Politecnico di Torino
Project and laboratory on embedded communication systems

Luca Bernardi - s319539,
Francesco Ilacqua- s303417
Federico Villata - s319922

Academic Year 2023-2024

Contents

1 Frontend

1.1 Introduction

This part of the report explores the frontend portion of a Flutter application, explaining the motivation behind choosing Dart and Flutter, the functionalities of different user roles, and the implemented pages. The application is designed to manage users, companies, and related statistics, providing an intuitive interface and advanced functionalities for access management and user emulation.

1.2 Flutter and Dart

Flutter, developed by Google, is a robust framework for building natively compiled mobile, web, and desktop applications from a single codebase. Another motivation behind Flutter's choice is its accelerated development cycle. Features like hot reload allow developers to see changes in real time without restarting the application. This greatly speeds up the development process. Additionally, developers can create a consistent and aesthetically pleasing user interface across various platforms with Flutter's large library of customizable widgets.

Dart, also developed by Google, offers a simple and intuitive syntax that makes it productive to work with. Flutter's performance is optimized for the Dart language.

Flutter and Dart together provide a powerful toolkit for developing high-quality applications efficiently.

Using Flutter, both the web page and the phone application were developed. This allows a faster and more convenient solution for door control access. In addition, several functions have been introduced that depend on the specific role the user plays in the company such as viewing statistics or the ability to emulate users and many others that will be described.

1.3 Functionality of Roles

The application implements a robust role management system to regulate access to different functionalities. The considered roles include Super Administrator (SA), Customer Administrator (CA), Customer Operator (CO), and User (USR).

1.3.1 System Administrator - SA

System Administrators (SA) have the highest level of access, allowing them to manage all aspects of the system. Users with this role can add, remove, and modify all users in the system and can also emulate them. They can see system statistics and extract data from the database.

1.3.2 Customer Administrator - CA

Customer Administrators (CA) oversee the operations of a specific company. Users with this role can add, remove, and modify users with lower roles, such as customer operators and users. They can emulate any other users associated with the specific company, see the system statistics, add or remove rooms from a specific company, change the rules of entry to certain rooms, and extract data from the database.

1.3.3 Customer Operator - CO

Customer Operators (CO) have more limited access compared to CAs but still play a crucial role in managing company operations. Users with this role can add, remove, and modify users with the user role. They can emulate any other users associated with the specific company with roles equal or lower. Customer operators can also change the rules of entry to certain rooms for users with the user's roles and extract USR's data from the database.

1.3.4 User - USR

Users (USR) have the most basic access level. Users with this role can register, manage their own profile, see their own statistics, and extract their data from the database.

1.4 Implemented Pages

1.4.1 Login Page

The login page is the initial interface where users authenticate themselves using their username and password. This page incorporates features like password visibility toggling and a virtual alphanumeric keyboard to enhance user convenience and security. Upon successful login, users are directed to the homepage. On this page, there are also two buttons useful to redirect to the registration page and the forgot password page.

1.4.2 Registration Page

The registration page is dedicated to onboarding new users. It includes fields for entering personal details such as name, surname, fiscal code, phone number, email, address, birth date, gender, and an RFID token. This comprehensive form guarantees that all required data is gathered during the registration process.

1.4.3 Forgot Password Page and Reset Password Page

The forgot password page provides a simple mechanism to initiate the password recovery process. By entering their registered email, users can request a reset code. Once the reset code is received, they can navigate to the reset password page to set a new password, ensuring that their account remains secure.

1.4.4 HomePage

The homepage is a central hub that provides access to various functionalities based on the user's role. The homepage dynamically displays available actions such as viewing and managing companies, emulating users, changing roles, and registering new companies and users. Each button on the homepage serves a specific function, tailored to enhance the user's experience. Below is a detailed explanation of these buttons and their respective functionalities:

- Register Company: Redirects users to the register company page.
- Companies: The user sees all the companies with which they are associated. Clicking on a company button navigates the user to the company page.
- Modify Profile: Redirects users to the modify profile page.
- Emulate User (visible to SA): Allows users with the SA role to emulate other users to test and verify permissions and functionalities.
- Change Role (visible to SA): Enables SA to change the role of a specified user within a specified company.
- Register User (visible to SA): Allows SA to register a new user by providing necessary details.
- Enroll (visible to SA): Allows SA to enroll a user into a company with a specified role.
- Remove User (visible to SA): Enables SA to remove a user from the system.
- Remove Company (visible to SA): Enables SA to remove a company from the system.
- User List (visible to SA): Allows SA to view and manage all users in the system.
- Company List (visible to SA): Allows SA to view and manage all companies in the system.

1.4.5 Register Company Page

The register company page allows users to register new companies by providing details such as the company name, VAT number, address, phone number, email, and country.

1.4.6 Modify Profile Page

The modify profile page provides fields for editing personal details, including name, surname, fiscal code, phone number, email, address, birth date, gender, and RFID token. Additionally, it includes an option to change the password, ensuring that users can maintain the security and accuracy of their profile information.

1.4.7 Company Page

The company page is a specialized interface for managing company-specific tasks. It offers features like checking into rooms, emulating users, and managing permissions. This page is especially helpful for administrators who have to manage business operations and make sure that users have the proper access rights.

- Back: Provides an easy way to return to the homepage.
- Change Company: Allows users to switch their active company context.
- Select Room: Users can select a room to check-in. The selected room is used for tracking and managing user activities within the company.
- Add Room (visible to CA): Allows CA to add new rooms to the company's facilities.
- Remove Room (visible to CA): Enables CA to remove rooms.
- Permission Rights (visible to CA and CO): Allows CA and CO to set or modify access rights for users, specifying whether they are allowed or denied access to certain rooms.
- Emulate User (visible to CA and CO): Enables CA and CO to emulate other users within the company to test permissions and functionalities.
- Change Role (visible to CA and CO): Allows administrators to change the role of a user within the company.
- Enroll User (visible to CA and CO): Enables administrators to enroll new users into the company with specific roles.
- Remove User (visible to CA and CO): Allows administrators to remove a user from the company's user list.
- Statistics: Redirects to the statistics page.

1.4.8 Statistics Page

The statistics page provides a comprehensive overview of user activities and room usage statistics. This page displays data filtered by company and user role, allowing administrators to monitor and analyze usage patterns.

2 Backend

2.1 System Architecture

2.1.1 Framework and Libraries

The backend is implemented using the Flask framework, which provides a lightweight and flexible structure for building web applications. Several key libraries are utilized to extend the functionality of Flask:

- **MySQL Connector**: For connecting to and interacting with the MySQL database.
- **JWT**: For generating and decoding JSON Web Tokens used in user authentication.
- **SMTP**: For sending emails, such as password reset links.
- **Logging**: To enable detailed logging of backend operations for debugging and monitoring.
- **CORS**: To handle Cross-Origin Resource Sharing for the frontend applications.

2.1.2 Database Management

The MySQL `aziendadb` database is structured to manage various aspects of a company's operations, including user management, customer data, room permissions, activity logs, and usage statistics. This database ensures efficient organization and retrieval of data critical to the company's functioning. Below is a detailed description of each table within the database.

- **User Table**
  The `user` table contains detailed information about the users within the system. User management and authentication depend on this information. The primary columns include:
  - `userID`: A unique identifier for each user.
  - `nome`: The first name of the user.
  - `cognome`: The last name of the user.
  - `username`: The user's login name.
  - `password`: The hashed password for user authentication.
  - `fiscal_code`: The user's fiscal code.
  - `phone_number`: The user's contact number.
  - `mail`: The user's email address.
  - `address`: The user's physical address.
  - `birth_date`: The user's date of birth.

  - gender: The user's gender.
  - token: A security token possibly used for session management or additional security features.

- **Customer Table**
The customer table stores information about the company's customers. This data is crucial for customer relationship management and business operations. The main columns include:
  - customerID: A unique identifier for each customer.
  - companyName: The name of the customer's company.
  - VAT_number: The VAT number of the customer's company.
  - address: The company's address.
  - phone_number: The company's contact number.
  - PEC: The certified email address of the company.
  - subscription: The subscription type the customer holds.
  - country: The country where the company is located.

- **Association Table**
The association table is designed to map the roles of different users within various companies. This table is essential for defining the hierarchical and functional structure of the company. The key columns in this table include:
  - username: Identifies the user within the system.
  - companyName: Specifies the company the user is associated with.
  - role: Denotes the user's role within the company, such as SA (System Administrator), CA (Customer Administrator), CO (Customer Operator), or USR (User).

- **Rooms Table**
The rooms table defines the access rules for various rooms within the companies, detailing who can enter which rooms under what conditions. The main columns include:
  - ruleId: A unique identifier for each rule.
  - companyName: The company to which the room belongs.
  - roomName: The name of the room.
  - username: Specifies the user or role the rule applies to.
  - allowed_denied: Indicates whether access is allowed (1) or denied (0).

- **Permission Table**
The permission table manages user access to different rooms within a company. It records the necessary information to track and control user movements and permissions.

  - `id`: The primary key for the table, uniquely identifying each record. It is an auto-increment integer and part of the `PRIMARY` index.
  - `username`: Stores the username of the user. It is part of the `unique_permission` composite index to ensure each user has a unique permission record per company.
  - `companyName`: Stores the name of the company. It is part of the `unique_permission` composite index to ensure each user has a unique permission record per company.
  - `room`: Indicates which room the user is accessing or has permission to access.
  - `check_in`: Records the time the user checked into the room, helping track the duration of the user's stay.

- **Log Table**
  The `log` table records user activities within various rooms, providing a detailed audit trail. This is essential for security and monitoring purposes.
  - `id`: A unique identifier for each log entry. It is the primary key and part of the `PRIMARY` index.
  - `username`: The user who performed the activity. It is part of the `unique_log` composite index to ensure unique log entries per user, company, and room.
  - `companyName`: The company the user belongs to. It is part of the `unique_log` composite index to ensure unique log entries per user, company, and room.
  - `room`: The room where the activity occurred. It is part of the `unique_log` composite index to ensure unique log entries per user, company, and room.
  - `check_out`: The timestamp when the user checked out of the room.
  - `time_in_room`: The duration the user spent in the room.

2.1.3 Security Measures

Security is a critical aspect of the backend implementation. Key security measures include:

- **Password Hashing**: User passwords are hashed using MD5 before being stored in the database.
- **Token-Based Authentication**: Secure tokens are generated using JWT for session management.
- **Input Validation**: All user inputs are validated to prevent SQL injection and other attacks.

Input Validation All user inputs are validated to prevent SQL injection and other attacks. This is achieved through the use of regular expressions (regex) to ensure that the input data conforms to the expected formats.

Regex-Based Validation The system uses a set of predefined regex patterns stored in a JSON file to validate various user inputs. These patterns ensure that the input data meets specific criteria, such as format and length, for fields like names, usernames, emails, phone numbers, and more.

- **Name and Surname**: Only allows alphabetic characters and specific special characters.
- **Username**: Allows alphanumeric characters and specific symbols, with a length between 3 and 16 characters.
- **Email**: Ensures a valid email format.
- **Phone Number**: Allows digits with an optional leading '+' sign, ensuring a length between 10 and 15 characters.
- **Fiscal Code**: Follows the specific format for fiscal codes.
- **Birth Date**: Validates the date format as DD/MM/YYYY.
- **Gender**: Accepts predefined gender values.

Regex Management The regex patterns are managed through a JSON file, which can be read and updated as needed. The following functions handle regex management:

- **read_json_file**: Reads the content of a JSON file and returns it as a dictionary.
- **write_json_file**: Writes a dictionary to a JSON file. It can overwrite existing files if the *force* flag is set to *True*.
- **load_patterns**: Loads regex patterns from the JSON file into the system.
- **regex_full_check**: Validates the input data against the loaded regex patterns. If any input does not match its corresponding pattern, the function returns an error with a suggestion.

2.2 User Management

2.2.1 User Registration and Login

The registration endpoint allows new users to sign up by providing the necessary details. During registration, user data is validated, and passwords are hashed before being stored.

The login endpoint authenticates users by verifying their credentials against the stored hashed passwords. Upon successful login, a JWT token is generated and returned to the user for session management.

2.2.2 Role-Based Access Control

The system supports multiple user roles, including System Administrator (SA), Customer Administrator (CA), Customer Operator (CO), and User (USR). Each role has specific permissions that define their access to various functionalities within the system.

2.3 Token Management

2.3.1 Token Generation

Tokens are generated using the JSON Web Token (JWT) standard. When a user successfully logs in, a JWT token is created containing user information, roles, and an expiration time. The token is then returned to the client to be used for authenticated requests.

- **Structure**: The token includes the username, roles, company information, and expiration timestamp.
- **Secret Key**: A secret key is used to sign the token, ensuring its integrity and authenticity.

2.3.2 Token Usage

Tokens are included in the Authorization header of subsequent requests to authenticate the user. The backend verifies the token's validity before processing the request.

2.3.3 Token Validation

Upon receiving a request, the backend extracts the token from the Authorization header and decodes it using the secret key. If the token is valid and not expired, the request is processed. Otherwise, an authentication error is returned.

- **Expiration**: Tokens have a set expiration time, after which they are no longer valid. This enhances security by limiting the window of opportunity for potential misuse.
- **Error Handling**: Invalid or expired tokens result in appropriate error responses, prompting the user to re-authenticate.

2.4 Endpoints and Functionalities

2.4.1 Company and Room Management

Endpoints are provided for managing companies and rooms within the system. Administrators can add, update, or remove companies and rooms as needed.

2.4.2 Check-in and Check-out

The backend supports check-in and check-out functionalities, allowing users to log their presence in specific rooms. This is critical for tracking user activities and managing access control.

2.4.3 Statistics and Logging

Detailed logs are maintained for all user activities, including room access and duration of stay. Administrators can view and analyze these statistics to ensure compliance and optimize resource utilization.

2.4.4 Password Management

The system includes endpoints for password recovery and reset. Users can request a password reset link via email, and reset their passwords securely. When a user requests to reset their password, a temporary password is generated and sent to the user's registered email address. The process involves the following steps:

- **Generate Temporary Password**: A temporary password is generated using a combination of letters, digits, and punctuation to ensure security. The `generate_password` function creates a random password with a default length of 12 characters.
- **Send Email**: The temporary password is sent to the user's email address using the `smtplib` library. An email is composed with the temporary password and sent using the SMTP protocol. The `send_mail_pwd` function handles the creation and sending of this email.
- **Update Database**: The temporary password is hashed and stored in the database, replacing the user's previous password. The user is then instructed to log in using the temporary password and change it immediately.

2.4.5 Role Management

System Administrators can change the roles of users within a company, allowing for flexible and dynamic access control. This includes promoting or demoting users to different roles as necessary.

2.4.6 User and Company Retrieval

The backend provides endpoints to retrieve details about users and companies. Administrators can fetch comprehensive details about users and companies for management and auditing purposes.

3 Integration with Hardware

3.0.1 Raspberry Pi and RFID

The system integrates with Raspberry Pi-based door access controllers equipped with RFID card readers. This allows for seamless and secure door access management using RFID tokens.

RPi Token Management The `rpi_token` endpoint facilitates the interaction between the backend system and Raspberry Pi-based door access controllers. This endpoint processes RFID token data to manage room access and check-in activities. The functionality is supported by various helper functions defined in the `revisited.py` module.

3.0.2 Room Access Verification

The function `check_room_access_revisited` checks whether a user has access to a specified room within a company. It verifies the user's role and specific permissions for the room.

- **Parameters**:
  - `current_username`: The user's username.
  - `room_name`: The room's name.
  - `company_name`: The company's name.
  - `user_role`: The user's role within the company.
- **Functionality**:
  - Checks explicit and role-based permissions for room access.
  - Returns a message and status code based on access verification.

3.0.3 User Check-In

The function `check_in_revisited` manages the user check-in process, recording the time spent in the previous room and updating the current room.

- **Parameters**:
  - `username`: The user's username.
  - `companyName`: The company's name.
  - `room`: The room's name.
- **Functionality**:
  - Records the check-out time and duration for the previous room.
  - Updates the database with the check-in time and current room.
  - Returns a success or failure message for the check-in operation.

3.0.4 Movement Logging

The function `move_to_room_w_log` logs user movements between rooms to maintain accurate records.

- **Parameters**:
  - `username`: The user's username.
  - `companyName`: The company's name.
  - `room`: The room's name.
- **Functionality**:
  - Updates the user's current room and check-in time.
  - Ensures movements are recorded for auditing purposes.
  - Returns a success or failure message for the operation.

3.0.5 Client-Side Interaction : application side

The script imports several standard and third-party Python libraries:

- `requests` for making HTTP requests.
- `tkinter` for GUI components.
- `json` for JSON file handling.
- `string` and `random` for generating random strings.
- `RPi.GPIO` and `mfrc522` for interacting with Raspberry Pi GPIO and RFID reader.

Main Functions

- **write_json_file**: Writes a dictionary to a JSON file, optionally forcing an overwrite.
- **read_json_file**: Reads and returns the contents of a JSON file as a dictionary.
- **fetch_data**: Sends data to a Flask server and handles the response.
- **generate_random_string**: Generates a random string of specified length.

RFID Functions for the module MFRC522

- **init**: Initializes the GPIO pins and RFID reader.
- **read_card**: Reads data from an RFID card.
- **write_card**: Writes data to an RFID card.
- **check_t**: Checks if the provided text matches the data on the RFID card.
- **blink_green**, **blink_red**: Blink LED lights to indicate status.
- **open_door**, **wrong_door**, **error_door**: OPTIONAL Control the door mechanism using LEDs.

GUI Functions

- **show_popup**: Displays a popup window with response data generated by the backend.
- **show_error**: Displays an error message in a popup window.

Execution and Signal Handling

- The script sets up a logger and writes the default initial JSON data.
- It contains an infinite loop for reading RFID data and processing it.
- Signal handlers are defined to gracefully exit on interruptions.

3.0.6 Client-Side Interaction: RPI hardware side

Also is provided another script, that initializes the Raspberry, in a way that disables all USB ports, and Bluetooth, and installs all dependencies if are not installed. Launches the Python client and also a chromium-browser window in kiosk mode, in order to make it impossible to do other actions on the Raspberry Pi. The ssh port is kept open in order to make, the RPI accessible via the reverse shell. The RPI setup is quite easy, almost plug and play, it is just need to download Raspbian and clone the GitHub project inside it, and run once an initialization script, which can be done, by placing it in a specific path.

Additional Hardware was not included, since that in a scalable distribution implies another cost to add to the bill, but also more problems to interface with. By utilizing a simple RPI with a touchscreen display, purchasable directly by the reseller without adding any part on top of it. It is possible to sell only the Software part without the hardware, without the need for technicians.

3.0.7 Smartphone Application

Since the whole app was developed in Flutter, it is easy to build apps for almost every device like, android smartphones, IOS smartphones, WEB apps, Linux, and Windows apps. This is very helpful, because by editing a single source code is possible to edit all devices, but also by publishing many applications, this can reduce the HTTP server workload, and keep only the backend part.

4 Conclusion

The backend implementation for the door and presence control system provides a robust and scalable foundation for managing access control within various types of companies and restricted areas. By leveraging Flask, MySQL, and JWT, the system ensures secure and efficient handling of user authentication, role-based access control, and data management. The integration with Raspberry Pi and optional smartphone application further enhances the system's capabilities, making it a comprehensive solution for modern access management needs.