IOT & EMBEDDED SYSTEMS PROJECT

< ACCESS > CONTROL

SECURE ACCESS. EFFORTLESS CONTROL.

CONTRIBUTORS

A collaborative project developed with passion by **PRIYANSHU DAS** and **PRIYANSHU MISHRA** with expertise, vision, and creativity.

SUMMER INTERNSHIP 2023 PROJECT

SIT, BBSR & PHYTEC INDIA

Table of Contents

01 Project Overview

02 Problem Statement

03 Solutions Proposed

04 Tools and Technologies

05 Schematic Architecture

06 Flowchart and Flowcontrol

07 Executable Snippets

08 Code Segmentation

09 Scopes and Applications

10 Testing and Observation

11 Conclusion

At a Glance

An Overview of the Project.

The Access Control project ensures secure and reliable access management to controlled areas. It utilizes RFID card credentials for granting or denying access and offers features like time tracking and remote administration.

Hardware components include a rugged board, RFID reader module, and RTC module for accurate timekeeping. I2C and UART protocols enable seamless communication. A user-friendly web interface is created using HTML, CSS, and JavaScript, allowing administrators to efficiently manage access settings and user information. A locally hosted Python server handles data storage and access control logic.

Multithreading and the MRAA embedded library optimize performance. Linux is chosen for stability and security. Overall, the Access Control project provides a scalable solution for secure access management, combining hardware components and software technologies effectively.

Problem Statement

CHALLENGES

The existing access control systems for managing entry to controlled areas lack the desired level of reliability, security, and flexibility. Current systems often face challenges in effectively granting or denying access based on RFID card credentials. Additionally, there is a lack of comprehensive features such as time tracking and remote administration, which are crucial for efficient access management.

Moreover, the absence of a user-friendly interface hinders administrators from conveniently managing access control settings and user information. The lack of efficient communication protocols and hardware integration further hampers system performance, leading to delays and potential security vulnerabilities.

Furthermore, the absence of robust data storage and access control logic adds to the limitations of the current systems. The lack of support for handling multiple access requests simultaneously under high traffic conditions results in potential bottlenecks and operational inefficiencies.

Hence, the problem lies in the need for a reliable and secure access control system that combines advanced hardware components, seamless communication protocols, user-friendly interfaces, and robust backend functionalities. This system should offer features like RFID card-based access management, accurate time tracking, remote administration capabilities, and efficient handling of multiple access requests.

Proposed Solution

SOLUTION

Integrated Access Control System

01

Card-based Access Control

The system utilizes RFID card credentials to accurately grant or deny access, ensuring a secure and efficient entry process.

02

Comprehensive Features

The solution includes essential features such as time tracking and remote administration, allowing administrators to monitor access events and manage the system remotely for enhanced control and convenience.

03

User-Friendly Web Interface

This interface enables administrators to easily manage access control settings and user information, simplifying the overall system management process using HTML, CSS, and JavaScript.

04

Seamless Communication

The communication protocols ensures communication between hardware components, facilitating efficient retrieval of RFID card credentials, accurate timekeeping, real-time monitoring, and remote administration.

05

Robust Backend Functionality

A locally hosted Python server acts as the backend, offering robust data storage, user authentication, and access control logic. This ensures data integrity, system stability, and efficient processing of access requests.

Proposed Solution

SOLUTION

Integrated Access Control System

06

Multithreading and Hardware Integration

The system leverages multithreading and the MRAA embedded library to efficiently handle multiple access requests concurrently, ensuring smooth operation even during high traffic periods.

07

Linux Operating System

The choice of the Linux OS provides a stable and secure platform for hosting the access control system, enhancing system reliability and minimizing vulnerabilities.

Overall, the solution provides a comprehensive and scalable access control system that combines reliable hardware components, powerful software technologies, intuitive user interfaces, and efficient backend functionalities. It addresses the limitations of existing systems, ensuring reliable access management, enhanced security, and streamlined administration processes.

The proposed solution is an integrated Access Control system that addresses the limitations of existing systems. It offers a reliable, secure, and flexible approach to managing access to controlled areas.

Tools and Technologies

Group 1 : Hardware Components

RUGGED BOARD
The main hardware platform providing processing power and connectivity options.

EM18 RFID READER MODULE
Reads RFID card credentials for access control.

DS1307 RTC MODULE
Ensures accurate timekeeping for logging access events.

Group 2 : Communication Protocols

I2C (INTER-INTEGRATED CIRCUIT)
Facilitates seamless retrieval of RFID card credentials and accurate timekeeping.

UART (UNIVERSAL ASYNCHRONOUS RECEIVER-TRANSMITTER)
Enables real-time monitoring and remote administration of the system.

Tools and Technologies

Group 3 : Web Development Technologies

HTML (HYPERTEXT MARKUP LANGUAGE)
Used for structuring the web interface.

CSS (CASCADING STYLE SHEETS)
Applied for styling and visual presentation of the web interface.

JAVASCRIPT
Adds interactivity and user experience on the web interface.

Group 4 : Backend Technologies

PYTHON SERVER
A locally hosted server that serves as the backend for data storage, user authentication, and access control logic.

MULTITHREADING & PARALLEL PROCESSING IN PYTHON
Techniques used for efficient handling of multiple access requests simultaneously.

Tools and Technologies

Group 5 : Supporting Libraries and Tools

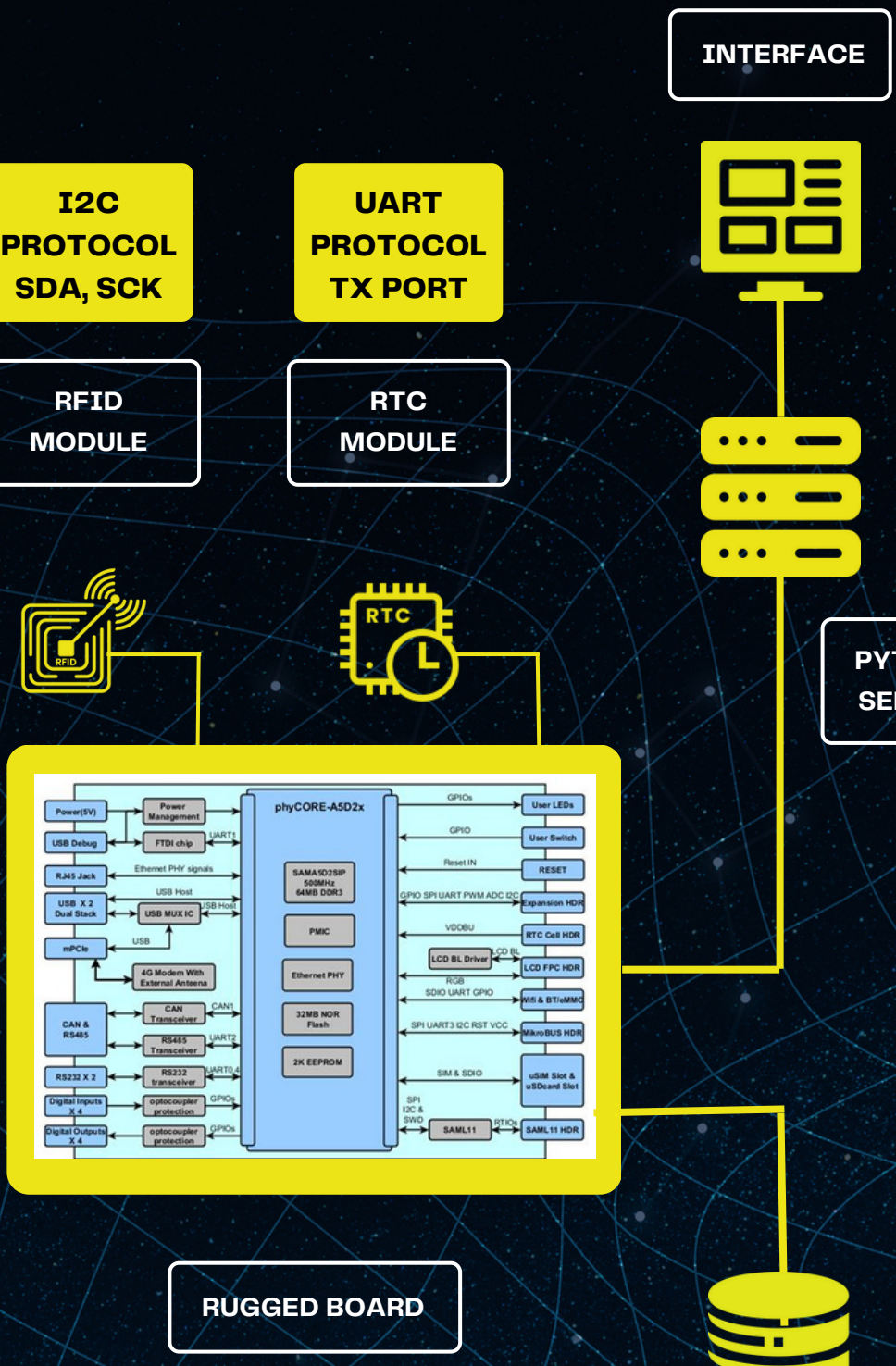
MRAA EMBEDDED LIBRARY

Provides a high-level API for interacting with hardware components connected to the rugged board.

LINUX OS

Chosen operating system for its stability, security, and flexibility.

Schematic Architecture



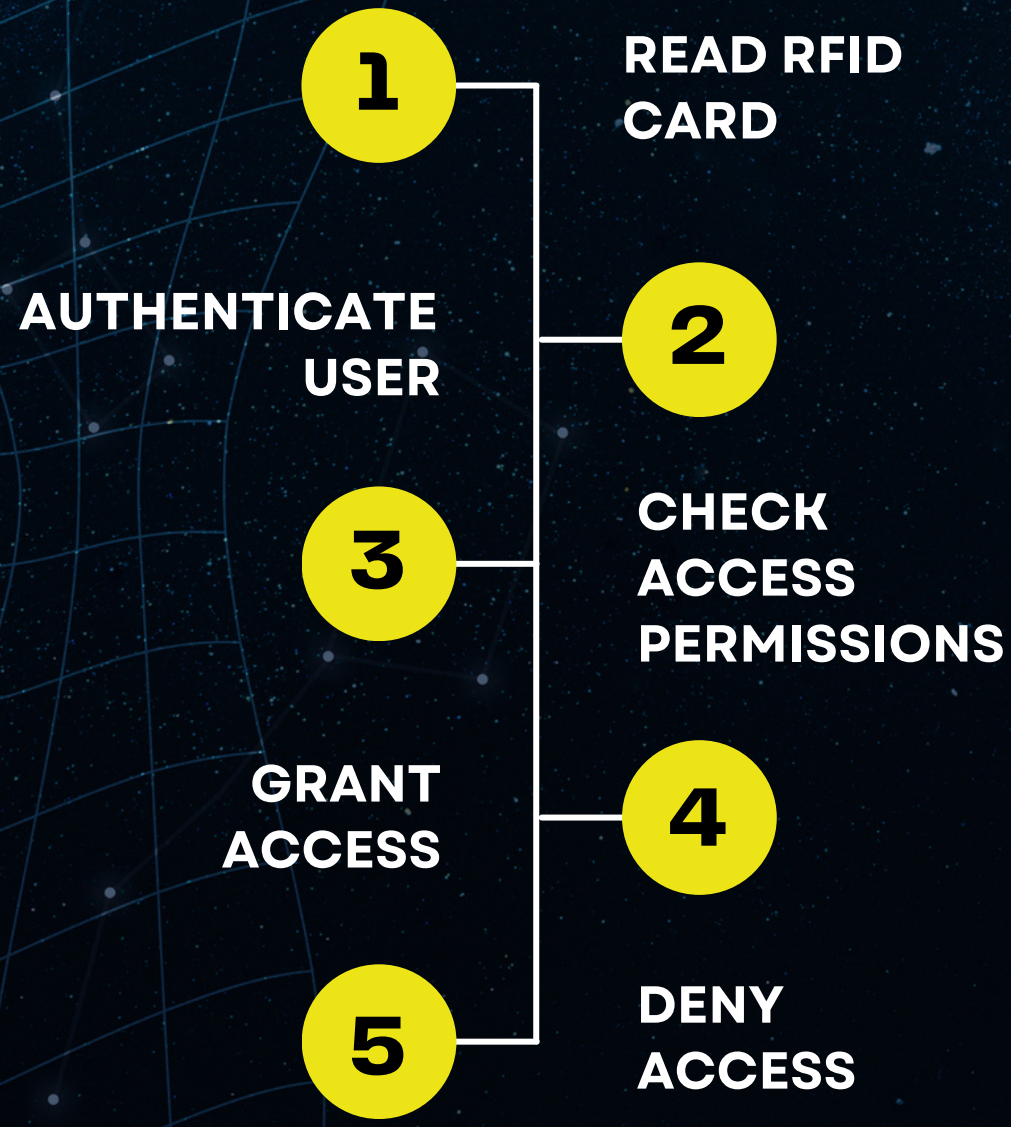
Flowchart (FSMD)

STATES

The flowchart outlines the sequence of actions and decision points involved in the access control process, including reading the RFID card credentials, authenticating the user, and granting or denying access based on the verification results.

Flow Control (UMLD)

STATES



The activity diagram visually illustrates the flow of control or sequence of activities involved in the access control process.The diagram would also depict decision points or conditions where the flow can branch based on the authentication results or access permissions.

Executable Snippets

01

I2C COMMUNICATION PROTOCOL

```python
def i2c_init(self,twi = 0,address=0x00):
    try:
        self.i2c = mraa.I2c(twi)
        if address:
            self.i2c.address(address)
        return self.i2c
    except Exception:
        raise Exception

def read_i2c_data(self,address):
    try:
        return self.i2c.readReg(address)
    except Exception:
        raise Exception

def write_i2c_data(self,address,data_byte):
    try:
        self.i2c.writeReg(address,data_byte)
    except Exception:
        raise Exception

```

Executable Snippets

02

UART COMMUNICATION PROTOCOL

```python
def uart_init(self,port,baud_rate=9600,data_bits=8,stop_bit=1,parity_bit=mraa.UART_PARITY_NONE):

        try:
            # serial port
            self.port = port
            # initialise UART
            self.uart = mraa.Uart(port)
            self.uart.setBaudRate(baud_rate)
            self.uart.setMode(data_bits, parity_bit, stop_bit)
            self.uart.setFlowcontrol(False, False)

            return self.uart
        except:
            print("Error opening port for UART communication")


    def data_available(self):
        try:
            return self.uart.dataAvailable()
        except:
            print("Error in UART")

    def read_uart_data(self,data_length=1):
        try:
            return self.uart.readStr(data_length)

        except:
            print("Error reading the data from UART")

    def write_uart_data(self,data):
        try:
            # send data through UART
            self.uart.write(bytearray(data, 'utf-8'))

        except:
            print("Error writing to UART")

```

Executable Snippets

03

EM18 RFID READER MODULE

```python
def run_rfid_process():

    try:
        mraa_uart = MRAA_Helper()
        rfid = RFID_Handler(mraa_uart)
        access_keys = rfid.read_access_keys("data.json")
        mraa_uart.uart_init("/dev/ttyS3")
        print(mraa_uart)
        data = []

        while (True):

            while mraa_uart.data_available():
                data.append(mraa_uart.read_uart_data())

            if len(data) > 0:

                rfid_uid = "".join(data)

                if rfid.get_mode() == RFID_Handler.ACCESS:

                    if rfid_uid in access_keys:
                        print("Access Granted")
                        rfid.run_access_procedure(rfid_uid)
                    else:
                        rfid.log_info(rfid_uid, "Access Denied")
                        print("Access Denied")

                elif rfid.get_mode() == RFID_Handler.READ:
                    print(rfid_uid)
                    if rfid_uid not in access_keys:
                        rfid.add_access_keys(rfid_uid)

                data = []

    except:
        raise Exception

```

Executable Snippets

04

DS1307 RTC MODULE

```python
def write_all(self, seconds=None, minutes=None, hours=None, day=None,
              date=None, month=None, year=None, save_as_24h=True):
        """Direct write un-none value.
        Range: seconds [0,59], minutes [0,59], hours [0,23],
               day [0,7], date [1-31], month [1-12], year [0-99].
        """

        # print(seconds,minutes,hours,day,date,month,year)
        if seconds is not None:
            if seconds < 0 or seconds > 59:
                raise ValueError('Seconds is out of range [0,59].')
            self.mraa_i2c.write_i2c_data(self._REG_SECONDS, self.int_to_bcd(seconds))

        if minutes is not None:
            if minutes < 0 or minutes > 59:
                raise ValueError('Minutes is out of range [0,59].')
            self.mraa_i2c.write_i2c_data(self._REG_MINUTES, self.int_to_bcd(minutes))

        if hours is not None:
            if hours < 0 or hours > 23:
                raise ValueError('Hours is out of range [0,23].')
            if save_as_24h:
                self.mraa_i2c.write_i2c_data(self._REG_HOURS, self.int_to_bcd(hours) & 0xbf)
            else:
                if hours == 0:
                    h = self.int_to_bcd(12) | 0x32
                elif hours <= 12:
                    h = self.int_to_bcd(hours)
                else:
                    h = self.int_to_bcd(hours - 12) | 0x32
                self.mraa_i2c.write_i2c_data(self._REG_HOURS, h)

        if year is not None:
            if year < 0 or year > 99:
                raise ValueError('Years is out of range [0,99].')
            self.mraa_i2c.write_i2c_data(self._REG_YEAR, self.int_to_bcd(year))

        if month is not None:
            if month < 1 or month > 12:
                raise ValueError('Month is out of range [1,12].')
            self.mraa_i2c.write_i2c_data(self._REG_MONTH, self.int_to_bcd(month))

        if date is not None:
            if date < 1 or date > 31:
                raise ValueError('Date is out of range [1,31].')
            self.mraa_i2c.write_i2c_data(self._REG_DATE, self.int_to_bcd(date))

        if day is not None:
            if day < 1 or day > 7:
                raise ValueError('Day is out of range [1,7].')
            self.mraa_i2c.write_i2c_data(self._REG_DAY, self.int_to_bcd(day))

```

Executable Snippets

05

DATAHANDLING

```python
import json


class DB_Handler:

    def __init__(self):
        pass

    def get_logs(self):
        with open("log.txt", "r") as f:
            return f.read()

    def get_last_access_time(self, key):
        with open("log.txt", "r") as f:
            data = f.read().split("\n")[::-1]
            # print(data)
            for line in data:
                # print(key,line)
                if key in line.strip():
                    # print(line.strip().split(" ")[:2])
                    return " ".join(line.strip().split(" ")[:2])

    def is_key_valid(self, key):
        with open("data.json", 'r') as f:
            access_keys = json.load(f)
            access_keys = access_keys["data"]["access-keys"]
            return key in access_keys

    def get_data(self):
        with open("data.json", 'r') as f:
            return json.load(f)

    def add_data(self, data):
        with open("data.json", "w") as f:
            f.write(json.dumps(data, sort_keys=True, indent=4))

    def get_access_keys(self):
        with open("data.json", 'r') as f:
            res = json.dumps(f.read())
            return res

    def add_access_key_and_name(self, name, key):
        data = self.get_data()
        key = key.strip()
        data["data"]["access-keys"].append(key)
        data["data"]["name"][key] = name.strip()
        print(data)
        self.add_data(data)

    def get_mode(self):
        with open("mode.txt", "r") as f:
            return f.read().strip()

    def change_name(self, name, key):
        data = self.get_data()
        data["data"]["name"][key] = name.strip()
        print(data)
        self.add_data(data)

    def delete_data(self, key):
        data = self.get_data()
        data["data"]["access-keys"].remove(key)
        data["data"]["name"].pop(key)
        self.add_data(data)

```

Executable Snippets

06

MULTITHREADING

```python

from threading import Thread


from request_handler import run_server_process
from rfid_handler import RFID_Handler,run_rfid_process

if __name__ == "__main__":


    t1 = Thread(target = run_server_process)
    t2 = Thread(target = run_rfid_process)
    t1.setDaemon(True)
    t2.setDaemon(True)
    t1.start()
    t2.start()

    while(True):
        pass


```

Executable Snippets

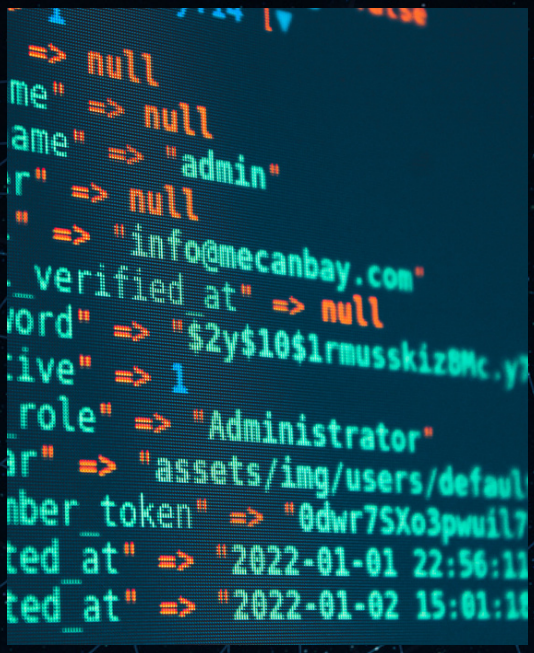
07

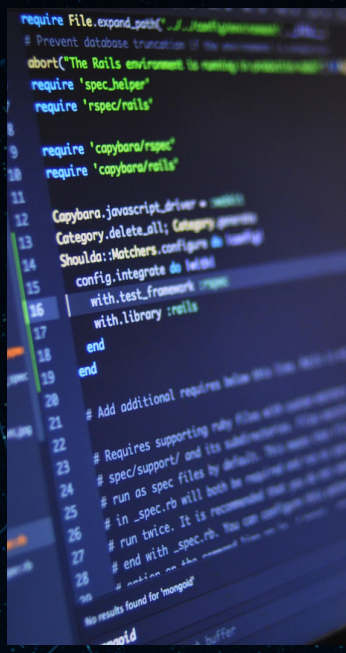
DATABASE

```
{
  "data": {
    "access-keys": [
      "1800891E5CD3",
      "3E00684A4A56"
    ],
    "name": {
      "1800891E5CD3": "Priyanshu Das",
      "3E00684A4A56": "Priyanshu Mishra"
    }
  }
}
```

Code Segmentation (G-Drive)

Part Four





CLICK HERE TO ACCESS THE CODE FILES OF THE PROJECT

Code Files

CLICK!

Testing and Observation (G-Drive)

Part Four

CLICK HERE TO ACCESS THE TESTING VIDEO FILE OF THE PROJECT

Video File

CLICK!

Scopes and Applications

SCOPES

ACCESS CONTROL MANAGEMENT

- The project enables organizations to effectively control access to restricted areas, ensuring that only authorized individuals can enter. This enhances security, protects sensitive information, and safeguards valuable assets.

TIME TRACKING

- The inclusion of time tracking features allows organizations to monitor and record the entry and exit times of individuals. This helps in tracking attendance, managing work hours, and ensuring compliance with scheduling requirements.

REMOTE ADMINISTRATION

- The project offers remote administration capabilities through a locally hosted server and a user-friendly web interface. This allows administrators to manage access control settings and user information from anywhere, providing convenience and flexibility in system administration.

Scopes and Applications

APPLICATIONS

CORPORATE AND OFFICE ENVIRONMENTS

- The project finds extensive application in corporate offices, enabling controlled access to areas such as server rooms, data centers, executive offices, and confidential document repositories. It ensures that only authorized personnel can enter these areas, protecting sensitive information and maintaining the integrity of critical systems.

EDUCATIONAL INSTITUTIONS

- Schools, colleges, and universities can utilize the project to manage access to areas such as laboratories, research facilities, administrative offices, or library archives. It ensures that only authorized students, faculty, and staff can access these areas, enhancing safety and protecting valuable resources.

HEALTHCARE FACILITIES

- The project is highly applicable in healthcare facilities, including hospitals, clinics, and medical centers. It helps control access to areas such as operating rooms, medication storage, patient records, and laboratories. This ensures patient privacy, protects sensitive medical data, and promotes compliance with regulatory requirements.

Scopes and Applications

APPLICATIONS

INDUSTRIAL AND MANUFACTURING FACILITIES

- Manufacturing plants, factories, and industrial sites can benefit from the project by securing areas containing hazardous materials, sensitive processes, or high-value equipment. It helps prevent unauthorized access, enhances workplace safety, and protects intellectual property.

RESIDENTIAL BUILDINGS AND COMMUNITIES

- The project can be implemented in residential buildings and gated communities to manage access to common areas such as gyms, swimming pools, parking lots, and shared facilities. It provides residents with a secure and convenient way to access amenities, ensuring a controlled living environment.

TRANSPORTATION HUBS

- Airports, train stations, and transportation hubs can benefit from the project by implementing access control for critical areas such as control rooms, baggage handling areas, or restricted zones. This ensures the safety and security of passengers, staff, and critical infrastructure.

THANK YOU!

TOGETHER, WE HAVE ACHIEVED AN EXCEPTIONAL MILESTONE, AND WE LOOK FORWARD TO CONTINUING OUR PURSUIT OF EXCELLENCE IN FUTURE ENDEAVORS. THANK YOU FOR YOUR SUPPORT.

PRIYANSHU DAS

20BCSE21

PRIYANSHU MISHRA

20BEEB84