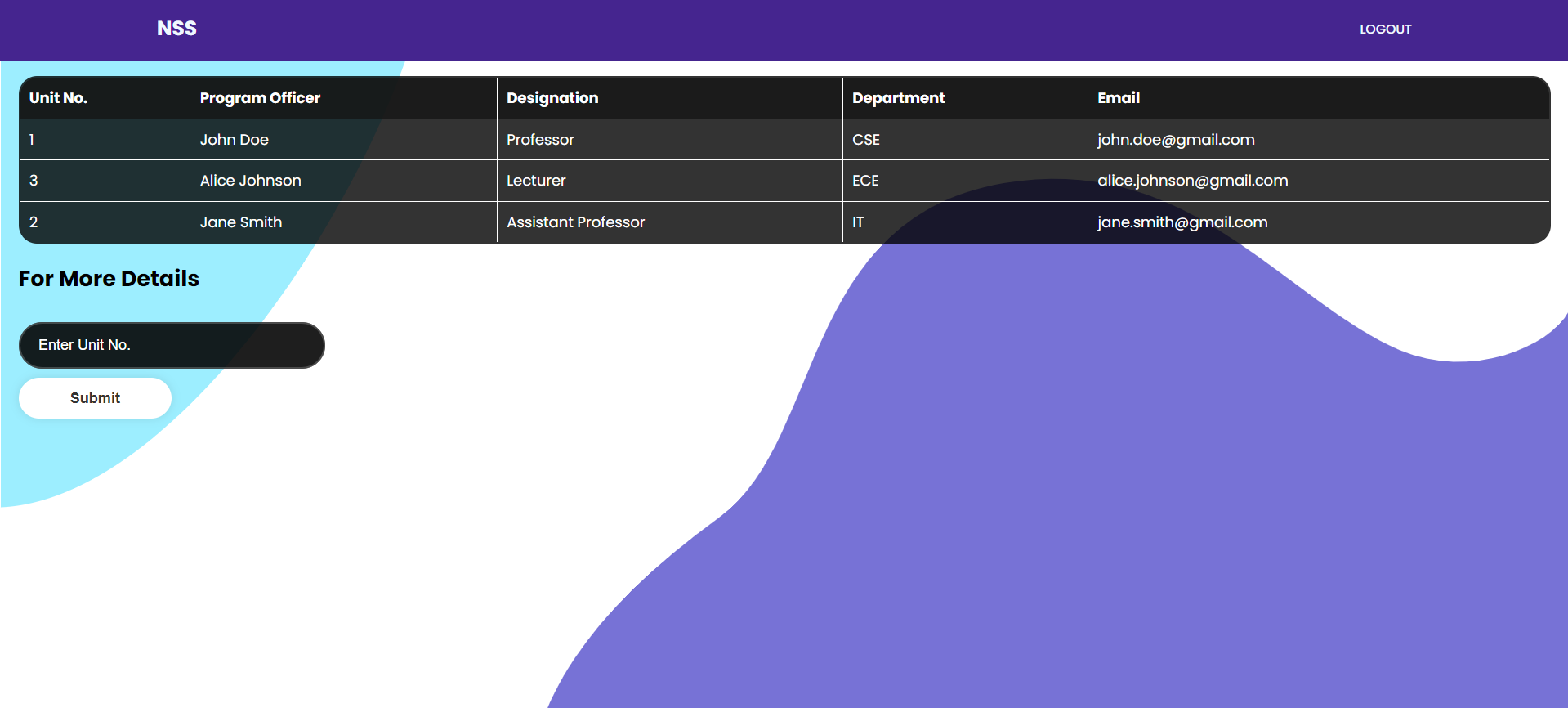
**Volunteers Details:**

****

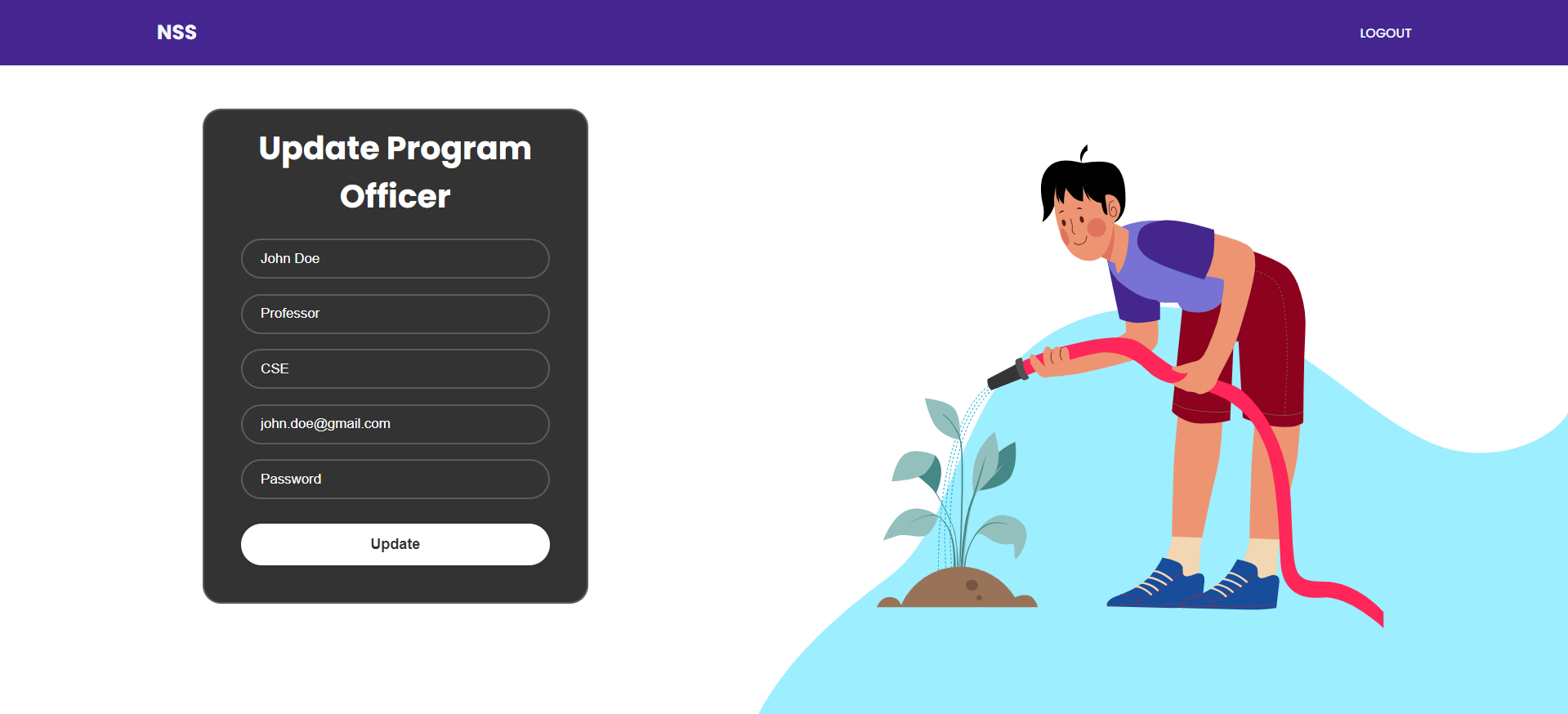
**Insert Manuals:**



**Admin Login Response:**

****

**Update PO:**

****

**CONCLUSION:**

In conclusion, the NSS organization database project aims to enhance the overall effectiveness of social service initiatives by NSS units within a college. Future enhancements may include additional features such as Personnel Announcements and Q&A /Feedback Options. It can also be performed on a broad scale by integrating with other colleges which can improve the overall output of NSS initiatives.