VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Santhibastawad Road, Machhe
Belagavi - 590018, Karnataka, India

DBMS LABORATORY WITH MINI PROJECT (21CSL55) REPORT
ON

"NSS MANAGEMENT SYSTEM"

Submitted in the partial fulfillment of the requirements for the award of the degree of

BACHELOR OF ENGINEERING
IN
INFORMATION SCIENCE AND ENGINEERING

For the Academic Year 2023-2024

Submitted by

M M Chidvilas Chowdary	**1JS21IS055**
M Sumit	**1JS21IS127**
Rajesh K	**1JS21IS128**
Vasanth Sai T	**1JS21IS129**

Under the Guidance of

Dr. Sowmya K N, Associate Professor, Dept. of ISE, JSSATEB
Mrs. Gayathri K, Assistant Professor, Dept. of ISE, JSSATEB

2023-2024

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

JSS ACADEMY OF TECHNICAL EDUCATION

JSS Campus, Dr.Vishnuvardhan Road, Bengaluru-560060

JSS MAHAVIDYAPEETHA, MYSURU

JSS ACADEMY OF TECHNICAL EDUCATION

JSS Campus, Dr.Vishnuvardhan Road, Bengaluru-560060

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that DBMS LABORATORY WITH MINI PROJECT (21CSL55) Report entitled "**NSS Management system**" is a Bonafede work carried out by **M M Chidvilas Chowdary [1JS21IS055], M Sumit [1JS21IS127], Rajesh K [1JS21IS128], Vasanth Sai T [1JS21IS129]** in partial fulfillment for the award of degree of Bachelor of Engineering in Information Science and Engineering of Visvesvaraya Technological University Belagavi during the year 2023- 2024.

Signature of the Guide

Dr. Sowmya KN
Associate Professor,
Dept. of ISE
JSSATEB

Mrs. Gayathri K
Assistant Professor,
Dept. of ISE
JSSATEB

Signature of the HOD

Dr. Rekha PM,
Professor & HOD
Dept. of ISE,
JSSATEB

TABLE OF CONTENTS

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible. So, with gratitude, we acknowledge all those whose guidance and encouragement crowned our efforts with success.

First and foremost, we would like to thank his **Holiness Jagadguru Sri Shivarathri Deshikendra Mahaswamiji** and **Dr Bhimasen Soragaon**, Principal, JSSATE, Bangalore for providing an opportunity to carry out the Project Work as a part of our curriculum in the partial fulfilment of the degree course.

We express our sincere gratitude for our beloved Head of the department, **Dr.Rekha P. M**, for her co-operation and encouragement at all the moments of our approach.

It is our pleasant duty to place on record our deepest sense of gratitude to our respected guide **Dr. Sowmya K N**, Associate Professor and **Mrs. Gayathri Kamath**, Assistant Professor for the constant encouragement, valuable help and assistance in every possible way.

We would like to thank all ISE Department Teachers, non-teaching staff and Library staff for providing us with their valuable guidance and for being there at all stages of our work.

M M Chidvilas Chowdary [1JS21IS055]
M Sumit [1JS21IS127]
Rajesh K [1JS21IS128]
Vasanth Sai T [1JS21IS129]

ABSTRACT

Developing and maintaining a National Service Scheme (NSS) benefits students, faculty mentors, and the university. Integrating NSS activities alongside a robust academic curriculum fosters independent critical thinking skills and enhances oral and written communication abilities. The NSS experience instills valuable learning objectives that endure as students transition into professional service. Faculty members at educational institutions can enrich student learning through NSS involvement while advancing their own research goals. The university gains from increased visibility through NSS events and publications.

Faculty members play a crucial role in NSS by guiding students, sharing knowledge, and employing systematic approaches to community service. Engaged faculty members earn recognition from peers, remain at the forefront of their fields, and contribute to interdisciplinary collaborations and grant opportunities.

A significant challenge for colleges is organizing and integrating NSS data efficiently. With NSS activities generating large volumes of digital data, there is a pressing need for a centralized platform. Our project addresses this challenge by collecting requirements from various stakeholders and departments, aiming to create a unified NSS Management System.

The primary objective of our project is to ensure the availability and accessibility of NSS data across departments. This includes facilitating long-term data preservation and providing faculty members with a user-friendly interface to manage and update NSS-related information. Through this initiative, we aim to streamline NSS administration and empower faculty members to conduct impactful community service activities with ease.

CHAPTER 1
INTRODUCTION

INTRODUCTION TO NSS MANAGEMENT SYSTEM

The National Service Scheme (NSS) has been a cornerstone of youth engagement and community development in Indian colleges since its inception in 1969. With its rich history and extensive reach, NSS has played a vital role in shaping the social consciousness and leadership skills of students across the nation. However, the traditional methods of managing NSS activities have become increasingly outdated and inefficient in the face of evolving challenges and technological advancements. In response to these challenges, our project endeavors to develop a robust and user-friendly NSS management system tailored to the unique needs of colleges.

Problem Statement:

In National Service Scheme (NSS) initiative, the current volunteer management system relies heavily on conventional paper-based processes for enrolment and document verification, leading to inefficiencies, delays, and administrative challenges. The absence of an efficient search mechanism further exacerbates these issues, making it difficult to retrieve and manage volunteer information promptly and accurately. To address these challenges, there is a pressing need to transition to a modern, user-friendly platform that enhances participant engagement and administrative efficiency. By implementing a digital solution tailored to the needs of volunteers, guides, team leaders, and administrators, we aim to streamline enrolment processes, improve search functionalities, boost volunteer participation, and enhance attendance management within the NSS ecosystem.

Setting the Objectives:

At the heart of our project lies a commitment to address the key points and challenges faced by colleges in managing NSS activities. Our objectives include the development of a comprehensive NSS management system that integrates seamlessly with existing college infrastructure, improves data management capabilities, and enhances user experience. By focusing on usability, scalability, and adaptability, we aim to create a solution that meets the diverse needs of colleges while empowering them to maximize the impact of their NSS initiatives.

Key Features and Functionality:

The NSS management system will offer a range of features designed to streamline various aspects of NSS administration, including volunteer enrollment, activity planning, attendance tracking, and performance evaluation. Through intuitive interfaces and automated workflows, users will be able to easily manage and monitor NSS activities in real-time, facilitating better decision-making and resource allocation. Additionally, the system will support customizable reporting and analytics, providing colleges with valuable insights into volunteer engagement and program effectiveness.

Scope and Implementation Strategy:

Our project's scope extends beyond the development of a standalone software solution to encompass a holistic approach to NSS management. We will work closely with college administrators, NSS coordinators, and student volunteers to understand their needs and requirements, ensuring that the system is tailored to their specific context and challenges. The implementation strategy will prioritize user training, change management, and continuous improvement to ensure a seamless transition and long-term success.

Historical Significance:

The National Service Scheme (NSS) has been instrumental in fostering a spirit of social responsibility and civic engagement among college students since its inception. Over the years, NSS activities have contributed significantly to community development, environmental conservation, and disaster relief efforts across India. By modernizing the management of NSS activities, colleges can build upon this rich legacy and further amplify their impact on society.

Student Empowerment:

The NSS management system presents an opportunity to empower students by providing them with access to a digital platform that facilitates active participation and leadership development. By engaging students in the design and implementation of the system, colleges can nurture a sense of ownership and accountability, fostering a culture of proactive citizenship and social entrepreneurship.

OBJECTIVES

The Objectives of **NSS Management System** are:

1. The primary goal of our project is to create a user-friendly platform for managing all activities related to the National Service Scheme within a college or university.
2. This project aims to facilitate better coordination and communication among NSS volunteers, coordinators, and administrators by providing a centralized system for information sharing.
3. It seeks to streamline the process of organizing and documenting NSS events, volunteer registrations, and participation records.
4. The NSS Management System will enable efficient tracking of volunteer activities, including participation in community service projects, workshops, and awareness campaigns.
5. It aims to enhance engagement and involvement of volunteers by providing them with an intuitive platform to sign up for events, track their progress, and contribute to the NSS community.

ORGANIZATION OF THE REPORT

Chapter 1 provides the information about the basics of file structures, the history of DBMS and the objectives. In **Chapter 2** we discuss about the literature survey and the normalization. In Page | 11 NSS Management System **Chapter 3**, we discuss the software and hardware requirements to run the above applications. **Chapter 4** gives the idea of the system design which includes unit testing, integration testing and system testing. **Chapter 5** gives a clear picture about the project and its actual implementation. In **Chapter 6** we discuss about the unit testing, system testing and integration testing. **Chapter 7** discusses about the results and discussions of the program. **Chapter 8** concludes by giving the direction for future enhancement. **Chapter 9** includes the references.

CHAPTER 2

LITERATURE SURVEY

INTRODUCTION

A database management system (DBMS) refers to the technology for creating and managing databases. Basically, DBMS is a software tool to organize (create, retrieve, update and manage) data in a database. The main aim of a DBMS is to supply a way to store up and retrieve database information that is both convenient and efficient. By data, we mean known facts that can be recorded and that have embedded meaning. Normally people use software such as DBASE IV or V, Microsoft ACCESS, or EXCEL to store data in the form of database. A datum is a unit of data. Meaningful data combined to form information. Hence, information is interpreted data – data provided with semantics. MS. ACCESS is one of the most common examples of database management software. The name indicates what database is. Database is one of the important components for many applications and is used for storing a series of data in a single set. In other words, it is a group / package of information that is put in order so that it can be easily access, manage and update.

MySQL

MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. It is developed, marketed and supported by MySQL AB, which is a Swedish company.

MySQL is becoming so popular because of many good reasons −It is released under an open-source license. So, you have nothing to pay to use it, it is a very powerful program in its own right and handles a large subset of the functionality of the most expensive and powerful database packages. MySQL uses a standard form of the well-known SQL data language. It works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc and works very quickly and works well even with large data sets.

Fig 2.1 MySQL

Node.JS

Node.js is a JavaScript runtime environment that enables server-side programming, utilizing an event-driven, non-blocking I/O model for efficient handling of concurrent connections. With Node.js, developers can write both client-side and server-side code using JavaScript, leveraging its asynchronous programming paradigm to handle multiple tasks simultaneously. Node.js benefits from a rich ecosystem of open-source packages available through npm, facilitating the extension of application capabilities. Drawing inspiration from JavaScript's event-driven nature and the non-blocking I/O model of languages like Python, Node.js offers scalability, performance, and responsiveness for building web servers, real-time web applications, APIs, and microservices.

Fig 2.2 NodeJS

CSS

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML.CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate .CSS file which reduces complexity and repetition

in the structural content as well as enabling the .css file to be cached to improve the page load speed between the pages that share the file and its formatting. The name cascading comes from the specified priority scheme to determine which style rule applies if more than one rule matches a particular element. This cascading priority scheme is predictable.

Fig 2.3 CSS

HTML

Hypertext Markup Language (HTML) is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.

Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages. HTML describes the structure of a web page semantically and originally included cues for the appearance of the document.

HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page. HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by tags, written using angle brackets. Tags such as <img /> and <input /> directly introduce content into the page. Other tags such as <p> surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags, but use them to interpret the content of the page.

HTML can embed programs written in a scripting language such as JavaScript, which affects the behavior and content of web pages. Inclusion of CSS defines the look and layout of content. The World Wide Web Consortium (W3C), former maintainer of the HTML and current maintainer of

the CSS standards, has encouraged the use of CSS over explicit presentational HTML since 1997.

Fig 2.4 HTML

JAVASCRIPT

JavaScript abbreviated as JS, is a programming language that conforms to the ECMAScript specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions.

Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. JavaScript enables interactive web pages and is an essential part of web applications. The vast majority of websites use it for client-side page behavior, and all major web browsers have a dedicated JavaScript engine to execute it.

As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative programming styles. It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM).

JavaScript engines were originally used only in web browsers, but they are now embedded in some servers, usually via Node.js. They are also embedded in a variety of applications created with frameworks such as Electron and Cordova.

Fig 2.5 Java Script

Windows 10

Windows 10 is a major release of the Windows NT operating system developed by Microsoft. It is the successor to Windows 8.1, which was released nearly two years earlier, and itself was released to manufacturing on July 15, 2015, and broadly released for the general public on July 29, 2015. Windows 10 was made available for download via MSDN and TechNet, as a free upgrade for retail copies of Windows 8 and Windows 8.1 users via the Windows Store, and to Windows 7 users via Windows Update. Windows 10 receives new builds on an ongoing basis, which are available at no additional cost to users, in addition to additional test builds of Windows 10, which are available to Windows Insiders. Devices in enterprise environments can receive these updates at a slower pace, or use long-term support milestones that only receive critical updates, such as security patches, over their ten-year lifespan of extended support.

Mac OS X

Mac OS X was originally presented as the tenth major version of Apple's operating system for Macintosh computers; until 2020, versions of macOS retained the major version number "10". The letter "X" in Mac OS X's name refers to the number 10, a Roman numeral, and Apple has stated that it should be pronounced "ten" in this context. However, it is also commonly pronounced like the letter "X". Previous Macintosh operating systems (versions of the classic Mac OS) were named using Arabicnumerals, as with Mac OS 8 and Mac OS 9.

NORMALIZATION

Database normalization is a database schema design technique, by which an existing schema is modified to minimize redundancy and dependency of data. Normalization split a large table into smaller tables and define relationships between them to increases the clarity in organizing data.
Database normalization types

First Normal Form (1NF)

• First normal form (1NF) deals with the `shape' of the record type
• A relation is in 1NF if, and only if, it contains no repeating attributes or groups of attributes.

• Example: The student table with the repeating group is not in 1NF

Second Normal Form (2NF)

• A relation is in 2NF if, and only if, it is in 1NF and every non-key attribute is fully functionally dependent on the whole key.

Third Normal Form (3NF)

• A relation is in 3NF if, and only if, it is in 2NF and there are no transitive functional dependencies.

• Transitive functional dependencies arise:

• when one non-key attribute is functionally dependent on another non-key attribute

• FD: non-key attribute -> non-key attribute

• and when there is redundancy in the database

Boyce-Codd Normal Form (BCNF)

• When a relation has more than one candidate key, anomalies may result even though the relation is in 3NF.

• 3NF does not deal satisfactorily with the case of a relation with overlapping candidate keys

• i.e. composite candidate keys with at least one attribute in common.

• BCNF is based on the concept of a determinant.

Fourth Normal Form (4NF)

• It is a normal form used in database normalization Introduced by Ronald Fagin in 1977, 4NF is the next level of normalization after Boyce–Codd normal form (BCNF). Whereas the second, third, and Boyce–Codd normal forms are concerned with functional dependencies, 4NF is concerned with a more general type of dependency known as a multivalued dependency

Fifth Normal Form (5NF)

• It is also known as project-join normal form (PJ/NF) is a level of database normalization designed to reduce redundancy in relational databases recording multi-valued facts by isolating semantically related multiple relationships. A table is said to be in the 5NF if and only if every nontrivial join dependency in that table is implied by the candidate keys.

CHAPTER 3

REQUIREMENT SPECIFICATIONS

SOFTWARE SPECFICATION

- Project Type: Web-Based Application
- Front-end Tech: HTML, CSS, BOOTSTRAP, JavaScript
- Database Tool: MySQL
- Back-end Tech: Node.js (with Express.js Framework)
- OS: Windows, Linux and macOS compatible.
- Browser: Chrome, Firefox, Safari, Edge
- Software: Node.js, npm, MySQL Server, any text editor (eg. Visual Studio Code)

HARDWARE SPECFICATION

- Processor: x86 compatible processor with 1.7 GHz Clock Speed
- RAM: 2 GB or greater
- Hard Disk: 50 GB or greater
- Monitor: VGA/SVGA or higher resolution
- Keyboard: Standard 104 keys
- Mouse: 2/3 button Optical/Mechanical

FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

Functional Requirements:

1. User Registration: The system should allow users to register with unique usernames and passwords. Users must provide their email addresses and select their user type, which can be Volunteer, Team Leader, Guide, or Administrator. User registration information should be stored securely in the database (USERS table).

2. Volunteer Profile Management: Volunteers should be able to manage their profiles by updating personal information such as first name, last name, gender, date of birth, contact details, Aadhar number, address, and blood group. This information should be stored in the VOLUNTEERS table.
3. Activity Creation and Management: Administrators and team leaders should have the capability to create and manage NSS activities. This includes providing details such as activity name, description, duration, and organizer. The system should assign a unique activity ID to each activity and store these details in the ACTIVITY_DETAILS table.
4. Volunteer Enrollment: Volunteers should be able to enroll in various NSS activities. The system should allow volunteers to select activities from a list and enroll by providing their enrollment ID. The enrollment details should be recorded in the ENROLLMENT table, linking each volunteer to the activities they have enrolled in.
5. Attendance Tracking: Team leaders and administrators should be able to track volunteer attendance for each NSS activity. The system should record attendance details such as enrollment ID, event ID, date, check-in time, check-out time, location, and daily hours in the ATTENDANCE_TRACKING table.

Non-Functional Requirements:

1. Security: The system should ensure the confidentiality and integrity of user data, activity details, and attendance records. Passwords and sensitive information should be securely encrypted. Access to certain features and data should be restricted based on user roles.
2. Performance: The system should be responsive and capable of handling simultaneous user interactions, especially during peak times when many volunteers are enrolling in activities or checking attendance. Queries to retrieve user information, activity details, and attendance records should execute efficiently.
3. Scalability: The system should be designed to accommodate a growing number of users, activities, and attendance records over time. The database structure and application architecture should be scalable to support increased workload without compromising performance.
4. Reliability: The system should be reliable and available for use at all times. Measures such as regular backups, data redundancy, and error handling should be implemented to prevent data loss and minimize downtime.
5. Usability: The user interface should be intuitive and easy to navigate, allowing volunteers, team leaders, and administrators to perform tasks with minimal training. Clear instructions and error messages should be provided to guide users through the system. Additionally, the system should be accessible from different devices and browsers.

CHAPTER 4

SYSTEM DESIGN

INTRODUCTION TO SYSTEM DESIGN

System is a collection of an interrelated components that works together to achieve a purpose. System analysis is referred to the systematic examination or detailed study of a system in order to identify problems of the system, and using the information gathered in the analysis stage to recommend improvements or solution to the system.

System design is an abstract representation of a system component and their relationship and which describe the aggregated functionality and performance of the system. System design is also the overall plan or blueprint for how to obtain answer to the question being asked. The design specifies which of the various type of approach.

Systems design is the process or art of defining the architecture, components modules, interfaces, and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development.

DESIGN PLAN

User Interface: The user interface includes features such as sign-up and login pages, allowing new users to register and existing users to authenticate themselves. It also provides intuitive navigation for users to access various functionalities based on their roles.

Backend Server: The backend server, powered by Node.js, facilitates communication between the frontend and the database. It handles user requests, processes data, and ensures the integrity and security of the system. In our project, the backend server also manages user authentication and authorization, directing users to specific functionalities based on their roles.

SQL Database: The MySQL database stores essential data related to users, volunteers, activities, and attendance. It provides a structured storage mechanism for efficient data retrieval, storage, and management. The database ensures data consistency and integrity, supporting various operations such as user registration, volunteer enrollment, activity management, and attendance

tracking.

User Authentication/Authorization: This component manages user authentication and authorization using Role-Based Access Control (RBAC). It verifies user credentials during login and grants appropriate access rights based on predefined roles such as volunteer or administrator. RBAC ensures that users can only access functionalities relevant to their roles, enhancing system security and data privacy.

Volunteer Enrollment: This feature allows volunteers to register for various activities or programs offered by the organization. It includes functionalities for capturing volunteer information, such as personal details, contact information, and preferences.

Activity Details Management: This feature enables administrators to manage activity details, including descriptions, durations, and organizers. Administrators can add, update, or delete activities as needed, providing volunteers with comprehensive information about available opportunities.

Attendance Tracking: This feature allows volunteers to track their attendance for specific activities they have enrolled in. It includes functionalities for recording check-in/check-out times, locations, and daily hours spent on each activity. Additionally, administrators can view and approve volunteer attendance records, ensuring accurate tracking and reporting of volunteer participation.

ARCHITECTURE DIAGRAM

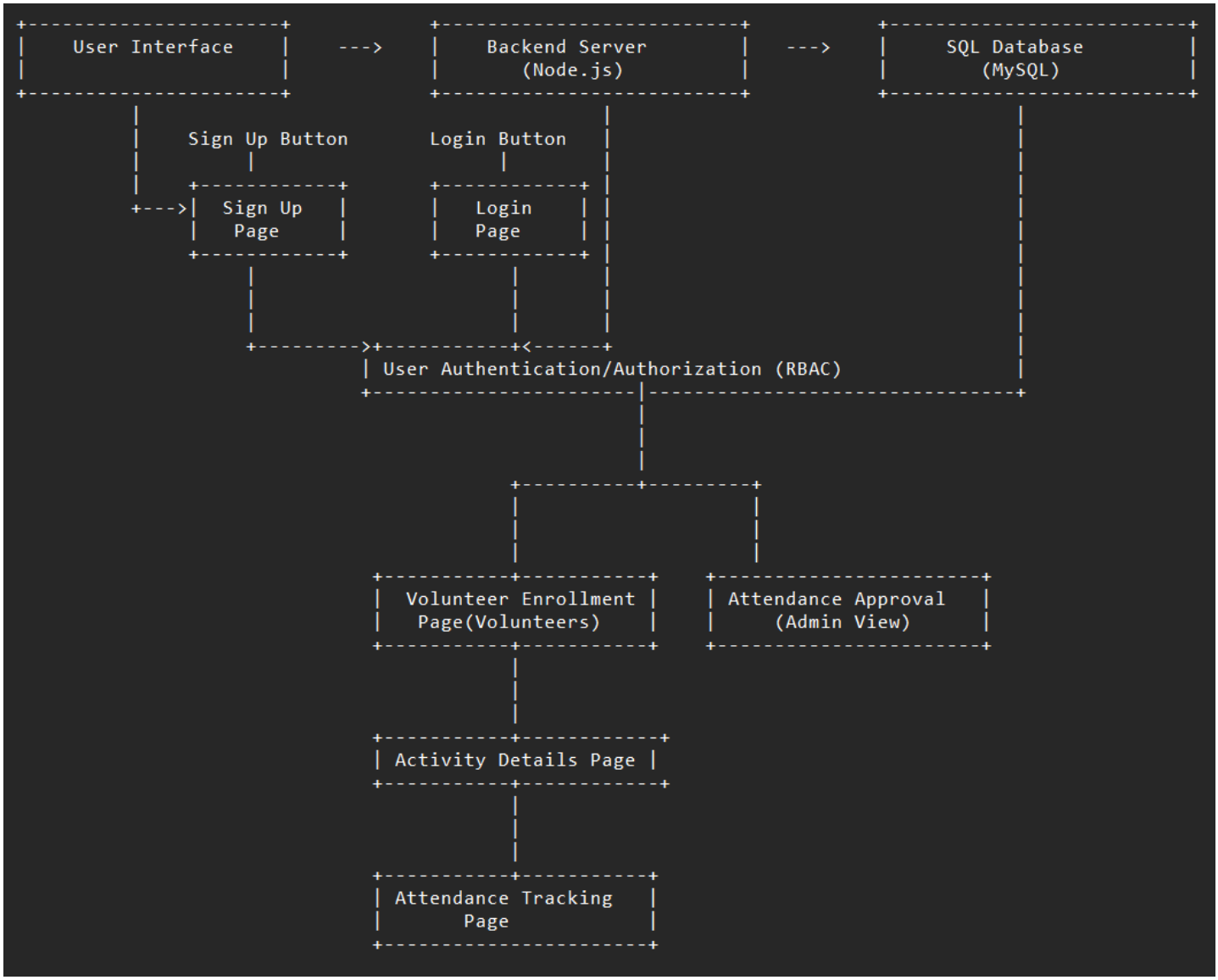


Fig 4.1 Architecture Diagram of NSS Management System

Database Design is a collection of processes that facilitate the designing, development, implementation and maintenance of enterprise data management systems. Properly designed database is easy to maintain, improves data consistency and are cost effective in terms of disk storage space. The database designer decides how the data elements correlate and what data must be stored.

The main objectives of database designing are to produce logical and physical designs models of the proposed database system.

ATTRIBUTES

Attributes define the properties of a data object and take on one of three different characteristics. They can be used to:

- Name an instance of data object.
- Describe the instance.

The following figure shows the Relational model for NSS Management System:

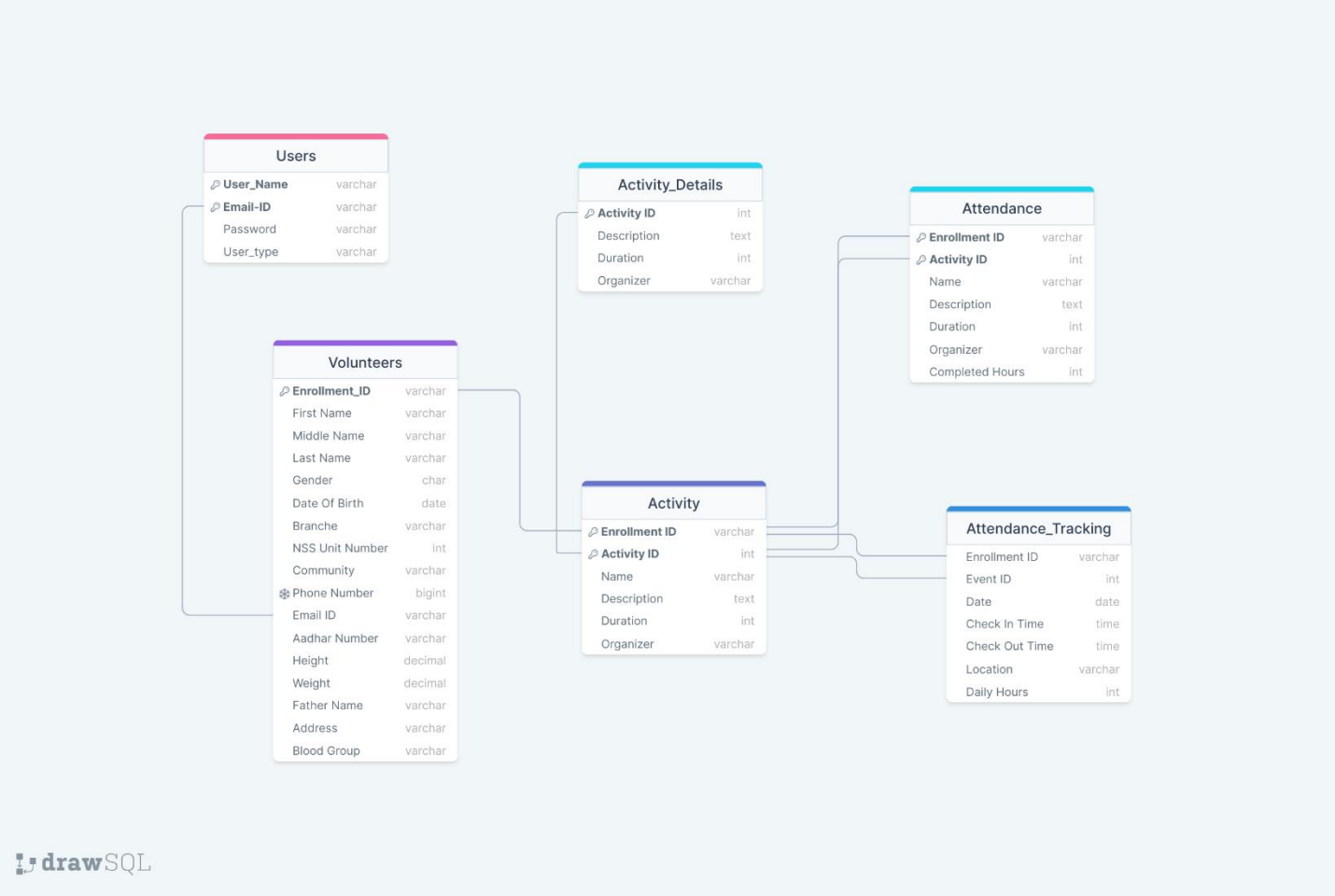


Fig 4.2 Relational View of NSS Management System

Relational Model was proposed by E.F. Codd to model data in the form of relations or tables. After designing the conceptual model of Database using ER diagram, we need to convert the conceptual model in the relational model which can be implemented using any RDBMS languages like Oracle SQL, MySQL etc. So, we will see what Relational Model is. Relational Model represents how data is stored in Relational Databases. A relational database stores data in the form of relations (tables).

SCHEMA DIAGRAM

A **database schema** is the skeleton structure that represents the logical view of the entire database. A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams.

A **schema diagram** contains entities and the attributes that will define that **schema**. It only shows us the database design. It does not show the actual data of the database. **Schema** can be a single table or it can have more than one table which is related.

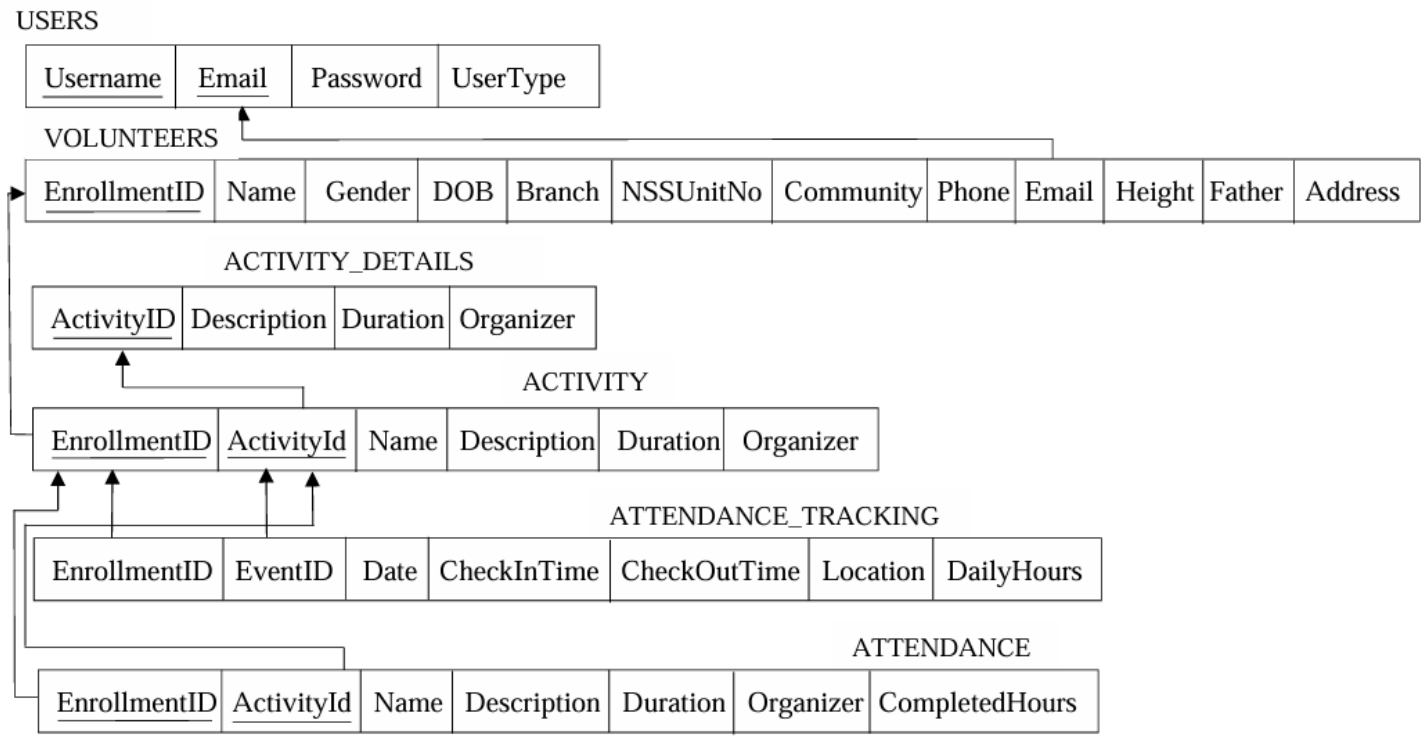


Fig 4.3 Schema Diagram of NSS Management System

The term "schema" refers to the organization of data as a blueprint of how the database is constructed. These integrity constraints ensure compatibility between parts of the schema. All constraints are expressible in the same language. A database can be considered a structure in realization of the database language.

The states of a created conceptual schema are transformed into an explicit mapping, the database schema. This describes how real-world entities are modeled in the database. All the various table used are described in the following schema. The necessary Primary key's and the corresponding foreign keys are also represented.

ER DIAGRAM

An **Entity–relationship model (ER model)** describes the structure of a database with the help of a diagram, which is known as **Entity Relationship Diagram (ER Diagram)**. An ER model is a design or blueprint of a database that can later be implemented as a database. The main components of E-R model are: entity set and relationship set.

The ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database.
Here are the geometric shapes and their meaning in an E-R Diagram.

Rectangle: Represents Entity sets.
Ellipses: Attributes
Diamonds: Relationship Set
Lines: They link attributes to Entity Sets and Entity sets to Relationship Set
Double Ellipses: Multivalued Attributes
Dashed Ellipses: Derived Attributes
Double Rectangles: Weak Entity Sets
Double Lines: Total participation of an entity in a relationship set

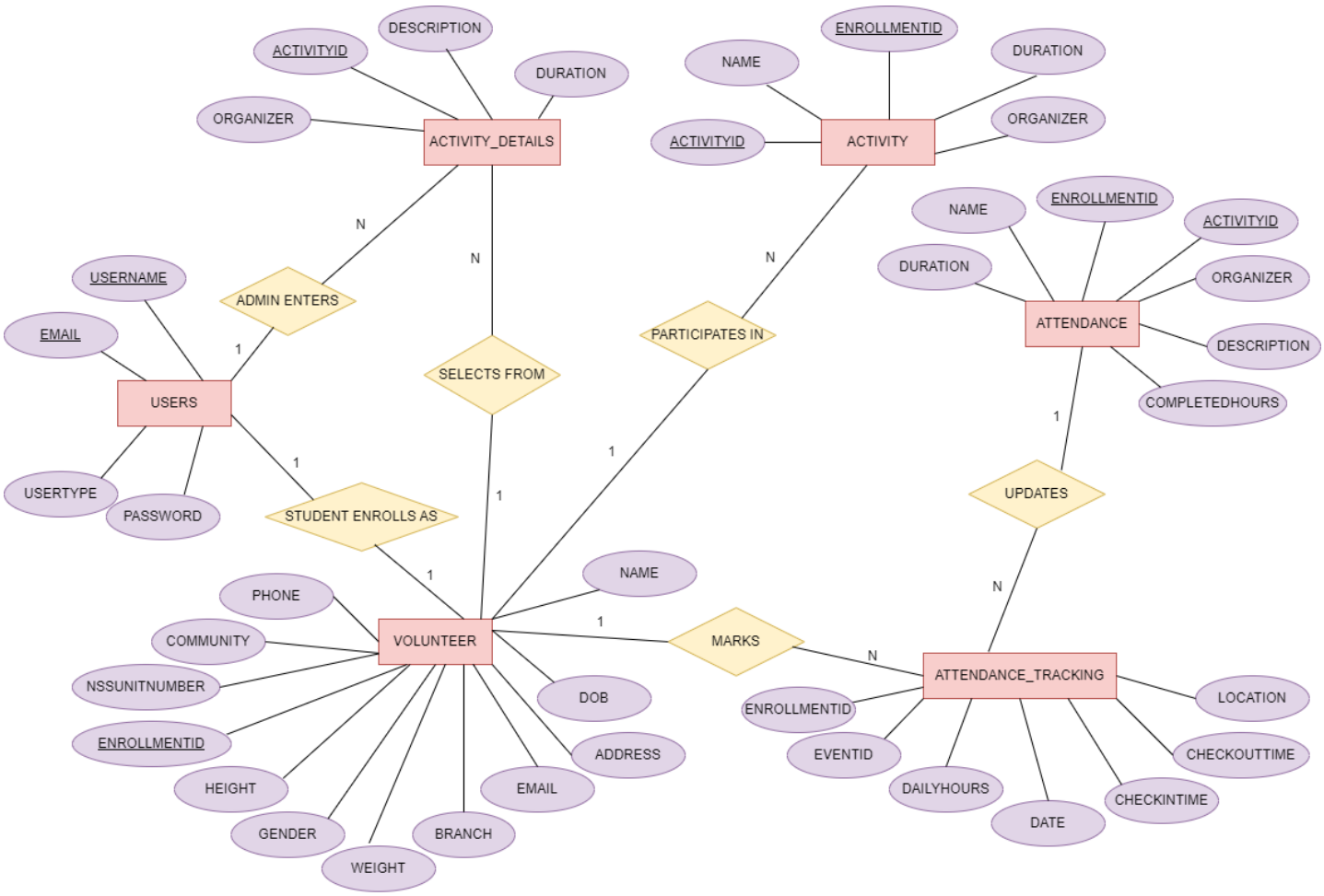


Fig 4.4 ER Diagram of NSS Management System

An entity–relationship model or the ER Diagram describes inter-related things of interest in a specific domain of knowledge. An ER model is composed of entity types and specifies relationships that can exist between instances of those entity types.

WIREFRAMES

A wireframe is a rough schematic created in the early stages of digital product design to help visualize and communicate the structure of a product or website.

Why are wireframes used?

The same screen can be built in a lot of different ways, but only a few of them will get your message across correctly and result in an easy-to-use software or website.

The purpose of a wireframe is to define a skeletal layout that is easy to understand, and encourages iteration and feedback. Getting to agreement on a good interface structure is a critical part of the software design process.

Wireframes are important because doing this work now, before any code is written and before the visual design is finalized, will save you lots of time and painful adjustment work later.

Description: Home Page

The wireframe for the home page of the NSS Management System would prominently feature the NSS logo and include navigation buttons for easy access to different sections of the system. It would welcome users with a brief message and provide an overview of upcoming events or activities. Additionally, the home page would offer options for new users to sign up and existing users to log in, ensuring a seamless entry into the system. It is shown in Fig 4.5.

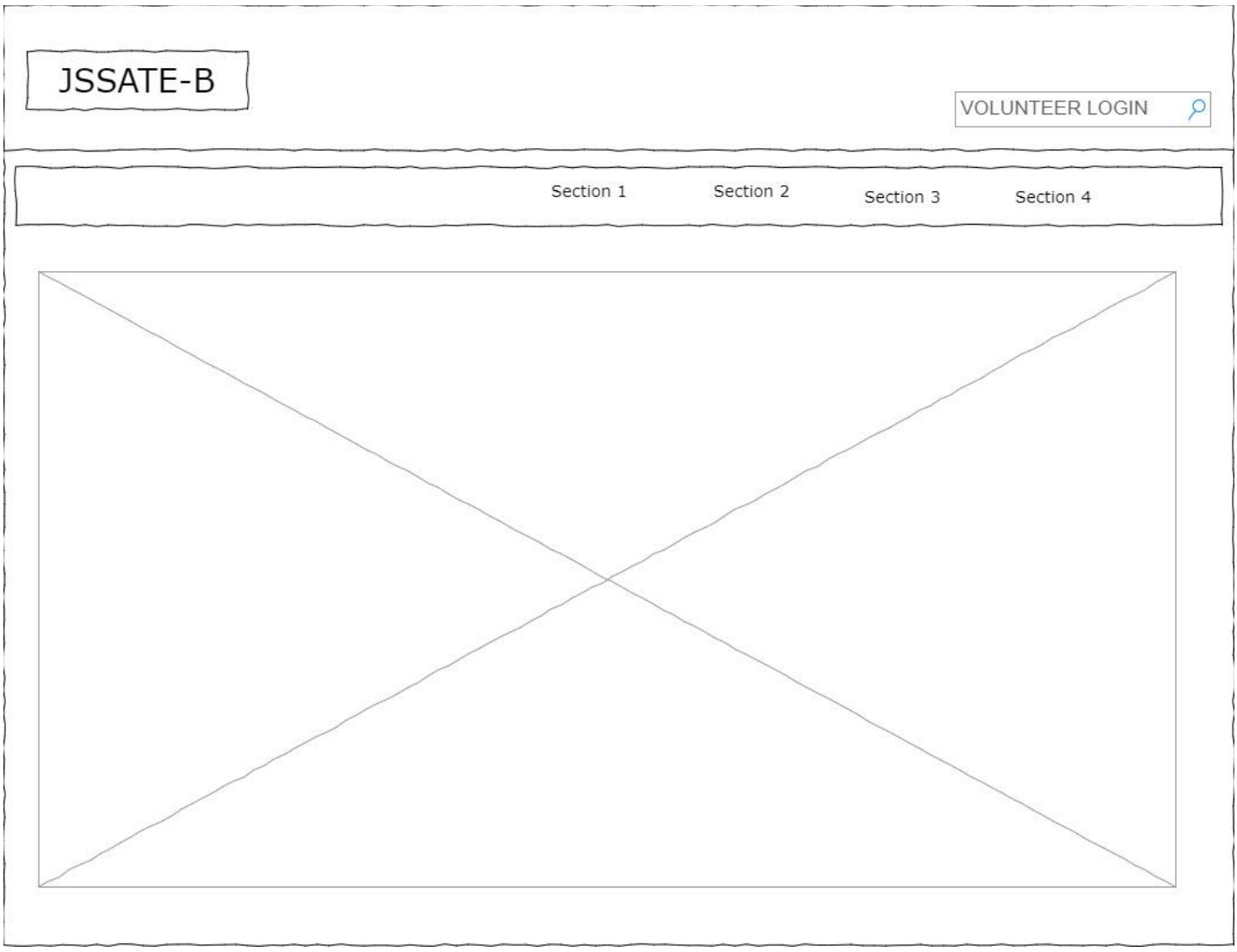


Fig 4.5 Wireframe of the Home Page

Description: Login Page

The wireframe for the login page would prioritize usability and clarity, featuring input fields for email, password and userType. It is shown in Fig 4.6.

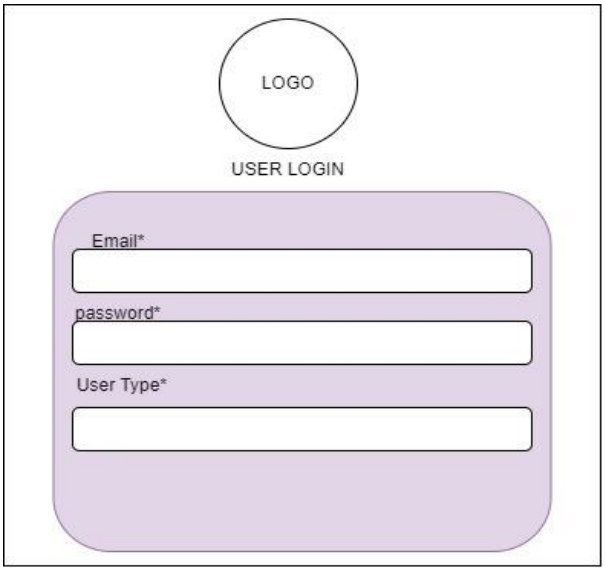


Fig 4.6 Login Page

Description: Display Activity Details

The wireframe for displaying activity details would present a list of clickable activity cards, each featuring essential information such as activity name, description, date, and organizer. It would offer intuitive selection buttons or checkboxes to enable volunteers to choose their preferred activities easily. Additionally, the wireframe would include filters and navigation options to streamline the browsing experience and facilitate efficient activity selection for volunteers. It is shown in Fig 4.7.

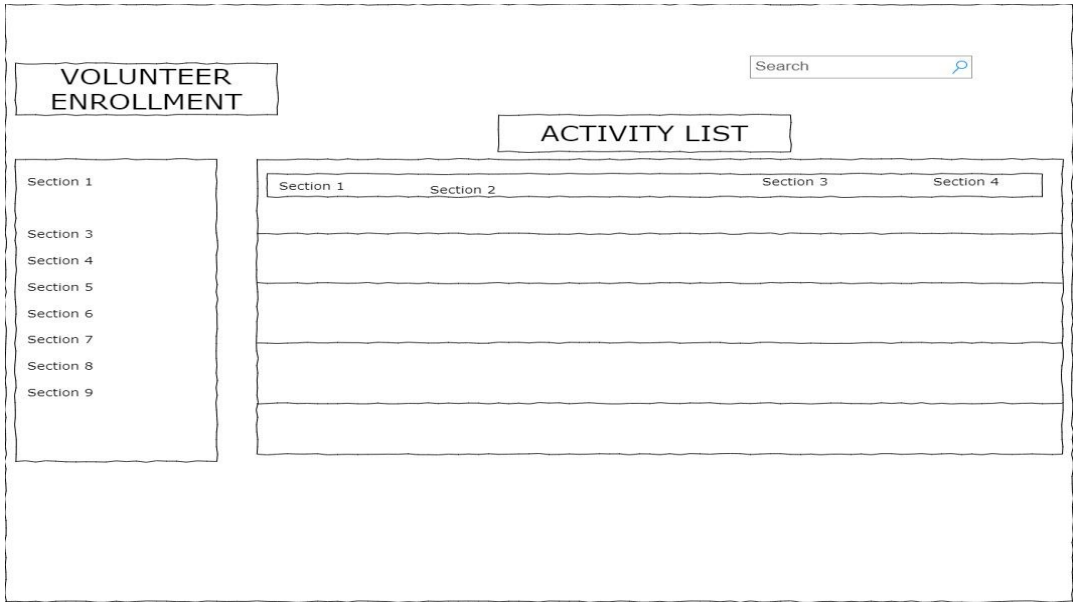


Fig 4.7 Activity Selection Page

STRUCTURE OF THE DATABASE

```
mysql> USE NSS1;
Database changed
mysql> SHOW TABLES;
+----------------------+
| Tables_in_nss1       |
+----------------------+
| activity             |
| activity_details     |
| attendance           |
| attendance_tracking  |
| users                |
| volunteers           |
+----------------------+
6 rows in set (0.01 sec)

mysql> SELECT
    ->     CONSTRAINT_NAME,
    ->     TABLE_NAME,
    ->     COLUMN_NAME,
    ->     REFERENCED_TABLE_NAME,
    ->     REFERENCED_COLUMN_NAME
    -> FROM
    ->     INFORMATION_SCHEMA.KEY_COLUMN_USAGE
    -> WHERE
    ->     TABLE_SCHEMA = 'nss1'
    ->     AND TABLE_NAME IN ('users', 'volunteers', 'activity', 'activity_details', 'attendance_tracking', 'attendance');
+--------------------------------+---------------------+--------------+-----------------------+------------------------+
| CONSTRAINT_NAME                | TABLE_NAME          | COLUMN_NAME  | REFERENCED_TABLE_NAME | REFERENCED_COLUMN_NAME |
+--------------------------------+---------------------+--------------+-----------------------+------------------------+
| PRIMARY                        | activity            | EnrollmentID | NULL                  | NULL                   |
| PRIMARY                        | activity            | ActivityID   | NULL                  | NULL                   |
| fk_activity                    | activity            | ActivityID   | activity_details      | ActivityID             |
| fk_enrollment                  | activity            | EnrollmentID | volunteers            | EnrollmentID           |
| PRIMARY                        | activity_details    | ActivityID   | NULL                  | NULL                   |
| PRIMARY                        | attendance          | EnrollmentID | NULL                  | NULL                   |
| PRIMARY                        | attendance          | ActivityID   | NULL                  | NULL                   |
| fk_attendance_activity         | attendance          | EnrollmentID | activity              | EnrollmentID           |
| fk_attendance_activity         | attendance          | ActivityID   | activity              | ActivityID             |
| fk_attendance_tracking_activity | attendance_tracking | EnrollmentID | activity              | EnrollmentID           |
| fk_attendance_tracking_activity | attendance_tracking | EventID      | activity              | ActivityID             |
| PRIMARY                        | users               | username     | NULL                  | NULL                   |
| PRIMARY                        | users               | email        | NULL                  | NULL                   |
| PRIMARY                        | volunteers          | EnrollmentID | NULL                  | NULL                   |
| volunteers_ibfk_1              | volunteers          | EmailID      | users                 | email                  |
+--------------------------------+---------------------+--------------+-----------------------+------------------------+
15 rows in set (0.02 sec)

mysql>
```

Fig 4.7 Constraints and Relation Schema of NSS Management System

The database consists of six tables, each serving a distinct purpose within the NSS management system. The **USERS** table stores user credentials, including usernames, emails, passwords, and user types, essential for authentication. **VOLUNTEERS** table houses personal details and enrollment information of volunteers, facilitating management of volunteer data. **ACTIVITY_DETAILS** table contains descriptive information about various activities, aiding in organizing and categorizing activities. **ACTIVITY** table links volunteers with specific activities, tracking their participation and roles. **ATTENDANCE_TRACKING** table records attendance details such as date, time, and location, providing insights into volunteer participation. **ATTENDANCE** table correlates volunteer attendance with specific activities, enabling monitoring of completed hours. The **ATTENDANCE** table gets updated by the **TRIGGER** (upon insertion to the **ATTENDANCE_TRACKING** table)

CHAPTER 5

PROJECT IMPLEMENTATION

INTRODUCTION

Implementation is a crucial stage where the theoretical design of the NSS Management System is transformed into a functional system. This phase is vital for ensuring the system meets the requirements and instills confidence in users regarding its effectiveness.

Implementation can proceed only after thorough testing to ensure adherence to specifications. It involves meticulous planning, examination of the current system and its limitations, designing methods for changeover, and evaluating changeover strategies. Education and training of users, along with system testing, are essential components of implementation.

The following codes will facilitate the comprehensive implementation of our NSS Management System design.

CREATING TABLES

TABLE USERS

```
CREATE TABLE USERS (
username VARCHAR(50) NOT NULL,
email VARCHAR(100) NOT NULL,
password VARCHAR(255),
userType ENUM('Volunteer','Team Leader','Guide','Administrator'),
PRIMARY KEY (username, email),
INDEX (email)
);
```

The Description of Users Table is given in Fig 5.1

```
mysql> DESCRIBE USERS;
+----------+-----------------------------------------------------------+------+-----+---------+-------+
| Field    | Type                                                      | Null | Key | Default | Extra |
+----------+-----------------------------------------------------------+------+-----+---------+-------+
| username | varchar(50)                                               | NO   | PRI | NULL    |       |
| email    | varchar(100)                                              | NO   | PRI | NULL    |       |
| password | varchar(255)                                              | YES  |     | NULL    |       |
| userType | enum('Volunteer','Team Leader','Guide','Administrator')   | YES  |     | NULL    |       |
+----------+-----------------------------------------------------------+------+-----+---------+-------+
4 rows in set (0.03 sec)
```

Fig 5.1 USERS Table

TABLE VOLUNTEERS

```
CREATE TABLE VOLUNTEERS (
  EnrollmentID VARCHAR(20) NOT NULL,
  FirstName VARCHAR(50),
  MiddleName VARCHAR(50),
  LastName VARCHAR(50),
  Gender ENUM('Male','Female','Other'),
  DateOfBirth DATE,
  Branch VARCHAR(50),
  NSSUnitNumber VARCHAR(50),
  Community VARCHAR(50),
  PhoneNumber VARCHAR(15),
  EmailID VARCHAR(255),
  AadharNumber VARCHAR(20),
  Height DECIMAL(5,2),
  Weight DECIMAL(5,2),
  FatherName VARCHAR(100),
  Address VARCHAR(255),
  BloodGroup VARCHAR(5),
  PRIMARY KEY (EnrollmentID),
  FOREIGN KEY (EmailID) REFERENCES USERS(email)
  );
```

The Description of Volunteers Table is given in Fig 5.2

```
mysql> DESCRIBE VOLUNTEERS;
+---------------+------------------------------+------+-----+---------+-------+
| Field         | Type                         | Null | Key | Default | Extra |
+---------------+------------------------------+------+-----+---------+-------+
| EnrollmentID  | varchar(20)                  | NO   | PRI | NULL    |       |
| FirstName     | varchar(50)                  | YES  |     | NULL    |       |
| MiddleName    | varchar(50)                  | YES  |     | NULL    |       |
| LastName      | varchar(50)                  | YES  |     | NULL    |       |
| Gender        | enum('Male','Female','Other') | YES  |     | NULL    |       |
| DateOfBirth   | date                         | YES  |     | NULL    |       |
| Branch        | varchar(50)                  | YES  |     | NULL    |       |
| NSSUnitNumber | varchar(50)                  | YES  |     | NULL    |       |
| Community     | varchar(50)                  | YES  |     | NULL    |       |
| PhoneNumber   | varchar(15)                  | YES  |     | NULL    |       |
| EmailID       | varchar(255)                 | YES  | MUL | NULL    |       |
| AadharNumber  | varchar(20)                  | YES  |     | NULL    |       |
| Height        | decimal(5,2)                 | YES  |     | NULL    |       |
| Weight        | decimal(5,2)                 | YES  |     | NULL    |       |
| FatherName    | varchar(100)                 | YES  |     | NULL    |       |
| Address       | varchar(255)                 | YES  |     | NULL    |       |
| BloodGroup    | varchar(5)                   | YES  |     | NULL    |       |
+---------------+------------------------------+------+-----+---------+-------+
17 rows in set (0.00 sec)
```

Fig 5.2 VOLUNTEERS Table

TABLE ACTIVITY_DETAILS

```
CREATE TABLE ACTIVITY_DETAILS (
ActivityID INT NOT NULL,
Description TEXT,
Duration INT,
Organizer VARCHAR(100),
PRIMARY KEY (ActivityID)
);
```

The Description of Activity_Details Table is given in Fig 5.3

```
mysql> DESCRIBE ACTIVITY_DETAILS;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| ActivityID  | int          | NO   | PRI | NULL    |       |
| Description | text         | YES  |     | NULL    |       |
| Duration    | int          | YES  |     | NULL    |       |
| Organizer   | varchar(100) | YES  |     | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Fig 5.3 ACTIVITY_DETAILS Table

TABLE ACTIVITY

```
CREATE TABLE ACTIVITY (
EnrollmentID VARCHAR(20) NOT NULL,
ActivityID INT NOT NULL,
Name VARCHAR(100),
Description TEXT,
Duration INT,
Organizer VARCHAR(100),
PRIMARY KEY (EnrollmentID, ActivityID),
CONSTRAINT fk_activity FOREIGN KEY (ActivityID) REFERENCES
ACTIVITY_DETAILS(ActivityID),
CONSTRAINT fk_enrollment FOREIGN KEY (EnrollmentID) REFERENCES
VOLUNTEERS(EnrollmentID)
);
```

The Description of Activity Table is given in Fig 5.4

```
mysql> DESCRIBE ACTIVITY;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| EnrollmentID | varchar(20)  | NO   | PRI | NULL    |       |
| ActivityID   | int          | NO   | PRI | NULL    |       |
| Name         | varchar(100) | YES  |     | NULL    |       |
| Description  | text         | YES  |     | NULL    |       |
| Duration     | int          | YES  |     | NULL    |       |
| Organizer    | varchar(100) | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
6 rows in set (0.00 sec)
```

Fig 5.4 ACTIVITY Table

TABLE ATTENDANCE_TRACKING

```
CREATE TABLE ATTENDANCE_TRACKING (
EnrollmentID VARCHAR(20) NOT NULL,
EventID INT NOT NULL,
Date DATE,
CheckInTime TIME,
CheckOutTime TIME,
Location VARCHAR(100),
DailyHours INT,
CONSTRAINT fk_attendance_tracking_activity FOREIGN KEY (EnrollmentID, EventID)
REFERENCES ACTIVITY(EnrollmentID, ActivityID)
);
```

The Description of Attendance_Tracking Table is given in Fig 5.5

```
mysql> DESCRIBE ATTENDANCE_TRACKING;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| EnrollmentID | varchar(20)  | NO   | MUL | NULL    |       |
| EventID      | int          | NO   |     | NULL    |       |
| Date         | date         | YES  |     | NULL    |       |
| CheckInTime  | time         | YES  |     | NULL    |       |
| CheckOutTime | time         | YES  |     | NULL    |       |
| Location     | varchar(100) | YES  |     | NULL    |       |
| DailyHours   | int          | YES  |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

Fig 5.5 ATTENDANCE_TRACKING Table

TABLE ATTENDANCE

```
CREATE TABLE ATTENDANCE (
EnrollmentID VARCHAR(20) NOT NULL,
ActivityID INT NOT NULL,
Name VARCHAR(100),
Description TEXT,
Duration INT,
Organizer VARCHAR(100),
Completed_hours INT,
PRIMARY KEY (EnrollmentID, ActivityID),
CONSTRAINT fk_attendance_activity_tracking FOREIGN KEY (EnrollmentID, ActivityID)
REFERENCES ACTIVITY(EnrollmentID, EventID)
);
```

The Description of Attendance Table is given in Fig 5.6

```
mysql> DESCRIBE ATTENDANCE;
+-----------------+--------------+------+-----+---------+-------+
| Field           | Type         | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
| EnrollmentID    | varchar(20)  | NO   | PRI | NULL    |       |
| ActivityID      | int          | NO   | PRI | NULL    |       |
| Name            | varchar(100) | YES  |     | NULL    |       |
| Description     | text         | YES  |     | NULL    |       |
| Duration        | int          | YES  |     | NULL    |       |
| Organizer       | varchar(100) | YES  |     | NULL    |       |
| Completed_hours | int          | YES  |     | NULL    |       |
+-----------------+--------------+------+-----+---------+-------+
7 rows in set (0.00 sec)
```

Fig 5.6 ATTENDANCE Table

QUERIES

ACTIVITY TABLE

Query:

This query retrieves information about the activities a volunteer (EnrollmentID = '11') has participated in. (Including the percentage of work done)

```
SELECT
    a.ActivityID,
    ad.Description AS ActivityDescription,
    ad.Duration AS TotalDuration,
    SUM(IFNULL(at.Completed_hours, 0)) AS CompletedHours,
    (SUM(IFNULL(at.Completed_hours, 0)) / ad.Duration) * 100 AS PercentageOfWorkDone
FROM
    ACTIVITY a
JOIN
    ACTIVITY_DETAILS ad ON a.ActivityID = ad.ActivityID
LEFT JOIN
    ATTENDANCE at ON a.EnrollmentID = at.EnrollmentID AND a.ActivityID = at.ActivityID
WHERE
    a.EnrollmentID = '11'
GROUP BY
    a.ActivityID;
```

The output of the above query is given in Fig 5.7

```
+------------+----------------------+---------------+----------------+----------------------+
| ActivityID | ActivityDescription  | TotalDuration | CompletedHours | PercentageOfWorkDone |
+------------+----------------------+---------------+----------------+----------------------+
|        100 | Plant Adoption Drive |            10 |              7 |              70.0000 |
|        200 | Blood Donation Drive |             2 |              0 |               0.0000 |
+------------+----------------------+---------------+----------------+----------------------+
```

Fig 5.7 Query 1

Description:

SQL JOINS are used to retrieve data from multiple tables. A SQL JOIN is performed whenever two or more tables are listed in a SQL statement.

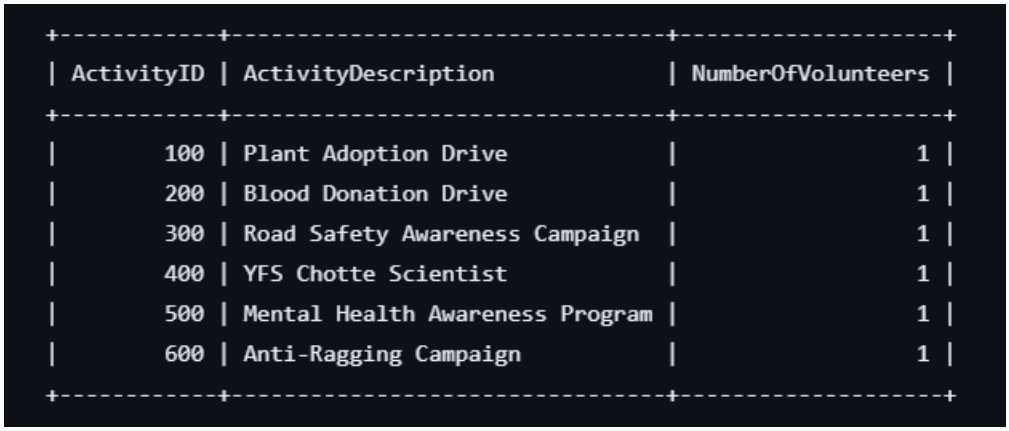
Activity and Activity_details

Query:

This query retrieves all activities along with the count of volunteers participating in each activity.

```
SELECT
   a.ActivityID,
   ad.Description AS ActivityDescription,
   COUNT(DISTINCT a.EnrollmentID) AS NumberOfVolunteers
FROM
   ACTIVITY a
JOIN
   ACTIVITY_DETAILS ad ON a.ActivityID = ad.ActivityID
GROUP BY
   a.ActivityID, ad.Description;
```

The output of the above query is given in Fig 5.8



ActivityID	ActivityDescription	NumberOfVolunteers
100	Plant Adoption Drive	1
200	Blood Donation Drive	1
300	Road Safety Awareness Campaign	1
400	YFS Chotte Scientist	1
500	Mental Health Awareness Program	1
600	Anti-Ragging Campaign	1

Fig 5.8 Query 2

Description:

GROUP BY is a SQL clause used to group rows that have the same values into summary rows. It's often used in conjunction with aggregate functions like SUM, COUNT, AVG, MAX, and MIN to perform calculations on these groups. This clause is particularly useful for generating summary reports or analyzing data based on specific criteria. When used with SELECT statements, GROUP BY divides the result set into groups, based on one or more columns specified in the GROUP BY clause. The result is a set of summary rows, where each row represents a unique combination of values in the grouped columns.

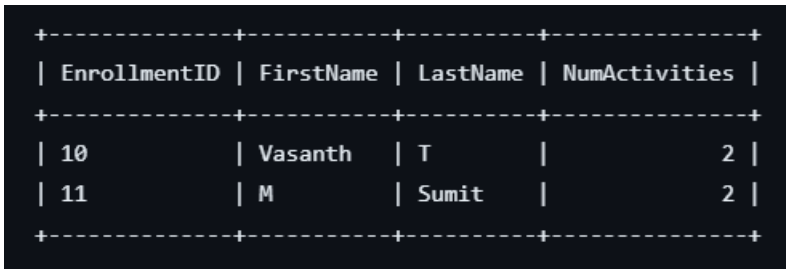
Activity and Volunteers

Query:

This query identifies volunteers who have participated in multiple activities by counting the distinct activities for each volunteer.

```
SELECT
   v.EnrollmentID,
   v.FirstName,
   v.LastName,
   COUNT(DISTINCT a.ActivityID) AS NumActivities
FROM
   VOLUNTEERS v
JOIN
   ACTIVITY a ON v.EnrollmentID = a.EnrollmentID-
GROUP BY
   v.EnrollmentID, v.FirstName, v.LastName
HAVING
   COUNT(DISTINCT a.ActivityID) > 1;
```

The output of the above query is given in Fig 5.9



EnrollmentID	FirstName	LastName	NumActivities
10	Vasanth	T	2
11	M	Sumit	2

Fig 5.9 Query 3

Description:

MySQL's aggregate function is used to perform calculations on multiple values and return the result in a single value like the average of all values**,** the sum of all values, and maximum & minimum value among certain groups of values. We mostly use the aggregate functions with SELECT statements in the data query languages.

***Syntax*:**

The following are the syntax to use aggregate functions in MySQL:

*function_name (**DISTINCT** | ALL expression)*

In the previous syntax, we had used the following parameters:

- First, we need to specify the name of the aggregate function.
- Second, we use the **DISTINCT** modifier when we want to calculate the result based on distinct values or **ALL** modifiers when we calculate all values, including duplicates. The default is ALL.
- Third, we need to specify the expression that involves columns and arithmetic operators.

Here we have used MAX( ), MIN ( ) and SUM ( ) aggregate functions to perform mathematical operations on the *funding* attribute.

TRIGGERS

A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

Syntax:

```
create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
[for each row]
[trigger_body]
```

Trigger for ATTENDANCE

The trigger update_completed_hours is designed to automatically update the Completed_hours column in the attendance table whenever a new record is inserted into the attendance_tracking table.

```
CREATE TRIGGER update_completed_hours

AFTER INSERT ON attendance_tracking

FOR EACH ROW

BEGIN

    UPDATE attendance

    SET Completed_hours = Completed_hours + NEW.DailyHours

    WHERE EnrollmentID = NEW.EnrollmentID AND ActivityID = NEW.EventID;

END
```

PSEUDO CODE

Pseudocode is an informal high-level description of the operating principle of a computer program or other algorithm. It uses the structural conventions of a normal programming language, but is intended for human reading rather than machine reading.

Algorithm for Table Display

Step1: BEGIN

Step 2: Establish connection with the database using the username and password of the database.

Step 3: Define the select query to retrieve all the values from the DBMS

Step 4: Define Default Table Model for the table and fetch the details from the database.

Step 5: END

Algorithm for Insert

Step 1: BEGIN

Step 2: Get all the necessary values required for insertion into variable defined in the method.

Step 3: Define the query for insertion as stated above.

Step 4: Execute the Query using the **(Select * from)** the required table.

Step 5: END

Algorithm for Update

Step 1: BEGIN

Step 2: Get all the necessary values required for updating the values into the variable defined in the method.

Step 3: UPDATE table name

SET column1 = value1, column2 = value2, ...

WHERE condition;

Step 4: Define the Query for Updating as stated above.

Step 5: Execute the Query using the **executeUpdate()** method defined.

Step 6: END

Algorithm for Delete

Step 1: BEGIN

Step 2: Get the model number of the instrument which is to be deleted into a variable defined in the method.

Step 3: Delete from table_name where condition;

Step 4: Define the Query for deleting as stated above.

Step 5: Execute the Query using the **executeUpdate()** method defined.

Step 6: END

Algorithm for Joins

Nested Loop Join

In the nested loop join algorithm, for each tuple in outer relation, we have to compare it with all the tuples in the inner relation then only the next tuple of outer relation is considered. All pairs of tuples which satisfy the condition are added in the result of the join.

for each tuple t_R in T_R**do**

for each tuple t_s in T_s**do**

compare (t_R, t_s) if they satisfy the condition add them in the result of the join

end

end

Block Nested Loop Join:

In block nested loop join, for a block of outer relation, all the tuples in that block are compared with all the tuples of the inner relation, then only the next block of outer relation is considered. All pairs of tuples which satisfy the condition are added in the result of the join.

for each block b_R in B_R**do**

for each block b_s in B_s**do**

for each tuple t_R in T_R**do**

for each tuple t_s in T_s**do**

compare (t_R, t_s) if they satisfy the condition

add them in the result of the join

end

end

end

end

Algorithm for Triggers

Step 1: create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.

Step 2: [before | after]: This specifies when the trigger will be executed.

Step 3: {insert | update | delete}: This specifies the DML operation.

Step 4: on [table_name]: This specifies the name of the table associated with the trigger.

Step 5: [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.

Step 6: [trigger_body]: This provides the operation to be performed as trigger is fired

HTML CODE

register.html

```
<html>
<body>
  <h2>User Registration Form</h2>
  <form id="registrationForm" style="width: 33%; margin-bottom: 2rem;">
    <div>
      <label for="username">User Name:</label>
      <input type="text" id="username" name="username" required>
    </div><br>
    <div>
      <label for="email">Email-ID:</label>
      <input type="email" id="email" name="email" required>
    </div><br>
    <div>
      <label for="password">Password:</label>
      <input type="password" id="password" name="password" required>
    </div><br>
    <div>
      <label for="userType">User Type:</label>
      <select id="userType" name="userType" required>
        <option value="">Select User Type</option>
        <option value="Volunteer">Volunteer</option>
        <option value="Team Leader">Team Leader</option>
        <option value="Guide">Guide</option>
        <option value="Administrator">Administrator</option>
      </select>
    </div><br>
    <div>
      <center>
        <button type="submit">Submit</button>
      </center>
    </div>
  </form>
  <script src="register.js"></script>
</body>
```

```
</html>
```

Similar html pages are created to accept user details like volunteer's details, administrator's details, activity details and attendance tracking details.

JAVASCRIPT CODE

register.js

```
document.getElementById('registrationForm').addEventListener('submit', async (event) => {
    event.preventDefault(); // Prevent the default form submission
    const form = event.target;
    const formData = {
        username: form.elements.username.value,
        email: form.elements.email.value,
        password: form.elements.password.value,
        userType: form.elements.userType.value
    };try {
        const response = await fetch('http://localhost:8000/submit_form', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(formData)
        });
        if (response.ok) {
            console.log('Form submitted successfully');
            alert('Form submitted successfully');
            form.reset(); // Reset the form after successful submission
        } else {
            console.error('Error submitting form');
            alert('Error submitting form');
        }
    } catch (error) {
        console.error('Error submitting form:', error);
        alert('Error submitting form');
    }
});
```

The following code is to allow volunteers to opt for the activities (which are present in the activity details table)

```
<script>
    document.addEventListener('DOMContentLoaded', () => {
        // Retrieve enrollment ID and full name from localStorage
        const enrollmentID = localStorage.getItem('enrollmentID');
        const fullName = localStorage.getItem('fullName');

        // Set enrollment ID and full name as values of input fields
        document.getElementById('enrollment-id').value = enrollmentID;
        document.getElementById('volunteer-name').value = fullName;
    });
</script>
<script>
    // Add an event listener to the attendanceButton
    document.getElementById('attendanceButton').addEventListener('click', () => {
        // Open attendance_tracking.html in a new tab
        window.open('attendance_tracking.html', '_blank');
    });
    async function fetchActivityData() {
        try {
            const response = await fetch('http://localhost:8000/activity_data');
            if (!response.ok) {
                throw new Error('Failed to fetch data');
            }
            const data = await response.json();
            displayActivityData(data);
        } catch (error) {
            console.error('Error fetching activity data:', error);
        }
    }
    function displayActivityData(data) {
        const activityDataElement = document.getElementById('activityData');
        activityDataElement.innerHTML = '';
        data.forEach(activity => {
            const row = document.createElement('tr');
            Object.values(activity).forEach(value => {
                const cell = document.createElement('td');
                cell.textContent = value;
                row.appendChild(cell);
```

```
        });
        const buttonCell = document.createElement('td');
        const button = document.createElement('button');
        button.textContent = '+';
        button.addEventListener('click', () => {
            populateFields(activity);
        });
        buttonCell.appendChild(button);
        row.appendChild(buttonCell);
        activityDataElement.appendChild(row);
    });
}
function populateFields(activity) {
    document.getElementById('activity-id').value = activity['ActivityID'];
    document.getElementById('activity-name').value = activity['Description'];
    document.getElementById('duration').value = activity['Duration'];
    document.getElementById('organizer').value = activity['Organizer'];
}
fetchActivityData();
// Add the event listener for form submission
document.getElementById('display_activity').addEventListener('submit', async (event) => {
    event.preventDefault(); // Prevent the default form submission
    const form = event.target;
    const formData = {
        enrollmentid: form.elements['enrollment-id'].value,
        volunteername: form.elements['volunteer-name'].value,
        activityid: form.elements['activity-id'].value,
        activityname: form.elements['activity-name'].value,
        duration: form.elements['duration'].value,
        organizer: form.elements['organizer'].value
    };
    try {
        const response = await fetch('http://localhost:8000/display_activity', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(formData)
```

```
            });
            if (response.ok) {
                console.log('Activity added successfully');
                alert('Activity added successfully');
                form.reset(); // Reset the form after successful addition
                fetchActivityData(); // Update the activity list after adding a new activity
            } else {
                console.error('Error adding activity');
                alert('Error adding activity');
            }
        } catch (error) {
            console.error('Error adding activity:', error);
            alert('Error adding activity');
        }
    });
</script>
<script>
    document.getElementById('enrollForm').addEventListener('submit', async (event) => {
        event.preventDefault(); // Prevent the default form submission

        const form = event.target;
        const formData = new FormData(form);

        const jsonObject = {};
        formData.forEach((value, key) => {
            jsonObject[key] = value;
        });
        try {
            const response = await fetch('http://localhost:8000/enroll', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(jsonObject)
            });
            const responseData = await response.json();
            if (response.ok) {
                console.log('Volunteer enrolled successfully');
```

```
                alert('Volunteer enrolled successfully');
                form.reset(); // Reset the form after successful enrollment
                window.open('display_activity_details.html', '_blank'); // Open display_activity_details.html in
a new tab
            } else {
                console.error('Error enrolling volunteer');
                alert('Error enrolling volunteer');
            }
        } catch (error) {
            console.error('Error enrolling volunteer:', error);
            alert('Error enrolling volunteer');
        }
    });
  </script>
```

index.js

```
const express = require('express');
const cors = require('cors');
const mysql = require('mysql2');
const bodyParser = require('body-parser');
const app = express();
const port = 8000;
// Enable CORS
app.use(cors());
// Create MySQL connection
const db = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'VASANTH',
  database: 'nss'
});
// Connect to MySQL
db.connect((err) => {
  if (err) {
    throw err;
  }
  console.log('Connected to MySQL database');
```

```
});
// Middleware to parse JSON bodies
app.use(bodyParser.json());
// Route to handle form submission
app.post('/submit_form', (req, res) => {
  const { username, email, password, userType } = req.body;
  console.log('Received form data:', req.body); // Log the request body

  // Check if username field is provided and not empty
  if (!username || username.trim() === '') {
    console.error('Username is required');
    return res.status(400).send('Username is required');
  }
  const sql = 'INSERT INTO users (username, email, password, userType) VALUES (?, ?, ?, ?)';
  db.query(sql, [username, email, password, userType], (err, result) => {
    if (err) {
      console.error('Error executing SQL query:', err);
      return res.status(500).send('Error submitting form'); // Log the error
    }
    console.log('Form submitted successfully');
    res.status(200).send('Form submitted successfully');
  });
});
app.post('/login', (req, res) => {
  const { VOLUNTEER_NAME, VOLUNTEER_PASSWORD, USER_TYPE } = req.body;
  // Check if all required fields are provided
  if (!VOLUNTEER_NAME || !VOLUNTEER_PASSWORD || !USER_TYPE) {
      console.error('Missing required fields');
      return res.status(400).send('Missing required fields');
  }
  // Query to check if the user exists in the table
  const sql = 'SELECT email , password , userType FROM users WHERE email = ? AND password
= ? AND userType = ?';
  db.query(sql, [VOLUNTEER_NAME, VOLUNTEER_PASSWORD, USER_TYPE], (err, result) =>
{
      if (err) {
          console.error('Error executing SQL query:', err);
          return res.status(500).send('Error logging in');
```

```
        }
        // Check if the user exists
        if (result.length === 1) {
            console.log('Login successful');
            // Send success message to client
            res.status(200).send('Login successful');
        } else {
            console.error('Invalid credentials');
            return res.status(401).send('Invalid credentials');
        }
    });
});
// Route to handle volunteer enrollment
app.post('/enroll', (req, res) => {
  const
{enrollmentID,firstName,middleName,lastName,gender,dateOfBirth,branch,nssUnitNumber,commun
ity,phoneNumber,emailID,aadharNumber,height,weight,fatherName,address,bloodGroup} =
req.body;
  // Insert the volunteer record into the database
  const sql = `INSERT INTO volunteers (EnrollmentID, FirstName, MiddleName, LastName, Gender,
DateOfBirth, Branch, NSSUnitNumber, Community, PhoneNumber, EmailID, AadharNumber,
Height, Weight, FatherName, Address, BloodGroup)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)`;
  db.query(sql, [enrollmentID, firstName, middleName, lastName, gender, dateOfBirth, branch,
nssUnitNumber, community, phoneNumber, emailID, aadharNumber, height, weight, fatherName,
address, bloodGroup], (err, result) => {
    if (err) {
        console.error('Error executing SQL query:', err);
        return res.status(500).send('Error enrolling volunteer: ' + err.message);
    }
    console.log('Volunteer enrolled successfully');
    res.status(200).send('Volunteer enrolled successfully');
  });
});
// Endpoint to handle attendance submission
app.post('/attendance', (req, res) => {
  const { enrollmentID, eventID, eventdate, CheckInTime, CheckOutTime, location, dailyHours } =
req.body;
  const sql = `INSERT INTO attendance_tracking(EnrollmentID, EventID, Date, CheckInTime,
```

```
CheckOutTime, Location, DailyHours) VALUES(?, ?, ?, ?, ?, ?, ?)`;
  db.query(sql, [enrollmentID, eventID, eventdate, CheckInTime, CheckOutTime, location,
dailyHours], (err, result) => {
    if (err) {
      console.error('Error inserting attendance:', err);
      res.status(500).json({ error: 'Error inserting attendance' });
      return;
    }
    console.log('Attendance inserted successfully');
    res.status(200).json({ message: 'Attendance inserted successfully' });
  });
});
// Route to handle report issuance
app.post('/issue_report', (req, res) => {
  const { eventID, fullName, enrollmentID, description } = req.body;
  console.log('Received report data:', req.body); // Log the request body
  try {
    // Check if all required fields are provided
    if (!eventID || !fullName || !enrollmentID || !description) {
      console.error('Missing required fields');
      return res.status(400).send('Missing required fields');
    }
    // Perform database insert operation
    const sql = 'INSERT INTO report_issuing (EventID, FullName, EnrollmentID, Description)
VALUES (?, ?, ?, ?)';
    db.query(sql, [eventID, fullName, enrollmentID, description], (err, result) => {
      if (err) {
        console.error('Error executing SQL query:', err);
        return res.status(500).send('Error issuing report: ' + err.message); // Log the error
      }
      console.log('Report issued successfully');
      res.status(200).send('Report issued successfully');
    });
  } catch (error) {
    console.error('Error handling report issuance:', error);
    res.status(500).send('Error handling report issuance');
  }
});
```

```
// Route to handle adding activity details
app.post('/add_activity', (req, res) => {
  const { activityID, description, duration, organizer } = req.body;
  // Insert the activity details into the database
  const sql = 'INSERT INTO activity_details (ActivityID, Description, Duration, Organizer) VALUES (?, ?, ?, ?)';
  db.query(sql, [activityID, description, duration, organizer], (err, result) => {
    if (err) {
      console.error('Error executing SQL query:', err);
      return res.status(500).json({ error: 'Error adding activity details' });
    }
    console.log('Activity details added successfully');
    res.status(200).json({ message: 'Activity details added successfully' });
  });
});
// Route to fetch attendance data
app.get('/attendance_data', (req, res) => {
  const sql = 'SELECT * FROM attendance';
  db.query(sql, (err, result) => {
    if (err) {
      console.error('Error executing SQL query:', err);
      return res.status(500).send('Error fetching attendance data');
    }
    console.log('Attendance data:', result);
    res.json(result);
  });
});
// Route to handle adding activity details and inserting into attendance table
app.post('/add_activity', (req, res) => {
  const { activityID, description, duration, organizer } = req.body;
  // Insert the activity details into the database
  const sql = 'INSERT INTO activity_details (ActivityID, Description, Duration, Organizer) VALUES (?, ?, ?, ?)';
  db.query(sql, [activityID, description, duration, organizer], (err, result) => {
    if (err) {
      console.error('Error executing SQL query:', err);
      return res.status(500).json({ error: 'Error adding activity details' });
    }
```

```
    console.log('Activity details added successfully');
    // Insert the same details into the attendance table with Completed_hours set to 0
    const attendanceSql = 'INSERT INTO attendance (ActivityID, Description, Duration, Organizer, Completed_hours) VALUES (?, ?, ?, ?, ?)';
    db.query(attendanceSql, [activityID, description, duration, organizer, 0], (err, result) => {
      if (err) {
        console.error('Error executing SQL query:', err);
        return res.status(500).json({ error: 'Error adding activity details to attendance table' });
      }
      console.log('Activity details added to attendance table successfully');
      res.status(200).json({ message: 'Activity details added successfully' });
    });
  });
});
// Route to handle adding and displaying activity details
app.post('/display_activity', (req, res) => {
  const { enrollmentid, volunteername, activityid, activityname, duration, organizer } = req.body;
  // Insert the activity details into the database
  const activitySql = 'INSERT INTO activity (EnrollmentID, ActivityID, Name, Description, Duration, Organizer) VALUES (?, ?, ?, ?, ?, ?)';
  db.query(activitySql, [enrollmentid, activityid, volunteername, activityname, duration, organizer], (err, result) => {
    if (err) {
      console.error('Error executing SQL query:', err);
      return res.status(500).json({ error: 'Error adding activity details' });
    }
    console.log('Activity details added successfully');
    // Insert the same details into the attendance table with Completed_hours set to 0
    const attendanceSql = 'INSERT INTO attendance (EnrollmentID, ActivityID, Name, Description, Duration, Organizer, Completed_hours) VALUES (?, ?, ?, ?, ?, ?, ?)';
    db.query(attendanceSql, [enrollmentid, activityid, volunteername, activityname, duration, organizer, 0], (err, result) => {
      if (err) {
        console.error('Error executing SQL query:', err);
        return res.status(500).json({ error: 'Error adding activity details to attendance table' });
      }
      console.log('Activity details added to attendance table successfully');
      res.status(200).json({ message: 'Activity details added successfully' });
    });
```

```
  });});
app.get('/student_data', (req, res) => {
  const sql = 'SELECT * FROM  activity_details';
  db.query(sql, (err, result) => {
     if (err) {
        console.error('Error executing SQL query:', err);
        return res.status(500).send('Error fetching student data');
     }
     console.log('Student data:', result);
     res.json(result);
  });
});
// Route to fetch activity data
app.get('/activity_data', (req, res) => {
  const sql = 'SELECT ActivityID, Description, Duration ,Organizer FROM activity_details';
  db.query(sql, (err, result) => {
     if (err) {
        console.error('Error executing SQL query:', err);
        return res.status(500).send('Error fetching activity data');
     }
     console.log('Activity data:', result);
     res.json(result);
  });
});
// Start the server`
app.listen(port, () => {
  console.log(`Server is running on http://localhost:${port}`);
})
```

CHAPTER 6

SYSTEM TESTING

INTRODUCTION

Testing plays a vital role in the success of the system. System testing makes a logical assumption that if all parts of the system are correct, the goal will be successfully achieved. Once program code has been developed, testing begins. The testing process focuses on the logical internals of the software, ensuring that all statements have been tested, and on the functional externals, that is conducted tests to uncover errors and ensure that defined input will produce actual results that agree with required results. Broadly speaking, there are at least three levels of testing: unit testing, integration testing, and system testing

TYPES OF TESTING

Unit testing

Unit testing refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level,
and the minimal unit tests include the constructors and destructors. These types of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to ensure that the building blocks of the software work independently from each other.

Unit testing is a software development process that involves a synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs. It is performed by the software developer or engineer during the construction phase of the software development lifecycle. Unit testing aims to eliminate construction errors before code is promoted to additional testing; this strategy is intended to

increase the quality of the resulting software as well as the efficiency of the overall development process.

Unit Testing for Login Feature: The results of unit testing on the Login Feature of the Project are shown below in Fig 6.1

Fig 6.1 Unit Testing for Login Feature

SL NO.	Test Case	Input	Expected Output	Actual Output	Remarks
1.	Adding valid email and user-type	Valid Email and username entered by user	Accept the Email and Username Fields	Email and Username Fields accepted	TEST PASSED
2.	Adding Email and User-type (Not present in the USERS table)	Adding Email and Username not present in the Database	Detect the invalid input from the user	An alert message “Invalid Credentials” is displayed	TEST PASSED
3.	Adding Password corresponding to email and user-type in the USERS table	Correct Password entered by the user	Login must be successful	An alert message “Login Successful” is displayed	TEST PASSED
4.	Adding Password not matching with the value present in the USERS table (For email and user-type)	User enters incorrect password for their account	“Incorrect password” must be displayed	An alert message “Login Unsuccessful” is displayed	TEST PASSED

Unit Testing for Volunteer Enrollment Feature: The volunteer enrollment unit test case evaluates the functionality of the enrollment feature within the National Service Scheme (NSS) project. This test ensures that volunteers can successfully submit their details through the enrollment form. It examines various scenarios, including the submission of valid volunteer information, handling incomplete form submissions, and verifying the redirection to a new tab upon successful enrollment. By rigorously testing these aspects, the unit test ensures the robustness and reliability of the volunteer enrollment process within the NSS system. It is shown in Fig 6.2

Fig 6.2 Unit Testing for Volunteer Enrollment Feature

SL NO.	Test Case	Input	Expected Output	Actual Output	Remarks
1.	Successful Enrollment submission	Valid volunteer details entered	Volunteer enrolled successfully	An alert message "Volunteer enrolled successfully" is displayed	TEST PASSED
2.	Enrollment form validation	Incomplete form submission	Please fill out all required fields	An alert message "Please fill out all required fields" is displayed	TEST PASSED
3.	New tab opened after successful enrollment	Valid volunteer details submitted	New tab opened with activity details	An alert message "New tab opened with activity details" is displayed	TEST PASSED

Unit Testing on Attendance feature of NSS Management System: It involves thoroughly examining the functionality responsible for recording volunteer attendance. This verification process ensures that the system accurately captures attendance information, handles user inputs correctly, avoids redundant data entries, and provides clear error notifications when necessary. Each test case assesses various aspects of the feature, including successful data submission, validation of input forms, error management, and ensuring that new tabs open correctly after completing actions. By conducting comprehensive unit tests, the reliability and effectiveness of the Attendance Tracking feature are validated, enhancing the overall dependability and usability of the NSS Management System. It is shown in Fig 6.3.

Fig 6.3 Unit Testing for Attendance Feature

SL NO.	Test Case	Input	Expected Output	Actual Output	Remarks
1.	Successful attendance tracking	Valid attendance details submitted	Attendance recorded successfully	An alert message "Attendance recorded successfully" is displayed	TEST PASSED
2.	Form validation	Incomplete attendance details	Please fill out all required fields	An alert message "Please fill out all required fields" is displayed	TEST PASSED
3.	New tab opened after tracking	Valid attendance details submitted	New tab opened with event details	An alert message "New tab opened with event details" is displayed	TEST PASSED
4.	Error Handling	Server error during submission	Error recording attendance	An alert message "Error recording attendance" is displayed	TEST PASSED
5.	Duplicate attendance prevention	Same attendance details submitted	Attendance already recorded	An alert message "Attendance already recorded" is displayed	TEST PASSED

Unit Testing for adding Activity Details by Administrator Feature: Unit testing was conducted to ensure the robustness and accuracy of the "Adding Activity Details" feature in the NSS Management System project. Various test cases were designed to validate different aspects of the functionality, including successful submission of activity details, handling of incomplete form submissions, error handling for server connectivity issues and invalid data formats, as well as the display of newly added activity details in the table. Each test case compared the expected output, such as specific alert messages or table updates, with the actual output observed during testing. Through thorough unit testing, the feature was verified to perform as intended, enhancing the reliability and functionality of the system. It is shown in Fig 6.4.

Fig 6.4 Unit Testing for Adding Activity Details by Administrator

SL NO.	Test Case	Input	Expected Output	Actual Output	Remarks
1.	Successful submission of activity details	Valid activity details entered	User should see an alert message “Activity details added successfully”	An alert message “Activity details added successfully” is displayed	TEST PASSED
2.	Incomplete form submission	Incomplete fields submitted	User should see an error message “Please fill out all required fields”	An error message “Please fill out all required fields” is displayed	TEST PASSED
3.	Error handling: Server unreachable	Valid activity details entered	User should see an error message “Failed to connect to server”	An error message “Failed to connect to server” is displayed	TEST PASSED
4.	Error handling: Invalid data format sent to server	Invalid data format submitted	User should see an error message “Invalid data format”	An alert message “Activity details added successfully”	TEST PASSED
5.	Display of activity data in the table upon successful submission	N/A	New activity details should be displayed in the table	New activity details should be displayed in the table	TEST PASSED

Integration Testing

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be located more quickly and fixed.

Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

Description: Integration testing ensures that the login functionality of the NSS Management System works flawlessly for both volunteer and administrator user types, handling valid and invalid credentials, managing user sessions, and enforcing access restrictions accurately. It is shown in Fig 6.2.

Fig 6.2 Integration Testing

SL NO.	Test Case Objective	Test Case Description	Expected Result
1	Check the link between Login Page and Enrollment Page	User should get directed to Enrollment Page so that he/she can enter his/her details	Redirection to Enrollment Page
2	Check the link between Login Page and Administrator Portal	Admin portal opens after logging in (so that the admin can enter activity details and approve volunteers for certification)	Redirection to Administrator Portal
3	Checking the link between Enrollment Page and Activity Selection	Volunteer must be able to select activity he/she is willing to participate in	Redirection to Display activities page
4	Checking the link between Activity Selection page and Attendance Tracking	Volunteer must be able to log his/her daily attendance	Redirection to Attendance tracking page so that volunteer can check in check out

System Testing

System testing tests a completely integrated system to verify that the system meets its requirements. For example, a system test might involve testing a logon interface, then creating and editing an entry, plus sending or printing results, followed by then logoff.

Description: System testing validates the functionality of role-based access control, user authentication with password encryption, and accurate redirection to all pages based on user roles, ensuring secure access and seamless navigation within the NSS Management System. It is shown in Fig 6.3.

Fig 6.3 System Testing

Test Case Type	Description	Test Step	Expected Result	Status
Functionality	Role based Access control, Volunteer Enrollment, Volunteer Attendance	Incorporating login, enroll, attendance tracking and admin portal web pages	Redirection to webpages with enrollment, attendance and activity management.	PASS
Security	User Authentication using Password	Entry of Password by the user during login	User able to Login only by entering valid credentials	PASS
Usability	Redirection to all pages appropriately	Have users click on various buttons	Clicking on buttons will redirect users to appropriate web pages	PASS

CHAPTER 7

RESULTS AND DISCUSSIONS

Home page of NSS Management System as shown in Fig 7.1

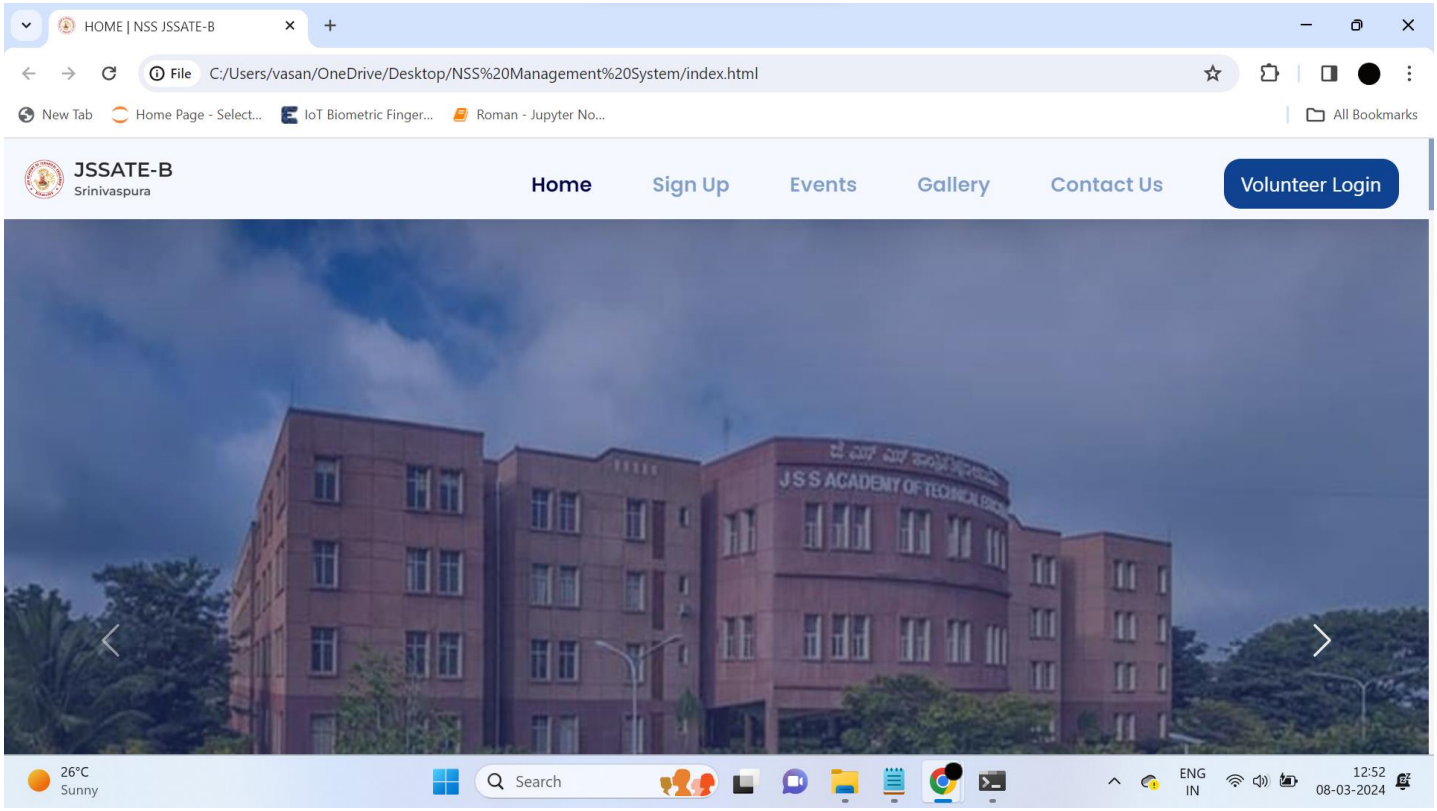


Fig 7.1 Depicts the Home Page

The Gallery Section of the NSS Management System is shown in Fig 7.2

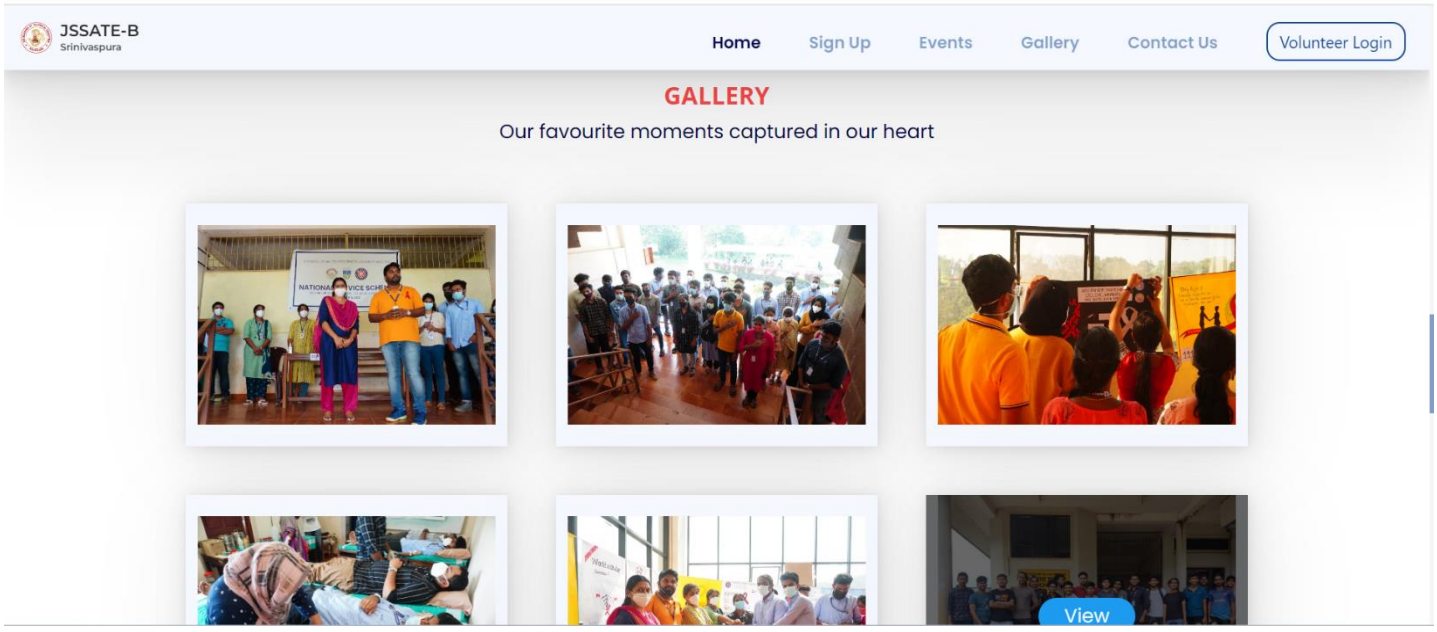


Fig 7.2 Gallery Section

Latest Programs and Activities carried out is shown in Fig 7.3

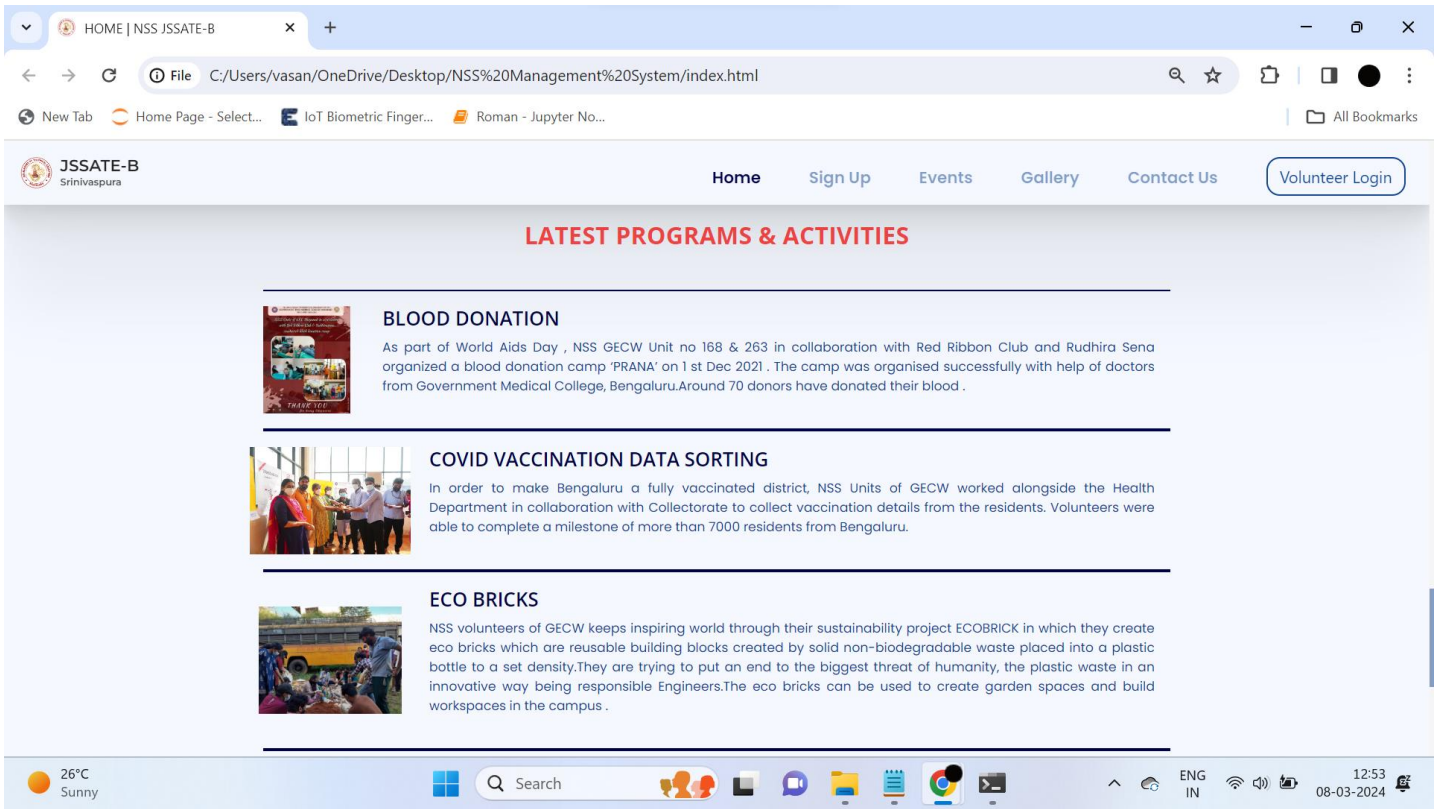


Fig 7.3 Latest Programs and Activities

Footer Section of the Home Page is shown in Fig 7.4

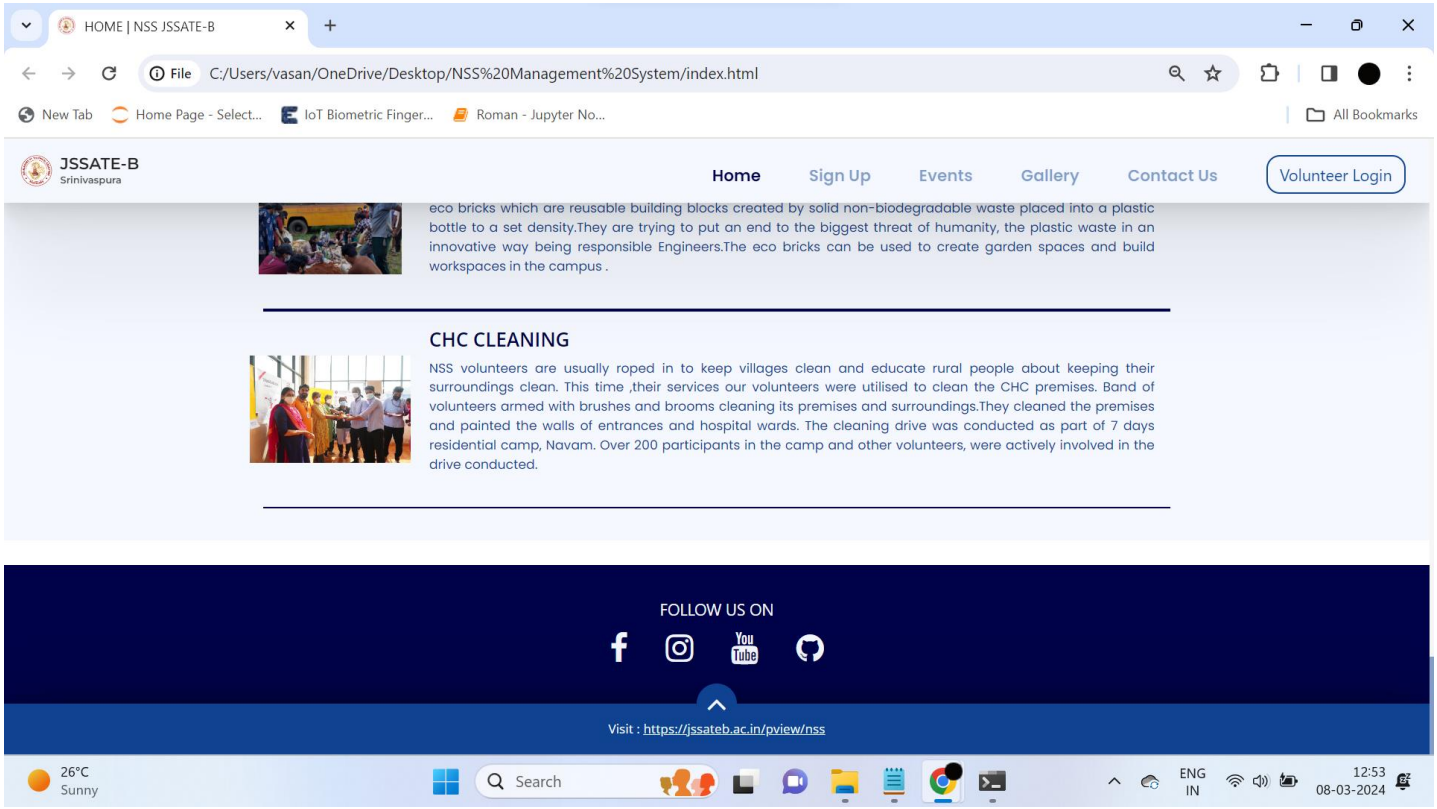


Fig 7.4 Footer Section

Signing Up based on User-Type is shown in Fig 7.5

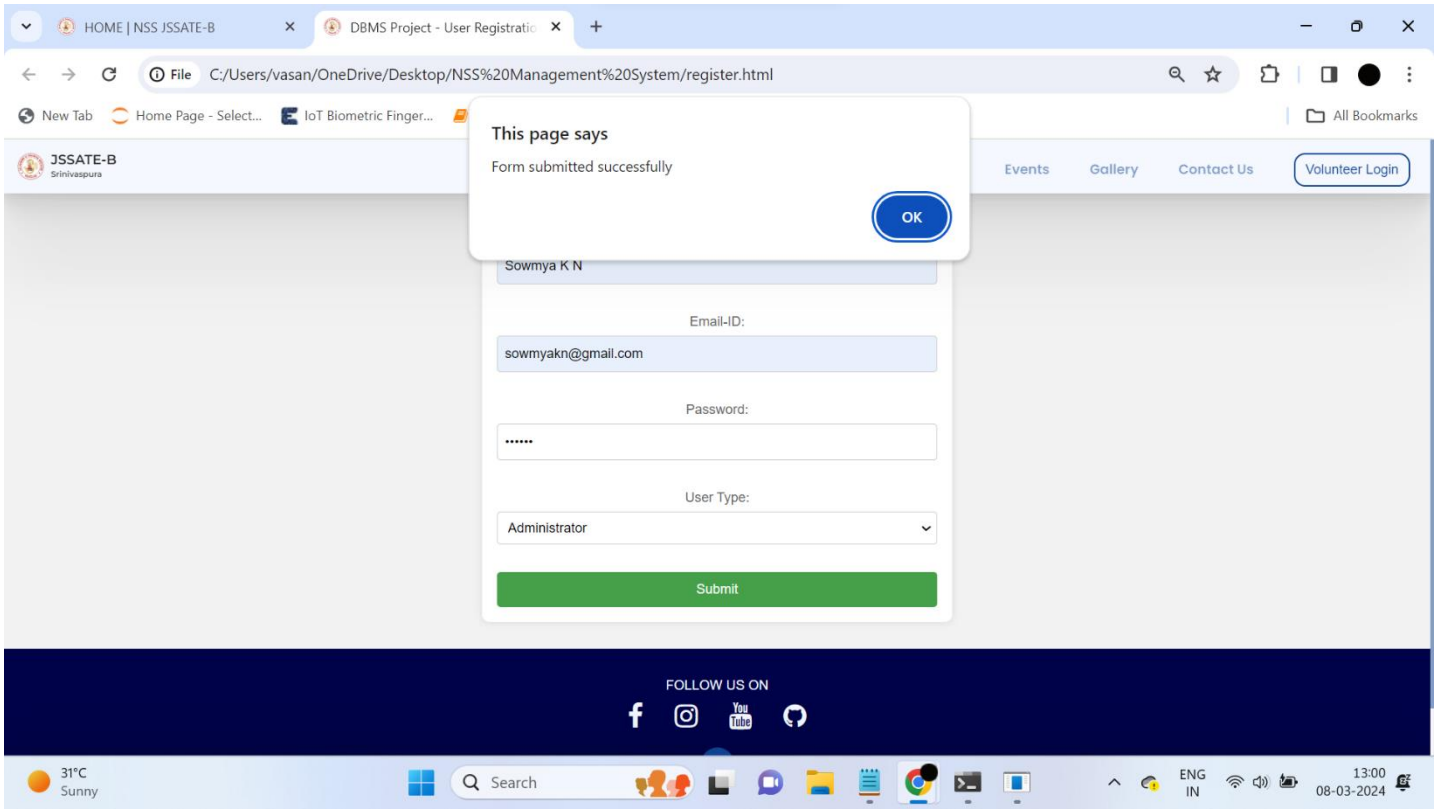


Fig 7.5 Sign Up Page

Data Parsed into JSON and sent to Database is shown in Fig 7.6

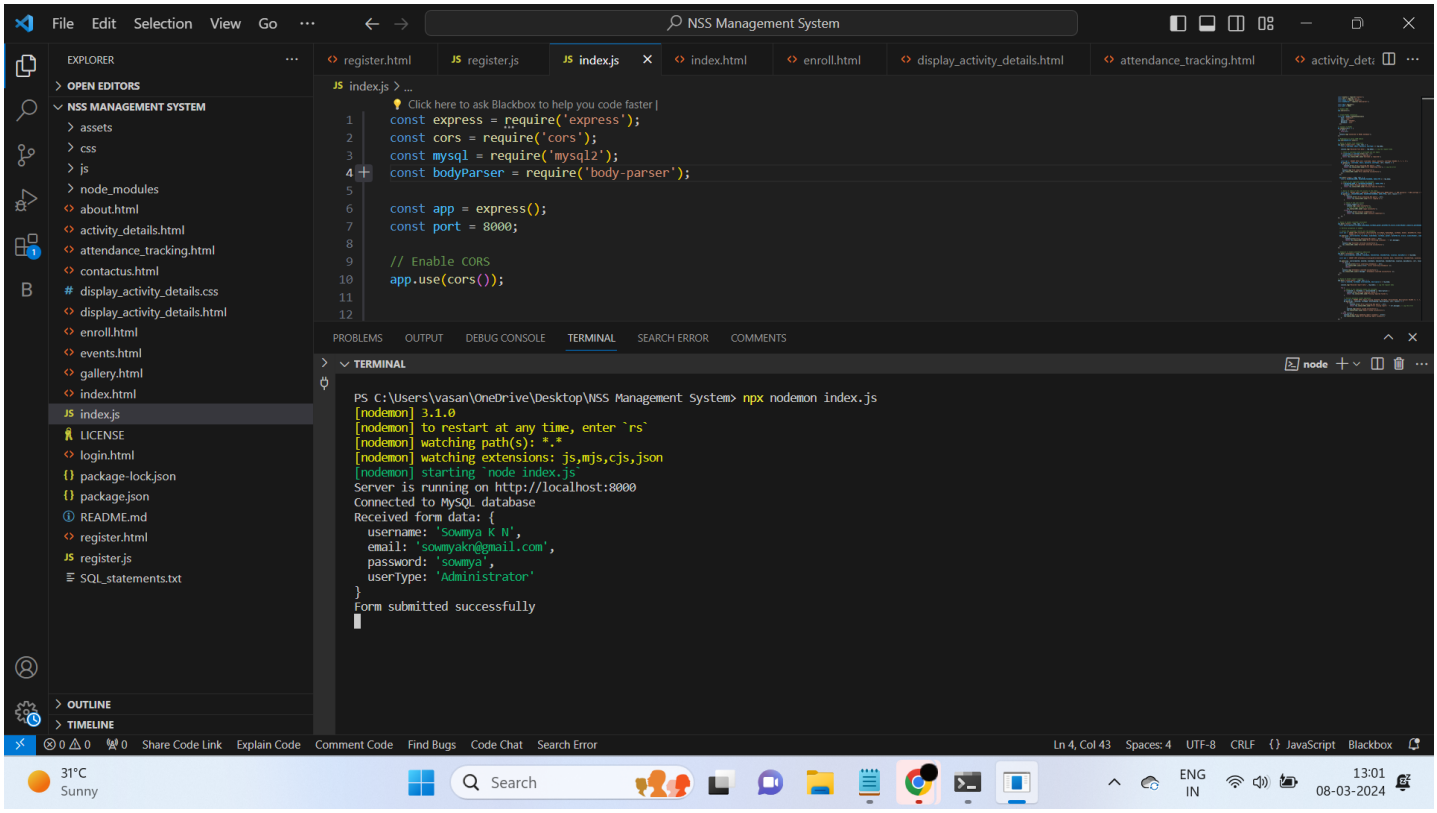


Fig 7.6 Data Parsed into JSON

Data being added to the respective Table is shown in Fig 7.5

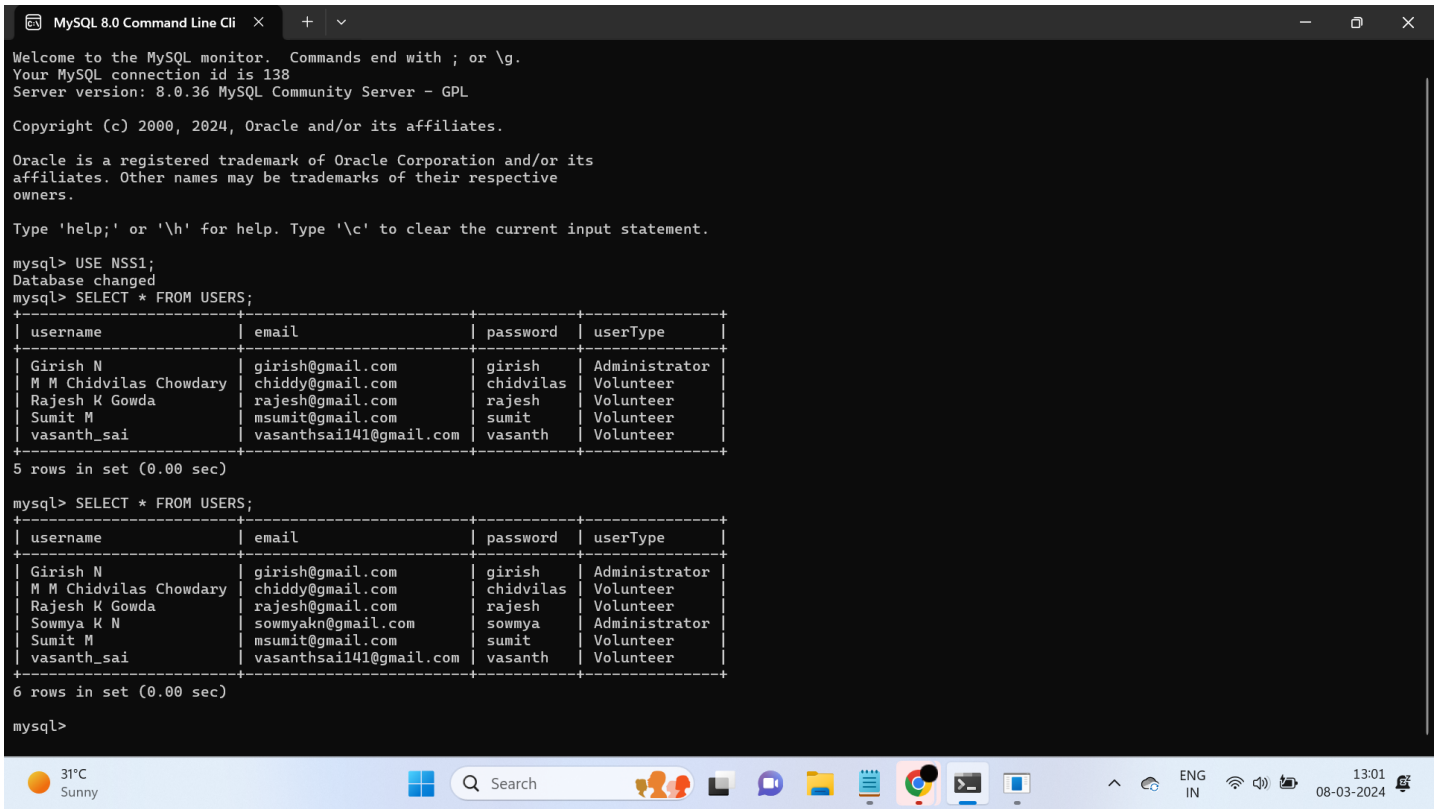


Fig 7.7 USERS Table

Logging in by User Authentication is shown in Fig 7.8

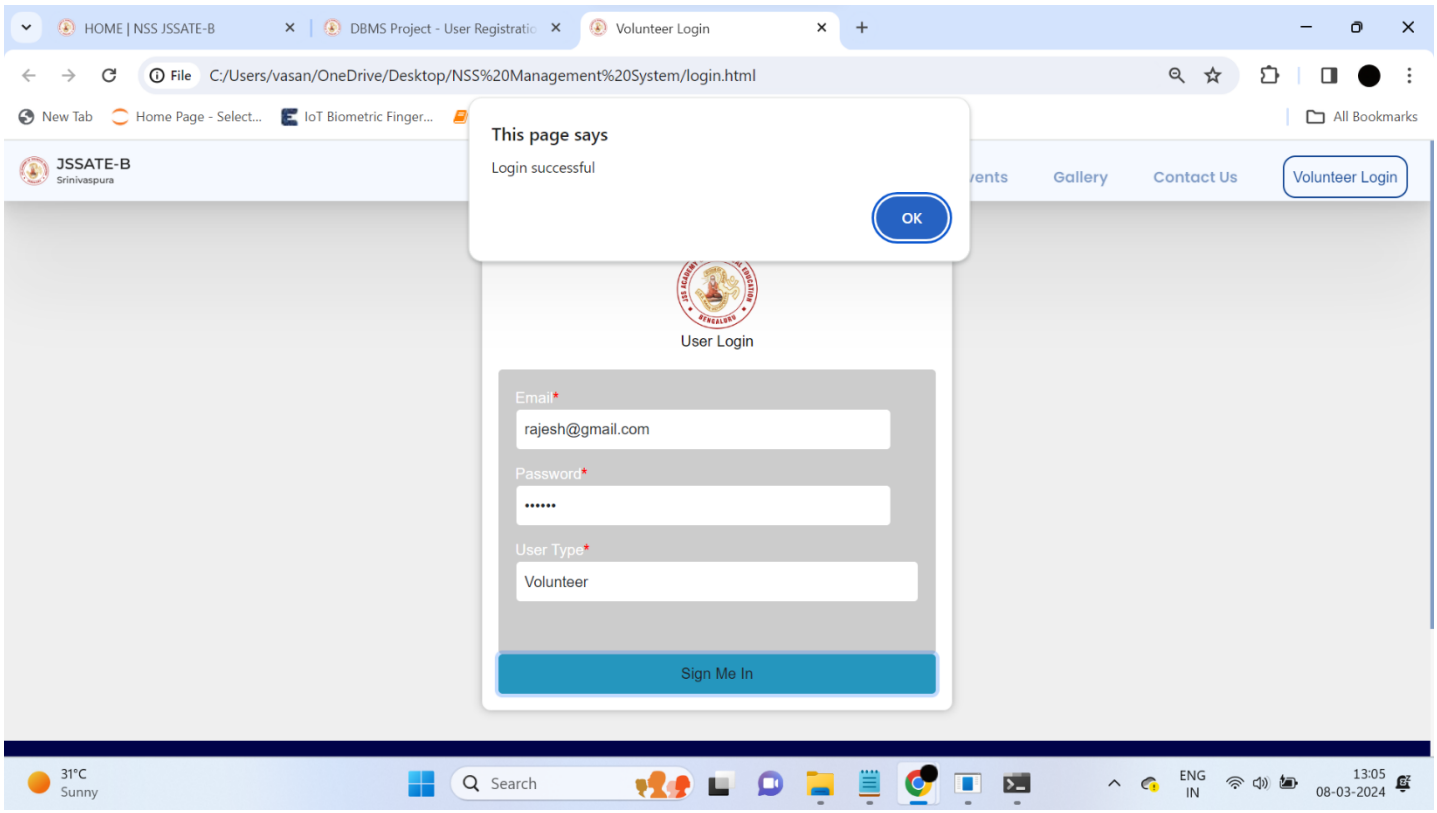


Fig 7.8 Login Page

Volunteer Enrollment Page is shown in Fig 7.9

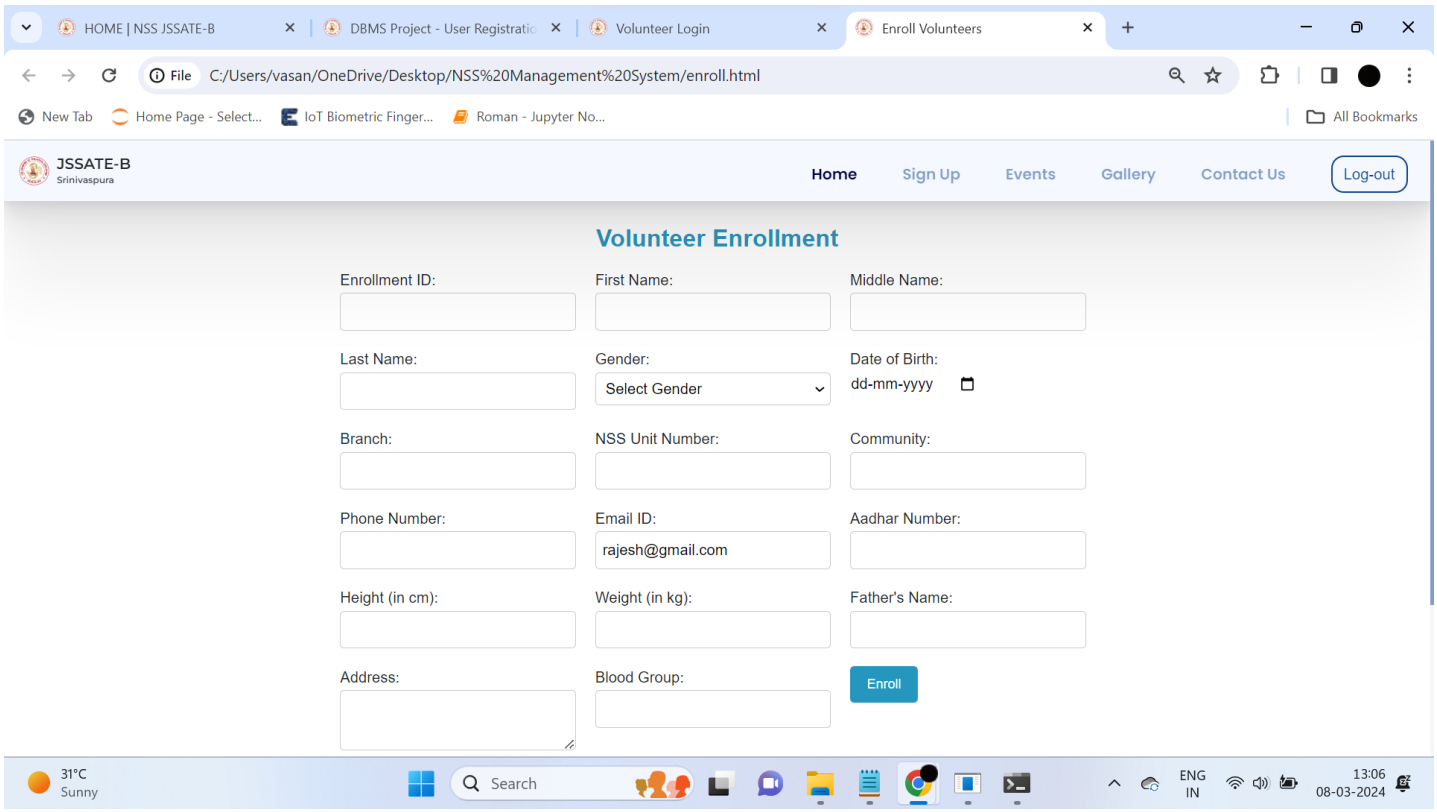


Fig 7.9 Volunteer Enrollment Page

Successful Enrollment of Volunteer is shown in Fig 7.10

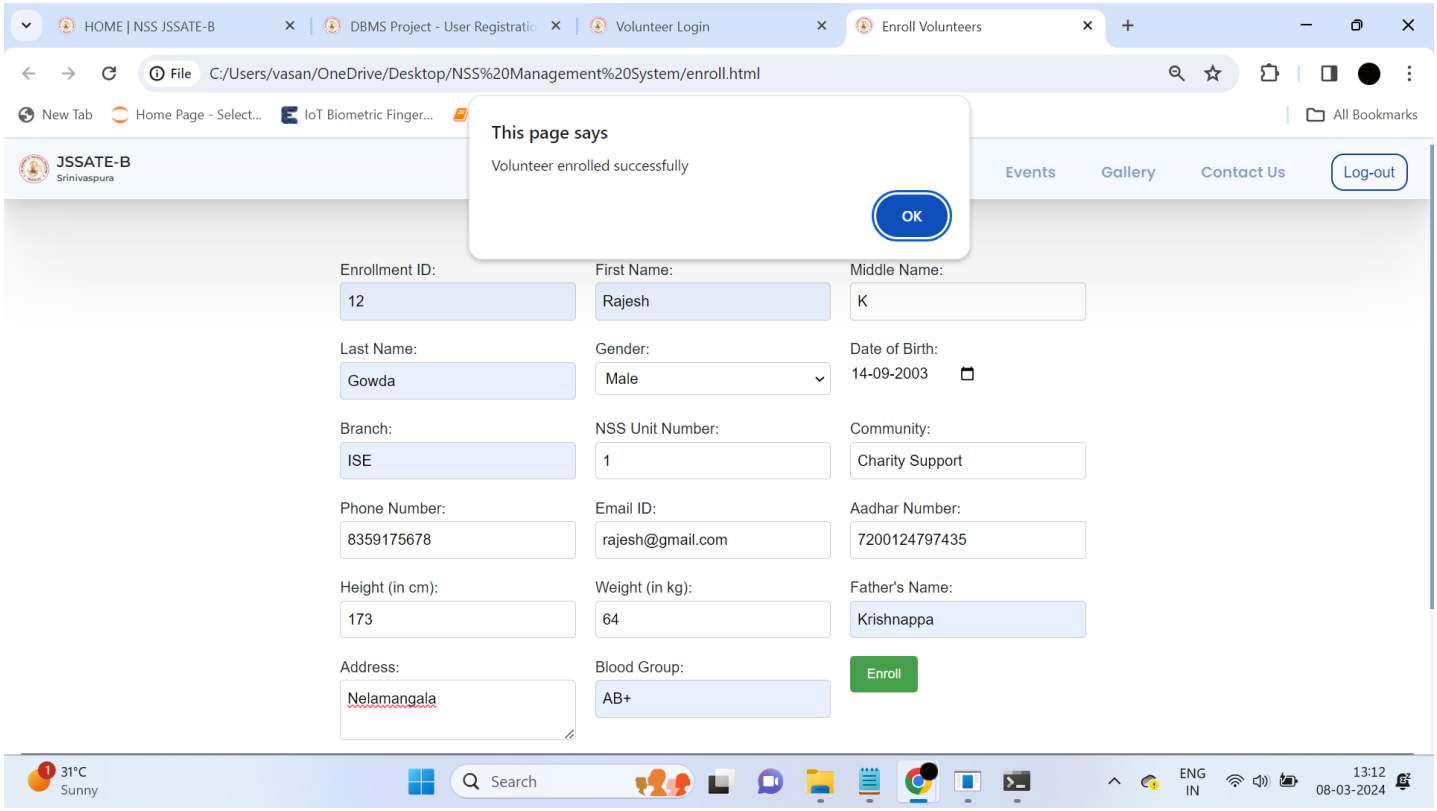


Fig 7.10 Volunteer Enrollment Successful

Volunteer's Data being added to the "volunteers" Table is shown in Fig 7.11

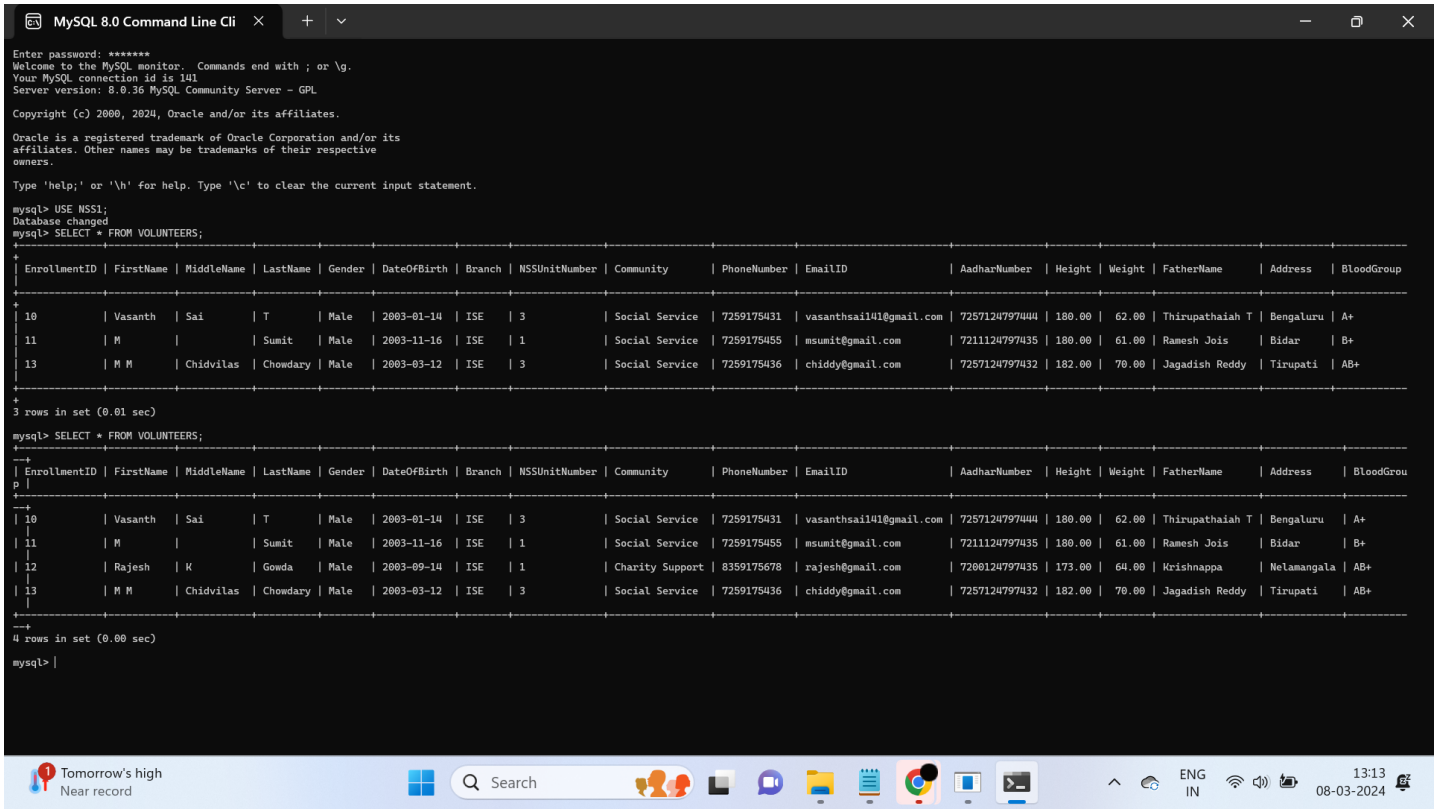


Fig 7.11 VOLUNTEERS Table

Volunteer Participating in an Activity is shown in Fig 7.12

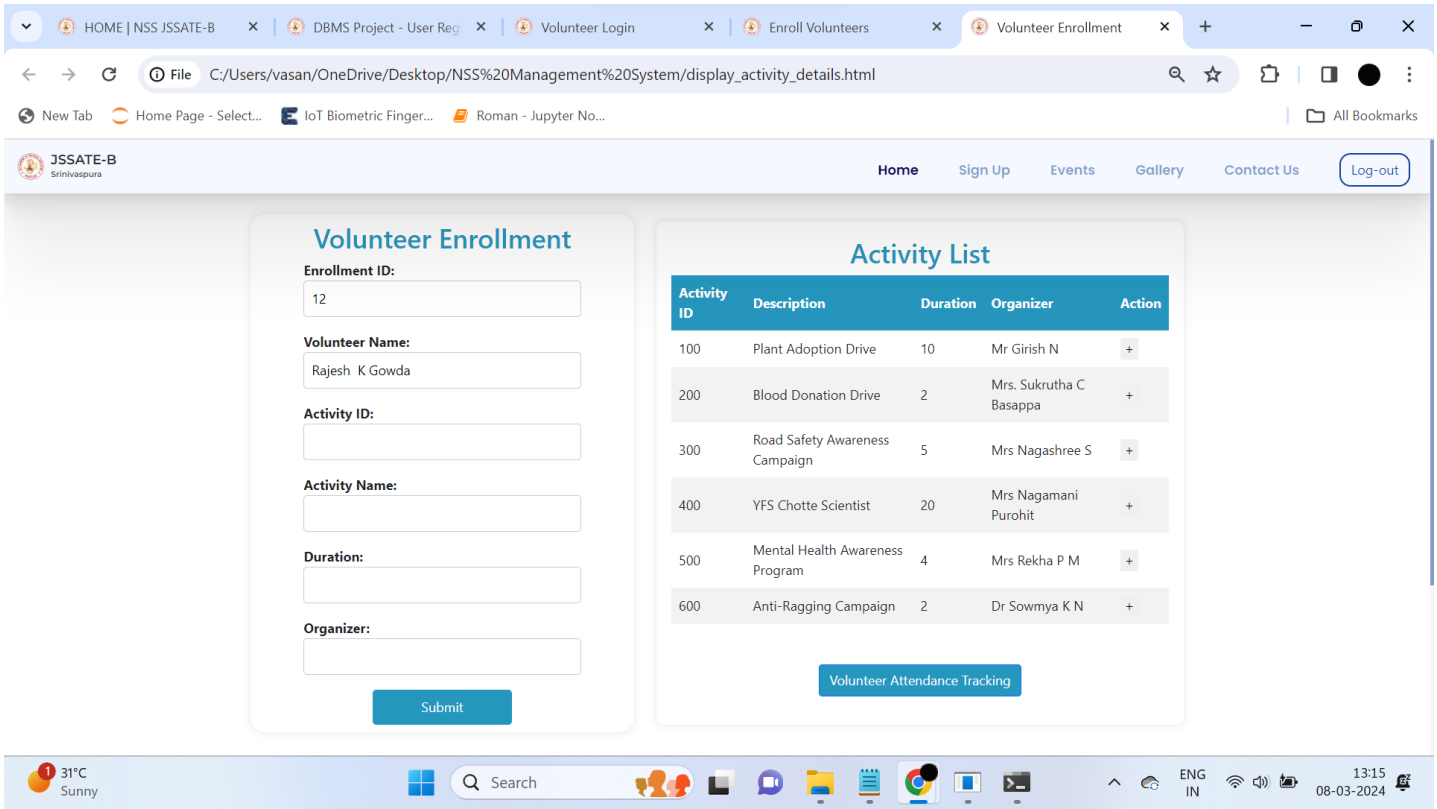


Fig 7.12 Display Activity Details Page

Successful Registration for an Activity by the Volunteer is shown in Fig 7.13

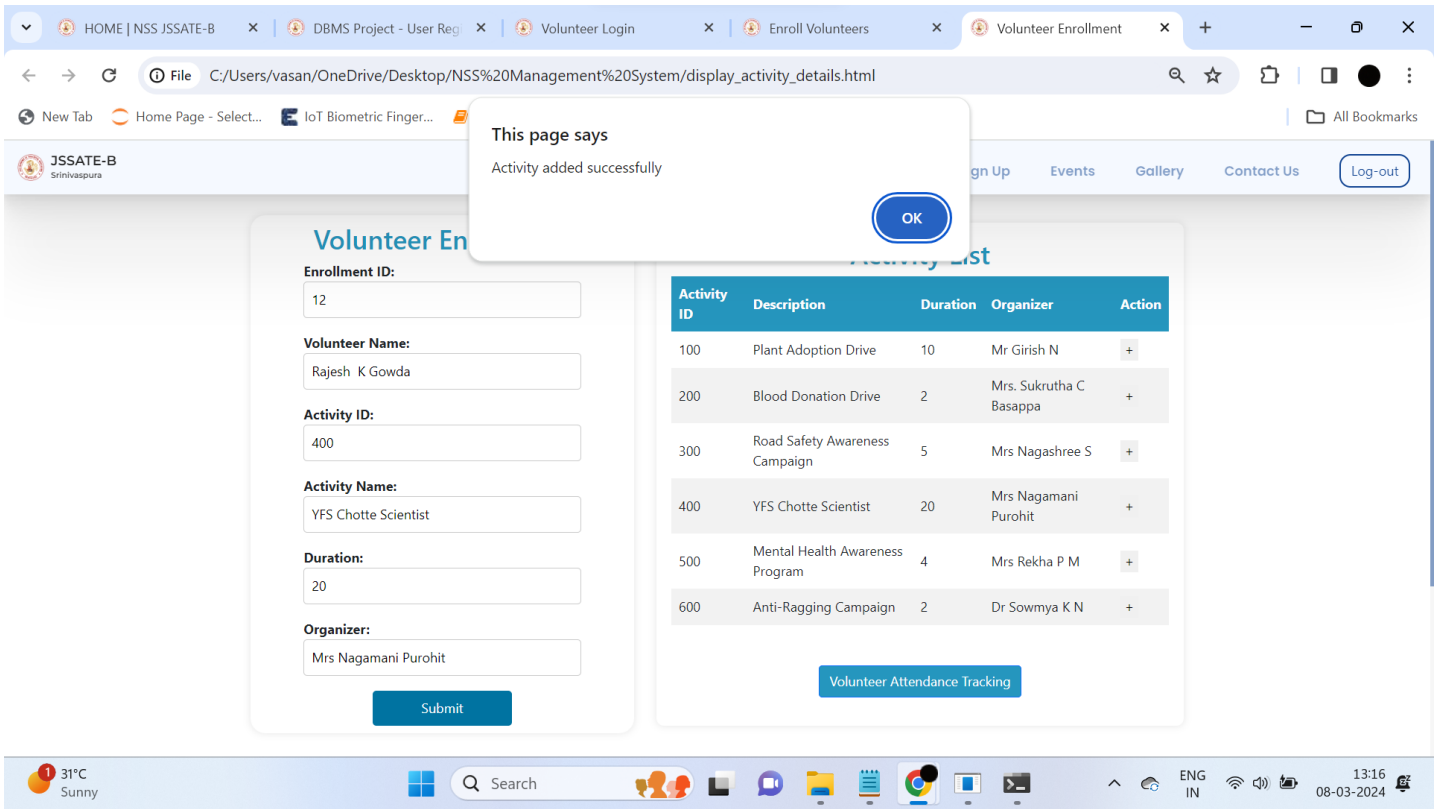


Fig 7.13 Activity Enrollment

Activity Details being added to the "Activity" table is shown in Fig 7.14

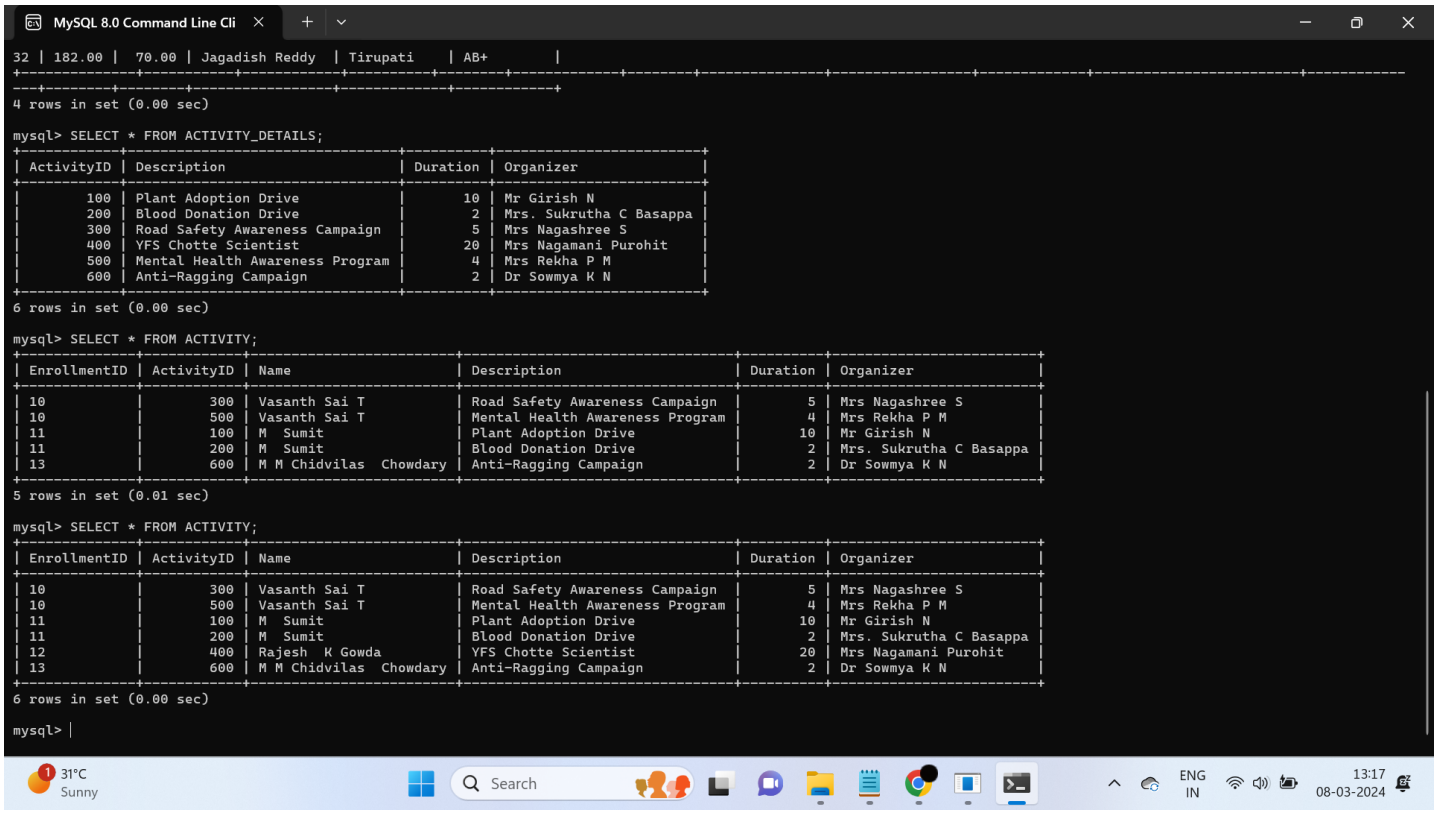


Fig 7.14 ACTIVITY Table

Daily Attendance Tracking of a Volunteer is shown in Fig 7.15

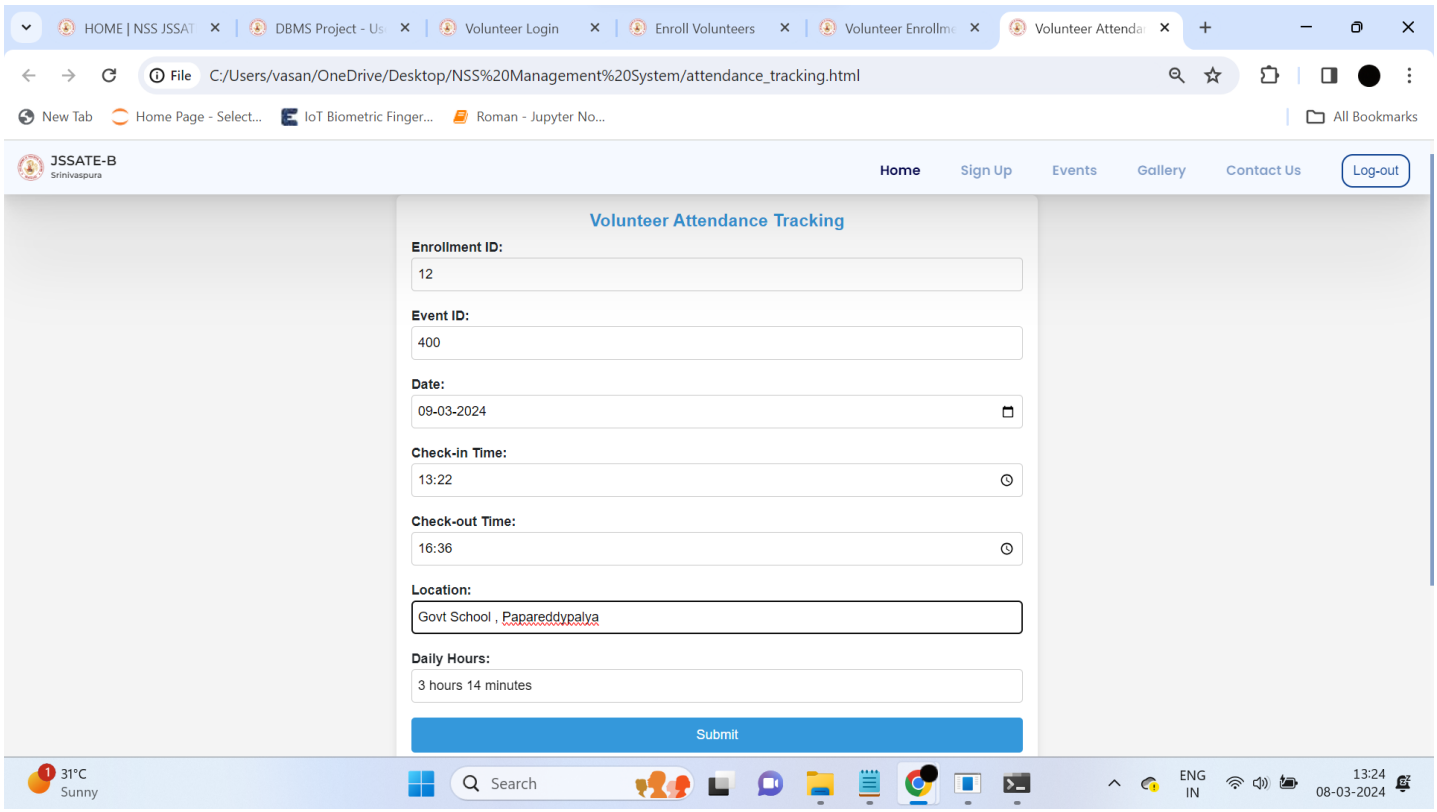


Fig 7.15 Attendance Tracking Page

Attendance Details being added to the "attendance_tracking" table and updation of "attendance" table through a Trigger is shown in Fig 7.16

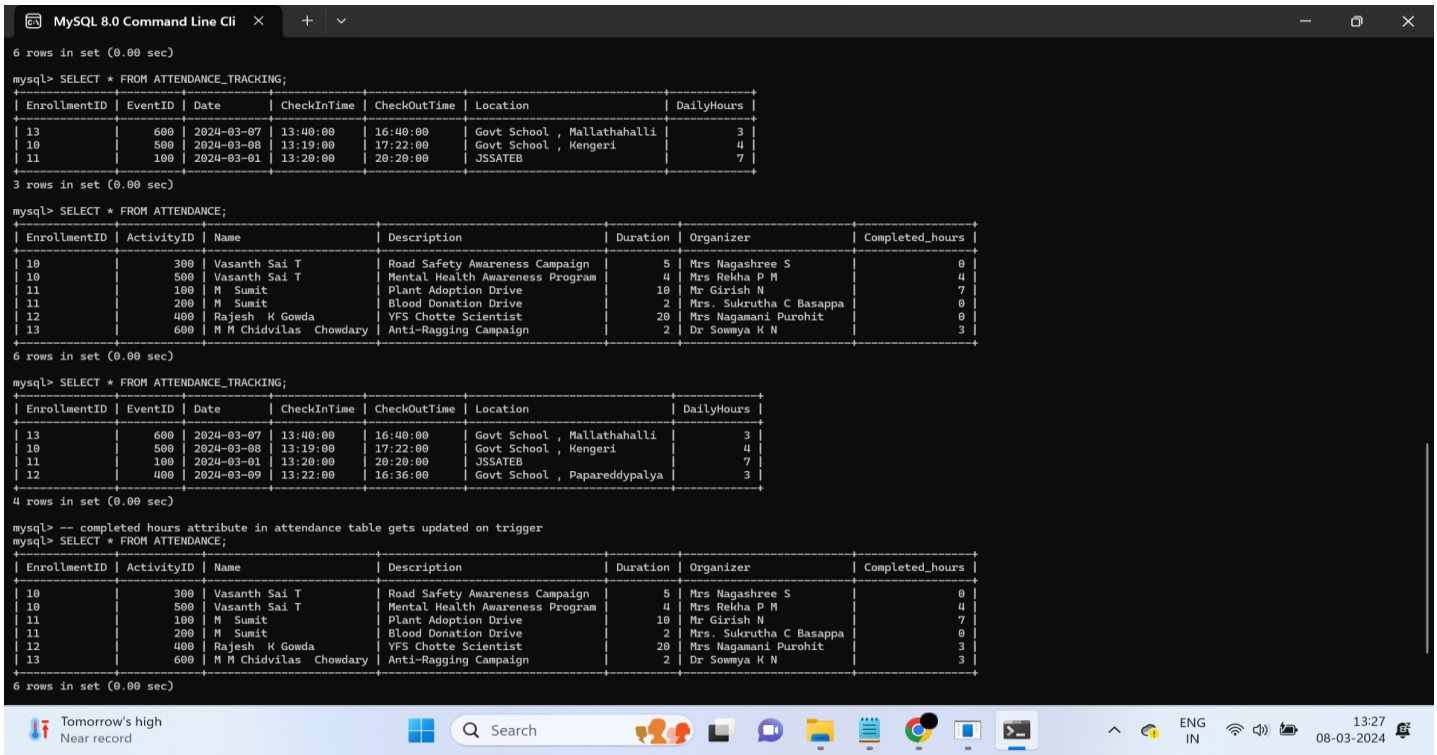


Fig 7.16 ATTENDANCE_TRACKING Table

Role Based Access Control for an Administrator is shown in Fig 7.17

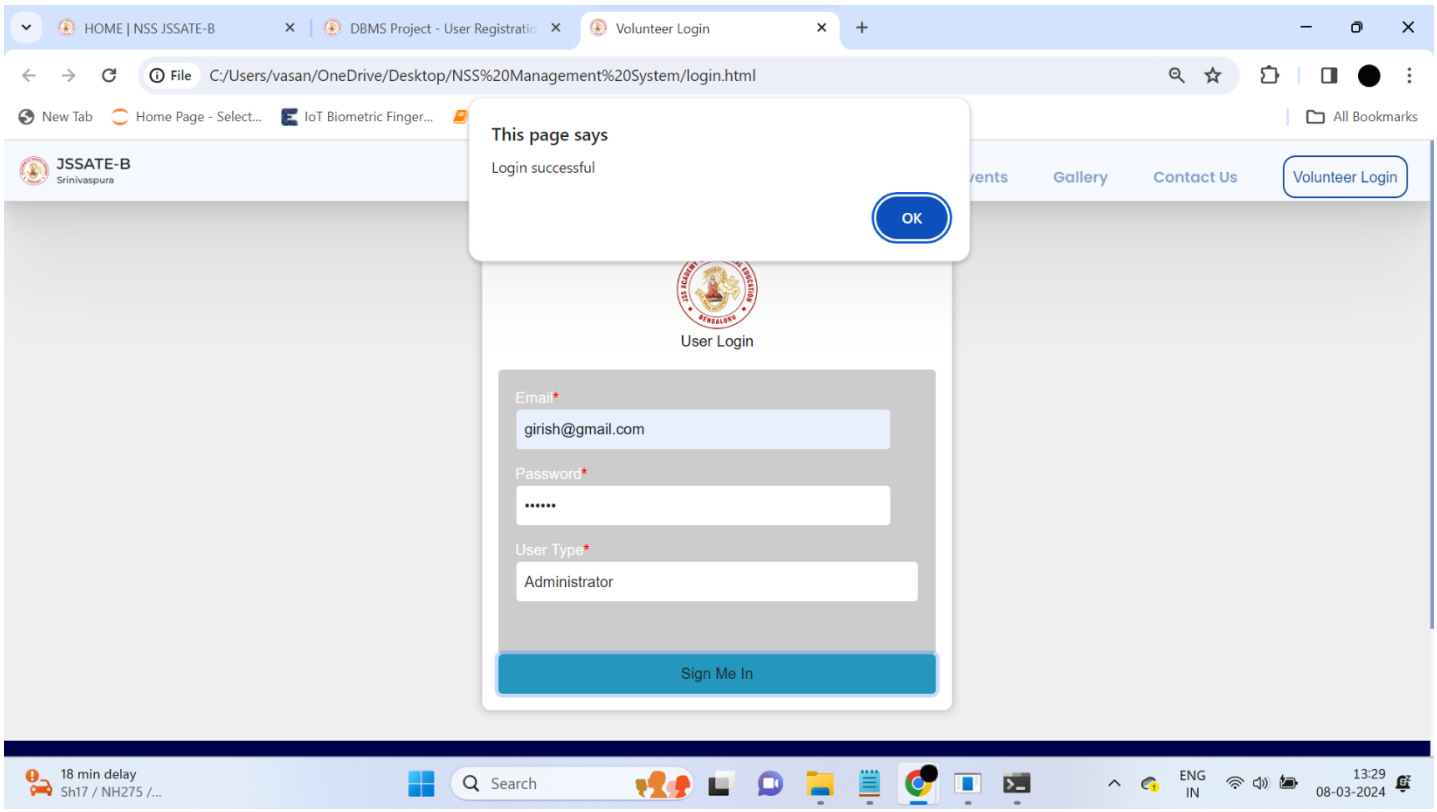


Fig 7.17 Administrator Logging In

Administrator adding the Activity Details is shown in Fig 7.18

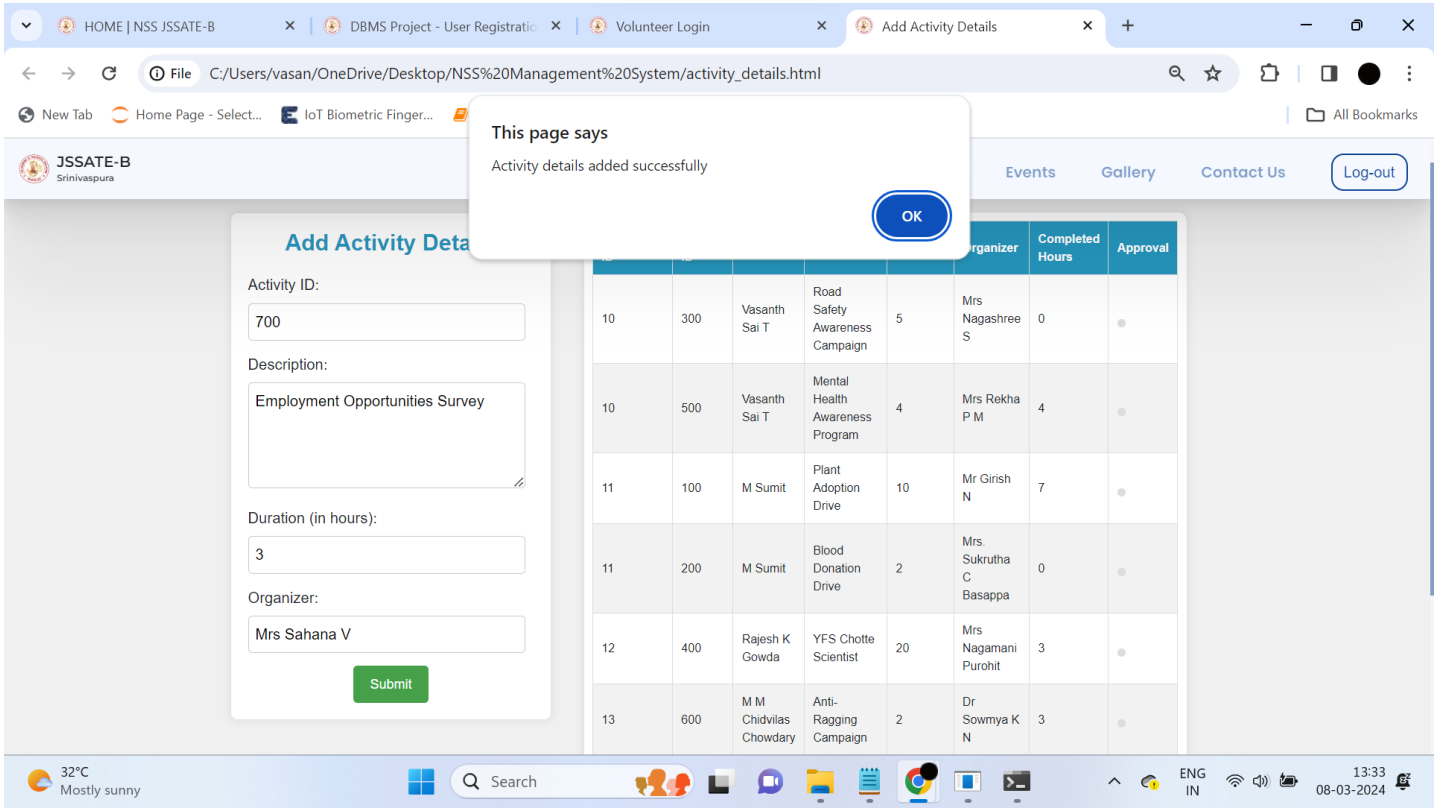


Fig 7.18 Activity Details Page

Administrator/Activity Organizer can add new activity details into the “activity_details” table

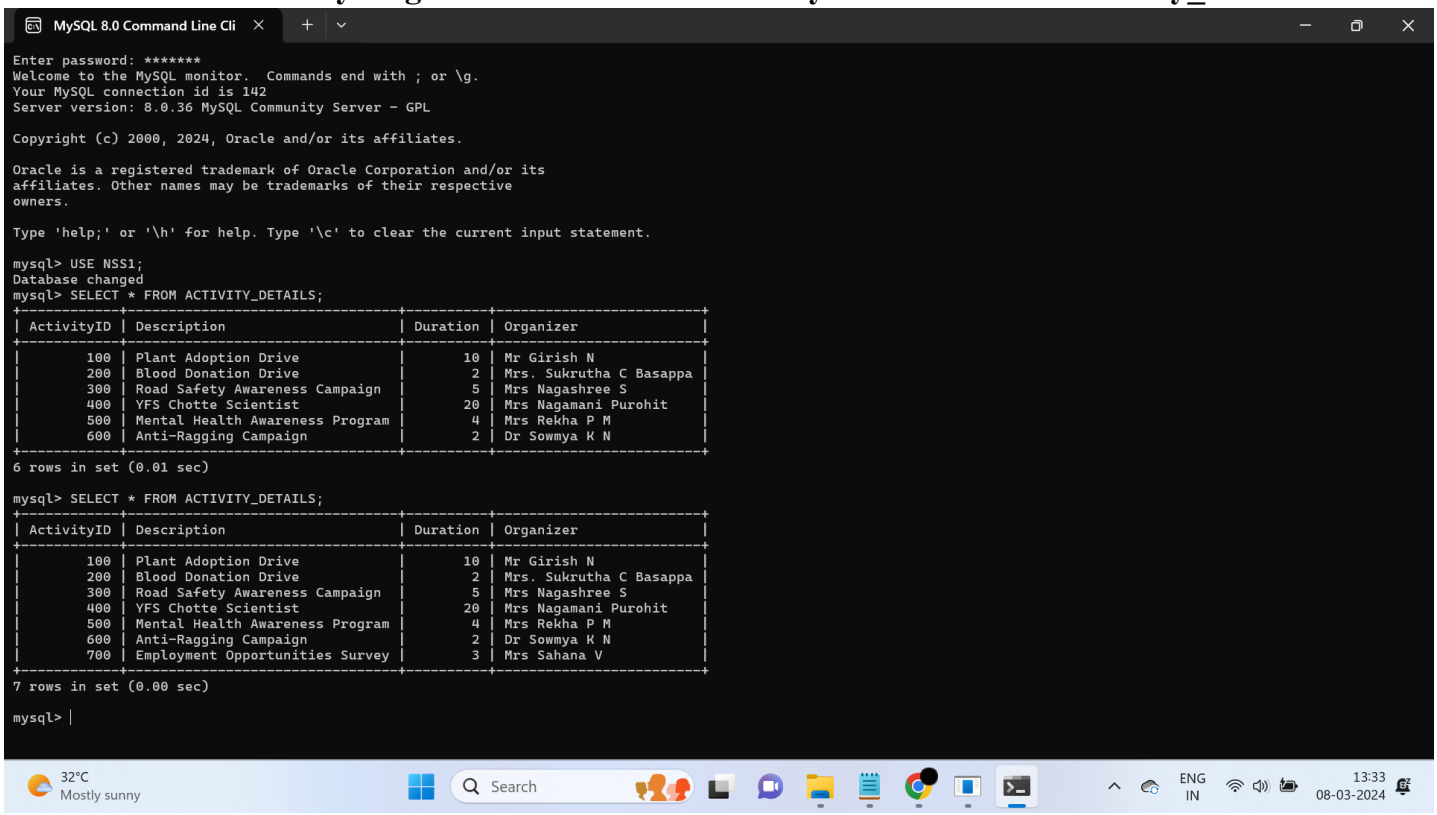


Fig 7.19 ACTIVITY_DETAILS Table

Administrator approving Certification for the Volunteers is shown in Fig 7.20

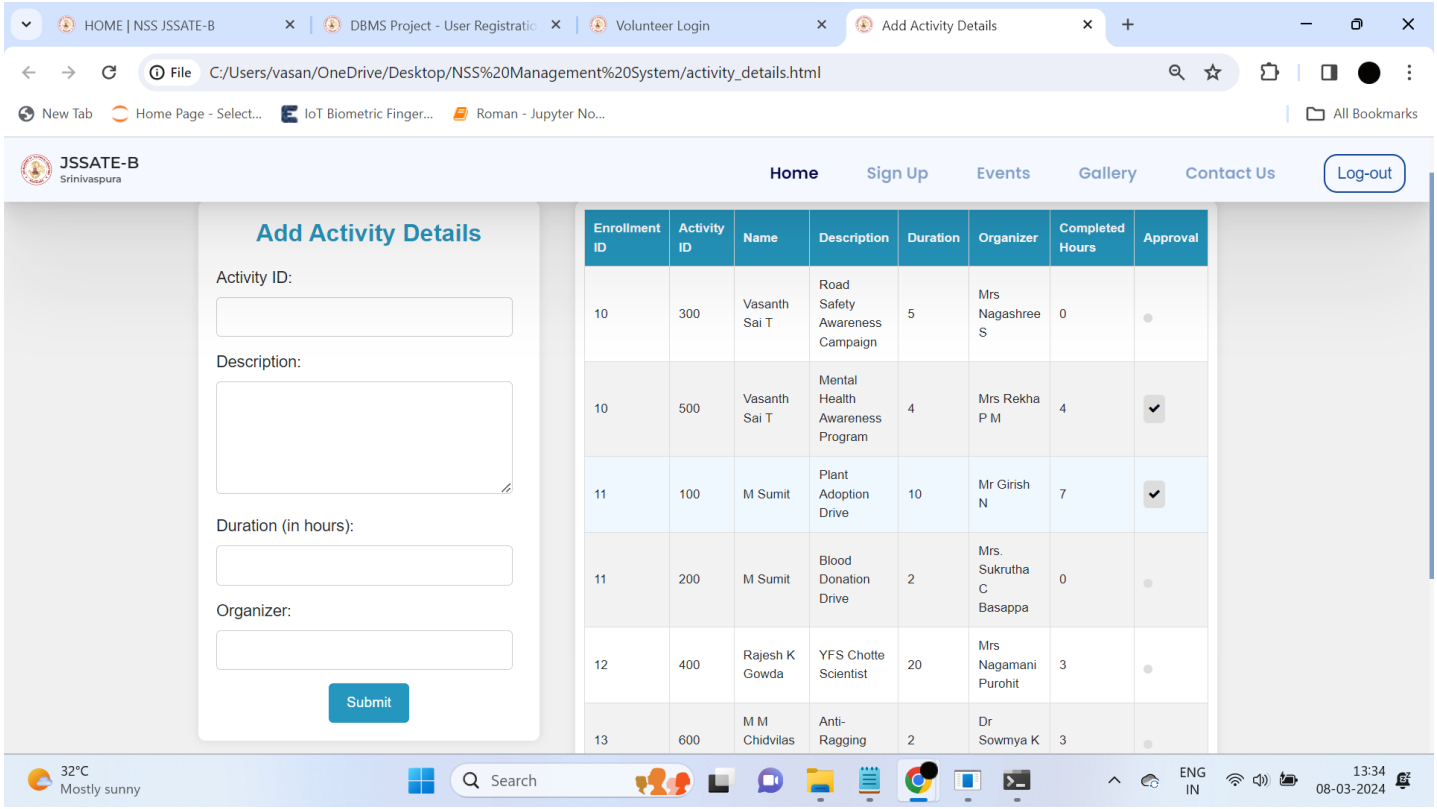


Fig 7.20 Approval of Certification by the Administrator

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENTS

CONCLUSION

The NSS Management System project was developed to address the needs of colleges and universities in effectively managing National Service Scheme (NSS) activities. Through this system, we aimed to streamline the process of organizing, tracking, and accessing data related to NSS volunteers, activities, and reports.

Throughout the development process, our goal was to create an intuitive and user-friendly interface that would facilitate easy access to NSS-related information for faculty, administrators, and volunteers. By implementing features such as user authentication, data validation, create, read and update operation, we aimed to ensure the system's functionality and reliability.

The NSS Management System provides a centralized platform for managing NSS activities across various departments within the institution. It enables efficient data organization, retrieval, and reporting, thereby enhancing collaboration and coordination among stakeholders involved in NSS initiatives.

While the current version of the NSS Management System serves as a prototype, it demonstrates the potential benefits of a comprehensive management application for NSS activities. Future iterations of the system could incorporate additional features and enhancements to meet the evolving needs of colleges and universities.

In conclusion, the NSS Management System project contributes to the efficiency and effectiveness of NSS operations by providing a robust platform for data management and coordination. By leveraging technology to streamline administrative processes, the system helps institutions optimize their NSS initiatives and contribute more effectively to community service and social development efforts.

FUTURE ENHANCEMENT

- **Integration with Inter-Collegiate NSS Activities:** Enhance the system to facilitate collaboration and information sharing among different colleges participating in NSS activities. This could include features for sharing resources, coordinating events, and exchanging best practices.

- **Online Hosting and Accessibility:** Host the NSS Management System on online servers to make it accessible from anywhere, enabling volunteers, faculty, and administrators to access the platform remotely.

- **Automated Backup Mechanism:** Implement an automated backup system for regularly backing up both the codebase and database of the NSS Management System. This ensures data integrity and disaster recovery preparedness.

- **Load Distribution:** Optimize the system architecture to distribute the workload across multiple servers, improving scalability and performance, especially during peak usage periods.

- **Event Notification System:** Integrate a feature for notifying users about upcoming NSS events, conferences, seminars, and other activities. This could include email alerts, push notifications, or in-app notifications to keep users informed and engaged.

CHAPTER 9

REFERENCES

BOOK REFERENCES

- "Fundamentals of Database Systems" by Ramez Elmasri and Shamkant B. Navathe, 7th Edition, 2015, Pearson.
- "Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design" by Michael J. Hernandez, 3rd Edition, 2013, Addison-Wesley Professional.
- "SQL and Relational Theory: How to Write Accurate SQL Code" by C.J. Date, 2nd Edition, 2011, O'Reilly Media.
- "MySQL Explained: Your Step-by-Step Guide" by Andrew Comeau, 2015, Addison-Wesley Professional.

WEB REFERENCES

- Node.js Documentation (https://nodejs.org/en/docs/)
- Express.js Documentation (https://expressjs.com/)
- MySQL Documentation (https://dev.mysql.com/doc/)
- JavaScript MDN Web Docs (https://developer.mozilla.org/en-US/docs/Web/JavaScript)
- Stack Overflow (https://stackoverflow.com/questions)
- npm (https://www.npmjs.com)