SMARTOffice

technical documentation

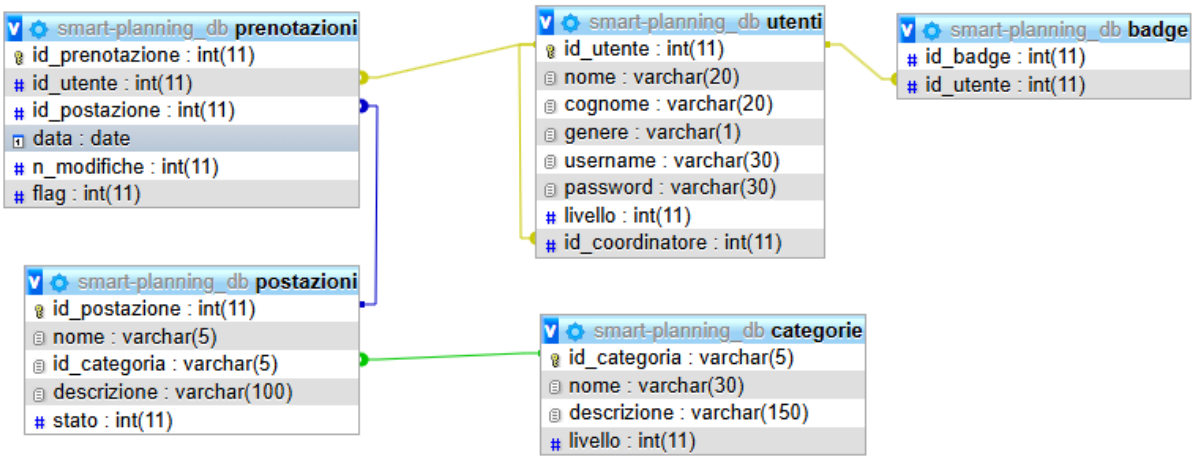
Caldarone Ilaria – Cerri Sara – D'Aranno Rebecca - Ruggeri Elena

Index

SMARTPlanning

database

The database **smart-planning_db** on EasyPHP MySQL (InnoDB) defines relational tables with primary and foreign keys for data integrity. It includes indexed integer and varchar fields, date attributes, and supports relationships between users, bookings, workstations, categories, and badges.



Nuxt JS/dipendenze

Nuxt 3.10

Description

Nuxt.js is a framework built on top of Vue.js that enables the development of universal web applications, running both on the client side and the server side. Nuxt version 3.10 introduces a variety of improvements, including support for Vue 3's Composition API, enhanced performance, and a simplified configuration process.

Official link

- Official NUXT JS website

Installation

npm install nuxt@3.10

V-Calendar 3.10

Description

V-Calendar is a calendar library for Vue.js that allows developers to easily integrate interactive calendars into their applications. Version 3.10 is the latest release and includes new features for customizing and embedding calendars in your web pages.

Link ufficiale

- Official V-Calendar website

Installation

V-Calendar has been installed as a project dependency using the following command:

npm install v-calendar@next

Pinia

Descrizione

Pinia is a state management library for Vue 3, developed as an alternative to Vuex. It is designed to be simple, scalable, and highly integrated with Vue 3. Pinia is particularly useful when you need to manage global state within your application.

Officila link

- Official Pinia website

Installation

npm install pinia

STORES

Store/models

Categoria

```
interface Categoria {
  id_categoria: String;
  nome: String;
  descrizione: String;
  livello: number;
}
```

Utente

```
interface Utente {
  id_utente: number;
  nome: String;
  username: String;
  cognome: String;
  genere: String;
  livello: number;
  id_coordinatore: number;
}
```

Postazione

```
interface Postazione {
  id_postazione: number;
```

```
  nome: String;
  id_categoria: String;
  descrizione: String;
  stato: number;
}
```

Prenotazione

```
interface Prenotazione {
  id_prenotazione: number;
  id_utente: number;
  id_postazione: number;
  data: Date;
  n_modifiche: number;
  flag: number;
}
```

Utente

```
interface Utente {
  id_utente: number;
  nome: String;
  username: String;
  cognome: String;
  genere: String;
  livello: number;
  id_coordinatore: number;
}
```

Store/prenotazioni.ts

Stato

prenotazioni: Prenotazione[]

---

getPrenotazioni(): Promise<void>

Recupera le prenotazioni dal server per l'utente loggato.

- Se l'utente ha livello 3 (admin), carica tutte le prenotazioni presenti sul database.
- Ordina le prenotazioni per data.
- Richiede validazione della sessione

API chiamata:

- POST getPrenotazioni.php per utenti di livello 1 e 2
- POST getPrenotazioniAdmin.php per admin (livello 3)

---

nuovaPrenotazione(data: Date, id_postazione: number): Promise<void>

Crea una nuova prenotazione per l'utente corrente.
Invia id_utente, id_postazione e data al server.

- Dopo l'inserimento, aggiorna le prenotazioni e le ordina.
- Reindirizza alla pagina "home".

Parametri:

- data: Data della prenotazione
- id_postazione: ID della postazione da prenotare

API chiamata:

- POST insertPrenotazioni.php

---

modificaPrenotazione(prenotazione: Prenotazione): Promise<void>

Modifica una prenotazione esistente.
Invia le modifiche al backend (id_prenotazione, data, id_postazione, n_modifiche).

- Dopo la modifica, ricarica le prenotazioni e reindirizza alla "home".

API chiamata:

- POST /updatePrenotazione.php

---

eliminaPrenotazione(id: number): Promise<void>

Elimina la prenotazione con l'id fornito.
Rimuove la prenotazione dallo stato locale e aggiorna i dati da server.

API chiamata:

- POST eliminaPrenotazione.php

---

filtraData(data: string): Prenotazione[]

Filtra le prenotazioni aventi una determinata data.

Parametri:

- data: Data in formato stringa
-

---

filtraCategoria(categoria: string): Prenotazione[]

Filtra le prenotazioni in base alla categoria della postazione.

Parametri:

- categoria: ID della categoria da filtrare

Nota: Richiede accesso allo store postazioni-store.

---

ordinaData(): void

Ordina l'array prenotazioni in ordine decrescente per data.

---

getPrenotazioneById(id: number): Prenotazione | undefined

Restituisce una copia della prenotazione corrispondente all'id fornito.

---

abilita(id_prenotazione: string): Promise<void>

Abilita una prenotazione specifica.

Effetti collaterali: Aggiorna lo stato ricaricando tutte le prenotazioni.

API chiamata:

- POST /abilitaPrenotazione.php

---

disabilita(id_prenotazione: string): Promise<void>

Disabilita una prenotazione specifica.

Effetti collaterali: Aggiorna lo stato ricaricando tutte le prenotazioni.

API chiamata:

- POST /disabilitaPrenotazione.php

getDatePrenotate(): string[]

Restituisce un array con le date di tutte le prenotazioni presenti nello stato.

Dipendenze esterne

- **useAuth()**: per recuperare utente, address, validare la sessione
- **usePostazioni()**: per ottenere info su postazioni e categorie
- **$fetch**: wrapper Nuxt per fetch API
- **useRouter()**: per effettuare redirect lato client

Store/postazioni.ts

State

postazioni: Postazione[]

categorie: Categoria[]

occupate: any[]

disabilitate: any[]

---

getPostazioni(): Promise<void>

Loads the list of workstations from the backend (only if not already loaded).

API call:

- POST getPostazioni.php

---

getCategorie(): Promise<void>

Loads the list of workstation categories from the backend (only if not already loaded).

API call:

- POST getCategorie.php

---

checkPostazioniOccupate(data: Date): Promise<void>

Checks which workstations are occupied on a given date.

API call:

- POST checkPostazioniOccupate.php

---

getCategoria(postazione: Postazione): Categoria | undefined

Returns the category associated with a given workstation.
Parameters: Postazione
Output: Associated category

---

getPostazione(prenotazione: Prenotazione): Postazione | undefined

Returns the workstation associated with a booking.
Parameters: Prenotazione
Output: Associated workstation

---

getPostazioneById(id: number): Postazione | undefined

Returns a workstation by its ID.
Parameters: numeric id
Output: Corresponding workstation

---

abilita(id_postazione: string): Promise<void>

Enables a workstation (reactivates it).
API call:
- **POST** /abilitaPostazione.php

---

disabilita(id_postazione: string): Promise<void>

Disables a workstation (makes it unbookable).

API call:

• POST /disabilitaPostazione.php

---

getPostazioniDisabilitate(): Promise<void>

Retrieves the list of disabled workstations.

API call:

• POST getPostazioniDisabilitate.php

---

getDateOccupate(id_postazione: number): Promise<string[]>

Returns an array of dates on which a workstation is already occupied.

API call:

• POST getDateOccupate.php

External dependencies

• useAuth(): for authentication, session management, and API address

• $fetch: for making HTTP requests

• Models: Postazione, Categoria, Prenotazione

Store/dipendenti.ts

State

dipendenti: Utente[] // Lista dei dipendenti caricati

---

getDipendentiCoordinatore(id: number): Promise<void>

Retrieves the list of employees associated with a coordinator.

API call:

- POST getDipendentiCoordinatore.php

Filter employees by a keyword (username or id_utente)

Input: search string

Output: filtered list of Utente[]

Case-insensitive match:

- username
- id_utente

---

getPrenotazioniDipendente(id: string): Promise<Prenotazione[]>

Returns all bookings made by an employee.

API call:

- POST getPrenotazioni.php

---

insertUtente(utente: Utente, password: string): Promise<any>

Inserts a new user into the system.

API call:

• POST insertUtente.php

---

updateUtente(utente: Utente, password: string): Promise<void>

Updates the information of an existing user.

API call:

• POST updateUtente.php

Request body: identical to insertUtente

---

getDipendenteById(id: number): Utente | undefined

Returns a Utente object searched by ID.

Input: user ID

Output: copy of the found Utente object (or undefined)

---

getCoordinatori(): Utente[]

Returns the list of users with level == 2 (coordinators).

Output: list of Utente[]

External dependencies

• useAuth(): for session management and API address

• $fetch: for making HTTP requests

• Models: Prenotazione, Utente

Store/auth.ts

Manages authentication, user session, login/logout, and local persistence (localStorage).

State

utente: Utente // Utente loggato

sessione: {

id_sessione: number, // 0 = non attiva

scadenza: number,

controlCode: number // Token anti-bruteforce

}

address: string // URL base for API

---

init(): void

Initializes data from localStorage (if present).
Used to restore the user session after a refresh.
Effects: Populates user and session

---

login(id: number, password: string): Promise<string | void>

Performs user login.
API call:
• **POST** login.php

controllaSessione(): Promise<boolean>

Checks if the session is still valid.

API call:

- POST sessionCheck.php

---

logout(): Promise<boolean>

Terminates the user session.

API call:

- POST logout.php

---

setLocalStorage(): void

Saves user and session to localStorage.

Effects: Persists session data across page refreshes

---

clearLocalStorage(): void

Removes user and session from localStorage

---

testaPassword(password: string): boolean

Checks password strength according to 5 criteria:

- At least 8 characters
- At least one uppercase letter
- At least one lowercase letter

• At least one number
• At least one special character
Output: true if valid, otherwise false

External dependencies

• useRouter() for redirects
• $fetch for HTTP requests
• localStorage for client-side persistence
• Utente model

API

Headers:

- Content-Type: application/json
- Access-Control-Allow-Origin: *
- Access-Control-Allow-Methods: POST, OPTIONS

API/abilitaPostazione.php

This API updates the status of a workstation to "enabled" (status = 0) and resets the flags of bookings marked as disabled due to disabled workstations (flag = 2).

Request body (JSON)

- Id_postazione

API Response Format

Indicates the result of activating a workstation and resetting the flags of associated bookings.

- Response (JSON):

```
{
  "stato": "OK"
}
```

- If the database connection fails:

```
{ "errore": "errore di connessione" }
```

API/abilitaPrenotazione.php

This API allows activating a specific booking by setting the flag field to 0.

Request body (JSON)

- Id_prenotazione

API Response Format

Returns all booking dates for a specific workstation identified by id_postazione.

Response (JSON):

- If bookings exist:

```
[
  { "data": "YYYY-MM-DD" },
  { "data": "YYYY-MM-DD" },  ...
]
```

- If no bookings exist: []
- If the database connection fails:

```
{ "errore": "errore di connessione" }
```

API/checkPostazioniOccupate.php

Returns all workstations occupied on a specific date. Request body (JSON)

Request body (JSON)

- Data

API Response Format

Returns all workstation IDs that are booked for a specific date.

Response (JSON):

- If there are occupied workstations:

```
{ "stato": "occupate", "occupate": [1, 2, 5, 7] }
```

- If there are no occupied workstations:

```
{ "stato": "libera" }
```

API/disabilitaPostazione.php

Updates the status of a workstation to "disabled" (status = 1) and sets the flags of the affected bookings to 2.

Request body (JSON)

Id_postazione (the ID of the workstation to disable)

API Response Format

Updates the status of a workstation to "disabled" (stato = 1) and sets the flags of associated bookings to 2 (for bookings that were previously active).

Response (JSON):

- If the operation is successful:
  { "stato": "OK" }

API/disabilitaPrenotazione.php

This API allows deactivating a specific booking by setting the flag field to 1.

Request body (JSON)

Id_prenotazione (the ID of the booking to disable)

API Response Format

returns “OK” if the operation is succesfull

Response (JSON):

- If the operation is successful:
  { "stato": "OK" }

API/eliminaPrenotazione.php

This API allows deleting a specific booking.

Request body (JSON)

- Id_prenotazione

API Response Format

Deletes a specific booking from the system.

Response (JSON):

- If the operation is successful:
  { "stato": "OK" }

API/getCategorie.php

Returns a list of all categories in the system.

API Response Format

Returns a list of all categories in the system.

Response (JSON):

- If categories exist:
  [
  { "id_categoria": 1, "nome": "Category A", "descrizione": "Description A", "livello": 1 },
  { "id_categoria": 2, "nome": "Category B", "descrizione": "Description B", "livello": 2 },
  ...
  ]
- If no categories exist: [ ]

API/getDateOccupate.php

Returns all booking dates for a specific workstation, identified by its workstation ID.

Request body (JSON)

- Id_postazione

API Response Format

Returns all booking dates for a specific workstation identified by id_postazione.

Response (JSON):

- If bookings exist:

```
[
{ "data": "YYYY-MM-DD" },
{ "data": "YYYY-MM-DD" },
...
]
```

- If no bookings exist: [ ]

API/getDipendentiCoordinatore.php

Retrieves the list of users associated with a specific coordinator (coordinator ID). If the coordinator has level 3 (administrator), it returns all users.

Request body (JSON)

- Id_utente

API Response Format

Returns the list of users associated with a specific coordinator (id_coordinatore). If the coordinator has level 3 (administrator), all users are returned.

Response (JSON):

- If users exist:

```
[ {
"id_utente": 1,
"nome": "Mario",
"cognome": "Rossi",
"genere": "M",
```

"username": "mrossi",
"livello": 2,
"id_coordinatore": 10
}]

- If no users exist: [ ]

API/getPostazioni.php

Returns a list of all workstations in the system.

API Response Format

Returns the list of all workstations available in the system.

Response (JSON):

- If workstations exist:

[{
"id_postazione": 1,
"nome": "Desk A1",
"id_categoria": 2,
"descrizione": "Near the window",
"stato": 0
}]

- If no workstations exist:

{ "errore": "Nessuna postazione trovata" }

- If the database connection fails:

{ "errore": "nessunrisultato" }

API/getPostazioniAttive.php

Returns all workstations that are already booked for a specific date. Only considers not disabled bookings (stato != 1.)

Request body (JSON)

- data : Date to check for active bookings (format YYYY-MM-DD)

API Response Format

Returns the list of workstation IDs booked on a specific date.

Response (JSON):

- If bookings exist

```
{
"postazioni": "Desk A1;Desk B3;Desk C2;",
"errore": "NONE"
}
```

- If no bookings exist:

```
{
"msg": "NONE",
"errore": "errore"
}
```

- If the database connection fails or the query fails:

```
{ "errore": "errore" }
```

API/getPostazioniDisabilitate.php

Returns a list of all disabled workstations (flag = 1) in the system.

API Response Format

Returns all workstations with status = 1 (disabled workstations).

Response (JSON):

- If disabled workstations exist:

```
[ { "id_postazione": "1" },
{ "id_postazione": "2" },
... ]
```

- If no disabled workstations exist:

```
{ "errore": "Nessuna postazione trovata" }
```

- If the database connection fails:

```
{ "errore": "nessunrisultato" }
```

API/getPrenotazioni.php

Returns all bookings made by a specific user (user ID), sorted from most recent to oldest.

Request body (JSON)

- Id_utente

API Response Format

Returns all bookings made by a specific user, ordered from most recent to oldest.

Response (JSON):

- If bookings exist:
  [{
  "id_prenotazione": "1",
  "id_utente": "5",
  "id_postazione": "12",
  "data": "YYYY-MM-DD",
  "n_modifiche": "0",
  "flag": "0"
  }]
- If no bookings exist: [ ]
- If the database connection fails:
  { "errore": "nessunrisultato" }

API/getPrenotazioniAdmin.php

Returns a list of all bookings in the system (admin-only).

API Response Format

Returns all bookings in the system, ordered from most recent to oldest.
Access restricted to users with livello = 3 (admin).

Response (JSON):

- If authorized and bookings exist:

[{ "id_prenotazione": "1",

"id_utente": "5",

"id_postazione": "12",

"data": "YYYY-MM-DD",

"n_modifiche": "0",

"flag": "0" },

{.. }]

- If authorized but no bookings exist: [ ]

API/getPrenotazioniParcheggioFromIdBadge.php

Request body (JSON)

- Data

API Response Format

Retrieves all workstation names booked by a user associated with a specific badge on a given date, filtered by category "C" (parking) and considering only active bookings (flag == 0).

Request (JSON):

- id_badge (number) – Badge ID assigned to the user.
- data (string, format YYYY-MM-DD) – Date for which bookings should be checked.

Response (JSON):

- If one or more bookings exist:

{ "msg": "OK",

"errore": "NONE" }

- If no bookings exist or criteria are not met:

{ "msg": "errore",

"errore": "errore" }

- If the database connection fails or the first query fails:

{ "errore": "errore" }

API/getPrenotazioniPostazioniFromIdBadge.php

Request body (JSON)

- Data

API Response Format

Retrieves all workstation names booked by a user associated with a specific badge on a given date, excluded category "C" (parking) and considering only active bookings (flag == 0).

Request (JSON):

- id_badge (number) – Badge ID assigned to the user.
- data (string, format YYYY-MM-DD) – Date for which bookings should be checked.

Response (JSON):

- If one or more bookings exist:

{ "postazioni": "NONE",

"username": "user123",
"errore": "NONE" }

- If no bookings exist or criteria are not met:

{ "msg": "errore",

"errore": "errore" }

- If the database connection fails or the first query fails:

{ "errore": "errore" }

API/getUtenteById.php

Retrieves the personal information of a user identified by their user ID.

Request body (JSON)

- Id_utente

API Response Format

Returns the personal information of a specific user identified by id_utente.

Response (JSON):

- If the user exists:


```
[ {
  "id_utente": "5",
  "nome": "Mario",
  "cognome": "Rossi",
  "genere": "M",
  "username": "mrossi",
  "livello": "2"
} ]
```


- If the user does not exist: [ ]
- if the database connection fails:
  { "errore": "nessunrisultato" }

API/insertPrenotazioni.php

Creates a new booking for a specific workstation by a user on a specified date.

Request body (JSON)

- Id_utente
- Id_postazione
- data

API Response Format

Inserts a new booking for a specific workstation (id_postazione) by a user (id_utente) on a given date (data).

Response (JSON):

- If the booking is successfully inserted:
  { "stato": "OK" }
- If the database connection fails or the query fails:
  { "errore": "nessunrisultato" }

API/insertUtente.php

Inserts a new booking for a specific workstation by a user on a given date.

Request body (JSON)

- nome
- cognome
- username
- genere
- password
- livello
- id_coordinatore

API Response Format

Inserts a new user into the system with personal and access details.

Response (JSON):

- If the user is successfully inserted:
  { "stato": "OK" }
- If the database connection fails or the query fails:
  { "errore": "nessunrisultato" }

API/login.php

This API allows a user to authenticate in the system. The password is verified, and if correct, a session is created with a unique identifier and a control code.

Request body (JSON)

- Id_utente
- password
- id_sessione

API Response Format

Authenticates a user and starts a session with a unique control code and expiration time.

Response (JSON):

- If login is successful:
  { "id_sessione": "SESSION_ID",
  "scadenza": 202506201050,
  "id_utente": 123,
  "nome": "FirstName",
  "cognome": "LastName",
  "genere": "M/F",
  "username": "user123",
  "livello": 2,
  "id_coordinatore": 456,
  "controlCode": 54321,
  "errore": "" }
- If the password is incorrect:
  { "errore": "passworderrata" }
- If the user does not exist or the database connection fails:
  { "errore": "nessunrisultato" }

API/logout.php

This API allows a user to log out of the system. The session is validated using the controlCode, and if the control code is correct, the session is destroyed.

Request body (JSON)

- controlCode
- id_sessione

API Response Format

Logs out a user by destroying the session if the control code is valid.

Request fields (JSON):

- id_sessione (string) – session ID
- controlCode (number) – control code assigned at login

Response (JSON):

- If logout is successful:
  { "logout": "OK" }
- If the control code is invalid:

  { "logout": "errore" }

API/sessionCheck.php

This API checks whether the user's session is still valid. If the session has expired or the controlCode is incorrect, the session is destroyed. If the session is valid, a confirmation message is returned.

Request body (JSON)

- controlCode
- id_sessione

API Response Format
Checks if a user session is still valid.

Response (JSON):

- If the session is valid:

  { "alive": "OK" }

- If the control code is incorrect, If the session variables are missing, If the session has expired:

  { "alive": "errore" }

API/updatePasswrd.php

This API allows a user to update their password in the system. The client sends the user ID and the new password, and the API updates the password field in the database.

Request body (JSON)

- id_utente

- password

API Response Format

Updates the password for a specific user.

Response (JSON):

- If the password is successfully updated:
  { "stato": "OK" }
- If the database connection fails:
  { "errore": "nessunrisultato" }

API/updatePrenotazione.php

This API allows updating the details of an existing booking, including the workstation ID, the date, and the number of modifications made. The request requires the booking ID, the new workstation, the new date, and the number of modifications, which is automatically incremented.

Request body (JSON)

- id_prenotazione
- id_postazione
- data
- n_modifiche

API Response Format

Updates an existing booking with a new workstation, date, and increments the modification counter.

Response (JSON):

- If the booking is successfully updated:
  { "stato": "OK" }
- If the database connection fails:
  { "errore": "nessunrisultato" }
- If the input data is invalid:
  { "errore": "Dati non validi" }

API/updateUtente.php

This API allows updating the details of an existing user in the database. The data to be updated includes the first name, last name, gender, username, password, user level, and coordinator ID.

Request body (JSON)

- nome
- cognome
- username
- genere
- password
- livello
- id_coordinatore

API Response Format

Updates an existing booking with a new workstation, date, and increments the modification counter.

Response (JSON):

- If the booking is successfully updated:
  { "stato": "OK" }
- If the database connection fails:
  { "errore": "nessunrisultato" }
- If the input data is invalid:
  { "errore": "Dati non validi" }

SMART Office

Introduction

Smart Office is the development and implementation of an automated office model using an Arduino Nano, integrating NFC badge readers, servo motors, LED matrices, and an LCD display to manage access control, display messages, and visualize workstation and parking availability.

Python Mediator

This Python script acts as a **mediator** between an **Arduino device** and a **booking database**.
Its main purposes are:

- Receive commands from Arduino via **serial communication**.
- Query the database for **workstation** or **parking** reservations.
- Send formatted messages back to Arduino (for display output or LED control).

This allows Arduino to focus on user interaction (badge scanning, display, LEDs) while Python manages the communication with the database.

Serial commands

Arduino:

POS:<id_badge> //chiama API controllo postazioni

PAR:<id_badge> //chiama API controllo parcheggio

Python mediator:

D:<text/text> //write on display LCD (‘/’ split into lines)

SI // move hall servo

SP //move parking servo

M<x0y0x1y1x2y2.> //turn on specific leds on the matrix

Libraries

- **requests** → handles HTTP requests to the booking database (REST API).
- **time** → used for delays and timing operations.
- **serial** (pyserial) → enables serial communication with Arduino.
- **datetime** → retrieves and formats the current date.

Functions

cmdDetector(msg: str)-> None

Parses a command received from Arduino via serial and executes the corresponding function.

- Input:
  - msg → string formatted read from the Serial
- Output:
  - No return value.
  - Executes the function depending on the command:
    - POS:<id_badge> → calls postazioni(id_badge)
    - PAR:<id_badge> → calls parcheggio(id_badge)

cmdDetector(msg: str)-> None

Parses a command received from Arduino via serial and executes the corresponding function.

- **Input:**
  - msg → string formatted as "CMD:ID_BADGE".
- **Output:**
  - No return value.
  - Executes the function depending on the command:

- POS:<id_badge> → calls postazioni(id_badge)
- PAR:<id_badge> → calls parcheggio(id_badge)

---

postazioni(id_badge: str)-> None

Retrieves and sends to Arduino the **workstation reservations** associated with a user badge.

- **Input:**
  - id_badge → user badge ID (string).
- **Output:**
  - No return value.
  - Sends to Arduino:
    - An error message if the badge is invalid.
    - A welcome message and the list of booked workstations, or a notice if no reservation exists.
- **API called:** getPrenotazioniParcheggioFromIdBadge.php

---

postazioni(id_badge: str)-> None

Retrieves and sends to Arduino the **workstation reservations** associated with a user badge.

- **Input:**
  - id_badge → user badge ID (string).
- **Output:**
  - No return value.
  - Sends to Arduino:
    - An error message if the badge is invalid.
    - A welcome message and the list of booked workstations, or a notice if no reservation exists.

- API called:
    - getPrenotazioniPostazioniFromIdBadge.php

---

postazioniAttive()-> None

Retrieves the list of active workstation bookings for the current day and sends the corresponding LED codes to Arduino.

- Input:
    - None.
- Output:
    - No return value.
    - Sends to Arduino a message "M..." containing the LED coordinates of all active workstations.
- API called:
    - getPostazioniAttive.php

Arduino nano

This Arduino Nano sketch acts as a hardware mediator between NFC badge readers, servos, LED matrices, and an LCD display.
Its main goal is to interact with a Python backend to validate user badges, display messages, and control access to workstations and parking spaces.

The Arduino handles:

- Reading user badges via two NFC readers through an I2C multiplexer (TCA9548A).
- Updating LED matrices to visualize active workstations.
- Displaying messages on a 20x4 LCD.

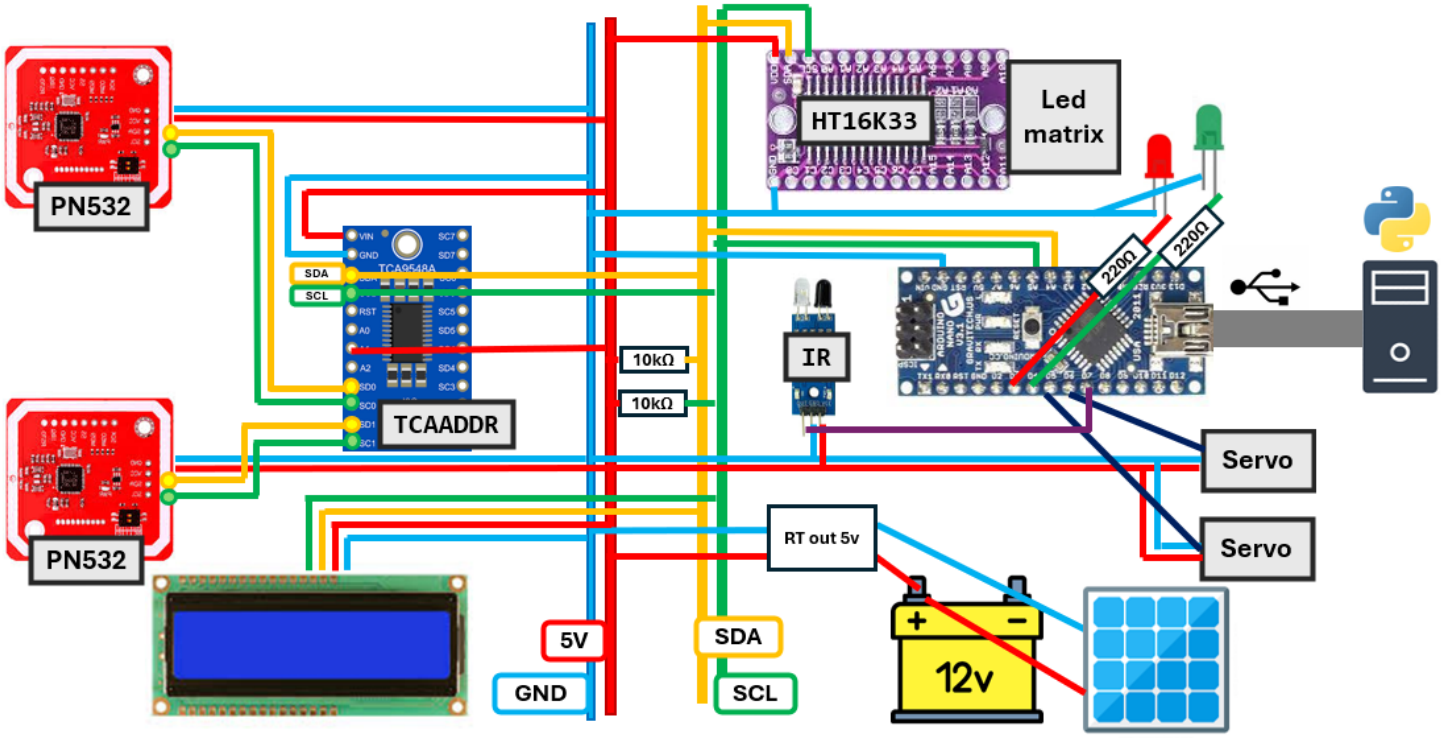
- Controlling **servo motors** for entrance and parking access.
- Detecting obstacles using an **IR sensor**.
- Communicating with the Python mediator using **serial commands**.

This system allows a seamless integration of hardware for smart office management, separating **physical access control** (Arduino) from **data validation and logic** (Python).

Hardware

Wiring Diagram

- NFC readers (PN532) → connected via I2C through TCA9548A multiplexer (channels 1 and 2).
- HT16K33 LED matrix → I2C address 0x70.
- LCD display → I2C address 0x27.
- Servo motors → entrance (servoI) pin 5, parking (servoP) pin 6.
- IR sensor → obstacle detection, pin 7.
- LED indicators → green pin 4, red pin 3.
- TCA9548A I2C multiplexer → address 0x72, manages multiple I2C devices (NFC readers).



Peripherals

HT16K33 LED Matrix

- Displays active workstations.
- Receives coordinate data from Python via serial.

IR Sensor

- Detects obstacles in front of the parking servo.
- Pin 7, returns LOW when an obstacle is present.

LCD Display

- 20x4 I2C display.
- Shows welcome messages and reservation information received from Python.

PN532 NFC

- Reads user badges.

- Two readers connected via TCA9548A multiplexer.
- Sends UID via serial to Python as POS:<uid> or PAR:<uid>.

Servo

- Entrance servo (servoI) → controls the entry door.
- Parking servo (servoP) → controls parking gate.
- Asynchronous timer control for automatic closing.

TCA9548A

- I2C multiplexer to switch between NFC readers.
- Supports up to 8 channels.

Script

Libreries

1. Adafruit GFX Library
   Provides core graphics functions for displays.
   https://github.com/adafruit/Adafruit-GFX-Library
2. Adafruit LED Backpack Library
   For controlling LED backpacks like 8x8 matrices and 7-segment displays.
   https://github.com/adafruit/Adafruit_LED_Backpack
3. Adafruit PN532 Library
   Library for the PN532 NFC/RFID breakout boards.
   https://github.com/adafruit/Adafruit-PN532
4. Adafruit PWM Servo Driver Library
   Controls the 16-channel PWM/Servo driver.
   https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library
5. Adafruit SoftServo Library
   A lightweight software servo driver.
   https://github.com/adafruit/Adafruit_SoftServo

6. LiquidCrystal I2C Library
   For controlling I2C LCD displays.
   https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library
7. Wire Library
   Allows communication with I2C devices.
   https://www.arduino.cc/en/Reference/Wire

Setup

- Initializes I2C (Wire) and serial communication.
- Configures pins for servos, LEDs, and IR sensor.
- Initializes LCD display and LED matrix.
- Initializes NFC readers on channels 1 and 2.
- Prints debug messages to the serial monitor.

Loop

- Infinite loop that:
  1. Selects NFC channel 1 and reads workstation badge → sends POS:<uid> to Python.
  2. Reads commands from Python (serialReader()).
  3. Selects NFC channel 2 and reads parking badge → sends PAR:<uid> to Python.
  4. Reads commands from Python again.
  5. Updates LCD and checks servo positions using checkLCDdisplay() and checkServo().
  6. Includes small delays to stabilize execution.

Functions

uint64ToString(uint64_t v)

Converts a 64-bit UID to a string.

- **Input:** 64-bit integer UID.
- **Output:** String representation of UID.

---

nfcReader(int n)

Reads NFC badge from a selected reader and sends serial message.

- **Input:**
  - n = 1 → workstation reader.
  - n = 2 → parking reader.
- **Output:** Serial output: POS:<uid> or PAR:<uid>.

---

channelSelect(uint8_t channel)

Selects an I2C channel on TCA9548A to activate the desired NFC reader.

- **Input:** Channel number 1–7.
- **Output:** Configures multiplexer.

---

serialReader()

Reads serial commands from Python and executes the corresponding actions.

- Commands handled:
    - "SI" → move entrance servo.
    - "SP" → move parking servo and update LEDs.
    - "M:..." → update LED matrix.
    - "D:..." → update LCD display.

---

rilevaOstacolo()

Checks for obstacles in front of the parking servo.

- Output: Returns true if an obstacle is detected, otherwise false.

---

checkServo()

Asynchronous control of servos.

- Functionality:
    - Returns servo positions to initial state after timer interval.
    - Checks for obstacles before closing parking servo.

---

checkLCDdisplay()

Manages the LCD display timing.

- Functionality: Clears the display after a defined interval if a message is active.

aggiornaMatrice(String coordinate)

Updates the LED matrix according to a string of coordinate pairs received from Python.

- **Input:** Coordinate string.
- **Output:** Updates LED matrix display.

LCDdisplay(String msg)

Displays a formatted message on the LCD, splitting lines at /.

- **Input:** Message string.
- **Output:** Updates the LCD display.

---

resetLCDdisplay()

Resets the LCD display to the default welcome screen.

- **Output:** "Benvenuto in SMART Office / avvicina il badge".
