A close-up of a logo

AI-generated content may be incorrect.

**Proiect PIISW**

**RetroPie IoT**

PETREANU MARIUS

SECI-RCD, anul I

BRAȘOV, 2025

## Introducere

Domeniul IoT (Internet of Things) a cunoscut o dezvoltare accelerată, fiind integrat în diverse arii precum automatizări industriale, locuințe inteligente, agricultură sau chiar divertisment. Acest proiect își propune să exploreze aplicabilitatea conceptului IoT într-un mediu atipic: o consolă retro de jocuri video. RetroPie este o platformă cunoscută pentru emularea jocurilor clasice pe Raspberry Pi, iar integrarea sa cu servicii Cloud oferă posibilitatea extinderii funcționalității către analiză, monitorizare și vizualizare a activității utilizatorului.

Prin dezvoltarea acestui proiect, se creează o punte între lumea retro-gaming-ului și tehnologiile moderne de tip IoT, oferind utilizatorului nu doar experiența nostalgică a jocurilor, ci și o perspectivă asupra modului în care timpul petrecut este folosit, cu date salvate în timp real în Cloud.

### 1.1 Scopul proiectului

Scopul principal al proiectului este de a crea o soluție complet bazată pe Raspberry Pi și RetroPie, care sa permită monitorizarea automata a sesiunilor de joc, salvarea acestora in Google Firebase sub forma unor log-uri si afișarea unor statistici relevante într-un dashboard web interactiv.

### 1.2 Obiective urmărite

* Implementarea unui sistem de urmărire automată a sesiunilor de joc pe RetroPie;
* Colectarea datelor despre fiecare sesiune: numele jocului, durata, platforma emulată;
* Stocarea acestor date în Google Firebase Firestore;
* Afișarea statisticilor într-un dashboard web (top jocuri, activitate zilnică, top platforme);
* Posibilitatea extinderii sistemului cu comenzi în timp real sau notificări;
* Comunicație bidirecționala: comenzi trimise din Firebase catre Raspberry Pi.

### 1.3 Tehnologii utilizate

Pentru realizarea obiectivelor proiectului, s-au folosit următoarele tehnoogii si instrumente hardware si software:

Hardware:

* Raspberry Pi 5
* Card microSD cu viteze mari de scriere si citire
* Controller USB
* Monitor

Software:

* RetroPie (platforma bazata pe Raspberry Pi OS, specializata in emularea jocurilor retro);
* Emulatori specifici integrati in RetroPie:
  + PSX: lr-pcsx-rearmed
  + PSP: ppsspp
  + PS2: aethersx2
* Python pentru dezvoltarea scripturilor de monitorizare automata;
* Firebase Admin SDK (pentru găzduirea interfeței web interactive);
* HTML, CSS, JavaScript pentru dezvoltarea dashboard-ului web;
* Chart.js (pentru afișarea grafica interactiva a statisticilor).

## Arhitectura sistemului

### Arhitectura hardware

### Descrierea componentei principale

Componenta hardware centrala este plăcuța Raspberry Pi 5 de 8GB. Acesta este un calculator de mici dimensiuni ( Single Board Computer – SBC ) compact si puternic, recunoscut pentru versatilitatea si eficienta sa energetica. Configuratia de 8GB de memorie RAM o face deosebit de potrivita pentru rularea unor sisteme de operare precum Raspberry Pi OS si pentru a gestiona sarcini multiple, inclusiv emularea jocurilor si executia scripturilor Python pentru monitorizarea datelor.

A close-up of a computer chip

AI-generated content may be incorrect.

Figura 1- Raspberry Pi 5

### Diagrama de arhitectura hardware

### 

Figure 2 - Arhitectura hardware

Diagrama din figura 2 ilustrează schematic inter conectivitatea componentelor hardware esențiale care alcătuiesc sistemul RetroPie. În centrul arhitecturii se află Raspberry Pi 5, care funcționează ca unitate centrală de procesare și control. Acesta este conectat la un Monitor via HDMI pentru afișarea interfeței grafice a RetroPie și a jocurilor, și la un Controller USB pentru interacțiunea utilizatorului cu jocurile. Un Card microSD este inserat în Raspberry Pi 5, servind ca mediu de stocare principal pentru sistemul de operare, jocurile emulatoare și datele proiectului. Pentru a îndeplini funcționalitățile IoT, Raspberry Pi 5 se conectează la Internet (prin Wi-Fi sau Ethernet), permițând comunicarea cu serviciile cloud, în special cu Google Firebase, unde sunt stocate datele colectate despre sesiunile de joc și de unde pot fi primite comenzi.

### Arhitectura software

Diagrama de mai jos detaliază fluxul logic al datelor și interacțiunea componentelor software. Punctul de plecare este Raspberry Pi cu RetroPie, platforma pe care utilizatorul interacționează cu jocurile. Un element cheie al acestei arhitecturi este scriptul game\_session\_tracker.py, dezvoltat în Python.

Acest script rulează în permanență în fundal pe Raspberry Pi, monitorizând activitatea de joc a utilizatorului. Atunci când utilizatorul intră într-un joc, scriptul înregistrează începutul sesiunii, iar la ieșirea din joc, calculează durata exactă a sesiunii (ore și minute) și identifică jocul jucat.

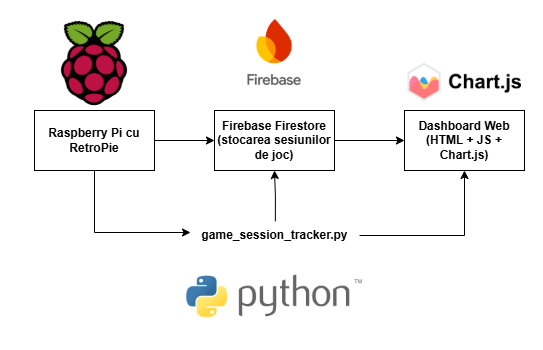


Figure 3 - Diagrama arhitecturii software

Informațiile colectate (numele jocului, durata sesiunii, platforma emulată ) sunt apoi trimise către Firebase Firestore, care servește drept bază de date pentru stocarea acestor log-uri de joc. Din Firebase Firestore, datele devin accesibile pentru Dashboard-ul Web. Acest dashboard, construit folosind HTML, CSS și JavaScript, preia datele din Firebase și le procesează pentru a afișa statistici relevante, cum ar fi topul jocurilor jucate, activitatea zilnică sau topul platformelor. Pentru o vizualizare grafică interactivă a acestor statistici, dashboard-ul utilizează biblioteca Chart.js. Astfel, se realizează o punte între experiența de retro-gaming și tehnologiile moderne IoT, oferind utilizatorului o perspectivă asupra modului în care timpul petrecut este utilizat, cu date salvate în timp real în Cloud.

## Design si implementare

Acest capitol detaliază aspectele tehnice ale proiectului, de la structura bazei de date până la logica scripturilor de monitorizare și dezvoltarea interfeței web. Fiecare componentă este proiectată și implementată pentru a asigura o integrare fluidă și o funcționalitate robustă.

### Design-ul bazei de date ( Google Firebase Firestore )

Pentru stocarea datelor sesiunilor de joc, a fost ales serviciul Google Firebase Firestore. Google Firebase este o platformă de dezvoltare de aplicații mobile și web, susținută de Google, care oferă o suită cuprinzătoare de servicii backend. Aceasta simplifică procesul de dezvoltare, permițând dezvoltatorilor să se concentreze pe crearea experienței utilizatorului, în timp ce Firebase gestionează infrastructura backend. Printre serviciile sale cheie se numără baze de date NoSQL în timp real (Firestore și Realtime Database), autentificare, găzduire (Hosting), stocare de fișiere (Cloud Storage) și funcții serverless (Cloud Functions).

Pentru proiectul "RetroPie IoT", Firebase Firestore este utilizată ca bază de date principală, oferind o soluție scalabilă și flexibilă pentru stocarea datelor sesiunilor de joc în timp real.

Datele sunt organizate într-o colecție principală, denumită game\_sessions. Fiecare sesiune de joc este reprezentată de un document individual în această colecție. Fiecare document conține următoarele câmpuri, observate direct în consola Firebase:

* *duration*: Reprezintă durata sesiunii de joc, exprimată în minute.
* *end\_time*: Timestamp-ul exact (data și ora) la care s-a încheiat sesiunea de joc. Acest câmp este esențial pentru ordonarea cronologică și pentru filtrarea datelor.
* *game*: Numele complet al jocului care a fost jucat (ex: "FIFA 14 (Europe)"). Această informație permite identificarea unică a titlului jocului.
* *platform*: Platforma de emulare pe care a rulat jocul (ex: "psp"). Acest câmp ajută la clasificarea și analiza preferințelor utilizatorului pe diverse platforme.
* *session\_name*: Un identificator sau nume descriptiv pentru sesiune, care include și numele jocului și data (ex: "FIFA 14 (Europe) 2025-05-09 23:16:35"). Acesta este util pentru o referință rapidă și lizibilă.
* *start\_time*: Timestamp-ul exact (data și ora) la care a început sesiunea de joc.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 4 - Consola Firebase

In Figura 4 este ilustrată consola Firebase. Aceasta vizualizare confirmă modul în care datele sunt organizate și stocate eficient pentru fiecare sesiune de joc. Datele sunt colectate și organizate in ordine descrescătoare în funcție de dată si de oră.

### Implementarea scripturilor Python

Python este un limbaj de programare interpretat, de nivel înalt, cu o sintaxă clară și ușor de citit, ceea ce îl face extrem de popular atât pentru începători, cât și pentru dezvoltatori experimentați. Cunoscut pentru versatilitatea sa, Python este utilizat într-o gamă largă de domenii, inclusiv dezvoltare web, analiză de date, inteligență artificială, automatizare și, așa cum este cazul în acest proiect, pentru aplicații de monitorizare și IoT. Unul dintre avantajele sale majore este bogăția de biblioteci și framework-uri, care facilitează dezvoltarea rapidă și eficientă a soluțiilor software. În cadrul proiectului "RetroPie IoT", Python este limbajul de bază ales pentru dezvoltarea proiectului, mai exact pentru dezvoltarea scripturilor responsabile de monitorizarea automată a sesiunilor de joc pe Raspberry Pi, colectarea datelor relevante și integrarea acestora cu serviciile Google Firebase.

Scriptul principal, *game\_session\_tracker.py*, este responsabil pentru monitorizarea, colectarea și transmiterea datelor sesiunilor de joc. Mai jos sunt detaliate cele mai importante blocuri de cod, esențiale pentru funcționalitatea sistemului.

def detect\_platform\_from\_path(rom\_path):

    rom\_path\_lower = rom\_path.lower()

    if "/roms/psx/" in rom\_path\_lower:

        return "psx"

    elif "/roms/ps2/" in rom\_path\_lower:

        return "ps2"

    elif "/roms/psp/" in rom\_path\_lower:

        return "psp"

    elif "/roms/nes/" in rom\_path\_lower:

        return "nes"

    elif "/roms/snes/" in rom\_path\_lower:

        return "snes"

    elif "/roms/gba/" in rom\_path\_lower:

        return "gba"

    elif "/roms/gb/" in rom\_path\_lower:

        return "gb"

    elif "/roms/n64/" in rom\_path\_lower:

        return "n64"

    else:

        return "unknown"

Funcția detect\_platform\_from\_path este reponsabilă pentru identificarea automată a platformei de joc pe baza căii fisierului ROM. Funcția primește ca argument rom\_path (calea completă a fișierului jocului). Prin verificarea prezenței sub-stringurilor specifice (/roms/psx/, /roms/ps2/, etc.) în calea fișierului (convertită la minuscule pentru a asigura insensibilitatea la majuscule/minuscule), scriptul determină platforma corectă. Această abordare elimină necesitatea unei configurări manuale a platformei pentru fiecare joc sau de a interoga baza de date RetroPie, simplificând procesul și făcându-l ușor de extins pentru noi platforme prin adăugarea unor verificări suplimentare. În cazul in care nu exista un argument valid, funcția returneaza “unknown”.

def detect\_new\_session():

    for proc in psutil.process\_iter(['pid', 'name', 'cmdline']):

        try:

            name = proc.info['name']

            cmd = proc.info['cmdline']

            pid = proc.info['pid']

            if not name or not cmd or pid in tracked\_pids:

                continue

            rom\_path = None

            rom\_extensions = ('.iso', '.chd', '.cue', '.bin', '.img', '.zip', 'cso')

            for arg in cmd:

                if arg.lower().endswith(rom\_extensions):

                    rom\_path = arg

                    break

            if not rom\_path:

                continue

          game\_raw = os.path.splitext(os.path.basename(rom\_path))[0]

            game = urllib.parse.unquote(game\_raw).replace("\\", "").strip()

            platform = detect\_platform\_from\_path(rom\_path)

            tracked\_pids.add(pid)

            start\_time = datetime.now()

            print(f"[START] {game} ({platform})")

            while psutil.pid\_exists(pid):

                time.sleep(2)

            end\_time = datetime.now()

            duration = end\_time - start\_time

            duration\_str = str(duration)

            session\_data = {

                "session\_name": f"{game} - {start\_time.strftime('%Y-%m-%d %H:%M:%S')}",

                "game": game,

                "platform": platform,

                "start\_time": start\_time.isoformat(),

                "end\_time": end\_time.isoformat(),

                "duration": duration\_str

            }

          db.collection("games\_sessions").add(session\_data)

            print(f"[STOP] Sesiune salvată în Firebase: {session\_data}")

        except (psutil.NoSuchProcess, psutil.psutil.AccessDenied):

            continue

Funcția *detect\_new\_session* este componenta cheie a scriptului de monitorizare, orchestrând întregul proces de la detectarea unui joc până la înregistrarea sesiunii. Această funcție scanează continuu procesele active de pe Raspberry Pi pentru a identifica lansarea unui emulator de joc. Odată ce un joc este detectat (pe baza căii fișierului ROM), funcția înregistrează momentul de început al sesiunii și continuă să monitorizeze procesul emulatorului. Când jocul este închis, funcția calculează durata exactă a sesiunii, extrage numele jocului și platforma emulată. Toate aceste informații sunt apoi structurate și trimise în timp real către Google Firebase Firestore pentru stocare, asigurând că fiecare sesiune de joc este logată cu precizie și disponibilitate imediată pentru analiză.

Pe lângă scriptul de monitorizare continuă, un alt script, send\_to\_firebase.py, demonstrează explicit mecanismul de bază al trimiterii datelor către Google Firebase Firestore. Acest script servește ca un exemplu clar și concis al modului în care o sesiune de joc colectată este structurată și apoi persista în baza de date Cloud.

Acest script Python este conceput pentru a ilustra fundamental procesul de inițializare a Firebase și de salvare a unui document de date într-o colecție Firestore. Structura sa simplificată permite înțelegerea clară a etapelor esențiale:

* Scriptul începe prin importarea modulelor necesare: firebase\_admin și componentele sale specifice (credentials, firestore) pentru interacțiunea cu Firebase, și datetime pentru gestionarea informațiilor temporale;
* Linia *cred = credentials.Certificate('/home/maurice/retropie-iot-firebase-adminsdk-fbsvc-b8ac8afda4.json')* reprezintă un pas crucial de securitate. Aceasta încarcă credențialele de autentificare ale proiectului Firebase dintr-un fișier JSON (.json) denumit cheie de serviciu. Acest fișier conține informațiile necesare pentru ca scriptul să se autentifice în siguranță la proiectul tău Firebase, fără a expune direct chei sau parole în cod.
* Un dicționar Python numit game\_session este creat pentru a reprezenta o singură sesiune de joc. Această structură demonstrează modelul de date care va fi stocat în Firestore. Câmpurile incluse sunt:
* 'game': Numele jocului (ex: "Super Mario Bros").
* 'platform': Platforma pe care a fost jucat jocul (ex: "NES").
* 'duration\_minutes': Durata sesiunii în minute (ex: 45).
* 'timestamp': Momentul exact la care a fost înregistrată sesiunea, generat folosind datetime.now().isoformat() pentru un format standardizat (ISO 8601), ușor de interpretat de baze de date și alte aplicații.
* Comanda *doc\_ref = db.collection('games\_sessions').add(game\_session)* este miezul funcționalității scriptului. Aceasta instruiește Firebase Firestore să adauge un nou document la colecția specificată (games\_sessions).
* Metoda .add() este utilizată pentru a permite Firestore să genereze automat un identificator unic (ID) pentru noul document, asigurând că fiecare intrare este distinctă și ușor de referențiat.
* Variabila doc\_ref capturează un tuplu care conține un referință la operațiunea de actualizare și referința la documentul nou creat.
* În final, scriptul afișează un mesaj de confirmare prin print(), indicând succesul operației și ID-ul documentului nou generat în Firestore. Acest mesaj este util pentru depanare și pentru a verifica vizual că datele au fost transmise.

Un aspect cheie al proiectului îl constituie capacitatea de comunicație bidirecțională, permițând trimiterea de comenzi de la Firebase către Raspberry Pi. Scriptul *firebase\_listener.py* (așa cum este sugerat de funcționalitate) este dedicat acestei funcționalități, transformând Raspberry Pi într-un dispozitiv capabil să primească și să execute instrucțiuni de la distanță. Acest script Python acționează ca un "listener" (ascultător) în timp real pentru comenzi trimise din Firebase Firestore, oferind un canal de control la distanță pentru Raspberry Pi. Funcționalitatea sa principală este de a monitoriza o colecție specifică din Firestore și de a executa acțiuni predefinite în momentul în care noi comenzi sunt adăugate.

Similar cu celelalte scripturi, scriptul începe prin importarea modulelor necesare *(firebase\_admin, credentials, firestore, time, os).* Se realizează inițializarea securizată a conexiunii Firebase, încărcând fișierul de cheie de serviciu (.json) și obținând instanța clientului *Firestore (db = firestore.client())*. Aceasta asigură că scriptul are permisiunea de a citi și șterge documente din baza de date.

Funcția de Callback on\_snapshot ( Procesorul de comenzi):

* Este inima funcționalitații de ascultare. on\_snapshop este o funcție de callback care este declanșată automat de SDK-ul Firebase ori de câte ori există modificări în colecția monitorizată;
* Primește trei argumente*: col\_snapshot* (starea curentă a colecției*), changes* (o listă a modificărilor specifice care au avut loc) și *read\_time* (momentul la care a fost citită starea).
* scriptul iterează prin *changes* (în special *change.document.to\_dict()* pentru a obține datele noilor comenzi).
* Pe baza câmpului 'action' din documentul de comandă, scriptul realizează acțiuni specifice pe Raspberry Pi:
  + 'reboot': Afișează un mesaj de repornire (și, dacă este decomentată, execută os.system('sudo reboot') pentru a reporni fizic dispozitivul).
  + 'shutdown': Afișează un mesaj de oprire (și, dacă este decomentată, execută os.system('sudo shutdown now') pentru a opri fizic dispozitivul).
  + 'message': Afișează un mesaj textual suplimentar, extras din câmpul 'text' al comenzii.
* După ce o comandă este procesată, scriptul șterge documentul respectiv din colecția *device\_commands* *(db.collection('device\_commands').document(change.document.id).delete()*). Acest pas este esențial pentru a preveni re-execuția aceleiași comenzi la următoarea declanșare a listener-ului și pentru a menține colecția curată.

Blocul *while True: time.sleep(1)* asigură că scriptul rulează continuu în fundal pe Raspberry Pi. Deși listener-ul on\_snapshot este asincron și nu blochează execuția, această buclă menține procesul Python activ, permițând listener-ului să opereze fără întrerupere și să reacționeze la comenzi în timp real.

### Dezvoltarea Dashboard-ului Web

Dashboard-ul web este interfața vizuală care permite utilizatorului să exploreze statisticile sesiunilor de joc. Acesta este o aplicație web statică, proiectată pentru a fi găzduită prin Firebase Hosting și pentru a prelua datele direct din Firebase Firestore.

Dashboard-ul utilizează HTML pentru a defini o structură clară, centrată pe afișarea sesiunilor de joc sub formă de carduri individuale. CSS-ul încorporat asigură o estetică modernă și intuitivă, cu un fundal curat, fonturi lizibile și elemente interactive (ex: animații la *hover* pe cardurile de sesiune) care îmbunătățesc experiența utilizatorului. Fiecare card de sesiune prezintă numele jocului, platforma, durata și data, formatate pentru o lizibilitate maximă.

Logica JavaScript și integrarea Firebase:

* Scriptul JavaScript (incluzând Firebase SDK și un fișier de configurare extern) inițializează conexiunea la Firebase Firestore;
* O funcție utilitară (formatDuration) este responsabilă pentru a transforma durata brută a sesiunii într-un format prietenos pentru utilizator;
* La încărcarea paginii, scriptul interoghează colecția games\_sessions din Firestore, preluând toate sesiunile de joc (sortate descrescător după data de început);
* Ulterior, JavaScript-ul generează dinamic elemente HTML (div-uri) pentru fiecare sesiune preluată și le adaugă în containerul desemnat al paginii, populând interfața cu date actuale din baza de date;

### Dezvoltarea paginii de statistici

Pagina *charts.html* este concepută pentru a oferi o reprezentare grafică a activității de joc, oferind o perspectivă mai profundă asupra datelor colectate. Pagina include trei elemente *<canvas>*, fiecare având un ID unic *(topGamesChart, dailyChart, platformChart)*, care vor servi drept suprafețe de desen pentru graficele generate de Chart.js.

Stilizarea CSS, încorporată în <head>, asigură un aspect curat și modern al graficelor și al paginii în ansamblu. Canvasele sunt stilizate cu fundal alb, umbre și colțuri rotunjite, iar titlurile sunt clare. Secțiunea #platformIcons este stilizată pentru a afișa iconițele platformelor într-un mod atractiv.

Funcția getTopGames() (Grafic Bară - Top 5 Jocuri):

* Această funcție asincronă preia toate sesiunile de joc din colecția *games\_sessions*.
* Procesează datele pentru a calcula durata totală de joc pentru fiecare titlu. Durata (inițial în format HH:MM:SS) este convertită în minute pentru a facilita agregarea.
* Sortează jocurile după timpul total jucat și selectează primele 5.
* Creează un grafic de tip bară (type: 'bar') utilizând Chart.js, vizualizând topul 5 jocuri după timpul total petrecut. Un *tooltip* personalizat afișează numele complet al jocului la *hover*.

Funcția getDailyActivity() preia toate sesiunile de joc. Aceasta agregă timpul total jucat pentur fiecare zi bazându-se pe *start\_time* al fiecărei sesiuni. Un grafic de tip linie cu Chart.js este creat, ilustrând tendințele activității de joc ale utilizatorului de-a lungul timpului.

Funcția getTopPlatforms():

* Această funcție asincronă preia sesiunile și calculează numărul de sesiuni per platformă;
* Sortează și selectează primele 3 platforme cel mai des utilizate;
* Creează un grafic de tip doughnut (type: 'doughnut') cu Chart.js, reprezentând distribuția sesiunilor pe platformele de top;
* Funcționalitate Avansată - Inserare Logo-uri Platforme: Include un plugin Chart.js personalizat (drawImages) care desenează dinamic logo-urile platformelor direct pe segmentele graficului doughnut. Acest plugin:
  + Preîncarcă imaginile logo-urilor (ex: platforms/nes.png).
  + Calculează poziția optimă pentru fiecare logo pe grafic.
  + Ajustează dimensiunea logo-ului la hover pentru o experiență interactivă îmbunătățită.
  + Asigură că imaginile sunt afișate doar după ce s-au încărcat complet.

## Rezultate obținute



Figure 5 - Montajul final

A screenshot of a computer

AI-generated content may be incorrect.

Figure 6 - Dashboard RetroPie IoT

Dashboard-ul web interactiv permite utilizatorului să vizualizeze sesiunile de joc înregistrate. Imaginea de mai jos oferă o captură de ecran a acestui dashboard principal, accesibil printr-un browser web. Prezintă o interfață curată și intuitivă, cu un titlu clar "RetroPie IoT - Sesiuni de joc". Designul minimalist și utilizarea spațiului alb contribuie la o lizibilitate crescută.

Fiecare sesiune de joc este afișată ca un "card" distinct, fiecare card reprezentând un document individual preluat din colecția games\_sessions din Firebase Firestore. Aceste carduri sunt aranjate vertical, cele mai recente sesiuni fiind afișate în partea superioară. Cardurile oferă informații despre sesiunea joc, cum ar fi: numele jocului, platforma, durata sesiunii și data și ora începerii sesiunii.

A green line graph on a white background

AI-generated content may be incorrect.

Figure 7 -Chart cu activitatea zilnică

Graficul de mai sus prezintă un exemplu de vizualizare a activității zilnice. Acesta este un grafic de tip linie care ilustrează numărul total de minute jucate pe zi. Axa orizontală (X) reprezintă datele calendaristice (ex: "2025-04-12", "2025-05-19"), iar axa verticală (Y) indică numărul de minute jucate. Tendințele vizibile pe grafic permit utilizatorului să identifice perioadele de activitate intensă sau, dimpotrivă, de pauză în sesiunile de joc. Legenda "Minute jucate" și titlul "Activitate zilnică" contribuie la claritatea interpretării datelor.

A screen shot of a computer

AI-generated content may be incorrect.

Figure 8 - Comunicatia bidirectională

Comunicația bidirecțională este realizată prin utilizarea Firebase Firestore ca un "broker" de mesaje. Raspberry Pi-ul, prin scriptul Python listen.py, acționează ca un client care monitorizează o colecție specifică din Firestore dedicată comenzilor. O colecție dedicată în Firestore, denumită device\_commands (așa cum se vede în Figura 8), este utilizată pentru a stoca comenzile în așteptare. Când o nouă comandă este adăugată în această colecție, Firebase notifică instantaneu scriptul de listen de pe Raspberry Pi.

Fiecare comandă este un document în Firestore, având un câmp principal action (ex: "reboot", "shutdown", "message"). Comenzile pot include și parametri suplimentari, cum ar fi un câmp text pentru comanda "message". In imaginea de mai sus, avem ca exemplu comanda de reboot.

Avantajul principal al acestei abordări este controlul flexibil și scalabil al dispozitivului la distanță, fără a necesita o conexiune directă sau un server dedicat pe Raspberry Pi pentru primirea comenzilor. Securitatea este asigurată prin mecanismele de autentificare și autorizare Firebase.

## Implementări viitoare

Proiectul "RetroPie IoT" a demonstrat cu succes conceptul de monitorizare și control la distanță a unei console de jocuri retro. Cu toate acestea, există numeroase oportunități de extindere și îmbunătățire a funcționalităților, transformând această soluție într-un sistem și mai robust și mai util.

* Notificări Firebase Cloud Messaging (FCM): Integrarea FCM ar permite trimiterea de notificări push către dispozitive mobile (smartphone-uri, tablete) atunci când o sesiune de joc începe sau se termină, sau când se atinge un anumit prag de timp de joc;
* Optimizarea Detecției Sesiunilor: Explorarea unor metode alternative sau suplimentare pentru detecția sesiunilor de joc, cum ar fi monitorizarea activității porturilor GPIO (pentru controller-e fizice) sau integrarea directă cu API-uri ale RetroPie (dacă sunt disponibile), pentru o precizie și mai mare;
* Dockerizare: Împachetarea scripturilor Python și a dependențelor într-un container Docker ar simplifica implementarea și gestionarea pe Raspberry Pi, asigurând un mediu consistent;
* Stocare Locală Temporară (Offline Sync): În cazul pierderii conexiunii la internet, scriptul ar putea stoca temporar datele local și le-ar sincroniza cu Firebase odată ce conexiunea este restabilită, asigurând că nicio sesiune nu este pierdută.

## Concluzii

Proiectul "RetroPie IoT" a demonstrat cu succes convergența tehnologiilor retro-gaming cu capabilitățile moderne ale Internet of Things. Pornind de la un concept relativ simplu de monitorizare a sesiunilor de joc pe un Raspberry Pi cu RetroPie, soluția s-a extins pentru a include stocarea datelor în timp real în Google Firebase Firestore și vizualizarea interactivă a acestora printr-un dashboard web intuitiv.

Am atins toate obiectivele propuse: implementarea unui sistem automat de urmărire a sesiunilor, colectarea detaliilor esențiale (nume joc, durată, platformă), stocarea securizată a acestora în Firebase, și afișarea statisticilor relevante printr-o interfață web. Mai mult, am demonstrat și capacitatea de comunicație bidirecțională, permițând trimiterea de comenzi din Cloud către Raspberry Pi.

Această soluție nu doar adaugă o nouă dimensiune experienței nostalgice a jocurilor retro, oferind utilizatorilor o perspectivă detaliată asupra obiceiurilor lor de joc și a modului în care timpul petrecut este folosit, cu date salvate în timp real în Cloud. De asemenea, subliniază flexibilitatea și puterea platformelor open-source precum Raspberry Pi și Python în combinație cu serviciile cloud scalabile și fiabile oferite de Firebase.

„RetroPie IoT" este o dovadă concretă a modului în care concepte atipice pot fi integrate cu succes în peisajul IoT, deschizând calea pentru inovații ulterioare în domenii diverse, de la divertisment personal la monitorizarea utilizării dispozitivelor în medii educaționale sau de cercetare. Potențialul de extindere este vast, sugerând că această bază solidă poate fi dezvoltată într-un sistem mult mai complex și mai interactiv în viitor.

[1. Introducere 2](#_Toc199290112)

[1.1 Scopul proiectului 2](#_Toc199290113)

[1.2 Obiective urmărite 2](#_Toc199290114)

[1.3 Tehnologii utilizate 3](#_Toc199290115)

[2. Arhitectura sistemului 3](#_Toc199290116)

[2.1 Arhitectura hardware 3](#_Toc199290117)

[2.1.1 Descrierea componentei principale 3](#_Toc199290118)

[2.1.2 Diagrama de arhitectura hardware 4](#_Toc199290119)

[2.2 Arhitectura software 5](#_Toc199290121)

[3. Design si implementare 6](#_Toc199290122)

[3.1 Design-ul bazei de date ( Google Firebase Firestore ) 6](#_Toc199290123)

[3.2 Implementarea scripturilor Python 7](#_Toc199290124)

[3.3 Dezvoltarea Dashboard-ului Web 13](#_Toc199290125)

[3.4 Dezvoltarea paginii de statistici 13](#_Toc199290126)

[4. Rezultate obținute 15](#_Toc199290127)

[5. Implementări viitoare 17](#_Toc199290128)

[6. Concluzii 18](#_Toc199290129)