Access Control System
Software Design and Architecture (CSC 419)
Group 3

1. Olorunfemi-Ojo Daniel Tomiwa 190805503
2. Sanusi Adeyemi Abdulazeez 180805050
3. Muhammed Sodiq Akande 180805056
4. Ezeani David 180805081
5. Ogunrinde Motunrayo Deborah 190805514

Table of Contents

1. Overview

The proposed Access Control System is designed to restrict access to designated spaces using RFID technology integrated with Raspberry Pi devices. This report provides an overview of the system's architecture, including its components, communication protocols, and workflow. The system would consist of the following architectures:

- Event-Driven Architecture
- Client-Server Architecture (Thin Client)
- Service-Oriented Architecture

1.1 Components of the System

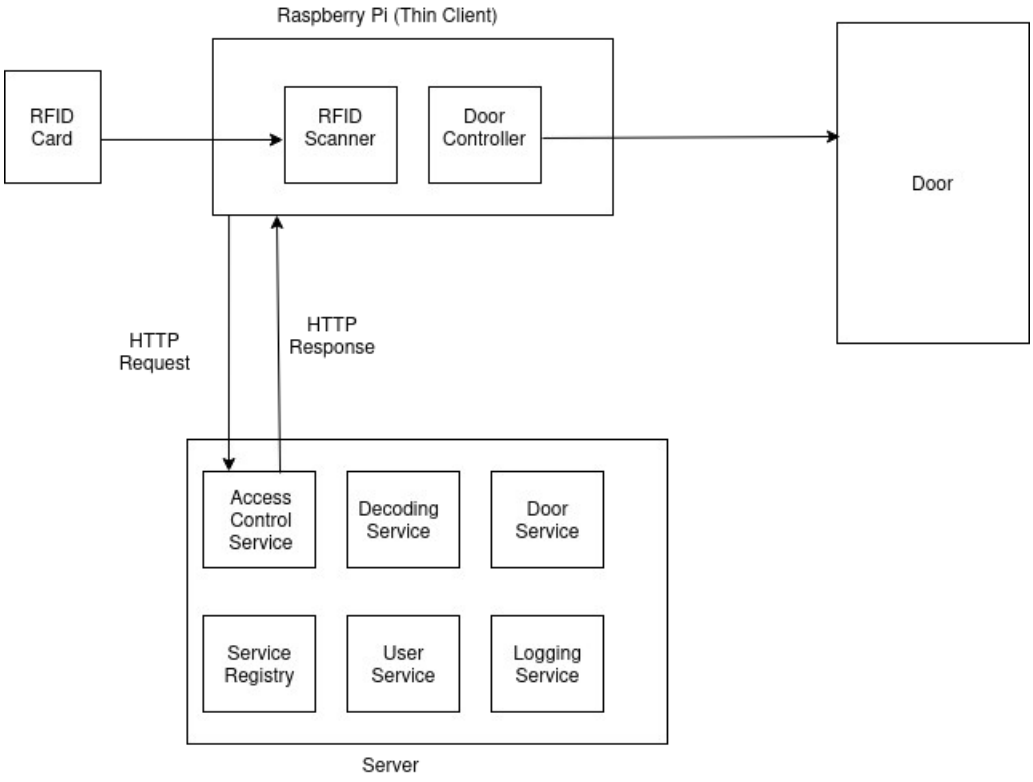
1. RFID Scanner:
    - An RFID (Radio-Frequency Identification) scanner is a device that reads RFID tags or cards to identify individuals or objects.
    - It consists of an antenna to transmit and receive radio signals, a reader to interpret the signals, and a controller to process the data.
    - The RFID scanner captures the unique identifier stored on RFID cards or tags when they are brought within proximity of the scanner.
    - In the context of the access control system, the RFID scanner serves as the entry point where users present their RFID cards for authentication.
2. Raspberry Pi:
    - A Raspberry Pi is a small, affordable single-board computer widely used for various embedded applications and projects.
    - In the access control system, the Raspberry Pi acts as a local processing unit or gateway for interfacing with the RFID scanner and transmitting data to the central server.
    - It typically runs software to manage communication with the RFID scanner, process authentication requests, and relay information to the central server.
    - The Raspberry Pi may also handle additional functionalities such as local caching of access control data, logging events, or implementing access control policies.
3. Central Server:
    - The central server serves as the backbone of the access control system, responsible for managing user authentication, access permissions, and system administration.
    - It stores a centralized database of user credentials, access rights, and access logs.
    - The central server communicates with RFID scanners, Raspberry Pi devices, and other components to coordinate access control operations.

- It enforces access policies, verifies user credentials, logs access events, and generates alerts for security incidents.
- Additionally, the central server may provide a web-based interface or APIs for administrators to configure access control settings, monitor system activity, and generate reports.

4. RFID Card:
   - An RFID card is a physical token containing an embedded RFID chip that stores a unique identifier.
   - Each RFID card is associated with a specific user or entity within the access control system.
   - When presented to an RFID scanner, the RFID card transmits its unique identifier to the scanner, which is then processed by the system to determine whether access should be granted or denied.
   - RFID cards are typically carried by users and used to authenticate their identity when accessing secured areas or resources.
   - The access control system maintains a mapping between RFID card identifiers and user profiles stored in the central server's database, allowing for efficient authentication and authorization processes.

These components work together to form a cohesive access control system, ensuring that only authorized individuals are granted access to designated areas or resources based on predefined security policies and permissions.

2. Proposed Architecture



2.1 Event-Driven Architecture

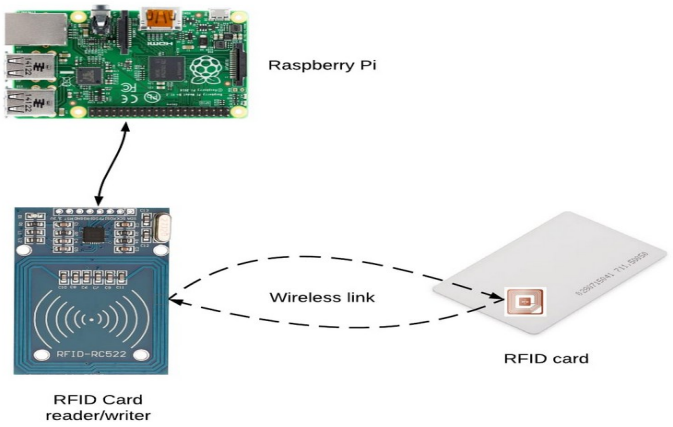
Event-driven architecture (EDA) is a design pattern that focuses on the production, detection, consumption, and reaction to events that occur in a system. In the context of the Access Control System, events could include actions such as a user presenting an RFID card for scanning, a door sensor detecting movement, or an access request initiated from a user interface.

One of the key advantages of EDA is its responsiveness to real-time events. When an event occurs, the system can immediately trigger the appropriate action, such as checking the access permissions associated with the RFID card and granting or denying access accordingly. This real-time responsiveness is critical for ensuring efficient and secure access control.

However, implementing an event-driven architecture requires careful consideration of event processing mechanisms, scalability, and fault tolerance. As the system scales up to handle a larger number of events or concurrent users, ensuring the reliability and consistency of event processing becomes increasingly challenging. Techniques such as event buffering, distributed event processing, and load balancing may be necessary to address these challenges effectively.

At every door, would be an RFID (Radio Frequency Identification) scanner connected to the GPIO (General-Purpose Input/Output) slot of a Raspberry Pi. The Raspberry Pi continuously listens for

events triggered by the RFID scanner. When an RFID card (contains a signature that uniquely identifies an individual) is scanned, an event is detected, and the Raspberry Pi initiates a process to send the scanned data to the Authentication Server. The data sent to the authentication server includes the RFID card signature and the ID of the door trying to be accessed. The hostname of the Raspberry Pi would serve as the Door ID. If access is to be granted, the controller opens the door.




```
import requests
from dnssd import service_discovery
from mfrc522 import SimpleMFRC522

# Door ID
def get_hostname():
    try:
        hostname = socket.gethostname()
        return hostname
    except Exception as e:
        print("An error occurred while getting hostname:", e)


rfid = SimpleMFRC522()
# Get access control service url from registry
url = service_discovery(‘access-control’)

# Listen for data from RFID scanner
while True:
    try:
        id, text = rfid.read()
        payload = {'door': get_hostname(), 'user': text}
        response = requests.post(url, json=payload)
        if response.status_code == 200:
            # Open door

    except:
        continue
```

2.2 Client-Server Architecture

The client-server architecture divides the system into two main components: the client, which handles user interactions and input, and the server, which performs the processing and decision-making tasks. In the context of the Access Control System, the client could be a thin client device equipped with RFID readers, while the server could be a central control unit

responsible for verifying access permissions and controlling physical access to designated spaces.

One of the primary advantages of the client-server architecture is its centralized control and management. By centralizing access control decisions and enforcement, the system can ensure consistency and adherence to security policies across all access points. Additionally, centralization simplifies system administration tasks such as user management, access policy configuration, and auditing.

However, the client-server architecture also introduces potential single points of failure and scalability bottlenecks. If the central server becomes unavailable or overloaded, it could disrupt access control operations across the entire system. Redundancy measures such as backup servers and load balancing can help mitigate these risks, but they also add complexity and cost to the system.

Furthermore, the reliance on a central server for access control decisions introduces security concerns. Malicious actors could target the server to gain unauthorized access or disrupt system operations. Implementing robust security measures such as encryption, access controls, and intrusion detection systems is essential to mitigate these risks effectively.

The Raspberry Pi acts as a client, sending access request data to the Authentication Server. The server processes the request, performs authentication tasks, and communicates the access decision back to the Raspberry Pi. The Raspberry Pi is a thin client reason being all the processing is done on the server side and therefore, just needs to know whether or not to grant access.

2.3 Service-Oriented Architecture

Service-oriented architecture (SOA) is an architectural style that emphasizes the use of services as the fundamental building blocks of a system. Each service encapsulates a specific functionality and exposes a well-defined interface for interaction with other services. In the context of the Access Control System, services could include authentication, authorization, logging, and event processing.

One of the key advantages of SOA is its modularity and reusability. By decomposing the system into independent services, each service can be developed, deployed, and maintained separately, allowing for greater flexibility and scalability. New services can be added or existing services can be modified without impacting the overall system architecture, facilitating agile development and integration.

Additionally, SOA promotes loose coupling between services, enabling them to communicate asynchronously through standardized interfaces such as RESTful APIs or message

queues. This loose coupling reduces dependencies between components, making the system more resilient to changes and failures.

Various services within the Authentication Server collaborate to handle access requests. These services include the Decoding Service, User Service, Door Service, Access Control Service, and Logging Service, each responsible for specific tasks in the authentication process.

1. **Decoding Service:** Decodes the RFID signature retrieved from the scanned data.
2. **User Service:** Fetches user data, including organizational group membership.
3. **Door Service:** Retrieves door-specific data, including organizational group access permissions.
4. **Access Control Service:** Processes access requests, compares user and door group permissions, and grants or denies access accordingly. This service is what the Raspberry Pi interacts with. This service queries a Service Registry to get the domain info with regards to other services and performs the Access Control logic.
5. **Logging Service:** Records access request details and decisions for auditing purposes.

```
// Dummy implementation
import org.springframework.stereotype.Service;

@Service
public class AccessControlService {

  public HTTPStatus doPost(JSONObject json) {
      String data = deocodeRFIDSignature(json.get("user"));
      User user = fetchUserData(data)
      Door door = fetchDoorData(json.get("hostname"));
      if (door.getPermittedGroups.contains(user.getGroup())) {
         logAccessRequest(...);
         return HTTPStatus.OK;
      }

      logAccessRequest(...);
      return HTTPStatus.UNAUTHORIZED;
  }

  // Log access requests
  public void logAccessRequest(String userId, String doorId, boolean
accessGranted, String timestamp) {
    Service loggingService = getServiceUrlFromRegistry("LoggingService");
    loggingService.send(userId, doorId, accessGranted, timestamp);
  }

  // Decoding Service: Decodes the RFID signature retrieved from the scanned
data
  public String decodeRFIDSignature(String scannedData) {
    Service decodingService = getServiceUrlFromRegistry("DecodingService");
```

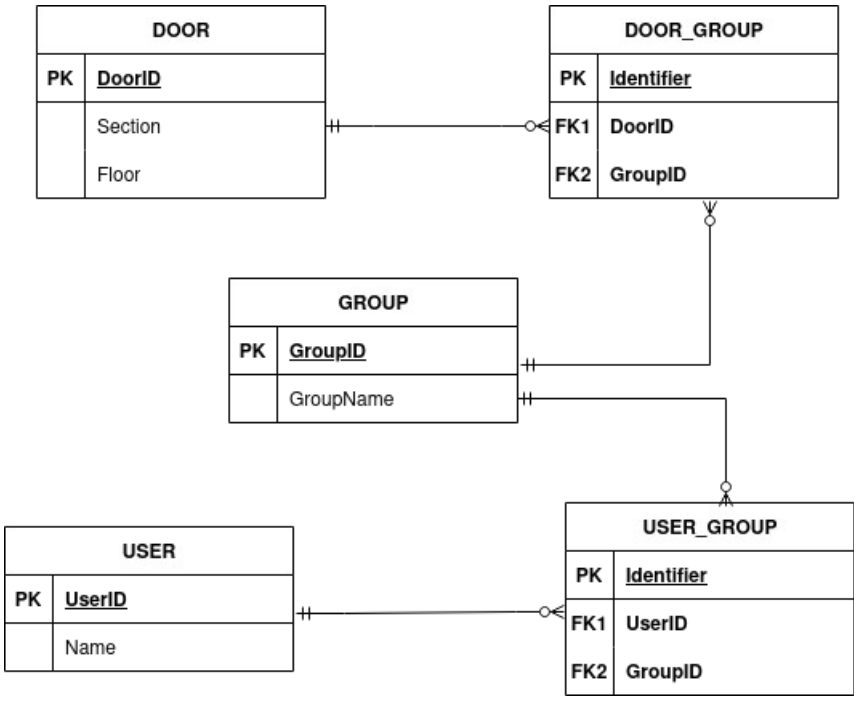
```
    return decodedService.decode(scannedData);
  }

  // User Service: Fetches user data, including organizational group membership
  public User fetchUserData(String userId) {
    Service userService = getServiceUrlFromRegistry("UserService");
    return userService.get(userId);
  }

  // Door Service: Retrieves door-specific data, including organizational group
access permissions
  public Door fetchDoorData(String doorId) {
    Service doorService = getServiceUrlFromRegistry("DoorService");
    return doorService.get(doorId);
  }

}
```

3. Entity Relationship Diagram



1. Door
   - The "Door" entity represents physical entry points or access points within the facility.
   - Each door has attributes such as a unique identifier (DoorID), door name, location, and possibly other properties like access control settings (e.g., access level required, schedule restrictions).
2. User
   - The "User" entity represents individuals or entities granted access to the facility or resources.
   - Attributes of the user entity may include a unique identifier (UserID), username, password, biometric data (if applicable), contact information, and other relevant details.
3. Door Group
   - The "Door Group" entity is a logical grouping of doors that share similar access control settings or policies.
   - Each door group has attributes like a unique identifier (GroupID), group name, description, and possibly other properties defining its membership or access rules.
4. User Group
   - The "User Group" entity represents a collection of users with similar access rights or roles within the organization.
   - Attributes of the user group entity may include a unique identifier (UserGroupID), group name, description, and possibly other properties defining its membership or permissions.
5. Group
   - The "Group" entity serves as a generic entity to establish a many-to-many relationship between users and doors or between user groups and door groups.

    - It acts as an intermediary entity to link users to doors and user groups to door groups, facilitating the assignment of access permissions.
    - The group entity may have attributes such as a unique identifier (GroupID) and possibly additional properties related to the association between users, doors, user groups, and door groups.

In this ERD:

- Each user can be associated with multiple groups (User to Group relationship).
- Each door can be associated with multiple groups (Door to Group relationship).
- Each user group can be associated with multiple doors (User Group to Door relationship).
- Each door group can be associated with multiple users (Door Group to User relationship).
- The "Group" entity serves as a junction table to manage the many-to-many relationships between users, doors, user groups, and door groups.

This ERD provides a clear visualization of how users, doors, user groups, and door groups are related within the access control system, facilitating the assignment of access permissions and the management of access control settings.

4. Benefits

- The event-driven architecture provides a streamlined approach to handling RFID scanner events, allowing the system to react promptly to access requests.
- The client-server architecture enables clear separation of concerns between the Raspberry Pi client and the Authentication Server, facilitating effective access control management.
- The service-oriented architecture fosters a flexible and modular approach to access control management, allowing for effective composition and orchestration of services to meet system requirements.

5. Conclusion

By employing event-driven, client-server, and service-oriented architectures, the Access Control System achieves a robust and efficient mechanism for managing access to restricted spaces. Each architectural approach contributes unique advantages to the system's overall design, ensuring reliability, scalability, and maintainability in access control operations.